

Proiectarea algoritmilor: complexitatea medie

Ștefan Ciobâcă, Dorel Lucanu

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
dlucanu@info.uaic.ro

PA 2015/2016

- 1 Reamintire cursul precedent
- 2 Timpul mediu pentru algoritmi determiniști
- 3 Studii de caz
 - Prima apariție într-o listă
 - Quicksort
 - Șuruburi și piulițe (Nuts and Bolts)
 - Treaps

Plan

- 1 Reamintire cursul precedent
- 2 Timpul mediu pentru algoritmi determiniști
- 3 Studii de caz
 - Prima apariție într-o listă
 - Quicksort
 - Șuruburi și piulițe (Nuts and Bolts)
 - Treaps

Algoritmi nedeterminiști

- pentru anumite stări există mai multe posibilități de a continua calculul (presupunem că acestea sunt în număr finit)
- în consecință, pentru aceeași intrare, algoritmul poate da rezultate (configurații finale) diferite
- instrucțiuni specifice:

`choose x in S ;` – întoarce un element din S ales arbitrar

Timp de execuție (uniform): $O(1)$

`choose x in S s.t. B ;` – întoarce un element din S care satisface condiția B ales arbitrar

Timp de execuție (uniform): $T(B)$

- un program nedeterminist are mai multe **fire de execuție**
- un **program nedeterminist rezolvă P** dacă $\forall x \in P \exists$ un fir de execuție care se termină și a cărei configurație finală include $P(x)$

Algoritmi nedeterminiști pentru probleme de decizie

- Activitatea unui algoritm nedeterminist pentru o problemă de decizie:
 - “se ghicește” o anumită structură S
 - verifică dacă S satisface o proprietățile cerute de întrebare
 - dacă proprietatea este satisfăcută, se apeleaza instrucțiunea de terminare success, altfel instrucțiunea de terminare failure;
- Extensia limbajului:
 - `success;` – semnalează terminarea verificării (și a a algoritmului) cu succes
 - `failure;` – semnalează terminarea fără succes

Algoritmi probabiliști

- caz particular de algoritmi nedeterminiști când există distribuție de probabilitate peste mulțimea alegerilor
- $\text{random}(n)$ - întoarce un număr $x \in \{0, 1, \dots, n-1\}$ ales aleatoriu uniform (i.e. cu probabilitatea $\frac{1}{n}$)

Timp de execuție: $O(1)$

- O **variabilă aleatorie** este o funcție X definită peste un câmp de evenimente Ω
- **Valoarea medie** a lui X : $M(X) = \sum_i x_i \cdot p_i$

Proprietăți ale mediei:

$$M(X + Y) = M(X) + M(Y)$$

$$M(X \cdot Y) = M(X) \cdot M(Y)$$

(X și Y independente)

Algoritmi Monte Carlo

Pot produce rezultate incorecte cu o mică probabilitate, i.e. componenta de "output" este variabilă aleatorie.

Dacă un astfel de algoritm este executat de mai multe ori peste alegeri independente alese aleatoriu de fiecare dată, probabilitatea de eșec poate fi oricât de mică dar cu un timp de execuție mai mare.

Exemplu: test de compoziționalitate (primalitate)

Algoritmi Las Vegas

Nu produc niciodată rezultate incorecte, dar timpul poate varia de la o execuție la alta (este variabilă aleatorie).

Alegerile aleatorii sunt făcute pentru a stabili timpul mediu de execuție, care în esență este independent de intrare.

Exemplu: k -mediana

Plan

- 1 Reamintire cursul precedent
- 2 Timpul mediu pentru algoritmi determiniști
- 3 Studii de caz
 - Prima apariție într-o listă
 - Quicksort
 - Șuruburi și piulițe (Nuts and Bolts)
 - Treaps

Motivație

- A rezolvă P , $x \in P$
- timpul în cazul cel mai nefavorabil:

$$T_A(n) = \sup\{time_A(x) \mid x \in P \wedge size(x) = n\}$$
- $T_A(n)$ este irelevant dacă numărul instanțelor x cu $size(x) = n \wedge time_A(x) = T_A(n)$ (sau $time(A, x) = T_A(n) - \varepsilon$) este foarte mic

Definiție

- $time_A(x)$ ca variabilă aleatorie:
 - eveniment = execuția algoritmului (determinist) pentru o instanță aleatorie x ,
 - valoarea asociată evenimentului = durata execuției algoritmului
- $time_A^n$ restricția lui $time_A$ la instanțele x cu $size(x) = n$
- legea de repartiție a acestei variabile aleatorii: $Pr(time_A^n = t)$
- **timpul de execuție mediu** = media acestei variabile aleatorii

$$exp-time(n) = M(time_A^n)$$

- Caz particular: $\{time_A(x) \mid size(x) = n\} = \{t_0, t_1, \dots\}$,
 $Pr(time_A^n = t_i) = p_i$

$$exp-time(n) = \sum_i t_i \cdot p_i$$

Plan

- 1 Reamintire cursul precedent
- 2 Timpul mediu pentru algoritmi determiniști
- 3 Studii de caz**
 - Prima apariție într-o listă
 - Quicksort
 - Șuruburi și piulițe (Nuts and Bolts)
 - Treaps

Plan

- 1 Reamintire cursul precedent
- 2 Timpul mediu pentru algoritmi determinați
- 3 Studii de caz
 - Prima apariție într-o listă
 - Quicksort
 - Șuruburi și piulițe (Nuts and Bolts)
 - Treaps

Problema

Problema FIRST OCCURRENCE

Input: $n, a = (a_0, \dots, a_{n-1}), z$, toate numere întregi.

Output: $poz = \begin{cases} \min\{i \mid a_i = z\} & \text{dacă } \{i \mid a_i = z\} \neq \emptyset, \\ -1 & \text{altfel.} \end{cases}$

Algoritm pentru FIRST OCCURRENCE

Algoritmul FOAlg descris de următorul program rezolvă FIRST OCCURRENCE:

```
//@input: un tablou a cu n elemente, z
//@output: pozitia primului element din a egal cu z,
//          -1 daca nu exista un astfel de element
i = 0;
while (a[i] != z) && (i < n-1)
    i = i+1;
if (a[i] == z) poz = i;
else poz = -1;
```

Am văzut în cursul 3 că numărul de comparații în cazul cel mai nefavorabil este n .

Timpul mediu pentru FOAlg 1/2

- dimensiune instanță: $n = a.size()$
- operații măsurate: comparații care implică elemente ale lui a
- $\{time_{FOAlg}(x) \mid size(x) = n\} = \{i \mid 2 \leq i \leq n\}$
- e dificil de spus probabilitatea cu care apare o anumită instanță

Presupuneri:

- probabilitatea ca $z \in \{a_0, \dots, a_{n-1}\}$ este q și
- probabilitatea ca z să apară prima dată pe poziția i este $\frac{q}{n}$ (indicii i candidează cu aceeași probabilitate pentru prima apariție a lui z).

Timpul mediu pentru FOAlg 2/2

$$Pr(z \notin \{a_0, \dots, a_{n-1}\}) = 1 - q$$

$$Pr(time_{FOAlg}^n(p) = i) = \frac{q}{n} = p_i, \quad 1 \leq i < n$$

$$Pr(time_{FOAlg}^n(p) = n) = \frac{q}{n} + (1 - q) = p_n$$

Timpul mediu de execuție este:

$$\begin{aligned} exp-time(n) &= \sum_{i=2}^n p_i \cdot i \\ &= \sum_{i=2}^{n-1} \frac{q}{n} \cdot i + \left(\frac{q}{n} + (1 - q)\right) \cdot n \\ &= n - \frac{q}{n} - \frac{n-1}{2} \cdot q \end{aligned}$$

Pentru $q = \frac{1}{2}$ avem $exp-time(n) = \frac{3n+1}{4} - \frac{1}{2n}$.

Pentru $q = 1$ avem $exp-time(n) = \frac{n+1}{2} - \frac{1}{n}$.

Plan

- 1 Reamintire cursul precedent
- 2 Timpul mediu pentru algoritmi determiniști
- 3 Studii de caz
 - Prima apariție într-o listă
 - **Quicksort**
 - Șuruburi și piulițe (Nuts and Bolts)
 - Treaps

Quicksort: descriere

Este proiectat pe paradigma divide-et-impera.

Algoritmul Quicksort

Input: $S = \{a_0, \dots, a_{n-1}\}$

Output: o secvență cu elementele a_i în ordine crescătoare

- ① se alege $x = a_k \in S$
- ② calculează
 $S_{<} = \{a_i \mid a_i < x\}$ $S_{=} = \{a_i \mid a_i = x\}$ $S_{>} = \{a_i \mid a_i > x\}$
- ③ sortează recursiv $S_{<}$ și $S_{>}$ producând $Seq_{<}$ și $Seq_{>}$, respectiv
- ④ întoarce secvența $Seq_{<}, S_{=}, Seq_{>}$

Quicksort: partiționarea

Putem utiliza algoritmul Lomuto pentru k -mediană:

```
partition(out a, p, q)
{
    pivot = a[q];
    i = p - 1;
    for (j = p; j < q; ++j)
        if (a[j] <= pivot) {
            i = i + 1;
            swap(a, i, j);
        }
    swap(a, i+1, q);
    return i + 1;
}
```

Quicksort: algorithm

@input: $a = (a[p], \dots, a[q])$

@output: elementele secvenței a în ordine crescătoare

```
qsortRec(out a, p, q)
```

```
{  
    if (p < q) {  
        k = partition(a, p, q);  
        qsort(a, p, k-1);  
        qsort(a, k+1, q);  
    }  
}
```

```
qsort(out a)
```

```
{  
    qsortRec(a, 0, n-1);  
}
```

Quicksort: timpul în cazul cel mai nefavorabil

- dimensiune instanță: $n = a.size()$
- operații măsurate: comparații care implică elementele tabloului
- cazul cel mai nefavorabil: tabloul deja ordonat
- numărul de comparații pentru acest caz:
 $(n - 1) + (n - 2) + \dots + 1 = O(n^2)$

Quicksort: timpul mediu

Timpul mediu pentru qsort este egal cu timpul mediu pentru varianta probabilistă ("randomized quicksort").

"Randomized Quicksort", intuitiv

- exemplul canonic pentru algoritmi Las Vegas

Algoritmul RQS

Input: $S = \{a_0, \dots, a_{n-1}\}$

Output: elementele a_i în ordine crescătoare

- 1 dacă $n = 1$ întoarce a_0 , altfel **alege aleatoriu** $x = a_k \in S$
- 2 calculează
$$S_{<} = \{a_i \mid a_i < a_k\}$$
$$S_{=} = \{a_i \mid a_i = a_k\}$$
$$S_{>} = \{a_i \mid a_i > a_k\}$$
- 3 sortează recursiv $S_{<}$ și $S_{>}$ producând $Seq_{<}$ și $Seq_{>}$, resp
- 4 întoarce secvența $Seq_{<}, Seq_{=}, Seq_{>}$

"Randomized Quicksort", algoritmic

Se utilizează algoritmul de partiționare aleatorie:

```
randPartition(out a, p, q) {  
    if (p < q) {  
        i = p + random(q - p);  
        swap(a, i, q);  
        return partition(a, p, q);  
    }  
}
```

```
randQsortRec(out a, p, q) {  
    if (p < q) {  
        k = randPartition(a, p, q);  
        qsort(a, p, k-1);  
        qsort(a, k+1, q);  
    }  
}
```

```
RQS(out a) { randQsortRec(a, 0, n-1); }
```

Analiza algoritmului RQS

Fie funcția $rank$ astfel încât $a_{rank(0)} \leq \dots \leq a_{rank(n-1)}$.

Definim $X_{ij} = \begin{cases} 1 & a_{rank(i)} \text{ și } a_{rank(j)} \text{ sunt comparate} \\ 0 & \text{altfel} \end{cases}$

X_{ij} numără comparațiile dintre $a_{rank(i)}$ și $a_{rank(j)}$

X_{ij} este variabilă aleatorie și se numește **indicator de variabilă**

$C(n)$ - variabilă aleatorie care dă numărul de comparații

$$C(n) = \sum_{j>i} X_{ij}$$

Numărul mediu de comparații este

$$M(C(n)) = M\left(\sum_{i=0}^{n-1} \sum_{j>i} X_{ij}\right) = \sum_{i=0}^{n-1} \sum_{j>i} M(X_{ij})$$

Analiza algoritmului RQS 1/3

p_{ij} probabilitatea ca $a_{rank(i)}$ și $a_{rank(j)}$ să fie comparate într-o execuție

$$M(X_{ij}) = p_{ij} \times 1 + (1 - p_{ij}) \times 0 = p_{ij}$$

Presupunem $i < j$; $a_{rank(i)}$ și $a_{rank(j)}$ sunt comparate dacă și numai dacă primul pivot din secvența

$$a_{rank(i)}, \dots, a_{rank(j)}$$

este $a_{rank(i)}$ sau $a_{rank(j)}$. Altfel pivotul separă cele două elemente în subsecvențe diferite.

$$\text{Rezultă } p_{ij} = \frac{2}{j - i + 1}.$$

Analiza algoritmului RQS 2/3

$$\begin{aligned}\sum_{i=0}^{n-2} \sum_{j>i} p_{ij} &= \sum_{i=0}^{n-2} \sum_{j>i} \frac{2}{j-i+1} \\ &\leq \sum_{i=0}^{n-2} \sum_{k=1}^{n-i-1} \frac{2}{k} \\ &\leq 2 \sum_{i=0}^{n-2} \sum_{k=1}^{n-i-1} \frac{1}{k}\end{aligned}$$

Analiza algoritmului RQS 3/3

Avem:

$$\sum_{k=1}^{n-i-1} \frac{1}{k} = H_{n-i-1} = \Theta(\log(n-i-1)),$$

$$\sum_{i=0}^{n-2} \Theta(\log(n-i-1)) = \Theta(\log(n-1)!) = \Theta(n \log n).$$

Theorem

Numărul mediu de comparații într-o execuție al algoritmului RQS este cel mult $2nH_n = O(n \log n)$.

Exerciții

Să se determine în aceeași manieră complexitățile medii pentru insertSort și bubbleSort. Se vor considera două cazuri:

- se numără interschimbările (inversiunile) din tablou
- se numără deplasările elementelor din tablou

Quicksort: timpul mediu, recursiv 1/2

- lungimea secvenței: $q + 1 - p = n$
- probabilitatea ca x sa fie al k -lea element: $\frac{1}{n}$
- dimensiuni subprobleme: $k - p = i - 1$ și $q - k = n - i$
- $C(n)$ - numărul de comparații
- fie Y_i variabila aleatorie care întoarce numărul mediu de comparații dat de apelurile recursive dacă funcția de partiționare întoarce k , 0 altfel

$$Y_i = \begin{cases} M(C(i-1)) + M(C(n-i)) & , i = k \\ 0 & i \neq k \end{cases}$$

- $M(Y_i) = \frac{1}{n}(M(C(i-1)) + M(C(n-i)))$
- numărul de comaparații:

$$C(n) = (n-1) + \sum_{i=1}^n Y_i$$

Quicksort: timpul mediu, recursiv 2/2

- numărul mediu de comparații:

$$\begin{aligned}
 \text{exp-time}(n) &= M(C(n)) = M((n-1) + \sum_{i=1}^n Y_i) \\
 &= \begin{cases} (n-1) + \frac{1}{n} \sum_{i=1}^n (M(C(i-1)) + M(C(n-i))) & , \text{dacă } n \geq 1 \\ 1 & , \text{dacă } n = 0 \end{cases}
 \end{aligned}$$

Theorem

Complexitatea medie a algoritmului QuickSort este $O(n \log_2 n)$.

Rezolvare recurență

Notăție: $c(n) = M(C(n))$

$$c(n) = (n-1) + \frac{2}{n}(c(0) + \dots + c(n-1))$$

$$nc(n) = n(n-1) + 2(c(0) + \dots + c(n-1))$$

Trecem pe n în $n-1$:

$$(n-1)c(n-1) = (n-1)(n-2) + 2(c(0) + \dots + c(n-2))$$

Scădem:

$$nc(n) = 2(n-1) + (n+1)c(n-1)$$

Împărțim prin $n(n+1)$ și rezolvăm recurența obținută:

$$\begin{aligned} \frac{c(n)}{n+1} &= \frac{c(n-1)}{n} + \frac{2}{n+1} - \frac{2}{n(n+1)} \\ &= \frac{c(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} - \left(\frac{2}{(n-1)n} + \frac{2}{n(n+1)} \right) \\ &= \dots \\ &= \frac{c(0)}{1} + \frac{2}{1} + \dots + \frac{2}{n+1} - \left(\frac{2}{1 \cdot 2} + \dots + \frac{2}{n(n+1)} \right) \\ &= O(\log n) - O(1) \end{aligned}$$

Plan

- 1 Reamintire cursul precedent
- 2 Timpul mediu pentru algoritmi determinați
- 3 Studii de caz
 - Prima apariție într-o listă
 - Quicksort
 - Șuruburi și piulițe (Nuts and Bolts)
 - Treaps

Domeniul problemei

- n șuruburi și n piulițe de mărimi aproximativ egale
- dar nu se poate decide care șurub (piuliță) e mai mare și care e mai mic(ă) (nu se pot compara direct două șuruburi între ele; la fel pentru piulițe)
- fiecare piuliță se potrivește exact cu un șurub
- când se compară un șurub cu o piuliță, se poate vedea dacă se potrivesc sau dacă șurubul este mai mare sau mai mic decât piuliță
- sarcina de rezolvat constă în găsirea piuliței potrivite pentru fiecare șurub

Obs. Abia în 1995 s-a proiectat un algoritm care rezolvă problema în timpul $O(n \log n)$:

Janos Komlos, Yuan Ma, and Endre Szemerédi. Sorting nuts and bolts in $O(n \log n)$ time, SIAM J. Discrete Math 11(3):347-372, 1998. Technical Report MPI-I-95-1-025, Max-Planck-Institut für Informatik, September 1995.

Versiunea simplificată

Presupunem că trebuie să găsim piulița potrivită pentru un șurub dat.

Fie $\text{cmp}(N, B)$ o funcție care întoarce:

−1 dacă piulița N este mai mică decât șurubul B ,

0 dacă piulița N se potrivește cu șurubul B ,

1 dacă piulița N este mai mare decât șurubul B .

$\text{cmp}(P, S)$ este defintă pentru toate datele de intrare.

`NutsAndBolt (NL, B)`

```
{  
    n = NL.size();  
    for (i=0; i < n - 1; ++i)  
        if (cmp(NL[i], B) == 0) return i;  
    return n - 1;  
}
```

Numărul mediu de comparații

Presupunem că ordinea piulițelor în lista NL (care coincide cu ordinea în care sunt testate) este o variabilă aleatorie uniformă.

Rezultă că probabilitatea ca piulița căutată să fie pe poziția i este $\frac{1}{n}$ (de ce?).

Fie $C(n)$ variabila aleatorie care dă numărul de comparații.

Presupunem că NL este nevidă și totdeauna există o piuliță care se potrivește cu B .

Valorile posibile pentru $C(n)$: $1, 2, \dots, n - 1$.

Probabilitatea ca $C(n) = i$, $1 \leq i < n - 1$: $\frac{1}{n}$

Probabilitatea $C(n) = n - 1$: $\frac{2}{n}$

Calculul numărului mediu de comparații

$$\begin{aligned}M(C(n)) &= \sum_{i=1}^{n-2} i \cdot \frac{1}{n} + (n-1) \cdot \frac{2}{n} \\&= \sum_{i=1}^{n-1} \frac{i}{n} + \frac{n-1}{n} \\&= \frac{n(n-1)}{2n} + \frac{n-1}{n} \\&= \frac{n+1}{2} - \frac{1}{n}\end{aligned}$$

Calculul numărului mediu de comparații, recursiv 1/2

Dacă $n > 1$, primul element este totdeauna testat (i.e. probabilitatea de testare a lui este 1). Reamintim că probabilitatea ca algoritmul să se oprească după primul test este $\frac{1}{n}$, rezultă că restul listei este procesată cu probabilitatea $1 - \frac{1}{n} = \frac{n-1}{n}$.

Y - variabila aleatoare care dă numărul mediu de comparații $M(C(n-1))$ pentru apelul recursiv dacă prima piuliță nu e potrivită, 0 altfel.

Avem relația:

$$C(1) = 0, C(n) = 1 + Y \text{ pentru } n > 0$$

de unde rezultă

$$M(C(n)) = 1 + M(Y) = 1 + \frac{n-1}{n} M(C(n-1)) \text{ pentru } n > 1.$$

Calculul numărului mediu de comparații, recursiv 2/2

Notăm $c(n) = nM(C(n))$.

Rezultă: $c(1) = 0$, $c(n) = n + c(n-1)$ pentru $n > 1$.

Avem: $c(n) = \sum_{i=2}^n i = \frac{n(n+1)}{2} - 1$.

care implică $M(C(n)) = \frac{c(n)}{n} = \frac{n+1}{2} - \frac{1}{n}$.

Cazul general, soluția 1 (naivă)

```
NutsAndBolts (NL, BL)
{
    M = emptyList;
    forall B in BL {
        n = NL.size();
        forall N in NL
            if (cmp(N, B) == 0) M.pushBack([B, N]);
    }
    return M;
}
```

Numărul de comparații în cazul cel mai nefavorabil

=

Numărul mediu de comparații

=

$$O(n^2)$$

Soluția de mai sus nu ține seama de semnificațiile valorilor -1 și 1 întoarse de `cmp`.

Cazul general, soluția 2: ideea

Soluția e de natură recursivă.

Pasul curent:

- 1 se alege un șurub ca pivot și se compară cu toate piulițele;
- 2 se ia piulița care se potrivește cu șurubul pivot și se compară cu toate celelalte șuruburi;
- 3 după cele $2(n - 1)$ comparații, se împarte fiecare dintre cele două mulțimi (liste) în două: mulțimea șuruburilor/piulițelor mai mici, respectiv mai mari, decât piulița/șurubul pivot;

Apeluri recursive:

- un apel pentru potrivirea piulițelor mai mici cu șuruburi mai mici
- un apel pentru potrivirea piulițelor mai mari cu șuruburi mai mari

Exercițiu. Să se scrie în Alk și să se testeze algoritmul de mai sus.

Analiza în cazul cel mai nefavorabil

$$C(0) = 0$$

$$\begin{aligned} C(n) &= 2n - 1 + \max_{k=1, \dots, n} (C(k-1) + C(n-k)) \\ &= 2n - 1 + C(n-1) \end{aligned}$$

Rezolvând recurența, obținem

$$C(n) = \sum_{i=1}^n (2i - 1) = O(n^2)$$

Numărul mediu de comparații 1/3

Presupunem că șurubul pivot este ales aleatoriu uniform.

Fie B_i al i -lea șurub în lista sortată. Similar N_j .

X_{ij} variabila aleatorie (**indicatorul de variabilă**) care întoarce 1 dacă B_i și N_j sunt comparate, 0 altfel.

Presupunem $i < j$. B_i și N_j sunt comparate ddacă primul pivot din B_i, \dots, B_j este B_i sau B_j (de ce?).

Rezultă $M(X_{ij}) = Pr(X_{ij} = 1) = \frac{2}{j+1-i}, i < j$.

Relația simetrică este obținută la fel: $M(X_{ij}) = \frac{2}{i+1-j}, i > j$.

Dacă $i = j$, $M(X_{ij}) = 1$ (deși aceste comparații ar putea fi evitate).

Numărul mediu de comparații 2/3

Deoarece $C(n) = \sum_{i=1}^n \sum_{j=1}^n X_{ij}$, rezultă

$$\begin{aligned} M(C(n)) &= M\left(\sum_{i=1}^n \sum_{j=1}^n X_{ij}\right) \\ &= \sum_{i=1}^n \sum_{j=1}^n M(X_{ij}) \\ &= \sum_{i=1}^n M(X_{ii}) + 2 \sum_{i=1}^n \sum_{j=i+1}^n M(X_{ij}) \\ &= n + 4 \sum_{i=1}^n \sum_{j=i+1}^n \frac{1}{j+1-i} \\ &= n + 4(nH_n - 2n + H_n) \end{aligned}$$

Numărul mediu de comparații 3/3

S-a utilizat:

$$\begin{aligned}
 \sum_{i=1}^n \sum_{j=i+1}^n \frac{1}{j+1-i} &= \sum_{i=1}^n \sum_{k=2}^{n+1-i} \frac{1}{k} \\
 &= \sum_{k=2}^n \sum_{i=1}^{n+1-k} \frac{1}{k} \\
 &= \sum_{k=2}^n \frac{n+1-k}{k} \\
 &= (n+1) \sum_{k=2}^n \frac{1}{k} - \sum_{k=2}^n 1 \\
 &= (n+1)(H_n - 1) - (n-1)
 \end{aligned}$$

Deoarece $H_n = \sum_{k=1}^n \frac{1}{k} = \Theta(\log n)$, rezultă

$$M(C(n)) = \Theta(n \log n)$$

Numărul mediu de comparații, recursiv 1/3

Presupunem că pivot este ales aleatoriu uniform și că un șurub B este egal probabil al k -lea cel mai mic element în listă, i.e. B apare pe poziția k în lista sortată cu probabilitatea $\frac{1}{n}$.

Fie Y_k variabila aleatoare care întoarce numărul mediu de comparații dat de apelurile recursive dacă pivotul este al k -lea element, 0 altfel:

$$Y_k = \begin{cases} M(C(k-1)) + M(C(n-k)) & \text{dacă pivotul este } B_k \\ 0 & \text{altfel} \end{cases}$$

Avem

$$C(n) = 2n - 1 + \sum_{k=1}^n Y_k$$

$$M(Y_k) = \frac{1}{n}(M(C(k-1)) + M(C(n-k))).$$

Numărul mediu de comparații, recursiv 2/3

$$\begin{aligned}
 M(C(n)) &= M(2n - 1 + \sum_{k=1}^n Y_k) \\
 &= 2n - 1 + \sum_{k=1}^n M(Y_k) \\
 &= 2n - 1 + \frac{1}{n} \sum_{k=1}^n (M(C(k-1)) + M(C(n-k))) \\
 &= 2n - 1 + \frac{2}{n} \sum_{k=0}^{n-1} M(C(k))
 \end{aligned}$$

de unde obținem

$$nC(n) = n(2n - 1) + 2 \sum_{k=0}^{n-1} M(C(k))$$

Numărul mediu de comparații, recursiv 3/3

$$\frac{C(n)}{n+1} = \frac{C(n-1)}{n} + \frac{4}{n+1} - \frac{3}{n(n+1)}$$

$$\frac{C(n-1)}{n} = \frac{C(n-2)}{n-1} + \frac{4}{n} - \frac{3}{(n-1)n}$$

...

$$\frac{C(2)}{3} = \frac{C(1)}{2} + \frac{4}{2} - \frac{3}{1(2)}$$

de unde obținem

$$\frac{C(n)}{n+1} = \frac{C(1)}{2} + \Theta(\log n) - \Theta(1)$$

care implică

$$C(n) = \Theta(n \log n)$$

Reducerea la sortare

- două șuruburi se pot compara indirect prin intermediul piulițelor (cum?)
- la fel, două piulițe se pot compara indirect prin intermediul șuruburilor (cum?)
- se definesc funcții de comparare (care este timpul pentru comparare?)
- se apelează un algoritm de sortare pentru ordonarea crescătoare a șuruburilor și, respectiv, piulițelor cu funcțiile de comparare corespunzătoare
- se întorc perechile ($BL[i]$, $NL[i]$) din listele sortate

Reducerea sortării la NutsAndBolts

- se duplică lista de sortat
- o copie joacă rolul șuruburilor, cealaltă a piulițelor
- se apelează algoritmul NutsAndBolts
- se "contopește" lista sortată de perechi întoarsă de NutsAndBolts într-o listă sortată a elementelor inițiale

Plan

- 1 Reamintire cursul precedent
- 2 Timpul mediu pentru algoritmi determiniști
- 3 Studii de caz**
 - Prima apariție într-o listă
 - Quicksort
 - Șuruburi și piulițe (Nuts and Bolts)
 - **Treaps**

Definiție

- combină structura de arbore binar de căutare cu cea de max-heap
- structura de arbore binar de căutare se face pe baza unei chei `key` asociate fiecărei valori
- structura de max-heap se face pe baza unei priorități `priority` asociate fiecărei valori (diferită de cheie)
- operația de căutare se face ca în arborii binari de căutare

Exemplu pe tablă.

Raimund Seidel and Cecilia R. Aragon. Randomized search trees. *Algorithmica*, 16(4/5):464-497, 1996. Available at <http://people.ischool.berkeley.edu/~aragon/pubs/rst96.pdf>.

Operația de inserare

- se inserează elementul într-un nod frunză ca la arborii de căutare
- restaurare proprietate max-heap: prin operații de rotație se merge înapoi spre rădăcină până când se întâlnește un nod cu prioritate mai mare sau egală (ca la max-heap)

Exemplu pe tablă.

Operația de ștergere

- este inversă operației de inserare
- prin operații de rotație se merge spre frontieră (interschimbând de fiecare dată cu nodul copil cu prioritate mai mare)

Numărul mediu de comparații pentru căutare 1/3

- presupunem că valorile cheilor sunt $1, 2, \dots, n$
- indicator de variabilă: A_{ij} este variabila aleatorie care întoarce 1 ddacă i este un predecesor al lui j
- i va avea prioritatea maximă pentru toate cheile din intervalul $[\min(i, j) .. \max(i, j)]$
- rezultă $M(A_{ij}) = \frac{1}{|i - j| + 1}$
- numărul de operații pentru un nod j (căutat/inserat/șters) este proporțional cu adâncimea în arbore a lui j
- definim $depth(j) = \sum_i A_{ij} - 1$ (consideră că rădăcina are adâncimea 0)

Numărul mediu de comparații pentru căutare 2/3

$$\begin{aligned}M(\text{depth}(j)) &= M\left(\sum_i A_{ij} - 1\right) \\&= \sum_i M(A_{ij}) - 1 \\&= \sum_i \left(\frac{1}{|i-j|+1}\right) - 1 \\&= \sum_{i=1}^j \left(\frac{1}{|i-j|+1}\right) + \sum_{i=j+1}^n \left(\frac{1}{|i-j|+1}\right) - 1 \\&= \sum_{k=1}^j \frac{1}{k} + \sum_{k=2}^{n-j+1} \frac{1}{k} - 1 \\&= H_j + H_{n-j+1} - 2\end{aligned}$$

Numărul mediu de comparații pentru căutare 3/3

$\max_j \text{depth}(j)$ este atins pentru $j = \frac{n+1}{2}$, care este egală cu

$$2H_{\frac{n+1}{2}} - 2 = 2 \ln n + O(1)$$

Aceasta ne dă o margine superioară pentru numărul mediu de comparații.

Un treap aleatoriu (randomized) este un treap un obținut prin generarea aleatorie uniformă a priorităților (numere reale $\hat{[0, 1]}$) la inserare;

Dacă cheile sunt inserate în ordinea descrescătoare a priorităților, atunci rezultatul este un treap.

Analiza de mai sus este valabilă și pentru arbori binari aleatori (construiți cu inserții aleatorii, fără priorități).

Quicksort din nou

Se consideră următorul algoritm de sortare:

T = un arbore binar de căutare vid
inserează cheile în T în ordine aleatorie
returnează secvența inordine a lui T

Care este legătura cu Quicksort?

Se obține aceeași complexitate medie?