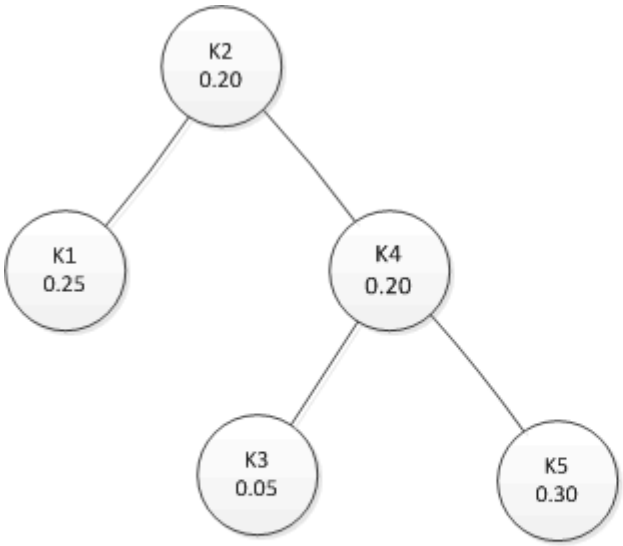
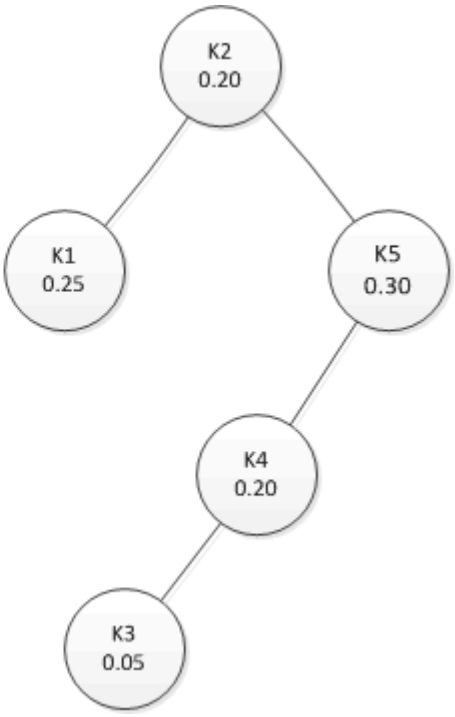




ALBERI BINARI DI RICERCA OTTIMALI

Progetto di Laboratorio di Algoritmi
Corso di Laurea in Informatica dell'Università di Catania

Elaborato di:

Martin Gibilterra

Contenuti

1. Preliminari

Un albero binario di ricerca (BST) è un grafo, i cui archi sono orientati solo verso il basso, dove ogni nodo ha archi entranti tranne uno, quello più in alto, che è detto radice dell'albero. In un albero binario di ricerca, ogni nodo può avere al più due figli e un nodo senza figli è chiamato nodo foglia. Il BST ha la caratteristica principale di essere una struttura dati utile per fare ricerche in base ad un criterio di ordinamento totale che prevede che per ogni dato nodo x, il suo sottoalbero sinistro conterrà solo nodi con chiave minore della chiave del nodo x, mentre il suo sottoalbero destro conterrà solo nodi con chiave maggiore della chiave del nodo x.

2. Definizione formale di albero binario di ricerca ottimale

Data una sequenza K = < k_1, k_2, …, k_n> di n chiavi **distinte e ordinate** in modo crescente e una sequenza P = <p_1, p_2, …, p_n> di n valori di probabilità, dove p_i rappresenta la probabilità che la chiave k_i sia oggetto di una ricerca all'interno dell'albero, si definisce albero binario di ricerca ottimale (che chiameremo optimal BST per brevità) un albero binario di ricerca costruito in modo tale da minimizzare il costo atteso di ogni ricerca, cioè il numero di nodi visitati prima di arrivare al nodo da cercare, conoscendo per ogni nodo la possibilità che sia esattamente quello da cercare.

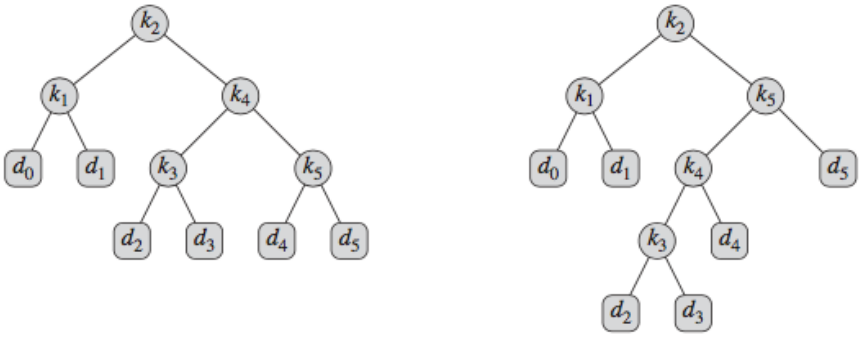


Figure 15.9 Two binary search trees for a set of $n = 5$ keys with the following probabilities:

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

(a) A binary search tree with expected search cost 2.80. **(b)** A binary search tree with expected search cost 2.75. This tree is optimal.

Figura 1: Due possibili BST per la sequenza di chiavi data

Caratteristiche di un optimal BST

La caratteristica principale di un optimal BST è che ogni ricerca può riguardare un valore appartenente all'insieme delle chiavi reali K oppure un valore che non si trova nella sequenza K. Di conseguenza, si introduce una sequenza D = $<d_0, d_1, \ldots, d_n>$ di n + 1 chiavi **fittizie** (dette anche chiavi *dummy*), rappresentanti valori non contenuti nella sequenza K, con le relative probabilità di ricerca Q = $<q_1, q_2, \ldots, q_n>$. La chiave dummy d_0 rappresenterà i valori più piccoli di k_1, d_n rappresenterà i valori più grandi di k_n e una generica di rappresenterà tutti i valori compresi tra k_i e k_{i+1}, per $1 \leq i \leq n-1$.

Di conseguenza, si può affermare che ogni chiave reale k_i è un nodo interno e ogni chiave dummy d_i è una foglia **nulla.**

Operazioni di dizionario

Le operazioni di inserimento e cancellazione di questa struttura dati sono perfettamente identiche in termini di complessità rispetto a un qualsiasi albero binario di ricerca. L'unica a variare è la ricerca, che in un normale BST ha complessità O(log n). Lo scopo del problema del calcolo dell'optimal BST date n chiavi è proprio quello di creare un albero binario di ricerca che abbassi questo limite asintotico.

Si definisce una qualsiasi ricerca all'interno dell'optimal BST come **riuscita** quando viene restituita una chiave reale, **fallita** quando viene restituita una chiave dummy.

Avendo, per qualsiasi ricerca, il 100% di possibilità di ottenere in output una chiave, si può affermare che:

$$\sum_{i=1}^{n} \boldsymbol{p_i} + \sum_{i=0}^{n} \boldsymbol{q_i} = \mathbf{1}$$

(1)

Il costo atteso di una ricerca in un albero binario di ricerca T, conoscendo la probabilità di ricerca di ogni chiave reale e dummy e supponendo che il costo di una ricerca sia la profondità del nodo + 1, con Pr_T profondità dell'albero T, è il seguente:

$$\sum_{i=1}^{n} (Pr_T(k_i) + 1) \, p_i + \sum_{i=0}^{n} (Pr_T(d_i) + 1) \, q_i =$$

$$1 + \sum_{i=1}^{n} Pr_T(k_i) \, p_i + \sum_{i=0}^{n} Pr_T(d_i) \, q_i \ .$$

(2)

Si consideri che l'ottimalità dell'albero non dipende dall'altezza totale, in quanto per una sequenza di n chiavi, considerando due alberi di altezza h_1 e h_2, dove $h_1 > h_2$, l'albero di altezza h_1 potrebbe avere costo minore rispetto all'albero di altezza h_2.

3. Costruzione di un optimal BST

Si analizzano di seguito le diverse tecniche che si possono utilizzare per trovare un optimal BST data la sequenza di chiavi con rispettive probabilità.

a. Approccio bruteforce

Per trovare un optimal BST, si possono considerare a turno tutte le possibili configurazioni di un albero binario con n chiavi e scegliere quella più conveniente in termini di costo. Tuttavia, questo approccio significherebbe computare un massimo di $(C_n)n!$ alberi binari, dove C_n è il numero di Catalan [4] e corrisponde all'espressione

$$\frac{(2n)!}{(n+1)!\,n!}$$

(3)

dove n è il numero di chiavi contenute nell'albero. Si dimostra che la ricerca di un optimal BST tra tutte le possibili configurazioni ha complessità $\Omega(4^n / n^{3/2})$. [1]

Tuttavia, un optimal BST non è un semplice albero binario ma è un albero binario **di ricerca** e l'insieme degli alberi binari di ricerca è contenuto nell'insieme degli alberi binari.
Di conseguenza, con l'approccio bruteforce si controlleranno al massimo C_n BST diversi.

b. Algoritmo di Knuth

Per abbassare la complessità sopradescritta, la soluzione migliore è usare una tecnica di programmazione dinamica bottom-up, perché usando la ricorsione i sottoproblemi che costituiscono il problema originale verrebbero ricalcolati inutilmente.

La programmazione dinamica si può applicare solo se il problema in questione è una struttura ottima costituita da sottostrutture ottime e l'algoritmo di Knuth per la costruzione di un optimal BST si basa sulle tecniche di programmazione dinamica.

a. Algoritmo di Knuth: analisi della sottostruttura ottima

L'analisi della sottostruttura ottima del problema di un optimal BST parte da un preconcetto fondamentale: in quanto l'optimal BST è un albero binario di ricerca, anche i suoi sottoalberi saranno degli alberi binari di ricerca.
Considerando quindi un sottoalbero qualsiasi di un BST si può affermare che esso contiene n nodi con chiavi contenute nell'intervallo $k_i \dots k_j$, dove $1 <= i <= j <= n$. Tuttavia, nell'optimal BST, un

sottoalbero contiene anche chiavi dummy e quindi oltre alle chiavi k_i … k_j comprenderà le chiavi d_{i-1} … d_j.

Quindi, se un optimal BST chiamato T ha un sottoalbero T' che contiene le chiavi k_i … k_j, allora T' deve essere ottimale anche per il sottoproblema che contiene le chiavi k_i … k_j e d_{i-1} … d_j. Si supponga adesso di dimostrare l'esistenza di un sottoalbero T'', il cui costo atteso è minore di quello di T'. Se esso esistesse, lo si potrebbe sostituire a T' ottenendo un albero T che ha un costo atteso minore rispetto a prima, però questo contraddice l'ipotesi iniziale per la quale T sarebbe un optimal BST. Si conclude quindi che ogni sottoalbero di un optimal BST è una sottostruttura ottima.

Dato l'utilizzo dell'approccio bottom-up per la risoluzione del problema, la soluzione al problema principale si otterrà sfruttando le soluzioni dei sottoproblemi. Tuttavia, bisogna distinguere i sottoproblemi del problema di trovare l'optimal BST a seconda della natura del sottoproblema stesso, in quanto esso può riguardare un sottoalbero non vuoto o un sottoalbero vuoto.

Nel caso di un sottoalbero non vuoto, esso conterrà una sequenza di chiavi reali k_i … k_j. Si consideri una chiave k_h qualsiasi all'interno della sequenza: essa sarà sempre la radice di un ulteriore sottoproblema il cui sottoalbero sinistro conterrà una sequenza di chiavi reali k_i … k_{r-1} e fittizie d_{i-1} … d_{r-1} e il cui sottoalbero destro conterrà le sequenze di chiavi k_{r+1} … k_j e fittizie d_r … d_j. Per trovare un optimal BST allora si determineranno, per ogni chiave radice k_r con i <= r <= j, tutti i sottoalberi ottimi che contengono le chiavi k_i … k_{r-1} e k_{r+1} … k_j, scegliendo quello che ha la radice che garantisce l'ottimalità.

Nel caso di un sottoalbero vuoto, che si ottiene scegliendo come radice la chiave k_i o k_j, dal precedente ragionamento si deduce che entrambi i sottoalberi di entrambe le chiavi radice non contengono nessuna chiave reale ma contengono rispettivamente le chiavi fittizie d_{i-1} e d_j.

ii. Algoritmo di Knuth: soluzione ricorsiva

Ogni sottoproblema ottimo va risolto ricorsivamente fino ad arrivare alla soluzione del problema principale. Ciò vuol dire che ogni risoluzione passa dalla ricerca di un optimal BST che contenga le chiavi k_i … k_j, per i >= 1, i-1 >= j >= n.

Sia e[i,j] il costo di ricerca atteso in un optimal BST contenente la sequenza di chiavi k_i … k_j. Questo valore viene calcolato grazie ad una formula di ricorsione composta da un caso base e da uno generico.
Il caso base si verifica quando j = i-1: ciò vuol dire che la sequenza di chiavi comprende le chiavi dalla i alla i-1, risultando quindi priva di chiavi reali e comprendendo solo quella dummy d_{i-1}. In questo caso, e[i,i-1] = q_{i-1}, cioè il valore di probabilità di estrazione della chiave dummy di indice i-1. Quando si verifica il caso generico j >= i, bisogna selezionare a turno ognuna delle radici k_r comprese tra k_i e k_j. Una volta scelto k_r, esso genererà due sottoalberi optimal BST, il sinistro che contiene le chiavi comprese tra k_i e k_{r-1} e il destro che contiene quelle comprese tra k_{r+1} e k_j.

Una volta scelta la radice e creati i suoi due sottoalberi, la profondità dei nodi di quei sottoalberi aumenta di 1 e, per l'equazione **(2)** il costo atteso di ricerca di ognuno dei due sottoalberi aumenta della somma di tutte le probabilità dei nodi del sottoalbero.

Sia **w[i,j]** la somma delle probabilità dei nodi del sottoalbero contenente le chiavi k_i … k_j, definita come segue:

$$\boldsymbol{w[i,j]} = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

Quindi, con k_r radice di un sottoalbero ottimo, si ha che:

$$e[i,j] = e[i,r-1] + e[r+1,j] + w(i,j)$$

L'equazione ricorsiva finale che si ottiene, per il calcolo dell'optimal BST secondo Knuth, è la seguente:

$$e[i,j] = \begin{cases} q_{i-1} & se\ j = i-1 \\ \min(i \le h \le j)\ \{ e[i,r-1] + e[r+1,j] + w(i,j)\} & se\ j \ge i \end{cases} \qquad \textbf{(4)}$$

dove e[i,j] è il valore del costo atteso di ricerca nell'optimal BST contenente chiavi da k_i a k_j.

iii. Algoritmo di Knuth: calcolo del costo di un optimal BST

Implementare l'equazione **(4)** per ricorsione diretta sarebbe totalmente inefficiente, quindi, usando le tecniche di programmazione dinamica, si alloca una tabella E[n+2, n+1] che contiene i valori e[i,j]. Il primo indice va da 1 a n+1 perché si deve calcolare la cella e[n+1, n] in cui si ottiene un sottoalbero con la sola chiave fittizia d_n, il secondo va da 0 a n perché si deve calcolare la cella e[1,0] in cui si ottiene un sottoalbero con la sola chiave fittizia d_0.

Si alloca una tabella root[i,j] che servirà a memorizzare la radice del sottoalbero contenente le chiavi da k_i a k_j.

Si alloca anche un'altra tabella W[n+2,n+1] per calcolare i valori w(i,j) senza ricalcolarli ad ogni valutazione di e[i,j].

A questo punto calcoliamo le celle del caso base w[i, i-1] = q_{i-1} per 1 <= i <= n+1 e successivamente verranno calcolati gli n^2 valori w[i,j] = w[i, j-1] + p_j + q_j.

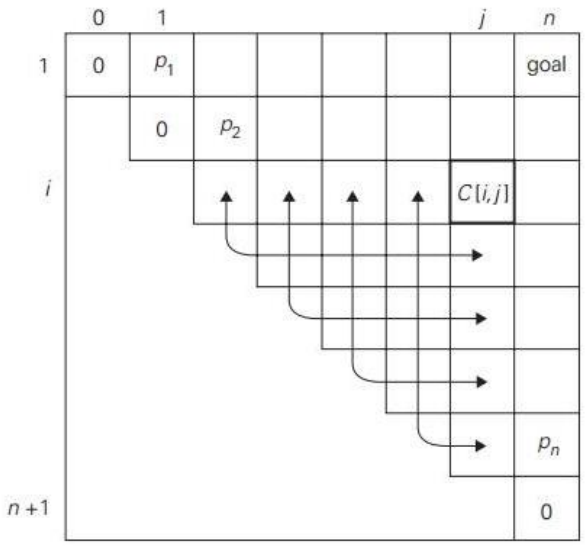


Tabella 1 - C[i,j] è il costo atteso e[i,j]

Di seguito viene presentato uno pseudocodice dell'algoritmo di programmazione dinamica di Knuth.

```
OPTIMAL-BST(p, q, n)
 1  let e[1..n+1, 0..n], w[1..n+1, 0..n],
          and root[1..n, 1..n] be new tables
 2  for i = 1 to n + 1
 3      e[i, i − 1] = q_{i−1}
 4      w[i, i − 1] = q_{i−1}
 5  for l = 1 to n
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          e[i, j] = ∞
 9          w[i, j] = w[i, j − 1] + p_j + q_j
10          for r = i to j
11              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
12              if t < e[i, j]
13                  e[i, j] = t
14                  root[i, j] = r
15  return e and root
```

iv. Complessità dell'algoritmo

Questo algoritmo ha una complessità pari all'algoritmo di moltiplicazione di sequenze di matrici, cioè $\Omega(n^3)$. Lo si deduce dalla presenza **di 3 for annidati** che eseguono n iterazioni ciascuno.

L'algoritmo di Knuth è l'unico algoritmo esistente a restituire un perfetto optimal BST, mentre ne esistono altri, anche più efficienti, che non sempre restituiscono un optimal BST veramente ottimale.

4. Cenni su altri algoritmi: le due regole di Knuth

a. Ottimizzazione dell'algoritmo di Knuth

Knuth stesso, ideando il suo algoritmo, suggerì due regole per eventualmente renderlo più efficiente.
La prima regola consigliata da Knuth è quella di scegliere come radice dell'eventuale optimal BST, per ogni range di probabilità fornito dai dati in input, sempre la chiave con la probabilità più alta (in caso di n chiavi con la probabilità più alta si sceglie quella con indice **mediano**), per poi creare i sottoalberi sinistro e destro contenenti le chiavi con probabilità subito più bassa. La conseguenza principale di questo approccio è che vicino alla radice dell'optimal BST compariranno chiavi con probabilità di estrazione molto alta e ciò potrebbe diminuire il tempo atteso di ricerca nell'albero.

Tuttavia, non sempre questo approccio genera un optimal BST, però abbasserebbe la complessità dell'algoritmo a $O(n^2)$. [5]

b. Algoritmo di approssimazione di Mehlhorn

L'algoritmo di approssimazione ideato da Mehlhorn per la costruzione di optimal BST si basa sulla seconda regola di Knuth. La seconda regola è quella di scegliere, per ogni range di probabilità fornito dai dati in input, sempre la radice che equalizzi le due somme delle probabilità di estrazione delle chiavi (la somma delle probabilità a sinistra della radice e quella delle probabilità alla sua destra) il più possibile. Questo approccio potrebbe generare alberi molto sbilanciati in termini di altezza ma molto bilanciati in termini di peso, facendo sì che ogni nodo possa essere raggiunto in tempi di ricerca approssimativamente ottimali.

Questo algoritmo è utile quando il numero di nodi presenti nella sequenza delle chiavi è enorme e restituisce un optimal BST in tempo O(n).
Con questo algoritmo la lunghezza massima di un cammino pesato all'interno dell'albero sarà esattamente:

$$2 + \frac{H}{1 - \log(\sqrt{5} - 1)}$$

dove H è l'entropia della distribuzione delle probabilità delle chiavi.

Si conclude che, dal momento che la lunghezza massima di un cammino pesato dell'albero non può essere maggiore di H / log(3), l'algoritmo restituisce un albero quasi ottimale. [2] [5]

c. Algoritmo di Garsia-Wachs

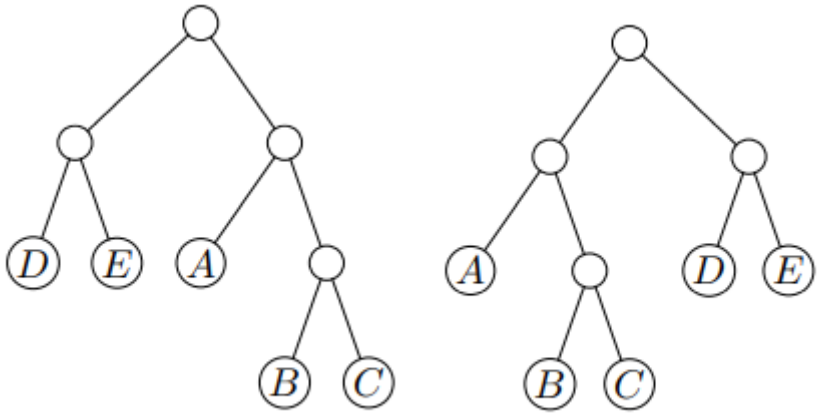
Dati in input una sequenza di valori $k_0 \dots k_n$ associati ai rispettivi pesi o probabilità $p_0 \dots p_n$, l'algoritmo di Garsia-Wachs costruisce un albero binario in cui i valori $k_0 \dots k_n$ sono le foglie $f_0 \dots f_n$ dell'albero e in cui la sommatoria:

$$\sum_{i=0}^{n} p_i D_i$$

dove D_i è la distanza del nodo foglia f_i dalla radice, è minima.

L'algoritmo si articola in tre fasi fondamentali:

1. Viene costruito un optimal BST i cui nodi foglia sono in disordine, cioè in cui esiste almeno un i tale che $f_i \neq k_i$
2. Con una visita dell'albero, si calcola la profondità di ciascuna foglia f_i
3. Viene costruito un ulteriore optimal BST i cui nodi foglia sono in ordine corretto rispetto quello stabilito nella fase 1 e rendono vera $f_i = k_i$ per ogni i.



result of phase 1 final tree

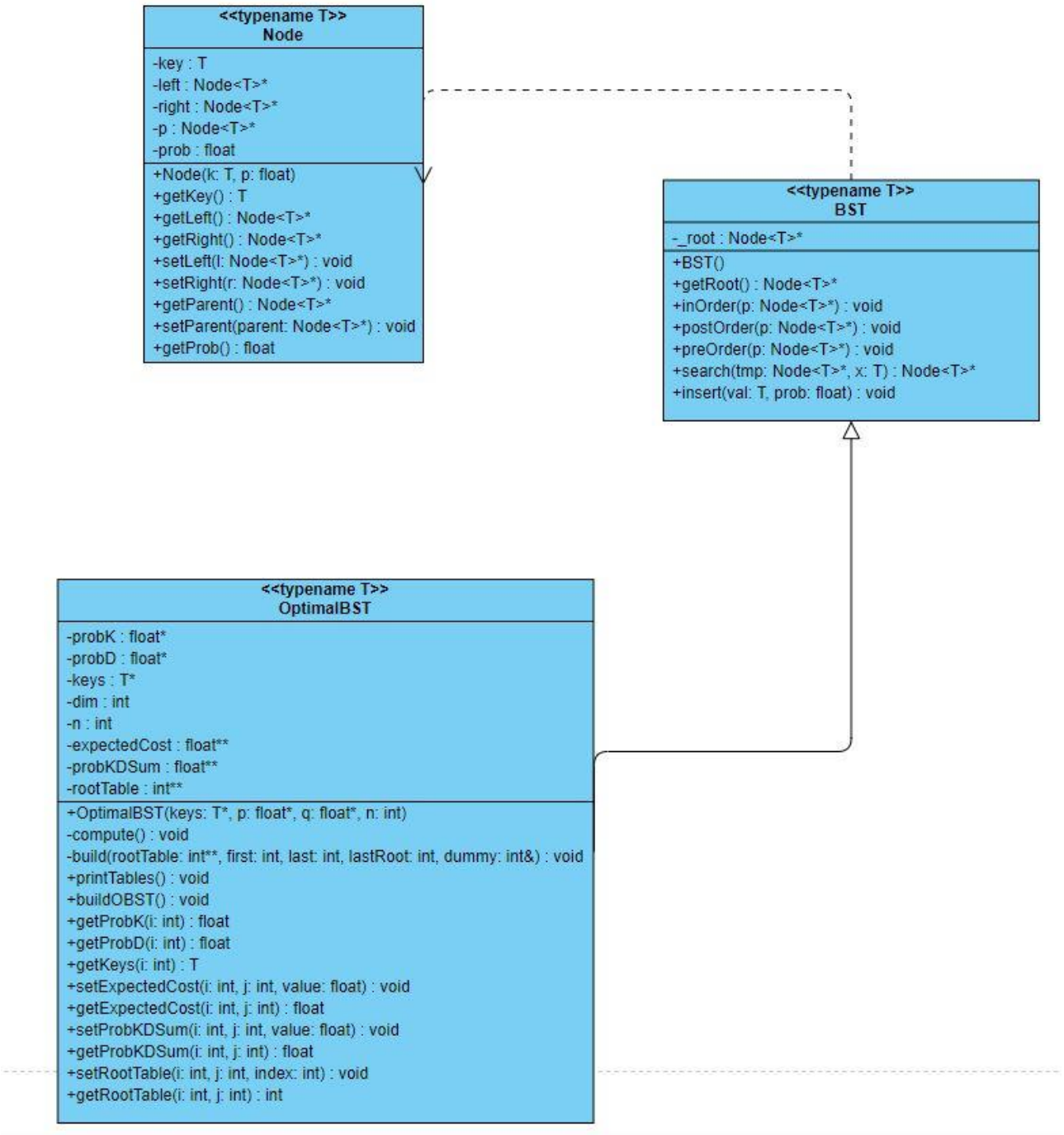
La prima fase inizia creando una lista L di alberi pesati t_0 … t_n dove inizialmente $t_i = k_i$. Fino a quando la lista L conterrà più di un elemento, si esegue il seguente ciclo di passaggi:

1. Si applica una scelta di programmazione greedy, determinando ad ogni ciclo, se esiste, il più piccolo i tale che $p_{i-1} <= p_{i+1}$. Questo numero esisterà sempre se si pongono agli estremi della lista due valori sentinella sufficientemente grandi in modo tale da verificare sempre la disequazione $p_{i-1} <= p_{i+1}$ quando uno dei due membri è un valore sentinella;
2. si estraggono dalla lista L i due alberi t_{i-1} e t_i, che diventeranno rispettivamente il sottoalbero sinistro e destro di un nuovo albero T, il cui peso diventerà $p_{i-1} + p_i$;
3. T viene inserito nella lista al posto dei due alberi estratti;

La prima fase darà in output un albero T, l'unico rimasto nella lista L, i cui nodi foglia sono in disordine. Dopo il calcolo delle profondità dei nodi foglia fi nell'albero T, la terza fase ricostruisce l'albero binario inserendo ogni nodo k_i alla profondità di valore D_i, ottenendo un optimal BST in cui ogni nodo foglia $f_i = k_i$.

Si dimostra che la complessità di questo algoritmo, su un input di n nodi, è O(n log n). [3]

5. Implementazione dell'algoritmo di Knuth



Il diagramma UML mostra la gerarchia delle classi utilizzate per realizzare la struttura dati dell'optimal BST e il calcolo del suo costo.

In particolare, la classe OptimalBST eredita i metodi della classe BST, costituita da strutture di tipo Node collegate tra loro. Nella classe OptimalBST, sono forniti:

- metodi getter per ottenere la probabilità di estrazione di una chiave reale (getProbK()) e di una chiave dummy (getProbD()), dato l'indice della chiave in esame (il cui valore è contenuto nell'array keys);

- metodi getter per ottenere, dati due indici i e j corrispondenti all'indice della prima e ultima chiave della sequenza di chiavi k_i … k_j considerata, il costo dell'OBST contenente quella sequenza di chiavi (getExpectedCost()), la somma delle probabilità di quelle chiavi (getProbKDSum()) e la radice migliore tra le chiavi della sequenza (getRootTable());
- un metodo getter per ottenere una chiave dato il suo indice nell'array keys;
- metodi setter per modificare i valori illustrati al secondo punto;
- un metodo di costruzione della struttura dati OBST data la tabella delle radici migliori, tramite inserimenti (build()).

Bibliografia

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, L. Colussi – "Introduzione agli algoritmi e strutture dati, terza edizione" – McGrawHill, 2010

[2] Kurt Mehlhorn – "Nearly Optimal Binary Search Trees" – Acta Informatica – Springer-Verlag, 1975

[3] Jean-Christophe Filliatre – "A Functional Implementation of the Garsia–Wachs Algorithm (functional pearl)" – LRI, Univ Paris-Sud, Orsay F-91405 – 2008

[4] David Witmer, Ellango Jothimurugesan, Ziqiang Feng – "Dynamic Programming I: Optimal BSTs" – 2016

[5] Corianna Jacoby, Alex King – "Comparing Implementations of Optimal Binary Search Trees" – Tufts University, 2017