Algorytmy i struktury danych

Literatura:

1) Corman, Leiserson, Rivest „Wprowadzenie do algorytmów" WNT
2) Neapolitan, Naimipour „Podstawy algorytmów" Helion
3) Aho, Ullman, Hopcroft „Analiza i projektowanie algorytmów" Helion
4) Banachowski, Diks, Rytter „ASD" WNT
5) www.algorytm.cad.pl
6) Ian Parberry → skrypt
7) Drozdek „C++ Alg. i str. danych"

Konsultacje → Krzysztof Sep IBS

Egzamin quasi-testowy

$\boxed{9.10.2005}$

Algorytmika

ZAD Mamy tablicę jednowymiarową 2, 3, -1, 2, -10, 8, ...
Znaleźć przedział o maksymalnej sumie

1. Szukamy jedną pętlą for wszystkie możliwe początki, drugą pętlą wszyst. możliwe końce, trzecią for sumujemy i szukamy max
Złożoność N^3

2. Pierwszą pętlą for znajdujemy wszystkie możl. początki, drugą for do początku dodajemy kolejne liczby, sumując je i sprawdzając max.
Złożoność N^2

pocz

2, 3, -1, 2, -10, 8, ...

sum1 sum2 sum3 ....

3. „Dziel i zwyciężaj"

Tablicę dzielimy na 2 połowy, znajdujemy max po lewej, max po prawej, ale co ze środkiem?

Od środkowego elementu szukamy po obu stronach z algorytmu sumy max

3 for 4 for

2, 3, -1, 2, -10, 8, 5

1 for 2 for

3 algorytmem początek znany → szukamy dodając kolejno następne

Złożoność

$$T(N) = 2 \cdot T(N/2) + N$$

2 strony P i Lewa — z + środek

$$\frac{T(N)}{N} = \frac{T(N/2)}{N/2} + 1$$

$C(N) = C(N/2) + 1$ f-cja logarytmiczna

$T(N) = N \cdot \log N$ ← złożoność

4. Mamy tablicę 

koniec przedziału

początek gdzie będę

Szukamy sumy max

2, 3, -1, 2, -10, 2, 5, -3, 1 ← koniec tablicy

tworzymy nową tabelę złożoną z sum.

2	5	4	6	0	2	7	4	5
	2+3	2+3-1	2+3-1+2	bo	zaczynamy od nowa	2+5	2+5-3	2+5-3

-10 niweczy zyski i tego w ogóle nie liczymy na tym nie zaczepiamy końca

2 zmienne, 1 pętla for

$S[i] := Max\{S[i-1] + D[i], 0\}$

Złożoność 2N (czyli po prostu N)

! Inny problem: Czy istnieje przedział o sumie większej od A a mniejszej od B

INDUKCJA MATEMATYCZNA

ZAD Mamy wyspę. ~~..~~ Można ją podzielić autostradami. Obsługujemy strony autostrad tak że po prawej jest inna a po lewej inna

Inaczej problem mapy dwukolorowej

Dla N=1 ok Dla N=2 ok

Jeżeli dla N jest ok to wprowadzając N+1 autostradę zmieniamy kolor po jednej ze stron autostrady na przeciwny.

Po 1 stronie jest ok, Po 2 jest ok no środku też (zmiana koloru)

OK

Scalanie przez sortowanie (Merge Sort)

8 [5, 7, 1, 2,] [3, 4, 6, 8]

↓ ↓ ⇐ sortowanie

1, 2, 5, 7   3, 4, 6, 8

ustawiamy znacznik na początkach i szukamy min, po znalezieniu ustawiamy znacznik na następnym

[illegible] 1 2 3 6
[illegible] 1, 2, 5, 7
[illegible] 1 4 5 7
[illegible] 3, 4, 6, 8

→ 1, 2, 3, 4, 5, 6, 7, 8,

$$\left.\begin{array}{l} \boxed{5, 7, 1, 2, 3, 4, 6, 8} \\ \boxed{5, 7}\ \boxed{1, 2}\ \boxed{3, 4}\ \boxed{6, 8} \\ \boxed{1, 2, 5, 7}\ \boxed{3, 4, 6, 8} \\ \boxed{1, 2, 3, 4, 5, 6, 7, 8} \end{array}\right\} \ln N$$

N (jeden przebieg)

złożoność

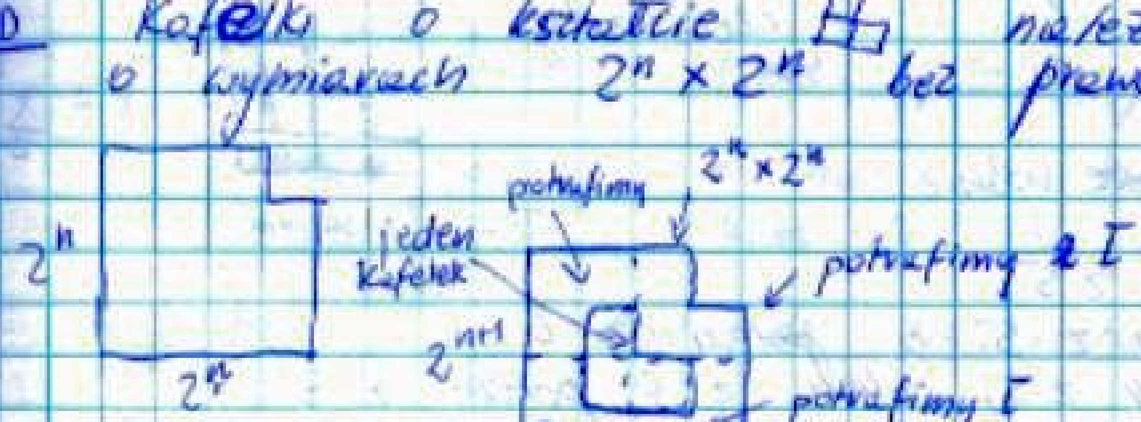
$T(N) = N \ln N$

$$T(N) = T(N/2) + T(N/2) + N = N \ln N$$

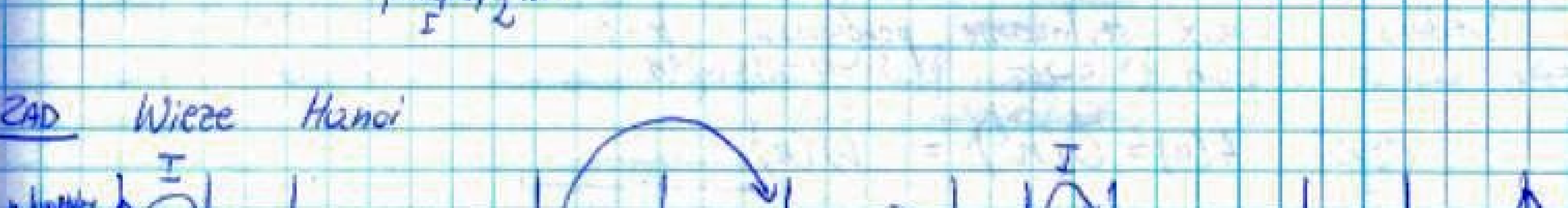
↑ zmergeowanie jednej połowy   ↑ –"– drugiej –"–   ↑ –"– całości

Ponieważ merge'owanie jest prawdziwe dla $N=1$ i N to i dla $N+1$ też (indukcja matem.)

ZAD Kafelki o kształcie ⌐ należy ułożyć na szachownicy o wymiarach $2^n \times 2^n$ bez prawego górnego rogu



ZAD Wieże Hanoi



Trójkąt Sierpińskiego (fraktal) ⇐ tr. Pascala / modulo 2

Złożoność CZASOWA

→ pesymistyczne
→ ~~optymistyczne~~ średni

1. Pesymistyczna

$W(n) = \max\{ time_A(d) : d \in D_n \}$

2. Średnie

$A(n) = \sum_{k>0} k \; P_r(\{d \in D_n : time_A(d) = k\})$

Oznaczenia : $f(n) = O(g(n))$ $\exists c, n_0 \in R^+ \; \forall_n > n_0 \quad f(n) \leq c \cdot g(n)$

np. dla złożoności $\frac{3}{4} n^2 + 5n - 1$ $\quad O(n^2) \quad \Theta(n^2)$ dla $n > 7$

Ω
Θ

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

ZAD Search → szukamy danej wartości w tabeli

$W(n) = O(n)$

d = Perm [n] $\quad x \in \{1, \dots, n\}$

Sortowanie przez wstawianie

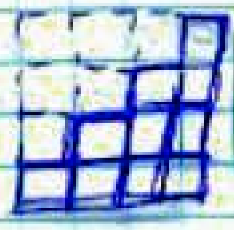
5, 2, 4, 3, 1.

5, 2, 4, 3, 1 ⇒ 2, 5, 4, 3, 1 ⇒ 2, 4, 5, 3, 1 ⇒ 2, 3, 4, 5, 1 ⇒ 1, 2, 3, 4, 5

dla odwrotnie posortowanego

$W(n) = n^2$

z czekoladą



$\int x = \frac{x^2}{2}$

~~Po~~ Liczba porównań stale rośnie

5, 4, 3, 2, 1, $\quad$ 4 por. z 5
4, 5, 3, 2, 1 $\quad$ 3 por z 5 a potem z 4
3, 4, 5, 2, 1 $\quad$ 2 por z 5, potem z 4, potem z 3
itd.

średnia $\quad$ max sortowanie porównań $\quad k-1$
min $\quad$ -||- $\quad$ -||- $\quad$ 1

czyli $A(n) = O(n^2) = W(n)$

ZAD
```
max = tab [0]
for (i=1 ; i<n; i++)
  if (tab[i] > max)
    max = tab [i]
```

$1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = H_n = \Theta(\log n)$

~~Słowa~~ Problemy binarne mają złożoność log n

[6.11.05]

Quicksort

Hoare „Quicksort" w „Computer Journal" 1962.

ciąg binarny 01101010011

mamy robocika który może przestawić (naraz) dwie dane

01101010011

początek L — koniec P

01101010011

00101010111

00001011111

00000111111

złożoność liniowa N

A co z liczbami „nie binarnymi"

1011000

58176243

53176248

53146278

53142678

teraz wprowadzamy 5 jako „jedynkę"

wybieramy ~~jedną~~ pierwszą z liczb i porównujemy z innymi (mniejsze – 0, większe – 1)

[2314] 5 [678]

mniejsze — większe

a teraz już z powrotem możemy brać jako liczbę (2) itd.

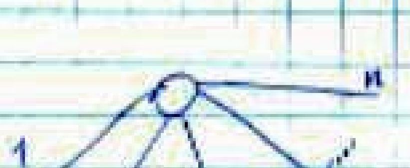
Najbardziej złożone dane dla quicksortu są danymi posortowanymi

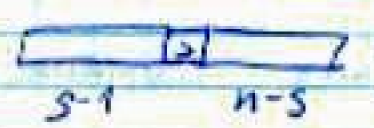
Dla tego przypadku złożoność pesymistyczna jest rzędu n^2

przestrzeń probabilistyczna

Ω = Perm [n]

{1,2,3}, {3,2,1}, {2,3,1} ... itd są równoważne (takie samo prawdopodobieństwo trafienia)





$$A(n) = (n+1) + \frac{1}{n}\sum_{s=1}^{n}\left(A(s-1) + A(n-s)\right)$$

$(n+1)$ — partycjonowanie na części mniejszą i większą

$A(s-1)$ — partycjonowanie lewej części (mniejszych liczb)

$A(n-s)$ — –"– prawej części (większych)

Bardziej cywilizowane postać:

$$A(n) = (n+1) + \frac{2}{n}\sum_{s=1}^{n} A(s-1) \quad \Big| \cdot n$$

$$n A(n) = n(n+1) + 2\sum_{s=1}^{n} A(s-1)$$ jako $n = n-1$ (mogę iść)

$$(n-1) A(n-1) = (n-1)n + 2\sum_{s=1}^{n-1} A(s-1)$$ odejmując te równania dostaję

$$n A(n) - (n-1) A(n-1) = 2n + 2A(n-1)$$

$$n A(n) = (n+1) A(n-1) + 2n \quad \Big| \frac{1}{n(n+1)}$$

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

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

uwaga: $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$

$O(n) = \log n$

$$\boxed{O(\log n)}$$

Zmierzyć się z ~~...~~ implementacją iteracyjną i rekurencyjną quicksortu.

ZAD Sortowanie ~~...~~ monet o różnych wagach na wadze szalkowej (dla 5 monet → 7 ważeń)

Heapsort

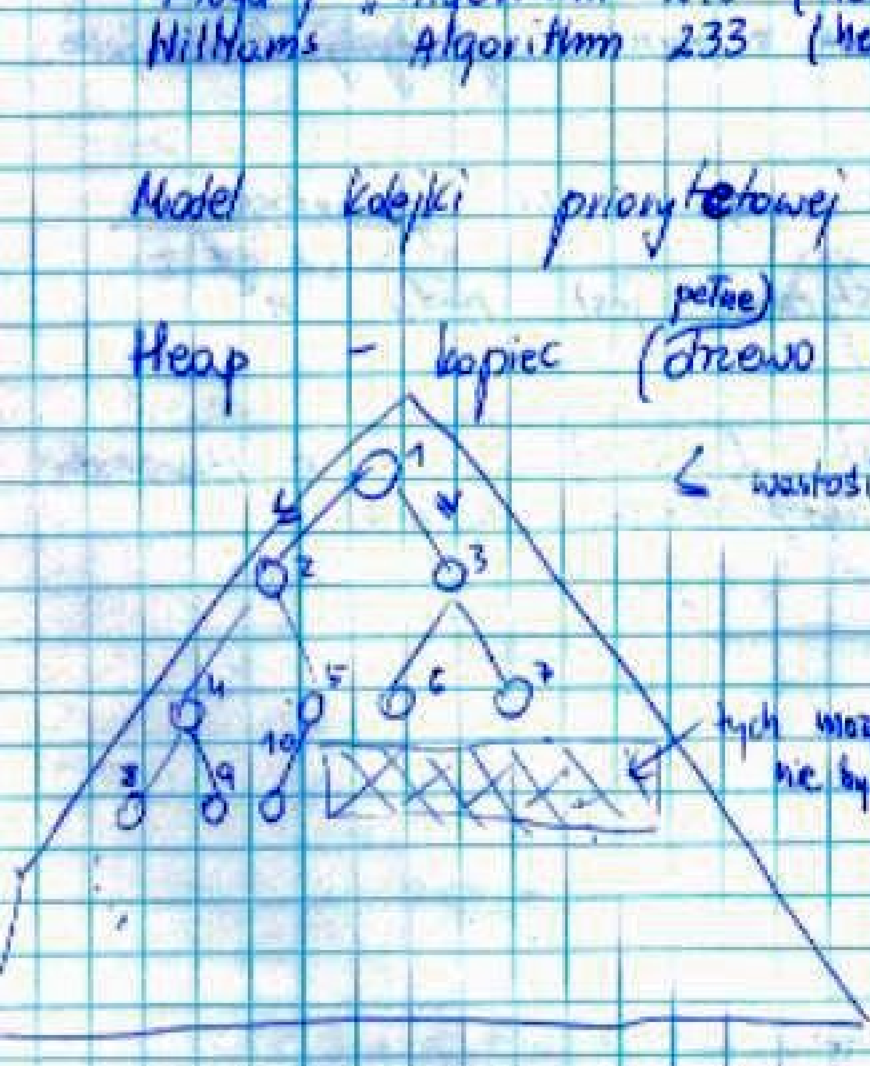
Floyd, „Algorithm 245 (Heapsort) 1964

Williams „Algorithm 233 (Heapsort) 1964

Model kolejki priorytetowej

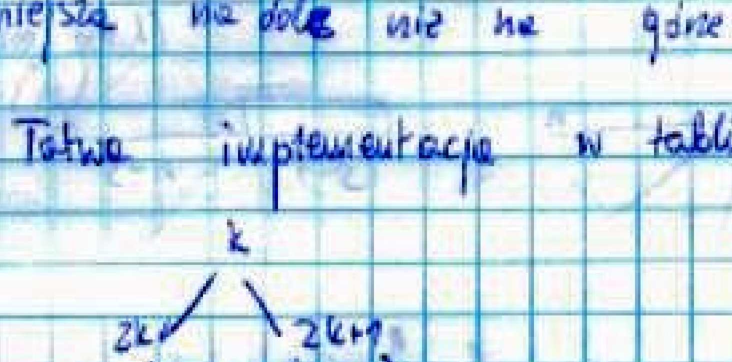
priorytet (w zależności od zadania albo się ustala w priorytet w górę lub w dół)

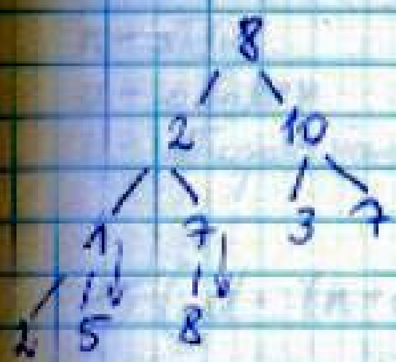
Heap - kopiec (drzewo binarne pełne z wyjątkiem ostatnich liści)



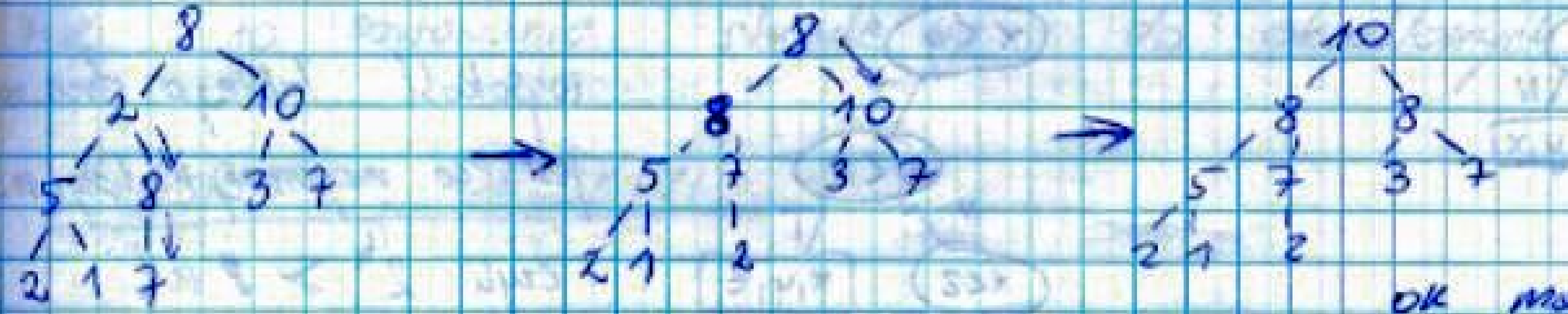
≤ wartość mniejsza na dole niż na górze.

Łatwa implementacja w tablicy





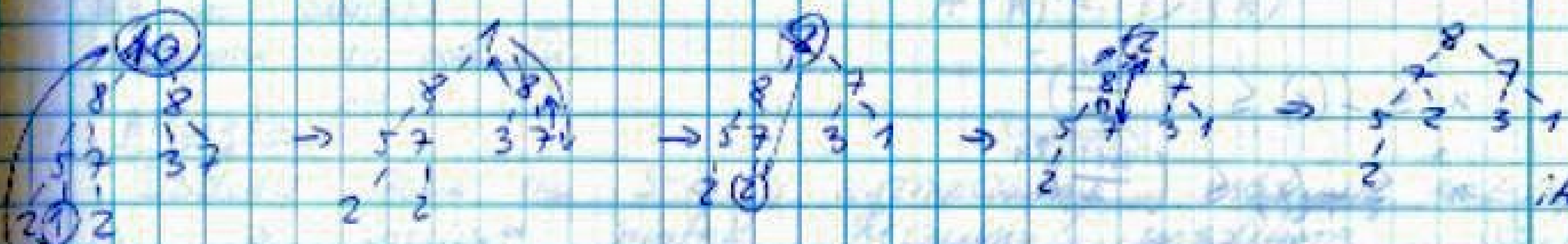
bierzemy delikwenta i jeżeli jest nie na swej pozycji to spychamy go na dół



ok mamy strukturę kopca

Ale sortowanie działa inaczej. Podkradamy maksymalne wyniki sortując na nowo i podkradzione wyniki wykorzystujemy ponownie w tej samej tablicy.

Do sortowania wykorzystujemy liście

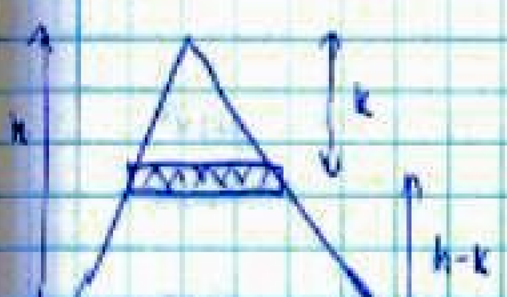


itd.

10, 8, ...

Koszt takiego sortowania to $n \log n$

budowa kopca jest liniowa, ale sortowanie jest logarytmiczne



Wierzchołki z k-tego poziomu mogą być zepchnięte o $h-k$ poziomów

$$\sum_{k=0}^{h} (h-k)2^k = h\underbrace{\sum_{k=0}^{h} 2^k}_{h(2^{h+1}-1)} - \sum_{k=0}^{h} k\cdot 2^k$$

$$2^0 + \dots + 2^h = (2^{h+1}-1)(2-1)$$

Suma

$$S = 1\cdot 2^1 + 2\cdot 2^2 + 3\cdot 2^3 + \dots + h\cdot 2^h$$

$$2\cdot S = 1\cdot 2^2 + 2\cdot 2^3 + 3\cdot 2^4 + \dots + h\cdot 2^{h+1}$$

odejmujemy

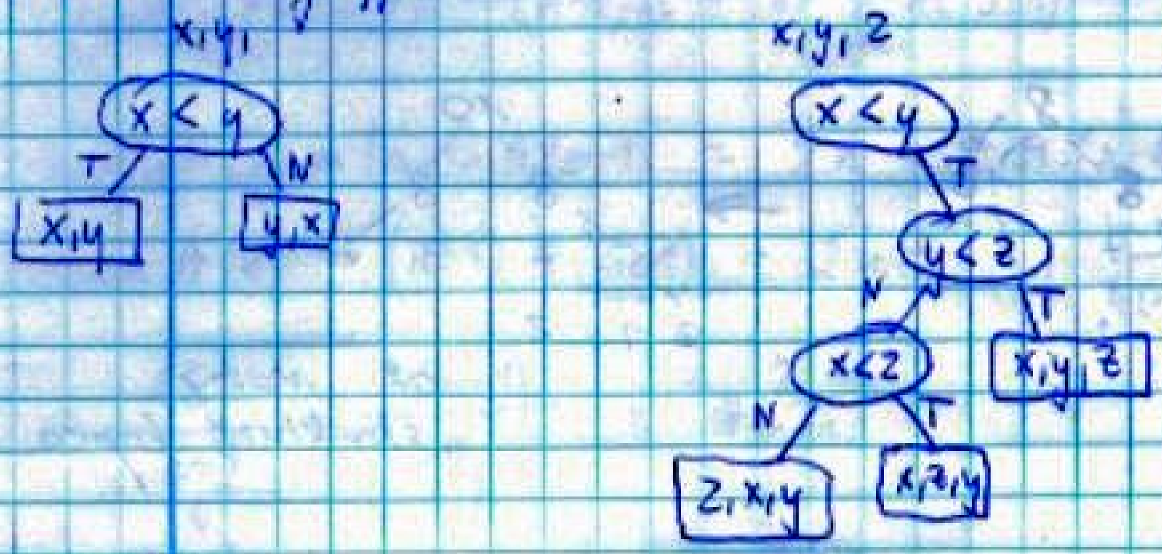
$$-S = 1\cdot 2^1 + 2^2 + 2^3 + \dots + 2^h - h\cdot 2^{h+1}$$

$$S = \cancel{h\, 2^{h+1} - h} \; 2^{h+1} - 2 - h\cdot 2^{h+1}$$

$= -n + 2^{n+1} - 2 = 2^n$?

Złożoność liniowa $O(n)$

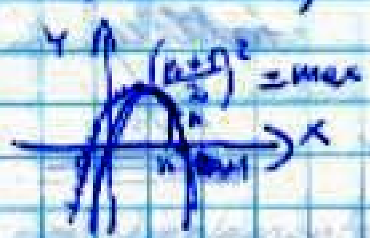
Drzewa decyzyjne



wysokość takiego drzewa jest co najmniej $h \geq \log(n!)$ czyli $2^h \geq n!$

$(n!)^2 = (1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n)(1 \cdot 2 \cdot 3 \cdot \ldots \cdot n) = (1 \cdot n) \cdot (2 \cdot (n-1)) \cdot (3 \cdot (n-2)) \cdot \ldots$

$f(x) = x(n+1-x)$



min = n

$n \leq f(x) \leq \left(\frac{n+1}{2}\right)^2$

$n^n \leq f(x) \leq \left(\frac{n+1}{2}\right)^{2n}$

$\Downarrow$

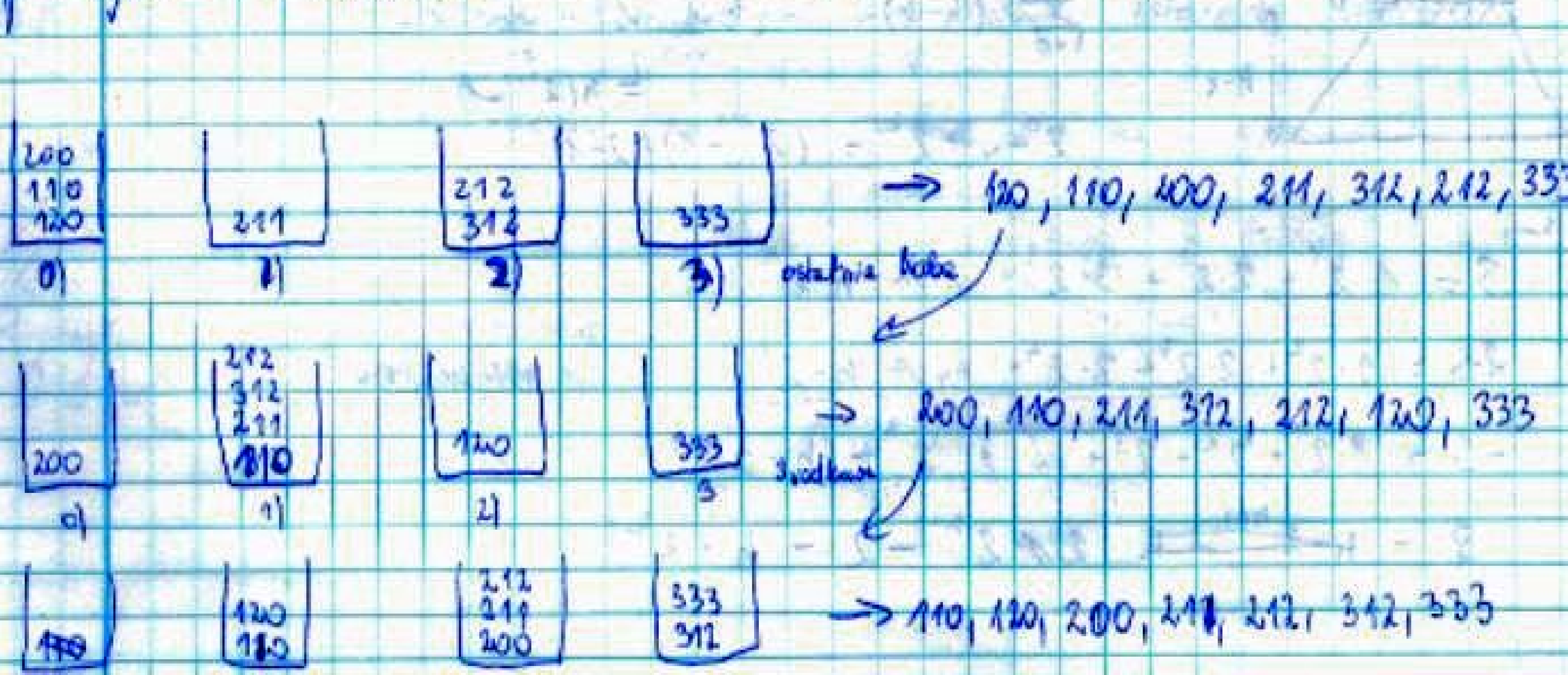
$\log(n!)$

Czyli tak czy owak nie przekroczymy $n \log n$

Sortowanie pozycyjne

ZAD 120, 312, 110, 211, 200, 333, 212,

Założenie: Duże tablice ale wąski zakres danych np tylko 5 możliwości



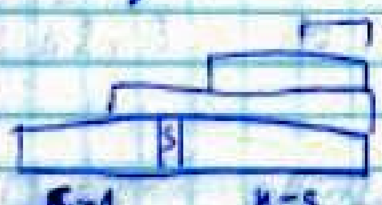
n - słów 7 (200 : 4)
a - alfabet 4 (0,1,2,3)
d - długość słowa 3 (200)

$\mathcal{O}(d \cdot (n+a))$ jeżeli d, a - const to $\mathcal{O}(n)$

Ale to sortowanie stosuje się do określonych danych!

ALGORYTMY SELEKCJI

np. jak wyszukać medianę, bo max, min liczymy w czasie liniowym.



Wybór elementu do podziału → pierwszy. Nieskuteczny dla posortowanego

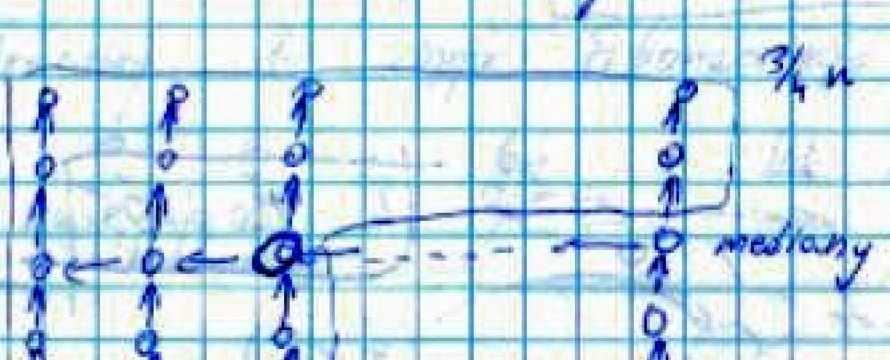
(partitioning i sortowanie jest skuteczne w czasie $W(n) = \mathcal{O}(n^2)$

$A(n) = \mathcal{O}(n)$

Nie ma dobrego elementu do podziału.

Algorytm 5°

Dzielimy dane na 5-ki, wyznaczamy medianę każdej piątki, i z median piątek liczymy medianę wszystkich.



β mniejszy niż następnik

$W(n) = c_1 n$

WAŻNE:

$W(n) \le \underbrace{20 \max\{c_1, c_2\}}_{c}$ zgodnie z zał. indukc.

$W(n) = \underset{\substack{\uparrow \\ \text{sortowanie piątek} \\ \text{liniowe}}}{c_2 n} + \underset{\substack{\uparrow \\ \text{wyszukiwanie mediany median} \\ \text{liniowe}}}{W\left(\frac{n}{5}\right)} + \underset{\substack{\uparrow \\ \text{oszacowanie 3/4 tabeli} \\ \text{liniowe}}}{W\left(\frac{3}{4} n\right)}$

dla $n \le 50$

$W(n) \le c_2 n + c\left(\frac{n}{5}\right) + c\left(\frac{3n}{4}\right)$

$n\left(c_2 + c\left(\frac{1}{5} + \frac{3}{4}\right)\right)$

$n\left(c_2 + \frac{19}{20} c\right)$

$n\left(c_2 + 20 \max\{c_1, c_2\} \cdot \frac{19}{20}\right) \leq n\left(c_2 + 20 \max\{c_1, c_2\}\right)$ ok.

Dzielimy na 5 bo tylko wtedy jest spełnione te zadanie.

Oszacowanie z dołu → nie może być lepsze niż.

ZAD Dominanta → ta opcja która jest częściej niż w połowie

```
x = a[0];   l = 0;
for (i=1; i<n; i++)
   { if a[i] = x    l++ ;
     else { if (l)  l-- ;        if(l) -zwykły-> l > 0
     else x = a[i] ; }
```

Analiza efektywności algorytmów sortowania
długość tablic etc.

Prog. dynamiczne

4. 12. 05

Ciąg Fibonacciego $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

Aby go wyliczyć należy zastosować rekurencję, ale jest to bardzo pamięciożerne (złożoność wykładnicza)

Aby to obejść lepiej zastosować iterację.

$0, 1, 1$. przepisujemy

$1, 1, 2$ ← dodajemy dwie poprzednie

$1, 2, 3$ itd. ... złożoność (n)

Chytra sztuczka chcemy liczyć x^n

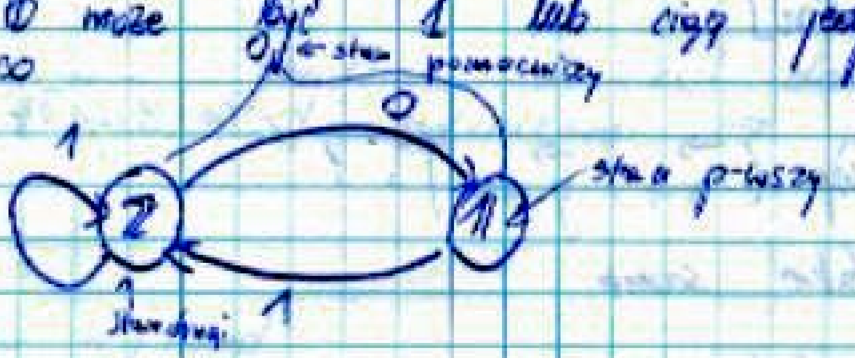
$x^{13} = x^{1+4+8}$

x^1, x^2, x^4, x^8 do kwadratu złożoność $(\log n)$

$\underbrace{x \cdot x \cdot x \cdot \ldots \cdot x}_{12 \text{ razy}}$ złożoność (n)

Wracamy do ciągu Fibonacciego

Po 0 może być 1 lub ciąg jedynek, możemy to zilustrować następująco



Macierz przejścia ze stanu 1-go do drugiego

$$\begin{array}{c c} & \begin{array}{cc} 1 & 2 \end{array} \\ \begin{array}{c} 1 \\ 2 \end{array} & \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \end{array}$$

Zobaczmy jak w przypadku potęgowania.

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{bmatrix} = \begin{bmatrix} f_n & f_{n+1} \\ f_{n+1} & f_{n+2} \end{bmatrix}$$

i tak możemy potęgować uzyskując kolejne wyrazy ciągu Fib.

A^n $O(\lg n)$

Z historii wiemy, że $f_n = \frac{1}{\sqrt{5}}(\alpha^n + \beta^n)$ gdzie α, β niewymierne

coś $\frac{1+\sqrt{5}}{...}$ ...

Możemy także korzystać ze wz. Newtona

$\binom{n}{k}$ d. Newtona

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1} \qquad \binom{n}{n} = \binom{n}{0} = 1.$$

Tw. Pascala (mówiliśmy o tym już wcześniej)

W quicksorcie rekurencja jest ok, w pozostałych przypadkach uciekać od niej jak najdalej.

Trzeci przykład wykorzystania prog. dynamicznego

najdłuższy wspólny podciąg — mamy ciąg abacbacb i wybieramy podciąg aabb

$nwp(\vec{x}, \vec{y})$

np. dwa ciągi abacbacb

baccaba

nwp = bacba

ciągi o długości n i m. ostatnie ~~litery~~ znaki dla ciągów są takie same

$$x[n] = y[m]$$

wtedy

$$nwp(\vec{x}_n, \vec{y}_m) = nwp(\vec{x}_{n-1}, \vec{y}_{m-1}) + 1$$ bo liczy się ten ostatni znak

gdy znaki są nie takie same

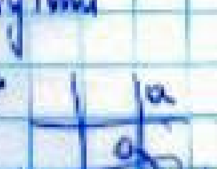
wtedy musimy opuścić ostatni znak i szukać rozwiązania. Ponieważ mamy dwa ciągi, więc robimy to dla każdego oddzielnie i wybieramy rozwiązanie lepsze

$$nwp(\vec{x}_n, \vec{y}_m) = \max \begin{Bmatrix} nwp(\vec{x}_n, \vec{y}_{m-1}); \\ nwp(\vec{x}_{n-1}, \vec{y}_m) \end{Bmatrix}$$

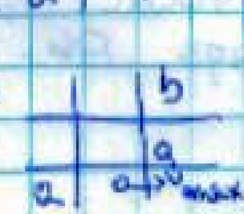
Wracamy do naszych ciągów i prog. dynamicznego

Wzór algorytmu

Z zasady 1°



2°



różne = 0
takie same = 0+1

	a	b	a	c	b	a	c	b
b	0	1	1	1	1	1	1	1
a	1	1	2	2	2	2	2	2
c	1	1	2	3	3	3	3	3
c	1	1	2	3	3	3	4	4
a	1	1	2	3	3	4	4	4
b	1	2	2	3	4	4	4	5
a	1	2	3	3	4	5	5	5

$\Theta(n \cdot m)$

$O(n+m)$

← max ale litery różne, więc są max-y z góry lub z boku.

idziemy do góry itd.

wychodzi nam bacch

ale zauważmy, że gdybyśmy poszli w lewo dostaniemy inny ciąg
bacba

A co z problemem ciągu rosnącego np z ciągu 7, 2, 4, 8, 6, 9, 1, 10 wyznaczyć ciąg rosnący np 7, 8, 9, 10, itd ale ma być najdłuższy. Jak to zrobić? Porównać go z ciągiem porządkowym 1, 2, 3, 4, 5, 6, 7 ... i znaleźć nwp.

Zad. ucz, cieszy, bawi i Mamy dwa teksty ⇒ z dwóch tekstów wyznaczyć jeden tekst (zawierający oba) ale najkrótszy!

np abcab ⇒ abcccab
accca

Metody zachłanne (greedy)

Zadania typu przydział lekcji do sal, → jak najwięcej lekcji w sali
pokrywanie obszaru odcinkami → jak najmniej aby całość zakryć.

1948 Shannon / Fano

kompresja pliku

A	B	C	D	E
0,35	0,17	0,17	0,16	0,15
0	0	1	1	1
0	1	0	1	1
			0	1

1 znak = 8 bitów

Kompresja tekstu → gdy znaki równomiernie ⇒ wszystkie po 1B
ale gdy np. $a = 3 \cdot b$ to opłaca się zmniejszyć wielkość „a" kosztem „b" by w rezultacie zyskać

bo $3a + b = 4$ (normal)
$3a + b = 2\frac{2}{3}$ (kompresja)

$a = \frac{1}{3}$
$b = 1\frac{2}{3}$

Powyżej tabelka S-F która nie jest optymalna

Algorytm Hoffmana

1° Ustal częstość powtarzania się znaku

(f, 5), (e, 9), (c, 12), (b, 13), (d, 16), (a, 45)

2° Usuń dwie najmniejsze, scal i wstaw do kolejki

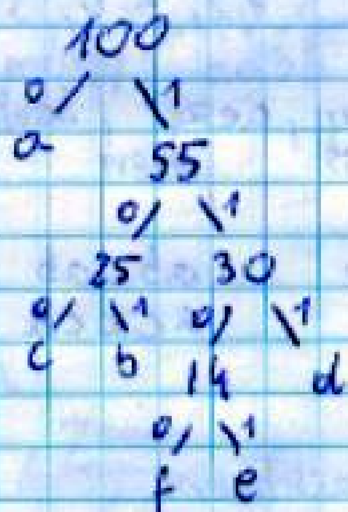
(c, 12); (b, 13); (14) f e; (d, 16); (a, 45)

3° Powtórz do otrzymania kopca

(14) f e; (d, 16); (25) c b; (a, 45)

(25) c b; (30) (14) f e d; (a, 45)

4° Uzyskujemy kopiec



5° Kopcowi nadajemy strukturę wartości 0, 1

6° Kodowanie jest następujące

a	b	c	e
0	101	100	1101

$\frac{\# out}{\# in}$ → minimalne dla k. H.

Wracamy do prog. dynam.

<u>Pattern matching</u>

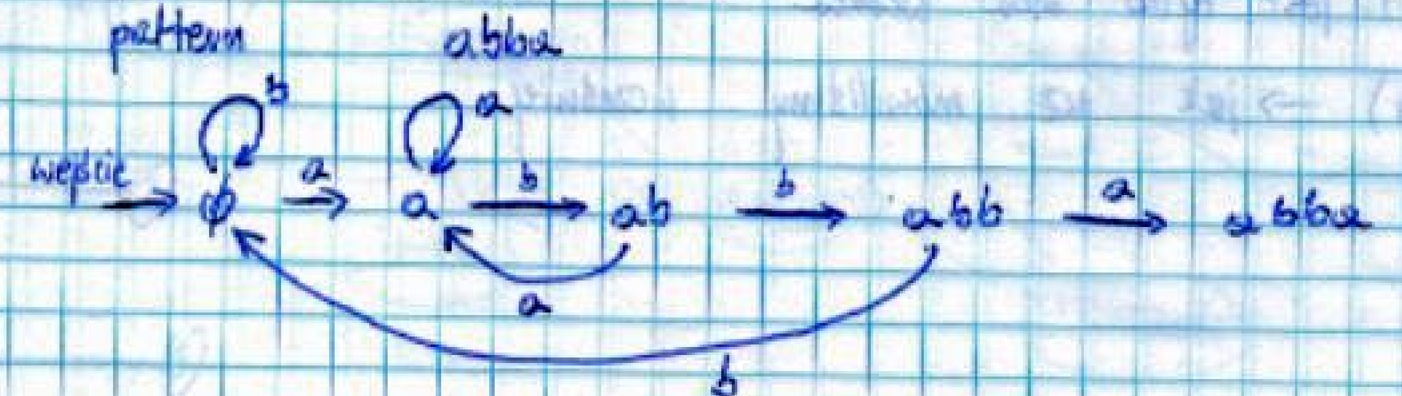
Masz wzorzec → znajdź go w tekście lub czymś podobnym.

1° Algorytm naiwny

porównaj i jak nie, przesuń w prawo o 1

problem z długimi 11... 110 / 11...10 wtedy złożoność kwadratowa

2° Metoda ze switchem.

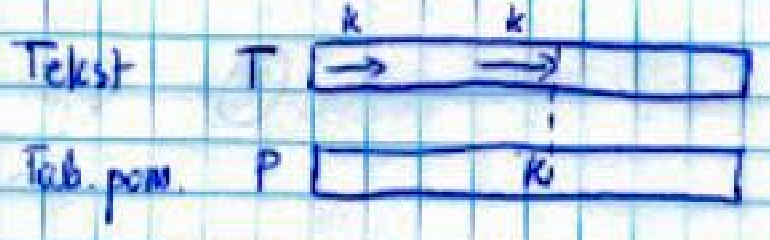


Problem → dla alfabetu jest bardzo czułe, dobre dla małych wzorców, dla długich nie

Aczkolwiek wyliczono że istnieje kod o $\Theta(n+m)$ ← długość wzorca, długość tekstu

Jakie jest rozwiązanie?

3° Stworzenie tablicy pomocniczej



prefix - sufix

k jest patternem który można odszukać jeszcze gdzieś w tekście
jego długość wpisujemy w tab. pomoc.

Przykład

prefix sufix

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

T a b c a b a a d b b a b c a b c

P 0 0 0 0 1 2 1 1 0 0 0 1 2 3 4 5 3 abc

a jest patternem dla a; jest a; jest ab; jest a; jest ab; jest abc; abca; abcab

Uwaga: z reguły ustalamy na początku jakiś wzorzec.

Fragment kodu:

```
P[0] = P[1] = 0 ; t = 0 ;
for (j=2 ; j < m ; j++)
{ while ((t > 0) && (x[t+1] = x[j]) t = P[t];
  if ( x[t+1] == x[j]) t++ ;
  P[j] = t ;
}
```

Na tym poziomie jest to algorytm <u>liniowy</u>

Algorytm KMP

Tablica P jest tylko dla wzorca

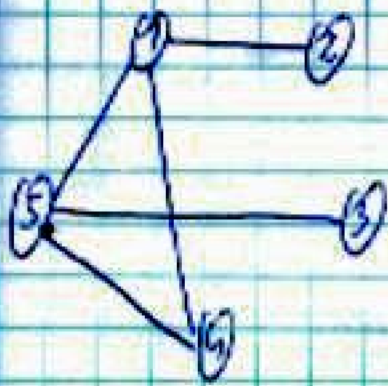
$O(n+m)$ → jak już mówiliśmy wcześniej

15.01.06

$\#V = n$ $\#E = m$

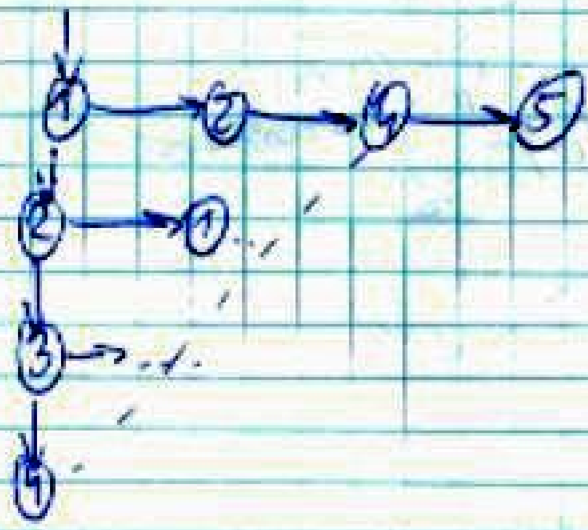
15.01.06

Graf $G(V, E)$ $E \in V^2$



$\{a, b\}$

	{1,2}	{1,5}
1	1	1
2	1	0
3	0	0 ...
4	0	0
5	0	1



Algorytmy grafowe

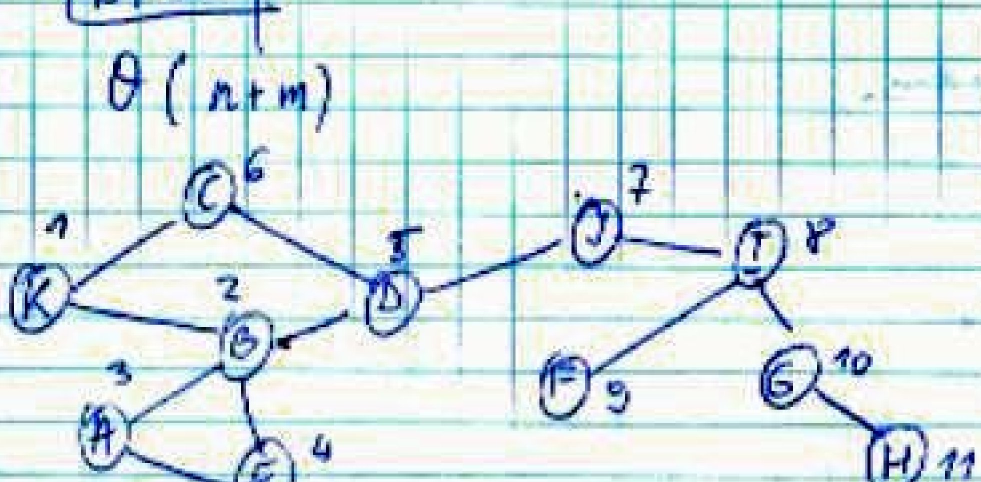
Analogia do labiryntu

Nić Ariadny → stos (lista dynamiczna)

Przeszukiwanie w głąb (patrz materiały p. Sikorskiego)

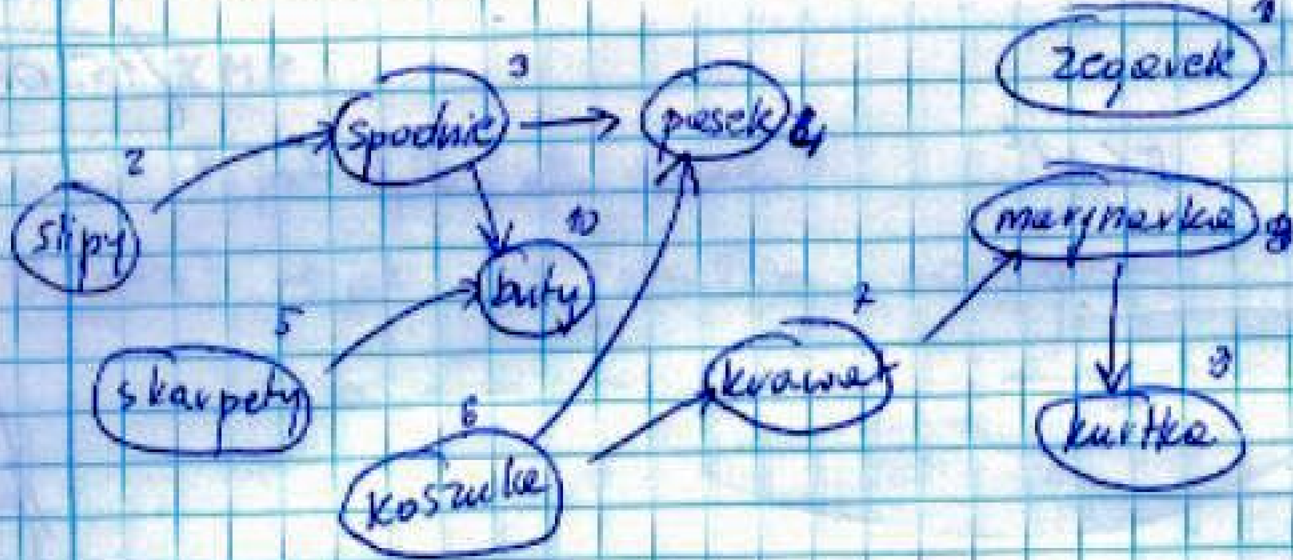
$\boxed{\text{DFS}}$

$\Theta(n+m)$



KBAEDCJIFGH

Sortowanie topologiczne

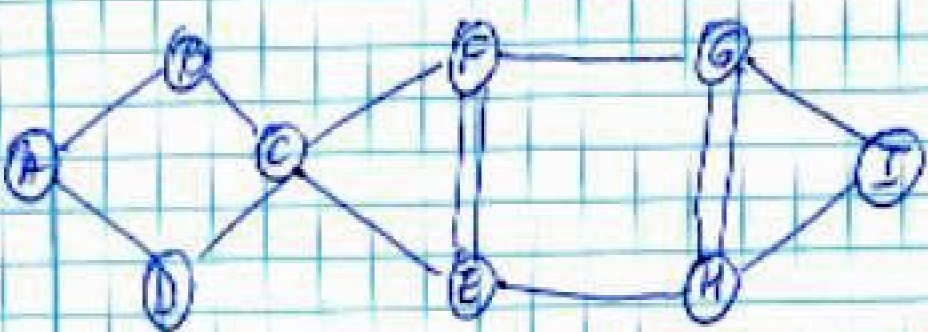


$i < j$

Sortowanie topologiczne jest ok, ale w przypadku, gdy mamy naniesione <u>wszystkie</u> krawędzie

$\Theta(n+m)$

<u>Euler</u> $CE \Leftrightarrow \forall v \in V \quad 2 \mid deg(v)$ ← liczba sąsiadów v

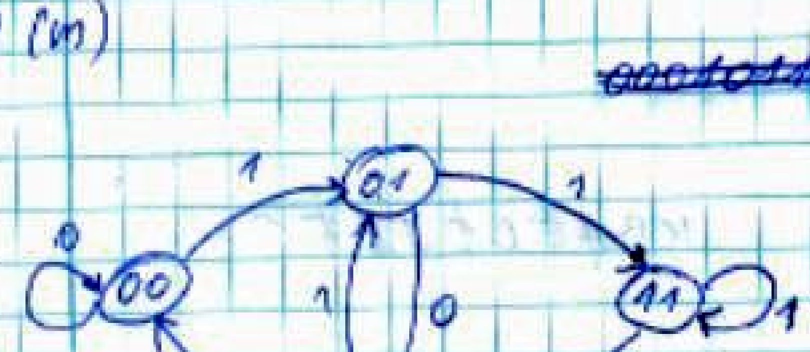


Stos wyboru A D C B A F E C F G H E G I H i teraz od końca zrzucamy na backup

backup A B C E H I G H G F E F C D A → mamy cykl Eulera

$\Theta(m)$

~~00010111~~

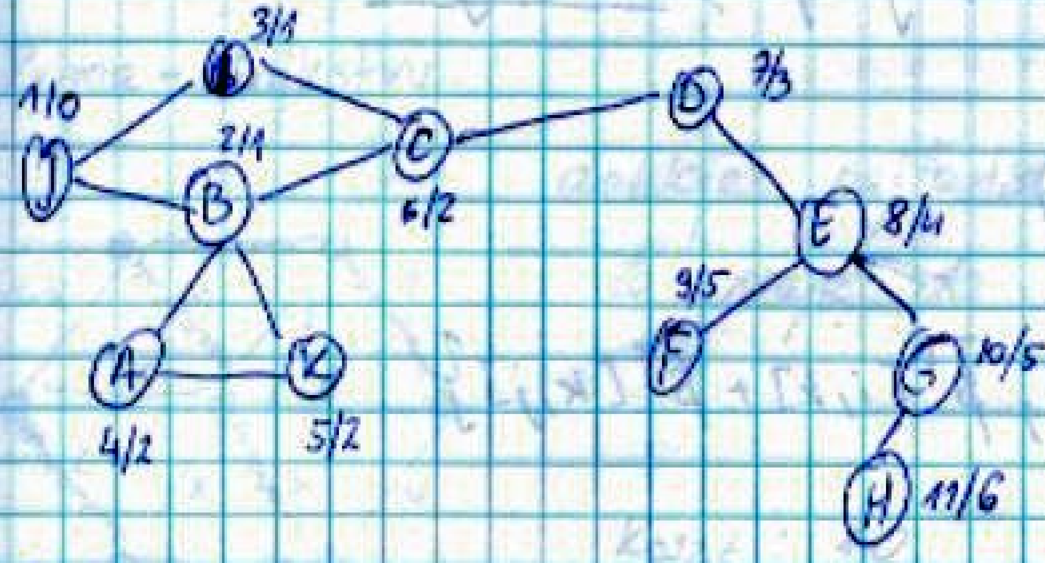


Ciąg deBruijn? $\underbrace{00}_{00}\underbrace{010111 00}_{\text{krawędzie}}$

Przeszukiwanie wszerz

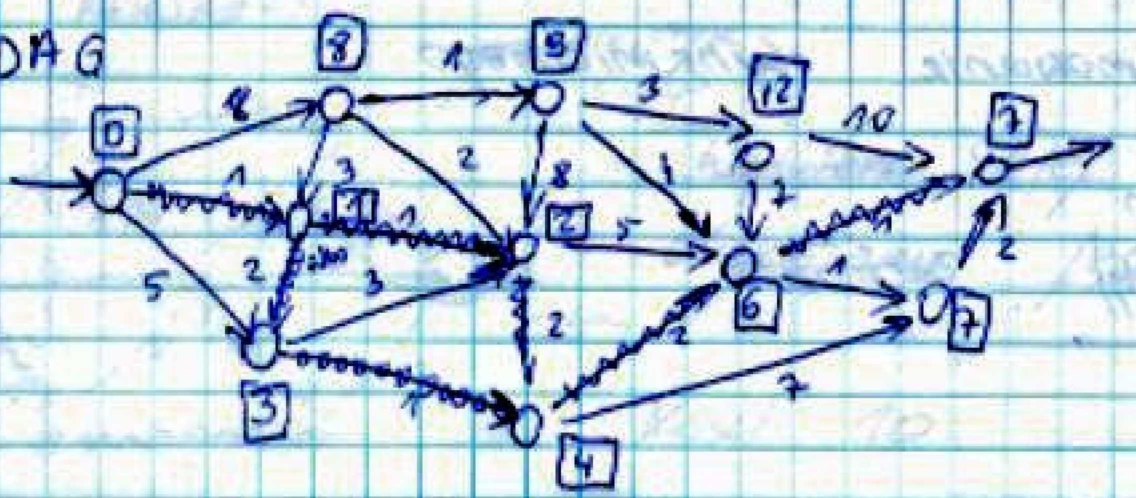
BFS

Za pomocą kolejki



$\Theta(n+m)$

DAG

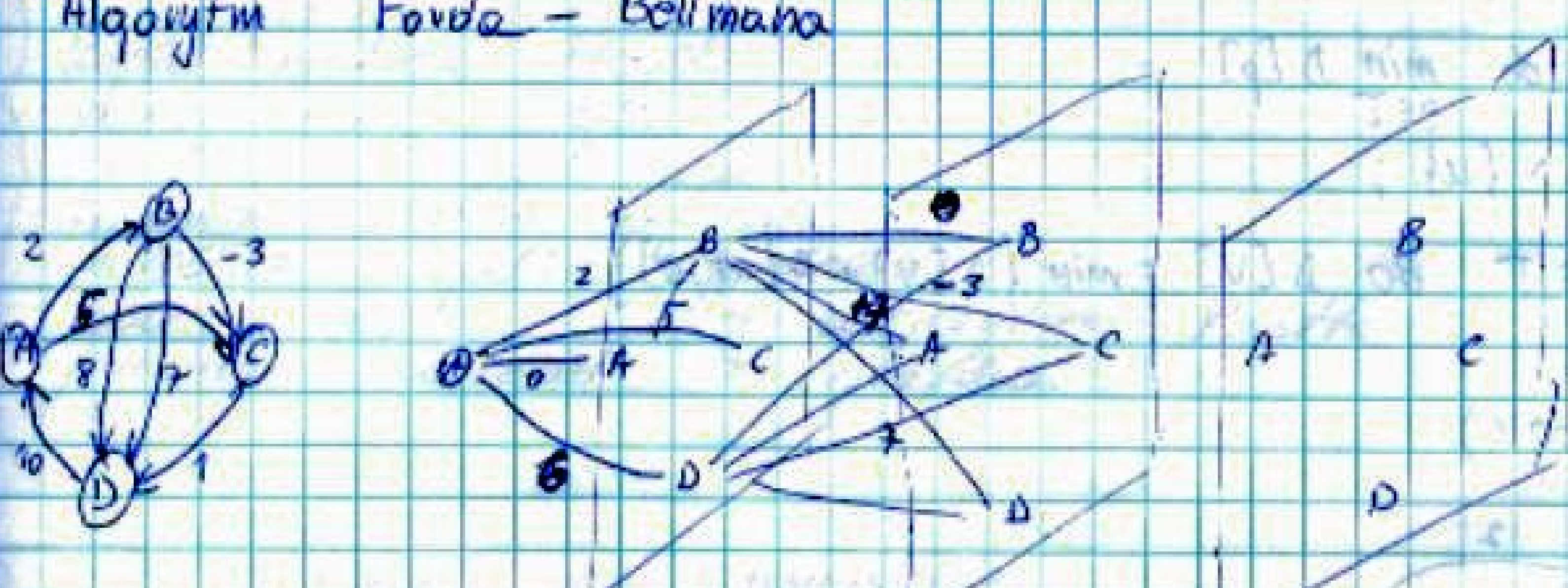


Coś na kształt przepływu w sieci

PERT

CEMP

Algorytm Forda - Bellmana



itd. ...

$\Theta(n^3)$

$$\$^m[i] = \min_j \{ \$^{m-1}[j] + A[j,i] \}$$

jeśli było by więcej

to kopiujemy dalej ...

ale jest to skrót myślowy,

tak naprawdę wystarcza

jedna tablica

Algorytm ~~Floydaa~~ Floyda

(wcześniej pomysł Kleene'ego)

Wylicza najkrótszą drogę między parą dowolnych wierzchołków

$$\delta^k[i,j] := \min\{\delta^{k-1}[i,j], \delta^k[i,k] + \delta^k[k,j]\}$$

for k
 for i
  for j

Korzysta z programowania dynamicznego

Algorytm Dijkstry (Greedy?) zachłanne

Wagi ≥ 0 !

for $v \in V$ do $D[s] = A[s,v]$,

$D[i] = 0$;

$T = V \setminus \{s\}$

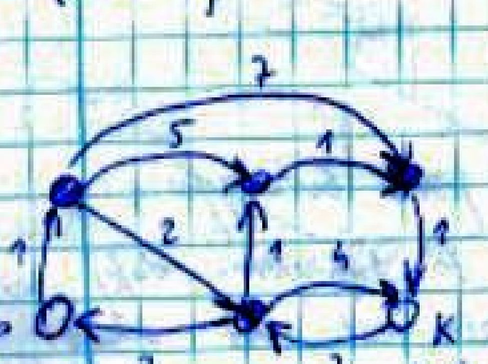
while $(T \neq \emptyset)$ do

$u := p$ do $\min_{p \in T} D[p]$

$T := T \setminus \{u\}$;

for $v \in T$ do $D[v] = \min\{D[v] + A[u,v]\}$

$\Theta(m \log n)$

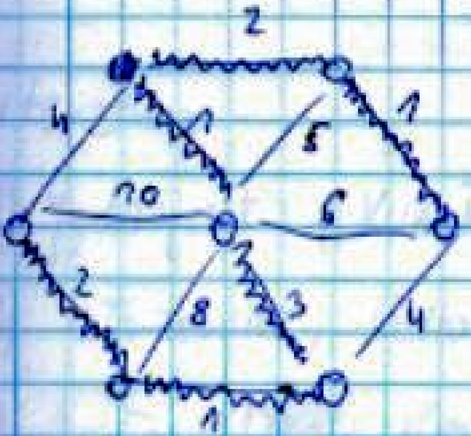


Drewo rozpinające graf

SKANY
dla Tomka Szeszko

MST

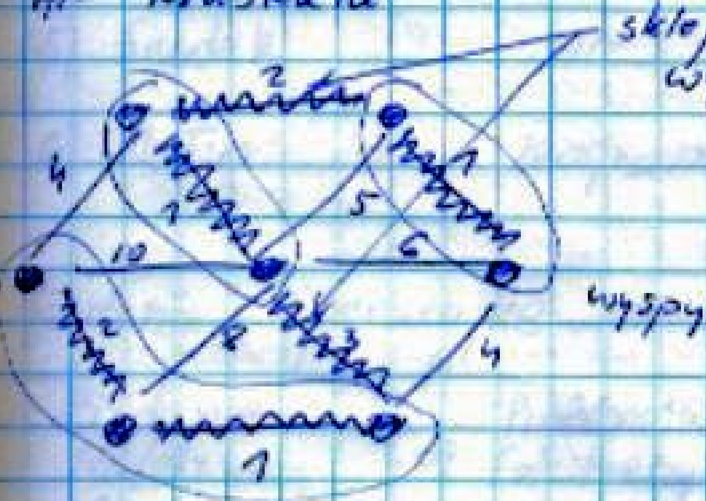
Al. Prima - Dijkstry



dokleja najtańszą krawędź z tych co mają do wyboru

Koszt: 10

Al. Kruskala



~~Wybierz~~ Sklej wyspy których koszt doklejenia jest najtańszy

Koszt: 10

UWAGA: UNIKAMY CYKLI!
(bo to ma być drzewo)

3
- radixsort → sortowanie pozycyjne
- mergesort
- kopiec metodą Floyda
- partitioning qui
- BSD
- słownik kodów Hoffmanna
- najdłuż. wspólny podciąg
- oblicz tablicę p z algorytmu Pratta
- pytanko koszęcel z grafów.

4,5 → zadanie na pomyślenie

masz coś dane / wymyśl algorytm, policz złożoność