

Dane: $P[1 \dots m]$

wzór

$T[1 \dots n]$

tekst

WYSZUKIWANIE

WZORCA

(P sufixem T_{s+m})

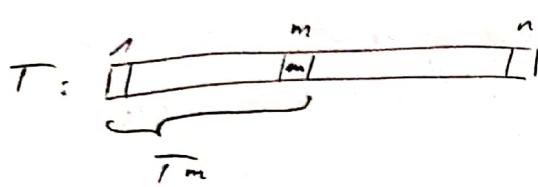
1

Problem: Znaleźć wystąpienie wzorca P w T t.j. $P \sqsubset T_{s+m}$

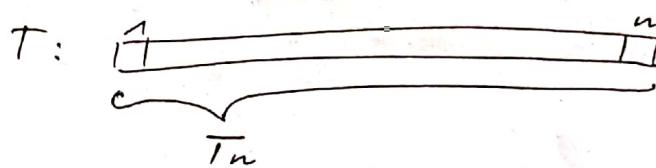
(T_{s+m} - $s+m$ elementary prefixes tekstu $T[1 \dots n]$, s - przeniesienie $\in \langle 0, n-m \rangle$)



• $s = 0, P \sqsubset T_m$



• $s = n-m, P \sqsubset T_{n-m+m} = T_n$



$P \sqsubset T_{s+m}$

$\in P, \text{bo ma alt. } m$

Algorytm:

1° Naiwy: Pista po T

KoST: $O((n-m+1)m)$

bo P alt. m,
porównywanie kroki
zakł

idx-wystąpien = 1

for $s \leftarrow 0$ to $s \leftarrow n-m$:

if $P[1 \dots m] == T[s+1 \dots s+m]$

idx-wystąpien.append(s)

2° Algorytm Karpka - Rabinera

- Każdy symbol z alfabetu Σ to cyfra w systemie d -ryfrogram, np. $d=16$
- $\Sigma = \{1, 2, 3, \dots, 9, A, B, \dots, F\}$,
- $|\Sigma| = d$.

Tekst z k -symbolami to linia k -ryfrogramu.

p - wz. wzorca $P[1 \dots m]$

t_s - wz. linia $T[s+1 \dots s+m]$

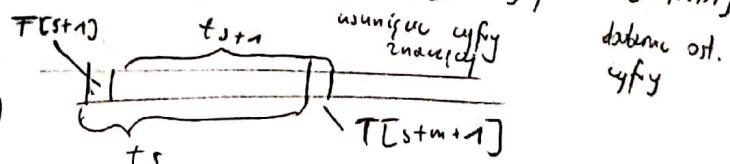
$P[1 \dots m] = T[s+1 \dots s+m] \Rightarrow p = t_s$

Obliczenie p i t_s dla $d=10$:

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \dots + 10P[1]))$$

KoST: $O(m)$

$$t_{s+m} = 10(t_s - 10^{m-1} \cdot F[s+1]) + T[s+m+1]$$



KoST: $O(m)$

- Gdy dlugosc urova (m) jest duza, to kubka reprezentacja stara moze nie miec sie w slowniku mazywacym, Dlatego, by unikac konfliktow operacji, na porbowywanie kubek robiacy mod q , gdzie q to taka liczba (zwykly pierwszy), iż d \cdot q miej sie w slowniku mazywacym.

Algorytm:

```

 $\begin{cases} \text{idx\_wynik} = 1 \\ n, m - dl. T : P \\ h = d^{m-1} \mod q \quad // \text{wystarczaj do mnozenia wybranych znakow cyfry w T} \\ p = 0 \quad (\text{by obliczanie kolejnych poteg}) \\ t_0 = 0 \end{cases}$ 
for  $i \leq 0$  to  $i \leq m$  // uzyjcie wartosci p[i] to (sumujec wart. zakresu
    p =  $(d \cdot p + P[i]) \mod q$  // do kolejnego wyniku)
    t_0 =  $(d \cdot t_0 + T[i]) \mod q$ 
for  $s \leq 0$  to  $s \leq n-m$  // wyszukiwanie
    if  $p == t_s$  // jesli True, to moze wystepowac, ale niekoniecznie
        if  $P[1..m] = T[s+1..s+m]$  then wynikwyszukiwanie
            idx_wyst.append(s)
    if  $s < n-m$  // jesli to out. s, to nie obliczamy
        ts+1 =  $(d \cdot (ts - T[s+1] \cdot h) + T[s+1+m]) \mod q$ 
    
```

KOST: ~~Wyszukiwanie~~ - najgorzej przypadek, urowan we jego "zdawkach palce"
 $O((n-m+1) \cdot m)$ wystepuje duzo razy

~~Wyszukiwanie~~

$O(nm + km)$ - średni koszt, gdzie k to l. wystepowań urova

bo $O(m) + O(n + kn)$

|
tak naprawde $n-m$, ale mozliwe?

3° Algorytm KMP (a raczej MP (Morrison - Pratt), bo taki był przedstawiony na wykładzie pod tym nazwą. KMP ma funkcję π , która daje lepsze wyniki (sybory czasu))

3

Działanie: konstanty z funkcji π , która reprezentuje informacje, o tym gdzie zacząć szukać w przypadku wystąpienia niezgodności. Dzięki temu zyskujemy linowy czas działania $O(n+m)$.

Punkt 1:

$P = abacabab$

	a	b	a	c	a	b	a	b
q	1	2	3	4	5	6	7	8
$\pi(q)$	0	0	1	0	1	2	3	2

dt. zgodnego podzięga
do końca indeks

Algorytm KMP: Koszt: $O(n+m)$

$n \leftarrow$ dt. T

$m \leftarrow$ dt. P

$q \leftarrow 0$

$\pi \leftarrow$ oblicz-funkcja- π -dla-P

for $i \leftarrow 1$ to n do

(while ($q > 0$ and $P[q+1] \neq T[i]$)
do $q \leftarrow \pi(q)$)

(if ($P[q+1] == T[i]$)
then $q \leftarrow q+1$)

if ($q = m$) then unie ("znaleziono zgodność", $i-m$);

$q \leftarrow \pi(q)$

Obliczanie π : Koszt: $O(m)$

$m \leftarrow$ dt. P

$\pi(1) \leftarrow 0$; $k \leftarrow 0$

for $q \leftarrow 2$ to m do

(while ($k > 0$ and $P[k+1] \neq P[q]$)
do $k \leftarrow \pi(k)$)

(if ($P[k+1] == P[q]$) then $k \leftarrow k+1$;
 $\pi(q) \leftarrow k$)

return π

Funkcja π

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

$\pi(q) = \max \{k < q : P_k = P_q\}$

np.

$P_q = abacabab$ $q=6$

$P_k = ab$ $k=2$

P_k to maksymalny sufiks P_q , który jest jednocześnie jego prefiksem.

Punkt 2:

	a	a	a	a	b	a
q	1	2	3	4	5	6
$\pi(q)$	0	1	2	3	0	1

5. $k=1$: $\pi(1) = ?$ $\because k > 0$ and $P[1] \neq P[1]$
 $q=6$: if ($P[1] == P[6]$)
 $k=1, 1=2$
 $\pi(6)=2$

Punkt 1:

1. $k=0$: $\pi(2) = ?$ $\because k > 0$ and $P[1] \neq P[2]$
 $q=2$: if ($P[1] == P[2]$) $\pi(2)=1$ $\rightarrow \pi(1)=0$

2. $k=0$: $\pi(3) = ?$ $\because k > 0$ and $P[1] \neq P[3]$
 $q=3$: if ($P[1] == P[3]$) $\pi(3)=1$ $\rightarrow k=0+1=1$

3. $k=1$: $\pi(4) = ?$ $\because k > 0$ and $P[2] \neq P[4]$
 $q=4$: if ($P[2] == P[4]$) $\pi(4)=1$ $\rightarrow k=1+1=2$

4. $k=0$: $\pi(5) = ?$ $\because k > 0$ and $P[1] \neq P[5]$
 $q=5$: if ($P[1] == P[5]$) $\pi(5)=1$ $\rightarrow k=0+1=1$

Przykład 1 - krok algorytmu :

4

	a	b	a	c	a	b	a	b
q	1	2	3	4	5	6	7	8
$\pi(q)$	0	0	1	0	1	2	3	2

1. $k=0$ • while ($k > 0 \dots$) X
 $q=2$ • if ($P[1] == P[6]$) X $\pi(2) = k=0$
 then
2. $k=0$ • while ($k > 0 \dots$) X
 $q=3$ • if ($P[1] == P[3]$) ✓
 $k = k+1 = 1$ $\pi(3) = k=1$

3. $k=1$ • while ($k > 0$ and $P[2] \neq P[4]$) ✓
 $q=4$ $k = \pi(k) = \pi(1) = 0$

stępujący prefiks wynosiła 1, zatem wstawiany k na dle. prefiksu, który odpowiada pierwemu elementu.

- while ($k > 0 \dots$) X
 • if ($P[1] == P[4]$) X $\pi(4) = k=0$

4. $k=0$ • while ($k > 0 \dots$) X
 $q=5$ • if ($P[1] == P[5]$) ✓
 $k = k+1 = 1$ $\pi(5) = 1$

5. $k=1$ • while ($k > 0$ and $P[2] \neq P[6]$) X
 $q=6$ • if ($P[2] == P[6]$) ✓
 $k = k+1 = 2$ $\pi(6) = 2$

6. $k=2$ • while ($k > 0$ and $P[3] \neq P[7]$) X
 $q=7$ • if ($P[3] == P[7]$) ✓
 $k = k+1 = 3$ $\pi(7) = 3$

7. $k=3$ • while ($k > 0$ and $P[4] \neq P[8]$) ✓
 $q=8$ $k = \pi(k) = 1$

- while ($k > 0$ and $P[2] \neq P[8]$) X
 • if ($P[2] == P[8]$) ✓
 $k = k+1 = 2$ $\pi(8) = 2$

many wspólny podciąg
dla k i tworzący na
różny element

k wstawiane na dle.
najnowszego podciągu
od którego mniej
budowią nowy podciąg.

Jeli element sprawdzi all
nego k, to tworzący
odpowiedni dle. podciąg,
jeli nie, to schodząc z k
tak długo aż podciąg będzie
dł. 0.

$$\text{wrong } P = \underline{\alpha}6\alpha66$$

	a	b	a	b	b
q	1	2	3	4	5
$\pi(q)$	0	0	1	2	0

$$P = \underline{\alpha}6\alpha66$$

$$T = \underline{\alpha}666aa\underline{\alpha}baba66a$$

$$n = 11$$

$$m = 5$$

$$q = 0$$

$$\pi = [0, 0, 1, 2, 0]$$

for $i = 1$ to 11 do

while ($q > 0$ and $P[q+1] \neq T[i]$) // to same
co ur π

$$\text{do } q = \pi(q)$$

if ($P[q+1] == T[i]$)

$$\text{then } q = q+1$$

if $q = m$ // wrong inclusion

then while ("wrong inclusion" \rightarrow proszajac "od" $i=m$);

$$q = \pi(q)$$

1. $i = 1, q = 0$

• while ($q > 0$...) X

• if ($P[1] == T[1]$) V

$$q = q+1 = 1$$

• if ($q = m$) X

2. $i = 2, q = 1$

• while ($q > 0$ and $P[2] \neq T[2]$) X

• if ($P[2] == T[2]$) V

$$q = q+1 = 2$$

3. $i = 3, q = 2$

• while ($q > 0$ and $P[3] \neq T[3]$) V

$$q = \pi(q) = 0 \quad // \text{incorrect move or padding}$$

$$(q > 0 \text{ and } P[2] \neq T[3])$$

• if ($P[1] == T[3]$) X

4. $i = 4, q = 0$

• while X

• if ($P[1] == T[4]$) X

5. $i = 5, q = 0$

• while X

• if ($P[1] == T[5]$) V

$$q = q+1 = 1$$

6. $i = 6, q = 1$

$$6 \times 6$$

• while ($q > 0$ and $P[2] \neq T[6]$) X

• if () V

$$q = q+1 = 2$$

7. $i = 7, q = 2$

• while ($q > 0$ and X)

$$q = q+1 = 3$$

$$(\dots) \quad q = m \quad V$$

4^o Algorytm Boyera-Moore'a (alg. wyszukiwania w orku, leży w KMP)

6

ale ta ta wymaga buforowania danych, więc
wyszukanie może być zatrzymane - wiele jest na systemie
a nie w pamięci komputera

Idea: Wszelkie sprawdzenia jest odtworzone.

- Stosowane d. heurystyki: 1. "zły znak"
2. "dobry sufix"

Ad. 1

Jesli następuje niezgodność: $P[j] \neq T[s+j]$, to

$$k = \max \{ z \mid P[z] = T[s+z] \} \text{ jeśli taki z istnieje}$$

Przesuniecie o $j-k$ znaków, jeśli $j > k$, wpp. nie.

Przykład: $P = ABCAB$, $T = ACBABAABCABD$

1. $A C B A D B A B C A B D$

$A B C A$
 B

$j=5$

$k = 0, 6, \dots$ i P nie ma D (z nikturyc)

$$j > k \rightarrow \text{przesuniecie } 5-0 = 5$$

2. $A C B A D | B A B C A B D$

$A B C A$
 B

$k = 4, 6, \dots P[4] = T[5+5]$

$$j > 4 \rightarrow \text{przesuniecie } 5-4 = 1$$

3. $A C B A D B | A B C A B D$

$A B C A B$

zakl. $P \sim T$ ($s=6$)

Ad. 2

Q jest podobne do R , jeśli $Q \supset R$ lub $R \supset Q$.
($Q \sim R$)

Heurystyka mówi, że po napotkaniu niezgodności $P[j] \neq T[s+j]$
mocnym zakończeniem przeszukiwania o $m - \max \{ k \mid 0 \leq k < m \text{ i } P[j+k] \sim P_k \}$.

jeśli nie ma podobieństwa, to
 $\max = 0$

5° Shift - And

Idea: W trakcie cyklu tekstu pamiętamy inf. o wszystkich prefiksach wortca, które są sufiksami przetwarzanego fragmentu tekstu. (u KMP "zajdźmy")

Początkowy do wyszukiwania krótkich wortów.

(weltor)

- Najpierw tworzymy tabelę B, która trzyma inf. o tym, na który pozytyjny we wortcu wystąpiły litery z alfabetu, np.

$$\Sigma = \{a, i, p, s\}$$

$$P = assi$$

	a	s	s	i
B[a]	1	0	0	0
B[i]	0	0	0	1
B[p]	0	0	0	0
B[s]	0	1	1	0

$$\forall c \in \Sigma$$

$$\begin{cases} P[i] = c, \text{ to } B[c].i = 1 \\ P[i] \neq c, \text{ to } B[c].i = 0 \end{cases}$$

Algorytm wyszukiwania wortca:

Kiedy zauważono wortce? : gdy $D.(m-1) = 1$

T - tekst $T[0, \dots, n]$ n - dl. T (najwyżej licząc od 1)
 P - wortec $P[0, \dots, m]$ m - dl. P (albo go tu na koniec)

$\begin{cases} \text{for } c \in \Sigma \text{ do } B[c] \leftarrow 0 \\ \text{for } i \leftarrow 0 \text{ to } m-1 \text{ do } B[P[i]] \leftarrow B[P[i]] + 2^i & // B[P[i]].i \leftarrow 1 \\ D \leftarrow 0 \\ \text{for } j \leftarrow 0 \text{ to } n-1 \text{ do} \\ \quad D \leftarrow (D \ll 1) + 1 & // \\ \quad D \leftarrow D \& B[T[j]] & // usuwając prefiks, olla kiedy T[j] nie powie (np. py "ip") \\ \quad \text{if } D \& 2^{m-1} \neq 0 \text{ then return } j-m+1 & // zauważono wortce \end{cases}$

(1000...00)

KOST: przygotowanie: $O(\delta + m)$
scenariusz: $O(n)$

	a	s	s	i				
a	1	0	0	0	j=0	$\bullet D \leftarrow 0$	$000\dots 001$	$T[0] = a$
P	0	0	0	0	j=1	$\bullet (D \ll 1) + 1$	$100\dots 001$	$B[a] = 10000 0001$
a	1	0	0	0	j=2	$\bullet D \& B[T[j]]$	$= 0001$	$T[1] = p$
S	0	1	0	0	j=3	$\bullet (D \ll 1) + 1$	0011	$B[p] = 0000$
S	0	0	1	0	j=4	$\bullet D \& B[T[j]]$	0001	$T[2] = a$
i	0	0	0	1	j=5	$\bullet (D \ll 1) + 1$	0011	$B[a] = 0001$
						$\bullet D \& B[T[j]]$	0010	$T[3] = s$
						$\bullet (D \ll 1) + 1$	0101	$B[s] = 0110$
						$\bullet D \& B[T[j]]$	0100	$T[4] = s$
						$\bullet (D \ll 1) + 1$	1011	$B[s] = 0110$
						$\bullet D \& B[T[j]]$	1000	$T[5] = i$
								$B[i] = 1000$

zapisywany na odwrotnie

6° Algorytm Karpa - Millera - Rosenberga (KMR)

8

- Idea:
- tworzymy słowo $u = PT$, gdzie konkatenacją $P \cdot T$ np. $P = abba$
 $T = abbaaabba$
 - numerujemy wszystkie podstare m-litersowe słowa w $u = abbaaabba$
(taki samie słowa otrzymujemy ten sam numer)
 - wyszczególnia na tych przypadkach $k > m$ (za P), na których zauważają się podstare o numerach takich jak podstawa na pozycji 1 (za P)

Numerowanie podstew:

$$l = 2^k$$

- $l \in \mathbb{N}$, zaczynając od 1
- np. $ababbcaad$ $l = 1$ (podstawa dl. 1)
 $\begin{matrix} 1 & 2 & 1 & 2 & 2 & 3 & 1 & 4 \end{matrix}$
- następne numerowanie podstare o dł. 2: $l' = 2l$

<u>ab</u>	<u>ba</u>	<u>bb</u>	<u>bc</u>	<u>ca</u>	<u>ad</u>
1,2	2,1	1,2	2,2	2,3	3,1

\rightarrow sortujemy pary (x,y)
i nadajemy im numery
 $\in \mathbb{N}$, od 1

<u>ab</u>	<u>ba</u>	<u>bb</u>	<u>bc</u>	<u>ca</u>	<u>ad</u>
1	3	1	4	5	2

- dł. 4: $l'' = 2l' = 4l$

<u>abba</u>	<u>baab</u>	<u>abbb</u>	<u>bbbc</u>	<u>bcca</u>	<u>caad</u>
1,3	3,1	1,4	4,5	5,6	6,2

\rightarrow

1	3	2	4	5	6
---	---	---	---	---	---

KOST: $O(n \log m)$

Jesli chcemyindektowaćnumerydla słów m-litersowych, to musimy obliczyć numery $\lceil \log m \rceil$ różnych dłuższych podstew (l_0 $l=2^k$).

A n dlatego, iż u bieżąco z far przedstawiamy przez wszystkie litery z tekstu.

KMR może być wykorzystany np. do znajdowania najdłuższego powtarzającego się podstowa.

7⁰Shift-Anal (wersja z wykłasu)

9

(5¹⁰)

$C_j[0 \dots m]$ - uktor charakterystyczny zbioru prefiksów wzorca, które są sufiksami po przesunięciu j -tej litery (litera ta jest)

(tj. $C_j[k] = \text{true} \Leftrightarrow P_k \sqsupseteq T_j$)



P_k Sufiksem T_j

ma długosć wzorca
i jest prawdziwy, gdy P_k jest
sufikiem T_j np. $T_j = \alpha\beta\gamma\delta$, $P_6 = \alpha\beta\gamma$
 $P_5 = \alpha\beta\gamma\delta$



Punkt 1: $\Sigma = \{a, b, c, d\}$

$P = \overset{1}{a} \overset{2}{b} \overset{3}{a} \overset{4}{b} \overset{5}{c} \overset{6}{a} \overset{7}{c}$

$R_a = 11010101$

$R_b = 10101000$

$R_c = 10000010$

$R_d = 10000000$

↑
bit
wzory
 $R_d[2] = 0$

$C_j = \text{shift}(C_{j-1}) \text{ AND } R_j$

przesunięcie o 1 bit w prawo
z ustawieniem bitu wzoru na 1

R dla j=0
utulni (pod P_j)

(hyba coś
nic tak)

Punkt 2: $\Sigma = \{a, i, p, s\}$

$P = assi$

$T = \overset{1}{\alpha} \overset{2}{p} \overset{3}{a} \overset{4}{s} \overset{5}{s} \overset{6}{i}$

$R_a = 11000$

$R_i = 10001$

$R_p = 10000$

$R_s = 10110$

jedzie po literach tekstu, C oznacza P

$C_0 = 0$

for $j \in 1 \dots n$ do

$C_j = \text{shift}(C_{j-1})$
AND R_{P_j}

$C_0 = 11000$

$C_1 = M000 \text{ AND } R_a$

$= 11000 \text{ AND } 11000 = 01000$

$C_2 = 10100 \text{ AND } R_p =$

$= 10100 \text{ AND } 10000 = 10000$

$C_3 = 11000 \text{ AND } R_a =$

$= 11000 \text{ AND } 11000 = 11000$

$C_4 = M100 \text{ AND } R_s =$

$= 11100 \text{ AND } 10110 = 10100$

$C_5 = M010 \text{ AND } R_s =$

$= 10100 \text{ AND } 10110 = 10010$

$C_6 = 11001 \text{ AND } R_i =$

$= 11001 \text{ AND } 10001 = 10001$

	a	s	s	i
a	1	0	0	0
p	1	0	0	0
s	1	1	0	0
s	1	0	1	0
i	1	0	0	1

?

Problem: Mnożenie wielomianów
(dzielenie ich postaci do operacji)

10

SZYBKA TRANSFORMACJA FOURIERA (FFT)

Dane: a_0, a_1, \dots, a_{n-1} ← współczynniki
 b_0, b_1, \dots, b_{n-1} ← wielomianów
 $A(X) : B(X)$

(ZAMIANA REPREZENTACJI)
WIELOMIANU

Wynik: c_0, c_1, \dots, c_{n-1} ← współczynniki wielomianu $C(X) = A(X) \cdot B(X)$

Reprezentacje wielomianów (tu wielomian A stopnia $n-1$)
 $\sum_{i=0}^{n-1} a_i X^i = a_0 + a_1 X + \dots + a_{n-1} X^{n-1}$

- [Wsp] współczynnikowa (n -el. wektor): $\langle a_0, a_1, \dots, a_{n-1} \rangle$

- [War] zbiór wartości w n różnych punktach: $\{f(x_i, y_i) : i = 0, \dots, n-1\}$
 $\begin{cases} i \neq j \leq n-1 & x_i \neq x_j \\ y_i = f(x_i) \end{cases}$

Operacje na wielomianach:

- dodawanie: $O(n)$ (dla obydwu reprezentacji)
- odzielenie wartości w punktach: - łatwe dla reprezentacji [Wsp] (up. schemat Hornera) $O(n)$
- trudne dla reprezentacji [War]
- mnożenie: - łatwe przy reprezentacji [War] ($O(n^2)$)
- trudne przy reprezentacji [Wsp]

Schemat algorytmu:



- $[\text{Wsp}] \rightarrow [\text{War}]$ * podziel A na dwie części: jest rekurencyjny
 (dziel i zwyczaj),]
- modyfikacja i jednoscia (na pierwotną rekurencję)
 - podziel A na dwa wielomiany:upt. parzyste i nieparzyste (A_{par} , A_{niepar})
 - otrzymuj A jako linię przedziałów ($A_{\text{par}}(z)$ i $A_{\text{niepar}}(z)$)
- W
SKRÓCIE

$[\text{War}] \rightarrow [\text{Wsp}]$

$$\left[\begin{array}{c} \text{macierz} \\ \text{z pierwiastkami} \\ \text{jednostki} \\ \text{mającą } V_n \end{array} \right] \cdot \left[\begin{array}{c} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{array} \right] = \left[\begin{array}{c} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{array} \right]$$

wektory \bar{a} wektory \bar{y}

* mamy \bar{y} : pierwiastki

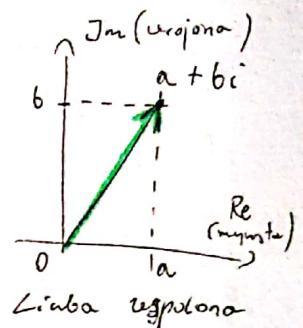
* chcemy otrzymać \bar{a} , czyli $\frac{1}{n} \cdot [V_n^{-1}] \cdot \bar{y}$

Jak wybrać punkty do reprezentacji [War] ([Wsp] → [War])

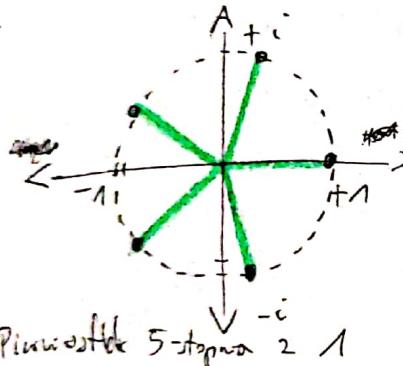
11

- trzeba znaleźć n -te pierwiastki z jedności: $z: z^n = 1$, gdzie $z \in \mathbb{C}$ (complex = rozpolone) (mają one rezywty i urojone)
- wtedy będą też rozpolonych istnieje dolicznicie n n -tych pierwiastków z jedności.
- $\sum_{k=0}^{n-1} e^{\frac{2\pi ik}{n}}$, $i = 0, \dots, n-1$
- n -ty pierwiastek ~~z jedności~~ pierwotny z jedności to taki pierwiastek, który generuje wszystkie inne pierwiastki (n -te) z jedności.

Na płaszczyźnie rozpolonej pierwiastki n -tego stopnia z jednością wiedzącymi wielkością formnego o n kątach wspólnego uгла jednostkowego, którego jeden z wierzchołków leży w punkcie 1. Resztyce są położone tego okręgu na n równych częściach.



np.



Dla $n > 1$ wszystkie pierwiastki z jedności n -tej st. sumują się do 0.

$$\sum_{k=0}^{n-1} e^{\frac{2\pi ik}{n}} = 0$$

Pierwiastek pierwotny: $e^{\frac{2\pi i}{n}}$ ($k=1$)

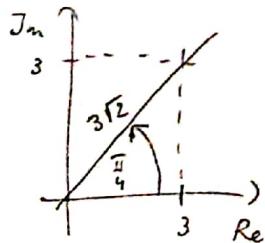
Wszystkie n -te pierwiastki z jednością: $\{w_n^0, w_n^1, \dots, w_n^{n-1}\}$

Obszary: n -kątowe, $\{w_n^0, w_n^1, \dots, w_n^{n-1}\}$ - tb. n -te pierwiastki

$$(w_n^0)^2, (w_n^1)^2, \dots, (w_n^{n-1})^2 = \{w_{\frac{n}{2}}^0, w_{\frac{n}{2}}^1, \dots, w_{\frac{n}{2}}^{\frac{n}{2}-1}\}$$

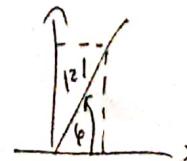
* zapis liczby rozpolonej w postaci trygonometrycznej:

$$z = 3 + 3i = 3\sqrt{2} \left(\cos \frac{\pi}{4} + i \sin \frac{\pi}{4} \right) \quad | \quad z = |z|(\cos \varphi + i \sin \varphi)$$



Potęgowanie l. rozpolonej:

$$z^n = |z|^n (\cos n\varphi + i \sin n\varphi)$$



Redukcja problemu:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}$$

Twierdzenie 2 uogólniający Ap i An t.j. $A(x) = A_p(x^2) + A_n(x^2)$, gdzie Ap daje się podać indywidualnie, a An niewystępuje.

$$A_p(z) = a_0 + a_1 z + a_2 z^2 + \dots + a_{n-2} z^{\frac{n}{2}-1}$$

$$A_n(z) = a_1 z + a_3 z^2 + a_5 z^4 + \dots + a_{n-1} z^{\frac{n}{2}-1}$$

Chęć otrzymać wektor' A(x) dla $x \in \{w_n^0, w_n^1, w_n^2, \dots, w_n^{n-1}\}$

obliczamy wartości Ap(z) i An(z) dla $z \in \{(w_n^0)^2, (w_n^1)^2, \dots, (w_n^{n-1})^2\}$,
czyli $z \in \{w_{\frac{n}{2}}^0, w_{\frac{n}{2}}^1, \dots, w_{\frac{n}{2}}^{\frac{n}{2}-1}\}$

=====

Niech $\bar{a} = a_0, a_1, \dots, a_{n-1}$ - wstępn. uogólnienie A

Algorytm:

FFT(\bar{a}) $\quad [\text{Wsp}] \rightarrow [\text{War}]$

// do FFT podajemy wektor wstępn. uogólnienia

if ($n=1$)

return \bar{a}

$w_n \leftarrow e^{\frac{2\pi i}{n}}$

$u \leftarrow 1$

$\bar{a}_p \leftarrow \langle a_0, a_2, \dots, a_{n-2} \rangle$ // wektor wstępn. rozdzielony na dwa wektory

$\bar{a}_n \leftarrow \langle a_1, a_3, \dots, a_{n-1} \rangle$

$\bar{y}^p \leftarrow \text{FFT}(\bar{a}_p)$ // rekurencyjne wykonywanie n-mu FFT

$\bar{y}_n \leftarrow \text{FFT}(\bar{a}_n)$

for $k \leftarrow 0$ to $\frac{n}{2}-1$ k-th element wektora \bar{y}^p

$y_k \leftarrow \bar{y}_k^p + w_k \bar{y}_k^n$

$y_{k+\frac{n}{2}} \leftarrow \bar{y}_k^p + w_k \bar{y}_k^n$

$u \leftarrow u \cdot w_n$

return $\bar{y} = \langle y_0, \dots, y_{n-2}, y_{n-1} \rangle$ // zwracamy wektor dla n-tyle pierwiastków jednostki (redukując do uogólnienia)

$A(w_n^0) \quad A(w_n^{n-2}) \quad A(w_n^{n-1})$

do stosowania zasada działać i rekurencyjnie, względnie na 2 i

Czas: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n) \rightarrow T(n) = O(n \log n)$ rekurencja FFT na połowe wektora

Jak przejść z $[W_{\text{or}}]$ do $[W_{\text{sp}}]$ ($[W_{\text{or}}] \rightarrow [W_{\text{sp}}]$)

13

Na równe $\bar{a} = \langle a_0, a_1, \dots, a_{n-1} \rangle$
 Obliczającą $\bar{y} = \langle y_0, y_1, \dots, y_{n-1} \rangle$ t.i. $y_k = \sum_{j=0}^{n-1} a_j (w_n^k)^j$
 (wektor y j.u. nazywamy Dyskretną Transformacją Fouriera wektora a)

$$V_n \begin{bmatrix} w_n^{00} & w_n^{01} & \dots & w_n^{0(n-1)} \\ w_n^{10} & w_n^{11} & \dots & w_n^{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_n^{(n-1)0} & w_n^{(n-1)1} & \dots & w_n^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

V_n (macierz Vandermonda) \bar{a} \bar{y}

W i-tym wierszu i j-tą kolumnie V_n jest w_n^{ij} .



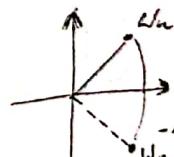
Znajmy \bar{y} . Chcemy obliczyć \bar{a} .

- V_n jest odwracalna.
 W i-tym wierszu i j-tą kolumnie macierzy V_n^{-1} jest $\frac{w_n^{-ij}}{n}$.

Po mnożeniu $\frac{1}{n}$ przez macierz V_n^{-1} na miejscu i,j będzie $(w_n^{-1})^{ij}$.

↗ to tu są $n-1$ pierwiastki pierwotny z jedności.

CZYLI DO OBLICZENIA $\frac{1}{n} [V_n^{-1}] \cdot \bar{y}$ MOŻNA ZASTOSOWAĆ FFT.



PDF o FFT

DFT: mapuje wektor \bar{a} na \hat{a} : $\hat{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \rightarrow \hat{a} = \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \vdots \\ \hat{a}_{n-1} \end{pmatrix}$, gdzie $\hat{a}_j = \sum_{k=0}^{n-1} a_k w_n^{jk}$ $j = 0, 1, \dots, n-1$

A jaka macierz:

$$V_n \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w_n & w_n^2 & \dots & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & \dots & w_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_n^{n-1} & w_n^{2(n-1)} & \dots & w_n^{(n-1)(n-1)} \end{bmatrix}$$

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \vdots \\ \hat{a}_{n-1} \end{bmatrix}$$

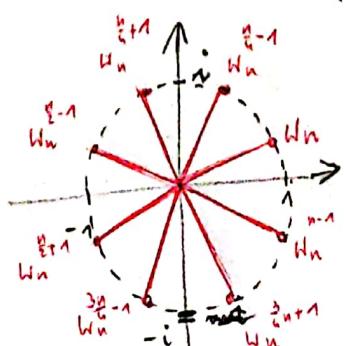
$$\hat{a}_k = A(w_n^k)$$

↑ suma esp. \hat{a} ujemnych
 plus kolejne pierwiastki jednostki

Zakładamy, że n jest parzyste.

W FFT wykorzystywane własności w_n :

$$\begin{aligned} w_n^n &= 1 & w_n^{\frac{n}{2}} &= -1 \\ w_n^{\frac{n}{2}k} &= w^k & w_n^{\frac{n}{2}+k} &= -w_n^k \end{aligned}$$



macierz dla transformacji Fouriera

$$w_n = e^{\frac{2\pi i k}{n}}$$

Dział i zapisiaj

$[W_{sp}] \rightarrow [W_{kt}]$

14

Algorytm działania $A(x)$ na dwa widomia: $A_p(x) \in A_n(x)$ stopnia $\frac{n}{2} - 1$.

W ten sposób redukując problem obliczenia $w_n^0, w_n^1, \dots, w_n^{n-1}$ dla $A(x)$ do dalszych kroków:

1. obliczenie $(w_n^0)^2, (w_n^1)^2, \dots, (w_n^{n-1})^2$ dla $A_p(x) \in A_n(x)$
2. oblicze FFT dla $A_p(x) \in A_n(x)$???

Zmiana odnotowa $[W_{kt}] \rightarrow [W_{sp}]$

Ciąg policyjny $a = \sqrt[n]{\cdot} \cdot \hat{a}$, czyli ciąg zadeń uspójczalnić
widomianie $A(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \dots + a_{n-1} \cdot x^{n-1}$.

Ten mowa to mówiąc $\mathcal{O}(n \log n)$.

$$\sqrt[n]{\cdot}^{-1} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_n^{-1 \cdot 1} & w_n^{-1 \cdot 2} & \cdots & w_n^{-1(n-1)} \\ 1 & w_n^{-2 \cdot 1} & w_n^{-2 \cdot 2} & \cdots & w_n^{-2(n-1)} \\ \vdots & & & & \\ 1 & w_n^{-(n-1) \cdot 1} & w_n^{-(n-1) \cdot 2} & \cdots & w_n^{-(n-1)^2} \end{bmatrix}$$

przykład mówiąc

zadeń w miarach i_j

obecnie: $\frac{w_n^{-i_j}}{n}$,

ale wygrywając $\frac{1}{n}$.

Skoro wantne da się zapisać w taki
postaci jako $[W_{sp}] \rightarrow [W_{kt}]$, to mamy ic
py $[W_{kt}] \rightarrow [W_{sp}]$ równie mniej złożonej
tak same algorytm!

DRZEWNCE (drzewo + kopiec)

(Tree)

Drzewo posortowane binarnego. Kiedy ~~węzeł~~ ma klucz i priorytet.
Priorytet dla klucza wybierany losowo.

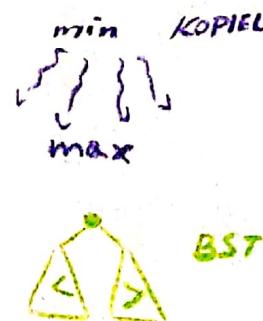
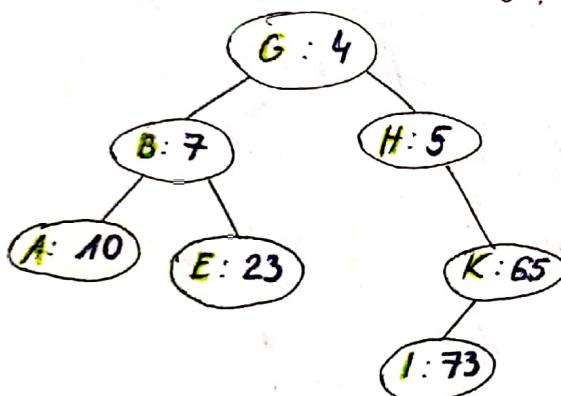
Klucz * spełniają poniższe BST:

- v w lewym poddrzewie $u \Rightarrow v.\text{key} < u.\text{key}$
(u jest v)
- v w prawym poddrzewie $u \Rightarrow v.\text{key} > u.\text{key}$

Priorytety spełniają poniższe kopiec min (mocne klucz max):

- v synem $u \Rightarrow v.\text{priorytet} > u.\text{priorytet}$

Priorytet drzewa: (klucz : priorytet)



- Dla każdego zestawu kluczy i priorytetów istnieje drzewo.
- Jeśli klucz i priorytet są różne, to drzewo jest unikalne.

Operacje:

- find - jak w BST
- insert (k): 1) ustawienie jaka w BST
2) losujemy dla k priorytet
3) rotacjami przesuwamy k tak, by przybrać poniżej kopiec (jaki zaburzy)
- delete (k): 1) find (k)
2) rotacjami przesuwamy k do końca
3) odjmujemy liczbę z k

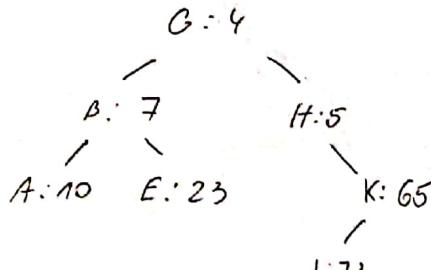
Oczekiwana wysokość drzewa: $O(\log n)$

Find: $O(\log n)$

Insert: $O(\log n)$

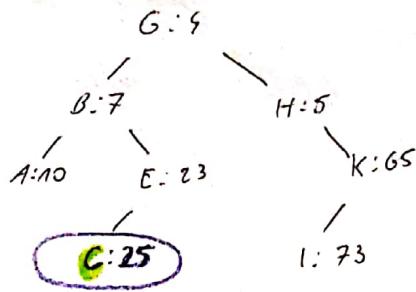
Delete: $O(\log n)$

insert



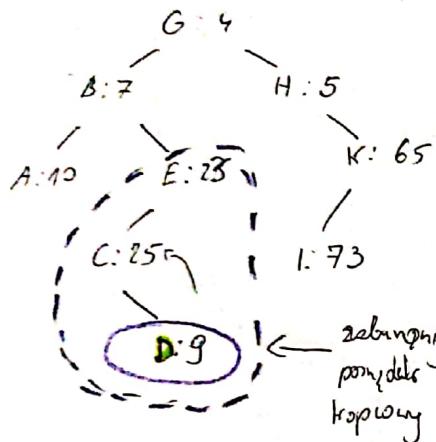
insert(C: 25)

- wypisuj
- wstaw C w g
- porządkuj BST
- jeśli 25 większe niż wszystkie kazywać to go przenieść



insert(D: 9)

nic relacjny przekłada kopia



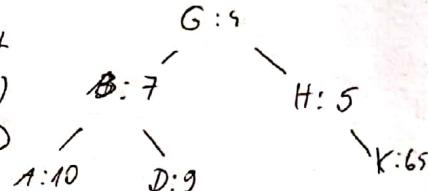
wypisuj
przydłu
kopiowy

E: 23

D: 9

F: 2

insert
(F: 2)



F: 2

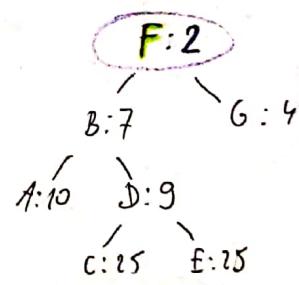
wypisuj
przydłu

D: 9

B: 7

X

→

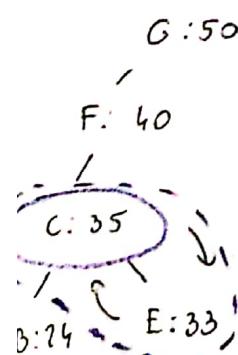


I: 73

C: 15

E: 23

ddlc (zostając po priorytetach?)



ddlc(C)

1: 25

• przenieś C
do liścia

• usuń C

G: 50

F: 40

E: 33

C: 35

B: 24

A: 21

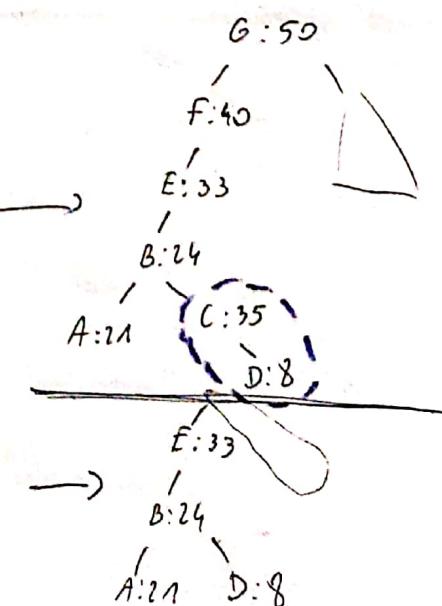
D: 8

(do kopię max)

(rolując z dnia chwil)

(o wyższym priorytecie)

wysiąkać E



DRZEWA DECYZYJNE

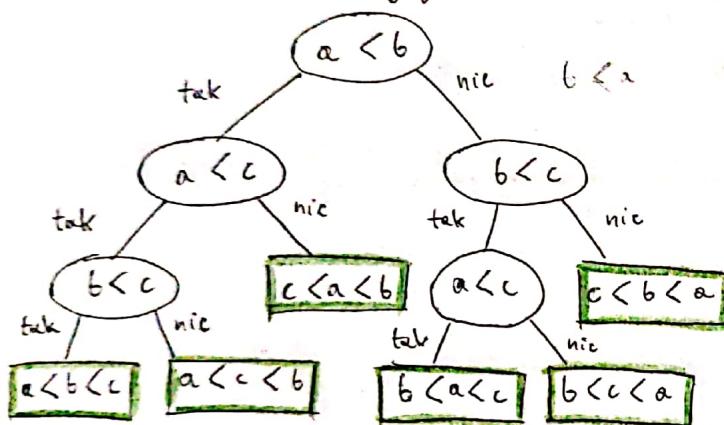
17

Ścieżka do reprezentacji działania algorytmów, które na el. ciągu wejściowego wykonują jedynie operacje porównania.

- wielodziałki wejściowe reprezentują porównania
- liść reprezentuje wynik obliczeń
- krawędź reprezentuje działania wykonane przez alg. pomiędzy porównaniami

Wykorzystuje się je do wyznaczenia dolnej granicy na liczbę operacji, które musi wykonać każdy alg. rozwiązywający dany problem.

np. optymalne drzewo decyzyjne dla alg. sortującego ciągu 3-el. (a, b, c)



n - rozmiar danych
Dla każdego rozm. danych n mamy inne drzewo decyzyjne D_n .

FAKT

Niektóre alg. sortujące, $\{D_i\}_{i=1}^{\infty}$ - adwersyjna
ma rodząca drzewa decyzyjne.
Wtedy drzewo D_n posiada min. $n!$ liści.

$$\leftarrow n = 3, n! = 6$$

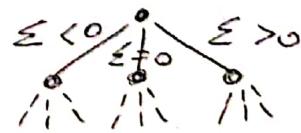
$\Rightarrow D_n$ musi mieć wysokość $\Omega(\log n!)$,
czyli $\Omega(n \log n)$ (z użyciem Stirlinga)

Wysokość drzewa odpowiadająca liczbie porównań
wykonanych w napisanym przypadku, zatem:

[Kiedy alg. sortujący za pomocą porównań
musi wykonać unilogiczną porównanie, $c > 0$.]

LINIOWE DRZEWA DECYZYJNE

Wprowadzony model drzew decyzyjnych - 3-ärne,
w wielodziałku pytamy co $>$, $=$, $<$.



$$\sum_{i=1}^n a_i x_i \stackrel{?}{=} 0$$

a_1, \dots, a_n - stałe

x_1, \dots, x_n - dane

PROBLEM : ELEMENT UNIQUENESS

18

Dane: $x_1, \dots, x_n \in \mathbb{R}$

Wynik: $\begin{cases} \text{TAK}, & \forall i \neq j \quad x_i \neq x_j \\ \text{NIE}, & \text{inne} \end{cases}$

Model: na danych x_1, \dots, x_n wykonyujemy tylko porównania.

Myslmy o danych jak o punktach $\in \mathbb{R}^n$.

WYKORZYSTAJEMY LINIOWE
DRZEWA DECYZYJNE.

Problem: sortujemy liste

i sprawdzamy czy wystepują takie same elementy.

Oznaczenia: D_n - liniowe drzewo decyzyjne dla n danych

v - wierzchołek w D_n

$S(v)$ - zb. punktów $\in \mathbb{R}^n$, z którymi dochodzimy do wierzchołka v w D_n .

Obserwacja: Zbiór $S(v)$ jest zbiorem wykłtym. (czyli jest taki spójny)

Niech $P_0 = \langle 1, 2, 3, \dots, n \rangle$ (P -rodzina punktów)

P_i - i -ta permutacja ciągu $\in P_0$ ($n!$ punktów)

FAKT: $\forall_{i \neq j} \quad p_i, p_j$ w drzewie decyzyjnym (liniowym) muszą dojść do różnych liści.

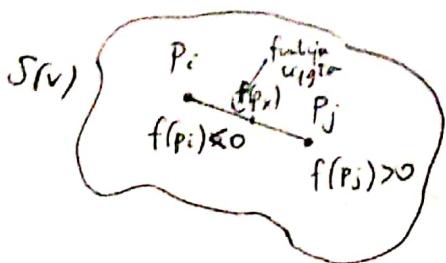
Dowód (na prost):

Niech $i \neq j$. ~~Wszystkie~~ Z p_i i p_j dochodzą do tego samego liścia (wierzchołka v).

Wtedy, i.e. $S(v)$ jest zbiorem spójnym, więc p_i i p_j muszą połączyć kryzż ułamek leżący w $S(v)$.

$p_i: x_1, \dots, \overset{\text{funkcja ciągła}}{k}, \dots, x_n$
 $p_j: x_1, \dots, \overset{\text{funkcja ciągła}}{k}, \dots, x_n$ (permutacje $\langle 1, \dots, n \rangle$)

Niech k -min liczb k jest nie inny punkt w p_i i p_j . Zatem w O_{xy} jest więcej od k . Niech $f(y_1 \dots y_n) = y_r - y_s$. Wtedy, i.e. ~~Wszystkie~~ $f(p_i) < 0$, a $f(p_j) > 0$.



f jest funkcją ciągłą.

Na odcinku $p_i \dots p_j$ musi istnieć punkt p_x t.i. $f(p_x) = 0$, a w.p. s-ta i rota zg. taki sam ($y_r - y_s = 0$).

Algorytm powinien dać odp. NIE, a nie TAK, ~~wszystkie~~

GRA Z ADVERSARZEM

(2 graczy: algorytm i adwersarz)

- Cel:
- Algorytm: wskazanie $i, j \in \mathbb{N}$ t.w. $a_i = \min(A)$, $a_j = \max(A)$
 - Adwersarz: zmienianie algorytmu do $\geq \lceil \frac{3}{2}n - 2 \rceil$ porównan

Sytuacja pojęcia:

- Algorytm nie zna A , zna $n = |A|$
- Adwersarz twierdzi, iż zna trudny zbiór A

- Ruchy:
- Alg. zadaje pytanie: $a_k < a_r$
 - Adwersarz odpowiada na to pytanie

Koniec gry:

- Alg. podaje indeks i oraz j
- Adwersarz odstanie dane

W trakcie gry Adwersarz pamięta elementy:

$$A = \{a_i : \text{nie brąży uchwał w porównaniach}\}$$

$$B = \{b_i : \text{brązy uchwał w porównaniach i wrogie wygrany (gdy wieksze)}\}$$

$$C = \{c_i : \text{brązy uchwał w porównaniach i wrogie przegryw}\}$$

D = pozostałe

Na początku: $|A| = n$, $|B| = |C| = |D| = 0$

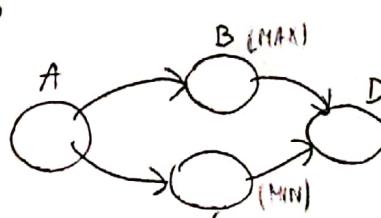
Na koniec: $|A| = 0$, $|B| = |C| = 1$, $|D| = n - 2$

Strategia Adwersara:

- (i) ustala początkowe wartości dla a_1, \dots, a_n
- (ii) w razie pytania brąże te wartości zmieniać, aby tak by zbydu łatwiej odpowiedzieć tym pytaniom
- (iii) elementy a_1, \dots, a_n będą umieszczane w zbiorach A, B, C, D

Ad. (iii) Adwersarz brąże utygnie niemniej:

$$\begin{matrix} A & H & H & H & H \\ a \in A & b \in B & c \in C & d \in D \end{matrix} \quad b > a > c \quad b > d > c$$



typ porównania	A	B	C	D
AA	-2	+1	+1	0
AB	-1	0	+1	0
AC	-1	+1	0	0
AD	-1	+1	-	-
BB	0	-1	0	+1
BC	0	0	0	0
BD	0	0	0	0
CC	0	0	-1	+1
CD	0	0	0	0
DD	0	0	0	0

- Aby elementy opisane w A podana $\lceil \frac{n}{2} \rceil$ położone (przygry w sobie, pośrodku B , otwarcie do C).
- Aby $n-2$ el. trafili do D podana $n-2$ położenie.
(B to tyle elementów ma trafić do D . Mały x el. w B , wiec wyrażenie $x+1$, y el. w C , wiec wyrażenie $y+1$; $x+y = n$)

Pozkłódz

$$A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

- $a_5 < a_7 \rightarrow B = \{7\}, C = \{5\}$
- $a_1 < a_4 \rightarrow B = \{7, 4\}, C = \{5, 1\}$

↑
adresan może zmienić 4 na 14, bo nie zależy to od położenia dotyczących odpowiedzi

Gdyby nie zmienił, to przy położeniu $a_4 < a_5$ licy 4 i 5 trafiły do D w jeden tun (nic jest to w istocie adresowania, bo on ustawia).

RÓWNANIA REKURENCYJNE

$$\bullet T(n) = 2T\left(\frac{n}{2}\right) + \gamma$$

$$n = 2^k$$

$$T(2^k) = 2T(2^{k-1}) + \gamma$$

$$T(2^{k-1}) = 2T(2^{k-2}) + \gamma$$

$$T(n) = 2T(2^{k-1}) + \gamma =$$

$$= 2(2T(2^{k-2}) + \gamma) + \gamma$$

$$T(n) = 2T(2^{k-1}) + c$$

$$= 2(2T(2^{k-2}) + c) + c$$

$$= 2(2(2T(2^{k-3}) + c) + c) + c$$

$$\underbrace{2(4T(1) + 3c)}_{= 8T(1 + 7c)} + c$$

$$= 2(2(2(2T(2^{k-4}) + c) + c) + c) + c$$

$$2(8T(1 + 7c) + c) + c$$

$$= 16T(2^{k-4}) + 15c$$

$\uparrow 2^4$

→
 $T(n) = 2^k \cdot T(2^{k-k}) + (2^k - 1) \cdot \gamma$

?
 $T(n) = 2^k \cdot T(1) + (2^k - 1) \cdot \gamma$

$2^k = n$
 $T(n) = n \cdot T(1) + (n-1) \cdot \gamma = O(n)$

$$T(n) = 2^k \cdot T(2^{k-k}) + (2^k - 1) \cdot \gamma$$

$$= 2^k \cdot T(1) + (2^k - 1) \cdot \gamma$$

$$2^k = n$$

$$T(n) = n \cdot T(1) + (n-1) \cdot \gamma = O(n)$$

21

$$\begin{aligned} 2^3 &= 8 \\ \log_2 8 &= 3 \\ 4^k &= n \\ \log_4 n &= k \end{aligned}$$

$$T(n) = T\left(\frac{n}{4}\right) + \gamma$$

$$n = 4^k \rightarrow k = \log_4 n$$

$$T(4^k) = T(4^{k-1}) + \gamma$$

$$T(4^{k-1}) = T(4^{k-2}) + \gamma$$

$$T(n) = T(4^{k-1}) + \gamma =$$

$$= (T(4^{k-2}) + \gamma) + \gamma = \dots$$

$$= (((T(1) + \gamma) + \gamma) + \dots + \gamma) + \gamma$$

$$= T(1) + \gamma k = \begin{array}{l} \text{mnożymy} \\ \text{wykonujemy} \\ \text{kroki} \\ \text{wzwyżku} \end{array}$$

$$= T(1) + \gamma \cdot \log_4 n = O(\log_4 n)$$

$$T(n) = T\left(\frac{n}{i}\right)$$

$$\rightarrow n = i^k$$

$$T(n) = T(\sqrt{n})$$

$$n = 2^k$$

$$U(k) = T(2^k)$$

Scanned with CamScanner

$$\bullet T(n) = \begin{cases} 1 & , n=2 \\ 4T(\sqrt{n}) + O(1) & , n>1 \end{cases}$$

$$\sqrt{n} = n^{\frac{1}{2}}$$

$$\circ n = 2^k \rightarrow k = \log_2 n$$

$$T(2^k) = 4T(2^{\frac{k}{2}}) + O(1)$$

$$\circ U(k) = T(2^k)$$

$$U(k) = 4 \cdot U\left(\frac{k}{2}\right) + O(1)$$

$$U\left(\frac{k}{2}\right) = 4 \cdot U\left(\frac{k}{4}\right) + O(1)$$

$$= 4 \cdot U\left(\frac{k}{2}\right) + O(k^0)$$

$$= 4 (4 \cdot U\left(\frac{k}{4}\right) + O(1)) + O(1)$$

= (...) *wykonaj log₂ k iterację*

$$= 4^{\log_2 k} \cdot U(1) + (4^{\log_2 k} - 1) \cdot O(1)$$

tylko iterację

$$= \log_2^2 n \cdot U(1) + (\log_2^2 n - 1) \cdot O(1)$$

$$= \log_2^2 n \cdot 1 + (\log_2^2 n - 1) \cdot O(1)$$

$$\approx O(\log_2^2 n)$$

$$\begin{aligned} 4^{\log_2 k} &= 2^{2 \cdot \log_2 k} = \\ &= 2^{\log_2 k^2} = k^2 = \\ &= (\log_2 n)^2 \end{aligned}$$

$$\bullet T(n) = T(\sqrt{n}) + O(1) \quad \sqrt{n} = n^{\frac{1}{2}}$$

$$\circ n = 2^k \quad k = \log_2 n$$

$$T(2^k) = T(2^{\frac{k}{2}}) + O(1)$$

$$\bullet U(k) = T(2^k)$$

$$U(k) = U\left(\frac{k}{2}\right) + O(1)$$

$$U\left(\frac{k}{2}\right) = U\left(\frac{k}{4}\right) + O(1)$$

$$= (U\left(\frac{k}{4}\right) + O(1)) + O(1)$$

= (...) *wykonaj log₂ k iterację*

$$= ((U(1) + O(1)) \dots) + O(1)$$

$$?$$

$$= U(1) + \log_2 k \cdot O(1)$$

$$= T(2) + \log_2(\log_2 n) \cdot O(1) \approx O(\log_2(\log_2 n))$$

✓

KLASY ZŁOŻONOŚCI

23

Logarytm iterowany:

$$\log^* n = \begin{cases} 0 & , n \leq 1 \\ 1 + \log^*(\log n) & , n > 1 \end{cases}$$

Funkcje w notacji asymptotycznej od najmniej rosnących do największych:

$$O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(\log_2 n) < O(2^n) < O(n!)$$

Zadanie:

Porównaj następujące klasy funkcji: $O((\log^* n)^{\log n})$

$$O((\log n)^{\log^* n})$$

~~$O(\log n)$~~

WAŻNE

$$\frac{\log_e b}{\log_a b} = \log_a b$$

ALGORYTHM WYBORU K-TEGO ELEMENTU (MEDIANY)

24

Stosowane np. przy wyborze piwota w quicksort.

QUICKSORT

sortuje od el. p-ręgo do n-ręgo

Quicksort ($A[1\dots n]$, p, r)
 if $r-p$ then insertsort ($A[p\dots r]$)
 else choosepivot (A, p, r)
 $q = \text{partition}(A[p, r])$ // wybór piwota
quicksort (A, p, q) // partitionowania elementów tabl. A
quicksort ($A, q+1, r$) (zaktasany, i.e. pivot znajduje się w $A[p, r]$)

partition ($A[1\dots n]$, p, r) // partitionowanie el. tabl. A, taki by el. \leq pivot
 $x = A[p]$ // pivot
 $i = p-1$ // indeks pivot
 $j = r+1$ i za tabl. A
 while $i < j$ do partitioning dopoki warunki nie będą true, to stop
 repeat $j = j-1$ until $A[j] \leq x$
 repeat $i = i+1$ until $A[i] > x$
 if $i < j$ then swap ($A[i], A[j]$)
 else return j

up-	0 1 2 3 4 5 6 7
	9 7 5 M 2 14 3 6
	6 7 5 M 2 14 3 9

$$\begin{aligned} x &= 6 \\ i &= 0-1 = -1 \\ j &= 7+1 = 8 \end{aligned}$$

while $i < j$

1) repeat $j = j-1$ until $A[j] \leq x$

$$A[6] = 3 \leq 6 \quad X$$

repeat $i = i+1$ until $A[i] > x$

$$A[0] = 6 > 6 \quad V$$

$$A[1] = 7 > 6 \quad V$$

$$A[2] = 5 > 6 \quad X$$

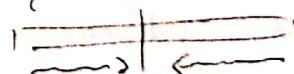
if $i < j$ (2 < 7) V

swap ($A[i], A[j]$)

0 1 2 3 4 5 6 7
6 7 3 M 2 14 3 9

ZASADA

- zunikać uniechy po prawej, uniknąć uniechy po lewej (od pivotu) i je u sile zatrzymać



2) repeat $j = j-1$ until $A[j] \leq x$

$$A[6] = 3 \leq 6 \quad V$$

$$A[5] = 14 \leq 6 \quad X$$

repeat $i = i+1$ - - -

$$A[3] = M \geq 6 \quad V$$

$$A[4] = 2 \geq 6 \quad X$$

if $i < j$ (4 < 5)

swap ($A[i], A[j]$)

0 1 2 3 4 5 6 7
6 7 3 M 2 14 3 9

3) - - -

na tym krańcu

jeżeli zatrzymać,

to $j = 4$, wtedy

$A[j] = 14$, a $i = 5$,

wtedy $A[i] = 2$, więc

zatrzymać j

Wybór pivota determinującą stałość. Pasywnyca stałość
mocie wynieść $O(n^2)$, optymistyczna $O(n \log n)$.

↑ up. gł. działa tabl. $\xrightarrow{\text{dla}} \rightarrow$ w bardziej trudnym

Wybór pivota:

- musi być szybki ~~100%~~
 - deterministyczny lub losowy
- ↓
up. pivots el. tabl.

1^o Mocie mieć pełna, up. tablica może być już posortowana \rightarrow czas $O(n^2)$ (ale dla większości losowych działa szybko)

2^o up. choosepivot($A[1..n]$, p, r)
 $i = \text{random}(p, r)$
 swap $A[p], A[i]$

Pp. Czasu bieżącego zaznaczono na czerwono.
 Teraz nie istnieje lepsze niż gorsze dane wejściowe.

* Modyfikacja

Wybranie pivota z 3 el. tabl. :- (determin.) up. pivory, średnicy, ostatni
 - (losowy) up. 3 losowe ~~100%~~ el. tablicy

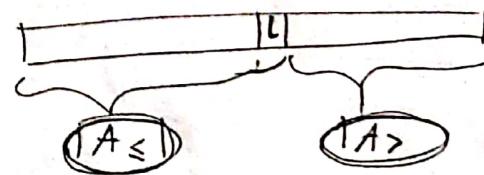
Sredni koszt: $O(n \log n)$

Algorytm Hoare'a

Dzieli w podobny sposób do quicksorta (partycji).
 Szukamy k-tej co do wielkości liczby.

Kroki:

- 1. ~~losuj pivot~~ pivot c
 rozbiór A (n -elementowego)
- 2. podziel A na liczby $> c$: $\leq c$
- 3. - jeśli $|A \leq| > k$, to rekurencyjnie szukaj k-tego el. w $A \leq$
 - jeśli $|A \leq| < k$, to rekurencyjnie szukaj $k - |A \leq|$ -tego el. el. w $A >$



Czas:
 pasywny: $O(n^2)$
 optymistyczny: $O(n)$

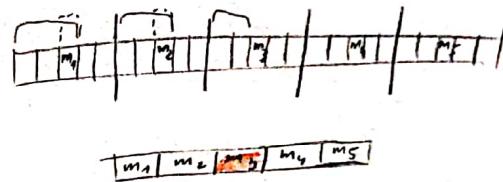
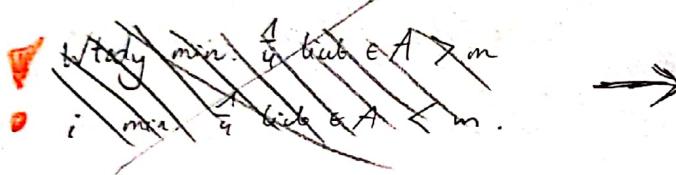
Algorytm magicznych piętek (median a median)

Cel: znaleźć k -ty element do kolejności licby
To ulepszona wersja algorytmu Hoare'a.

A - tab. niesortowana

Kroki:

- dzielimy A na części: 5-elementowe (ost. może być więcej)
- ①
 - każdy z podzieliów sortujemy i bierzemy z nich medianę (3-cie element) (do momentu ≤ 5 el.)
 - wykorzystując rekurencyjny algorytm magicznych piętek dla zboru median A_m (gdzie $k \leq 5$ el.)
 - otrzymujemy medianę m i względem niej dzielimy el. z A na
 $\begin{matrix} < m \\ , = , \\ > \end{matrix}$ od m



Sortujemy k -tej co ob kolejności licby.

Mamy już medianę. Teraz dzielimy podzbiorze jedna z alg. Hoare'a, z tym że wykorzystujemy rekurencyjny alg. magicznych piętek:

- dzielimy A na liczby $<, =, >$ i $= m$
- ③
 - jeśli $k \leq |A|_< 1$, to wykorzystuję alg. dla $(A|_<, k)$
 - jeśli $k \leq |A|_< + |A|_< 1$, to ~~zwracamy medianę~~ zwracamy m (bo $k = |A|_<$)
 - jeśli $k > |A|_< + |A|_< 1$, to wykorzystuję alg. dla $(A|_>, k - |A|_< - |A|_< 1)$

Złożoność: **LINIOWA** (nowe przyrostowe)

$$T(n) \leq \Theta(n) + T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right)$$

znajdowanie
mediany piętek
(jakoś szybko zlokalizować)
(dzielić i zwierciadlać)

rekurencyjne
medianę zboru A_m
(bo nigdy dostać
na 5 zbiór A_m)

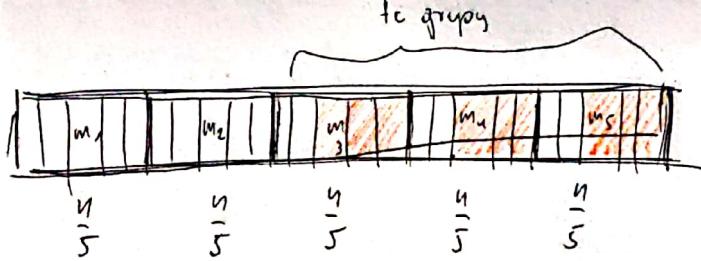
$$\textcircled{3} \frac{2}{5}n + 4$$

ustępowa
strona

(taka ilu el. będzie oglądane)
wykorzystując rekurencyjne i po kolei:
porównanie z k .

$$\text{czas: } T(\max(|A|_<, |A|_>))$$

~~Uwaga: nie ma el. $i = m$,
dziego $A|_> \leq \frac{4}{5}n$ i $A|_< \leq \frac{1}{5}n$.
Oczywiście $\frac{1}{5}n \leq \frac{4}{5}n$.~~



- Jest najszybszy $\frac{1}{2} \cdot \lceil \frac{n}{5} \rceil$ grup, których mediana $\geq m$. $(\frac{1}{2} \cdot \lceil \frac{25}{5} \rceil = 3)$
 - W każdej z nich znajdziemy się min 3 el $> m$.
 - Czyli tylekto el. jest $3 \cdot (\lceil \frac{n}{5} \rceil - 1)$ to jest m?
- Daje to $\frac{3}{2} \cdot \lceil \frac{n}{5} \rceil - 3 = \frac{3}{10} n - 4$

Pozostałych elementów jest wtedy: $n - (\frac{3}{10} n - 4) = \frac{7}{10} n + 4$

Lazy Select (probkowanie losowe)

S - 26. et.



\downarrow R - probkowanie losowe, niebyt liczne, posortowane

\downarrow
sortowany P

Algorytm:

1. Wybieramy losowo, niezależnie, 2 podziałkiem: próbki R rozłożone w $n^{\frac{3}{4}}$ ($\sqrt[4]{n^3}$) elementów z S.

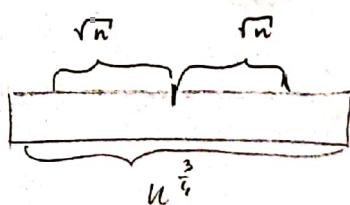
2. Sortujemy R w czasie $\Theta(n^{\frac{3}{4}} \log n)$.

3. Wyświetlamy elementy L i H:

$$L \leftarrow R[\frac{1}{2} \cdot n^{\frac{3}{4}} - \sqrt{n}]$$

$$H \leftarrow R[\frac{1}{2} \cdot n^{\frac{3}{4}} + \sqrt{n}]$$

R:



4. Porównujemy L i H w wybranymi el. z S:

$$l_s(L) \leftarrow |\{y \in S : y < L\}| = l \text{ d. } 2 \leq l < L$$

$$P \leftarrow \{y \in S : L \leq y \leq H\}$$

- $\frac{1}{2}$, bo dla mediany
- dla k-tego bytu k.n

5. Sprawdzamy, czy mediana 26. S znajduje się w P, czyli czy:

$$\rightarrow l_s(L) < \frac{n}{2} < (l_s(L) + |P|)$$

oraz $|P| \leq 4 \cdot n^{\frac{3}{4}} + 2$

poćwierdzenie P

Jeli nie, to PORAKA, powtarzamy alg. dla innego zb. R

6. Posortuj P

$$\text{return } P[\frac{n}{2} - (l_s(L) + 1)]$$

- k-th element
- k

Jeli nie zakończ PORAKA, to wynik otrzymujemy w czasie $O(n)$

IZOMORFIZM DRZEW

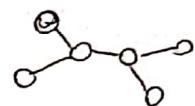
(olla dno nieukonczonej problemu
spr. czy dwa drzewa są izomorficne
jest NP-zupełny)

28

Dwoje ukonczone
(binarne)

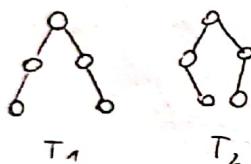
- dwoje 2 wyróżnionym wiernostkiem (korzeń), będącym stopnia co najwyżej 2 (mogące 2 spadać)

Dwoje nieukonczone
(binarne) - dwoje, w których każdy z wiernostek jest st. 1 lub 3, np.



IDEA SPRAWDZANIA IZOMORFIZMU

Aby sprawdzić, czy dwa drzewa T_1 i T_2 są izomorficne, idącą od ich najniższych poziomów do najwyższeego i sprawdzając, czy na każdym poziomie mają tyle samo wiernostek tego samego typu (wiernostki są tego samego typu, jeśli podkawa w nich reprezentują izomorfizm), np.



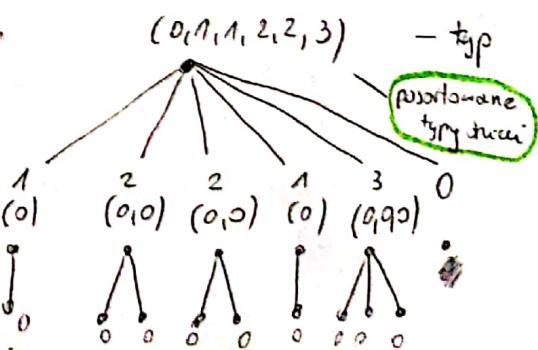
\Leftrightarrow izomorficzne

wielowy przekształcenie leksykograficzne :
 $(0,0), (0,0,1,3), (0,0,2),$
 $(0,1,0,3), (0,1,1), \dots$

Algorytm:

Zastanowmy się, T_1 i T_2 są tylor sami wykrojci i mają taką samą liczbę krawędzi poziomów.

1. $\forall v \in \text{wurz} T_i \quad \text{kod}(v) \leftarrow 0$
2. $\text{for } j = \text{depth}(T_1) \text{ do } 1$
3. $S_i = \{b. \text{ wiernostek } T_i \text{ z poziomu } j \text{ nie}\}$
będące liśćmi
4. $\forall v \in S_i \quad \text{key}(v) \leftarrow \text{vector } \langle i_1, \dots, i_k \rangle \text{ t.j. :}$
 - $i_1 \leq i_2 \leq \dots \leq i_k$ //permutacj $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$
 - v ma k synów u_1, \dots, u_k
 $i_l = \text{kod}(u_l)$
5. $L_i \leftarrow \text{lista wiernostek z } S_i \text{ posortowana leksykograficznie}\}$
wg wartości key
6. $L_i' \leftarrow \text{otynymi w ten sposób uporządkowaniem ciąg wertówek}$
7. $\text{if } L_1' \neq L_2' \text{ return ("nieizomorficzne")}$
8. $\forall v \in L_i \quad \text{kod}(v) \leftarrow 1 + \text{rank}(\text{key}(v), \{\text{key}(u) | u \in L_j\})$
9. $\text{Na permutacji wątków lista z poziomu } j \text{ drzewa } T_i$
10. $\text{return ("izomorficzne")}$



$(0) - 1$

$(0,0) - 2$

$(0,0,0) - 3$

(\dots)

Czas linowy : $O(n)$

(sortowanie leksykograficzne działa
w czasie $O(n \log n)$ dalej)

UNION - FIND

29

- U - zbiór stanowiący (uniwersum)
- rozłączne podzbiory U
- wszystko kiedy $x \in U$ jest w jednej z podzbiory $\{x\}$

Operacje:

- FIND(x) - wynik: podzbiór, do którego należy x (sprawdzenie, czy element należy do $x \in U$)
- UNION(A, B, C) - połączenie podzbiorów $A; B$ w C ($C \leftarrow A \cup B$),
 A, B - rozłączne
podzbiory U
uniując $A \cap B$

Chęć reprezentować strukturę sylabko złożoną zigg i instrukcji UNION i FIND.

Zastosowanie: Konstrukcja MST (tworząc zbiory rozłączne)
(alg. Kruskala)

W rozumieniu naięźwym koszt UNION(\dots) to $O(n^2)$, chęć zrybać.

1. Implementacja listowa: • Każdy zbiór reprezentowany jako lista swoich elementów.

- Reprezentantem zbioru jest pierwszy element na liście.
- Każdy el. listy ma wskaznik na reprezentanta

Kost: Find(x): $O(1)$, bo ↑ (representanta)

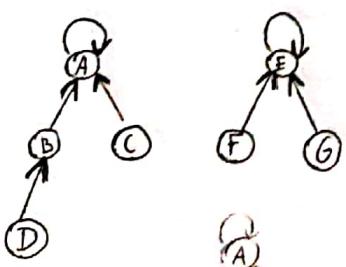
Union(x, y): $O(n \log n)$, zatem dłuższy

czytaj dwa zbiory złożone
 x i y . Mapujemy na mniejszy
zbiór robimy (st. chom) operację
MAKE-SET, a później połączyc
dłuższy do drugiego zbioru (UNION).
krótszy listę do dłuższej i uaktualniaj
wskaznik na reprezentanta
(kont. napisu dalej tut jest stary)

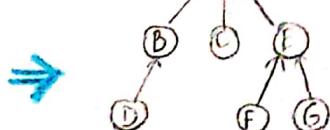
2. Struktury drzewiste: • Las zbiorów rozłącznych (kiedy podzbiór to drzewo)

- syn ma wskaznik na ojca
- koren' ma wskaznik na siebie

• Kost: Union(\dots): $O(1)$, bo przenosimy wskaznik
kronika jednego z drzew na koren' innego



Find(x): $O(\log n)$, bo Union wykorzystywany
w sposób rekurencyjny (przyłączamy nizsze
ob nizszych; działa tym samym h),
jest to możliwe z nieuwzględnieniem kronika



Komprezja ścieżek

Mając stosować przy wykonywaniu operacji $\text{Find}(x)$ w str. drzewastych.

Po zindeksowaniu koniaków dla x , poznając przekroj ścieżki i kiedy niepotrzebne ścieżki ~~pozyskiwanie~~ do koniaka. Zanegować to kartę pominąć jedyne operacji $\text{Find}(1)$.

(pozyskiwanie x i to co co w drodze do koniaka od x)

(Ale predysygnowane ataki wykorzystują zły kartotekę, dlatego wykonywanie npd \emptyset)

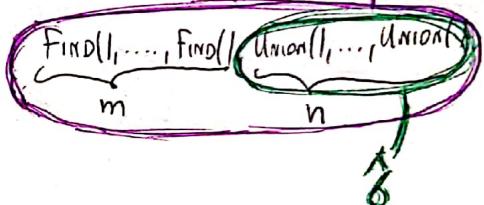
Anotacja koniaka

Niedł \mathfrak{G} to czas na operację Find i na operację Union , a \mathfrak{Z} to czas instancji Union otrzymywany po wykonywaniu wszystkich instrukcji Find .

Def: Rząd wiechostka v - wysokość w lesie powstającego po wykonyaniu \mathfrak{Z} (najdalsza ścieżka do liścia)

Dla operacji Union ~~zwykła~~ rząd drzewa jest wysokość:

- połączyc dwie o niespełnionej relacji do dwie o wyższej
- jeśli remis, to dodać 1, a rząd koniaka + 1



Wzrostosiu rządu:

- jeśli skontrahowane są elementy, to jest co najmniej $\frac{n}{2^r}$ węzłów w drzewie r , $r > 0$
- jeśli drzewo ma wysokość h (ranka reprezentanta = h), to ~~zwykła~~ ma ono co najmniej 2^h wiechostków.
- matematyczny rząd $\leq \log n$

FAKT Jeśli w trakcie wykonywania \mathfrak{Z} wiechostek v staje się potomkiem wiechostka u , to v ma rząd $<$ rządu u .

$$\begin{aligned} F(0) &= 1 \\ F(i) &= 2^{F(i-1)} \quad i \geq 1 \end{aligned}$$

$$\log^* n = \min \{ k : F(k) \geq n \}$$

grupa rządu: rząd $\boxed{\mathfrak{n}}$ jest w grupie $\log^* n$

i	$F(i)$
0	1
1	2
2	4
3	$2^4 = 16$
4	$2^{16} = 65536$
5	...

n	$\log^* n$
0	0
1	0
2	1
3	2
4	2
5	3
6	3
...	...
16	3
17	4
65536	4
65537	5

Kost δ = Kost inst. UNION + Kost inst. FIND

31

$\delta: u_1, u_2, F_1, F_2, \dots$

$$\text{Kost}(\delta) = \underbrace{\sum_{i=1}^n \text{kost}(u_i)}_{O(n), \text{bo union } \approx O(1)} + \underbrace{\sum_{i=1}^m \text{kost}(F_i)}_{\text{final}} = O(n + m \log^* n)$$

Kost pojedynczej instrukcji Find(x) = dt. ścieżki od x do korzenia.

Kost przedysyczego Find rozdzielnego pomazy:

- 1) niektóre wierzchołki ze ścieżki
- 2) samy instrukcje find

Za odnalezienie u ze ścieżki dającej Find, jeśli:

- u jest korzeniem
- u jest synem korzenia
- u ma nad u inny (najwyżej) przodkiciel niż nad ojca

Za odnalezienie porządku u obliczamy c .

Wówczas kost Find to: suma obliczeń punkt. + suma obliczeń wierszów

Pojęcia UNION

UNION(x, y)

$X: x_1, x_2, x_3, \dots, x_n$ - innego typu

Y

Operacje

MAKE-SET(x_1)

MAKE-SET(x_2)

:

MAKE-SET(x_n)

UNION(x_1, x_2)

UNION(x_1, x_3)

:

UNION(x_{n-1}, x_n)

Liczba zmienianych
drzewów

1

1

:

1

2

:

n-1

{ n operacji
w O(n)

{ n-1 operacji
na liście n-1 dla n-1

HASZOWANIE

- U - uniwersalne klucze
- funkcji przypisują klucz, a ona zwraca indeks
- wykorzystywane przy implementacji struktur

TABLICE Z ADRESOWANIEM BEZPOŚREDNIM (dla małego uniwersum)

$$U = \{0, 1, \dots, n-1\}$$

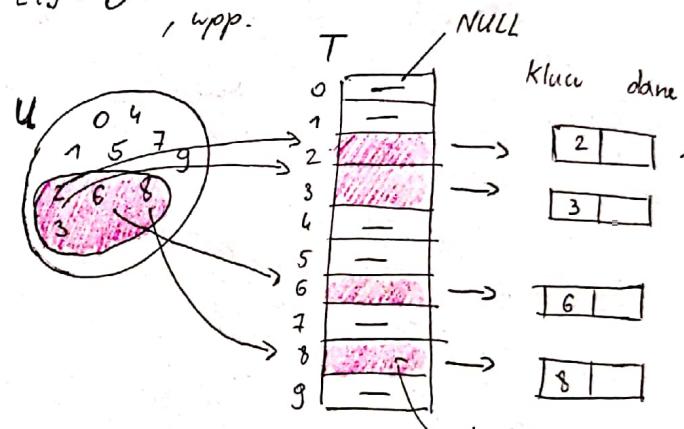
Mamy n -el. tablicę bitową, gdzie: $T[i] = 1$, gdy $i \in U$ i $T[i] = 0$, wpp.

Operacje:

Szukanie: $T[k]$

Wstaw: $T[x, key] = x$

Usunięcie: $T[x, key] = \text{NULL}$



wskazówki na el. o
kluczu k

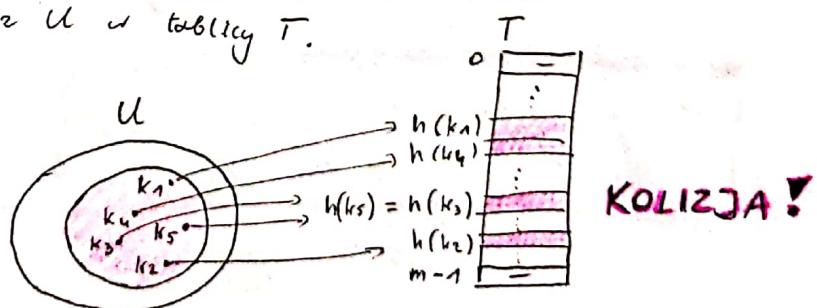
TABLICE Z HASZOWANIEM

Jeli U bardzo duże, to tabl. wielkości $|U|$ może być zbyt duża dla dostępnej pamięci lub być zbytnio marnowana dużą ilością miejsca.

Wtedy tworzymy tablicę $T[0, \dots, m-1]$, gdzie m proporcjonalne do maksymalnej wielkości stowarzyszenia.

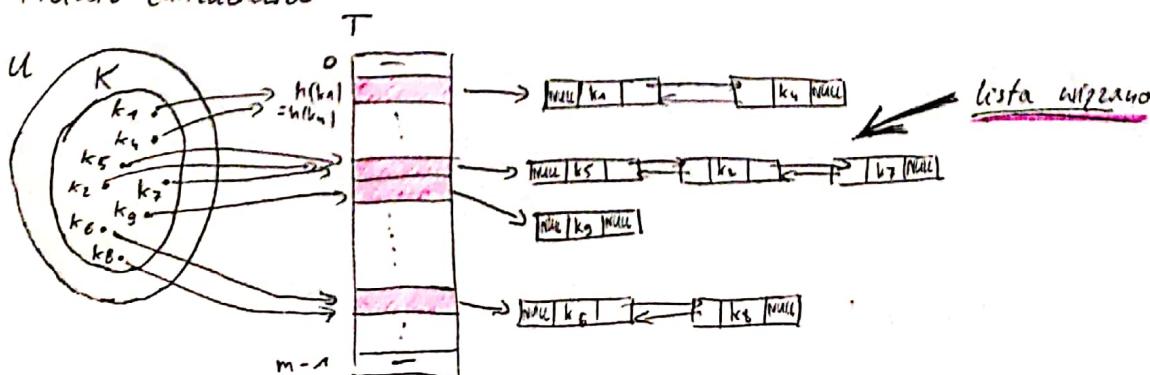
Wykonstytująca funkcja haszująca: $h: U \rightarrow \{0, 1, \dots, m-1\}$, która określa miejsce przechowywania el. z U w tablicy T .

[Klucz: k
przyjmuje w tablicy T : $h(k)$]



KOLIZJA - rozwijanie problemu

Metoda lancuchowa



PRZYKŁADOWE FUNKCJE HASZUJĄCE

- haszowanie modularne

$$h(k) = k \bmod m$$

$m \neq 2^p$, najlepiej gdy m będzie pierwszą potęgą od 2
 $\uparrow \neq 10^p$

Wtedy daje rygorystyczne kolizje, bo $h(k)$ jest równe ast. p. bitom klucza k

- haszowanie przez mnożenie

$$h(k) = \lfloor m(k \cdot A - \lfloor k \cdot A \rfloor) \rfloor$$

A - ustalona liczba $\in (0, 1)$, np. przykaz $A = \frac{\sqrt{5}-1}{2} \approx 0,618\dots$

m - dowolne, przykaz potęga 2, dla którego mnożenia

ADRESOWANIE OTWARTE

Elementy pamiętane bezpośrednio w tablicy T. Nie tracimy miejsca na pamiętanie wskaźników (jako w implementacji listowej).

Ma pojawić się inny problem - przepelnianie tablicy T.

Tablica przekształcona, gdy dany umieszcza m+1-szy element lub gdy jest ona już na tyle zapelniona, iż operacje stamkowe stają się kosztowne.

Wtedy powstaje tablica T ; zamiast funkcji hashującej dla nowego m i przypisujących elementy z użyciem nowej h.

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

Najpierw próbujemy umieścić k na pozycji $h(k, 0)$, jeśli zajdzie, to $h(k, 1)$ itd. dopóki nie znajdziemy wolnego miejsca.

Funkcja h powinna spełniać warunek: $\forall k \in U \quad \langle h(k, 0), \dots, h(k, m-1) \rangle$ jest permutacją zbioru $\{0, 1, \dots, m-1\}$

WARUNEK (PER)

FUNKCJE HASZUJĄCE W ADRESOWANIU OTWARTYM

- metoda liniowa

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

gdzie $h' : U \rightarrow \{0, 1, \dots, m-1\}$ jest pomocniczą funkcją haszującą (np. jaka przednia s. 33)

- metoda kaskadowa

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

gdzie h' jest wyciąg, $c_1, c_2 \neq 0$ state

state c_1, c_2, m dobrane tak, by zachodziła warunek (PER)

- podwójne haszowanie

$$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m$$

gdzie h_1, h_2 - pomocnicze funkcje haszujące

WAŻNE: $\forall k \in U \quad h_2(k)$ względem pierwotne $\in m$ (powinno być) (garancja liczb tablicy zostanie zapisane przekształcana)

Najlepsze haszowanie podwójne. W drugim i kaskadowym (co mniejszym stopniu) mogą tworzyć się "zlepki" - wartości obu tablicy H znajdują się dla el. z U.

HASZOWANIE UNIVERSALNE

Na początku działania programu wybieramy losowo funkcję haszującą z rodzinę funkcji haszujących (mniejsza szansa na kolizję).

Def: H - rodzina funkcji haszujących: $U \rightarrow \{0, 1, \dots, m-1\}$
 H uniwersalna, jeśli:

$$\forall x, y \in U, x \neq y \quad |\{h \in H : h(x) = h(y)\}| \leq \frac{|H|}{m}$$

Czyli szansa kolizji dla dowolnego utworzonego klucza x, y jest w średnim przypadku $\leq \frac{1}{m}$.

PRZYKŁAD RODZINY UNIIVERSALNEJ

① m -ta pierwotna, $|U| < m^{r+1}$

Dla kaidego $0 \leq a < m^{r+1}$ definiujemy funkcję h_a :

gdzie $\langle a_0, a_1, \dots, a_r \rangle$ i $\langle x_0, x_1, \dots, x_r \rangle$ są reprezentacjami

$$h_a(x) = \sum_{i=0}^r a_i x_i$$

Rodzina $H = \{h_a : 0 \leq a < m^{r+1}\}$ jest rodziną uniwersalną.

② p - l. pierwotna

$$\mathbb{Z}_p = \{0, 1, \dots, p-1\}$$

$$\mathbb{Z}_p^* = \{1, 2, \dots, p-1\}$$

m - równe tabl. hasilujacy

$$p > 100$$

$$p > m$$

Dla dowolnych $a \in \mathbb{Z}_p^*$ i $b \in \mathbb{Z}_p$:

$$hab(x) = ((ax + b) \bmod p) \bmod m$$

Rozumieć $H_{pm} = \{hab : a \in \mathbb{Z}_p^* \text{ i } b \in \mathbb{Z}_p\}$

jest rozm. uniwersalny.

Dowód:

k, l - dowolne różne klucze

$$r = (ak + b) \bmod p \quad \rightarrow \quad r - s \equiv a(k - l) \bmod p$$

$$s = (al + b) \bmod p$$

Widzimy, iż $a > 0$ i $k \neq l$
 $(a \neq 0)$ $(k-l \neq 0)$

oraz p jest l. pierwotny, czyli $r-s \neq 0$

$$r = h'(k)$$

$$s = h'(l)$$

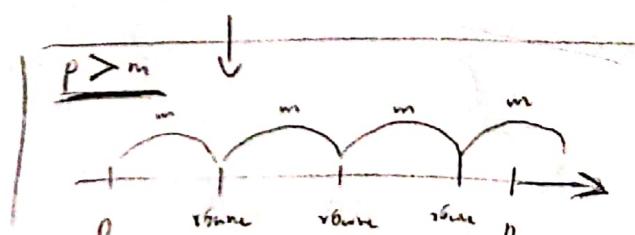
Przypodobnienie koduji (dla $hab(x)$)

- Pkt koduji klucz k i l jest równe ppb zgłoszenia parę r, s t.i.
 ↑ $r = s \bmod m$ spośród wszystkich par t.i. $s \neq r$.
- Funkcja hab jest $(p-1)p$, czyli tyle samo co par s, r t.i. $s \neq r$.
 tyle różnych a tyle różnych b .
- Każda funkcja hab przedstawia parę k, l na inną parę s, r .

Dla dowolnego (ale ustalonego) s wala tablicy r , iż $r = s \bmod m$ i $r \neq s$
 jest równa $\lceil \frac{p}{m} \rceil - 1$

$$\lceil \frac{p}{m} \rceil - 1 \leq \left(\frac{p+m-1}{m} \right) - 1 = \frac{p-1}{m}$$

Czyli ppb koduji s i r wynosi $\frac{1}{m}$.



$$\frac{p}{m} = 3,$$

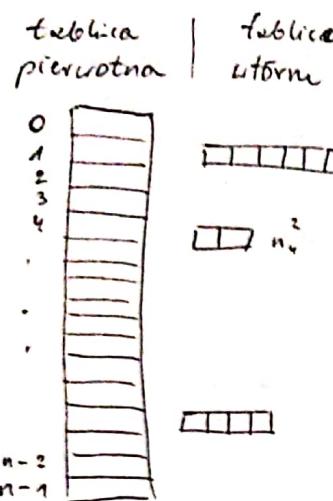
$$\lceil \frac{p}{m} \rceil = 4$$

$$\lceil \frac{p}{m} \rceil - 1 = 3$$

HARBUZANIE DWUPOZIOMOWE

36

- Chcemy stworzyć strukturę (stosunek) o rozm. $O(n)$.
- Czas tworzenia (asortywanego): wielomianowy. \uparrow
- Operacja FIND w czasie $O(1)$. # klucz



- rozmiar n^2 , gdzie n^2 to rozmiar kubetka j -tego

① Wstawiamy klucz do tablicy pierwotnej, wykorzystując wybraną funkcję z rosnącym uniwersalnym. (Dowodzimy?)

② Dla każdego niepustego kubetka klucza, który tam trafiły, rozstawiamy do tablicy rozmiaru n^2 , gdzie n^2 - liczba kluczy w kubetku (więcej pewności, iż nie będzie kolizji) wykorzystując funkcję hashującą, która ustawa d. w drugiej tablicy

Pogląd:

$$h_1(k) = ((3k+42) \bmod 101) \bmod 9$$

$$K = \{10, 22, 37, 40, 60, 70, 73\}$$

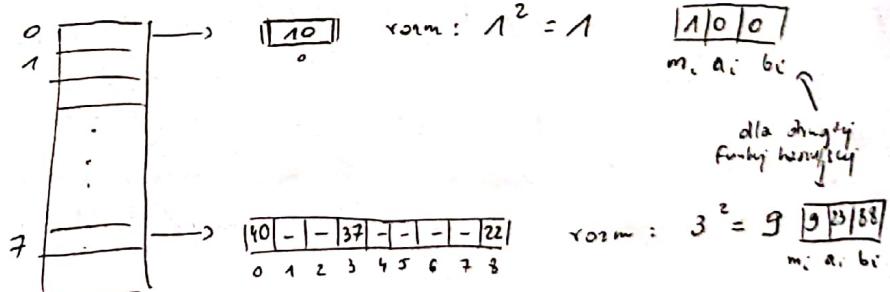
$$h_1(10) = 72 \bmod 9 = 0$$

$$h_1(22) = 7 \bmod 9 = 7$$

$$h_1(37) = 7 \bmod 9 = 7$$

$$h_1(40) = 61 \bmod 9 = 7$$

:



Ppb kolizji: $< \frac{1}{2}$. (dla tablicy o rozm. $m = n^2$ my używamy h z rozszerzającą uniwersalnością) $\binom{n}{2}$

Uzasadnienie

W stanie mamy n kluczy, ~~z których~~ cyli mamy $n \cdot (n-1)/2$ par kluczy, które mogą się sobie kolidować z ppb $\frac{1}{m}$ (m to rozm. tablicy). Wiemy, iż $m = n^2$, cyli:

$$\frac{n(n-1)}{2} \cdot \frac{1}{m} = \frac{n(n-1)}{2} \cdot \frac{1}{n^2} < \frac{\frac{n^2}{2}}{2} \cdot \frac{1}{n^2} < \frac{\frac{1}{2}}{2}$$

DRZEWNA VAN EMDE BOASA

u - #wst. unikalne
n - #wst. w drzewie

37

- Operacje:
 - Insert
 - Search
 - Delete
 - Predecessor (poprzedni)
 - Successor (następny)
 - minimum
 - maximum

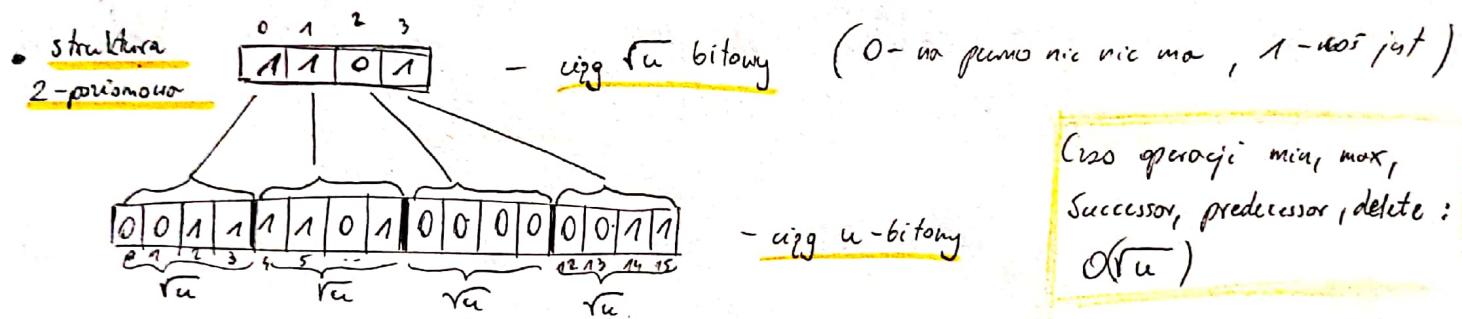


$$O(\log_2 \log_2 n), \text{ gdzie } n = 2^k$$

- rozmiar unikalne wartości, które moimy przechowywać w strukturze

Pamięć: $O(n)$

Szybki czas operacji NIEZALEŻNIE od wielkości drzewa!



Reprezentacja zbioru: $\{2, 3, 4, 5, 7, 14, 15\}$

* Dla danego wartości x , jej bit znajduje się ~~w~~ w grupie $\lfloor x/tu \rfloor$

Teren skojarz min do max poszły na strukturze zbioru w kolejności tablicowej od nich zaznaczonych numerów.

grupy $\lfloor x/tu \rfloor$

Skuteczność poprzekształcania: jeśli na liście 1, to on, jeśli nic, to $i = \lfloor x/tu \rfloor$, ?
 Które jest numerem komórką, za której znajdują się (jeśli 0, to pierwsza na liście 2 i 1). W biurowo najbardziej na prawo zaznaczony skuter 1.

- dla 7 - na liście pierwsza
- - $i = \lfloor 7/4 \rfloor = 1$: znajdują się na liście w liście numer 1.

Wstawianie elementów: Insert(v, x)

- | | |
|---|---|
| <p>tu
pobi
co nie
ma
min
max
najle
psze
ustaw
pl.
ost.</p> <ul style="list-style-type: none"> • jeśli drzewo puste, to wstaw $v.\min = x, v.\max = x$, rekursja • jeśli $v.\min = v.\max$, to \rightarrow jeśli $x < v.\min \rightarrow v.\min = x$
 $(\text{czyli } n=1) \quad \rightarrow \quad \text{jeśli } x > v.\max \rightarrow v.\max = x$ • jeśli $x < v.\min \rightarrow$ swap($x, v.\min$) $(\text{ustawiany } x)$ • jeśli $x > v.\max \rightarrow$ swap($x, v.\max$) $(\text{ustawiany } x)$ • jeśli $n > 2:$
 \rightarrow ustawiany do drzewa, bo $v.\min < x < v.\max$ $(\text{ustawiany na poziomie } \text{high}(x), \text{low}(x))$ | <p>reprezentowanie x:</p> <p>up. $7 = 01111$</p> <p>1 3</p> <p>1 tu. 2 tu.</p> |
|---|---|

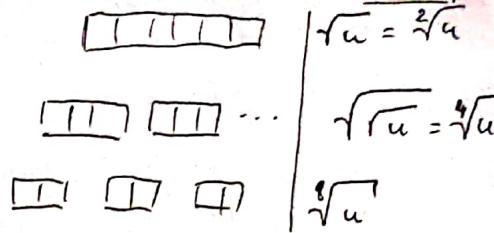
STRUKTURA REKURENCJNA

Struktura recm. k następującym str. recm. \sqrt{u} . Czyli u nas:

Dzigny do mówosci $O(\log_2 \log_2 n)$.

Pry tyle zorganizowanej struktury jest to możliwe.

Można to pokazać za pomocą rekurencyjnego:



$$u = 2^k$$

$$T(u) = T(\sqrt{u}) + O(1)$$

$$\bullet u = 2^k \rightarrow k = \log_2 u$$

$$T(2^k) = T(2^{\frac{k}{2}}) + O(1)$$

$$\bullet S(k) = T(2^{\frac{k}{2}})$$

$$S(k) = S\left(\frac{k}{2}\right) + O(1)$$

więc do poszukania $T(\sqrt{u})$, zadejm. na poziomie wyżej odznaczonej komórki $O(1)$

(problem 2,
master method)

$$\Rightarrow S(k) = O(\log k)$$

$$T(u) = T(2^k) = S(k) = O(\log_2 k) = O(\log_2 \log_2 u)$$

$k = \log_2 u$

$$\text{high}(x) = \lfloor x/\sqrt{u} \rfloor$$

$$\text{low}(x) = x \bmod \sqrt{u}$$

np. $x=7, u=16$ $\text{high}(x)=1$
 $\text{low}(x)=7 \bmod 4=3$

Substancja następnika

$\text{succ}(V, x)$

$j \leftarrow \text{succ}(\text{high}(x)), \text{low}(x))$

if $j < \infty$

return $j + \text{high}(x) \cdot \sqrt{u}$

use $i \leftarrow \text{succ}(\text{summary}(u), \text{high}(x))$

if $i < \infty$

$j \leftarrow \text{succ}(-\alpha, \text{subs}[i])$

return $j + i \cdot \sqrt{u}$

return ∞

— ale za wolny, bo $(\log_2 u)^{\log_2 3}$

Insert(V, x)

if $V.\min = \text{NULL}$:

ustaw do pustego ($\min = x, \max = x$)

else if $x < V.\min$:

$\text{swap}(x, V.\min)$

if $V.u > 2$:

if $\text{minimum}(V.\text{cluster}[\text{high}(x)]) = \text{NULL}$:

$\text{Insert}(V.\text{summary}, \text{high}(x))$

ustaw do pustego ($V.\text{cluster}[\text{high}(x)], \text{low}(x)$)

else:

$\text{Insert}(V.\text{cluster}[\text{high}(x)], \text{low}(x))$

if $x > V.\max$:

$V.\max = x$

• STRUKTURA REKURENCYJNA Z MIN I MAX

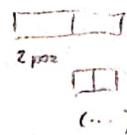
regis bitowy

X :

--	--	--

1 poz

39



- na kątym z poziomów tworzymy dodatkowe: cluster

• min - najmniejszy el. w drzewie (nie pojawi się w konsolidacji)

• max - największy - " - (pojawi się w konsolidacji)

• wskaźnik summary - wskazuje na strukturę tab[] złożoną z innych poddrzew, tyle ile ma się o tym.

• u - 1. elementem danego drzewa (lub poddrzewa)

Drzewo dla zbioru {2, 3, 4, 5, 7, 14, 15}

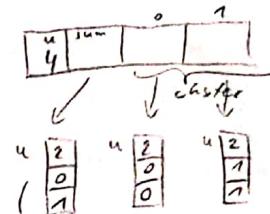
0	0	1	1	1	0	1	0	0	0	0	0	0	1	1	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
u	min	max	summary	0	1	2	3								

16 2 15

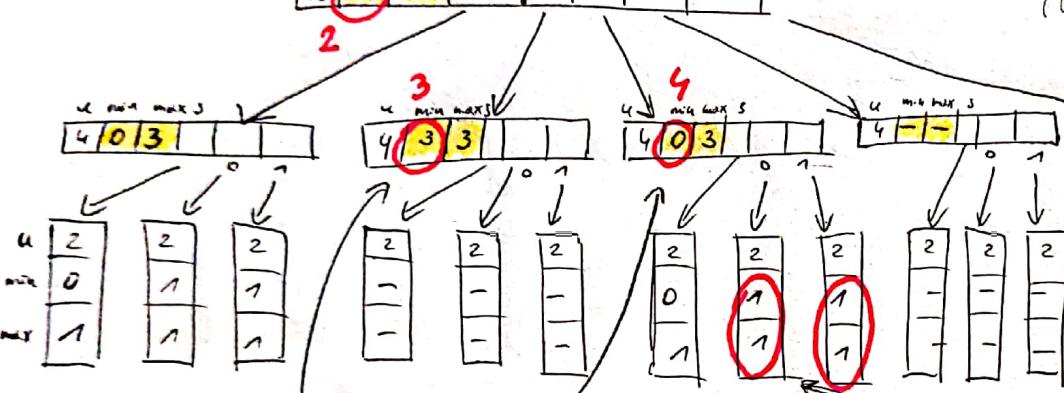
2

3

4

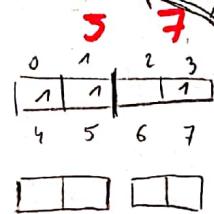


(bo nie pojawi się u, a
u drugiej sej elementy)

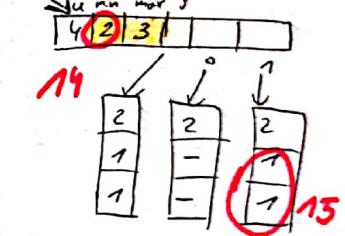


• na który pojedynczo
w konsolidacjach

• w konsolidacjach 0 nie występuje
jednak 2, to samo dla 1.0



sz jednoznaczne
min i max,
bo nie występuje
jednak 4
to samo
co zle



Jeli. dujo poziomow, to
summary moze zawierac wiele
innych summary (wizualny).

• Min i max służy do zredukowania liczby cyklicznych rekurencji i operacji na drzewie:

- operacja minimum i maximum $O(1)$, bo inf. w tabl. 1 poziomu

- przypisana grupa successor i predecessor: (successor - najmniejsza liczba, od której < max
predecessor - największa liczba, od której < min)
- dla successor mamy, że następne lery
w tym samym konsolidacji tylko gdy $x < \text{max}$

- analogicznie dla predecessor

- Drzewo skonstruowane jest puste lub ma tylko 1 el. (lub 2 - wtedy nie ma wskaźników)

SIECI PRZETĄCZNIKÓW

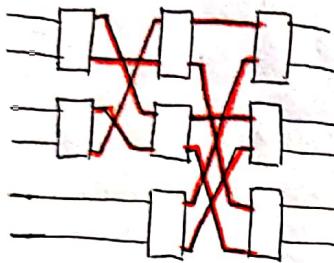
40

Przyczynki dwustanowe:



sieci przekształcające - połączone ze sobą przyczynki

np. sieci przekształcające o 6 wyjściach



Cel

Chcemy skonstruować sieć, która umożliwiłaoby uzyskanie wszystkich permutacji dla n elementów (wyjść).

Parametry jedności sieci:

- liczba przyczynników (rozmiar $S(n)$)
- głębokość sieci (maksymalna liczba przyczynników na ścieżce od wejścia do wyjścia) ($D(n)$)

Ograniczenia:

$$D(n) \geq \log n$$

$$\text{na rozmiar} : S(n) \geq \log(n!) \quad (\text{bo } 2^{\frac{S(n)}{n}} \geq n!)$$

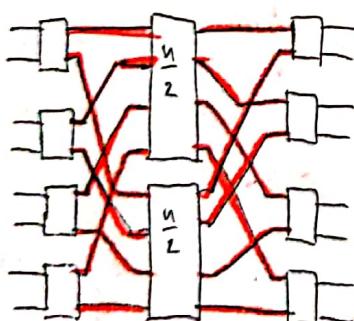
$S(n)$

↓

Sieci Benesa - Waksmana

Sieć B-W ma głębokość: $\begin{cases} 1, & n=2 \\ D\left(\frac{n}{2}\right)+2, & n>2 \end{cases}$ (jeśli przyczynki dwustanowe)

dla $n = 8$



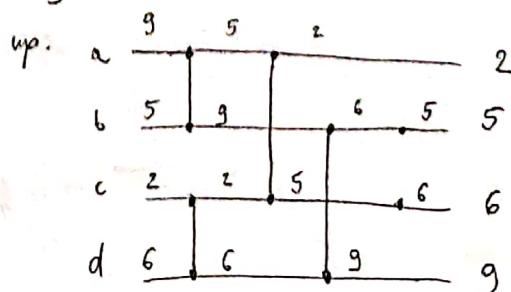
$$\text{dla } n=8 \text{ gęstość to: } D(8) = D(4)+2 = D(2)+2+2 = 5$$

* sieci sortujące - tworzące uporządkowany ciąg



$$x' = \min(x, y)$$

$$y' = \max(x, y)$$



JESLI SIEĆ SORTUJE POPRAWNIE DOWOLNY CIĄG ZŁOŻONY Z {0, 1}, TO SORTUJE TEŻ POPRAWNIE CIĄG DOWOLNYCH LICZB.

SIEĆ BITONICZNA

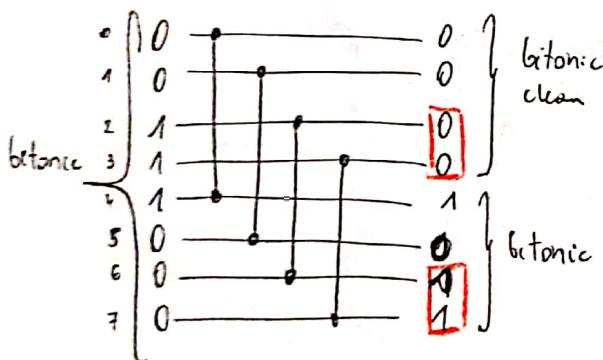
Ciąg 0-1: $0^i 1^j 0^k$ lub $1^i 0^j 1^k$

siec' pętlowąca

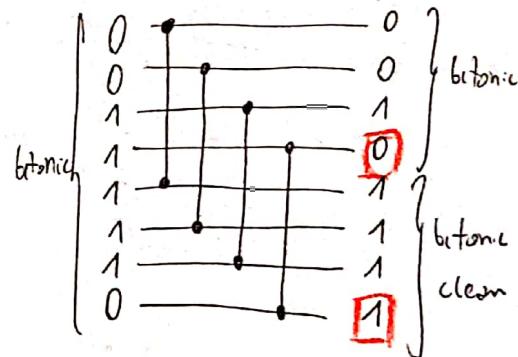
siec' bitoniczna składa się z kilku porządków, każdy z nich to half-cleaner, czyli sieć porządkująca gł. 1 (pętlaub?), w którym linia i jest porządkowana z linią $i + \frac{n}{2}$ w cyklu dangu.

$$i = 1, 2, \dots, \frac{n}{2}$$

(1)



(2)



[Siec' pętlowąca gwarantuje, że na wyjściu \leq góra połowa bitów będzie \leq dolnej.]

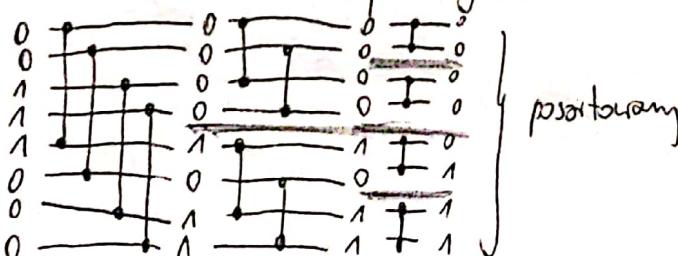
Dodatkowo dane części są bitoniczne, a min. jedna z nich jest czysta.

Ciąg

Bitoniczny - ciąg lub, które najpierw rosną, a później maleją, lub po przesunięciu cyklicznym może taki ciąg utworzyć, np.

$\begin{array}{ccccccc} 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 2 & 3 & 4 & 5 & 6 & 1 & 0 \\ 9 & 7 & 6 & 5 & 6 & 8 & \rightarrow 4 & 5 & 6 & 8 & 7 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{array}$

Sortowanie bitoniczne - reaktywującą ~~siec'~~ sieć bitoniczne



gł. bokor: $\log(n)$
(tu 3)

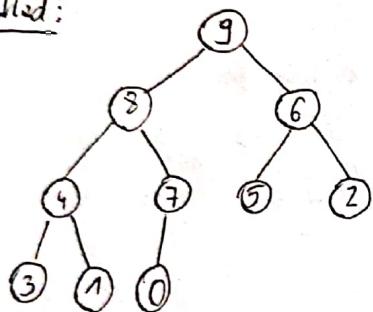
KOPIEC

Wzórki:

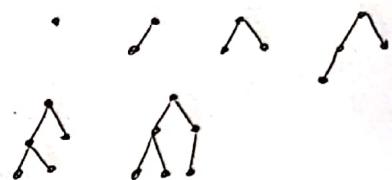
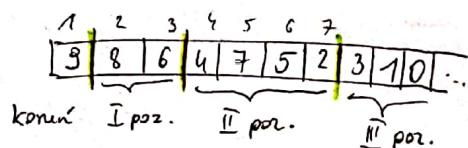
- jest kompletnym drzewem binarnym - posiada zapewnione wzorcze porządku i wyjście ostatniego (liść na poz. d lub d-1)
- wzorcze liści w pozycji d-1 leżą na prawo od wierzchołków z tego poziomu
- tylko jeden wierzchołek ($\leq d-1$) może mieć jednego syna i jest to wierzchołek pojedynczy najbardziej na prawo
- w kopcu maksymalnie więcej niż jeden syn dla każdego wierzchołka

PRZYKŁADY KOPIĘĆ:

Pojed:



Implementacja:



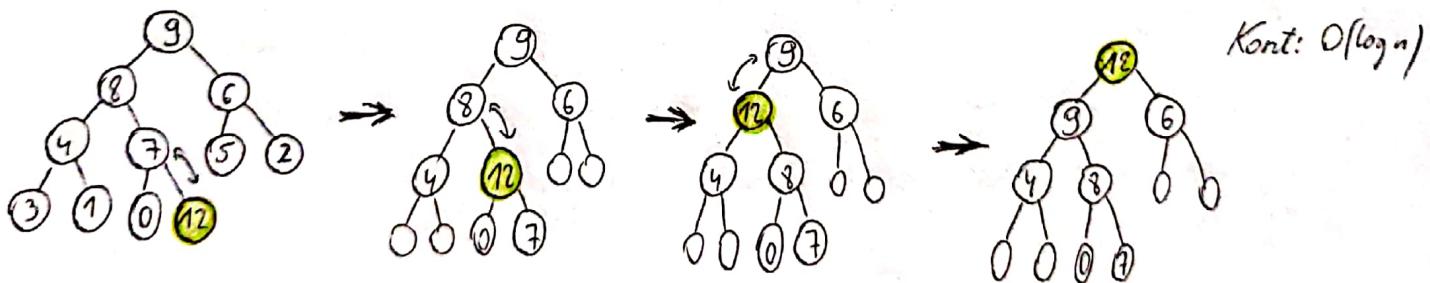
drugi wierzchołek znajduje się w: $K[2i]$: $K[2i+1]$ (numeracja od 1)

trzeci wierzchołek znajduje się w: $K[i/2]$

Operacje:

INSERT:

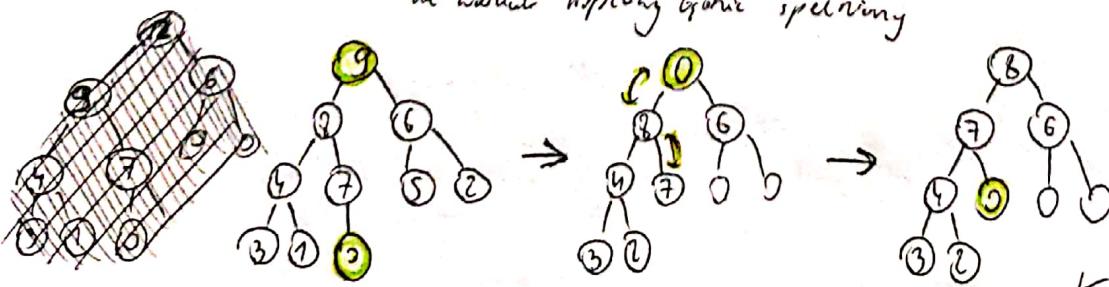
- wstawiany do listu i jeśli trba, przepychamy "do góry, idąc powrotnie"



DELETE - MAX:

- zamieniamy korzeń z ostatnim liściem (korzeń wykrywany)
- zamieniamy korzeń z największym synem (tutaj wykrywanym)
- ai wszelkie kroki są takie same

PORZĄDEK KOPIĘCY PRZYKROCONY



W kopcu o wysokości h
jest $\Theta(2^h)$ wierzchołków.

BUDOWA KOPCA : (syntka) - budowanie kopca od dołu

42

- ① najpierw budujemy logiczne 1-cl.
 - ② ujawniamy tym logiczna i nowe el. do stworzenia logiki 3-cl., po zbadaniu dla każdego koniunkcji (nowy el.) robimy „presuniecie”
 - ③ portanamy, tworząc ujawnione logiczne ai do utworzenia jednego (o ile trzeba)

class. $O(n)$

(2 log n problem)_{na problem}

PSEUDOKOD :

buduj - kopia ($K[1 \dots n]$):

for $i = \frac{n}{2}$ down to 1

$\rho_{\text{new}}(n, i)$

ZASTOSOVANIE KOPCÓN

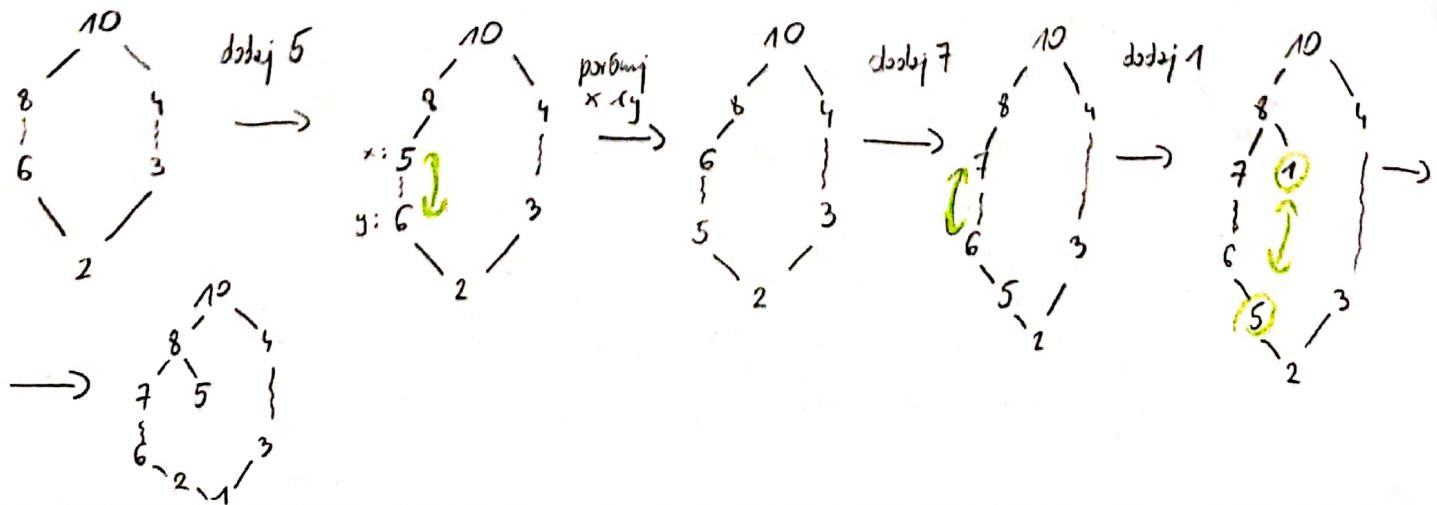
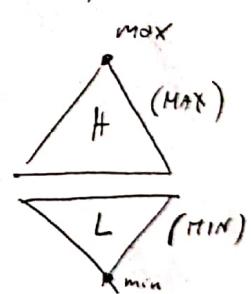
- heapsort ($O(n \log n)$)
 - złożony logicznie
 - dla el. od n do 2 wykonyj „przesuwanie”
 - krolikra priorytetowa (dodawanie el., usuwanie max, usuwanie max)
 - produkcyjna krolikra priorytetowa
 - ↗ jeśli dla kolejki

PODŁĘŻNA KOLEJKA PRIORYTETOWA:

- Budujemy dwa kopty ~~L~~^H i ~~H~~^L, w $L \subset L_2^n$ cl., w $H \subset R_2^n$, kopia ~~H~~^L jest rozyczka, L malizco.

Budowa :

- jeśli x kolumna pionek elementów, to daje do H , app. do L
 - wstawiany do pierwotnego wstępu i musimy sprawdzić czy trójkąt powstaje w gorszy w dół:
 - jeśli trójkąt zanikać wstawiany x (do H) z listem mu odpowiadającym wL , to to robimy. Wtedy powstaje x a kolumna kolumna kolumna L takiego że nie będzie żadnej reszty pionek



KOŚĆ OPERACJI

43

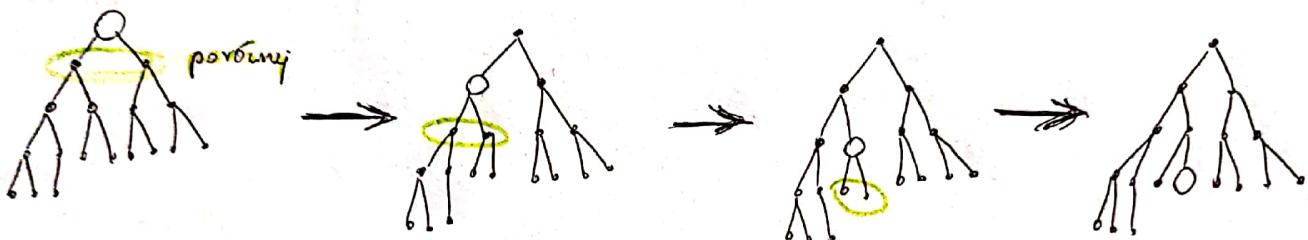
MIN, MAX $\rightarrow \Theta(1)$

INSERT $\rightarrow O(\text{avg. los} \alpha) = O(\log n)$

DELETE MIN, MAX $\rightarrow O(\log n)$

PRZYSPIESENIE DELETE

Zamień wartości przypisane w drzewie:iedy nie porównyj dwa razy ojca i dzieci, tylko same dzieci, więc mamy 1 porównanie zamiast 2.



Kiedy już zjednym się skróci, to ost. el. do drzewa. Możliwe, że struktury poza tym, więc taka ten el. przypadać w głąb.

Wartości określone przypisane ost. el. to 2, ponieważ w 2 ostatnich porównaniach jest ok. $\frac{n}{2}$ kluczy.

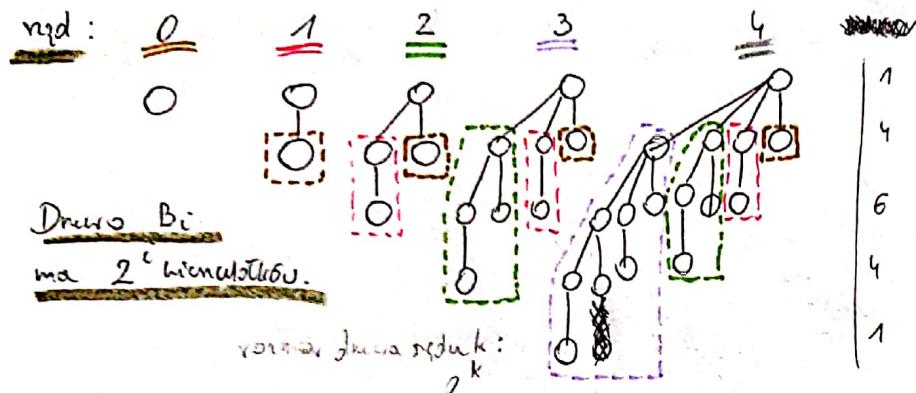
Zatem czas usuwania to $\log n + 2$.

KOPCE DWUMIANOWE

44

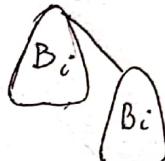
DRZEWA DWUMIANOWE:

- drzewo dwumianowe noda 0 składa się z 1 węzła (korzenia)
- / / - noda k składa się z korzenia oraz drzewem, tzn. jawnie są drzewa dwumianowe nodeli k-1, k-2, ..., 0.



MAJĄ DŁUGO

DZIECI

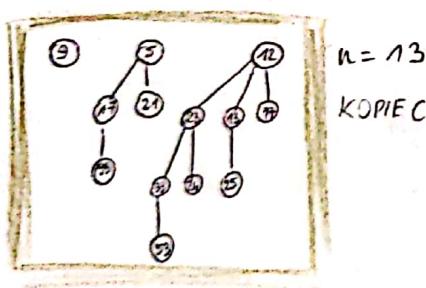


Bi

Bi+1

WŁAŚCIWOŚCI KOPCA:

- w kopcu może się znajdować co najwyżej 1 drzewo każde noda
- każde drzewo dwumianowe zachowuje właściwość kopca $\text{wart. węzła} \geq \text{wart. ojca}$
- kopiec dwumianowy o n el. składa się maxymalnie z $\log n + 1$ drzew dwumianowych
- ~~każde drzewo odpowiada kodowi binarnemu~~ kopca odpowiadająca liczba w reprezentacji bin, będąca liczbą jego elementów, np. $13 = 2^3 + 2^2 + 2^0 = 1101$



obraca się tutaj napis: (takie liczby 2^i mogą być w kopcu) (takie o t. el. drzewa nodeli)

Kopiec dwumianowy to zbiór drzew dwumianowych, które posiadają el. z uporządkowanego uniwersum zgodnie z porządkiem kopcowym.

OPERACJE:

- Findmin() - można przepiąć po korzeniach drzew dwumianowych (czas: $O(\log n)$), ale można też "wybić".
Można mieć wiele nodeli na el. minimum, ale trzeba go uzupełnić przy kodowaniu i porządkowaniu operacji. Wtedy czas to $O(1)$.
- Insert(k) - metodę ($H, matchexp(k)$) - zależy od tego czy cożycy cożycy
- Matchexp(k) - czas $O(1)$, bo kopca 1-d.
- Meld - wersja cożycy i cożycy \rightarrow

WERSJA EAGER (gothic)

45

- W tej wersji kopie ma kontekst dodały po zakończeniu operacji meld.
- W wersji lazy struktura po meld nie musi być zaktualizowana, przywracana jest podczas deleteMin.

Meld w wersji eager działa jak dodawanie binarne, np.

$$\begin{array}{r} 110101 \\ + 110110 \\ \hline 1101011 \end{array}$$

eagerMeld (H_1, H_2):

if ($\text{key}(\text{MIN}_{H_1}) < \text{key}(\text{MIN}_{H_2})$):

$$\text{MIN}_H = \text{MIN}_{H_1}$$

$$\text{else: } \text{MIN}_H = \text{MIN}_{H_2}$$

$$\text{carry} = \text{NULL}$$

for $i = 0$ to max heap size do

$$k = \# \text{ wskazówek} \neq \text{NULL} \text{ wśród } \{ \text{carry}, H_1[i], H_2[i] \}$$

case k :

$$0: H[i] = \text{NULL}$$

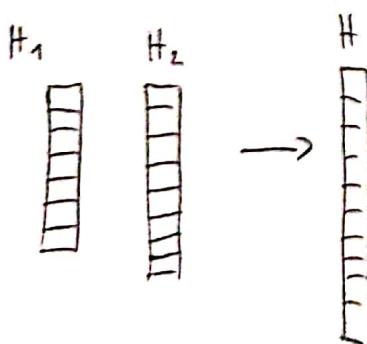
$$1: H[i] = \text{jedynie niepusta} \xrightarrow{\text{wskaźnik}} \text{wskaźnik} \text{ wskaźnik}$$

$$2: H[i] = \text{NULL};$$

carry = $\text{join}(B_1, B_2)$, gdzie B_1 : B_2 to drzewo zaktualizowane
przez niepuste wskazówki i

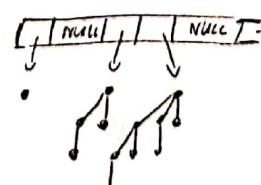
$$3: H[i] = H[i]$$

$$\text{carry} = \text{join}(H_2[i], \text{carry})$$



$H[i] = \text{wskazówka na drzewo } i\text{-tego nода}$

np.



Kost: $O(\log n)$

jeśli 3 drzewa
tego samego
noda, to T zamyka/
zamienia do carry

jeśli 1 a pozostałe 2
noda to bliski (do kopia o n el.
jeśli 2 zamyka/
zamienia do carry
szczoda się 2 maks. $\log n$
wśród drzew oznaczonych)

deleteMin(H):

- usuwanie drzewa zawierającego MIN (czyli na nimże wskazówce) z H
- usuwanie 2 tego drzewa $B \rightarrow$ oznajmij rodzeństwo drzew $B_0, B_1, \dots, B_{\text{nd}(B)-1}$ (dzieci B)
- utworzenie kopięrec H' z tym drzewem
- meld (H, H')
- min (H, H')

Kost: $O(\log n)$

Insert(k, H):
- utwórz 1-el. kopięrec H'
- meld (H', H)

Zamazywanie wsk: $O(1)$

Wszystkie operacje oprócz deleteMin kosztują $O(1)$.
 (czasy zanotowane)

Jedna reprezentacja

Nie mamy już ~~dwóch~~ tablic wskaźników. Zamiast tego drzewa dwumiarowe danego kępla połączone w listę cykliczną dżeków minibukowej.

LazyMeld (H_1, H_2)

- połączenie list H_1 i H_2 (zamiana 2 wskazówek)
- aktualizacja wskaźnika na MIN (próbne dwóch min)

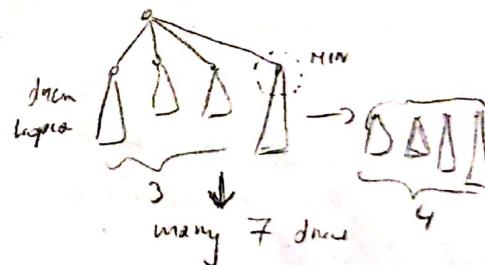
Czas: $O(1)$

Może wystąpić więcej niż 1 drzewo tego samego rędu

NAPRAWA W
DELETE-MIN

deleteMin (H):

- usuwamy koniec z drzewa B zanotowanego przez MIN
- dobieramy poddrzewa B do listy drzew kępla
- aktualizujemy wskaźnik na MIN
- redukujemy liczbę drzew \rightarrow maks. 1 danego rędu
 (czyli połączymy drzewa)



* Zauważmy, że może być krontowe (maks. $O(n)$, gdy mamy n drzew 1-ed.).
 Zanotowany koszt to $O(\log n)$.

KOPCE FIBONACCIEGO

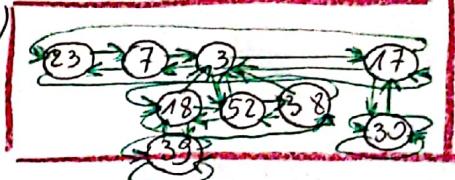
(Podobne do kopcoów dwumianowych lazy)

47

- wiechotki w porządku lepszym ($\text{wart. wiechotka} \geq \text{wart. ojca}$)
- drewa mogą mieć dwoje krotów, nie mogą być dwumianowe
- podobne jak w dwumianowych lazy! (ale dyleż chwilę, np.)
 - ustawimy na MIN
 - przedstaw jako lista głosiczną dwukierunkową
- w każdym wezgłowiu wiechotku kopca informują o tym, czy wiechotek utrać jednego z synów i synchronizują ~~z~~ operację CUT (wart. wyjścia)

jeśli w kopach dwumianowych lazy

PRZYKŁAD



OPERACJE:

O(1)

- makeheap(k) - jak w dwumianowych (1-d. kopca): O(1)
- insert(k, H) - — II — ($\textcircled{1}$ utwor 1-d. kopca, $\textcircled{2}$ meld(H', H)): O(1)
- findmin(H) - — II — (najm. ustawnik): O(1)
- meld(H_1, H_2) - — II — ($\textcircled{1}$ łączym listy H_1 i H_2 , $\textcircled{2}$ zmieniamy ustawnik na MIN): O(1)
- cut(H, p) - odnosi się do wiechotek wew. ustaw. ustawionego przez p od jego ojca p' i dotych (operacja meld) podobno zakonczone w p do listy drew kopca.
 - Jeli p jest pierwszym synem, jeliż utrać p' , to zauważ to w p'
 - Jeli p' jui utrać syna to wykonujemy cut(H, p')
 Portany do momentu uspokojenia wiechotka, który nie utrać syna lub brama.

* możliwa utrata porządku lepszego

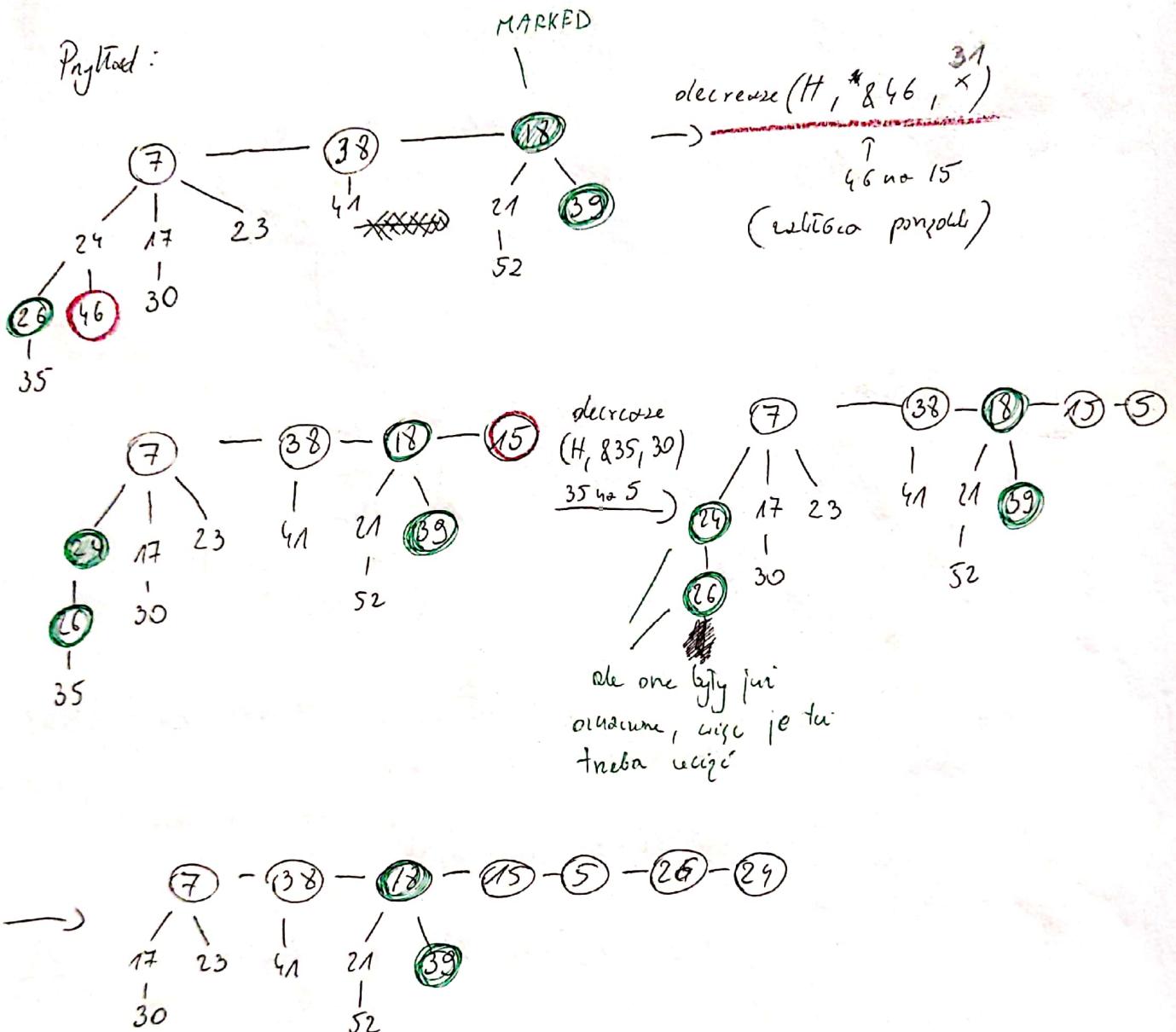
- decrement(H, p, Δ) - zmniejsza wartości elementu ustawianego przez p o Δ .
 Koszt amortyzacyjny: O(1)
 Jeśli nowa wartość zaktua porządku lepszy (mniejsza od ojca), to wykonujemy cut(H, p).
- deletemin(H) - jak w kopach dwumianowych, tyle ic przy tyczeniu obu drew tego samego nrodu, reguła jest liczba synów.

 Koszt amortyzacyjny: O(log n)
 (drewa tyczące reagują od najmniejszego nrodu)

bo stopień wiechotków w kopach Fib ograniczony przez O(log n).

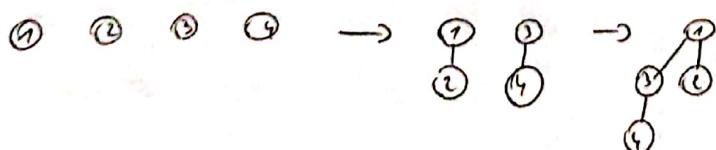
(czyt?)
 tci

- delete(H, p) - ustosuny w p minimum (~~decrement~~ decrement($H, p_1 \rightarrow \infty$)) 48
a priority robimy deletion(H) \rightarrow sort rosnajacy O($\log n$)



* po usunięciu, kiedyś dalej olla 1 przewin (nudz) ai nie
takie parzy.

np.



ALGORYTMY GRAFOWE + MST

49

DFS — przeszukiwanie w głębi (Depth-first search)
(np. do stwierdzenia istnienia wątpliwego?)



Działanie:

1. Zaczynamy od wierzchołka v , jakaś odwiedzony.
2. Przejść do kolejnego sąsiada v' : wykonaj dla niego (1, 2).
- jeśli v' odwiedzony, to do niego nie idziemy

- spójne składowe
- sortowanie topologiczne
- sprawdzenie istnienia ścieżki między dwoma wierzchołkami

Algorytm:

DFS (G):

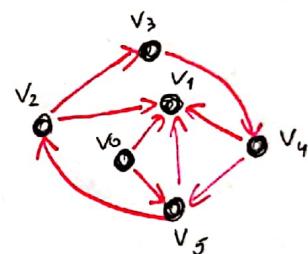
```
for  $v$  in wierzchołki-grafu —  $G$ :
    oznaczać  $v$  jako nieodwiedzony (false)
for  $v$  in wierzchołki-grafu —  $G$ :
    if  $v$  nieodwiedzony:
        VisitNode( $v$ )
```

VisitNode (v):

```
oznaczać  $v$  jako odwiedzony
for  $u$  in sąsiedzi —  $G$ :
    if  $u$  nieodwiedzony:
        VisitNode( $u$ )
```

• Skomplikowana: $O(|V| + |E|)$

zb. liczeniowe
zb. krawędziowe



• Skomplikowana: $O(h)$

zb. najdłuższej ścieżki skończonej

1. zaczynamy od v_0 : true.
2. v_1 : true \rightarrow nie ma spójnika, więc przenosimy do v_0
3. v_5 : true
4. v_2 : true
5. v_4 : false \rightarrow pamięć
6. v_3 : true
7. v_4 : true \rightarrow odwiedzono już węzeł, koniec

BFS — przeszukiwanie szerokości (breadth-first search)
(spójny jest na grafie przy dwóch wierszach)

Działanie:

1. Zaczynamy od wierzchołka v
2. Odkadamy jego sąsiadów
3. Odkadamy kolejnych sąsiadów
4. Odkadamy kolejnych sąsiadów sąsiadów

Algorytm:

BFS(G, s):

```
ustawić  $s$  do kolejki  $Q$ 
 $s.visited = \text{true}$ 
while !  $Q.empty()$ 
    wyciągnąć el. z  $Q$  i ustaw go z kolejki
    for  $u$  in sąsiedzi ( $v$ ):
         $Q.push(u)$ 
         $u.visited = \text{true}$ 
```

• Skomplikowana: $O(|V| + |E|)$

• Skomplikowana: $O(|V| + |E|)$

• dla City
• sąsiedztwa

• jedna wiersz
• sąsiedztwa, to
1/2

ALGORYTM BORUVKI (zajdwanie MST)

50

Algorytm: 1° $\forall v \in G$ znajdź najkrótszą ścieżkę z wierzchołka v do wierzchołka s

E¹-26. trang tri
MST

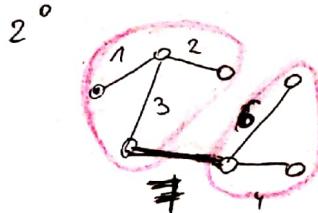
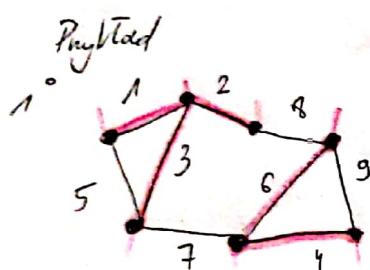
2° stebn my graf G' , jehic viedotkom odpovídají
spíne vtedace $\in E'$ (superincidentní - si).

Dara supervisorenbēhi Sj. i. Sj. Tęczyng (obroty wieczyste
z Sj. pożary z dowolnym wieczystkiem z Sj. u G).

Waga tyciąg brzmieć to minimum mówiąc o waga krajobrazu, która spełnia się utworem.

3° $G = G'$: ciòr do 1°.

Dziata populacji, gdy krajobraz ma głębsze wagę.



MST grafi G

zloženosť času: $O(\log N)$

MST (dla grafu G) - ~~druwo o minimalnej wadze spośród wszystkich drzew~~ rospiniżnych grafu G , gdzie drzewo rospiniżne to skierowane drzewo $T = (V, E')$ t.i. $E' \subseteq E$

(Wystarczy 3 alg. (Boruvka, Kruskala, Prima) oparte na strategii rozbijanej)

ALGORYTM KRUSKALA

Diction:

- Mamy postać eliż E' oraz eliż C,
poszukujemy równej E.
 - Rozpatrujemy kątowy z C o m.in.
wach. Jeżeli nie powoduje cyklu, to
dodajemy ją do E', usuwamy (w orzeczy)
z C.

Krok ten powtarza się do postrzelenia MST.

Unimodular class: $\mathcal{O}(E \cdot \log |V|)$

Algorytm

~~1) *Pyromorph*~~
T - 26. *pusly* (na Kraugster M.J.T)
C - 26. *Kraugster puslytowayn wog wog* ($O(E \log E)$)

2 - struktura zbioru robiących, na początek
który wersja to A.d. zbiór ($O(E \times FV)$)

for $i = 1, 2, \dots, n-1$ (This is due to the boundary values)

- $c = u - v$ (czy to dodać masy) rysunek funkcji
 - i $u \approx v$ $\Rightarrow c$
 - sprawdzić czy u_0 , do którego należą u i v są rożne (dla których brzegi nie sterczą cyfka)
 - * jeśli tak, to dodać c do T
 - 246. wybór $T \Rightarrow$ MST

ALGORYTM PRIMA

Działanie:

- wybieramy dowolny wierzchołek
- wybieramy najmniejszą krawędź wychodzącą z tego wierzchołka (dodaj do E')
- w kolejnych krokach wybieramy min. krawędź z C incydentną z jużą krawędzią z E' (t.j. której końca nie jest incydentne z E')

Złożoność czasowa: $O(|E| \cdot \log |V|)$

(moc biegi przeszła przy rozważaniu liczb Fibonacciego)



dla zwykłego

kopia
kolejka priorytetowa

$O(|E| + |V| \cdot \log |V|)$

(opisana przy grafie o dwiugim
zgromadzeniu - dwa krańce)



ZNAJDOWANIE NAJKRÓTSZEJ ŚCIĘZKI

ALGORYTM DIJKSTRY (dla $w \geq 0$)

Znajduje najkrótsze ścieżki pomiędzy wybranym wierzchołkiem a pozostałymi wierzchołkami grafu.
(wierzchołek źródłowy \rightarrow numer wagi)

Q - kolejka priorytetowa wszystkich wierzchołków grafu (wg najkrócej odległości od s (wierzchołek startowy))

d - tabela akt. wierzchołków grafu G od s

Algorytm:

for v in $V(G)$:

$d[v] = \infty$

poprzednik [v] = NULL

$d[s] = 0$

$Q = V$

while Q :

wybrany u najmniejszy odległość do środka (min. $d[u]$) i usunięto z Q

for v in sąsiadów (u):

if $d[v] > d[u] + w(u, v)$:

$d[v] = d[u] + w(u, v)$

poprzednik [v] = u

Algorytm:

T - tab. pusta (HST)

C - kolejka priorytetowa, na początku pusta, później reprezentująca wierzchołki osiągnięte z HST + krańce (i najmniejszą kolejkę niewykorzystaną)

~~do v do T jeden (dowolny) wierzchołek, a do C jego sąsiedów, usun v z V (ub. sasza)~~

- dodaj do T jeden (dowolny) wierzchołek, a do C jego sąsiedów, usun v z V (ub. sasza)
- for v in ~~V~~ C :

if v in T : continue

$T.push(v, e)$

$C.push(sąsiad - v)$

(wielokrotnie kolejne priorytetowe)

(algorytm zadawany)

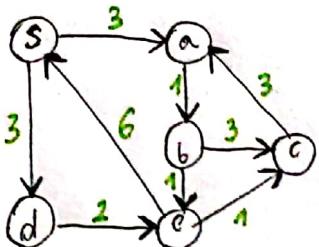
• Złożoność czasowa:

• przy naiwnej implementacji: (czytaj tablicę

jako Q): $O(|V|^2)$

• Q jako logrec: $O(|E| \cdot \log |V|)$

• Q jako logrec fib: $O(|E| + |V| \log |V|)$



u	s	a	b	c	d	e
d[u]	0	∞	∞	∞	∞	∞
p[u]	-	-	-	-	-	-

- wciż $s \in Q$ ($Q = [a, b, c, d, e]$)

• sprawdzić sąsiadów s : a i d :

$$\begin{aligned} - a: \quad d[a] &> d[s] + w(a, s) \rightarrow d[a] = 3 \\ &\quad \text{---} \quad \infty \quad 0 \quad 3 \\ - d: \quad d[d] &> d[s] + w(d, s) \rightarrow d[d] = 3 \\ &\quad \text{---} \quad \infty \quad 0 \quad 3 \end{aligned}$$

u	s	a	b	c	d	e
d[u]	0	3	∞	∞	3	∞
p[u]	-	s	-	-	s	-

- mamy teraz do wyboru: b, c, e, a, d w Q (\rightarrow reszta nie ma)

- wciż a $\in Q$ ($Q = [d, b, c, e]$)

• sprawdzić sąsiadów a: b

$$\begin{aligned} - b: \quad d[b] &> d[a] + w(a, b) \rightarrow d[b] = 4 \\ &\quad \text{---} \quad \infty \quad 3 \quad 1 \end{aligned}$$

$$\begin{aligned} p[b] &= a \\ &\quad \text{---} \end{aligned}$$

- wciż d $\in Q$: ($Q = [b, c, e]$)

• sprawdzić sąsiadów d: e

$$\begin{aligned} - e: \quad d[e] &> d[d] + w(d, e) \rightarrow d[e] = 5 \\ &\quad \text{---} \quad \infty \quad 3 \quad 2 \end{aligned}$$

$$\begin{aligned} p[e] &= d \\ &\quad \text{---} \end{aligned}$$

u	s	a	b	c	d	e
d[u]	0	3	4	∞	3	∞
p[u]	-	s	a	-	s	d

- wciż b $\in Q$: ($Q = [e, c]$)

• sprawdzić sąsiadów b: e i c

$$\begin{aligned} - e: \quad d[e] &> d[b] + w(b, e) \rightarrow \text{NIE} \\ &\quad \text{---} \quad 5 \quad 4 \quad 1 \end{aligned}$$

$$\begin{aligned} - c: \quad d[c] &> d[b] + w(b, c) \rightarrow d[c] = 7 \\ &\quad \text{---} \quad \infty \quad 4 \quad 3 \end{aligned}$$

$$\begin{aligned} p[c] &= b \\ &\quad \text{---} \end{aligned}$$

- wciż c $\in Q$:

• sprawdzić sąsiadów c: a

$$\begin{aligned} - a: \quad d[a] &> d[c] + w(c, a) \rightarrow \text{TAK} \quad d[a] = 6 \\ &\quad \text{---} \quad 7 \quad 5 \quad 1 \end{aligned}$$

$$\begin{aligned} p[a] &= c \\ &\quad \text{---} \end{aligned}$$

- wciż e $\in Q$:

• sprawdzić sąsiadów e: a

$$\begin{aligned} - a: \quad d[a] &> d[e] + w(e, a) \rightarrow \text{NIE} \\ &\quad \text{---} \quad 3 \quad 6 \quad 3 \end{aligned}$$

u	s	a	b	c	d	e
d[u]	0	3	4	6	3	∞
p[u]	-	s	a	e	s	d

KLASY ZŁOŻONOŚCI ALGORYTMÓW

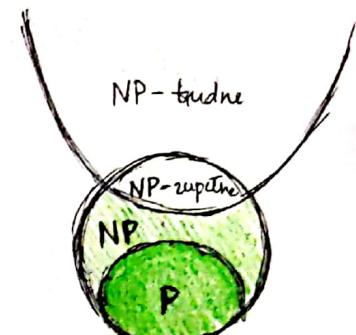
53

P - zur leichten reziproke or erweiternde Meliorierung (wirkt $O(n^k)$ dla stałego k)
 (np. cy graf jest 2-kolorowalny)

NP - sprawdzenie rozwiązań w czasie wielomianowym
(np. problem plebiscytu)

NP - zupełny (NPC) - najtrudniejsze problemy z NP. Jaki w NP-trudnych: każdy problem z NP może zawrzeć zredukowanie do tego problemu w czasie wielomianowym, ale problem z NPC musi być w NP.

NP - trudne - problem dla którego każdy problem z klasy NP (nie musi) może zredukować się do niego (redukowaczy w czasie wielomianowym). Sam problem nie musi być w NP. (np. zadanie o podklasach, który sumuje się do O)



$P \subseteq NP$

$$NPC \subseteq NP$$

NPH nec musi byc
w NP

Problem	Sprawdzenie row. w czasie wielomian.	Zasłoniene row. w czasie wielomianowym
P	✓	✓
NP	✓	✓ lub X (bo NP zawsze tyci problemy z NPC)
NP-zupełny	✓	X
NP-trudny	X	X

Algorytm pseudodominacyjny – algorytm, którego czas działania ograniczony jest przez wielomian栗inga (nie zależy od ilości danych, a o największym liczbie wejściu) np. algorytm dynamiczny dla problemu plecakowego

PROBLEM PŁECAKOWY

Mamy wiele n-el. o różnych wartościach wadach (może być też wiele z parametrem). Chcemy zebrać przedmioty o jak największej sumaryjnej wartości, tak aby zważyły się w plecaku (wg. wagi).

Rozw. programowanie dynamiczne

$W[1..n]$ - tablica wag
 $c[1..n]$ - tablica wartości
 Tycząca waga $\leq W$
 Szukamy: $A(W)$ - max. wart. dla wagi W

ELEMENTY

nr elem.	1	2	3	4
waga w_i	5	4	6	3
wartość c_i	10	40	30	50

$$A[i, w] = \begin{cases} A[i-1, w] & \text{jeśli } w_i > w \\ \max(A[i-1, w], A[i-1, w-w_i] + c_i) & \text{inne} \end{cases}$$

$i \setminus w$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50
3	0	0	0	0	40	40	40	40	40	50
4	0	0	0	50	50	50	90	90	90	90

\leftarrow maksymalna waga

\leftarrow przedmiot nr 1 waży 5,
wysokość od tego momentu masy
 $(w=5)$
go przedmiotu

$i-1, w-w_i$	\dots	$i-1, w$
i, w		

ultimo element

Alg: for $w=0$ to W :

$$A[0, w] = 0$$

(for $i=0$ to n :

$$A[i, 0] = 0$$

for $w=0$ to V :

if $w_i > w$: $A[i, w] = A[i-1, w]$

$$\text{else } A[i, w] = \max(A[i-1, w], A[i-1, w-w_i] + c_i)$$

W i-tym kroku decydujemy
czy wprowadzić się
wciąż do plecaka.

$$A[2, 5] = \begin{cases} A[1, 5] & \text{jeśli } 4 > 5 \\ \max(A[1, 5], A[1, 1] + 40) & \text{inne} \end{cases}$$

$$A[3, 4] = A[2, 4] \quad \text{jeśli } 6 > 4 \vee$$

Złożoność: $O(n \cdot W) \rightarrow O(n \cdot 2^x)$, gdzie x to liczba bitów dla zapisu W ,
czyli ograniczenie pojęte udomowiono zależy od wielkości wejścia \rightarrow pseudorekurencyjny

DREWIA AVL

H mówione po wierszach!

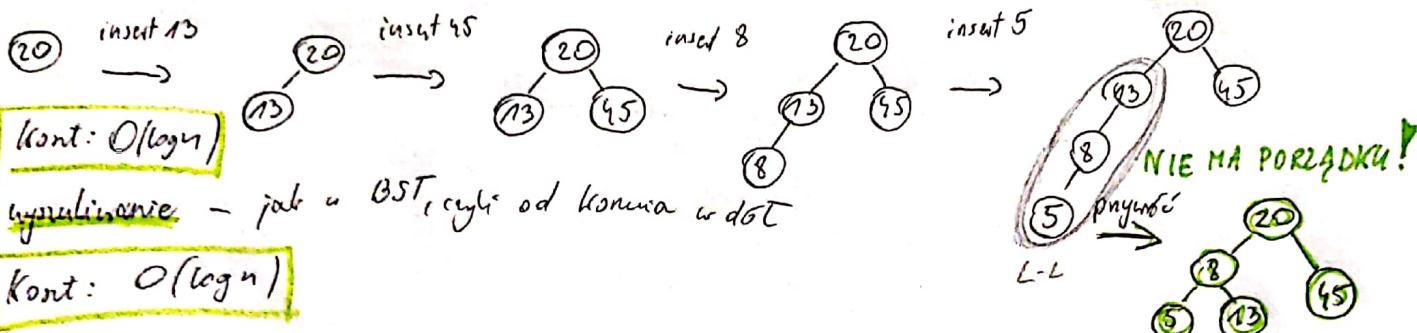
$$h: \log(n+1) \leq h < c \log(n+2) + b$$

$$c = 1,44, b = -0,328$$

Zrównoważone drzewo binarne t.zw. Hv /gdy przekroju podziału - avg. długość podziału ≤ 1 .

Operacje:

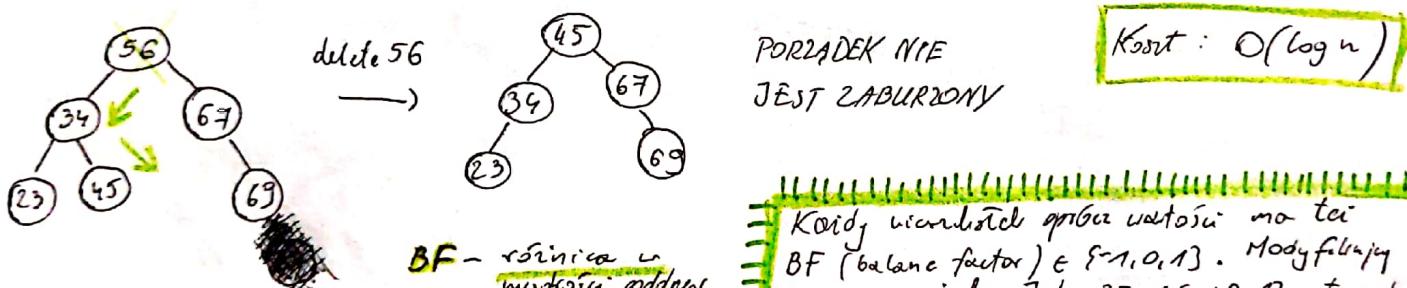
- insert - jak w BST, tylko iściemy od korzenia i porównując z wstawianym v. i ile wiele wartości może zostać **+ TRZEBA PRZYWRÓCIĆ PORZĄDEK**.



- wyrywanie - jak w BST, iściemy od korzenia w dół

Kost: $O(\log n)$

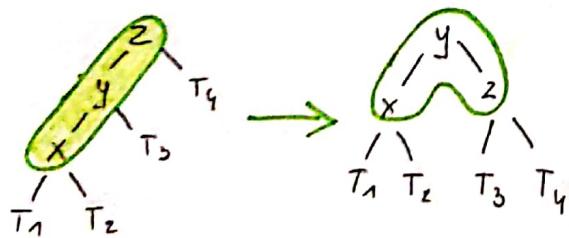
- delete - jak w BST, iściemy od korzenia i usuwamy wartości, usuwając je i ~~przypisując ją innemu miejscu~~
i ją miejsce ustawiamy odpowiadającego następcza **+ TRZEBA PRZYWRÓCIĆ PORZĄDEK**



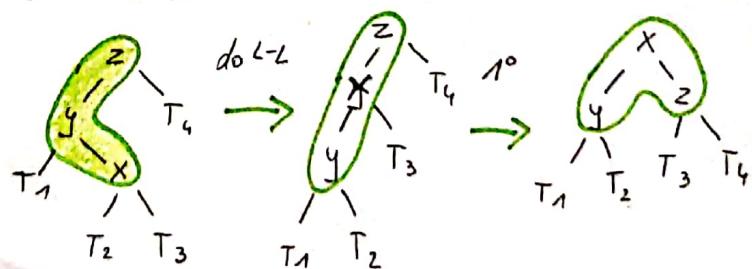
Kiedy wiele wartością której wartości ma ten BF (balance factor) jest -1, 0, 1. Modyfikujemy go przy operacjach. Jeśli BF jest -1, 0, 1, to rotacja

- rotując - do przywracania porządku w

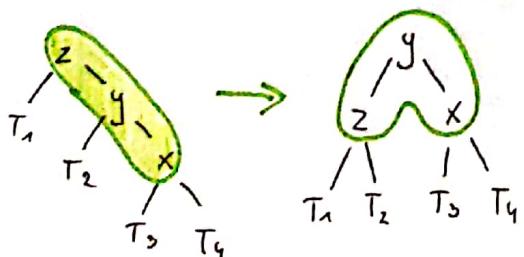
$1^{\circ} L-L$



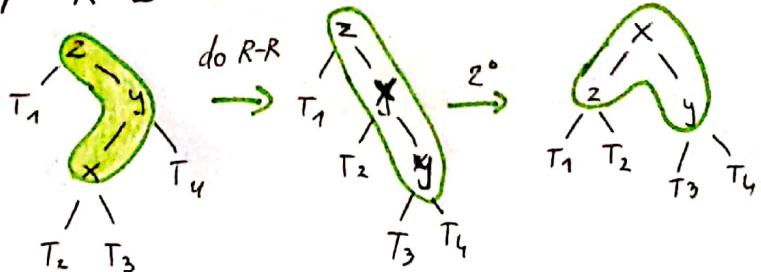
$3^{\circ} L-R$



$2^{\circ} R-R$



$4^{\circ} R-L$



B-DRZEW

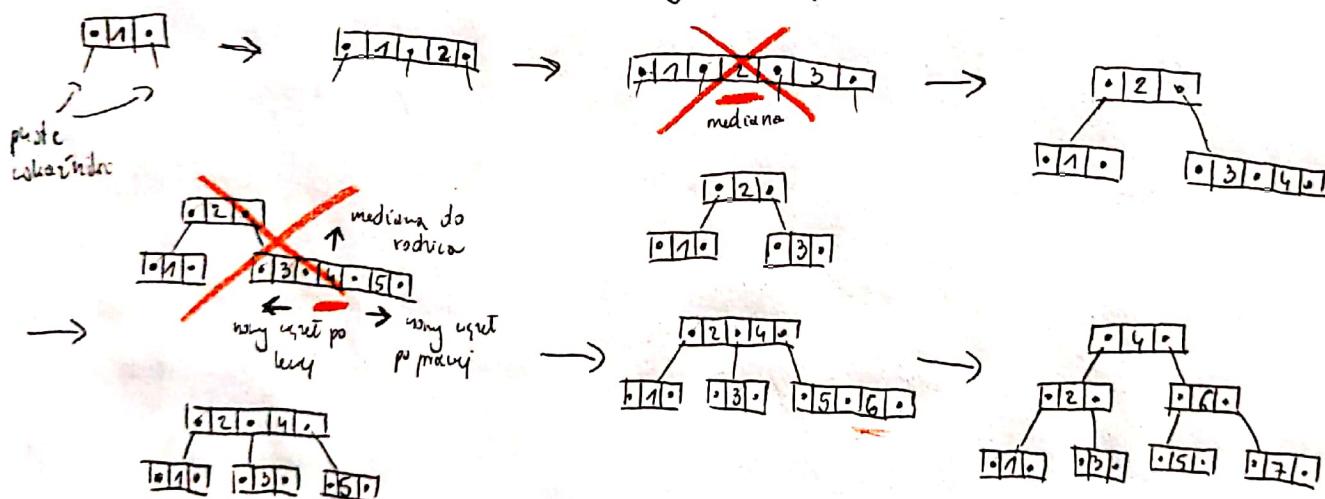
Zbalansowane drzewa przerzutowe. Przeznaczone do trzymania dużej ilości danych, np. bazy danych, dyski.

[W wierchołku moimy przechowywać więcej niż 1 element, a operacje (insert, delete, search) są curse logarytmicznym.]

Cechy B-drzewa niskie m:

- kaidy wierchołek ma max. m dzieci (dla $m = 3$)
- kaidy wierchołek nie będący liściem (wierchołek) ma przechowywać $\lceil \frac{m}{2} \rceil$ dzieci i co najwyżej 1 klucz
- koniec jest liściem lub ma od 2 do m dzieci
- wszystkie liście są na tej samej głębokości

Prykłod: B-drzewo niskie 3 ; wstawiamy $\{1, 2, 3, 4, 5, 6, 7\}$



- Wstawiając element idziemy od konca w dół.
- Jeśli w wierchołku $< \max_{(m-1)}$, to dodajemy.
- Wpp. dodajemy nowy element i usuwamy medianę.
Medianę idziemy do rodzica, + el. na lewo tuż po nowy węzeł po lewej stronie, a te po prawej po prawej stronie.

Wstawiany węzeł do drzewa !

DRZEWA SPLAY

Samoorganizująca się drzewa BST (przez dwoje uniejsie, lewe dwoje wyżej od rodzica).

Po każdej operacji (insert, delete, search) element, dla którego była wykonywana trafia do konca (w przypadku delete rodzice elementu) - operacja Splay (T, x). Ma to na celu przygotowanie przyjętych operacji.

- Dobre wykonanie rotacji pozwala na spłaszczenie drzewa.
- Jego wysokość ~~jest~~ NIE JEST OGRANIIONA do $O(\log n)$.

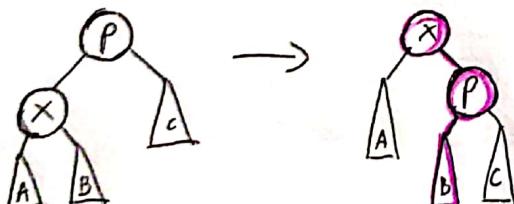
Operacje:

- insert: jakaś w BST (istnieje od końca w dół) + Splay (T, x).
- delete: jakaś w BST (istnieje od końca w dół) + Splay (T, λ) — usuwany element trafia do końca, utrudnia go usunięcie, a jego poddrzewa tworzą za pomocą join (S_1, S_2).
O ile mamy tutaj więcej niż dwa poddrzewa, to wówczas skomplikujemy x (usunięcie), czyli ustawimy najwyżej położone w drzewie i umieszczyć w koncu (za pomocą Splay (T, x)).
- search: szukamy jakaś w BST (od końca w dół) i jeśli znajdziemy, to robimy Splay (T, x)

3 przypadki kroków Splay (T, x):

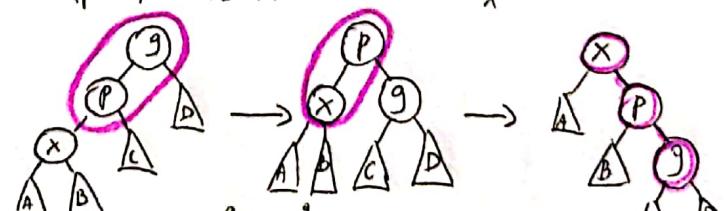
p - ojciec x

- zig — gdy p jest końcem (także ostatni krok procedury Splay)



- zig-zig — gdy p nie jest końcem, a (p, x) i (g, p) tworzą prostą

Wtedy:
 1^o rotacja w lewo (g, p)
 2^o rotacja w lewo (p, x)



- zig-zag — gdy p nie jest końcem, g, p, x ułożone są $p <_x b >_x p$

Wtedy:

- 1^o rotacja (p, x)
- 2^o rotacja (g, x)

