

ALG

Definice Big-O - upper bound (horní ohrada)

$$f(n) \in O(g(n)) \text{ iff } \exists (c, n_0) \forall n \geq n_0 f(n) \leq c \cdot g(n)$$

Definice. Big- Ω - lower bound (dolní odhad)

$$f(n) \in \Omega(g(n)) \text{ iff } \exists(c, n_0) \forall(n) \geq c \cdot g(n) \quad \forall n \geq n_0$$

Definition (H) - average bound

$f(n) \in \Theta(g(n))$ iff $\exists(c_1, c_2)$ $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$$\text{D}(g(n)) = \alpha(g(n)) \cap S(g(n))$$

Pr

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \dots < 2^n < n^n$$

lower bound ↑ upper bound
 avg bound

$$f(n) = n$$

- $f(n) \in O(n)$, $f(n) \in O(n \log n)$, $f(n) \in O(n^2)$.
 - $f(n) \in \Omega(n)$, $f(n) \in \Omega(\sqrt{n!})$
 - $f(n) \in \Theta(n)$

Věta Master theorem, koukej ^{na} lekce DMA - konec.

Řeší: $T(n) = \alpha T(n/b) + f(n)$, $\alpha > 1$; $b > 1$; $f(n)$ asymp. kladn.

1) Pokud $f(n) \in O(n^{\log_b(a) - \varepsilon})$ pro $\varepsilon > 0$, pak $T(n) \in O(n^{\log_b(a)})$

2) Pokud $f(n) \in \Theta(n^{\log_2(\alpha)})$, pak $T(n) \in \Theta(n^{\log_2(\alpha)} \log(n))$

3) Pokud $f(n) \in \mathcal{S}(n \log_b(a) + \varepsilon)$ pro $\varepsilon > 0$ a pokud

$af(n/b) \leq c f(n)$ pro niewiąż $c < 1$ a wówczas $T(n) \in \Theta(f(n))$

Převod rekurence na primé vyjádření složitosti

1. Master theorem

2. Metoda rekurzivního stromu

1) Nakreslíme strom rekurse

2) Sečteme values všech uzelů

3) Součet vš uzelů dava složitost $T(n)$

3. Substituční metoda

Odhadneme přesný tvar řešení \rightarrow Matematicky dokážeme

Stromy

$$\cdot G = (V, E)$$

• souvisly graf bez cyklů

• každe dva uzel spojuje jedna cesta

$$\cdot |E| = |V| - 1$$

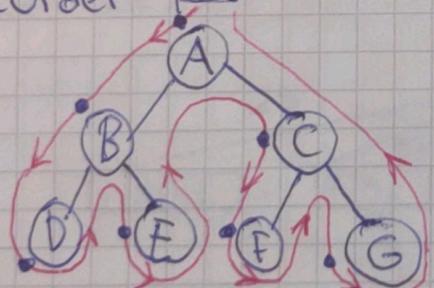
Binary tree

\hookrightarrow regular (pravidelný) - počet potomků - $0/2$

\hookrightarrow balanced (vyvážený) - hloubka je $\lceil \log_2(n+1) \rceil - 1 = h$

Průchod stromem DFS (Depth-first search) a BFS

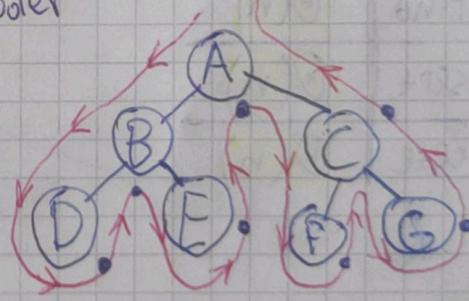
a) preorder



A B D E C F G

tisk - první navštěva

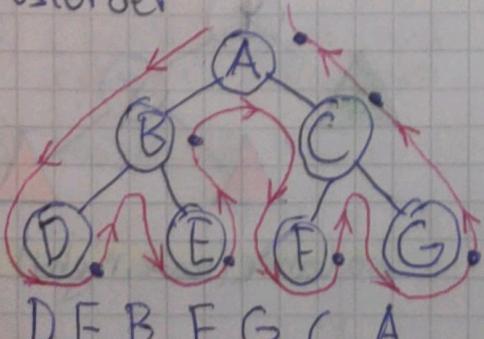
b) inoder



B D E A F G C

tisk - druhá navštěva

c) postorder



D E B F G C A

• DFS - $\Theta(|V| + |E|)$

• BFS - $\Theta(|V| + |E|)$

Array search

- Linear - $\Theta(n/2)$, $O(n)$
- ~~Interpolation~~ search ~~$\Theta(\log n)$~~ $O(\log \log n)$

↳ sorted array

$$\text{position} = \text{first} + \frac{q - a[\text{first}]}{a[\text{last}] - a[\text{first}]} \cdot (\text{last} - \text{first})$$

kde q - hledaný element, first/last - indexes of current segment

- Binary search $\Theta(\log n)$

Binary search tree

- Vlastnosti: $\text{node.left} < \text{node}$
 $\text{node.right} > \text{node}$
- Delete operation: mame smazat node se dvema potomky
~~node-to-del~~

$$\text{node-to-del} = \begin{cases} \text{get_min}(\text{node.right}) \\ \text{or} \\ \text{get_max}(\text{node.left}) \end{cases}$$

$$\text{node.value} = \text{node-to-del.value}$$

`delete(node-to-del)`

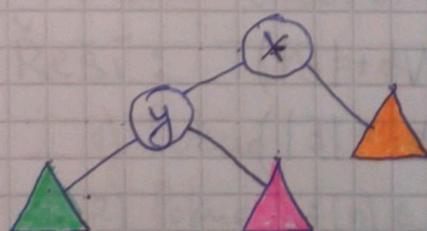
- Time

Find	$\Theta(n)$
Insert	$\Theta(n)$
Delete	$\Theta(n)$

AVL tree

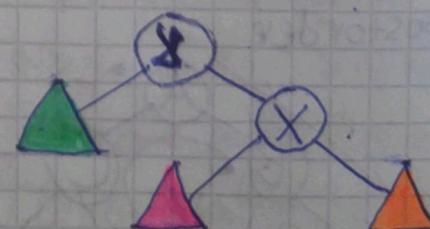
- Rotations

↳ R-rotation



$$\text{balance}(\text{node.left}) \geq 0$$

L-rotation ↵



$$\text{balance}(\text{node.right}) \leq 0$$

↳ LR-rotation

- Na začátku L, potom R
- $\text{balance}(\text{node.left}) < 0$

RL-rotation ↳

- Na začátku R, potom L
- $\text{balance}(\text{node.right}) > 0$

• Vlastnosti

$$\hookrightarrow h = \text{max} \cdot h(\text{node.left}) - h(\text{node.right})$$

$$h = -1/0/1$$

↳ Po insert/delete musíme přebalancovat

• Time

Find - $\Theta(\log n)$

Insert - $\Theta(\log n)$

Delete - $\Theta(\log n)$

B-tree

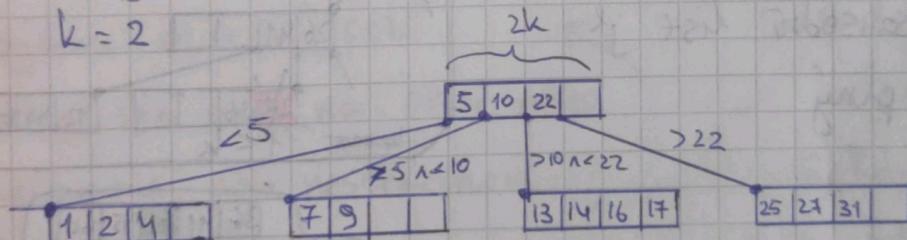
• Vlastnosti

↳ Každý uzel obsahuje min k a max 2k sorted klíčů

↳ Každý uzel má 0-1 více potomků, než má klíčů. (Kromě root)

↳ Všechny cesty z root do listu jsou stejně dlouhé.

• Příklad $k=2$



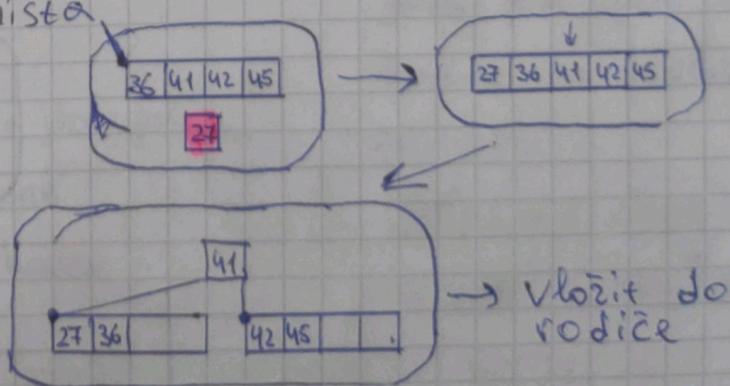
• Insert, když mají volná místa

↳ Rodič má volné místo

1) Sort all elements

2) Choose median and create new node

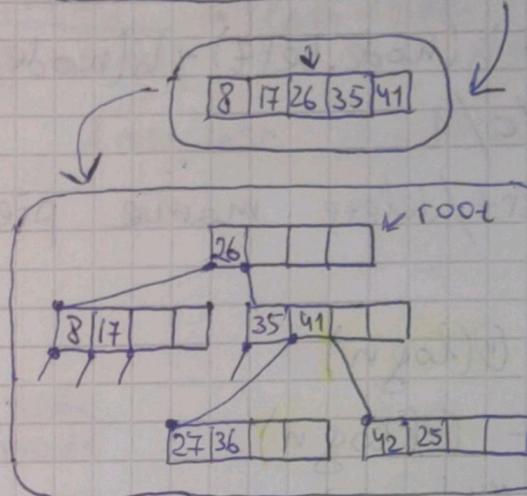
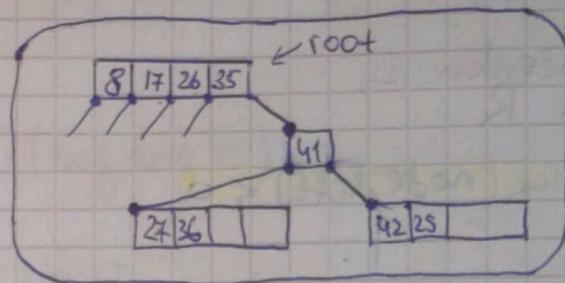
3) Vložit do rodiče



↳ Rodič nemá volné místo

1) Na začátku děláme
jako v předchozím
případě

2) Analogicky postup s
vložením 41 v kořen



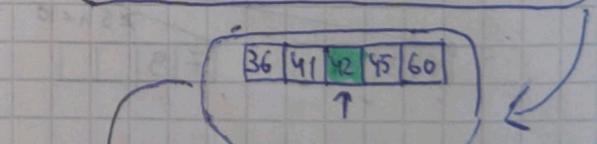
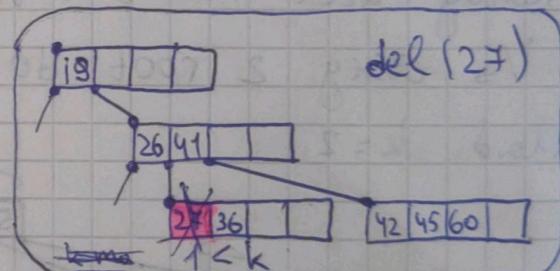
- Delete operation

↳ delete ve vnitřním uzlu

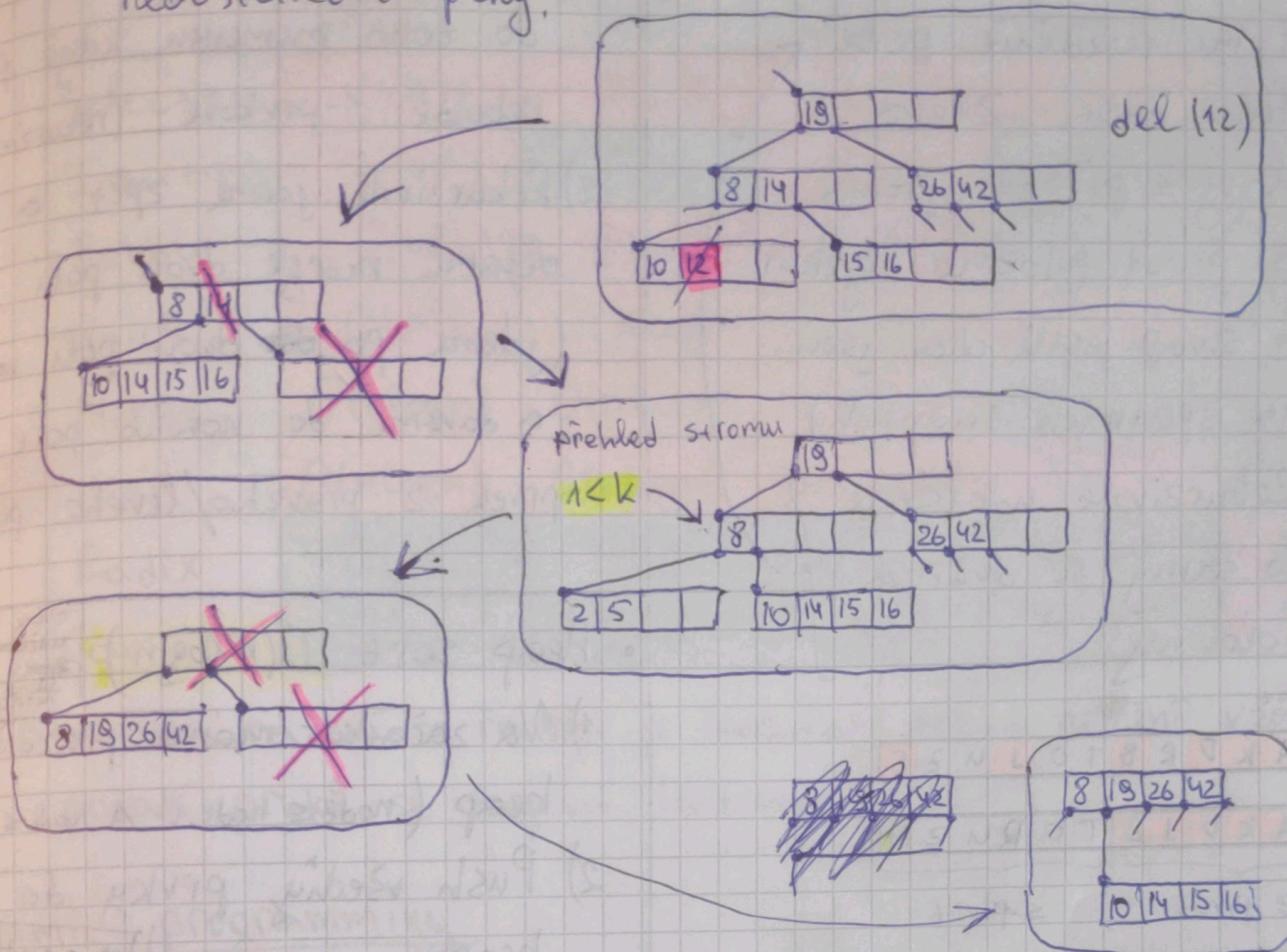
Máme najít největší (nejmenší) uzel v levém (pravém) uzlu
jako BST.

↳ přebalancování po delete

1) Mazání v nedostatečně plném
listu, ale sousední list je
dostatečně plný



2) Mazání v nedostatečně plném listu; ani sousední list nedostatečně plný.



Sorts

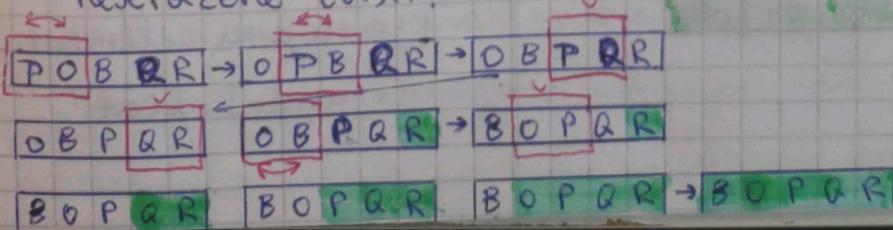
• Selection sort $O(n^2)$

- 1) Najdeme min v poli
- 2) Dáme ten min na první místo
- 3) Koukáme na nesřazenou částku pole }

• Bubble sort $O(n^2)$

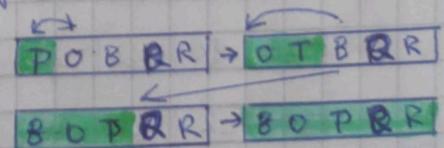
- 1) Koukáme na 2 elementy a jdeme poli
- 2) swap, jestli jsou nesřazene
- 3) Cíl: doložit max do konce

nesřazene časti.



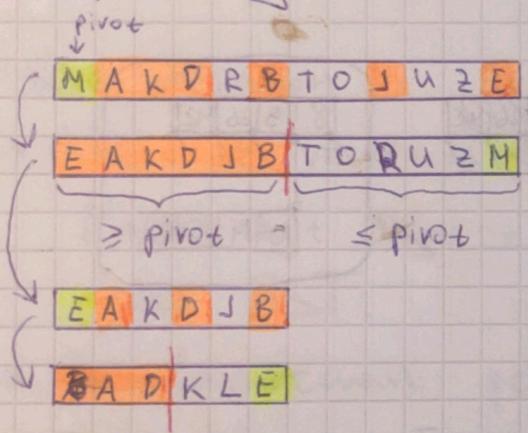
• Insertion sort $O(n^2)$

- 1) Zlepšovacího nebožího elementu
- 2) Když vidíme nějaký ještě nepřeložený element, jdeme poli zpět a dleláme na příslušnou pozici



• Quick sort $O(n^2)$

- 1) Nahodně zvolíme pivot
- 2) Máme rozdělit pole
 - na 2 části. Zleva \leq pivot, zprava \geq pivot. (jdeme cílem & swap jenžli oba jsou na špatných stranách).
 - 3) Rekurzivně uděláme to samý se dvěma polovinami.



• Merge sort $O(n \log n)$

- 1) Dělíme pole na 2 části do toho momentu, když pole nebude 1-prvkové. (rekurzivně)
- 2) Rekurzivně jdeme zpět a děláme merge dvou polí (jdeme po dvou polí najednou a dáváme do nového pole prvek z pravého/levého pole)

• Heap sort $O(n \log n)$

- máme heap copii
uděláme heapify
jdeme z kořene
heapify a siftdown
- 1) Na začátku máme přesný heap ($node \leq node.l \wedge node \leq node.r$)
 - 2) Push všechny prvky do heapu
 - 3) Potom pop pro všechny prvky (nejmenší prvek vždy v root)
- heapify!!!

• Radix sort $O(n)$

- 1) Koukáme na poslední cifru a seřazujeme do nového pole
- 2) Koukáme na předposlední cifru a seřazujeme do nového pole

170	45	75	90	802	24	2	66
170	90	802	2	24	45	75	66
802	2	24	45	66	170	75	90
2	24	45	66	75	90	170	802

• Counting sort $O(n)$

- 1) jdeme po poli a vyplňujeme pole četnosti
- 2) jdeme po poli četnosti a vypisujeme element četnosti krát.

• Time

	Nejhorší případ	Nejlepší případ	Běžný případ	Stabilní	Všeobecné
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	Ne	Ano
Insertion	$O(n^2)$	$O(n)$	$O(n^2)$	Ano	Ano
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	Ano	Ano
Quick	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	Ne	Ano
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Ano	Ano
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Ne	Ano
Radix	$O(n)$	$O(n)$	$O(n)$	Ano	Ne
Counting	$O(n)$	$O(n)$	$O(n)$	Ano	Ne

• Stabilita řazení - řazení je stabilní, pokud nemění vzájemné pořadí prvků se stejnou hodnotou.

Dynamic programming

• Tabelace - řešení úlohů použitím předchozích hodnot v tabulce

$$\text{Př: } f(x, y) = \begin{cases} 1 \\ 2 \cdot f(x, y-1) + f(x-1, y) \end{cases}$$

máme spočítat $f(8, 8)$ mnohokrát, ale když budeme zapisovat do tabulky — jen jednou.

• Topological sort $\Theta(N + M)$

1) Jdeme grafem pomocí DFS

2) Když se vrátíme z úzlu, označujeme ho visited, když není označovan a přidaváme ho do pole (do konce, před posledním el)

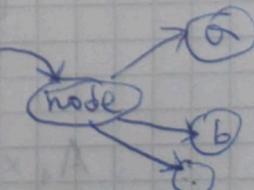
3) Děláme to, pokud nejsou všechny úzly visited.

• Nejdelší cesta v DAG $\Theta(N + M)$

1) Jdeme z konce topologicky uspořádaného pole a kontáme na sousední úzly

$$2) \text{node.a.weight} = \max(\text{node.a.weight}, \text{node.weight} + 1)$$

$$\text{node.b.weight} = \max(\text{node.b.weight}, \text{node.weight} + 1)$$



Nejdélší rostoucí podposloupnost v DAG

$O(n^2)$

1) Udelejme topologické uspořadání

2) Projdeme top. usp. pole a udelejme hrany $x \rightarrow y$, jestli $x < y$.

3) Najdeme nejdélší cestu v DAG

II způsob $O(N \log N)$

↳ V - pole čísel, k - indexy tohoto pole

iL - pole podposloupnosti různé délky, které se končí na nejménší element.

nemusíme
om擦kou

om čerku →

↳ pseudokód

Pro každý index k

Necht d je index max príru a $V[iL[d]] < V[k]$

~~V[iL[d]]~~

$iL[d+1] = k$

$P[k] = iL[d]$, pokud d existuje

Jinak

$iL[1] = k$

$P[k] = \text{null}$

Optimalní násobení matic

- počet operací: $a \cdot b \cdot c$

$$a \begin{smallmatrix} b \\ \hline c \end{smallmatrix} \times b \begin{smallmatrix} c \\ \hline c \end{smallmatrix} = a \begin{smallmatrix} c \\ \hline c \end{smallmatrix}$$

$$A_1 \times (A_2 \times A_3 \times A_4) = B[1,1] \times B[2,N]$$

$$\begin{aligned} (A_1 \times A_2) \times (A_3 \times A_4) &= B[1,2] \times B[3,N] \\ (A_1 \times A_2 \times A_3) \times A_4 &= B[1,3] \times B[4,N] \end{aligned} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \begin{array}{l} N-1 \text{ možnosti} \\ (\text{asociativní}) \end{array}$$

$B[i,j]$ - výsledek vynásobení odpovídajícího úseku

$r(B[i,j]) = r(A_i) - \text{řádky}$

$s(B[i,j]) = s(A_j) - \text{sloupce}$

Př

$$\begin{array}{c} A_1 \\ \hline 30 \times 35 \end{array} \times \begin{array}{c} A_2 \\ \hline 35 \times 15 \end{array} \times \begin{array}{c} A_3 \\ \hline 15 \times 5 \end{array}$$

$$A_1 \times A_2 \times A_3 \quad \begin{array}{l} (A_1 \times A_2) \times A_3 = 30 \times 35 \times 15 + 30 \times 15 \times 5 = 18000 \\ A_1 \times (A_2 \times A_3) = 30 \times 35 \times 5 + 35 \times 15 \times 5 = 7875 \end{array} \checkmark$$

Optimal Binary Search Tree

↪ Optimal BST — BST, ve kterém každý uzel má pravděpodobnost (hledání) a celkovou cenu.

$$\text{cena} = \sum (\text{node.h} \cdot \text{node.p}), \text{ pro všech node}$$

↪ Tvorba tabulky pro určení nejlepšího kořenu

C_{ij} — average number of key comparisons in a OBST with keys $k_i \dots k_j$

$$C_{ij} = \min_{i \leq l \leq j} (C_{i,l-1} + C_{l+1,j}) + \underbrace{\sum_{s=i}^j p(k_s)}_{\sum C_{ss}, s=1 \dots j}$$

$\text{roots}[i][j] = l$, které minimalizuje

↪ Tvorba OBST

Po tvorbě tabulky rekursivně ~~načázíme~~ tvoríme strom pomocí tabulky.

Nejdelsí společná podposloupnost

		B:							
		0	1	2	3	4	5	6	7
		D	E	C	D	B	A		
		0	0	0	0	0	0	0	0
A:		1	C	0	0	0	1	1	1
		2	B	0	0	0	1	1	2
		3	E	0	0	1	1	1	2
		4	A	0	0	1	1	1	2
		5	D	0	1	1	1	2	3
		6	D	0	1	1	1	2	3
		7	E	0	1	2	2	2	3
		8	A	0	1	2	2	2	3

← nebo ↑ odpočet

1) Jde o obyčejné polí

2) if ($A[i:j] = B[i:j]$):

$$C[i:j] = C[i-1:j-1] + 1$$

else:

bereme hodnotu
max(← nebo ↑)

$$\max(C[i:j-1], C[i-1:j])$$

• Knapsack 0/1 (každý předmět můžeme použít jenou)

val	wt	0	1	2	3	4	5	6	7	← počet kg, které můžeme nesít (kapacita)
(1)	1	0	1	1	1	1	1	1	1	
(4)	3	0	1	1	4	5	5	5	5	$K[i][j] = \max(\text{del}[i][j], K[i-1][j])$
(5)	4	0	1	1	4	5	6	6	8	$w[i] + K[i-1][j-w[i]]$
(7)	5	0	1	1	4	5	7	8	9	$K[i-1][j]$

↑
dostupné předměty

• Knapsack problem

DAG:

uzly: Kapacity 0, 1, 2... C

(cenamy)

hrany: Z X vedou ohodnocení hrany $X+w[i]$, $X+w[i-1]$

Pořadí:

Hledáme cestu s maximální cenou začínající od prvního uzlu. Max cenu zapisujeme do uzlu.

Hash

• Hashovací funkce - funkce, která přiřadí číslu/řetězci index v tabulce.

• Kolize - když hashovací funkce má stejně výsledky pro různé klíče, $k_1 \neq k_2$, $h(k_1) = h(k_2)$

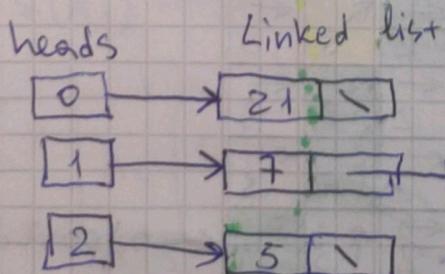
• Řešení kolizi

a) Linked hash - každý index je linked list

Př

$$h(k) = k \% 3$$

dáno: 1, 5, 21, 10, 7



Insert - $O(1)$

Search - $O(n)$, ale první $O(1+1)$

Delete - $O(n)$, ale první $O(1+1)$

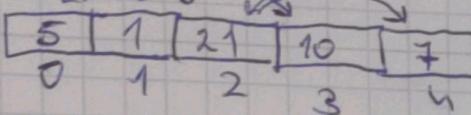
!!! Vkládá se na začátek

+ Nemusíme znát n předem
- dynamická paměť
- paměť na ukazatele

b) Open hash

↳ Linear probing - když pozice není prázdná, máme vložit prvek o pozici dal

Pr. $1, 5, 21, 10, 7$ $h(k) = k \% 5$



↳ Double hashing

- $\text{index} = \text{hash}_1(k) + j \cdot \text{hash}_2(k) \quad j \in [0; +\infty)$ (N^o)
- Když pozice není prázdná, spočteme $\text{hash}_2(k)$ a budeme sčítat dokud nebudeme mít prázdné místo
- Pr.

$$\text{hash}_1(k) = k \% 13 ; \quad \text{hash}_2(k) = 7 - k \% 7$$

keys $(18, 41, 22, 44)$

k	hash1	hash2
18	5	3
41	2	1
22	9	6
44	5	5

	41		18					22	44		
0	1	2	3	4	5	6	7	8	9	10	11