Complexități

I. $\Theta(g) = \{ f \mid \exists\, c_1, c_2 > 0 \ \exists\, n_0 \text{ a.î. } 0 \le c_1 g(n) \le f(n) \le c_2 g(n), \forall n \ge n_0 \}$

$O(g) = \{ f \mid \exists\, c > 0 \ \exists\, n_0 \text{ a.î. } 0 \le f(n) \le c \cdot g(n), \forall n \ge n_0 \}$

$\Omega(g) = \{ f \mid \exists\, c > 0 \ \exists\, n_0 \text{ a.î. } 0 \le c \cdot g(n) \le f(n), \forall n \ge n_0 \}$

II. $f \in \Theta(g) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}_{>0}$

$f \in O(g) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$

$f \in \Omega(g) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$

III. $f \in \Theta(g) \Leftrightarrow f = g$

$f \in O(g) \Leftrightarrow f \le g$

$f \in \Omega(g) \Leftrightarrow f \ge g$

Comportarea asimptotică a unei funcții (termenul dominant)

- ptr a compara 2 fct, ne interesează comportamentul lor pentru un rang foarte ma ($n \to \infty$)
- când comparăm 2 fct, comparăm termenii lor dominanți
- pas 1: eliminăm din termenii unei funcții pe toți cei care cresc încet și îl păstrăm doar pe cel care crește cel mai repede, când $n \to \infty$
- pas 2: eliminăm constantele din fața termenului

Ex: $f = 2n^2 + 3n + 1$

$g = 1000n + 500$

Pas 1: f: când n este foarte mare, $2n^2$ crește mai repede ca $3n$ și ca $1 \Rightarrow$ eliminăm $3n+1 \Rightarrow f = 2n^2$

g: $1000n$ crește mai repede ca $500 \Rightarrow g = 1000n$

Pas2: f: eliminăm constanta $2 \Rightarrow f = n^2$

g: ——— " ——— $1000 \Rightarrow g = n$

Comparăm $f = n^2$ cu $g = n$, atunci când n este foarte mare $\Rightarrow n^2 > n \Rightarrow f > g$

$\Rightarrow f \ge g \Rightarrow f \in \Omega(g)$ și $g \in O(f)$

$\Rightarrow 2n^2 + 3n + 1 \in \Omega(1000n + 50)$ și $1000n + 50 \in O(2n^2 + 3n + 1)$

! În cazul fct polinomiale, termenul dominant este n la cea mai mare putere nenulă

În cazul logaritmilor, baza nu contează, fiind o constantă ($\log_2 n$ și $\log_{100} n$ au ambele termenul dominant „$\log n$")

$\log n$ crește mai încet decât n^a, ptr orice a pozitiv

n^b crește mai încet decât a^n, $\forall a > 1, \forall b$

$1 < \log n < n < n \log n < n^2 < n^2 \log n < n^3 < \ldots < 2^n$

Insertion Sort

$C_{min} = n-1$

$C_{max} = \frac{n(n-1)}{2}$

$C_{mediu} = \frac{1}{2}(2+3+\dots+n) = \frac{1}{2}C_{max} = \frac{n(n-1)}{4}$

$M_{i\,min} = C_i + 1$

$M_{min} = 2(n-1)$

$M_{max} = \frac{n(n+1)}{2} - 1$

$M_{mediu} = \frac{(n-1)\cdot n}{2} + 2(n-1)$

Best case: n

Average: n^2

Worst: n^2

Memorie: 1

Stabilitate: DA

$T(n) = O(n^2) = \Omega(n)$] [$\Theta(n^2)$ - worst case]

După i pași iterativi, primele $i+1$ elemente sunt ordonate crescător

Selection Sort

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

$M_{min} = 3(n-1)$

$M_{max} = \frac{n(n-1)}{2} + 3(n-1)$

$M_{mediu} = n(\ln \gamma + e)$

$T_{comp}(n) = \Theta(n^2)$

$T_{mut}(n) = \Omega(n) = O(n^2)$

Best = Average = Worst = n^2

Memorie: 1

Stabilitate: NU

După i pași iterativi, sunt plasate pe pozițiile finale ~~primele~~ cele mai mici i elemente

Bubble Sort

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

$\text{interschimbări}_{min} = 0$

$\text{interschimbări}_{max} = \frac{n(n-1)}{2}$

$\text{interschimbări}_{mediu} = \frac{n(n-1)}{4}$

$T_{comp}(n) = \Theta(n^2)$

$T_{inf}(n) = \Omega(1) = O(n^2)$

Best case = n (optimizat)

Average = Worst = n^2

Memorie : 1

Stabilitate: DA

După i pași iterativi, sunt plasate pe pozițiile finale cele mai mici i elemente

Teorema Master

$T(n) = a \cdot T(\frac{n}{b}) + O(n^d)$

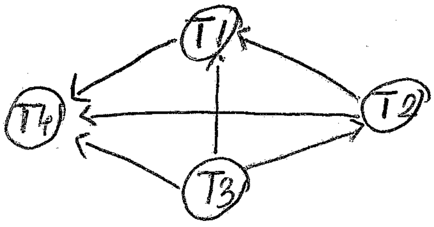
$$T(n) = \begin{cases} O(n^d \log n) & , a = b^d \\ O(n^d) & , a < b^d \\ O(n^{\log_b a}) & , a > b^d \end{cases}$$

Ex: T1: $8T(\frac{n}{2}) + 7n^3 \Rightarrow O(n^3 \log n)$

T2: $7T(\frac{n}{3}) + n^2 \Rightarrow O(n^2)$

T3: $2T(\frac{n}{2}) + n \Rightarrow O(n \log n)$

T4: $16T(\frac{n}{2}) + 2n^3 \Rightarrow \Theta(n^{\log_2 16})$



Desenați un graf orientat, având nodurile T1,T2,T3,T4 și muchii de la fiecare nod la no
de complexitate mai mare

Stive și cozi (LIFO, FIFO)

Ex: C1, C2 - cozi ; S - stivă

PUSH(X) - introduc X în C1
POP1 - scoate un element din C1 și îl introduce în C2
POP2 - scoate un element din C2 și îl introduce în S
POP3 - scoate un elemente din S

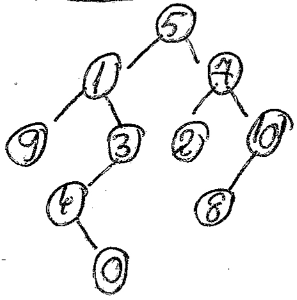
PUSH(2), PUSH(3), POP1, POP2, PUSH(5), POP1, POP(1), POP3, PUSH(7), PUSH(9), POP(2), POP(1), POP2, PUSH(8), POP2, POP 1

C1: ~~2~~ ~~3~~ ~~5~~ ~~7~~ ~~9~~ 8

C2: ~~2~~ ~~3~~ ~~5~~ ~~7~~ 9

S:
7
~~5~~
~~2~~ 3

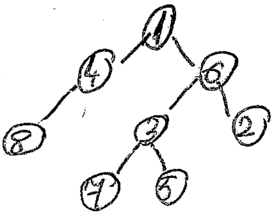
Arbori binari



RSD: ⑤ ; 5 ①⑦ ; 5 1 ⑨ ③ 7 ② ⑩ ; 5 1 9 3 ④ 7 2 10 ⑧
5 1 9 3 4 ⓪ 7 2 10 8 ; 5 1 9 3 4 0 7 2 10 8

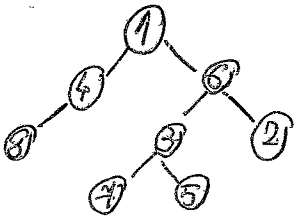
SRD: ⑤ ; ① 5 ⑦ ; ⑨ 1 ③ 5 ② 7 ⑩ ; 9 1 ④ 3 5 2 7 ⑧ 10 ;
9 1 4 ⓪ 3 5 2 7 8 10 ; 9 1 4 0 3 5 2 7 8 10

SDR: ⑤ ; ① ⑦ 5 ; ⑨ ③ 1 ② ⑩ 7 5 ; 9 ④ 3 1 2 ⑧ 10 7 5 ;
9 ⓪ 4 3 1 2 8 10 7 5 ; 9 0 4 3 1 2 8 10 7 5

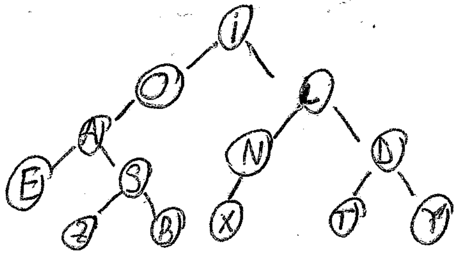


RSD: 1 4 8 6 3 7 5 2
SRD: 8 4 1 7 3 5 6 2
SDR: 8 4 7 5 3 2 6 1

RSD: $\boxed{1}$ 4 8 | $\boxed{6}$ 3 7 5 | 2
SRD: 8 4 $\boxed{1}$ 7 3 5 $\boxed{6}$ 2

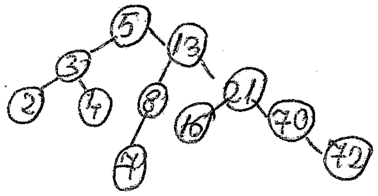


RSD: I O A E S Z B L N X D T Y
SDR: E Z B S A O X N T Y D L I



Arbori binari de căutare (x > Left(x), x < Right(x))

Ex: Constr ABC: ~5, 13, 21, 70, 72, 3, 2, 16, 4, 8, 7

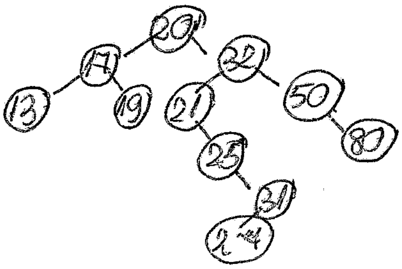


Ştergere 70: are un singur fiu => unicul fiu se leagă de tatăl lui 70:

- 20, 32, 17, 21, 19, 50, 80, 25, 31, 13, 27

Ştergere 50

Ştergere 32: are 2 fii: căutam predec în inordine (31), intersch. şi ştergem vechiul nod 31

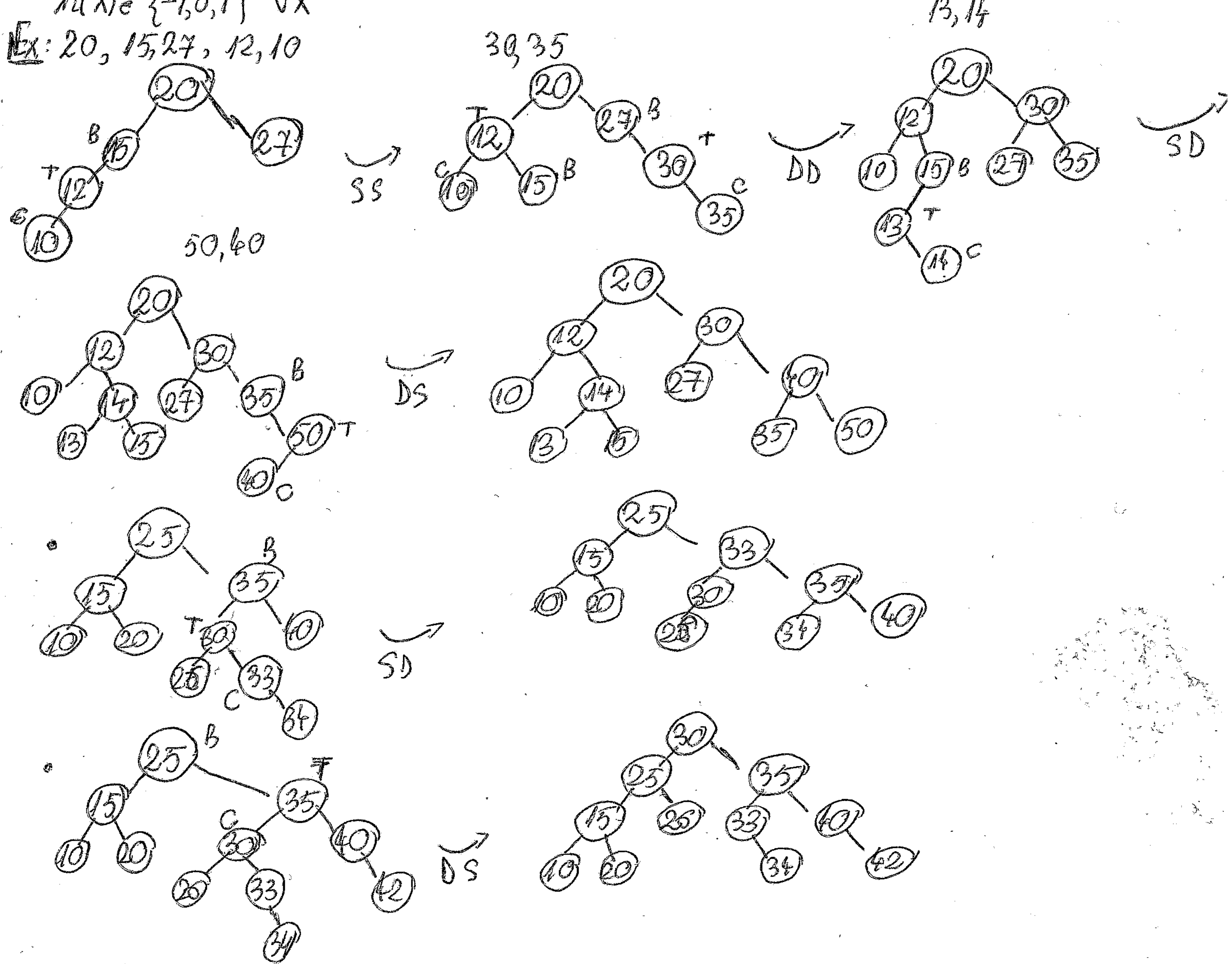


Arbori AVL

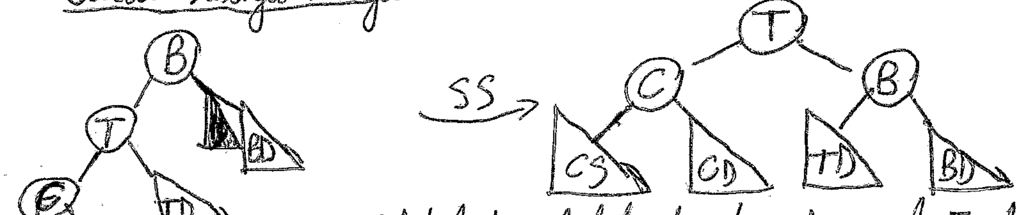
Factor de echilibru = h(subarbore drept) - h(subarbore stâng)

h(X) ∈ {-1, 0, 1} ∀X

Ex: 20, 15, 27, 12, 10

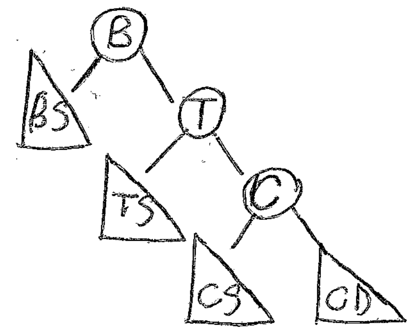


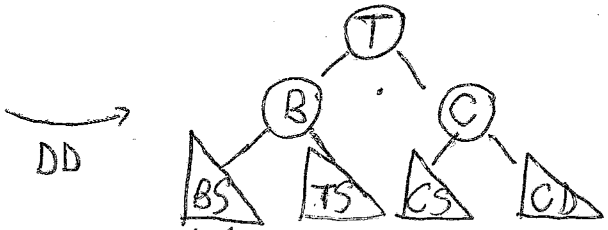
Cazul Stânga-Stânga



•Nodul dezechilibrat este B(bunicul). Fiul său stâng este T(tatăl), iar fiul stâng al tatălui este C(copilul). După rotația SS, rădăcina subarborelui va deveni T, având ca fiu stâng pe C, iar ca fiu drept pe B.
C își va păstra fiii (CS și CD). B își păstrează fiul drept(BD), iar fiul său stâng va deveni fostul fiu drept al tatălui(TD).

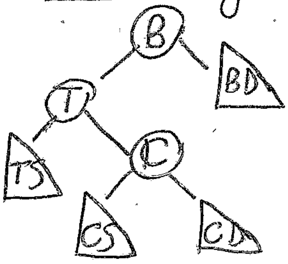
Cazul dreapta-dreapta

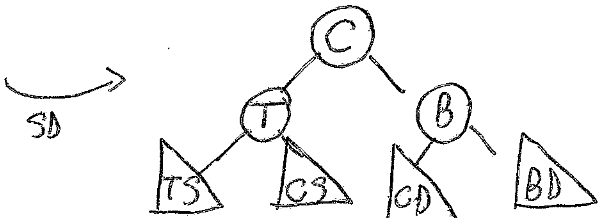




După rotația DD, rădăcina va deveni T, având ca fiu stâng B, și fiu drept C. C își păstrează ambii fii, B își păstrează fiul stâng, iar ca fiu drept va avea fostul fiu stâng al tatălui (TS).

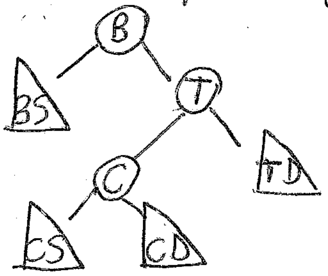
Cazul stânga-dreapta

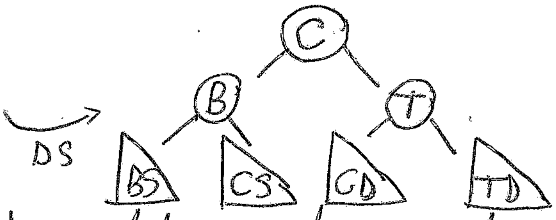




După rotația SD, rădăcina va deveni C, având ca fiu stâng T și fiu drept B. T își va păstra fiul stâng (TS), iar ca fiu drept va avea fostul fiu stâng al lui C(CS). B își va păstra fiul drept (BD), iar ca fiu stâng va avea fostul fiu drept al lui C (CD).

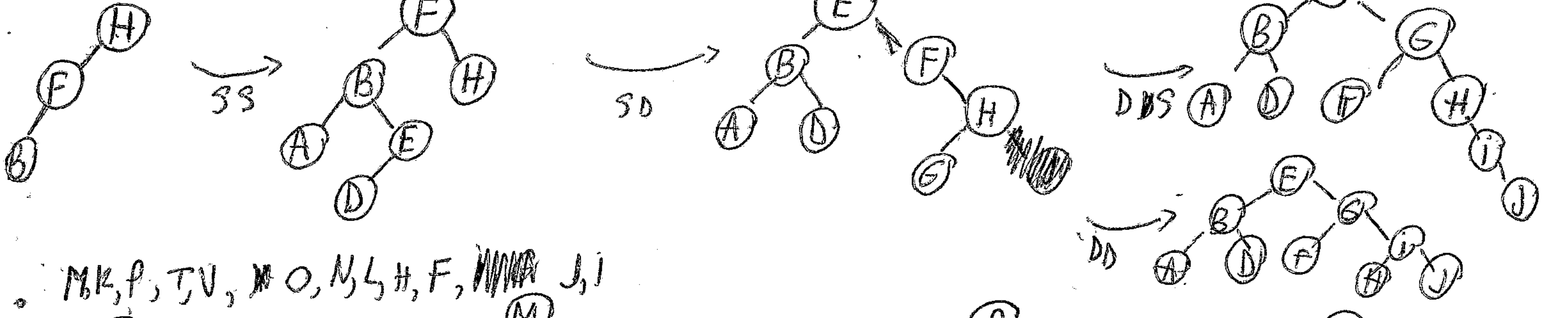
Cazul dreapta-stânga



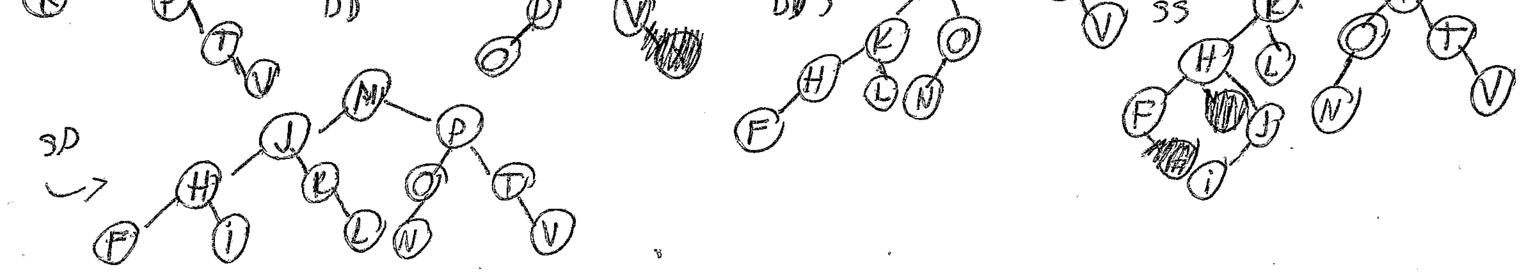


După rotația DS, rădăcina va deveni C, având ca fiu stâng B și fiu drept T. B își va păstra fiul stâng (BS), iar ca fiu drept va avea fostul fiu stâng al lui C(CS). T își va păstra fiul drept (TD), iar ca fiu stâng va avea fostul fiu drept al lui C (CD).

Ex: H, F, B, A, E, D, G, I, J



M, K, P, T, V, O, N, L, H, F, J, I

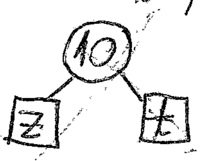


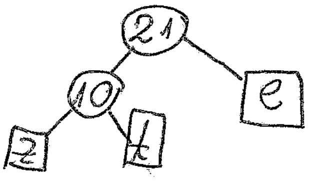
Coduri Huffman

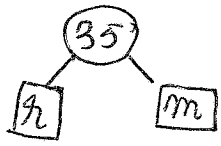
Se dau următoarele litere şi frecvenţele lor de apariţie într-un text. Să se construiască un arbore binar optim de codificare/decodificare, în care elementele cu frecvenţa mare să fie mai uşor de accesat.

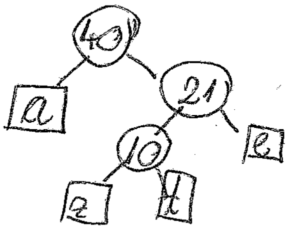
i	a	m	~~r~~	~~e~~	~~t~~	~~z~~
25	19	18	~~17~~	~~11~~	~~6~~	~~4~~

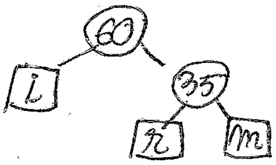
Iniţial, fiecare literă reprezintă un arbore. La fiecare pas, se iau arborii cu cele mai mici ponderi şi se unesc într-un nou arbore, care are ca rădăcină suma ponderilor lor, iar ca fiu stâng arborele de pondere mai mică.

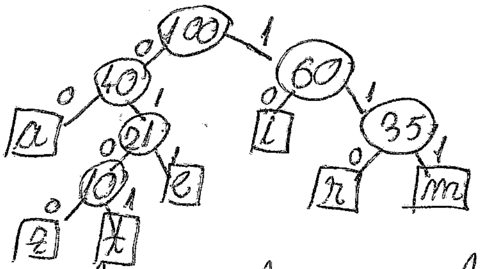












Codificare: se codifică cu 0 fiecare muchie spre un fiu stâng, şi cu 1 spre un fiu drept. Codul fiecărei litere este drumul de la rădăcină, până la ea.

a: 00 z: 0100 t: 0101 e: 011 i: 10 r: 110 m: 111

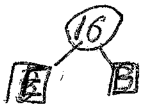
Ex: Codificaţi următoarele cuvinte: mama, mare, tare, marţi

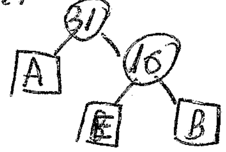
mama: 1110011100 ; mare: 11100110011 ; tare: 010100110011 ; marţi: 1110011001011

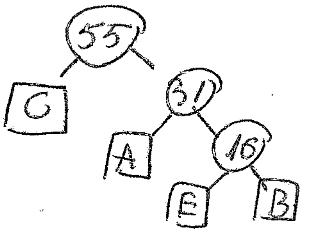
Decodificare: se porneşte de la rădăcină şi se citesc cifre din cuvânt, mergând spre stânga sau spre dreapta, în funcţie de 0 sau 1, până se ajunge la o frunză. Se porneşte din nou de la rădăcină, repetându-se procedeul cu cifrele rămase. După ultima cifră din cod, se poate ajunge la un nod intern, caz în care cuvântul nu este valid.

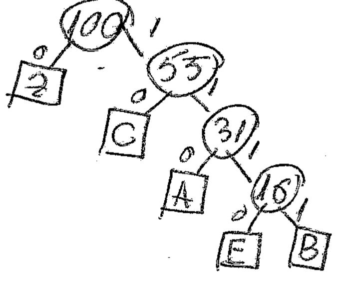
Ex: Decodificaţi:
111000100000110.011 - mazare
11100110010110011 - marţie
11100010101111101001 - materi 01 - invalid

Z=45, B=11, C=24, A=~~10~~15, E=5









Z: 0 , C: 10 , A: 110, E: 1110, B: 1111

Codif:
ZACE: 0110101110
BEC: 1111111010
ACE: 111 101110

Decod:
100110111010 : CZAEC
111001100110 : EZAZA
1111100111 : BAZ111 - invalid

Arbori binari stricți

- A.B în care fiecare nod are ori 2 fii (nod intern), ori niciunul (nod extern)

P1: $N_E = N_I + 1$ (N_E - nr. noduri externe, N_I - nr. noduri interne)

P2: $L_E = L_I + 2N_I$ ($L_E = \sum_{x \in E} l(r,x)$ - suma lung. drum de la rădăcină la nodurile exter

$L_I = \sum_{x \in I} l(r,x)$ ———— 〃 ———— interne

P3: $N_E \leq 2^d$ (d - adâncime arbore) $\Rightarrow d \geq \lceil \log_2 N_E \rceil$

~~P4: Dintre toți~~ Corolar: $d \geq \lceil \log_2 N_E \rceil$

P4: Dintre toți a.b.s. cu același N_E, au $L_{E\,min}$ aceia care au frunzele repartizate pe cel mult două niveluri adiacente.

P5: $L_{E\,min} = l \lfloor \lg l \rfloor + 2(l - 2^{\lfloor \lg l \rfloor})$ l - nr. frunze

P6: $L_{E\,medie} \geq \lfloor \lg l \rfloor$

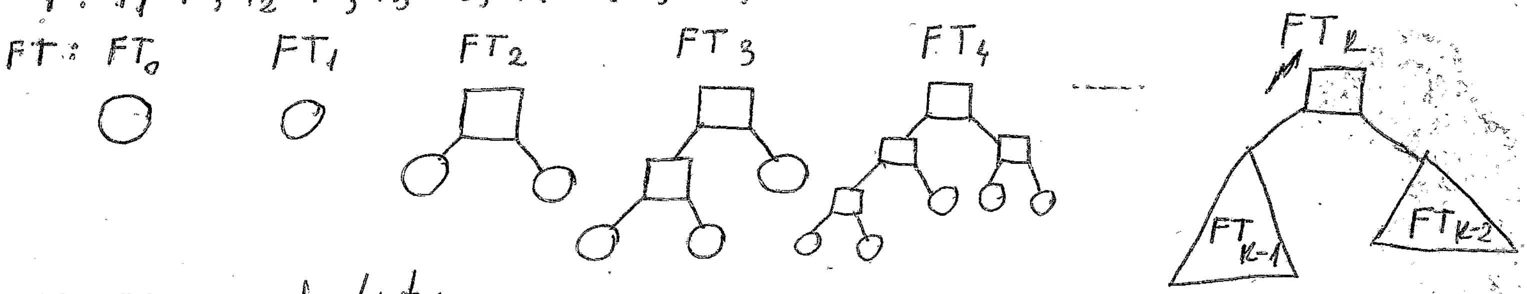
Teorema AVL: Fie T un abs, AVL, cu n noduri. Atunci $\log_2(n+1) \leq h(T) \leq 1.4404 \log_2(n+2) - 0.32$

Arbori Fibonacci:

- F_K - elementul de ordin k din șirul Fib

FT_K - arborele Fib de ordin k

F: $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, $F_4 = 3$, $F_5 = 5$, ... $F_k = F_{k-1} + F_{k-2}$



P1: FT_k e a.b. strict $\forall k \geq 0$

P2: $\forall k \geq 1$: a) $h(FT_k) = k-1$

b) $N_E(FT_k) = F_{k+1}$

c) $N_I(FT_k) = F_{k+1} - 1$

P3: $\forall k \geq 0$ FT_k este AVL

P4: În familia a.b.s., AVL, de înălțime h, arb. Fibonacci au un nr. minim de noduri inter

AVL - Căutare Binară

T1: Între arborii de decizie asociați unui algoritm de căutare într-o mulțime cu n chei sunt optimi (au L_E minimă) cei asociați căutării binare

T2: 1. $2^{k-1} \leq n < 2^k \Rightarrow$ o căutare cu succes necesită cel mult k comparații

2. $n = 2^k - 1 \Rightarrow$ — 〃 — fără succes necesită exact k comparații

3. $2^{k-1} \leq n < 2^k - 1 \Rightarrow$ o căutare fără succes necesită fie k-1, fie k comparații

T3: Căutarea binară minimizează nr. mediu de comparații, atât în cazul căutării cu succes, cât și în cazul căutării fără succes

C_n = nr. mediu de comparații la căutarea cu succes

C'_n = ———— 〃 ———— fără succes

Lemă: $C_n = (1 + \frac{1}{n}) C'_n - 1$

Huffman

L1: T arbore de codif. optim peste $\mathcal{E} \Leftrightarrow [w(e) < w(e') \Rightarrow h_T(e) \geq h_T(e')]$

L2: w_1, w_2 cele mai mici ponderi, coresp. lui e_1 şi e_2. Atunci exista un arbore optim în care e_1 şi e_2 sunt frați.

T1: Algoritmul lui Huffman construieşte un arbore binar de codificare optim.

Huffman

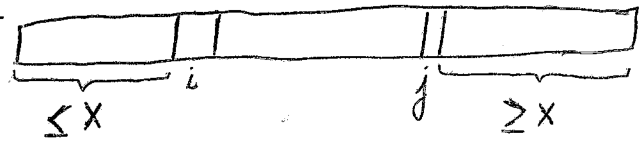
L1: T arbore de codif. optim peste $\mathcal{E} \Leftrightarrow [w(e) < w(e') \Rightarrow h_T(e) \geq h_T(e')]$

L2: w_1, w_2 cele mai mici ponderi, coresp lui e_1 și e_2. Atunci există un arbore optim în care e_1 și e_2 sunt frați.

T1: Algoritmul lui Huffman construiește un arbore binar de codificare optim.

Quick-Sort

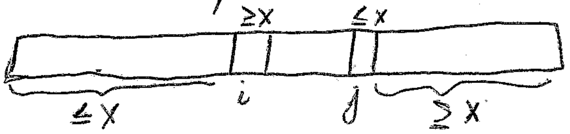
Hoare



i-ul va crește până când $a[i] \geq x$

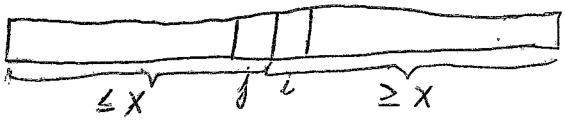
j-ul va scădea până când $a[j] \leq x$

Caz 1: În intervalul $[i, j]$ ∃ atât valori ≥ x, cât și ≤ x, iar indicele primei valori ≥ x este mai mic decât indicele ~~primei~~ ultimei valori ≤ x



⇒ Interschimbăm $a[i]$ cu $a[j]$ și ajungem din nou la config inițială ⇒ reluăm analiza

Caz 2: În $[i, j]$ ∃ atât valori ≥ x, cât și ≤ x, iar indicele primei valori ≥ x este mai mare decât indicele ultimei valori ≤ x



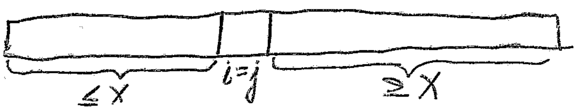
$j < i \Rightarrow$ algoritmul se oprește

În $[1, j]$ avem valori ≤ X

În $[j+1, n]$ avem valori ≥ X

⇒ apelăm QS(1, j) și QS(j+1, n)

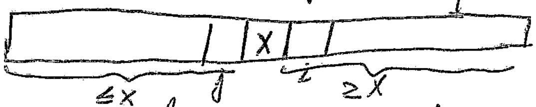
Caz 3: În $[i, j]$ ∃ atât valori ≥ x, cât și ≤ x, iar indicele primei valori ≥ x este egal cu indicele ultimei valori ≤ X



$$i = j \Leftrightarrow \left.\begin{array}{l} a[i] \geq X \\ a[j] \leq X \end{array}\right\} \Rightarrow a[i] = X$$

Intersch $a[i]$ cu $a[j]$ (niciun efect), i++, j--

⇒



$j < i$ ⇒ se termină

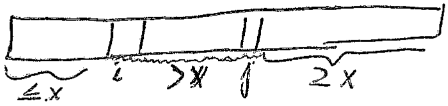
În $[1, j] \Rightarrow$ elem ≤ X

În $[j+1] \Rightarrow$ elem ≥ X

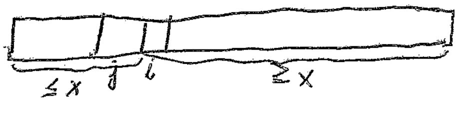
⇒ apelăm QS(1, j) și QS(j+1, n)

Caz 4: În $[i, j]$ ∃ doar valori > X.

Inițial:



⇒ devine



$j < i$ ⇒ alg. se termină

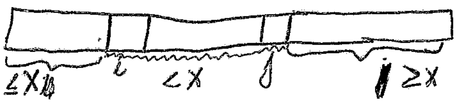
$[1, j]$ ≤ X

$[j+1, n]$ ≥ X

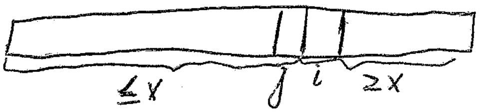
⇒ QS(1, j) și QS(j+1, n)

Caz 5: În $[i,j]$ ∃ doar valori $< x$

Initial



Devine



$j < i \Rightarrow$ alg. se termină

$[1,j] \leq x$
$[j+1,n] \geq x$ $\Rightarrow$ $QS(1,j)$ și $QS(j+1,n)$

Ex 1:

					x							
13	19	9	5	12	$\boxed{8}$	7	4	11	2	6	21	
i											j	
13	19	9	5	12	8	7	4	11	2	6	21	(Intersch 13↔6)
i										j		
6	19	9	5	12	8	7	4	11	2	13	21	(Intersch 19 ↔ 2)
	i								j			
6	2	9	5	12	8	7	4	11	19	13	21	
		i						j				
6	2	9	5	12	8	7	4	11	19	13	21	(9 ↔ 4)
		i					j					
6	2	4	5	12	8	7	9	11	19	13	21	
			i			j						
6	2	4	5	12	8	7	9	11	19	13	21	(12↔ 7)
				i		j						
6	2	4	5	7	8	12	9	11	19	13	21	($i \leq j \Rightarrow$ intersch 8 cu 8)
					i,j							
6	2	4	5	7	8	12	9	11	19	13	21	
				j		i						

$i > j \Rightarrow$ STOP $a[1,j] =$ 6 2 4 5 7 ≤ 8
$a[j+1,n] =$ 8 12 9 11 19 13 21 ≥ 8

Ex 2:

			x				
5	2	6	$\boxed{4}$	8	9	7	
i						j	
5	2	6	4	8	9	7	(5↔4)
i			j				
4	2	6	5	8	9	7	
	i	j					
4	2	6	5	8	9	7	
	j	i					

$i > j \Rightarrow$ STOP : $a[1,j] =$ 4 2 ≤ 4
$a[j+1,n] =$ 6 5 8 9 7 ≥ 4

Ex 3:

			x				
8	7	9	$\boxed{6}$	3	5	4	(8↔4)
i						j	
4	7	9	6	3	5	8	(7↔5)
	i				j		
4	5	9	6	3	7	8	(9↔3)
		i		j			
4	5	3	6	9	7	8	(6↔6)
			i j				
4	5	3	6	9	7	8	
		j		i			

$j > i \Rightarrow$ 4 5 3 ≤ 6
6 9 7 8 ≥ 6

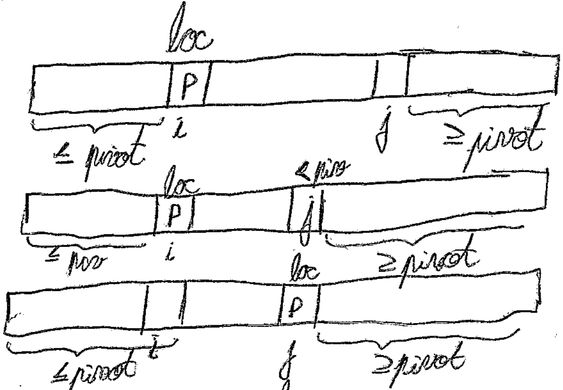
Ex 4:

			$\hat{}$				
10	25	17	$\boxed{15}$	13	6	11	
i						j	
10	25	17	15	13	6	11	(25 ↔ 11)
	i					j	
10	11	17	15	13	6	25	(17 ↔ 6)
		i			j		
10	11	6	15	13	17	25	(15 ↔ 13)
			i	j			
10	11	6	13	15	17	25	
			j	i			

$j > i \Rightarrow$ a[1, j] = 10 11 6 13
a[j+1, n] = 15 17 25

Partition 2

a)

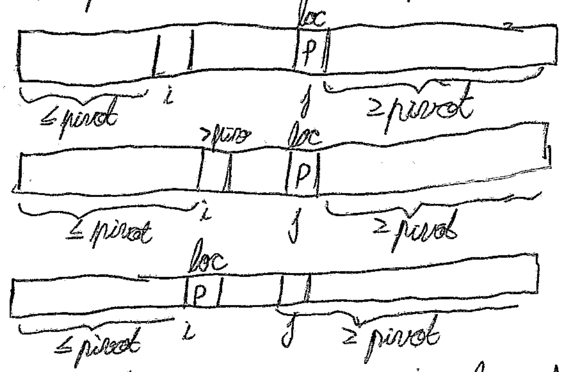


j-- până ajungem la o val. < pivot

(Intersch. $\underset{=\text{pivot}}{a[loc]}$ cu a[j], iar loc devine j)

la finalul a), . valorile din [j+1,
sunt ≥ pivot

b)



i++ până ajungem la o val > pivot

(Intersch. $\underset{\text{piv}}{a[loc]}$ cu a[i], iar loc devine i)

La finalul b), valorile din [1, i-1] sunt ≤ piv

La un moment dat, vom avea i = j = loc. Algoritmul se oprește. Avem:

[j+1, n] = [loc+1, n] avem valori ≥ pivot
[1, i-1] = [1, loc-1] avem valori ≤ pivot

⇒ a[loc] = pivot este pus pe poziția sa finală în vector
⇒ apelăm QS(1, loc-1)
QS(loc+1, n)

Ex. 1

loc							
$\boxed{5}$	3	8	4	7	1	6	
i						j	
loc							
5	3	8	4	7	1	6	(5 ↔ 1)
i					j loc		
1	3	8	4	7	5	6	
i					j loc		
1	3	8	4	7	5	6	(8 ↔ 5)
		i loc			j		
1	3	5	4	7	8	6	
		i loc			j		
1	3	5	4	7	8	6	(4 ↔ 5)
		i	j loc				
1	3	4	5	7	8	6	
		i	j				

1 3 4 5 7 8 6 (loc above 5; i j below 5)

i = j ⇒ STOP

$a[1, loc-1] \le$ pivot

$a[loc+1, n] \ge$ pivot

Ex.2

loc
$\boxed{88}$ 99 66 33 44 22   (88 ↔ 22)
i

j loc
22 99 66 33 44 88
i

j loc
22 99 66 33 44 88   (99 ↔ 88)

i loc / j
22 88 66 33 44 99

i loc / j
22 88 66 33 44 99   (88 ↔ 44)

i / j loc
22 44 66 33 88 99

i / j loc
22 44 66 33 88 99   i = j → STOP
i j

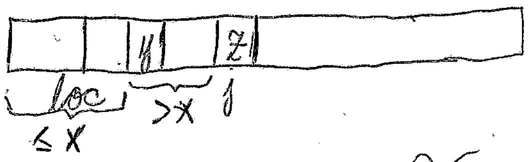
$a[1, loc-1] = 22\ 44\ 66\ 33 \le$

$a[loc+1, n] = 99 \ge 88$

Ev.3 4 7 6 3 2 5 1

1 2 3 4 6 5 7

Lomuto

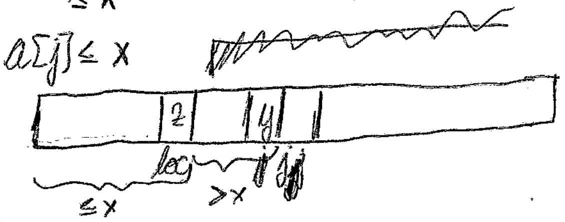


La fiecare pas: $[1, loc]$ elem $\le X$

$[loc+1, j-1]$ elem $> X$

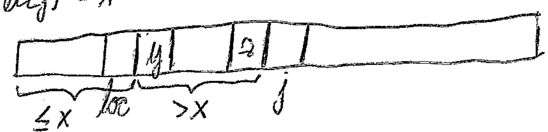
$[j, n]$ neparcurse

a) $a[j] \le X$

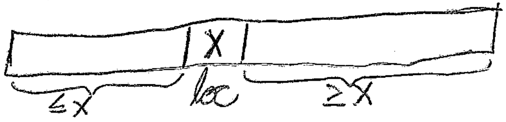


Creștem loc (indexul ultimului element $\le x$) și intersch $a[loc]$ cu $a[j]$

b) $a[j] > X$



În final:
($j = n+1$)



$[1, loc-1]$ elem $\le X$

$a[loc] = X$

$[loc+1, n]$ elem $> X$

⇒ QS(1, loc-1)

QS(loc+1, n)

Ex.1:

13 19 9 5 12 21 7 4 11 2 6 $\overset{X}{\boxed{8}}$
loc j

13 19 9 5 12 21 7 4 11 2 6 8

loc / j
13 19 9 5 12 21 7 4 11 2 6 8

loc / j
13 19 9 5 12 21 7 4 11 2 6 8

loc / j
5 19 9 13 12 21 7 4 11 2 6 8
loc

j
5 19 9 13 12 21 7 4 11 2 6 8
loc j

5	19	9	13	12	21	7	4	11	2	6	8	
loc						j						
5	7	9	13	12	21	19	4	11	2	6	8	
	loc						j					
5	7	4	13	12	21	19	9	11	2	6	8	
		loc						j				
5	7	4	13	12	21	19	9	11	2	6	8	
		loc							j			
5	7	4	2	12	21	19	9	11	13	6	8	
			loc							j		
5	7	4	2	6	21	19	9	11	13	12	8	
				loc							j	
5	7	4	2	6	8	19	9	11	13	12	21	
					loc							j

$a[1 \ldots loc-1] \leq 8$

$a[loc] = 8$

$a[loc+1 \ldots n] \geq 8$

							x	
Ex.2:	2	7	3	5	6	1	$\boxed{4}$	
loc	j							
	2	7	3	5	6	1	4	
	loc	j						
	2	7	3	5	6	1	4	
	loc		j					
	2	3	7	5	6	1	4	
		loc		j				
	2	3	7	5	6	1	4	
		loc			j			
	2	3	7	5	6	1	4	
		loc				j		
	2	3	1	5	6	7	4	
			loc				j	
	2	3	1	4	6	7	5	
				loc				j

$a[1, loc-1] \leq 4$

$a[loc] = 4$

$a[loc+1, n] \geq 4$

Ex 3: 2 4 3 5 6 1 7

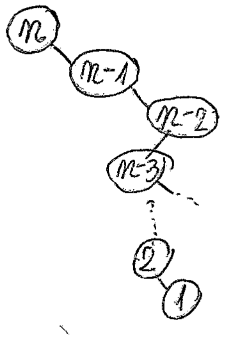
Complexitate QS:

$$T(n) = \underbrace{T(\overset{q-1}{\cancel{q}}) + T(n-q)}_{\text{Divide}} + \underbrace{\Theta(n)}_{\text{Partition}} \quad , \text{unde } q \in \{1, \ldots, n\}$$

$\forall\, q \in \{1, \ldots, n\}$ are o probab. de $\frac{1}{n}$ (prob. egală ca pivotul să fie pe poziția q)

$$T(n) = \frac{1}{n} \sum_{q=1}^{n} \left(T(q-1) + T(n-q) \right) + \Theta(n)$$

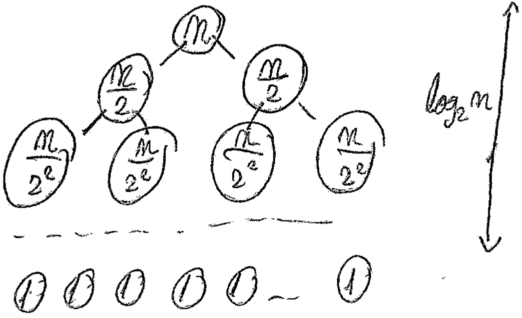
Worst Case: Subvectori de lungime dezechilibrată: $T(0)$ și $T(n-1)$ (pivotul este mereu ori minimu ori maximul din vector)



n
$n-1$
$n-2$
$n-3$
$\vdots$
2

$$T(n) = 2 + 3 + \ldots + n = \frac{(n+2)(n-1)}{2} = \frac{1}{2} n^2 + \frac{1}{2} n - 1 \Rightarrow T(n) \in O(n^2)$$

Best Case: Subvectori de aceeași lungime (sau diferență 1)



n

$\frac{n}{2} + \frac{n}{2} = n$

$4 \cdot \frac{n}{4} = n$

$\vdots$

$\frac{n}{2} \cdot 2 = n$

$\Omega(n \log n)$

Split constant: La fiecare partiție, raportul dintre subvectorul stâng și cel drept e același

$\Rightarrow T(n): n \log n$

Average Case: $n \log n$

Heap-uri

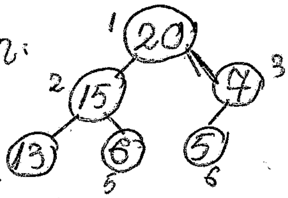
Def: Un (max) heap este un arbore binar cu urm. propr.:

1) este complet (toate nivelurile, cu excepția ultimului, sunt complete; ultimul nivel poate să nu fie complet, caz în care nodurile sunt așezate „cel mai la stânga")

2) fiecare nod este mai mare sau egal decât ambii fii

Un min-heap se definește asemănător: 1) — " —
2) ∀ nod ≤ fii

Heap:



reprezentat de vectorul

20	15	7	13	6	5
1	2	3	4	5	6

Consecințe:

1) rădăcina are valoarea maximă

2) ptr nodul i: fiul stâng $= 2i$
fiul drept $= 2i+1$
tatăl $= \left\lfloor \frac{i}{2} \right\rfloor$

3) un heap cu n elemente are $\lfloor \log_2 n \rfloor + 1$ niveluri

4) înălțimea unui nod = distanța maximă de la nod la o frunză
înălțimea unui heap = înălțimea rădăcinii
Înălțimea heap-ului $= \lfloor \log_2 n \rfloor$

5) $i < j \Rightarrow$ tată$(i) \leq$ tată(j)
Ultimul nod care nu e frunză este tatăl ultimului nod din heap
$n \Rightarrow$ tată$(n) = \left\lfloor \frac{n}{2} \right\rfloor$

$\Rightarrow$ heap cu n elemente: $\underbrace{\boxed{1} \; \ldots\ldots \; \boxed{\tfrac{n}{2}}}_{\text{noduri interne}} \; \underbrace{\boxed{\tfrac{n}{2}+1} \; \ldots\ldots \; \boxed{n}}_{\text{frunze}}$

Ex1: Verificați dacă arborii compleți reprezentați de urm. vectori sunt heap-uri:

a) 16, 14, 10, 8, 7, 9, 3, 2, 4, 1 (MAX) b) 2, 3, 4, 8, 9, 12, 14, 10, 11 (MIN)

c) 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 (NU)

Extragerea maximului / Ștergerea unui nod

Maximul se află pe poziția 1. Ptr. a-l extrage (elimina) din heap, îl interschimbăm cu ultimul element și, fiind frunză, îl eliminăm. Noul arbore are $n-1$ noduri, dar rădăcina nu mai respectă propr 2). Trebuie să o coborâm în arbore, interschimbând-o pe rând cu fiul mai mare, până când devine frunză sau este mai mare decât ambii fii.

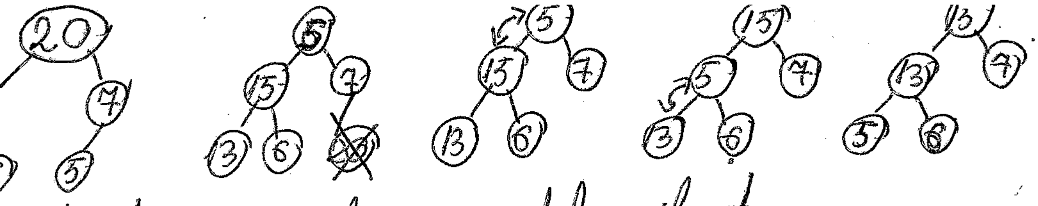
MaxHeapify(a[], n, i) // coboară nodul i în heap

Nr maxim de interschimbări (apare când nodul i va deveni frunză) = înălțimea nodului i

$T(i) = h_i \Rightarrow O(h_i)$

În particular, $T(1) = h_1 =$ înălțimea rădăcinii $= \lfloor \log_2 n \rfloor \Rightarrow O(\log n)$

Delete 20:



Ex.2: Ștergeți din urm. heap-uri nodul specificat:

a) 16, 14, 10, 8, 7, 9, 3, 2, 4, 1 (nodul 16)

b) 23, 17, 14, 6, 13, 10, 1, 5, 4, 12 (nodul 17)

Inserarea unui nod

Pentru a adăuga un nod într-un heap, îl plasăm inițial pe cea mai din stânga poziție liberă de pe ultimul nivel (sau pe prima dintr-un nivel nou, dacă ultimul era complet). Îl intersc băm pe rând cu tatăl lui, până când ajunge rădăcină sau este mai mic decât tatăl lui.

Insert(a[], n, i) // urcă nodul i în heap

$T(i)$ = nivelul pe care se află nodul i

$T(n) \in O(\log n)$

Ex.3: Inserați în heap-ul 20, 15, 7, 13, 6, 5 valoarea 10.

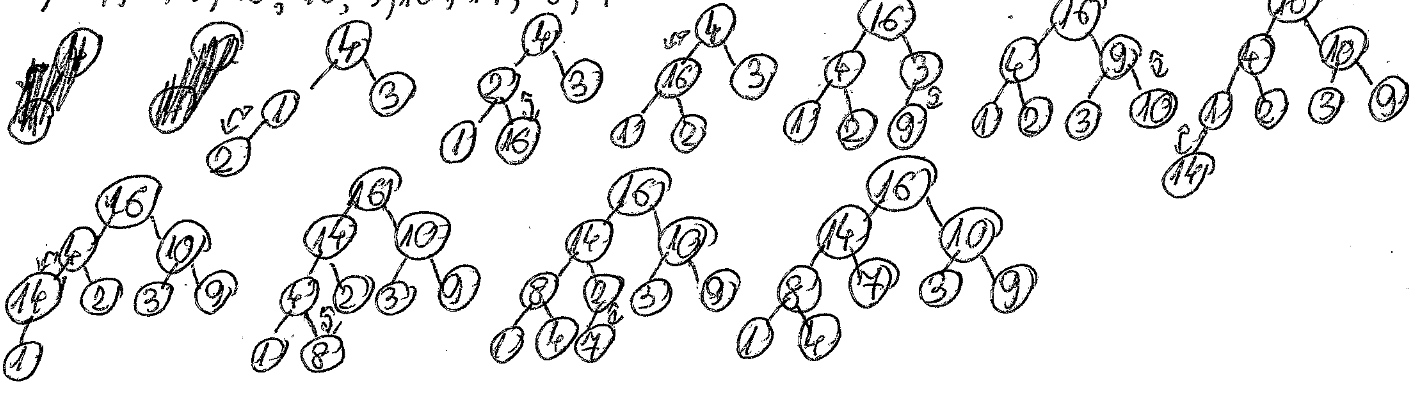
Crearea unui heap dintr-un vector

Metoda 1: Prin inserări repetate

Inițial, heap-ul este vid. La fiecare pas i, inserăm nodul i într-un heap de dim. i-

Ex.4: Creați, prin inserare, un heap din urm. vectori:

a) 4, 1, 3, 2, 16, 9, 10, 14, 8, 7



b) 5, 3, 17, 10, 84, 19, 6, 22, 9

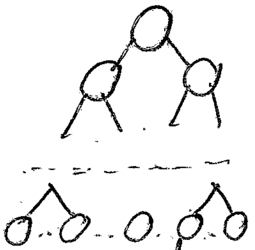
Complexitate: Se aplică de n ori Insert, de compl $\log n \Rightarrow O(n \log n)$

Metoda 2: Prin coborâri repetate, în ordine inversă

Inițial, considerăm vectorul ca fiind un arbore complet și aplicăm funcția MaxHeapify de la ultimul până la primul nod. Observăm că frunzele nu au unde să coboare, a că aplicăm doar ptr nodurile interne.

Complexitatea MaxHeapify este $O(\log n)$, dar această limită superioară se obține d ptr rădăcină, în rest se obțin valori mici. Deci complexitatea totală e mai bună ca $n \log n$.

Considerăm un arbore complet, cu ultimul nivel complet.



~~Nivel~~	Înălțime	Nr noduri
~~0~~	$\lfloor \log n \rfloor$	$1 = 2^0 = 2^{\lfloor \log n \rfloor - \lfloor \log n \rfloor}$
	$\lfloor \log n \rfloor - 1$	$2 = 2^1 = 2^{\lfloor \log n \rfloor - (\lfloor \log n \rfloor - 1)}$
~~i~~	i	$2^{\lfloor \log n \rfloor - i}$
	1	
~~$\lfloor \log n \rfloor$~~	0	$2^{\lfloor \log n \rfloor} = 2^{\lfloor \log n \rfloor - 0}$

$$T(n) = \sum_{i=0}^{\lfloor \log n \rfloor} 2^{\lfloor \log n \rfloor - i} \cdot i = \sum_{i=0}^{\lfloor \log n \rfloor} i \cdot \frac{2^{\lfloor \log n \rfloor}}{2^i} = 2^{\lfloor \log n \rfloor} \underbrace{\sum_{i=0}^{\lfloor \log n \rfloor} \frac{i}{2^i}}_{\leq 2}$$

$$T(n) \leq 2^{\lfloor \log n \rfloor + 1} \leq 2^{\log n + 1} = 2 \cdot 2^{\log n} = 2n \Rightarrow T(n) \in O(n)$$

Ex 5: Creați un heap, prin coborâri repetate:

a) 4, 1, 3, 2, 16, 9, 10, 14, 8, 7

b) 5, 3, 17, 10, 84, 19, 6, 22, 9

Heapsort

Construim întâi heap din vectorul nesortat. La fiecare pas, interschimbăm primul elem cu ultimul din heap, scădem dimensiunea heap-ului cu 1 și aplicăm Max Heapify pentru rădăcină în noul heap. După n-1 pași, vectorul va fi sortat.

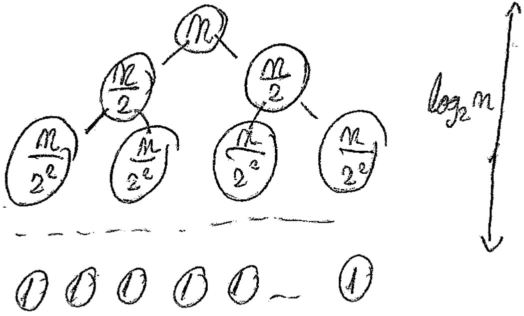
$$T(n) = \underset{build}{O(n)} + \underset{aplicare\ max\text{-}heapify}{(n-1) \cdot O(\log n)} = O(n \log n)$$

Ex 6: Aplicați heapsort pe acest vector:

a) 5, 13, 2, 25, 7, 17, 20, 8, 4

b) 4, 1, 3, 2, 16, 9, 10, 14, 8, 7

Best Case: Subvectori de aceeași lungime (sau diferență 1)



n

$\frac{n}{2} + \frac{n}{2} = n$

$4 \cdot \frac{n}{4} = n$

⋮

$\frac{n}{2} \cdot 2 = n$

$\Omega(n \log n)$

Split constant: La fiecare partiție, raportul dintre subvectorul stâng și cel drept e același

$\Rightarrow$ $T(n): n \log n$

Average Case: $n \log n$

Shell Sort

Generalizează Insertion Sort

Fie $h_1 > h_2 > h_3 > \dots > h_t = 1$ incremenți

Avem t pași. La pasul i: se sortează prin inserție subvectorii obținuți din vectorul inițial din componentele aflate la distanță h_i unele de altele:

$A[1], A[1+h_i], A[1+2h_i] \dots$

$A[2], A[2+h_i], \dots$

⋮

$A[h_i], A[2h_i], \dots$

La pasul t, se sortează prin inserție tot vectorul.

Incremenți:
- $1, 2, 4, 8, \dots$
  $h_m = 2 h_{m+1}$ și $h_t = 1$, $t = [\log_2 n]$
- $1, 4, 13, 40, \dots$
  $h_m = 3 h_{m+1} + 1$, $t = [\log_3 n] - 1$
- $1, 3, 7, 15, \dots$
  $h_m = 2 h_{m+1} + 1$, $t = [\log_2 n] - 1$

Complexitate: $n^{\frac{5}{3}}$

EX 1	62	83	18	53	7	17	95	86	47	69	25	28
$h_1 = 5$	17	28	18	47	7	25	83	86	53	69	62	95
$h_2 = 3$	17	7	18	47	28	25	69	62	53	83	86	95
$h_3 = 1$	7	17	18	25	28	47	53	62	69	83	86	95

Teorema limitei inferioare: Orice algoritm de sortare a n chei, bazat pe comparații, va face cel puțin: $\lceil \log_2 n! \rceil$ comparații în cazul cel mai nefavorabil

$[\log_2 n!]$ ~~cor~~ ——— n ——— mediu

Complexități

I. $\Theta(g) = \{ f \mid \exists\, c_1, c_2 > 0 \ \exists\, n_0 \text{ a.î. } 0 \le c_1 g(n) \le f(n) \le c_2 g(n), \forall n \ge n_0 \}$

$O(g) = \{ f \mid \exists\, c > 0 \ \exists\, n_0 \text{ a.î. } 0 \le f(n) \le c \cdot g(n), \forall n \ge n_0 \}$

$\Omega(g) = \{ f \mid \exists\, c > 0 \ \exists\, n_0 \text{ a.î. } 0 \le c \cdot g(n) \le f(n), \forall n \ge n_0 \}$

II. $f \in \Theta(g) \Leftrightarrow \lim\limits_{n \to \infty} \frac{f(n)}{g(n)} \in \mathbb{R}_{>0}$

$f \in O(g) \Leftrightarrow \lim\limits_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

$f \in \Omega(g) \Leftrightarrow \lim\limits_{n \to \infty} \frac{f(n)}{g(n)} > 0$

III. $f \in \Theta(g) \Leftrightarrow f = g$

$f \in O(g) \Leftrightarrow f \le g$

$f \in \Omega(g) \Leftrightarrow f \ge g$

Comportarea asimptotică a unei funcții (termenul dominant)

- ptr a compara 2 fct, ne interesează comportamentul lor pentru un rang foarte mar
($n \to \infty$)
- când comparăm 2 fct, comparăm termenii lor dominanți
- pas 1: eliminăm din termenii unei funcții pe toți cei care cresc încet și îl păstrăm doar pe cel care crește cel mai repede, când $n \to \infty$
- pas 2: eliminăm constantele din fața termenului

Ex: $f = 2n^2 + 3n + 1$

$g = 1000n + 500$

Pas 1: f: când n este foarte mare, $2n^2$ crește mai repede ca $3n$ și ca $1 \Rightarrow$ eliminăm $3n+1 \Rightarrow f = 2n^2$

g: $1000n$ crește mai repede ca $500 \Rightarrow g = 1000n$

Pas 2: f: eliminăm constanta $2 \Rightarrow f = n^2$

g: ——— " ——— $1000 \Rightarrow g = n$

Comparăm $f = n^2$ cu $g = n$, atunci când n ~~#~~ este foarte mare $\Rightarrow n^2 > n \Rightarrow f > g$

$\Rightarrow f \ge g \Rightarrow f \in \Omega(g)$ și $g \in O(f)$

$\Rightarrow 2n^2 + 3n + 1 \in \Omega(1000n + 50)$ și $1000n + 50 \in O(2n^2 + 3n + 1)$

! În cazul fct. polinomiale, termenul dominant este n la cea mai mare putere nenulă

În cazul logaritmilor, baza nu contează, fiind o constantă ($\log_2 n$ și $\log_{100} n$ au ambele termenul dominant "$\log n$")

$\log n$ crește mai încet decât n^a, ptr orice a pozitiv

n^b crește mai încet decât a^n, $\forall a > 1, \forall b$

$1 < \log n < n < n \log n < n^2 < n^2 \log n < n^3 < \ldots < 2^n$

Complexități

Ex 1: Term dominant: $17n+3$, $12n^2+3n$, $n+n\log n$, $n\sqrt{n}+n^2$, 2^n+n^7
$10n+7n^2+n^3$, $\log n+50$

Ex 2: $T(n)=3n^2+10n+7$ Atunci $T(n) \in$:
- $O(n^2)$, $\Omega(n^2)$, $O(n\log n)$, $O(n^3)$, $\Theta(n^2)$, $\Omega(n^3)$, $\Theta(n)$

Ex 3: T/F:
- $2n+3 \in O(n^2)$, • $n\log n \in O(n)$ • $5n^2 \in O(3n^2)$
- $15n+2 \in \Omega(\log n^2)$ • $130 \in \Omega(0{,}05n)$ • $2^n \in \Omega(n^{10})$
- $3\log n \in \Theta(\log n+50)$ • $n^3 \in \Theta(2n+4n^2)$ • $n^2 \in \Theta(n^2+n^3)$

Ex 4: Fie $f(n)=17n+3$. Dați exemplu de un $g(n)$ a.î.:
$f(n) \in O(g(n))$; $f(n) \in O(g(n))$ și $\notin \Theta(g(n))$; $f(n) \in \Omega(\overset{g(n)}{n})$ și $\in O(\overset{g(n)}{n})$
$f(n) \in \Omega(\underset{g(n)}{n})$ și $\notin O(\underset{g(n)}{n})$; $f(n) \in \Theta(\underset{g(n)}{n})$ și $f(n) \notin O(n)$

Ex 5: T/F:
- $f \in \Theta(n) \Rightarrow f \in O(n)$; $f \in \Theta(n) \Rightarrow f \notin \Omega(n)$; $f \in O(n^2) \Rightarrow f \in O(n^3)$
- $f \in \Omega(n) \Rightarrow f \in O(n^7)$; $f \in \Omega(n) \Rightarrow f \in \Theta(n)$; $f \in O(\log n) \Rightarrow f \in \Theta(1)$

Sortări

	Comp	Mutări	Complex	T(n)
Insertion	$\Omega(n), O(n^2)$	$\Omega(n), O(n^2)$	Best: n, Avg: n^2, W: n^2	$\Omega(n), O(n^2)$
Selection	$\Theta(n^2)$	$\Omega(n), O(n^2)$	$B=A=W=n^2$	$\Theta(n^2)$
Bubble	$\Theta(n^2)$	$\Omega(1), O(n^2)$	$B=n$ (optimizat) $A=W=n^2$	$\Omega(n), O(n^2)$

Insertion: După i paşi, primele i+1 elem sunt ordonate cresc.

Selection: După i paşi, ~~primele~~ sunt plasate pe poz. finale cele mai mici i elemente (sel. minim

_______ ″ _______ mari i elem (selecţia maxim

Bubble: După i paşi, sunt plasate pe poz finale cele mai mici i elem (parse dreapta-stânga)

_______ ″ _______ mari i elem (parse stânga-dreapta)

Ex1: Se dă vect: 22, 33, 55, 44, 88, 77, 66. ~~Care afirm~~ Vect. a fost obţinut prin:

a) 2 paşi ai alg. de selecţie a maximului

[✓ b)] 2 paşi sel. minimului

c) 3 paşi sel. minim

[✓ d)] 2 paşi bubble

[✓ e)] 2 paşi insertie

Ex2: Care din urm vectori sunt obţinuţi după 3 paşi ai bubble sort (prec. direcţia parselor

a) 11, 22, 55, 33, 77, 44, 66

[✓ b)] 22, 33, 44, 66, 55, 44, 77   D→S

c) 11, 77, 22, 33, 44, 55, 66

[✓ d)] 11, 22, 33, 66, 55, 44, 77   D→S

[✓ e)] 22, 44, 33, 11, 55, 66, 77   S→D

[✓ f)] 11, 22, 44, 66, 55, 77, 88, 99   S→D sau D→S

Ex3: Câte interschimbări sunt făcute ptr a ordona crescator prin bubble sort un vector sortat de

a) n b) n^2 [✓ c)] $\frac{n(n-1)}{2}$ d) $\frac{n(n+1)}{2}$ e) $\log_2 n$

Ex4: Care este nr maxim de paşi ~~de~~ ai insertion după care s-a obţinut vectorul:

22, 44, 66, 55, 77, 33   ?   2

Ex5: Este posibil ca urm. vectorul să fi fost obţinut după 3 paşi selection? Dar insertion

NU   DA

33, 55, 66, 77, 11, 44

Ex6: ~~Dacă~~ Fie T(n) nr de mutări efectuat de selection pe un vector oarecare. Atunci T(n) aparţin

[✓ a)] $O(n^2)$ b) $\Theta(n^2)$ [✓ c)] $\Omega(\log n)$ [✓ d)] $O(n^3)$ [✓ e)] $\Omega(n)$

Hash-uri

Adresare directă

Vector ce reține toate înregistrările din universul U.

Fiecare element k din U se va găsi la adresa $T[k]$.

Dezavantaje: - dimensiunea lui U foarte mare
- cheile care ne interesează din U sunt foarte puține, raportat la dimens. totala

Tabele de dispersie

Dimensiunea tabelului nu va mai fi $|U|$, ci un nr. m, mult mai mic, convenabil ales.

Fie $K \subset U$ mulțimea cheilor actuale. Un element k din K nu se va mai regăsi în tabel la adresa $T[k]$, ci la adresa $T[h(k)]$, unde h este o funcție de dispersie:

$h: U \rightarrow \{0, 1, \dots, m-1\}$

Problema apare atunci când 2 elemente din K sunt duse prin fct. h în aceeași valoare ($h(k_1) = h(k_2)$). Această situație se numește coliziune.

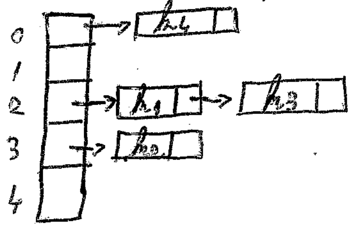
Metode de rezolvare a coliziunilor

I. Prin înlănțuire

Fiecare locație din tabel conține un pointer către o listă dublu înlănțuită, în care sunt stocate toate valorile duse prin fct. h în acea locație.

Ex: $m = 5$

$h(k_1) = 2$
$h(k_2) = 3$
$h(k_3) = 2$
$h(k_4) = 0$



Ex 1:

A	B	C	D	E	F	G	H
24	60	35	70	13	49	59	68

$h(x) = x \bmod 11$

Fie $n = |K|$
$m = |T|$. Atunci $\alpha = \frac{n}{m}$ s.n. factorul de încărcare al tabelului
(α e raportul dintre nr. de chei actuale și nr. de locații din tabel, i.e nr. mediu de chei stocate în fiecare locație)

De obicei, $\alpha > 1$ ptr. coliziuni rezolvate prin înlănțuire.

Timpul mediu la căutarea fără/cu succes: $O(1 + \alpha)$

II Prin adresare deschisă

Fiecare locație va avea maxim 1 element. Apar probleme când un element trebuie să se du într-o locație ocupată. Astfel, pentru un element k, funcția h nu va mai genera o adre ci un șir finit de adresă (va genera, aleator, fiecare din cele m adrese, și se va opri atun când va găsi o adresă liberă).

$\alpha < 1$

$h : U \times \{0, 1, \dots, m-1\} \to \{0, 1, \dots, m-1\}$

$k \mapsto \langle h(k,0), h(k,1), \dots, h(k, m-1) \rangle$ sn secvența de sondaj ptr cheia k

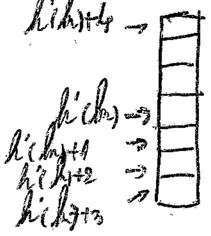
$m!$ șiruri de adrese posibile

a) Lineară

$h(k,i) = (h'(k) + i) \bmod m$

$h' : U \to \{0, 1, \dots, m-1\}$

$\langle h'(k), (h'(k)+1) \bmod m, \dots, (h'(k) + m-1) \bmod m \rangle$



~~A~~ Cunoscând $h'(k)$, putem afla șirul complet de adrese (valoarea primului termen determ tot șirul) ⇒ nr. de șiruri posibile este nr de valori posibile ptr primul termen

⇒ m secv. de sondaj distincte din cele $m!$

Ex 2:

A	B	C	D	E	F	G	H
24	60	35	40	13	49	59	68

$h(k,i) = (k + i) \bmod 11$

b) Pătratică

$h(k,i) = (h'(k) + c_1 i^2 + c_2 i) \bmod m$, $c_1 > 0$, $c_2 \geq 0$

Analog ⇒ m secv. de sondaj distincte din cele $m!$

Ex 3: $h(k,i) = (k + i^2) \bmod 11$

c) Dublă dispersie

$h(k,i) = (\underset{\text{adresă de bază}}{h_1(k)} + i \cdot \underset{\text{offset}}{h_2(k)}) \bmod m$

Cunoscând primul termen din secvența de sondaj $(h_1(k))$, nu e suficient ptr a ști tot șirul. Trebuie să cunoaștem și $h_2(k)$.

$h_1(k) \in \{0, \dots, m-1\}$
$h_2(k) \in \{0, \dots, m-1\}$ $\Big\} \Rightarrow m^2$ perechi

Ptr fiecare pereche $(h_1(k), h_2(k))$, avem o secvență diferită

$\Rightarrow m^2$ secv. de sondaj distincte din cele $m!$

Ex 4: $h(k,i) = (k + i(k \% 10 + 1)) \bmod 11$

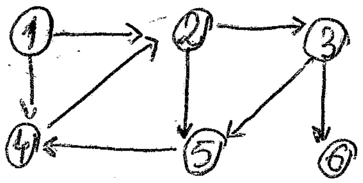
Ex 5:
A = 7
B = 25
C = 14
D = 27
E = 13
F = 33
G = 1
H = 15

a) $k \% 13$
b) $(k+i) \% 13$
c) $(k + i^2 + i) \% 13$
d) $(k + i * k \% 10) \% 13$

Grafuri

$G = (V, E)$

Reprezentări de grafuri: matrice de adiacență ($a[i][j]=1$, dc. ∃ muchie (i,j))
listă de adiacență (fiecare nod are o listă cu vecinii săi)

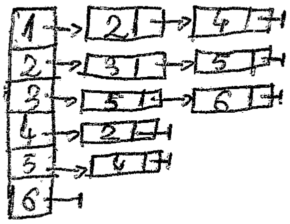


Matrice:

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 & 1 \\ 4 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Spațiu: $|V|^2$

Liste:



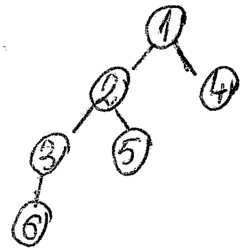
Spațiu: $|V| + |E|$

Parcurgerea în lățime (Breadth)

- adăugăm primul nod într-o coadă
- cât timp coada e nevidă, scoatem ~~un~~ primul nod din coadă și adăugăm la final toți vecinii săi nevizitați

Q: ~~1~~ ~~2~~ ~~4~~ ~~3~~ ~~5~~ ~~6~~ | Q: ~~1~~ ~~2~~ ~~4~~ ~~3~~ ~~5~~ ~~6~~

1, 2, 4, 3, 5, 6

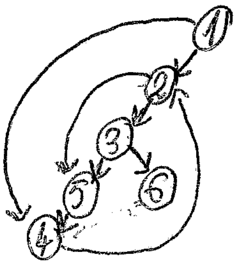


Complex: $O(V^2)$ - matrice
$O(V+E)$ - liste

Parcurgerea în adâncime (Depth)

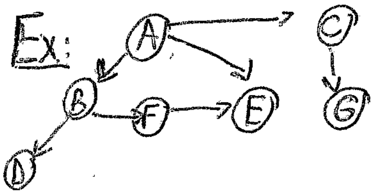
- se realizează cu aj. unei fct. recursive: marchează ca vizitat nodul parametru și apelează recu fct. ptr fiecare vecin nevizitat

1 2 3 5 4 6
(4) (5) (6)



Muchiile de tip "tree": (1,2), (2,3)...
Muchii "forward": (1,4), (2,5)
Muchii "back": (4,2)
Muchii "cross":

Ex:



Tăieturi: Sn tăietură a unui graf G o partiție a lui V: (S, V\S)
Fiecare tăietură generează un set de muchii ce respectă tăietura (cu ambele capete în S sau ambele în V\S).
O tăietură ⟺ o submulțime a lui V (și complementara)
2^n submulțimi ⟹ 2^n tăieturi
$n=|V|$

Arbori parțiali de cost minim

$G = (V, E, w)$ neorientat, conex

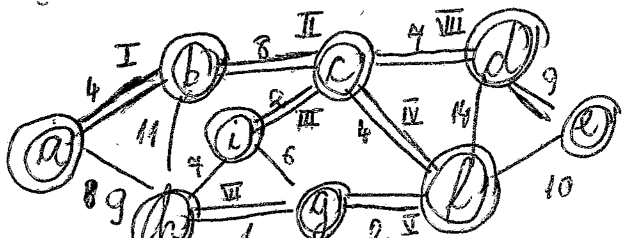
$w : E \to \mathbb{R}$

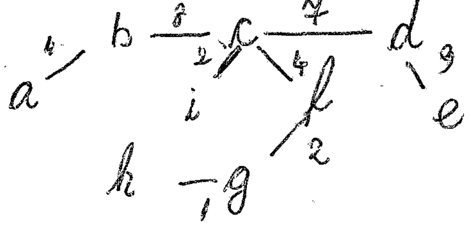
Căutăm subgraf T al lui G, conex, de cost minim => arbore

$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

Algoritmul lui Prim

- se pornește de la un nod sursă
- inițial, T conține ~~doar~~, ca nod, doar nodul sursă, iar ca muchii, mult. vidă





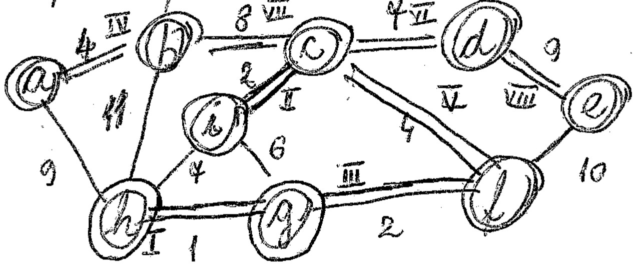
- la fiecare pas, adăugăm în T muchia cu ponderea cea mai mică, ce conectează un nod din T cu unul care nu e în T (adăugăm și acest nod în T)

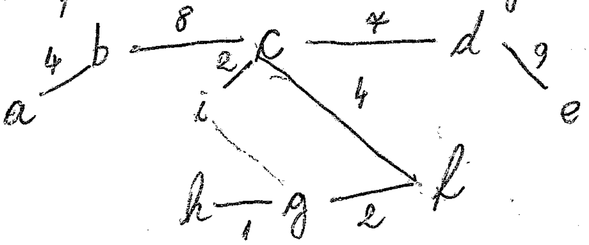
T: V: a, b, c, i, f, g, h, d, e

E: (a,b), (b,c), (c,i), (c,f), (f,g), (g,h), (c,d), (d,e)

Algoritmul lui Kruskal

- sortează crescător muchiile, după pondere
- la fiecare pas, ia muchia cea mai mică, ce nu ~~unește~~ formează un ciclu cu cele deja existente





Ex: