Dispense per il corso di algoritmi e complessità

Sebastiano Vigna

14 giugno 2019

1 Controesempio per l'analisi del secondo algoritmo di bilanciamento del carico

Abbiamo analizzato il secondo algoritmo per il bilanciamento del carico (quello che ordina i compiti in ordine di tempo decrescente) ottenendo un rapporto di prestazioni $3/2$. In realtà è possibile, con un'analisi più precisa, ottenere un rapporto di $4/3$, e questa analisi dell'algoritmo è essenzialmente ottima.

Consideriamo infatti k macchine, con k dispari, e $2k + 1$ compiti con tempi

$$2k-1, 2k-1, 2k-2, 2k-2, 2k-3, 2k-3, \ldots, k+1, k+1, k, k, k.$$

L'algoritmo assegnerà inizialmente i k compiti con tempi

$$2k-1, 2k-1, 2k-2, 2k-2, 2k-3, 2k-3, \ldots, \left\lceil \frac{3}{2}k \right\rceil, \left\lceil \frac{3}{2}k \right\rceil, \left\lfloor \frac{3}{2}k \right\rfloor$$

a macchine distinte. Da lì in poi procederà in ordine inverso, cioè partendo dalla macchine con assegnato il compito più lieve, assegnando ulteriori k compiti: alla fine del processo, tutte le macchine avranno un carico di $3k - 1$, e quindi ovunque venga piazzato l'ultimo compito il costo della soluzione sarà $4k - 1$.

È però possibile risolvere il problema con costo $3k$: basta aggregare i tre compiti di costo k su una macchina, e sulle macchine restanti accoppiare un compito di tempo $2k - 1$ con un compito di tempo $k + 1$, un compito di tempo $2k - 2$ con un compito di tempo $k + 2$, e così via.

2 Inapprossimabilità di MAXDISJOINTPATHS

Ricordiamo che un'istanza di MAXDISJOINTPATHS è data da un grafo e da una sequenza di k sorgenti/destinazioni $\langle s_i, t_i \rangle$, $i \in k$. Il problema chiede di connettere il numero massimo possibile di coppie tramite un cammino senza che un arco venga usato più di una volta (c'è anche una versione con capacità, in cui un arco può essere utilizzato c volte, ma il risultato ottimo di inapprossimabilità è solo per la versione $c = 1$, che sappiamo avere un algoritmo di $O(\sqrt{m})$-approssimazione, dove m è il numero di archi del grafo).

Per mostrare che MAXDISJOINTPATHS è inapprossimabile meglio di $O(\sqrt{m})$, consideriamo un'istanza di 2DIRPATH, che è un problema NP-completo. Un'istanza di 2DIRPATH è data da un grafo e due coppie $\langle s, t \rangle$ e $\langle u, v \rangle$ di vertici, e si pone la domanda se sia possibile collegare s a t e u a v tramite cammini che non passano per lo stesso vertice.

A noi in realtà interessa una versione in cui i due cammini non passano per lo stesso arco. È però molto facile dimostrare che anche la versione *arco*-disgiunta di 2DIRPATH è NP-completa. Data un'istanza $G = \langle V, A \rangle$ della versione vertice-disgiunta di 2DIRPATH, consideriamo un

grafo G' avente $2|V|$ vertici: il grafo contiene una copia x^- e una copia x^+ di ogni vertice di G. Aggiungiamo al grafo un arco $x^- \to x^+$ per ogni vertice di G, e trasformiamo ogni arco $x \to y$ di G in un arco $x^+ \to y^-$. I cammini arco-disgiunti in questo nuovo grafo devono andare da s^+ a t^- e da u^+ a v^-.

È immediato dimostrare che ogni coppia di cammini vertice-disgiunta in G dà origine a una coppia di cammini arco-disgiunta in G', e viceversa. Quindi, la versione arco-disgiunta di 2DIRPATH è NP-completa.

Supponiamo ora che esista un ipotetico algoritmo con fattore di approssimazione $m^{1/2-\varepsilon}$, per un qualunque $\varepsilon > 0$, per MAXDISJOINTPATHS, e consideriamo un'istanza $G = \langle V, A \rangle$ della versione arco-disgiunta di 2DIRPATH.

Andiamo a costruire un'istanza di MAXDISJOINTPATHS con $|A|^{1/\varepsilon}$ coppie da connettere come in figura 1 (nella figura, $|A|^{1/\varepsilon} = 5$). I due cammini all'interno delle copie dell'istanza di 2DIRPATH rappresentano due ipotetici cammini da s a t e da u a v, ma a seconda del tipo di istanza è possibile passare attraverso entrambi i cammini, o attraverso uno solo. Dato che utilizziamo sempre la stessa istanza di 2DIRPATH, G, l'istanza agisce come uno "scambio" che permette di far passare due cammini in ogni incrocio, o in nessun incrocio.

Si noti che se l'istanza di 2DIRPATH è un'istanza-no è possibile collegare *una sola* coppia di vertici. Qualunque coppia venga collegata, il cammino ostruirà tutte le altre, dato che non è possibile passare due volte attraverso G.

Il grafo contiene (a meno di fattori costanti) $m = |A|^{2/\varepsilon}|A| = |A|^{2/\varepsilon+1}$ lati. L'ipotetico algoritmo con fattore di approssimazione $m^{1/2-\varepsilon}$ avrebbe quindi un fattore di approssimazione (espresso in $|A|$)

$$(|A|^{2/\varepsilon+1})^{1/2-\varepsilon} = |A|^{1/\varepsilon-\varepsilon-3/2} = |A|^{1/\varepsilon}/|A|^{3/2+\varepsilon},$$

che è evidentemente minore di $|A|^{1/\varepsilon}$ (dato che $3/2 + \varepsilon > 1$). Un tale algoritmo permetterebbe quindi di distinguere l'istanza con soluzione ottima $|A|^{1/\varepsilon}$ (corrispondente a un'istanza-sì di 2DIRPATH) da quello con soluzione ottima 1 (corrispondente a un'istanza-no di 2DIRPATH).

3 Approssimazione di MAXE3SAT

È possibile in tempo polinomiale soddisfare almeno $7/8$ delle clausole di un'istanza di MAXE3SAT (soddisfare il massimo numero di clausole booleane contenenti *esattamente* tre letterali), o più in generale almeno $(2^k - 1)/2^k$ clausole di MAXEkSAT. Con la stessa tecnica è possibile soddisfare $1/2$ delle clausole di un'istanza di MAXSAT.

L'idea è quella di *derandomizzare* un algoritmo randomizzato. L'algoritmo estrae semplicemente un assegnamento a caso.

Qual è il numero medio di clausole di un'istanza di MAXE3SAT soddisfatte in questo modo? Notiamo subito che tra gli otto possibili assegnamenti delle variabili di una clausola, uno rende

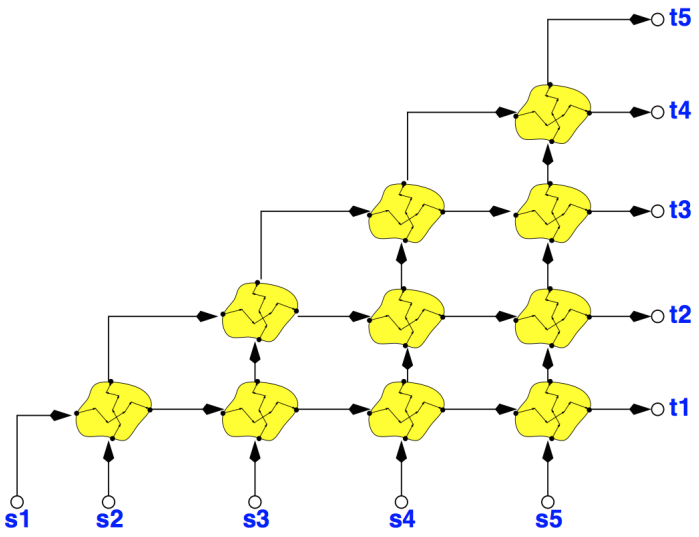


Figura 1: La costruzione dell'istanza di MAXDISJOINTPATHS che ne dimostra l'inapprossimabilità.

la clausola falsa e sette rendono la clausola vera. Quindi la probabilità che una specifica clausola sia vera, dato un assegnamento estratto a caso uniformemente, è $7/8$.

Qual è il numero atteso di clausole soddisfatte da un assegnamento estratto a caso uniformemente? Denotiamo con $f_i : 2^3 \to 2$ la funzione di verità a tre argomenti della clausola i, $i \in n$, dove n è il numero di clausole.

Il numero di clausole soddisfatte da un certo assegnamento alle variabili della formula è dato dalla somma delle f_i, valutate sui valori assegnati alle variabili della clausola i. La somma delle f_i è quindi una funzione $C : 2^k \to \mathbf{N}$ (dove k è il numero di variabili) che associa a ogni possibile assegnamento il numero di clausole soddisfatte.

Se denotiamo con X_i una variabile aleatoria che assume valore 0 o 1 equiprobabilmente,

$$C(X_0, X_1, \ldots, X_{k-1})$$

è una variabile aleatoria che ha come valore, per ogni assegnamento possibile, il numero di clausole soddisfatte.

Possiamo facilmente calcolare il valore atteso di $C(X_0, X_1, \ldots, X_{k-1})$ perché è, per linearità, la somma dei valori attesi delle f_i (ciascuna valutata sulle sue variabili di competenza), e dato che ogni f_i assume valore 0 con probabilità $1/8$ e valore 1 con probabilità $7/8$ concludiamo che il valore atteso di $C(X_0, X_1, \ldots, X_{k-1})$ è $7/8n$. Questo risultato ci fornisce un algoritmo randomizzato che ha come valore atteso del risultato l'approssimazione richiesta.

Passiamo ora a *derandomizzare* l'algoritmo utilizzando il *metodo delle probabilità condizionate*. Supponiamo di avere dimostrato

$$\mathbf{E}\big[C(X_0, X_1, \ldots, X_{k-1}) \mid X_0 = b_0, X_1 = b_1, \ldots X_{j-1} = b_{j-1}\big] \geq \frac{7}{8}n$$

per un $0 \leq j < k$ e $b_i \in 2$. Ovviamente per $j = 0$ non c'è condizionamento e l'asserto è quello che abbiamo dimostrato. Per definizione di valore atteso condizionato abbiamo

$$\begin{aligned}\frac{7}{8}n &\leq \mathbf{E}\big[C(X_0, X_1, \ldots, X_{k-1}) \mid X_0 = b_0, X_1 = b_1, \ldots X_{j-1} = b_{j-1}\big] \\ &= \frac{1}{2}\mathbf{E}\big[C(X_0, X_1, \ldots, X_{k-1}) \mid X_0 = b_0, X_1 = b_1, \ldots X_{j-1} = b_{j-1}, X_j = 0\big] \\ &\quad + \frac{1}{2}\mathbf{E}\big[C(X_0, X_1, \ldots, X_{k-1}) \mid X_0 = b_0, X_1 = b_1, \ldots X_{j-1} = b_{j-1}, X_j = 1\big]\end{aligned}$$

Deve esseci quindi un $b_j \in 2$ tale che

$$\mathbf{E}\big[C(X_0, X_1, \ldots, X_{k-1}) \mid X_0 = b_0, X_1 = b_1, \ldots X_{j-1} = b_{j-1}, X_j = b_j\big] \geq \frac{7}{8}n.$$

Trovare b_j non è difficile: per ogni clausola, valutiamo i letterali già assegnati. Avremo clausole con valore costante (che avranno valore atteso zero o uno), clausole con un letterale rimasto (che avranno valore atteso $1/2$), clausole con due letterali rimasti (valore atteso $3/4$) e clausole con tre letterali rimasti (valore atteso $7/8$). Sommando i valori attesi, otteniamo il valore atteso condizionato di $C(X_0, X_1, \ldots, X_{k-1})$, che ci permette di scegliere b_j. Infine, quando $j = k - 1$ otteniamo un assegnamento $b_0, b_1, \ldots, b_{k-1}$ che soddisfa almeno $7/8$ delle clausole.

Nel caso di MAXSAT possiamo solo garantire la soddisfacibilità di metà delle clausole perché le funzioni di verità delle clausole formate da un solo letterale hanno valore atteso $1/2$, e quindi possiamo solo dire che il valore il valore atteso $C(X_0, X_1, \ldots, X_{k-1})$ è maggiore o uguale a $n/2$. La dimostrazione procede poi allo stesso modo del caso MAXE3SAT.

4 Arrotondamento aleatorio per SET COVER

L'alea può essere utilizzata per ottenere soluzioni discrete a partire da un rilassamento lineare utilizzando la tecnica dell'*arrotondamento aleatorio* (*randomized rounding*). Come esempio, consideriamo la versione di programmazione intera di SET COVER, dove l'universo U contiene n elementi e dobbiamo coprirlo utilizzando i sottoinsiemi di U dati $S_0, S_1, \ldots, S_{k-1}$, con pesi associati $w_0, w_1, \ldots, w_{k-1}$. Le variabili $x_0, x_1, \ldots, x_{k-1}$ a valori 0 o 1 sono indicatori dell'appartanenza dell'insieme S_i alla soluzione, e le equazioni, una per ogni elemento v di U,

$$\sum_{v \in S_i} x_i \geq 1$$

garantiscono che se x_i ha valore 0 o 1 l'elemento v è certamente coperto. La funzione da minimizzare è ovviamente

$$\sum_{i=0}^{k-1} w_i x_i.$$

Se prendiamo in considerazione una soluzione $\bar{x}$ del rilassamento lineare, è evidente che la strategia di arrotondamento che abbiamo utilizzato per VERTEX COVER non potrà funzionare: un elemento può appartenere a insiemi di cardinalità arbitraria, mentre i lati di un grafo sono tutti sottoinsiemi di cardinalità due. Sappiamo però come al solito che $\bar{c} = \sum_{i=0}^{k-1} w_i \bar{x}_i$ è una minorazione del costo di una soluzione ottima.

Utilizziamo quindi la seguente *procedura di scelta*: per ogni i, inseriamo l'indice i nella soluzione I con probabilità $\bar{x}_i$.

La procedura di scelta così descritta lascerà molto probabilmente degli elementi di U scoperti. Possiamo però controllare la probabilità che uno specifico elemento v rimanga scoperto:

$$\prod_{v \in S_i} (1 - x_i) \leq \prod_{v \in S_i} e^{-x_i} \leq e^{-\sum_{v \in S_i} x_i} \leq e^{-1}.$$

Ne consegue che se ripetiamo la procedura $k + \ln n$ volte e consideriamo l'unione degli insiemi soluzione così generati, probabilità che un elemento non sia coperto è controllata da

$$\left(\frac{1}{e}\right)^{k+\ln n} = \frac{1}{n} e^{-k},$$

e utilizzando il limite per l'unione la probabilità che ci sia un qualche elemento scoperto è meno di e^{-k}.

La seconda osservazione è che se iteriamo la procedura di scelta $k + \ln n$ volte, il valore atteso della funzione obiettivo sull'insieme unione di tutte le $k + \ln n$ scelte generate è minore o uguale a $(k + \ln n)\bar{c}$ per linearità, e per la diseguaglianza di Markov sarà maggiore o uguale a $q(k + \ln n)\bar{c}$ con probabilità al più $1/q$ per ogni $q > 1$.

Si tratta solo, a questo punto, di tarare k: se scegliamo $k = 3$ la probabilità che non copriamo tutti gli elementi è meno di $e^{-3} \leq 0{,}05$, mentre la probabilità che ottieniamo una soluzione maggiore di $2(3 + \ln n)\bar{c}$ è minore di $1/2$. Messi insieme i due eventi, abbiamo probabilità maggiore o uguale a $0{,}45$ che l'algoritmo copra tutti gli elementi con un rapporto di prestazioni al più $6 + 2 \ln n$.

5 L'algoritmo di Miller–Rabin

Discutiamo ora un celebre algoritmo probabilistico, l'algoritmo di Miller–Rabin [**?**]. L'algoritmo utilizza dei bit aleatori per stabilire se un numero è composto: se la risposta è positiva, la

risposta è sempre corretta; se la risposta è negativa (cioè il numero viene considerato primo) c'è una probabilità di errore inferiore a $1/2$. Si noti che la probabilità d'errore è sui bit aleatori, e non sull'input. Denotiamo con $\mathbf{Z}_n$ l'anello delle classi di resto modulo n, e con $\mathbf{U}_n$ il suo *gruppo delle unità*, cioè il gruppo moltiplicativo formato dagli elementi diversi da zero e invertibili (vale dire, coprimi con n).

Definizione 1 Sia n un numero dispari, e $n-1 = 2^a b$, con b dispari. Dato $t \in \mathbf{Z}_n \setminus \{0,1\}$, consideriamo la sequenza

$$t^b, t^{2b}, t^{2^2 b}, t^{2^3 b}, \ldots, t^{n-1}$$

(cioè gli a quadrati successivi a partire da t^b). Diciamo che t è un *testimone di composizione di* n se $t \not\perp n$, se $t^{n-1} \neq 1$ o se esiste un i per cui $t^{2^i b} \neq \pm 1$ e $t^{2^{i+1} b} = 1$.

In sostanza, un testimone è un elemento di $\mathbf{Z}_n$ che ci mostra che n ha un divisore proprio, che $n-1$ non è un multiplo dell'ordine di $\mathbf{U}_n$ o che esistono radici quadrate dell'unità diverse da ± 1. Come vedremo, in tutti e tre i casi n non può essere primo. Dobbiamo ora mostrare che ci sono abbastanza testimoni, e a questo scopo utilizzeremo un risultato fondamentale di teoria dei numeri:

Teorema 1 (cinese del resto) Se $c \perp d$, allora

$$\mathbf{Z}_{cd} \cong \mathbf{Z}_c \times \mathbf{Z}_d.$$

Utilizzeremo anche alcune nozioni relative agli anelli di resti[1], e in particolare il seguente risultato:

Lemma 1 Sia n un numero composto dispari che non è una potenza di primo, e sia $m = n-1 = 2^a b$, con b dispari e $a > 0$. Allora, per almeno la metà degli elementi x di $\mathbf{U}_n$ tali che $x^m = 1$, la sequenza

$$x^b, x^{2b}, x^{2^2 b}, x^{2^3 b}, \ldots, x^{2^{a-1} b}, x^m = 1$$

contiene una radice quadrata dell'unità diversa da ± 1.

Dimostrazione. Sia $H = \{x \in \mathbf{U}_n \mid x^m = 1\}$. Vale a dire, H è il *sottogruppo delle radici m-esime dell'unità*. Notiamo innanzitutto che l'esponenziazione (per qualunque esponente fissato) è un endomorfismo[2] di H. Quindi, esiste un $j \in \mathbf{N}$ tale che l'endomorfismo di esponenziazione

[1] In quanto segue, quando x è un elemento di un ben specificato anello di resti $\mathbf{Z}_n$ utilizzeremo le normali notazioni di somma, prodotto e esponenziazione per indicare le corrispondenti operazioni in $\mathbf{Z}_n$, risparmiando la tediosa ripetizione della scrittura "mod n". Questa scelta può provocare qualche ambiguità se utilizziamo x anche come un particolare resto (per esempio, scelto in $\{0, 1, \ldots, n-1\}$). La confusione non provoca però danni se nel fare ciò è indifferente quale specifico resto scegliere; per esempio, la scrittura $x \perp n$ ha senso, perché $x \perp n \iff x + kn \perp n$.

[2] Un *endomorfismo* di un gruppo X è un omomorfismo di gruppi da X in X.

alla $2^{j+1}b$ ha come immagine l'unità, mentre l'endomorfismo α di esponenziazione alla $2^j b$ ha come immagine un sottogruppo non banale di H interamente costituito da radici quadrate dell'unità. In H esistono certamente altre radici oltre a 1 e -1, perché dato che n è prodotto di almeno due interi coprimi, il teorema cinese del resto garantisce l'esistenza in $\mathbf{U}_n$ di almeno quattro radici quadrate associate agli elementi $\langle 1, 1\rangle$, $\langle -1, 1\rangle$,$\langle 1, -1\rangle$ e $\langle -1, -1\rangle$ di $\mathbf{Z}_c \times \mathbf{Z}_d$, ove $n = cd$ e $c \perp d$, ed essendo m pari, H contiene tutte le radici quadrate dell'unità. Abbiamo quindi due casi:

1. L'immagine di α contiene solo 1 e una radice $r \neq \pm 1$. In questo caso, tutti gli elementi mappati in r da α generano sequenze contenenti r; inoltre, dato che in un omomorfismo di gruppi le retroimmagini di tutti gli elementi hanno la stessa cardinalità, esattamente $|H|/2$ elementi vengono mappati in r, e quindi almeno $|H|/2$ elementi generano sequenze contenenti r, come volevasi dimostrare.

2. L'immagine di α contiene 1 e almeno due radici diverse da ± 1. Anche in questo caso, è immediato mostrare che almeno metà degli elementi di H generano le sequenze desiderate.

Gli altri casi possono essere scartati come segue: se l'immagine di α contiene -1, dato che quest'ultimo è associato (nell'isomorfismo del teorema cinese del resto) alla coppia $\langle -1, -1\rangle$, esistono $x \in \mathbf{Z}_c$ e $y \in \mathbf{Z}_d$ tali che $x^{2^j b} = -1$ (in $\mathbf{Z}_c$) e $y^{2^j b} = -1$ (in $\mathbf{Z}_d$). In tal caso, $\langle x, 1\rangle^{2^j b} = \langle -1, 1\rangle$ e $\langle 1, y\rangle^{2^j b} = \langle 1, -1\rangle$, e quindi almeno due radici diverse da ± 1 devono appartenere all'immagine di α (sono potenze $2^j b$-esime degli elementi corrispondenti alle coppie $\langle x, 1\rangle$ e $\langle 1, y\rangle$). ∎

Teorema 2 (Miller [?]) Se n è primo, non ha testimoni. Se n è un numero dispari composto che non è una potenza di primo, almeno metà dei $t \in \mathbf{Z}_n \setminus \{0\}$ sono testimoni.[3]

Dimostrazione. Si noti innanzitutto che se n è primo tutti i numeri sono coprimi con n, e $n - 1$ è l'ordine del gruppo moltiplicativo di $\mathbf{Z}_n$; quindi, $x^{n-1} = 1$ per ogni $x \in \mathbf{Z}_n \setminus \{0\}$. Inoltre l'equazione $x^2 \equiv 1 \mod n$ ha esattamente le due soluzioni 1 e -1, dato che implica $n \mid (x - 1)(x + 1)$; quindi non esistono testimoni di n quando n è primo.

È chiaro che per concludere la dimostrazione è sufficiente mostrare che almeno metà degli elementi di $\mathbf{U}_n$ sono testimoni, dato che tutti gli elementi in $\mathbf{Z}_n \setminus (\mathbf{U}_n \cup \{0\})$, non essendo coprimi con n, sono testimoni. Inoltre, poiché gli elementi di $\mathbf{U}_n$ che non sono radici $(n - 1)$-esime dell'unità sono pure testimoni, è sufficiente dimostrare che almeno metà delle radici stesse sono testimoni. Ma questo è il contenuto del lemma 1. ∎

[3] È in realtà possibile mostrare (al prezzo di una dimostrazione del lemma 1 più complessa e cablata su $n - 1$ anziché su un pari qualunque) che almeno *tre quarti* degli elementi di $\mathbf{Z}_n \setminus \{0\}$ sono testimoni.

Abbiamo a questo punto un semplice algoritmo probabilistico polinomiale con errore unilaterale (appunto, l'algoritmo di Miller–Rabin) per stabilire se un numero dato n è composto[4]:

1. se n è pari è composto;
2. altrimenti, per bisezione controlliamo se n è una potenza (basta controllare gli esponenti da 2 a $\log n$; dobbiamo quindi operare $\log n$ ricerche da $\log n$ passi, e ogni passo richiede solo un'esponenziazione), e in caso positivo concludiamo che n è composto;
3. estraiamo in maniera uniforme un numero casuale $t \in \mathbf{Z}_n \setminus \{0\}$;
4. se $t \not\perp n$ concludiamo che n è composto;
5. se $t^{n-1} \neq 1$, concludiamo che n è composto;
6. infine, calcoliamo a e b tali che $n - 1 = 2^a b$ con b dispari e costruiamo la sequenza $t^b, t^{2b}, t^{2^2 b}, \ldots, t^{n-1}$. Se nella sequenza compare una radice dell'unità diversa da ± 1 concludiamo che n è composto;
7. concludiamo che n è primo.

Tutti i conti coinvolti sono banalmente eseguibili in tempo polinomiale (peraltro, con esponente basso) utilizzando tecniche standard di esponenziazione modulare. Per il teorema precedente, possiamo arrivare all'ultimo passo con n composto con probabilità al più $\frac{1}{2}$. Quindi ripetendo l'algoritmo k volte con esito negativo la possibilità di errore nello stabilire che n è primo è al più 2^{-k}.

Miller ha anche dimostrato [**?**] che se vale l'ipotesi generalizzata di Riemann esistono certamente certificati piccoli, e in tal caso è possibile trovarli in tempo quartico (nella lunghezza di n). Esiste quindi una versione deterministica polinomiale dell'algoritmo di Miller–Rabin, ma non siamo al momento in grado di dimostrare che dia risultati corretti.

6 Inapprossimabilità del commesso viaggiatore

Il problema del commesso viaggiatore generico (cioè senza restrizioni, per esempio di tipo metrico) si può facilmente dimostrare essere inapprossimabile per qualunque costante r con una tecnica molto simile a quella utilizzata per CENTERSELECTION.

Partiamo dal fatto che stabilire se un grafo non orientato $G = \langle V, E \rangle$ possiede un ciclo hamiltoniano (cioè un ciclo che passa esattamente una sola volta per ogni vertice) è NP-completo.

[4] Si noti che questo algoritmo è in effetti utilizzato in pratica per *produrre* numeri primi. La sua utilità è legata al fatto che il teorema fondamentale dell'aritmetica stabilisce che il numero di primi minori di n è asintotico a $n/\ln n$, e quindi i primi sono abbastanza densi da rendere computazionalmente ragionevole una ricerca esaustiva o aleatoria.

Mostriamo ora come assumere l'esistenza di un algoritmo polinomiale r-approssimante per il commesso viaggiatore permetterebbe di riconoscere in tempo polinomiale l'esistenza di un ciclo hamiltoniano.

A partire da G, costruiamo un'istanza G' del commesso viaggiatore con insieme di vertici V in cui

$$d(x, y) = \begin{cases} 1 & \text{se } \langle x, y\rangle \in E \\ \lceil nr \rceil + 1 & \text{se } \langle x, y\rangle \notin E, \end{cases}$$

ove $n = |V|$. Si noti la somiglianza con la costruzione utilizzata per CENTERSELECTION, dove utilizzavamo il peso 2 per le distanze associate a coppie di vertici non adiacenti in G.

Ora, è immediato che G ha un ciclo hamiltoniano se e solo se G' ha un ciclo hamiltoniano di peso n. Solo tale ciclo, infatti, usa solo lati di G. Qualunque ciclo hamiltoniamo di G' che utilizza un lato non in G ha peso almeno $\lceil nr \rceil + 1$. Se potessimo quindi approssimare il commesso viaggiatore meglio di $(\lceil nr \rceil + 1)/n > r$ potremmo decidere se G ha un ciclo hamiltoniano in tempo polinomiale.

7 NPO

Un problema A di ottimizzazione in NPO è dato da:

1. Un insieme di *istanze* riconoscibili in tempo polinomiale.[5]
2. Per ogni istanza x, un insieme di soluzioni [accettabili], che devono essere corte (esiste un polinomio p tale che $|y| \leq p(|x|)$ per ogni istanza x e per ogni soluzione y di x) e riconoscibili in tempo polinomiale (per ogni y di lunghezza inferiore a $p(|x|)$, è possibile riconoscere in tempo polinomiale se y è una soluzione di x).
3. Una funzione c_A che data un'istanza x e una sua soluzione y restituisce il *costo* (intero) di y (come soluzione di x). La funzione c_A deve essere calcolabile in tempo polinomiale.
4. Un tipo $t \in \{\max, \min\}$.

NPO è la classe dei problemi di ottimizzazione i cui associati problemi di decisione, per qualunque costante, sono in NP. La soluzione di un problema in NPO è un algoritmo che a fronte di un'istanza x trova una soluzione y *ottima*, cioè con costo

$$c_A(x, y) = t\{c_A(x, z) \mid z \text{ soluzione di } x\}.$$

Doteremo con $\text{opt}_A(x)$ il costo di una soluzione ottima per l'istanza x di A.

[5] Istanze e soluzioni sono assunte essere stringhe binarie.

Per esempio, MAXSAT è il problema le cui istanze sono le formule booleane in forma normale congiunta (e quindi esprimibile come insieme di clausole, che sono insiemi di letterali, cioè variabili o negazioni di variabili), le soluzioni sono assegnamenti di valori di verità alle variabili, e la funzione di costo è il numero di clausole soddisfatte dall'assegnamento. Una soluzione ottima soddisfa il più grande numero di clausole possibili.

MAX3SAT è invece la versione di MAXSAT in cui le clausole sono tutte formate da al più tre letterali. MAXCLIQUE ha come istanze i grafi non orientati, e come soluzioni sottoinsiemi di vertici mutuamente adiacenti (cricche). La funzione di costo è la cardinalità del sottoinsieme, e una soluzione ottima fornisce un insieme di cardinalità più grande possibile.

Per ogni soluzione y di un'istanza x di un problema A in NPO esiste un *rapporto di prestazioni*

$$R_A(x, y) = \max \left\{ \frac{c_A(x, y)}{\mathrm{opt}_A(x)}, \frac{\mathrm{opt}_A(x)}{c_A(x, y)} \right\}$$

che esprime la bontà della soluzione come un numero nell'intervallo $[1 \,..\, \infty)$ *indipendentemente dal tipo di problema* (min/max). Infatti, il massimo viene realizzato dal primo argomento nel caso di problemi di minimizzazione, e dal secondo nel caso di problemi di massimizzazione.

In questo modo, è possibile dare un concetto di "algoritmo r-approssimato" che non dipende dal tipo di problema: occorre che indipendentemente dall'istanza x e dalla soluzione y calcolata dall'algoritmo si abbia

$$R_A(x, y) \leq r.$$

La scelta è più discutibile per i problemi di massimizzazione, dove l'affermazione, per esempio, che il rapporto di prestazione dell'algoritmo per MAXE3SAT basato sulle probabilità condizionate è $8/7$ può sembrare innaturale (rispetto a $7/8$, che è effettivamente la frazione di clausole soddisfatta).

Si noti che è possibile mappare linguaggi in NP in problemi in NPO in maniera che il costo della soluzione ottima del problema in NPO per l'istanza x sia 0 o 1 a seconda che esista o meno un testimone di appartenenza per x. In particolare, calcolare la soluzione ottima del problema in NPO in tempo polinomiale implica riconoscere il linguaggio in NP in tempo polinomiale.

Per esempio, SAT può essere mappato in un problema di NPO in cui le istanze sono le formule booleane in forma normale congiunta, le soluzioni sono assegnamenti di valori di verità alle variabili, e la funzione di costo vale 1 se l'assegnamento rende vera la formula, 0 altrimenti. Chiaramente questa definizione soddisfa tutte le condizioni per un problema in NPO, e data un'istanza x il costo ottimo è 1 se e solo se la formula è soddisfacibile, cioè se esiste un assegnamento che la rende vera. In particolare, un algoritmo polinomiale in grado di trovare la soluzione ottima riconoscerebbe SAT in tempo polinomiale.

Si noti però una differenza importante: un algoritmo polinomiale per SAT non può essere utilizzato in linea di principio per risolvere il problema in NPO, perché l'unico risultato dall'esecuzione dell'algoritmo per SAT è una risposta sì/no, mentre risolvere il problema in NPO

richiede di produrre una soluzione ammissibile, e per ragioni generali di complessità questo potrebbe non essere possibile in tempo polinomiale (per esempio, 3-colorare un grafo 3-colorabile è NP-difficile).

Quest'idea è estendibile a qualunque problema di NP: dato un linguaggio L in NP, prendiamo come insieme di istanze 2^*, consideriamo una macchina di Turing che verifica L in tempo $p(|x|)$ su istanza $|x|$, e prendiamo come soluzioni ammissibili dell'istanza x tutte le stringhe binarie di lunghezza $p(|x|)$, assegnando alle soluzioni che sono testimoni di appartenenza a NP il costo 1, e a quelli che non lo sono il costo zero. Il verificatore in questo caso garantisce che la funzione di costo sia calcolabile in tempo polinomiale. In questo modo, l'esistenza di una soluzione ottima di costo 1 per l'istanza x è equivalente all'esistenza di un testimone per l'appartenenza di x a L.

In sostanza, le soluzioni ammissibili giocano nel caso di NPO lo stesso ruolo giocato dai testimoni d'appartenenza nel caso di NP. Il caso di NPO è più generale da un lato perché la funzione costo è arbitraria, e non è ristretta ai valori 0 e 1, ma soprattutto perché per risolvere il problema dobbiamo fornire una soluzione corta e verificabile in tempo polinomiale.

7.1 APX

APX è la sottoclasse di problemi di NPO che ammettono un algoritmo di approssimazione con rapporto di prestazioni controllato da una costante. Esiste cioè una costante r, indipendente dall'istanza x e dalla soluzione y, tale che $R_A(x, y) \leq r$.

Per esempio, MAXE3SAT è in APX (dato che il suo rapporto di prestazioni è minore o uguale a $8/7$) mentre MAXDISJOINTPATHS non lo è (dato che ci sono istanze le cui soluzioni hanno rapporto di prestazione $\Omega(\sqrt{m})$. Quindi APX $\neq$ NPO.

7.2 PTAS

PTAS è la sottoclasse di problemi di APX che ammettono uno schema di approssimazione polinomiale: esiste cioè un algoritmo che dato r e un'istanza x calcola in tempo polinomiale in $|x|$ (ma non necessariamente in r) una soluzione. Per esempio, il problema dello zaino è in PTAS, mentre CENTERSELECTION è in APX ma non è in PTAS, dato che abbiamo dimostrato che non esistono algoritmi polinomiali con rapporto di prestazione inferiore a due. Dunque PTAS $\neq$ APX.

8 Problemi di gap con promessa

Un *problema (di decisione) L con promessa* è specificato da due insiemi $L_{\text{yes}}, L_{\text{no}} \subseteq 2^*$ che soddisfano $L_{\text{yes}} \cap L_{\text{no}} = \varnothing$. La differenza fondamentale con un normale problema di decisione è che non necessariamente tutte le stringhe in 2^* sono istanze-sì o istanze-no: possono esistere

in generale stringhe che non appartengono né a L_{yes}, né a L_{no}, e nella definizione delle classi noi teniamo conto *solo* delle stringhe in $L_{\text{yes}} \cup L_{\text{no}}$: intuitivamente, *promettiamo* alla macchina di Turing che deve riconoscere L di dare in input solo stringhe che sono istanze. Ogni problema di decisione standard è semplicemente un problema con promessa vuota.

I problemi con promessa permettono di codificare in maniera naturale i risultati di inapprossimabilità tramite gap: per esempio, abbiamo dimostrato che non è possibile distinguere certe istanze di CENTERSELECTION che hanno soluzione ottima 1 o 2, a meno che P = NP. Questo risultato si può tradurre nel fatto che il problema con promessa che ha come istanze-sì le istanze di CENTERSELECTION con soluzione ottima 1 e ha come istanze-no le istanze di CENTERSELECTION con soluzione ottima 2 è NP-difficile. Si noti che in questa formalizzazione il problema di gap con promessa risultante è un problema di *decisione*, non di *ottimizzazione*, e quindi possiamo utilizzare gli strumenti tipici dei problemi di decisione.

In particolare, dati due problemi con promessa L e M una *riduzione* da L a M è una funzione $f : 2^* \to 2^*$, calcolabile in tempo polinomiale, tale che

$$\begin{aligned} x \in L_{\text{yes}} &\implies f(x) \in M_{\text{yes}} \\ x \in L_{\text{no}} &\implies f(x) \in M_{\text{no}} \end{aligned}$$

Vale a dire, f mappa istanze-si in istanze-si e istanze-no in istanze-no (si noti che la definizione tradizionale di riduzione, cioè "x è un'istanza-si di L se e solo se $f(x)$ è un'istanza-si di M, in questo caso, non è sufficiente, perché non esclude che istanze-no siano mappate al di fuori della promessa).

È evidente che se M è trattabile (cioè esiste una macchina di Turing che, a fronte di un input in $M_{\text{yes}} \cup M_{\text{no}}$, risponde correttamente in tempo polinomiale), anche L è trattabile: e quindi se L è difficile, anche M è difficile. Possiamo quindi propagare risultati di difficoltà di approssimazione verso nuovi problemi costruendo un problema di gap con promessa opportuno, e una riduzione opportuna. Vediamo alcuni esempi.

8.1 MAXE3SAT e MAXINDEPENDENTSET

Un problema di gap con promessa di MAXE3SAT si riduce a un problema di gap con promessa equivalente di MAXINDEPENDENTSET.

Utilizzando la riduzione classica da 3SAT a INDEPENDENTSET, cioè quella che associa a ogni clausola con k letterali una k-cricca i cui vertici sono decorati con i rispettivi letterali, e connette vertici di cricche diverse associati a letterali opposti, è facile vedere che il numero di clausole soddisfatte di un'istanza di MAXE3SAT è esattamente la dimensione del massimo insieme indipendente del grafo associato. Dato che le clausole di MAXE3SAT contengono esattamente tre letterali, il grafo associato a una formula con n clausole ha esattamente $3n$ vertici.

Se consideriamo un problema di gap con promessa che chiede di distinguere formule di MAXE3SAT soddisfacibili da formule con al più $1-\varepsilon$ clausole soddisfacibili (cioè con gap $1/(1-\varepsilon)$), vediamo subito che le formule vengono mappate in grafi con $3n$ vertici, dove n è il numero di vertici della formula. Nel caso soddisfacibile, il grafo ha un insieme indipendente di dimensione esattamente n, mentre nel secondo caso non può esistere un insieme indipendente di dimensione superiore a $(1-\varepsilon)n$. Il gap associato a questa promessa è quindi di nuovo $1/(1-\varepsilon)$, e non è quindi possibile approssimare meglio MAXINDEPENDENTSET. In particolare, utilizzando il risultato ottimo di Håstad $\varepsilon = \frac{1}{8}$ abbiamo che MAXINDEPENDENTSET non è approssimabile meglio di $\frac{8}{7}$, esattamente come MAXE3SAT.

8.2 MAXINDEPENDENTSET e MINVERTEXCOVER

Il problema di gap con promessa di MAXINDEPENDENTSET appena descritto può essere ridotto a un problema di gap con promessa per MINVERTEXCOVER. Dato che il complemento di un insieme indipendente è esattamente un insieme di vertici di copertura, la promessa diventa la seguente: si considerano grafi di dimensione $3n$ per qualche $n \geq 1$, e si sa che esiste un insieme di vertici di copertura più piccolo di $3n - n = 2n$, oppure uno maggiore di $3n - (1-\varepsilon)n = 2n + \varepsilon n$. Il gap associato è quindi $(2+\varepsilon)n/2n = 1 + \varepsilon/2$. Si noti che

$$1 + \frac{\varepsilon}{2} < 1 + \frac{\varepsilon}{1-\varepsilon} = \frac{1}{1-\varepsilon}$$

dato che $\varepsilon < 1$. Di nuovo, utilizzando il risultato ottimo di Håstad $\varepsilon = \frac{1}{8}$ abbiamo che MINVERTEXCOVER non è approssimabile meglio di $1 + \frac{1}{16} \approx 1{,}06$.

8.3 MAXE3SAT e MAX2SAT

Vediamo ora come ridurre un problema di gap con promessa di MAXE3SAT a un problema di gap con promessa di MAX2SAT. La dimostrazione è particolarmente interessante perché è una forma di riuso di una dimostrazione della fine degli anni settanta che provava che MAX2SAT (come problema di decisione) era NP-difficile.

Consideriamo un'istanza φ di MAXE3SAT con n clausole, ciascuna formata dai tre letterali a_i, b_i e c_i, $i \in n$. Associamo a ciascuna clausola dell'istanza il seguente insieme di dieci clausole:

$$a_i, \quad b_i, \quad c_i, \quad z_i, \quad \bar{a}_i \vee \bar{b}_i, \quad \bar{a}_i \vee \bar{c}_i, \quad \bar{b}_i \vee \bar{c}_i, \quad a_i \vee \bar{z}_i, \quad b_i \vee \bar{z}_i, \quad c_i \vee \bar{z}_i,$$

dove z_i è una nuova variabile. Si noti che prima abbiamo i singoli letterali della clausola, poi la nuova variabile, poi tutte le coppie di variabili della clausola originale negate, e infine ogni variabile originale della clausola insieme dalla nuova variabile z_i negata. L'istanza associata di MAX2SAT $\varphi' = f(\varphi)$ sarà l'insieme delle $10n$ clausole suddette.

Si notino ora due fatti:

- per ogni assegnamento dei letterali che rende vera la clausola esiste un valore di z_i che soddisfa sette delle dieci clausole;
- se la clausola non è soddisfatta è impossibile soddisfare più di sei clausole (ed è possibile soddisfare esattamente sei clausole).

Quindi, se prendiamo un problema di gap con promessa di MAXE3SAT che chiede di distinguere formule soddisfacibili da formule con al più una frazione $1 - \varepsilon$ di clausole soddisfacibili, le formule soddisfacibili di MAXE3SAT con n clausole sono ridotte a formule di MAX2SAT di $10n$ clausole di cui esattamente $7n$ soddisfacibili, mentre le formule da n clausole di MAXE3SAT con al più $(1 - \varepsilon)n$ clausole soddisfacibili sono ridotte a formule (sempre di $10n$ clausole) di MAX2SAT con al più $7(1 - \varepsilon)n + 6\varepsilon n$ clausole soddisfacibili.

Già a questo punto possiamo concludere che MAX2SAT *è un problema di decisione NP-difficile*, dato che potrebbe risolvere il problema di gap con promessa che abbiamo appena definito (ricordiamo che 2SAT è risolubile in tempo polinomiale). Per completare la costruzione, calcoliamo il gap:

$$\frac{7n}{7(1-\varepsilon)n + 6\varepsilon n} = \frac{7n}{7n - 7\varepsilon n + 6\varepsilon n} = \frac{7n}{7n - \varepsilon n} = \frac{7}{7 - \varepsilon} = 1 + \frac{\varepsilon}{7 - \varepsilon}.$$

Di nuovo, utilizzando il risultato ottimo di Håstad $\varepsilon = \frac{1}{8}$ otteniamo un gap di $\frac{56}{55}$.

9 Rango e selezione

Le strutture succinte si basano su una serie di primitive di base che forniscono (in genere in tempo costante) informazioni relative a un vettore $\boldsymbol{b}$ di n bit utilizzando spazio addizionale $o(n)$. Tali strutture vengono dette *sistematiche*, per distinguerle da quelle *non-sistematiche*, che invece rappresentano internamente il vettore.

L'operatore di *rango* (*rank*) è definito come segue:

$$\mathrm{rank}_{\boldsymbol{b}}(p) = \left|\left\{i \in p | b_i = 1\right\}\right| = \sum_{0 \le i < p} b_i.$$

Detto altrimenti, il rango alla posizione p è il numero di uni che precedono la posizione p (si noti che il vettore è indicizzato a partire da zero; anche l'indicizzazione da uno compare, sfortunatamente, in letteratura).

Se $\boldsymbol{b}$ è pensato come la rappresentazione di un sottoinsieme X di n, $\mathrm{rank}_{\boldsymbol{b}}(p)$ è semplicemente la cardinalità del sottoinsieme di X formato dagli elementi minori di p. Inoltre, $\mathrm{rank}_{\boldsymbol{b}}(n)$ è semplicemente il numero di bit a uno nel vettore.

L'operazione duale al rango è la *selezione* (*select*), definita da

$$\text{select}_{\boldsymbol{b}}(k) = \max\{p \mid \text{rank}_{\boldsymbol{b}}(p) \leq k\},$$

dove $0 \leq k < \text{rank}_{\boldsymbol{b}}(n)$. Detto altrimenti, $\text{select}_{\boldsymbol{b}}(k)$ è la posizione del k-esimo bit a uno nel vettore (dove il primo uno ha indice zero). Se $\boldsymbol{b}$ è pensato come la rappresentazione di un sottoinsieme X di n, $\text{select}_{\boldsymbol{b}}(k)$ è semplicemente il k-esimo elemento di X (dove gli elementi di X sono indicizzati a partire da zero nell'ordine naturale).

Dalle definizioni è immediato che

$$\begin{aligned}
&\text{rank}_{\boldsymbol{b}}(\text{select}_{\boldsymbol{b}}(k)) = k \\
&\text{select}_{\boldsymbol{b}}(\text{rank}_{\boldsymbol{b}}(p)) > p \text{ se } b_p = 0 \\
&\text{select}_{\boldsymbol{b}}(\text{rank}_{\boldsymbol{b}}(p)) = p \text{ se } b_p = 1
\end{aligned}$$

Si noti, in particolare, che in generale rango più selezione hanno l'effetto di calcolare il *successore* di un elemento, dove il successore in $X \subseteq n$ di un elemento $x \in n$ è il minimo $y \in X$ tale che $y \geq x$.

9.1 Strutture per il rango

La classica struttura per il rango in tempo costante è quella proposta da Jacobson [**?**] e utilizza il cosiddetto *metodo dei quattro russi* (in inglese, *four-Russians trick*). Il metodo, in generale, prevede di memorizzare in una tabella i risultati per piccoli blocchi dell'input (per esempio, le risposte per sottomatrici piccole di un problema matriciale).

La struttura prevede di dividere gli n bit del vettore in *blocchi* di lunghezza $\frac{1}{2}\log n$ e in *superblocchi* di lunghezza $(\log n)^2$. Si noti che una tabella che contiene tutte le risposte possibili per i blocchi è formata da $2^{\frac{1}{2}\log n} \cdot \frac{1}{2}\log n \cdot \log\log n = O(\sqrt{n}\log n \log\log n) = o(n)$ bit.

Per ogni superblocco, memorizziamo il rango alla posizione iniziale del superblocco stesso. Questo richiede $\log n \cdot n/(\log n)^2 = n/\log n = o(n)$ bit. Infine, per ogni blocco memorizziamo il delta di rango rispetto al superblocco, cioè rango alla posizione iniziale del blocco meno il rango alla posizione iniziale del superblocco che lo contiene (che è il numero di uni che intercorrono tra l'inizio del superblocco e l'inizio del blocco). Questo numero è minore di $(\log n)^2$, e quindi ha bisogno di $2\log\log n$ bit per essere scritto. Ma i blocchi sono $O(n/\log n)$, per cui i dati dei blocchi usano $O(n\log\log n/\log n) = o(n)$ bit.

Per calcolare il rango in posizione p, recuperiamo il rango all'inizio del superblocco in cui si trova p; sommando il delta di rango del blocco in cui si trova p abbiamo il numero di uni che precedono l'inizio del blocco che contiene p. A questo punto estraiamo in tempo costante dalla tabella il numero di uni tra l'inizio del blocco e p, lo sommiamo al numero di uni fino all'inizio

del blocco e restituiamo il valore così calcolato. Abbiamo ovviamente effettuato un numero costante di operazioni.

Si noti che sebbene l'occupazione in spazio della struttura sia in teoria asintoticamente evanescente, per valori anche significativi di n è molto grande: per esempio, se $n = 10^6$

$$2^{\frac{1}{2}\log n} \cdot \frac{1}{2} \log n \cdot \log\log n \approx 10^3 \frac{1}{2} \cdot 20 \cdot 5 = 50\,000,$$

e quindi le tabelle precalcolate richiedono il 5% di spazio in più. Se aggiungiamo anche i contatori dei blocchi, abbiamo bisogno di altri

$$2 \log\log n \cdot n / \log n \approx 2 \cdot 5 \cdot 10^6/20 = 500\,000$$

bit, cioè il 50% di spazio in più.

Implementazioni pratiche. In quanto segue assumeremo sempre che i vettori di bit siano memorizzati in un vettore di parole a 64 bit, e che l'elemento p del vettore sia memorizzato nel bit di indice p mod 64 della parola di indice $\lfloor p/64 \rfloor$.

I processori moderni hanno un'istruzione (di solito chiamata `POPCOUNT`) che conta il numero di uni in una parola, di 64 bit, in genere in un ciclo. Se quindi $\boldsymbol{b}$ è un vettore di 64 bit contenuto in un registro, per calcolare $\text{rank}_{\boldsymbol{b}}(p)$, $0 \leq p < 64$, è sufficiente applicare `POPCOUNT` all'espressione $\boldsymbol{b}\ \&\ (1 \ll p) - 1$. In sostanza, in tempo costante è possibile effetture il calcolo del rango su una parola macchina senza l'utilizzo di tabelle.

RANK9 [**?**] è una struttura di rango per vettori di al più 2^{64} bit che utilizza il 25% di spazio in più del vettore originale e permette di effettuare il calcolo di rango producendo al più due fallimenti della cache.

L'idea è di avere dei blocchi fissi da 64 bit (visto che possiamo effettuare rango al loro interno in tempo costante) e dei superblocchi fissi da $64 \cdot 8 = 512$ bit. A ogni superblocco associamo due parole da 64 bit: una contiene il rango all'inizio del superblocco, l'altra contiene, concatenati, i delta di rango per $0 \leq k \leq 7$: più precisamente, in posizione $63 - 9k$ scriviamo il delta di rango di indice k. Si noti che in realtà non abbiamo spazio per il primo delta di rango, ma dato che è sempre zero, l'espressione

$$(w \gg (63 - 9k))\ \&\ \text{0x7FF}$$

restituisce sempre il delta di rango corretto.

Per calcolare in generale $\text{rank}_{\boldsymbol{b}}(p)$ occorre quindi recuperare il rango all'inizio del superblocco e sommargli il delta di rango del blocco (e queste due operazioni producono al più un fallimento della cache, dato che le due parole sono consecutive), e infine sommargli il rango dall'interno del blocco, il che richiede per quanto detto tempo costante (e, potenzialmente, un altro fallimento della cache). L'intero processo non coinvolge nessun ciclo, nessun test, e nessun accesso addizionale alla memoria (per esempio, a tabelle precomputate).

Lo spazio occupato è esattamente di 128 bit ogni 512 bit—il 25% del vettore di bit originale.

9.2 Strutture per la selezione

La classica struttura per la selezione in tempo costante è quella proposta da Clarke [**?**], e si basa su una serie di *inventari* delle posizioni degli uni. Rispetto a quella per il rango, utilizza *tre* livelli: a ogni livello, decidiamo se il blocco in esame è così denso che conviene memorizzare la posizione di una frazione degli uni all'interno del blocco, o se è così sparso che possiamo scrivere la posizione di tutti gli uni all'interno del blocco. La dimensione dei blocchi è scelta in modo da rendere la quantità complessiva di bit aggiuntivi $o(n)$.

Inizialmente, memorizziamo la posizione degli uni multipli di $\log n \log\log n$. Questo richiede[6]

$$\frac{n}{\log n \log\log n} \log n = \frac{n}{\log\log n} = o(n)$$

bit.

Sia ora r la distanza tra due posizioni consecutive memorizzate nella tabella di primo livello. Se $r \geq (\log n \log\log n)^2$, scriviamo esplicitamente nel secondo livello la posizione di tutti gli uni, cosa che richiederà

$$(\log n \log\log n) \log r \leq (\log n \log\log n) \log n \leq \frac{r}{\log\log n}$$

bit di spazio. Altrimenti, memorizziamo le posizioni degli uni multipli di $\log r \log\log n$, il che richiede

$$\frac{\log n \log\log n}{\log r \log\log n} \log r \leq \frac{r}{\log r \log\log n} \log r = \frac{r}{\log\log n}$$

bit. Se sommiamo su tutti i blocchi, tenuto conto che la somma delle distanze tra posizioni consecutive è al più n abbiamo che lo spazio richiesto per il secondo livello è al più $n/\log\log n = o(n)$.

Sia ora r' la distanza tra due posizioni consecutive memorizzate nella tabella di secondo livello. Se $r' \geq \log r' \log r (\log\log n)^2$, memorizziamo le posizioni di tutti i bit a uno, il che richiede

$$(\log r \log\log n) \log r' \leq \frac{r'}{\log\log n}$$

bit. Si noti che se anche tutti i blocchi di secondo livello fossero memorizzati in questo modo, utilizzerebbero comunque $O(n/\log\log n) = o(n)$ bit.

Rimane il caso in cui $r' < \log r' \log r (\log\log n)^2$. Si noti che (dato che siamo arrivati al terzo livello) $\log r \leq \log\big((\log n \log\log n)^2\big) \leq 4 \log\log n$ e, banalmente, $\log r' \leq \log r$. Quindi

[6] In generale, le formule successive sono scritte facendo precedere il numero di uni da scrivere seguito dal numero di bit necessari per ciascun uno.

$r' \leq 16(\log\log n)^4$, e possiamo memorizzare in una tabella le risposte a tutte le sequenze di questa dimensione utilizzando

$$2^{16(\log\log n)^4} \cdot 16(\log\log n)^4 \cdot \log\bigl(16(\log\log n)^4\bigr) = o(n)$$

bit. Si noti che in questo caso la dimensione delle tabelle, anche per $n = 10^6$, è stratosferica:

$$\approx 2^{16\cdot 5^4} \cdot 16 \cdot 5^4 \cdot \log\bigl(16 \cdot 5^4\bigr) \approx 10^{150\,000}$$

Implementazioni pratiche. Il primo problema da affrontare è che non esiste, al momento, un'istruzione confrontabile alla `POPCOUNT` per la selezione. È però possibile effettuare la selezione in un numero costante di operazioni utilizzando l'algoritmo *broadword* [**?**] in figura 2.

L'idea di è di utilizzare l'algoritmo classico per il calcolo del numero di bit a uno in una parola (fase 1). L'algoritmo utilizza un primo insieme di operazioni logiche per far sì che in ogni coppia di bit si trovi il numero di bit a uno nelle stesse posizioni nella parola originale. A questo punto tramite traslazioni e maschere si sommano le posizioni consecutive, ottenendo così il numero di bit a uno in blocchi di quattro e infine di otto bit. Infine, si effettua una moltiplicazione per la maschera `ONES_STEP_8`, che ha un uno nel bit più basso di ciascun byte, ottenendo la distribuzione cumulativa degli uni (non è possibile che ci siano riporti da un byte a un altro).

Una volta ottenuta la distribuzione cumulativa, effettuiamo una sottrazione parallela a byte, e utilizziamo di nuovo una moltiplicazione per `ONES_STEP_8`, ottenendo il numero di byte in cui la distribuzione cumulativa è inferiore dal rango k desiderato: in pratica, l'indice del byte contenente il bit che ci interessa. Per completare il calcolo del risultato, utilizziamo una piccola tabella che memorizza per ogni byte e ogni $0 \leq j < 8$ la posizione del bit a uno di rango j.[7]

Si noti che, in generale, è possibile effettuare selezione del k-esimo bit rapidamente in un blocco di memoria di lunghezza arbitraria utilizzando `POPCOUNT`: è sufficiente, per ogni blocco di 64 bit, calcolare il numero c di uni: se è superiore a k, il bit desiderato si trova nella parola corrente, ed è possibile utilizzare l'algoritmo *broadword*. Altrimenti, si riparte dalla parola successiva cercando il bit di rango $k - c$.

Sfruttando quanto detto sinora, una struttura pratica per la selezione su vettori di al più 2^{64} bit è ottenibile utilizzando un semplice sistema di inventario a due livelli. Supponiamo che il vettore $\boldsymbol{b}$ di n bit contenga m uni. Fissiamo una costante L (diciamo, $L = 8192$) e memorizziamo esplicitamente la posizione degli uni di rango multiplo di $\lceil Lm/n \rceil$ (cioè uno ogni $\lceil Lm/n \rceil$). Per ogni blocco di $\lceil Lm/n \rceil$ uni, allochiamo un budget di M parole di 64 bit (diciamo, otto) per un inventario di secondo livello. Se l'ampiezza di un blocco è inferiore a 2^{16}, memorizziamo le posizioni degli uni di rango multiplo di $\lceil Lm/n \rceil / 4M$, altrimenti memorizziamo le posizioni

[7] È possibile completare il calcolo senza utilizzare tabelle calcolando la distribuzione cumulativa degli uni nel byte identificato nella prima fase [**?**], ma data la dimensione e la velocità delle cache recenti la procedura non è competitiva con quella descritta in figura 2.

```
#include <inttypes.h>

#define ONES_STEP_4 ( 0x1111111111111111ULL )
#define ONES_STEP_8 ( 0x0101010101010101ULL )

// For each byte, 0 if y is smaller than x, 1 otherwise
#define LEQ_STEP_8(x,y) ( ( ( ( ( (y) | MSBS_STEP_8 ) - ( x ) ) ) & MSBS_STEP_8 ) >> 7 )

int select_in_word( const uint64_t x, const int k ) {  /* k < number of ones in x. */

   // Phase 1: sums by byte
   uint64_t byte_sums = x - ( x >> 1 & 0x5ULL * ONES_STEP_4 );
   byte_sums = ( byte_sums & 3 * ONES_STEP_4 ) + ( ( byte_sums >> 2 ) & 3 * ONES_STEP_4 );
   byte_sums = ( byte_sums + ( byte_sums >> 4 ) ) & 0x0f * ONES_STEP_8;
   byte_sums *= ONES_STEP_8;

   // Phase 2: compare each byte sum with k
   const uint64_t k_step_8 = k * ONES_STEP_8;
   const int place = ( LEQ_STEP_8( byte_sums, k_step_8 ) * ONES_STEP_8 >> 53 ) & ~0x7;

   // Phase 3: Locate the relevant byte and look up the result in select_in_byte
   return place + select_in_byte[ x >> place & 0xFF
       | k - ( ( byte_sums << 8 ) >> place & 0xFF ) << 8 ];
}
```

Figura 2: L'algoritmo *broadword* per la selezione in una parola di 64 bit.

degli uni di rango multiplo di $\lceil Lm/n \rceil / M$. Per ragioni pratiche, tutti i valori (incluso $\lceil Lm/n \rceil$) vengono approssimati dal basso con una potenza di due, in modo da evitare moltiplicazioni e divisioni costose.

L'operazione di selezione del bit di rango k avviene cercando innanzitutto nell'inventario di primo livello il bit di rango $k\lfloor k/\lceil Lm/n \rceil \rfloor$; poi, all'interno del blocco successivo utilizziamo l'inventario di secondo livello per trovare il bit più vicino a quello desiderato. Da lì in poi, proseguiamo con una ricerca sequenziale.

A seconda della scelta di L e M si possono scegliere compromessi diversi tra tempo e spazio. Si noti in particolare che se $L \leq 8192 = 2^{13}$ in un vettore distribuito uniformemente l'ampiezza di un blocco dell'inventario di primo livello è

$$\frac{n}{m}\lceil Lm/n \rceil \leq L + \frac{n}{m}$$

Quindi o $n/m \leq L$, nel qual caso possiamo scrivere gli inventari di secondo livello usando 16 bit per elemento, o $n/m > L$, nel qual caso l'inventario di primo livello contiene già tutti gli uni del vettore. Inoltre, l'inventario di primo livello contiene $m/\lceil Lm/n \rceil \leq n/L$ elementi, e quindi utilizza solo $64n/L$ bit (nel caso $L = 8192$, meno dell'un per cento del vettore originale).

L'inventario di secondo livello contiene $Mm/\lceil Lm/n \rceil \leq Mn/L$ elementi, e quindi per $M = 8$ complessivamente utilizzeremo meno del dieci per cento del vettore originale.

Si noti che, in alternativa, è possibile distribuire l'inventario di secondo livello in maniera proporzionale alla dimensione dei blocchi dell'inventario di primo livello (come nella struttura $o(n)$), ma questo richiede un puntatore addizionale per ogni elemento nell'inventario di primo livello.

10 Alberi binari

Andiamo ora a descrivere una prima rappresentazione di base degli alberi binari a *livelli*. Ricordiamo che un albero binario è per definizione l'albero binario vuoto $\varnothing$ o una coppia $\langle S, D \rangle$ di alberi binari, nel qual caso S è detto sottoalbero *sinistro* e D sottoalbero *destro*.

A un albero binario è associata una rappresentazione in forma di albero (cioè grafo aciclico connesso) radicato e ordinato (cioè con un ordine inteso tra i figli) in cui tutti i nodi, divisi in *interni* ed *esterni*, hanno zero o due figli: l'albero vuoto è rappresentato da un singolo nodo *esterno*, mentre un albero $\langle S, D \rangle$ è rappresentato da un nodo interno, la radice, collegato alla radice della rappresentazione di S e alla radice della rappresentazione di D, in quest'ordine.

Tramite la rappresentazione suddetta possiamo descrivere un albero binario tramite un vettore $\boldsymbol{b}$ di bit per *livelli*: si parte dalla radice, si visita l'albero per livelli e si scrive un uno per ogni nodo interno e uno zero per ogni nodo esterno. In tutto scriviamo $2n + 1$ bit. Il numero di alberi

binari con n nodi interni (e quindi $2n+1$ nodi complessivi) è il numero di Catalan

$$C_n = \frac{1}{n+1}\binom{2n}{n},$$

definito da $C_0 = 1$ e

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i},$$

relazione da cui si ricava immediatamente il rapporto con il conteggio degli alberi binari. Si noti che utilizzando l'approssimazione di Stirling

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{1}{n+1}\frac{(2n)!}{(n!)^2} \sim \frac{1}{n+1}\frac{\left(\frac{2n}{e}\right)^{2n}\sqrt{2\pi 2n}}{\left(\frac{n}{e}\right)^{2n}2\pi n} \sim \frac{4^n}{\sqrt{\pi n^3}},$$

il che implica $\log C_n = 2n - O(\log n)$. Quindi, se con $o(n)$ bit addizionali siamo in grado di calcolare il genitore e il figlio di ogni nodo avremo ottenuto una struttura dati succinta.

Dato un nodo in posizione p, notiamo innanzitutto che possiamo capire se è un nodo interno o esterno guardando se b_p è uno o zero. Se il nodo è interno, il suo primo figlio deve comparire dopo tutti i nodi interni che lo precedono, che sono esattamente $\mathrm{rank}_{\boldsymbol{b}}(p)$, e dopo tutti i loro figli, che sono $\mathrm{rank}_{\boldsymbol{b}}(p)+1$: il figlio sinistro ha dunque indice $2\,\mathrm{rank}_{\boldsymbol{b}}(p)+1$ (e quello destro è una posizione dopo).

D'altra parte, se vogliamo calcolare il genitore di un nodo in posizione p, questo è il nodo interno di rango $\lfloor (p-1)/2 \rfloor$. Infatti, supponiamo che il nodo in posizione p sia un figlio sinistro. Il padre è il nodo in posizione q tale che $2\,\mathrm{rank}_{\boldsymbol{b}}(q)+1 = p$, cioè $\mathrm{rank}_{\boldsymbol{b}}(q) = (p-1)/2$. Ma dato che $b_q = 1$, $q = \mathrm{select}_{\boldsymbol{b}}(\mathrm{rank}_{\boldsymbol{b}}(q)) = \mathrm{select}_{\boldsymbol{b}}((p-1)/2)$. Un calcolo analogo per il figlio destro mostra che, alla fine, il padre del nodo in posizione p ha posizione $\mathrm{select}_{\boldsymbol{b}}(\lfloor (p-1)/2 \rfloor)$.

Si noti che se è necessario associare dati ancillari a ogni nodo (interno ed esterno), questo è possibile utilizzando semplicemente un vettore di lunghezza $2n+1$. Se invece si desidera associare dati a nodi interni, basta un vettore di lunghezza n che verrà indicizzato da $\mathrm{rank}_{\boldsymbol{b}}(p)$, dove p è l'indice di un nodo interno.

11 Successioni monotone

Descriviamo ora una semplice ma efficace rappresentazione delle sequenze monotone, la rappresentazione di Elias–Fano [**?**, **?**]. È di fatto la prima struttura (quasi) succinta apparsa in letteratura, e le idee alla sua base ricorrono nelle strutture succinte più moderne.

Abbiamo una sequenza monotona di numeri non negativi

$$0 \le x_0 \le x_1 \le \cdots \le x_{n-2} \le x_{n-1} < u,$$

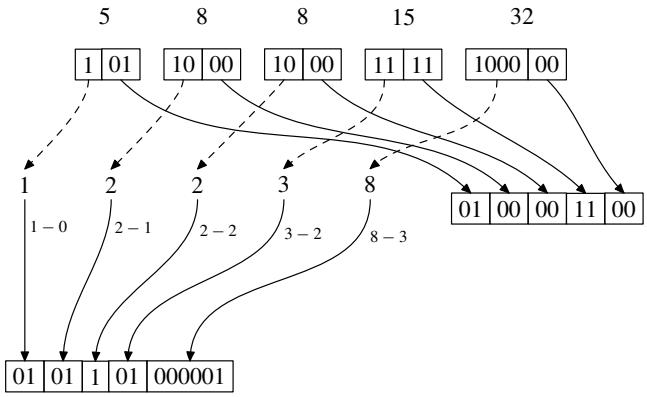


Figura 3: Un esempio della codifica quasi-succinta di Elias–Fano. Consideriamo la successione 5, 8, 8, 15, 32 con limite superiore 36, e quindi $\ell = \lfloor \log(36/5) \rfloor = 2$. Sulla destra, gli ℓ bit più bassi sono concatenati per formare il vettore dei bit inferiore. Sulla sinistra, gli scarti dei valori dei bit superiori sono memorizzati sequenzialmente con codifica unaria.

dove $u > 0$ è un qualunque limite superiore per l'ultimo valore.[8] La scelta $u = x_{n-1} + 1$ è naturalmente possibile (e ottima), ma in genere memorizzare esplicitamente x_{n-1} può essere costoso, e un valore ragionevole per u può essere disponibile tramite informazioni esterne.

Rappresenteremo la sequenza in due vettori di bit come segue:

- gli $\ell = \max\{0, \lfloor \log(u/n) \rfloor\}$ bit inferiori di ogni x_i sono memorizzati esplicitamente in maniera contigua nel *vettore dei bit inferiori*;
- i bit superiori sono memorizzati nel *vettore dei bit superiori* come sequenza di *scarti* codificati in unario (cioè scrivendo k come 0^k1).

Nella figura 3 viene mostrato un esempio. Notate che codifichiamo gli scarti tra i valori dei bit superiori, cioè $\left\lfloor x_i/2^\ell \right\rfloor - \left\lfloor x_{i-1}/2^\ell \right\rfloor$ (con la convenzione $x_{-1} = 0$).

La proprietà interessante di questa rappresentazione è che usa al più $2 + \lceil \log(u/n) \rceil$ bit per elemento: si può facilmente vedere dal fatto che ogni codice unario usa un bit di stop, e ogni altro bit scritto aumenta il valore dei bit superiori di 2^ℓ: chiaramente, questo non può succedere più di $\left\lfloor x_{n-1}/2^\ell \right\rfloor$ volte. Ma

$$\left\lfloor \frac{x_{n-1}}{2^\ell} \right\rfloor \leq \left\lfloor \frac{u}{2^\ell} \right\rfloor \leq \frac{u}{2^\ell} = \frac{u}{2^{\max\{0, \lfloor \log(u/n) \rfloor\}}} \leq 2n. \tag{1}$$

[8] Se $u = 0$, necessariamente $n = 0$, mentre se $u = 1$, la lista è interamente composta da zeri, e il suo contenuto è definito da n.

quindi, scriviamo al più n uni e $2n$ zeri, il che implica quanto affermato, dato che $\lceil \log(u/n) \rceil = \lfloor \log(u/n) \rfloor + 1$ a meno che u/n sia una potenza di due, ma in quel caso (1) in realtà finisce con $\leq n$, e quindi l'affermazione è ancora vera.

Si noti ora che il numero di successioni monotone di n elementi in un universo di u elementi è

$$\binom{u+n-1}{u-1} = \binom{u+n-1}{n} \tag{2}$$

Infatti, una tale successione monotona è equivalente a un multiinsieme di cardfinalità n su u elementi. Tali insiemi sono in biiezione con le soluzioni nonnegative dell'equazione

$$x_0 + x_1 + \cdots + x_{u-1} = n.$$

Le soluzioni possono essere espresse come il posizionamento di $u-1$ segni "+" in mezzo a n segni $\bullet$, che sono contati esattamente dal binomiale (2). Con l'assunzione $n \leq \sqrt{u}$ abbiamo allora

$$\begin{aligned}\left\lceil \log \binom{u+n-1}{n} \right\rceil &\approx n \log\left(\frac{u+n-1}{n}\right) = n \log\left(\frac{u}{n}\left(1 + \frac{n}{u} - \frac{1}{u}\right)\right) \\ &= n \log\left(\frac{u}{n}\right) + n \log\left(1 + \frac{n}{u} - \frac{1}{u}\right) \approx n \log\left(\frac{u}{n}\right) + \frac{n^2}{u}.\end{aligned}$$

Vediamo quindi che la rappresentazione è molto vicina a essere succinta quando il vettore è sufficientemente sparso.

Per ottenere x_i, effettuiamo una selezione dell'i-esimo bit sul vettore dei bit superiori, ottenendo la posizione p: il valore dei bit superiori di x_i è allora esattamente $p - i$; gli ℓ bit inferiori possono essere estratti con un accesso diretto, dal momento che sono memorizzati in posizione $i\ell$ nel vettore dei bit inferiori.

11.1 Elias–Fano come rappresentazione di vettori di bit

La struttura può anche essere vista come una struttura di selezione su vettori di bit *non sistematica*, e cioè che rimpiazza interamente il vettore di bit originale con una rappresentazione *opportunistica* che è funzionale ai vettori sparsi. In questo caso il limite teorico è ancora più basso, e quindi la struttura ancora più lontana dall'essere succinta, ma rimane estremamente utile in pratica. Per completezza, dobbiamo però discutere come ottenere una funzionalità di rango.

Il rango, nella rappresentazione di Elias–Fano, non è un'operazione costante, e corrisponde, dal punto di vista insiemistico, a contare quanti x_i sono minori (strettamente) di un dato p. Occorre costruire una struttura di selezione di *zeri* sul vettore dei bit superiori, e dato p trovare il bit di

rango $\lfloor p/2^\ell \rfloor$, perché, necessariamente, il primo x_i minore di p è associato a un uno che sta tra la posizione $\text{select}_0(\lfloor p/2^\ell \rfloor - 1)$ e la posizione $\text{select}_0(\lfloor p/2^\ell \rfloor)$.

A questo punto si scorre il vettore all'indietro, cercando gli uni (che corrispondono a elementi della lista) e estraendo il corrispondente valore finché non è minore o uguale p. L'indice dell'uno raggiunto è il rango di p. È possibile, in teoria, che la ricerca si protragga per $2^\ell = O(u/n)$ passi. Per esempio, se i primi valori della lista sono 0, 1,... 2^ℓ, calcolando rank(1) ci posizioneremo sullo zero associato al valore 2^ℓ, e dovremo scandire $2^\ell - 1$ elementi per arrivare a zero.

Si noti che a densità elevate (cioè quando $n \gtrsim u/3$) diventa più conveniente (dal punto di vista dello spazio) utilizzare un vettore caratteristico di bit con una struttura di rango/selezione.

11.2 Elias–Fano con inventari lineari

Un modo meno "succinto" ma estremamente utile in pratica di implementare Elias–Fano consiste nel memorizzare, dato un quanto q, un puntatore agli uni e agli zeri di indice qk, $k \geq 0$. In questo modo è possibile fare selezione semplicemente trovando l'uno (o lo zero) che precede immediatamente quello desiderato ed effettuando una scansione lineare. In pratica, stiamo utilizzando solo un inventario di primo livello, ma le performance sono eccellenti perché sappiamo in anticipo che il vettore dei bit superiori è approssimativamente formato da metà zeri e metà uni.

12 Parentesi bilanciate

La seconda tipologia di primitiva di base più tipica nella costruzione di strutture succinte è relativa a sequenze di *parentesi bilanciate*. Tali sequenze sono formate da parentesi aperte e chiuse correttamente accoppiate. Quindi "(()(()))" è una sequenza bilanciata, mentre "(()()" non lo è. Una sequenza di parentesi può essere rappresentata come un vettore di n bit $\boldsymbol{b}$ scrivendo un uno per ogni parentesi aperta e uno zero per ogni parentesi chiusa.

Definiamo la *funzione di eccesso (aperta)*

$$E_{\boldsymbol{b}}(i) = |\{b_j \mid j < i \wedge b_j = 1\}| - |\{b_j \mid j < i \wedge b_j = 0\}|,$$

che rappresenta l'eccesso di parantesi aperte rispetto a quelle chiuse in posizione i (esclusa). Il vettore $\boldsymbol{b}$ è *debolmente bilanciato* se la funzione di eccesso è sempre nonnegativa, e vale 0 in 0 e in n. È *fortemente bilanciato* se la funzione di eccesso è sempre strettamente positiva, tranne in 0 e in n, in cui vale 0.

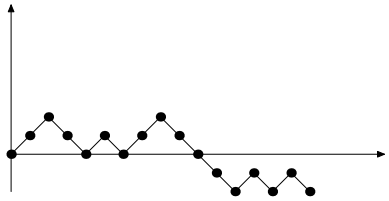
Per esempio, consideriamo i primi due byte di un vettore di bit:

$$\boxed{1\,1\,0\,0\,1\,0\,1\,1}\quad\boxed{0\,0\,0\,0\,1\,0\,1\,0}\ldots$$

Notate che il bit più a sinistra è il bit zero. Stiamo rappresentando la sequenza di parentesi

$$\boxed{(\,(\,)\,)\,(\,)\,(\,(} \quad \boxed{)\,)\,)\,)\,(\,)\,(\,)} \ldots$$

La funzione di eccesso si comporta come segue:



Le primitive che andremo a studiare sono:

- findopen(p), che trova la parentesi aperta corrispondente a quella chiusa in posizione p;
- findclose(p), che trova la parentesi chiusa corrispondente a quella aperta in posizione p;
- enclose(p), che trova la parentesi aperta più vicina q (posto che esista) tale che p sta tra la parentesi aperta in q e la sua parentesi chiusa corrispondente.

Notate che le prime due primitive si possono calcolare su qualunque vettore *debolmente* bilanciato. La terza si può calcolare solo sulle parentesi di eccesso aperto strettamente positivo: in un vettore *fortemente* bilanciato, queste sono tutte le parentesi aperte tranne la prima; eventualmente, per utilizzare enclose con vettori debolmente bilanciati si possono aggiungere due parentesi esterne fittizie, in modo che tutte le parentesi del vettore originale abbiano eccesso aperto strettamente positivo.

Per calcolare queste primitive in tempo costante dividiamo innanzitutto il vettore in blocchi: per ogni blocco, chiamiamo una parentesi *lontana* se la sua corrispondente è al di fuori del blocco, *vicina* altrimenti. Una parentesi lontana aperta (chiusa) p è un *pioniere* se la sua parentesi corrispondente giace in un blocco diverso da quello della parentesi corrispondente a quella lontana aperta (chiusa) immediatamente precedente (successiva) a p.

Per ogni pioniere, terremo traccia della parentesi corrispondente. Inoltre manterremo utilizzando Elias–Fano la lista delle posizioni dei pionieri (che è molto sparsa, come vedremo tra poco). Infine, per ogni blocco terremo traccia del valore della funzione di eccesso aperto all'inizio del blocco (questo si può fare con un vettore esplicito, o utilizzando una struttura di rango sul vettore di bit che rappresenta le parentesi).

L'osservazione fondamentale (fatta originariamente da Jacobson) è che se ci sono k blocchi, ci sono al più $2k-3$ pionieri. Questo deriva dal fatto che, se consideriamo il grafo $G = \langle V, E\rangle$ in cui V contiene tutti i bocchi e $(i, j) \in E$ se il blocco i contiene un pioniere e il blocco j la sua parentesi corrispondente, G è *planare esterno*, e tali grafi hanno al più $2k-3$ lati.

È anche possibile dare una dimostrazione diretta per induzione: consideriamo la lista dei k blocchi come punti su una linea nel piano e i lati che li connettono. Se rappresentiamo i lati come archi nella metà superiore del piano, questi non si possono intersecare. È anche evidente che per $k = 2$ la formula è vera (ci può essere un solo arco). Per $k > 2$, dopo aver tolto al più un arco dal primo all'ultimo elemento, ci deve essere un elemento pivot $p > 0$ sopra cui non passa nessun arco. Possiamo assumere induttivamente che la formula sia vera per i primi p e gli ultimi $k - p + 1$ elementi, ottenendo che il numero di lati è al più

$$2p - 3 + 2(k - p + 1) - 3 + 1 = 2k - 3.$$

Possiamo ora implementare findclose(p) come segue: prima di tutto, controlliamo se la parentesi in posizione p è vicina, cosa che possiamo fare con una scansione del blocco, nel qual caso restituiamo la posizione della parentesi corrispondente. Altrimenti, troviamo il pioniere associato (con un'operazione di predecessore che possiamo implementare tramite rango e selezione), e se il pioniere è la parentesi stessa restituiamo la parentesi corrispondente. Altrimenti, contiamo le parentesi aperte lontane tra il pioniere e p: è facile perché basta misurare la differenza in eccesso aperto al pioniere e in p, cose che possiamo fare utilizzando la tabella che ci dà l'eccesso aperto all'inizio dei due blocchi (quello che contiene il pioniere e quello che contiene p) e una scansione dei blocchi stessi. A ogni parentesi aperta deve corrispondere una parentesi chiusa lontana nel blocco di arrivo, e sappiamo per definizione che tutte le parentesi chiuse che dovremo contare saranno nello stesso blocco (per definizione di pioniere). Possiamo quindi concludere il calcolo con la scansione di un blocco.

Per implementare enclose(p), teniamo conto in una tabella, per ogni blocco, della prima parentesi aperta avente eccesso aperto di uno inferiore al minimo nel blocco. Data una parentesi aperta in posizione p, con parentesi chiusa associata in posizione q, la parentesi aperta che corrisponde alla coppia che racchiude la coppia $\langle p, q\rangle$ è la prima che precede p con eccesso inferiore di uno a quello di p. Se questa parentesi si trova nel blocco, la possiamo trovare con una scansione. Altrimenti, andiamo a vedere se p è lontana o vicina.

Nel primo caso, la funzione di eccesso aperto tra p e q non è mai inferiore al valore che ha in p, e quindi il valore in p è il minimo del blocco a cui p appartiene: possiamo trovare la (coppia di) parentesi che racchiude $\langle p, q\rangle$ utilizzando la tabella.

Nel secondo caso, scandiamo le parentesi che seguono q nel blocco. Se troviamo una parentesi chiusa di eccesso aperto uguale a p, la corrispondente aperta è quella che cerchiamo, e la possiamo recuperare con una findopen. Altrimenti, tutte le parentesi aperte rimanenti nel blocco hanno eccesso aperto maggiore o uguale a p, e ricadiamo nel caso precedente.

12.1 L'isomorfismo LC-RS

L'isomorfismo *left child-right sibling* (LC-RS) è un'isomorfismo tra le foreste ordinate (cioè insiemi ordinati di alberi ordinati) e gli alberi binari. Per ottenere un isomorfismo tra alberi e al-

beri binari è sufficiente considerare la foresta che si ottiene eliminando la radice dell'albero (che può essere la foresta vuota). È uno strumento fondamentale nella rappresentazione dati perché permette di rappresentare foreste ordinate e alberi binari interscambiabilmente, e di scegliere quindi di volta in volta la rappresentazione più comoda.

Il modo più semplice per definire l'isomorfismo LC-RS è quello di notare che le foreste si possono costruire in modo induttivo come segue:

- la foresta vuota è una foresta;
- date due foreste F e G, possiamo costruire una nuova foresta come segue: colleghiamo un nuovo nodo alle radici degli alberi di F, ottenendo così un albero, e concateniamo a questo albero la foresta G.

A questo punto possiamo specificare l'isomorfismo come segue:

- all'albero binario vuoto corrisponde la foresta vuota;
- dato un albero binario $\langle S, D \rangle$, la foresta corrispondente si ottiene applicando la seconda regola alla foresta associata a S e a quella associata a D.

È anche facile specificare l'inverso:

- alla foresta vuota corrisponde l'albero binario vuoto;
- data una foresta generata da F e G usando la seconda regola, se S è l'albero binario associato a F e D l'albero binario associato a G associamo alla foresta l'albero binario $\langle S, D \rangle$.

Il nome della rappresentazione deriva dal fatto che quando si scende a sinistra nell'albero binario si accede a un figlio (il figlio del primo albero della foresta), mentre quando si scende a destra si accede al primo fratello.

Si può esprimere in modo grafico l'isomorfismo inverso come segue: colleghiamo, nella foresta, i figli dello stesso nodo in orizzontale, e ogni genitore con il proprio primo figlio. È facile vedere che l'albero risultante è binario (bisogna ovviamente aggiungere nodi esterni ove mancanti). Questo è un esempio in cui i nodi esterni sono stati omessi:

La tabella 1 contiene un'enumerazione dei casi con al più tre nodi.

Albero binario	Parentesi (deb.)	Parentesi (fort.)	Foresta ordinata	Albero ordinato radicato
	ε	( )	ε	
	( )	(( ))		
	( ) ( )	(( ) ( ))		
	(( ))	((( )))		
	( ) ( ) ( )	(( ) ( ) ( ))		
	(( ) ( ))	((( ) ( )))		
	(( )) ( )	((( )) ( ))		
	( ) (( ))	(( ) (( )))		
	((( )))	(((( ))))		

Tabella 1: La corrispondenza tra espressioni (debolmente e fortemente) ben parentesizzate, alberi binari, foreste ordinate e alberi radicati ordinati fino a sei parentesi debolmente bilanciate, cioè fino a tre nodi (interni, nel caso degli alberi binari, o non-radice, nel caso degli alberi radicati). Per semplicità, i nodi esterni degli alberi binari non sono stati disegnati.

12.2 Rappresentare foreste e alberi binari tramite parentesi debolmente bilanciate

Una foresta può essere rappresentata facilmente scrivendo una serie di parentesi debolmente bilanciate durante una visita in profondità effettuata partendo dalla radice di ogni albero della foresta, nell'ordine naturale: ogni volta che entriamo in un albero scriviamo una parentesi aperta, e ogni volta che usciamo una parentesi chiusa. Questo è in effetti un isomorfismo tra sequenze di parentesi debolmente bilanciate e foreste. Volendo specificarlo induttivamente:

- alla foresta vuota corrisponde l'espressione ben parentesizzata vuota;
- data una foresta generata da F e G usando la seconda regola, se f è l'espressione associata a F e g quella associata a G associamo alla foresta l'espressione $(f)g$.

Si noti che, concettualmente, questa rappresentazione può anche essere pensata come una rappresentazione di alberi radicati ordinati, dato che esiste una biiezione tra foreste e alberi radicati ordinati (basta aggiungere o togliere la radice, e collegarla o scollegarla dagli alberi figli).

Vediamo ora come è possibile implementare in tempo costante le operazioni di una foresta. A ogni parentesi aperta facciamo corrispondere un nodo. Se la parentesi immediatamente successiva è chiusa, il nodo non ha figli. Altrimenti, il primo figlio è rappresentato dalla parentesi aperta immediatamente successiva; per individuare i figli successivi è sufficiente utilizzare findclose per trovare la parentesi chiusa corrispondente e controllare quella successiva. Il genitore di un nodo si ottiene tramite enclose.

Si noti che data una coppia di parentesi $\langle p, q\rangle$, che individua un dato nodo, il numero di foglie sotto il nodo è dato dal numero di sottosequenze della forma (), che è facilmente calcolabile tramite una modifica degli algoritmi per il rango, e il numero di discendenti propri semplicemente da $\lfloor (q - p)/2 \rfloor$. Questo è un esempio non banale di primitiva calcolabile facilmente e senza memorizzare elementi aggiuntivi su una struttura succinta, ma non su una struttura non succinta.

Ma passiamo ora alla rappresentazione degli alberi binari. Per rappresentare un albero binario lo trasformiamo nella foresta corrispondente e utilizziamo una rappresentazione a parentesi bilanciate. Volendo esplicitare induttivamente:

- all'albero binario vuoto corrisponde l'espressione ben parentesizzata vuota;
- dato un albero $\langle S, D\rangle$, se s è l'espressione associata a S e d quella associata a D associamo all'albero binario l'espressione $(s)d$.

Dobbiamo ora implementare le operazioni di accesso al sottoalbero sinistro, al sottoalbero destro e al genitore osservando la loro implementazione sulla rappresentazione a parentesi bilanciate della foresta.

Ora, l'isomorfismo mappa il primo figlio nel sottoalbero sinistro. Quindi, per conoscere il sottoalbero sinistro è sufficiente controllare la parentesi successiva a quella corrente: se è chiusa, l'albero binario corrente è vuoto, e non ci sono figli. Altrimenti, è la parentesi aperta associata al figlio sinistro. Per conoscere il sottoalbero destro dobbiamo prendere in considerazione il primo fratello del nodo corrente: occorre quindi prendere la parentesi chiusa associata e considerare quella aperta immediatamente successiva (se non c'è, il sotto albero destro è vuoto).

Notiamo ora che una parentesi aperta è preceduta da una chiusa se e solo se rappresenta un figlio successivo al primo. Quindi, i sottoalberi destri sono rappresentati da parentesi aperte precedute da chiuse, e i sottoalberi sinistri da parentesi aperte precedute da aperte. Nel primo caso il genitore è fornito da findopen, nel secondo è banale (è la parentesi aperta precedente).

Infine, notiamo come sia possibile, di nuovo, calcolare in tempo costante il numero di nodi di un sottoalbero la cui aperta parentesi è in posizione p. Se si tratta di un sottoalbero sinistro, basta trovare la parentesi chiusa associata, diciamo in posizione q: il numero di nodi è $\lfloor (q-p)/2 \rfloor$. Se si tratta di un sottoalbero destro, i nodi che vogliamo contare sono quelli che nell'albero ordinato sono fratelli destri del nodo associato a quello in posizione p. Potremmo elencare i fratelli destri e contare di volta in volta la dimensione dei sottoalberi, ma questo processo non avverrebbe in tempo costante. Possiamo invece avvalerci del fatto che enclose(p) ci fornisce una coppia di parentesi (quella del genitore) la cui parentesi chiusa, diciamo in posizione q, segue immediatamente la parentesi chiusa dell'ultimo fratello. Il numero di nodi nel sottoalbero è quindi di nuovo $\lfloor (q-p)/2 \rfloor$.

13 Funzioni succinte e *hash* minimali perfetti

Un problema interessante da risolvere in maniera succinta è la memorizzazione di una funzione $f : X \to 2^r$, dove X è un sottoinsieme di n elementi da un universo U di cardinalità u. Il numero di bit necessari a memorizzare una tale funzione è ovviamente almeno rn, posto che non specifichiamo il comportamento della funzione su un elemento fuori da X (altrimenti interverrebbe il limite per la memorizzazione di X come sottoinsieme di U).

Vediamo come risolvere questo problema utilizzando una tecnica basata sui grafi casuali: utilizza quindi la randomizzazione per la costruzione della struttura dati. La risposta della struttura dati è però deterministica.

Prendiamo due funzioni di *hash*[9] $h, g : U \to m$ con uno spazio dei valori m un po' più ampio della cardinalità n di X, e consideriamo un vettore $\boldsymbol{w}$ di variabili di dimensione m con valori in 2^r. Scriviamo ora il sistema di n equazioni

$$w_{h(x)} + w_{g(x)} = f(x) \mod 2^r.$$

[9]Come spesso avviene, al fine dell'analisi le due funzioni sono assunte essere scelte a caso e uniformemente tra tutte le funzioni possibili da U a m: in pratica, si usano funzioni "ben fatte" come MurmurHash3, CityHash o la funzione di Jenkins.

Si noti che avendo due funzioni, $m > n$ e il vettore $\boldsymbol{w}$ abbiamo — come dire — più spazio per giocare.

Ora, è chiaro che abbiamo a che fare con un sistema di n equazioni modulari nelle variabili w_i, che potrebbe non avere soluzione. Possiamo però rappresentare i vincoli imposti dal sistema come segue: costruiamo un grafo G non orientato con m vertici e n lati in cui per ogni $x \in X$ abbiamo che $h(x)$ è adiacente a $g(x)$ tramite un lato etichettato da $f(x)$.

Consideriamo ora una sequenza di *pelature* del grafo G. La prima pelatura, $F_0 \subseteq m$, è data dall'insieme dei vertici che sono una foglia (cioè hanno grado 0 o 1) nel grafo $G_0 = G$. La seconda, F_1, dall'insieme dei vertici che sono una foglia nel grafo G_1 ottenuto cancellando i vertici in F_0 (e i lati loro adiacenti) dal grafo. Continuiamo così finché non arriviamo a una pelatura F_k in cui non ci sono più foglie: G_k è l'insieme vuoto se e solo se il grafo è aciclico.

Se G_k è vuoto, possiamo risolvere ora il sistema nel seguente modo: partiamo da F_{k-1}, e per tutti i vertici x che sono di grado 0 in G_{k-1}, assegnamo $w_x = 0$. I vertici x di grado 1 sono invece adiacenti esattamente a un altro vertice y. In genere, w_y è già stato assegnato, dal momento che y fa parte di una pelatura successiva: fanno eccezione i vertici agli estremi di un lato isolato, che possono essere entrambi non assegnati, nel qual caso poniamo $w_y = 0$. Se v è il valore assegnato al lato che collega x e y, poniamo allora

$$w_x = v - w_y \mod 2^r.$$

Possiamo sempre effettuare l'assegnamento perché gli F_i sono un partizione dei vertici del grafo, e quindi non incontreremo mai un vertice già assegnato.

Ora, assumendo che h e g siano casuali e indipendenti, il grafo che andiamo a costruire è un grafo casuale di m vertici con n archi, e un risultato importante di teoria dei grafi casuali dice che per n sufficientemente grande quando $m > 2{,}09\,n$ il grafo è *quasi sempre* privo di cicli (quasi sempre significa che il rapporto tra il numero di grafi privi di cicli e il numero totale di grafi tende a uno). In sostanza, scegliendo bene h e g, e posto che il grafo non risulti degenere (cioè con cappi o archi paralleli) quasi tutti i grafi che otterremo permetteranno di risolvere il sistema.

Il caso ora descritto non è in realtà ottimo. La teoria degli ipergrafi casuali ci dice che utilizzando 3-ipergrafi[10] e generlizzando in maniera ovvia il processo di pelatura (possiamo pelare i vertici su cui insiste al più un iperlato) è possibile dare una nozione di aciclicità che permette di risolvere i sistemi nel caso di *tre* funzioni di *hash*, ma in questo caso il limite che garantisce l'aciclicità è $m > 1{,}23\,n$ — un miglioramento netto rispetto al caso di ordine 2.

Alla fine, per memorizzare la funzione dovremo utilizzare solo $1{,}23\,rn$ bit più lo spazio necessario a descrivere le funzioni h e g, che in pratica è di poche decine di bit.

[10] Un *k-ipergrafo* è dato da un insieme di sottoinsiemi di ordine k dell'insieme dei vertici, detti *iperlati*; un grafo standard è un 2-ipergrafo.

Si noti, inoltre, che dato che il vettore che contiene le soluzioni del sistema contiene una significativa quantità di zeri (almeno $0{,}23/1{,}23\% \approx 18.7\%$) può essere conveniente *compattare* il vettore di soluzioni del sistema utilizzando il sistema di marcatura delle posizioni nonzero (e una struttura di rango). In questo caso, il vettore utilizzerà solo $rn + 1{,}23n$ bit, più quanto richiesto dalla struttura di rango.

13.1 Hash minimali perfetti

Una funzione di *hash* per un insieme di chiavi $X \subseteq U$, dove U è l'universo di tutte le chiavi possibili, è una funzione $f : X \to m$. Quando la funzione è iniettiva (cioè non esistono *collisioni*) lo *hash* è detto *perfetto*. Se $|X| = m$, lo *hash* è detto *minimale*. Gli *hash* minimali perfetti sono molto utili per associare facilmente dei dati a ogni elemento di un insieme eterogeneo (per esempio, di stringhe): i dati vengono messi in un vettore, e i dati relativi a x vengono memorizzati in posizione $f(x)$.

Poniamoci innanzitutto il problema di quale sia il limite inferiore al numero di bit necessari per descrivere uno *hash* minimale perfetto. Per farlo, notiamo che una funzione di hash con n valori su U corrisponde esattamente a una *partizione* P di U con n parti non vuote: ogni parte è la fibra di un $x \in n$.[11] Una tale funzione è minimale perfetta per un insieme $X \subseteq U$ se P *separa* X, cioè se in ogni parte di P c'è esattamente un elemento di X. Diremo che un insieme di partizioni di U è un *n-sistema* se per ogni $X \subseteq U$ di cardinalità n esiste una partizione dell'insieme che separa X. Per stimare quanti bit sono necessari per scrivere una funzione di hash dobbiamo quindi studiare la minima dimensione $H_U(n)$ di un n-sistema per U.

Chiamiamo *volume* di una partizione il numero di insiemi che separa. Notate che se avessimo un limite superiore v al volume di una partizione, avremmo immediatamente che

$$H_U(n) \geq \frac{\binom{u}{n}}{v},$$

dato che dobbiamo separare $\binom{u}{n}$ insiemi, e al massimo una partizione ne può separare v. Ora, è immediato che una partizione ha volume massimo quando le sue parti sono approssimativamente uguali, e in quel caso il suo volume è

$$v \approx \left(\frac{u}{n}\right)^n.$$

Abbiamo quindi che

$$H_U(n) \geq \frac{\binom{u}{n}}{\left(\frac{u}{n}\right)^n} = \frac{\frac{u!}{n!(u-n)!}}{\frac{u^n}{n^n}} = \frac{n^n}{n!} \cdot \frac{u!}{u^n(u-n)!}.$$

[11] La *fibra* di un elemento y del codominio di una funzione è l'insieme degli elementi del dominio mappati in y.

Assumendo che n non sia troppo grande, e più precisamente che $n \leq \sqrt[2+\varepsilon]{u}$ (che non è un'assunzione troppo pesante dato che in genere $n \ll u$), si ha

$$\frac{u!}{u^n(u-n)!} = \frac{u(u-1)\cdots(u-n+1)}{u^n} \geq \left(\frac{u-n}{n}\right)^n$$
$$\geq \left(1 - \frac{\sqrt[2+\varepsilon]{u}}{u}\right)^{\sqrt[2+\varepsilon]{u}} = \left(1 - \frac{1}{u^{\frac{1+\varepsilon}{2+\varepsilon}}}\right)^{u^{\frac{1}{2+\varepsilon}}} = \left(1 - \frac{1}{u^{\frac{1+\varepsilon}{2+\varepsilon}}}\right)^{u^{\frac{1+\varepsilon}{2+\varepsilon} - \frac{1+\varepsilon}{2+\varepsilon} + \frac{1}{2+\varepsilon}}} \sim \left(e^{-1}\right)^{u^{\frac{-\varepsilon}{2+\varepsilon}}} \sim 1,$$

e quindi

$$H_U(n) \geq \frac{n^n}{n!} \cdot \frac{u!}{u^n(u-n)!} = \Omega\left(\frac{n^n}{n!}\right).$$

Prendendo i logaritmi naturali abbiamo

$$\ln H_U(n) \geq n \ln n - (n \ln n - n + O(\ln n)) = n + O(\ln n)$$

e, cambiando la base,

$$\log H_U(n) \geq n \log e + O(\log n),$$

Per memorizzare una funzione di hash minimale perfetto occorrono quindi, se n non è troppo grande, almeno $\log e + O(\log n/n) \approx 1{,}44$ bit per elemento.

Si noti che l'ipotesi su n è necessaria: per esempio, se $n = 3/4u$ possiamo creare una funzione di hash perfetto minimale utilizzando un vettore di u bit che rappresenta n e mettendoci sopra una struttura di rango. In questo caso, staremmo utilizzando $4/3 \approx 1{,}33$ per elemento.

Torniamo ora a porre l'attenzione sul processo di pelatura. Durante il processo, ogni volta che un nodo viene pelato è incidente su al più un lato. Come conseguenza, il processo di pelatura definisce un'iniezione di n in m, che associa a ogni lato il vertice la cui pelatura l'ha cancellato dal grafo. Il vertice associato viene detto il *cardine* del lato.

Consideriamo ora il sistema di equazioni

$$w_{h(x)} + w_{g(x)} = b \mod 2,$$

dove b è zero se il cardine del lato associato a x è $h(x)$, uno altrimenti, e memorizziamo la soluzione relativa. Possiamo, a partire dalla soluzione $\bar{\boldsymbol{w}}$, calcolare un'iniezione di n in m, cioè uno *hash* perfetto, ma non minimale, che mappa x nel cardine del lato associato a x. Per farlo, basta, dato x, calcolare $b = \bar{w}_{h(x)} + \bar{w}_{g(x)} \bmod 2$, e restituire $h(x)$ o $g(x)$ a seconda che b sia zero o uno. Il valore restituito è diverso per ogni x, dato che è il cardine del lato associato a x.

Per rendere questa funzione minimale basta costruire un vettore di bit parallelo a $\bar{\boldsymbol{w}}$ che memorizza le posizioni dei cardini e calcolare il rango del cardine associato a x. In questo modo siamo in grado di memorizzare una funzione di *hash* minimale perfetto utilizzando poco più di quattro bit per elemento.

Notiamo infine che, come al solito, conviene utilizzare 3-ipergrafi, e anche che c'è un trucco che permette di arrivare a circa 2,5 bit per elemento. Quando memorizziamo i valori delle variabili, possiamo utilizzare per i cardini il valore 3 invece del valore zero, dato che da un punto di vista modulare sono equivalenti. In questo modo, nel vettore delle variabili i valori corrispondenti ai cardini saranno sempre diversi da zero. È facile modificare le strutture standard di rango in maniera che calcolino il numero di coppie di bit non entrambi a zero che precedono la posizione corrente: in questo modo, non è necessario aggiungere un vettore parallelo.

13.2 Firme

Uno *hash* minimale perfetto non permette di riconoscere se un elemento fa parte di X o no. Per ovviare all'inconveniente, utilizziamo un insieme di *firme* associato all'insieme X delle chiavi. Consideriamo cioè una funzione $s : U \to 2^r$ che associ a ogni chiave possibile una sequenza di r bit "casuale", nel senso che la probabilità che $s(x) = s(y)$ se x e y sono presi uniformemente a caso da U è $1/2^r$ (per esempio, una buona funzione di *hash*). Oltre a f, memorizziamo ora una tabella di n firme a r bit S e mettiamo la firma $s(x)$ di $x \in X$ in S nella posizione $f(x)$.

Per interrogare la struttura risultante su input $x \in U$, agiamo come segue:

1. calcoliamo $f(x)$ (che è un numero in n);
2. recuperiamo $S_{f(x)}$;
3. se $S_{f(x)} = s(x)$, restituiamo $f(x)$; altrimenti, restituiamo -1 (a indicare che x non fa parte di X).

Si noti che se $x \in X$, $S_{f(x)}$ conterrà per definizione $s(x)$. Se invece $x \notin X$, $S_{f(x)}$ sarà una firma arbitraria, e quindi restituiremo quasi sempre -1, tranne nel caso ci sia una collisione di firme, il che avviene, come detto, con probabilità $1/2^r$. Tarando il numero r di bit delle firme è quindi possibile bilanciare spazio occupato e precisione della struttura.

13.3 Generalizzazioni e ottimizzazioni

È possibile migliorare ulteriormente l'occupazione in spazio quando r non è troppo piccolo utilizzando un *vettore compatto* per memorizzare i valori della soluzione del sistema. In effetti, per come abbiamo descritto il processo di soluzione non è possibile che vengano poste a valori diversi da zero più di n delle variabili. Se potessimo memorizzare, con una piccola perdita in spazio, *solo i valori diversi da zero* potremmo ridurre l'occupazione di memoria a quasi nr bit.

Per farlo, consideriamo un vettore di bit $\boldsymbol{b}$ che contiene in posizione i un uno se la variabile corrispondente w_i è diversa da zero, e zero altrimenti. Chiaramente, se mettiamo in un vettore

C i valori delle variabili w_i diversi da zero, abbiamo

$$w_i = \begin{cases} 0 & \text{se } b_i = 0; \\ C[\text{rank}_{\boldsymbol{b}}(i)] & \text{se } b_i \neq 0. \end{cases}$$

Alla fine, la funzione occuperà $nr + 1{,}23\,n$ bit, più il necessario per la struttura di rango.

13.4 Auto-firma

Un utilizzo interessante delle funzioni che abbiamo descritto è quello di mantenere efficientemente un dizionario statico in maniera approssimata. Per farlo, consideriamo un insieme $X \subseteq U$ che vogliamo rappresentare, e una funzione di firma $s : U \to 2^r$.

Possiamo utilizzare la tecnica generale di rappresentazione delle funzioni per scrivere in $nr + 1{,}23\,n$ bit la funzione $f : X \to 2^r$ data da $f(x) = s(x)$. La funzione f, cioè, mappa ogni chiave nella propria firma. Il dizionario viene a questo punto interrogato come segue:

1. calcoliamo $f(x)$;
2. se $f(x) = s(x)$, restituiamo "appartiene a X"; altrimenti, restituiamo "non appartiene a X".

Si noti che se $x \in X$, $f(x)$ è la sua firma $s(x)$, e quindi restituiremo "appartiene a X". Se invece $x \notin X$, $f(x)$ sarà un valore arbitrario, e quindi restituiremo quasi sempre "non appartiene a X" tranne nel caso ci sia una collisione di firme, il che avviene con probabilità $1/2^r$. Tarando il numero r di bit delle firme è quindi possibile bilanciare spazio occupato e precisione della struttura.

14 L'albero di Fenwick

Descriveremo ora una struttura per il rango e la selezione su un vettore di bit *dinamico*. Per questo caso, c'è purtroppo un limite inferiore invalicabile di $\Omega(\log n / \log w)$ per effettuare aggiornamenti o interrogazioni (è ovviamente possibile rendere gli aggiornamenti costanti usando tempo lineare per le interrogazioni). Questo limite deriva da risultati sul problema della *somma di prefisso modulo 2*, in cui, dato un vettore di interi, si chiede di determinare qual è la parità della somma dei primi k. Si può mostrare che qualunque algoritmo, a fronte di un'opportuna sequenza di m operazioni di modifica del vettore e di interrogazione, deve utilizzare tempo $\Omega(m \log n / \log w)$.

Da un punto di vista pratico, però esiste una struttura semplice ed elegante le cui operazioni impiegano tempo $\log n - \log w + c - \log c$, per qualunque costante c, e occupano n/c bit addizionali.

L'idea è di separare il problema di fare rango e selezione su blocchi di c parole, che possiamo risolvere in tempo $O(c)$ con tecniche broadword, dal mantenere un campionamento dinamico del rango ogni cw bit. Per quest'ultimo problema, qualunque struttura che permetta di calcolare le somme di prefissi di un vettore di interi è sufficiente: il vettore contiene tanti elementi quanti sono i blocci di cw bit, e ogni elemento memorizza il numero di uni nel blocco. Quando aggiorniamo un bit è sufficiente modificare la struttura incrementando l'elemento associato al blocco. Il rango viene calcolato in modo ovvio.

Il problema del calcolo delle somme di prefissi di interi ha due soluzioni ovvie: tenere il vettore così com'è, calcolando al volo le somme (aggiornamento in tempo costante, somme in tempo lineare), o tenere un vettore che contiene le somme di prefissi (aggiornamento in tempo lineare, somme in tempo costante).

L'*albero di Fenwick* [**?**] è una struttura implicita (come un mucchio) che utilizza semplicemente un vettore della stessa dimensione del vettore originale, e permette di effettuare sia aggiornamenti che calcolo delle somme di prefissi in tempo logaritmico. L'idea è semplice quanto ingegnosa: assumiamo di avere un vettore $\boldsymbol{v}$ di n interi, e numeriamoli a partire da uno. Consideriamo ora un indice i, e la massima potenza di due 2^b che divide i (b è cioè la posizione del bit a uno meno significativo nella rappresentazione binaria di i). Associamo ora a i i'intervallo $(i - 2^{\mathrm{LSB}(i)} \,..\, i]$. L'albero di Fenwick associato a $\boldsymbol{v}$ è semplicemente il vettore $\boldsymbol{t}$ che contiene in posizione i (a partire da i) la somma $\sum_{i-2^{\mathrm{LSB}(i)} < k \leq i} v_k$, cioè la somma degli elementi di v i cui indici sono contenuti nell'intervallo associato a i. Notiamo in particolare che se i è dispari $t_i = v_i$, perché l'intervallo ha lunghezza 1. Inoltre, gli intervalli di lunghezza 2^k hanno estremo destro della forma $m2^k$ con m dispari. Una visualizzazione è data dalla Figura 4.

Lettura. Leggere le somme di prefissi da un albero di Fenwick è molto semplice: data una posizione i (stiamo calcolando così la somma *dei primi i elementi*), t_i ci dà la somma degli elementi nell'intervallo $(i - 2^{\mathrm{LSB}(i)} \,..\, i]$. A questo punto dobbiamo solo trovare la somma per prefissi dei primi $i - 2^{\mathrm{LSB}(i)}$ elementi, e continuiamo allo stesso modo. Notiamo però che così facendo stiamo passando da i a $i - 2^{\mathrm{LSB}(i)}$, a $i - 2^{\mathrm{LSB}(i)} - 2^{i-2^{\mathrm{LSB}(i)}}$ e così via: più semplicemente, stiamo cancellando iterativamente il bit più basso dell'indice in esame. Dato che si può cancellare il bit più basso con `x &= x - 1`, per leggere a somma dei primi i elementi da un albero di Fenwick il codice è

```
s = 0;
while(i != 0) {
    s += t[i];
    i &= i - 1;
}
```

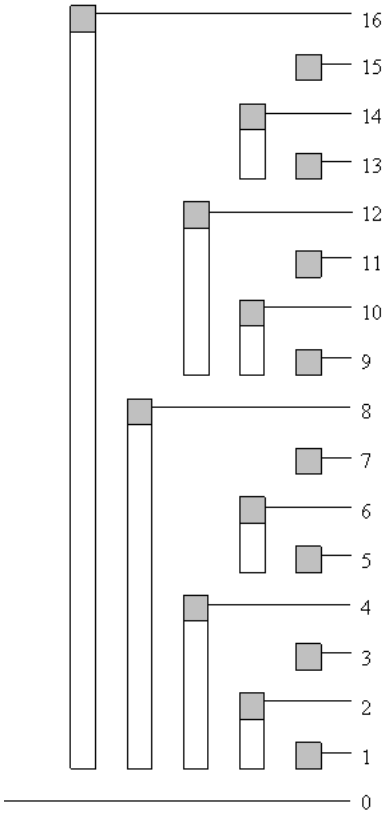


Figura 4: Una visualizzazione dell'albero di Fenwick su sedici elementi.

L'albero di Fenwick è detto, appunto, albero, perché l'insieme degli intervalli è strutturato come un albero in cui la relazione "padre di" si ottiene cancellando il bit più basso. Il codice appena descritto è, in questo senso, semplicemente una risalita verso la radice.

Ovviamente il ciclo non può essere eseguito più di $\log n$ volte, e verrà, in generale, eseguito un numero di volte pari al numero di uni nella rappresentazione binaria di i (si noti che funziona anche per $i = 0$). Il costo complessivo su tutti i valori di i è uguale alla somma delle cardinalità di tutti i sottoinsiemi di $\log n$, che è dell'ordine di $n \log n$: quindi, anche il costo ammortizzato per operazione è logaritmico (sebbene ci siano casi in cui il ciclo viene eseguito una sola volta).

Aggiornamento. Veniamo ora all'aggiornamento. Come abbiamo visto, l'albero di Fenwick non ha bisogno del vettore originale per calcolare le somme di prefissi, e in effetti anche l'aggiornamento non ne fa uso. Osserviamo che quando aggiorniamo con un incremento c (positivo o negativo) il vettore in posizione i, intanto dobbiamo aggiornare t_i, dato che l'intervallo associato contiene v_i. Ora dobbiamo però trovare tutti gli intervalli che contengono i

Chiaramente, per contenere i un intervallo deve cominciare nello stesso punto o prima dell'interallo che è associato a i, e terminare strettamente dopo, dato che non esistono due intervalli con intersezione non banale (cioè non vuota o uguale uno dei due intervalli). Ma il primo intervallo con questa proprietà è ovviamente l'intervallo che termina in $j = i + 2^{\mathrm{LSB}(i)}$, che è lungo almeno $2 \cdot 2^{\mathrm{LSB}(i)}$. Dato che non ci sono intervalli che si intersecano in maniera non banale, l'intervallo successivo che contiene i dovrà essere l'intervallo $(j - 2^{\mathrm{LSB}(j)} \,..\, j]$, e così via finche j non diventa più grande di n. Il codice è quindi

```
do {
    t[i] += c;
    i += (i & -i);
} while(i <= n);
```

Si noti l'espressione `i & -i`, che isola il bit più basso di i (cioè calcola $2^{\mathrm{LSB}(i)}$).

Accedere agli elementi del vettore. È interessante notare che l'albero di Fenwick permette anche di recuperare gli elementi originali del vettore *in tempo medio costante*. Per capire come mai ciò avvenga, notiamo che se vogliamo l'elemento in posizione i di $\boldsymbol{v}$, dobbiamo calcolare a differenza tra le somme di prefissi con i e $i - 1$ elementi. Ora, dato che queste somme vengono calcolate sommando i contributi di vari intervalli, possiamo limitarci a calcolare la differenza degli intervalli *che non sono comuni al calcolo delle due somme*. In particolare, si noti che per valutare la somma dei primi i elementi utilizzeremo l'intervallo di indice i e poi quello di indice $i - 2^{\mathrm{LSB}(i)}$: ma dato che $i - 1 \geq i - 2^{\mathrm{LSB}(i)}$, l'intervallo di indice $i - 2^{\mathrm{LSB}(i)}$ compare anche nella sommatoria relativa a $i - 1$. Basta sottrarre dal valore calcolato inizialmente in i i valori associati a $i - 1$ fino al raggiungimento dell'antenato comune $i - 2^{\mathrm{LSB}(i)}$:

```
e = t[i];
p = i & i - 1;
i = i - 1;
while(i != p) {
    e -= t[i];
    i &= i - 1;
}
```

Quante volte viene eseguito il ciclo interno? Esattamente $\mathrm{LSB}(i)$ volte (e infatti se i è dispari non viene eseguito). Quant'è allora il numero di accessi a $\boldsymbol{t}$ per recuperare tutti gli elementi del vettore? Per metà degli elementi c'è un solo accesso, per un quarto due, per un ottavo tre e così via. Ma $\sum_i i2^i = 2$, da cui deduciamo che il numero di accessi medio è solo due.

Ricerca per valore. Infine, notiamo che è molto semplice eseguire una ricerca binaria per localizzare la massima somma di prefissi minore o uguale a un numero dato. Basta costruire iterativamente la somma di prefissi desiderata: si parte con somma uguale zero e dall'indice dato dalla massima potenza di 2 minore di n: a ogni passo, si sceglie ridurre della metà l'intervallo in esame, o aggiungerne uno a destra, a seconda che la somma corrente più il valore dell'intervallo ecceda o no il numero dato.

Se l'intervallo corrente ha indice i, per dimezzare l'intervallo corrente è sufficiente sottrarre $2^{\mathrm{LSB}(i)-1}$, mentre per passare all'intervallo successivo occorre incrementare la somma corrente di t_i, e sommare, di nuovo, $2^{\mathrm{LSB}(i)-1}$, ma questi valori non è necessario calcolari, perché $\mathrm{LSB}(i)$ decresce a ogni iterazione:

```
i = 0;
m = 1 << MSB(n);
while(m != 0) {
    if (s >= t[i + m]) {
        i += m;
        s -= t[i];
    }
    m >>= 1;
}
```

Costruzione in tempo lineare. A partire da un vettore $\boldsymbol{v}$, il modo banale di costruire un albero di Fenwick è quello di incrementare l'indice i di v_i per tutti gli i, il che richiede tempo $O(n \log n)$. Come nel caso dei mucchi, però, esiste un semplice algoritmo lineare che costruisce l'albero direttamente. Innanzitutto poniamo $\boldsymbol{t} = \boldsymbol{v}$. Poi per i pari calcoliamo $t_i = t_i + v_{i-1}$, per i multiplo di quattro $t_i = t_i + v_{i-2}$ e così via. Le operazioni necessarie sono $n/2 + n/4 + n/8 + \cdots \leq n$. Partendo da un vettore $\boldsymbol{t}$ uguale a $\boldsymbol{v}$, il codice è come segue:

```
m = 2;
while(m <= n) {
    for(i = m; i <= n; i += m)
        t[i] += t[i - m/2];
    m <<= 1;
}
```

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.