Slovník & hashování
Dynamické programování

ACM: pokročilá algoritmizace a programovací techniky

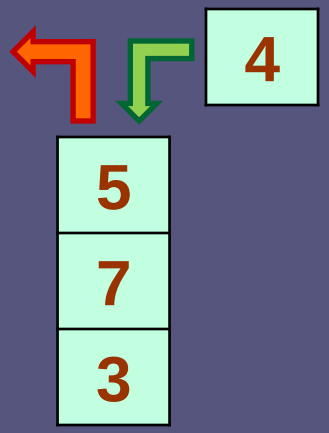
Martin Kačer

České vysoké učení technické v Praze

Slovník („mapa“) & hashování

Opakování:
Datová struktura Zásobník

- Init
- Push(x)
- Pop → x
- Is Empty



Opakování:
Datová struktura Fronta

- Init
- Enqueue(x)
- Dequeue → x
- Is Empty

Datová struktura Slovník

(= tabulka, mapa, asociativní pole)

- Init
- Insert(K, V)
- Find(K) → V
- Delete(K)

K	V
1212	Zlatá bula
1948	puč KSČ
935	sv. Václav
1918	Československo
1989	revoluce
1968	okupace

Datová struktura Slovník

- Jak implementovat?
- Pole
  - K = index
  - → přímočaré
  - Find(K)
    - O(1)

K	V
2	úterý
4	čtvrtek
6	sobota
7	neděle

Slovník

Jak implementovat?

- Pole K[], V[]
- Find(K)
  - for cyklus
  - O(n)

K
1212
1948
935
1918
1989
1968

V
Zlatá bula
puč KSČ
sv. Václav
Československo
revoluce
okupace

Slovník – inspirace

- A
- B
- C
- D
- E
- F
- G
- …

Slovník – seřazený

Seřazený seznam

- Find(K)
  - binární půlení
  - O(log n)
- Insert(K,V)
  - nutno posunout
  - O(n)

K	V
935	sv. Václav
1212	Zlatá bula
1918	Československo
1948	puč KSČ
1968	okupace
1989	revoluce

Slovník

- A
- B
- C
- D
- E
- F
- G
- …

Slovník – hashování

(= „rozptylování“, hešování)

Přihrádky

- dle funkce h(K)
  - např. **K % 10**
- Find(K)
  - průměrně O(n/m)

h(K)	K
0	
1	
2	1212
3	
4	
5	935
6	
7	
8	1948 1918 1968
9	1989

Hashovací funkce

- Ideální = „prakticky náhodná“
- Rozděluje („rozptyluje“) rovnoměrně
- → průměrná složitost **O(1)**

Neznámá velikost slovníku?

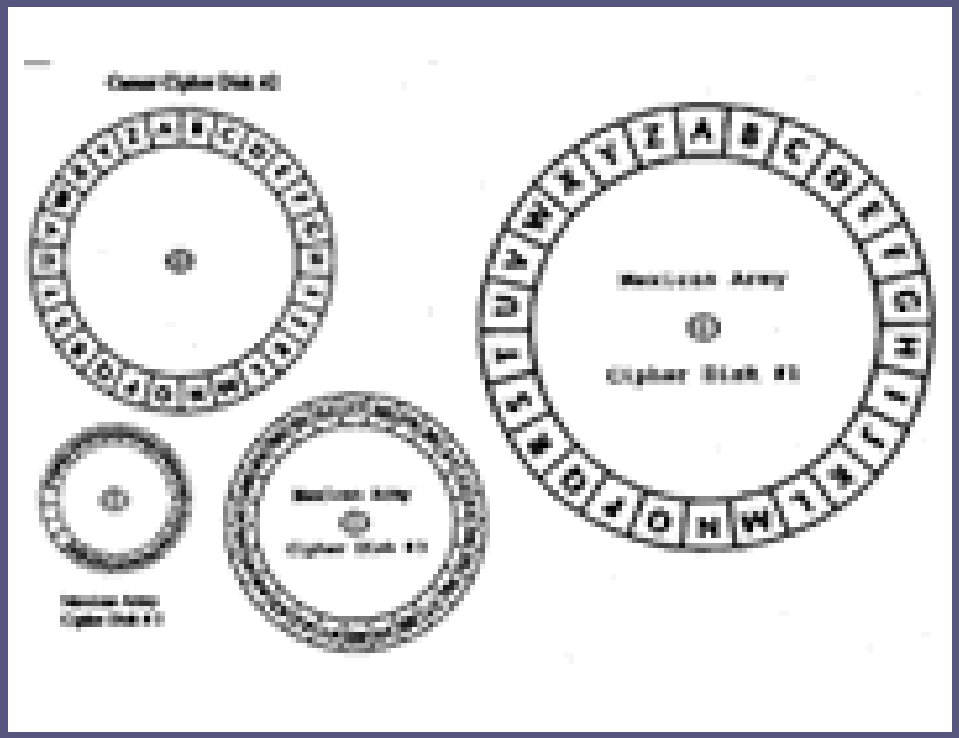
- Postupné zvětšování na dvojnásobek

Slovník – další možnosti

- Vyhledávací stromy

- … udržuje se seřazený
- O(log n)

PŘÍKLAD



Vigenèrova šifra

A+A → A

A+B → B

A+C → C

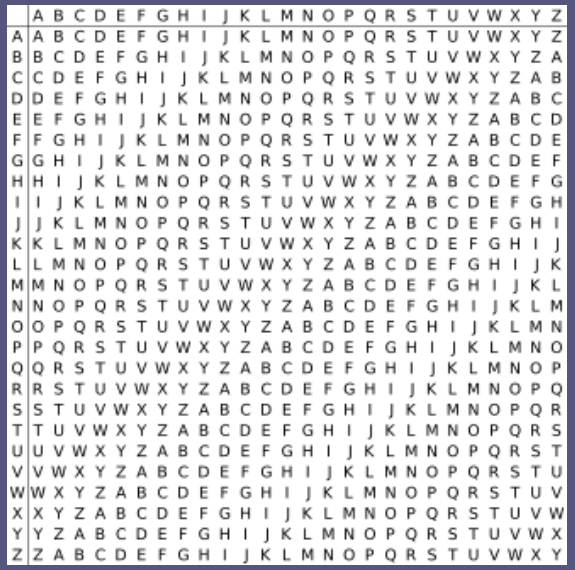
…

B+B → C

B+C → D

B+D → E

Heslo = **K O D**



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

T	A	J	N	A	Z	P	R	A	V	A
K	O	D	K	O	D	K	O	D	K	O
D	O	M	X	O	C	Z	F	D	F	O

Příklad: Zadání

- Vstup
  - Zašifrovaná zpráva
  - Maximální délka klíče (≤100)
  - 2 slova *(cribs)* z původní zprávy (≥ klíč)
- Výstup
  - Rozšifrujte původní zprávu!

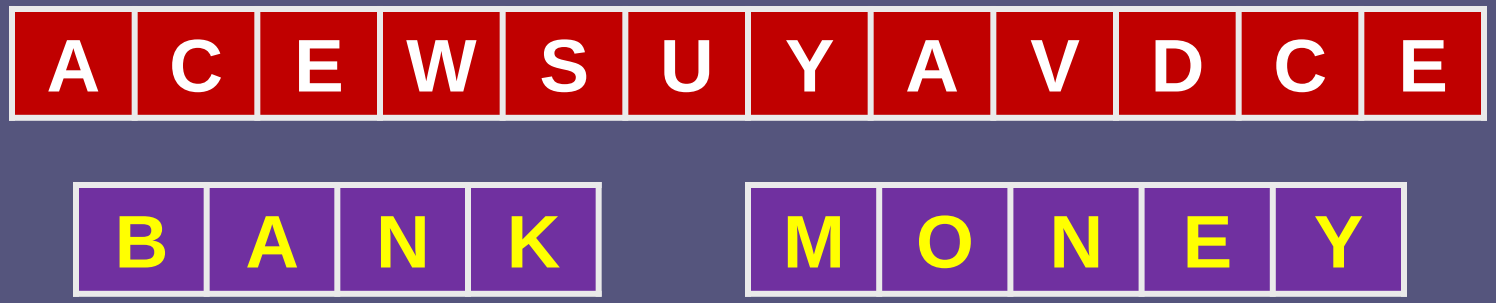
D	O	M	X	O	C	Z	F	D	F	O

T	A	J

P	R	A	V

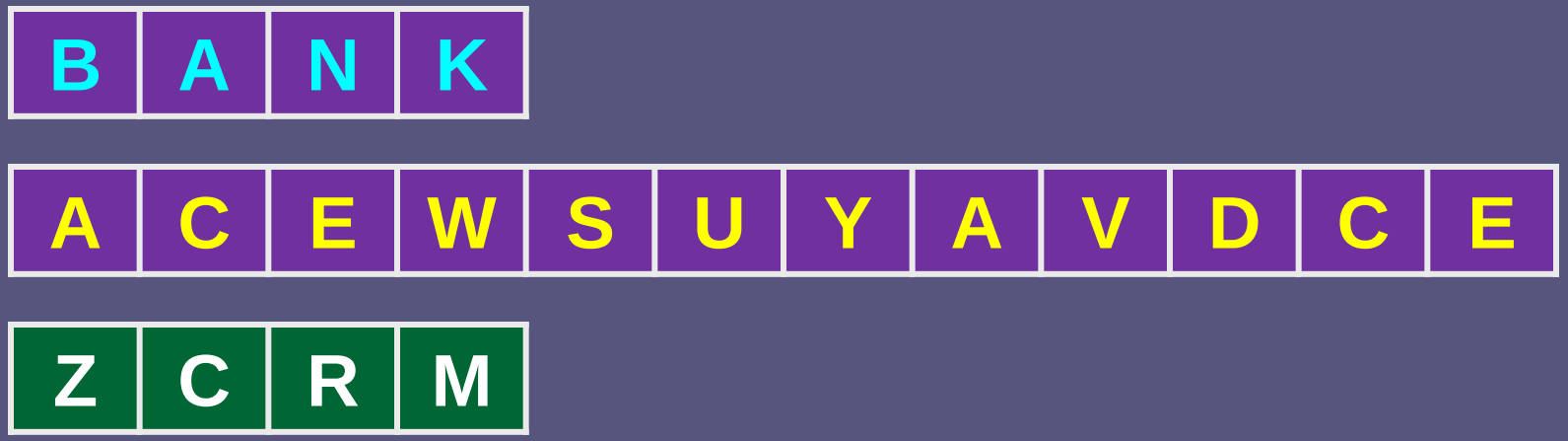
Jak na řešení?

- Zkoušet všechny klíče nejde
  - exponenciální složitost
  - $O(26^n)$



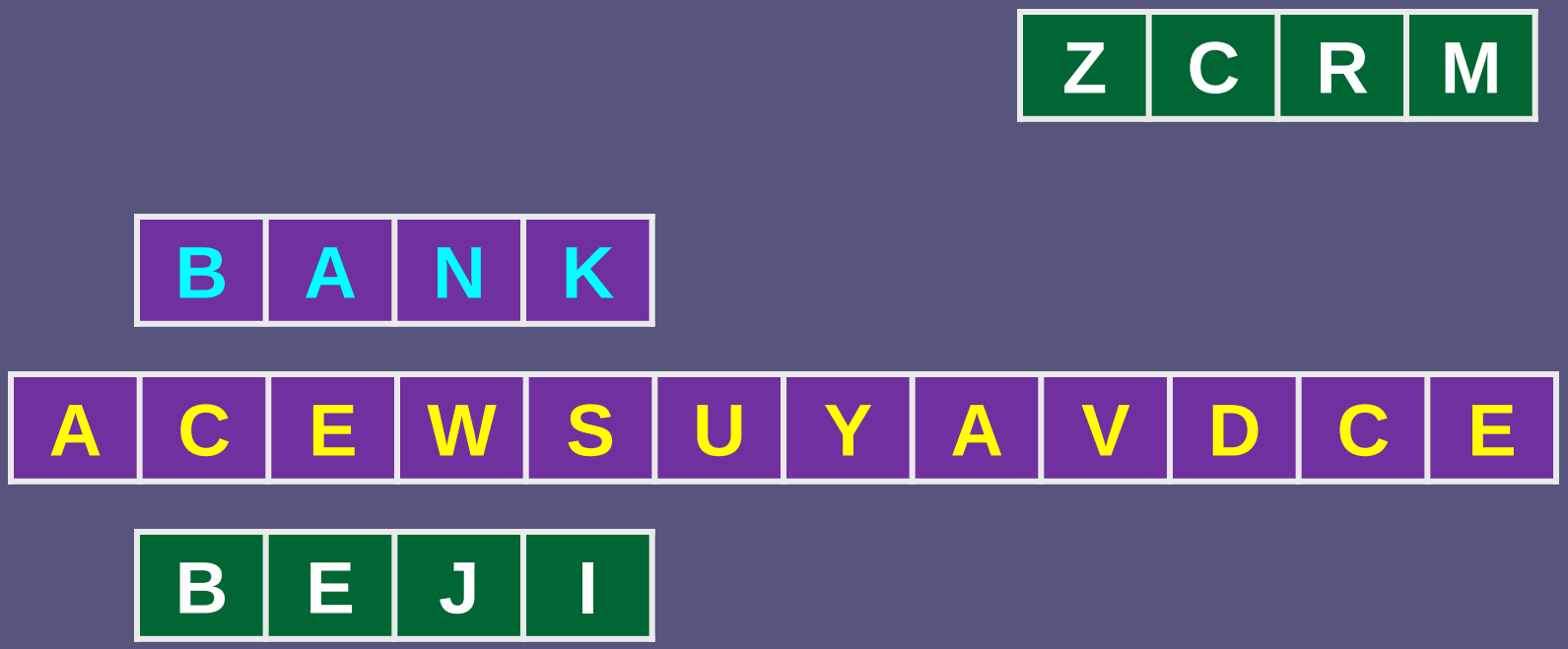
Způsob řešení

- Zkusíme „cribs“ ve všech pozicích



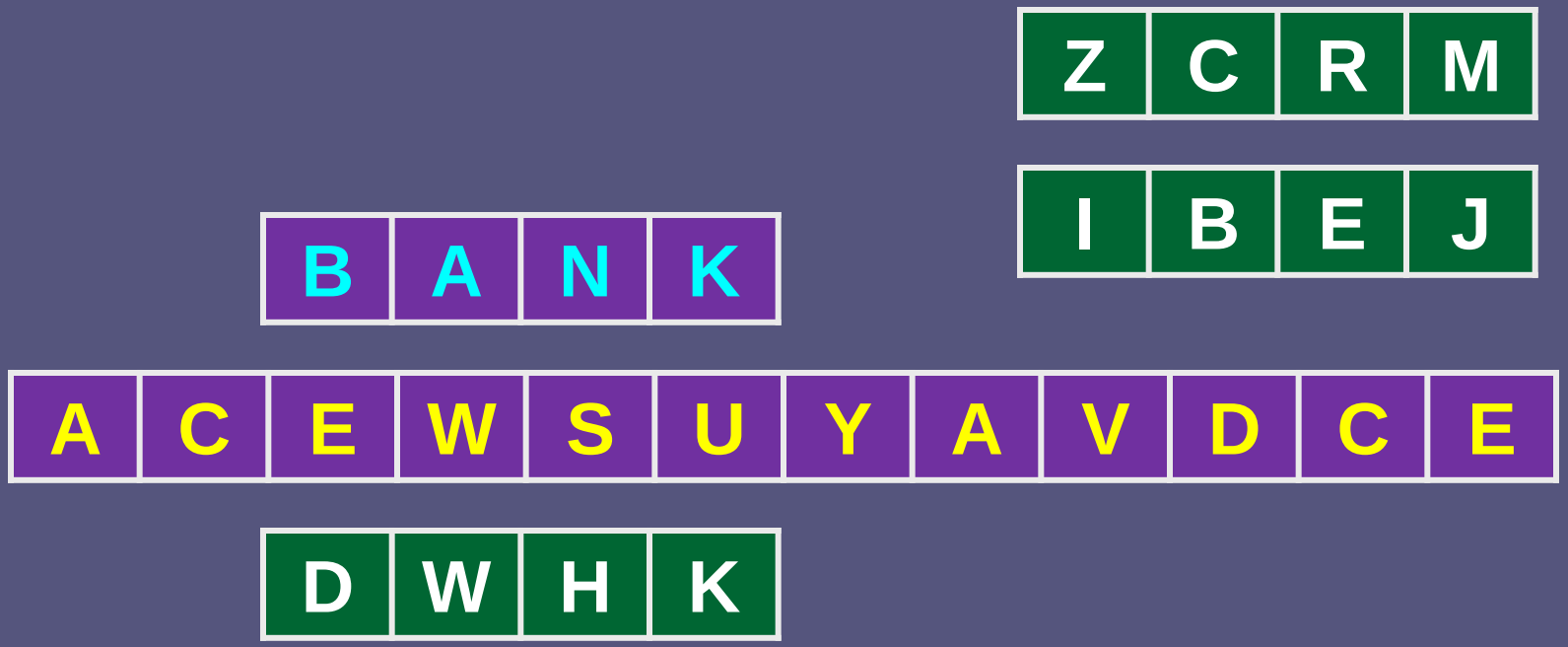
Způsob řešení

- Zkusíme „cribs“ ve všech pozicích



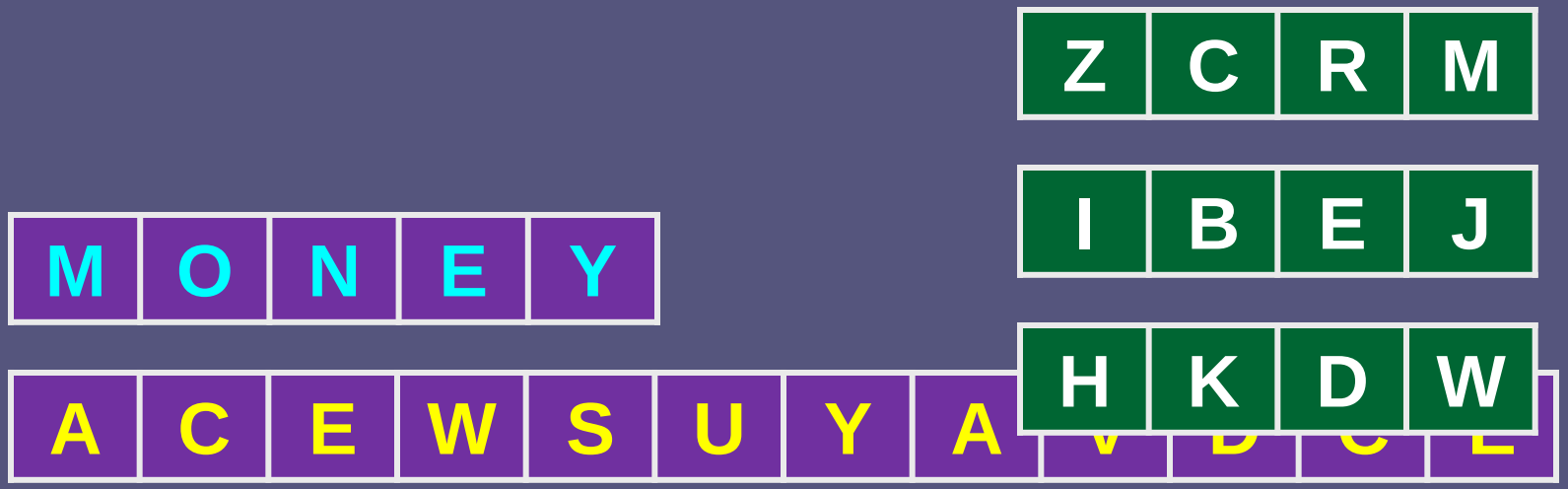
Způsob řešení

- Zkusíme „cribs“ ve všech pozicích



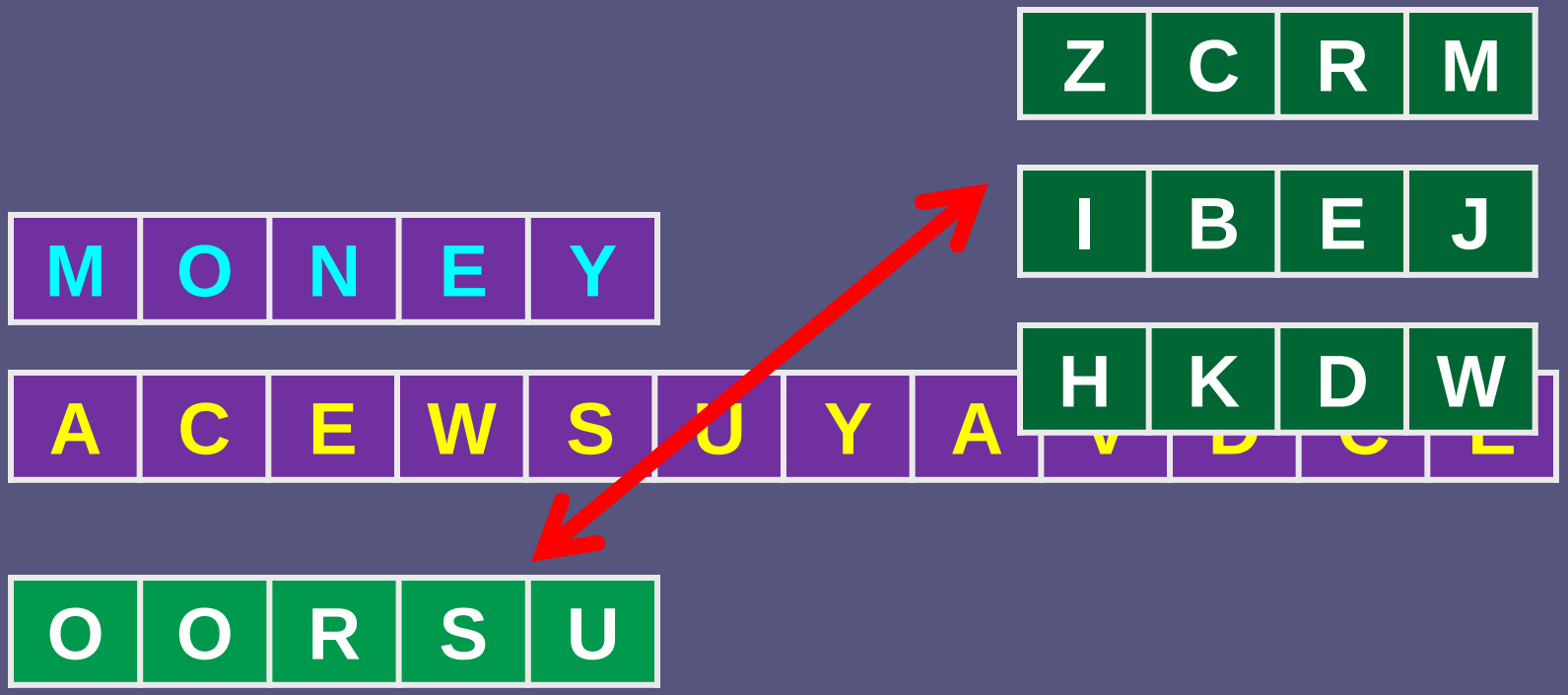
Způsob řešení

- Pro druhý „crib“ hledáme stejný klíč



Způsob řešení

- Pro druhý „crib“ hledáme stejný klíč



Operační složitost

- Všechna umístění pro první „crib“
  - O(n.k)
- Všechna umístění pro druhý „crib“
  - Kontrola pomocí hashovací tabulky
  - **O(n.k . H)**

Dokončení řešení

Ve skutečnosti je to trochu složitější

- Pozor na délku klíče a jeho opakování
  - ABCAB → možné klíče jsou i ABC, ABCA
- Překryv slov
  - V zadání požadavek na 2 samostatná slova (tj. nesměla se překrývat)

Dynamické programování

Dynamické programování

- Řešení problému redukcí na menší
- Omezení opakování výpočtu (rozdíl oproti rekurzi)

- Často zásadní vliv na složitost!
  - lineární/kvadratický vs. exponenciální
  - tj. použitelný vs. nepoužitelný

Příklad – Fibonacciho čísla

- 1, 1, 2, 3, 5, 8, 13, 21, 34, …
- F(1) = F(2) = 1
- pro N>2: F(N) = F(N-2) + F(N-1)

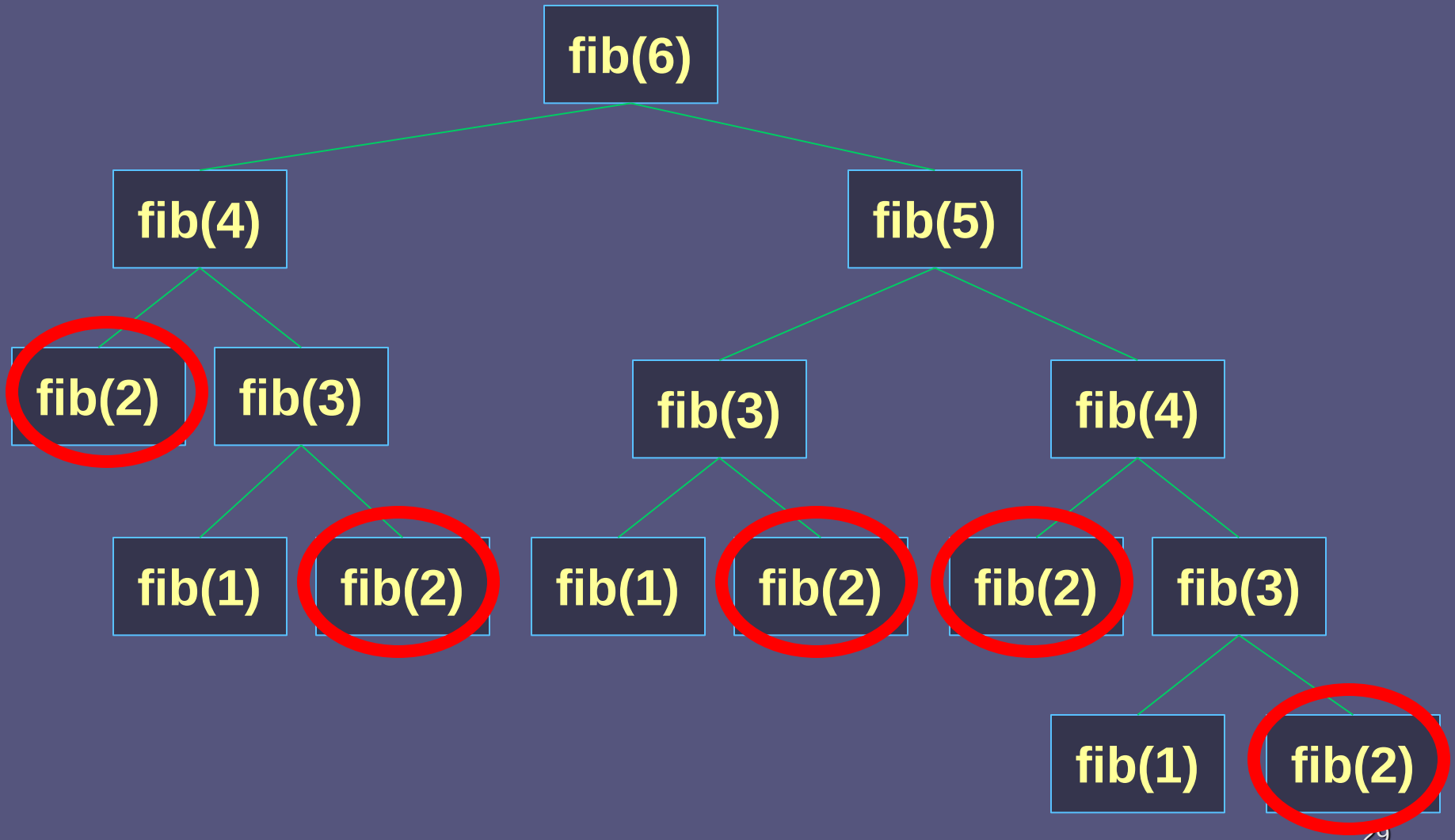
- Rekurzivní řešení se nabízí

Fibonacciho čísla – rekurze

- F(1) = F(2) = 1
- pro N>2: F(N) = F(N-2) + F(N-1)

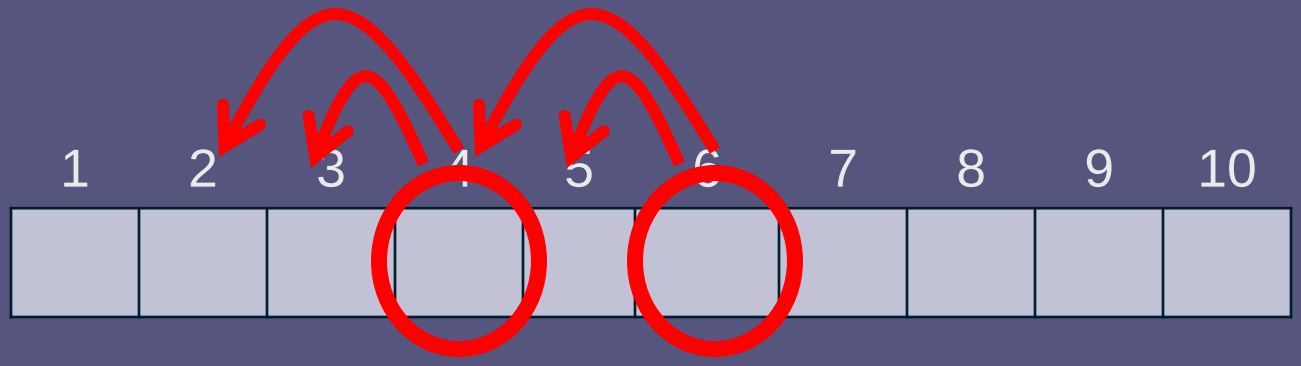
```
int fib(int n) {
    if (n <= 2) return 1;
    return fib(n-2) + fib(n-1);
}
```

Strom rekurzivního volání



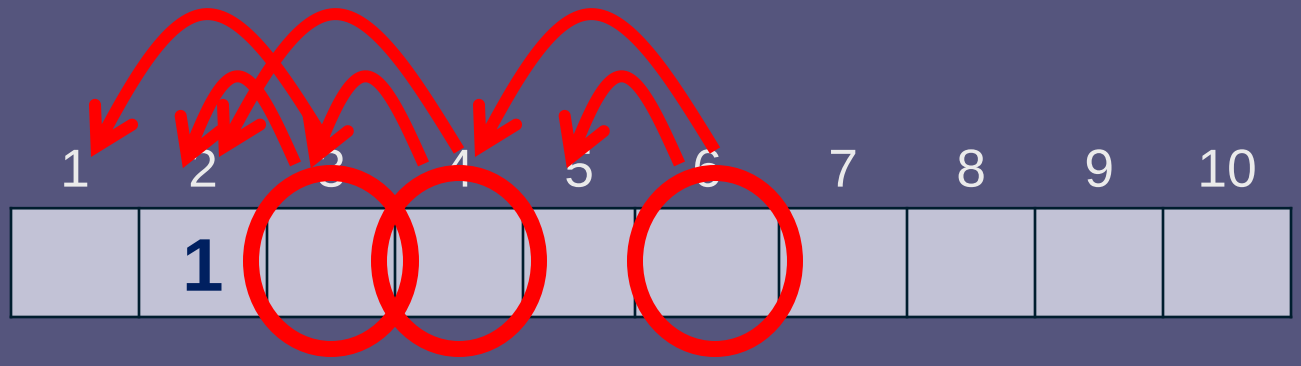
Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



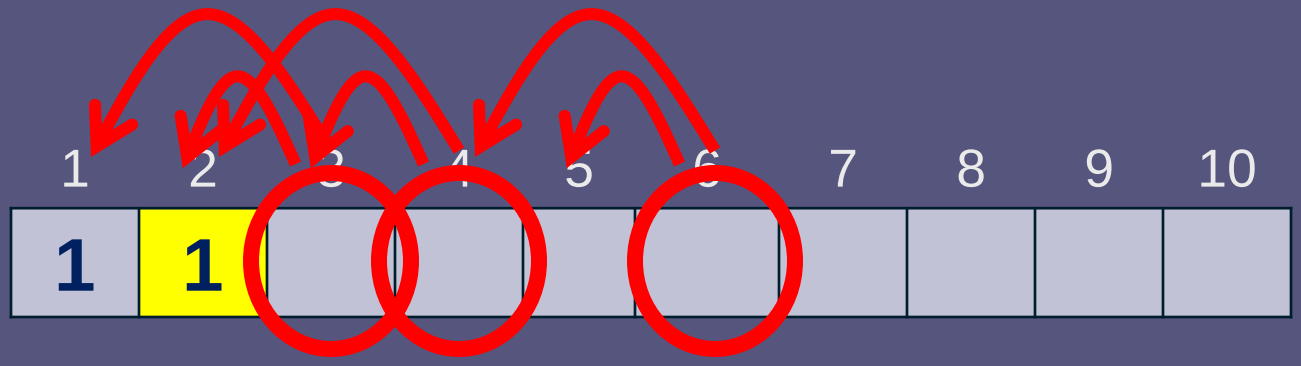
Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



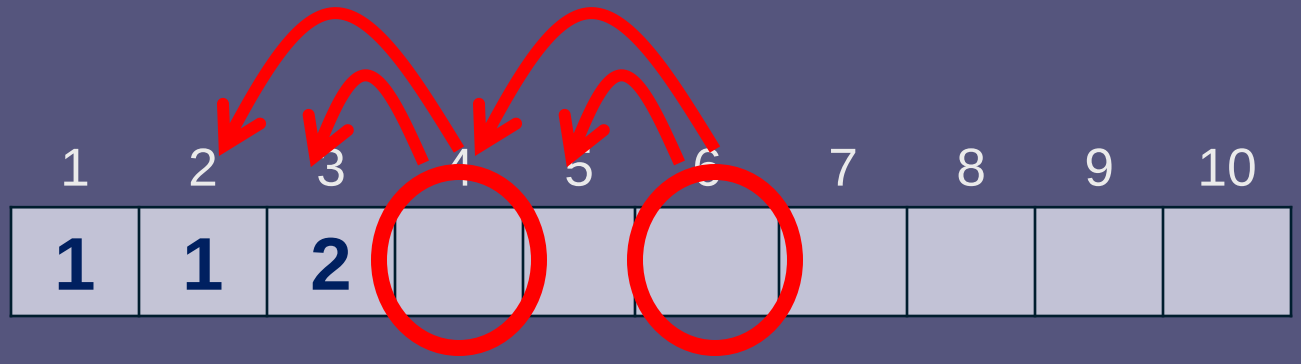
Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



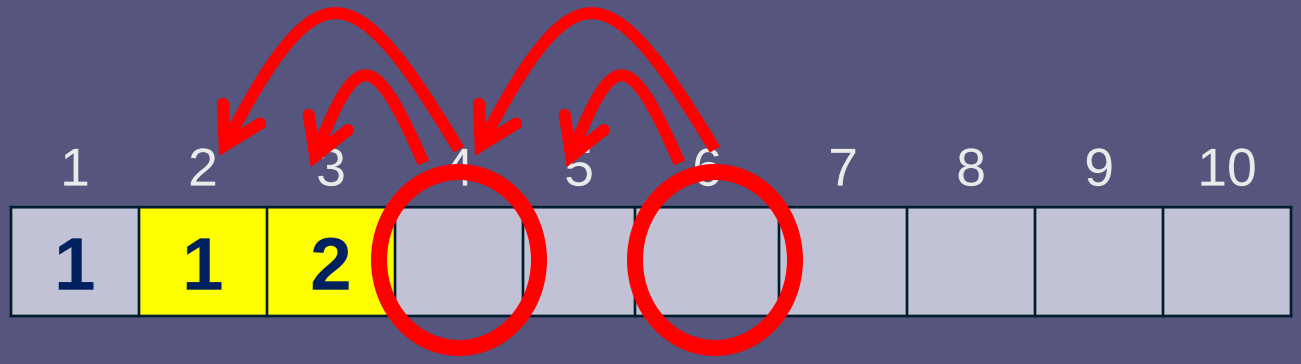
Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



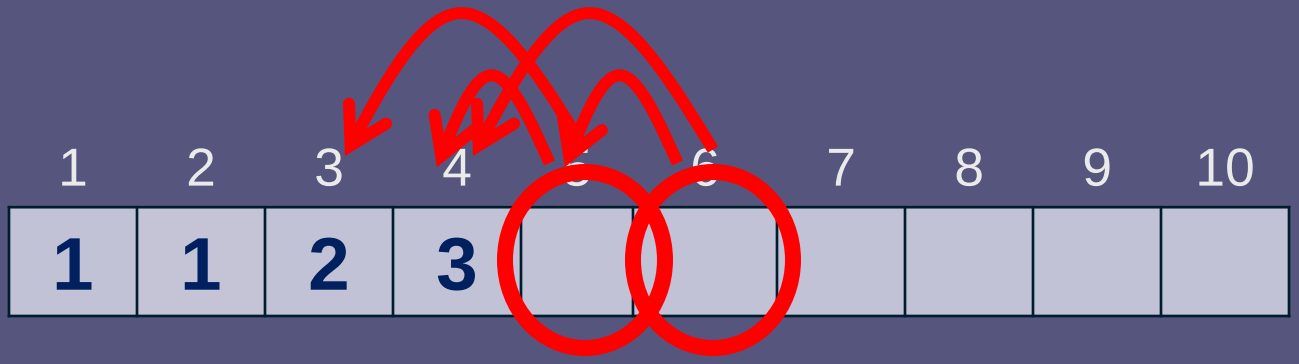
Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



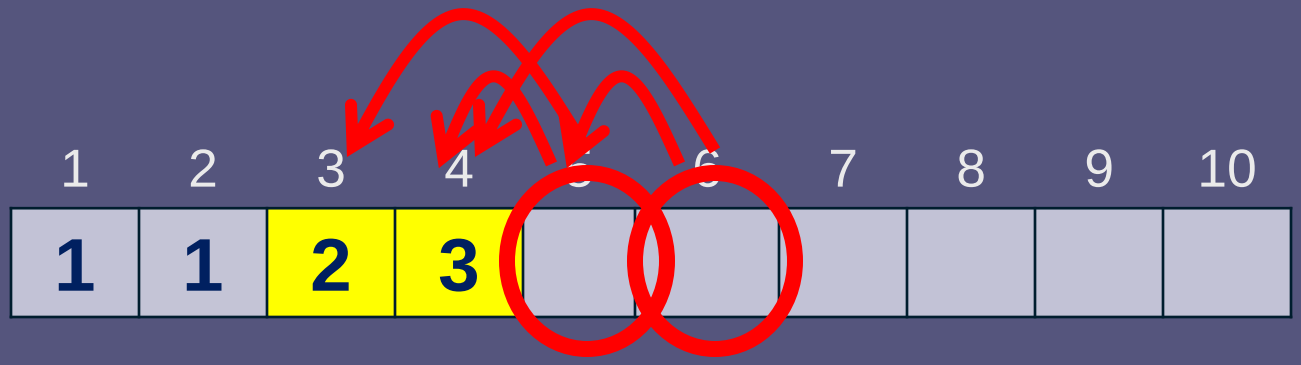
Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



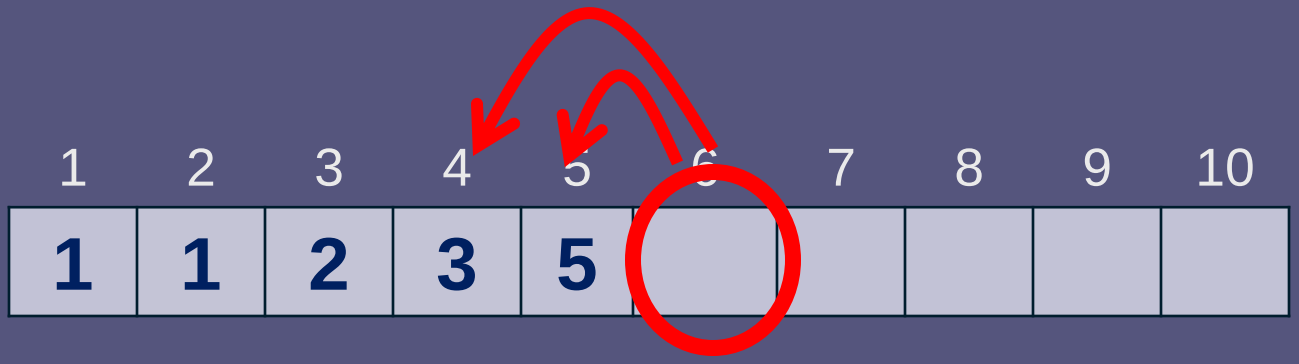
Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



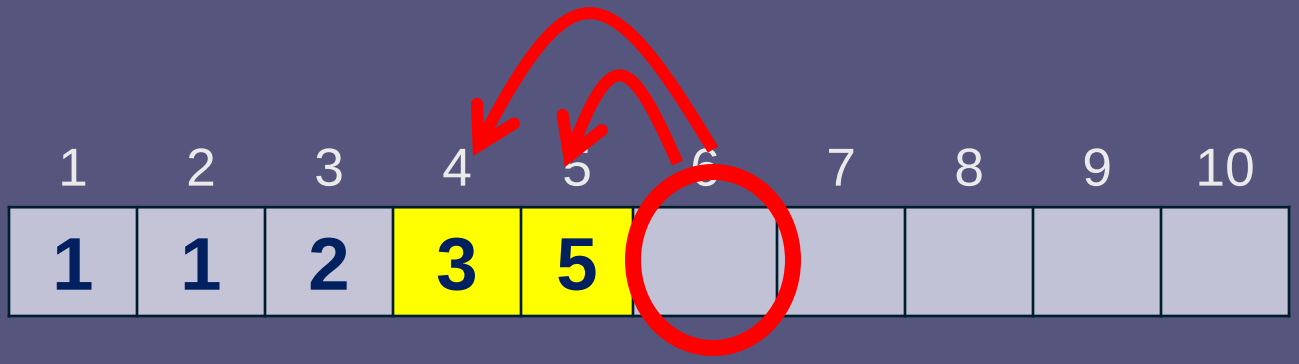
Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



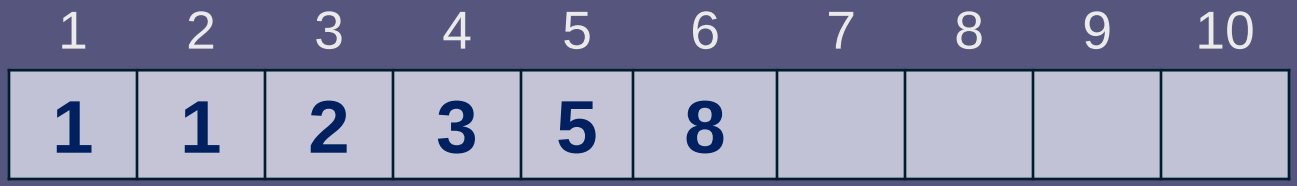
Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



Jak zamezit opakování?

- Pamatovat si, co už bylo spočítáno



Fibonacciho čísla – kód I

```
int fib(int n) {
    if (result[n] > 0)
        return result[n];
    if (n <= 2)
        return result[n] = 1;
    return result[n]
        = fib(n-2) + fib(n-1);
}
```

Přístup „shora“

- Kód velmi podobný rekurzi
- Pamatujeme si již spočítané
  - Nulování pole
  - Test, zda existuje hodnota
- Často nejjednodušší na představu
- Může být neefektivní

Přístup „zdola“

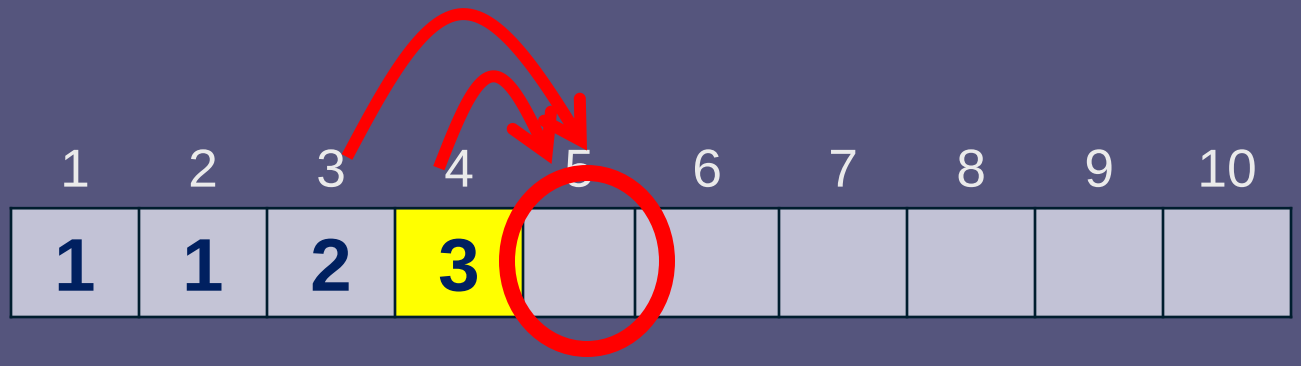
- Z menších instancí generujeme větší

Přístup „zdola“

- Z menších instanci generujeme větší

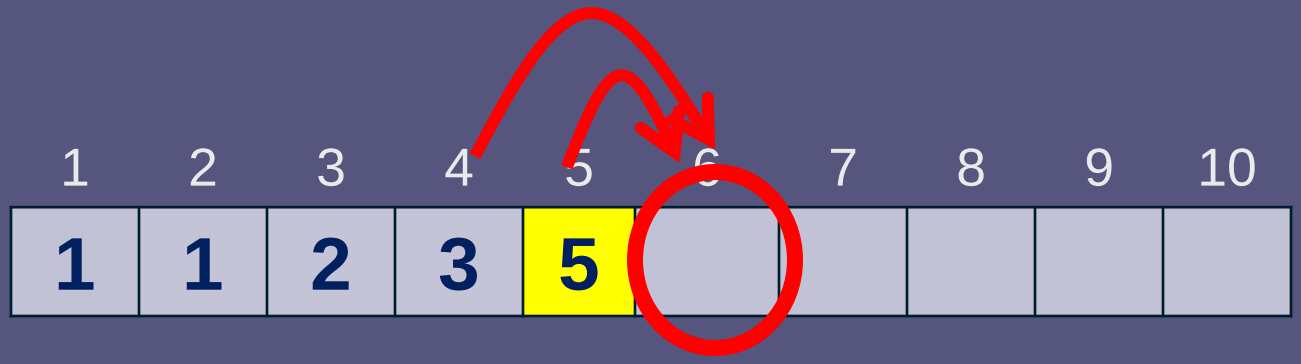
Přístup „zdola“

- Z menších instanci generujeme větší



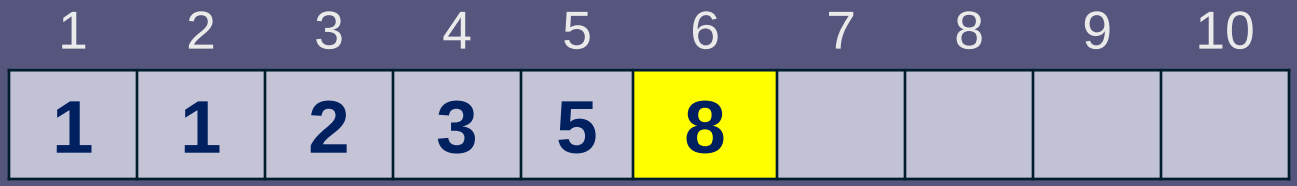
Přístup „zdola“

- Z menších instanci generujeme větší



Přístup „zdola“

- Z menších instanci generujeme větší

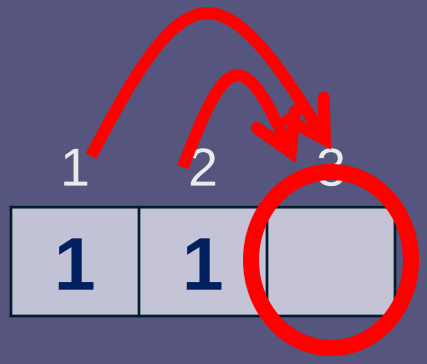


Fibonacciho čísla – kód II

```
int fib(int n) {
    result[1] = result[2] = 1;
    for (int i = 3; i <= n; ++i)
        result[i]
            = result[i-1]
            + result[i-2];
    return result[n];
}
```

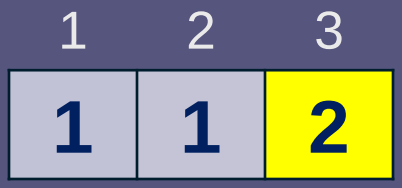
Snížení spotřeby paměti

- Postupně zapomínáme, co není potřeba



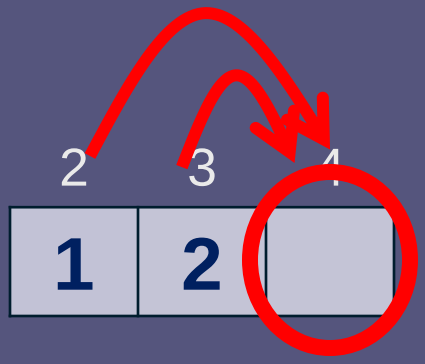
Snížení spotřeby paměti

- Postupně zapomínáme, co není potřeba



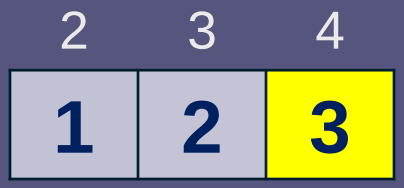
Snížení spotřeby paměti

- Postupně zapomínáme, co není potřeba



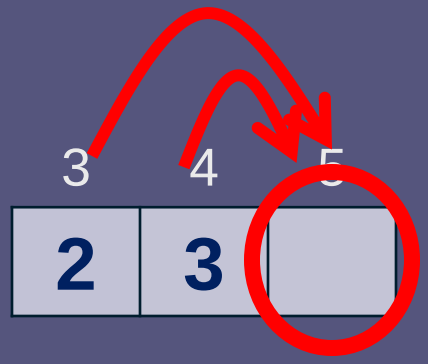
Snížení spotřeby paměti

- Postupně zapomínáme, co není potřeba



Snížení spotřeby paměti

- Postupně zapomínáme, co není potřeba



Snížení spotřeby paměti

- Postupně zapomínáme, co není potřeba

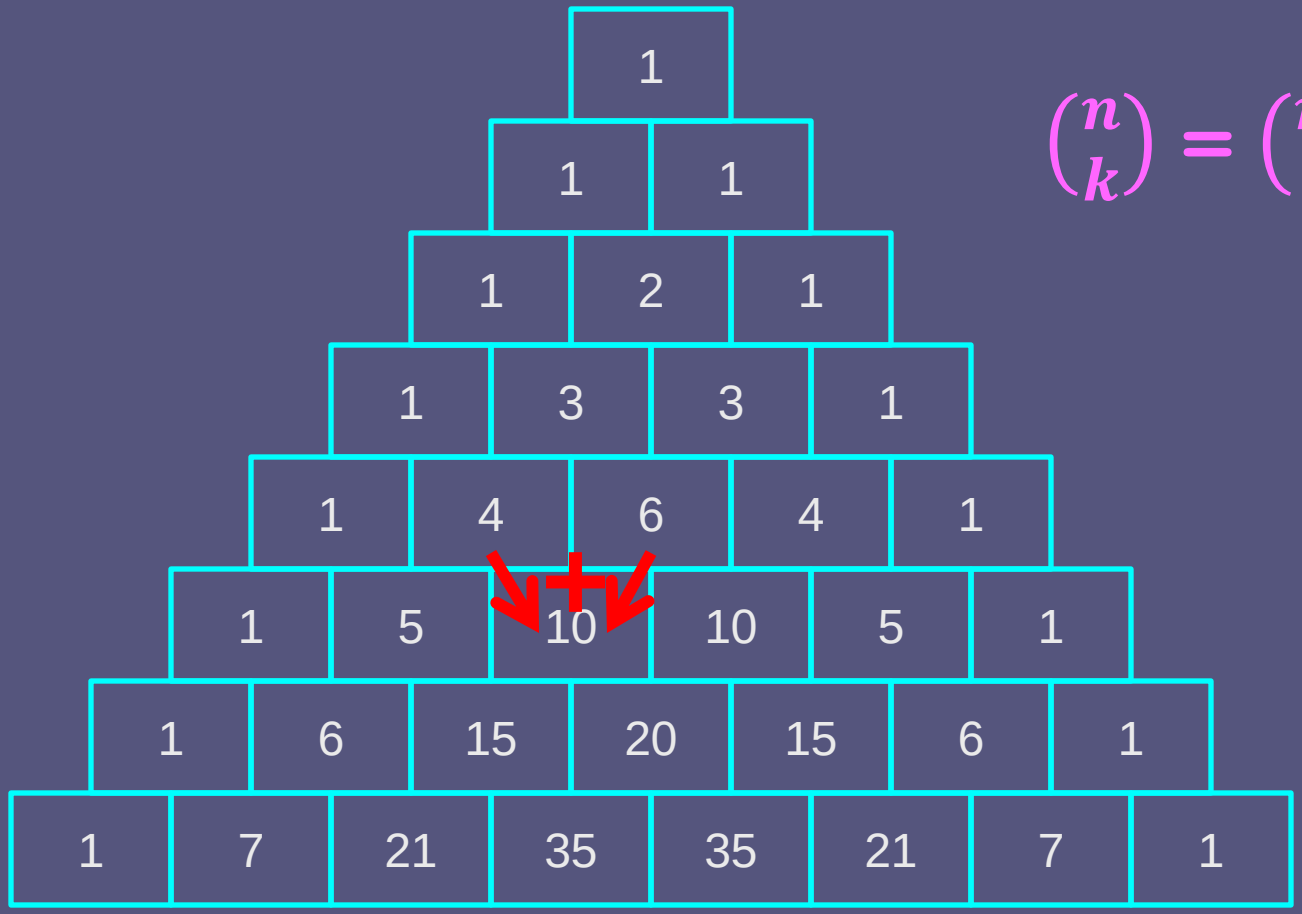
Fibonacciho čísla – kód III

```
int fib(int n) {
    int res = 1, prev = 1;
    for (int i = 3; i <= n; ++i) {
        int nxt = res + prev;
        prev = res;
        res = nxt;
    }
    return res;
}
```

Obecné schéma D.P.

- Vždy přítomno:
  - Řešení problému pomocí menší instance
  - Pamatování výsledků
    - Lze použít slovník (asociativní tabulku / Map)
- Obvykle jde:
  - Postup „zdola“, tj. od nejmenších instancí
  - Odstranění rekurze
- Často navíc:
  - „Zapomínání“ => menší spotřeba paměti

Pascalův trojúhelník



$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$\sum = 2^n$$

Společná podposloupnost

- Najděte nejdelší společnou podposloupnost (i nesouvislou) řetězců

a b r a k a d a b r a

b a r u n k a

Společná podposloupnost

- Najděte nejdelší společnou podposloupnost (i nesouvislou) řetězců

a b r a k a d a b r a

b a r u n k a

Společná podposloupnost

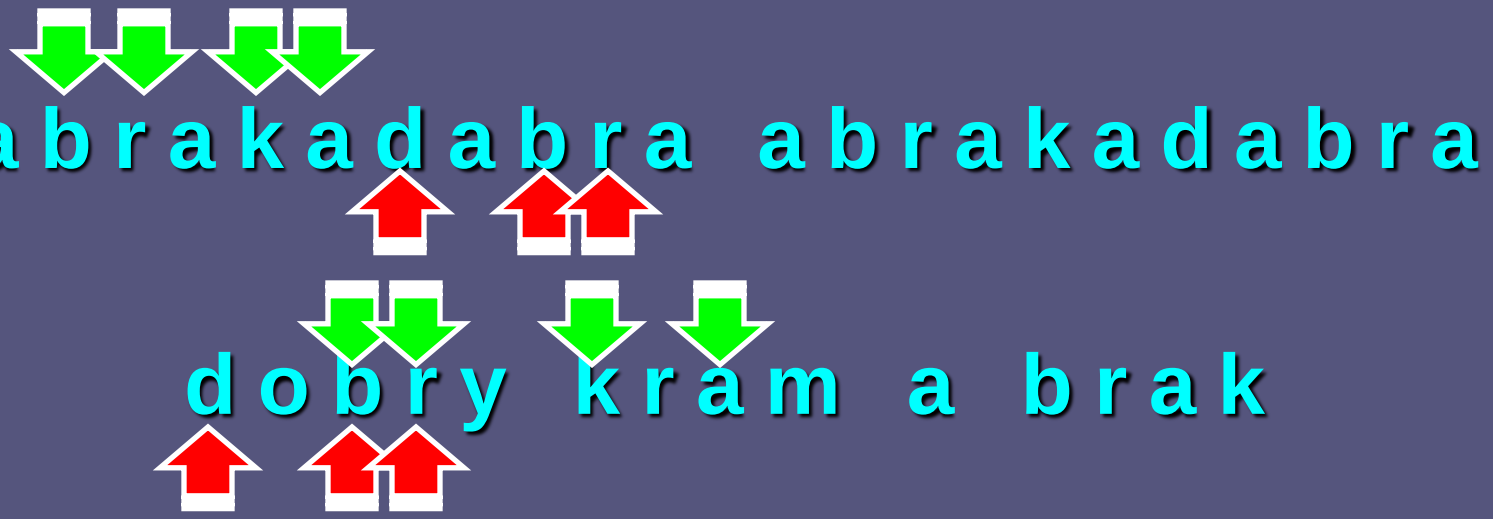
- Ono to ale vůbec nemusí být jednoduché
- Například:

a b r a k a d a b r a a b r a k a d a b r a

d o b r y k r a m a b r a k

Společná podposloupnost

- Ono to ale vůbec nemusí být jednoduché
- Například:



Rekurzivní řešení

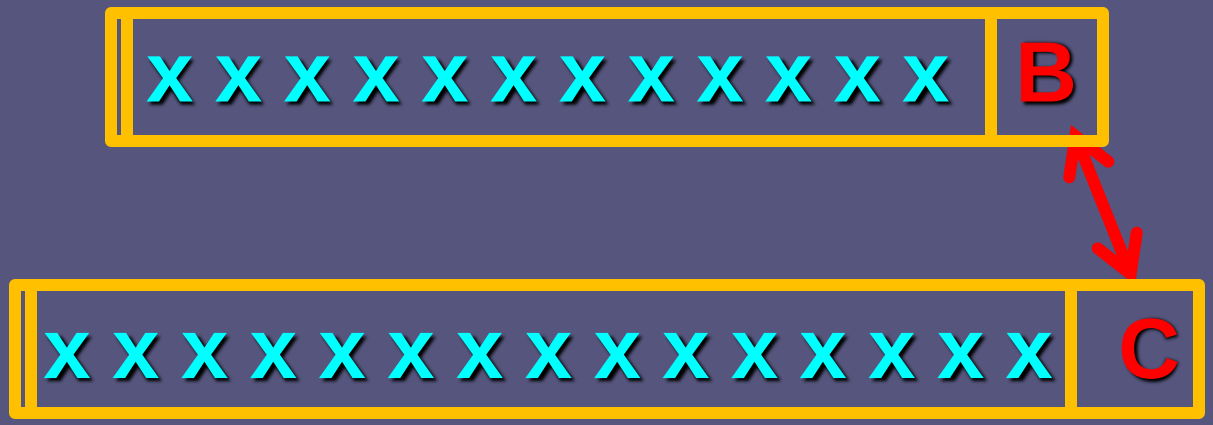
- Pomohlo by, kdybychom uměli řešení pro jakékoli kratší řetězce?

Rekurzivní řešení

- Pomohlo by, kdybychom uměli řešení pro jakékoli kratší řetězce?

Rekurzivní řešení

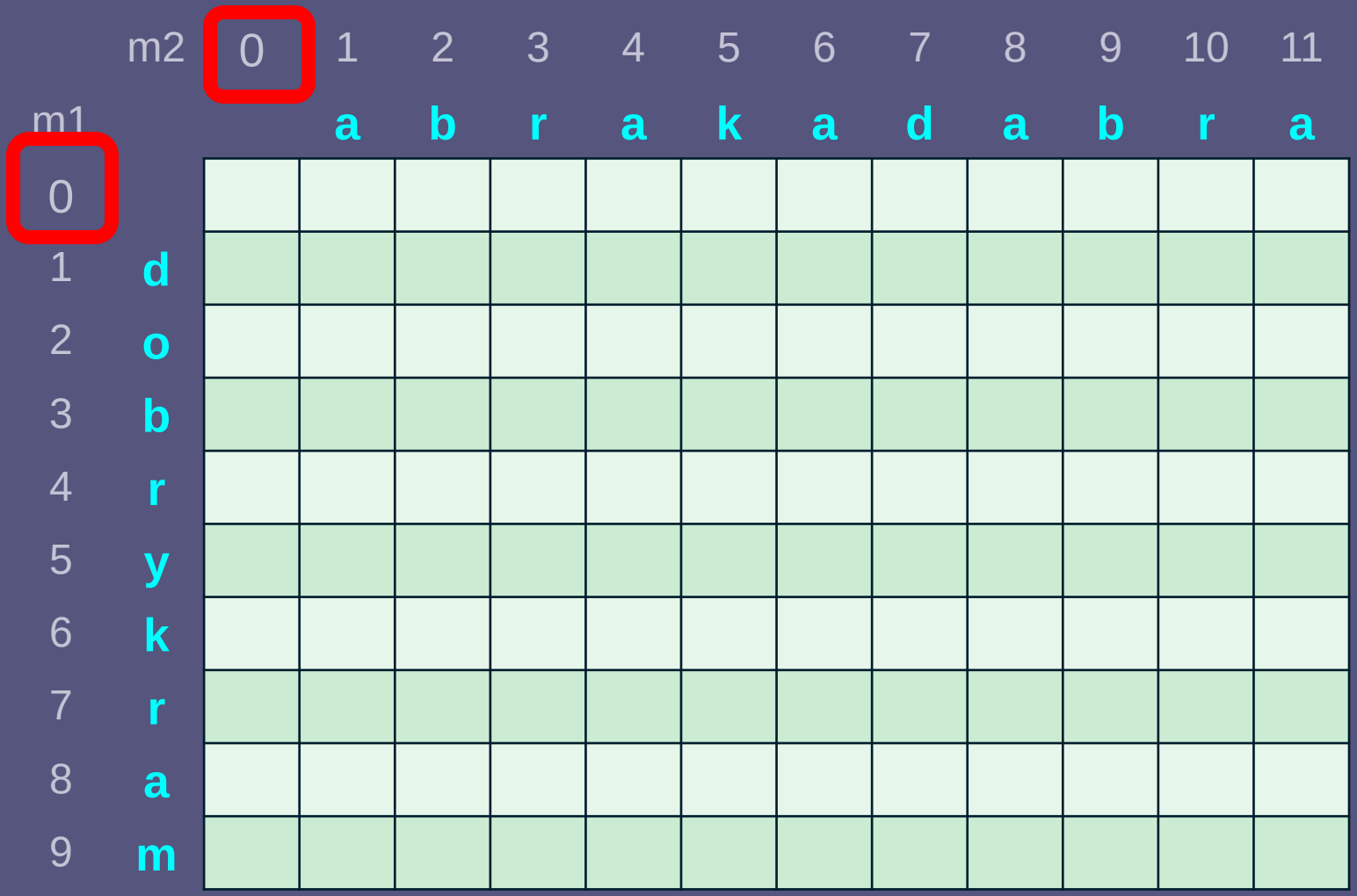
- Pomohlo by, kdybychom uměli řešení pro jakékoli kratší řetězce?



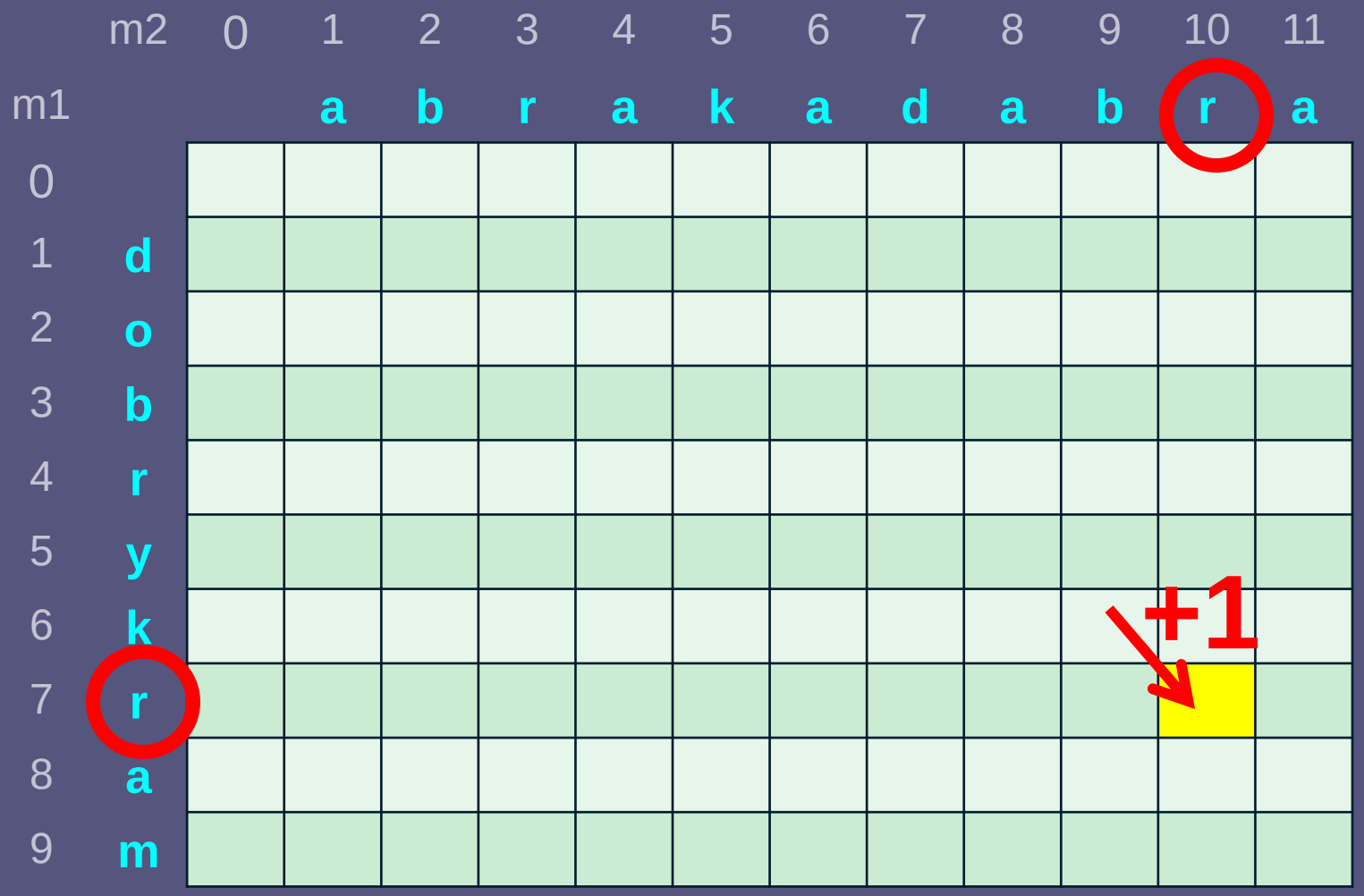
Podposloupnost rekurzivně

```
char[] s1, s2;
int commonLen (int m1, int m2) {
      if (m1 == 0 || m2 == 0)
            return 0;
      if (s1[m1-1] == s2[m2-1])
            return 1 + commonLen(m1-1, m2-1);
      return Math.max(
            commonLen(m1, m2-1),
            commonLen(m1-1, m2));
}
```

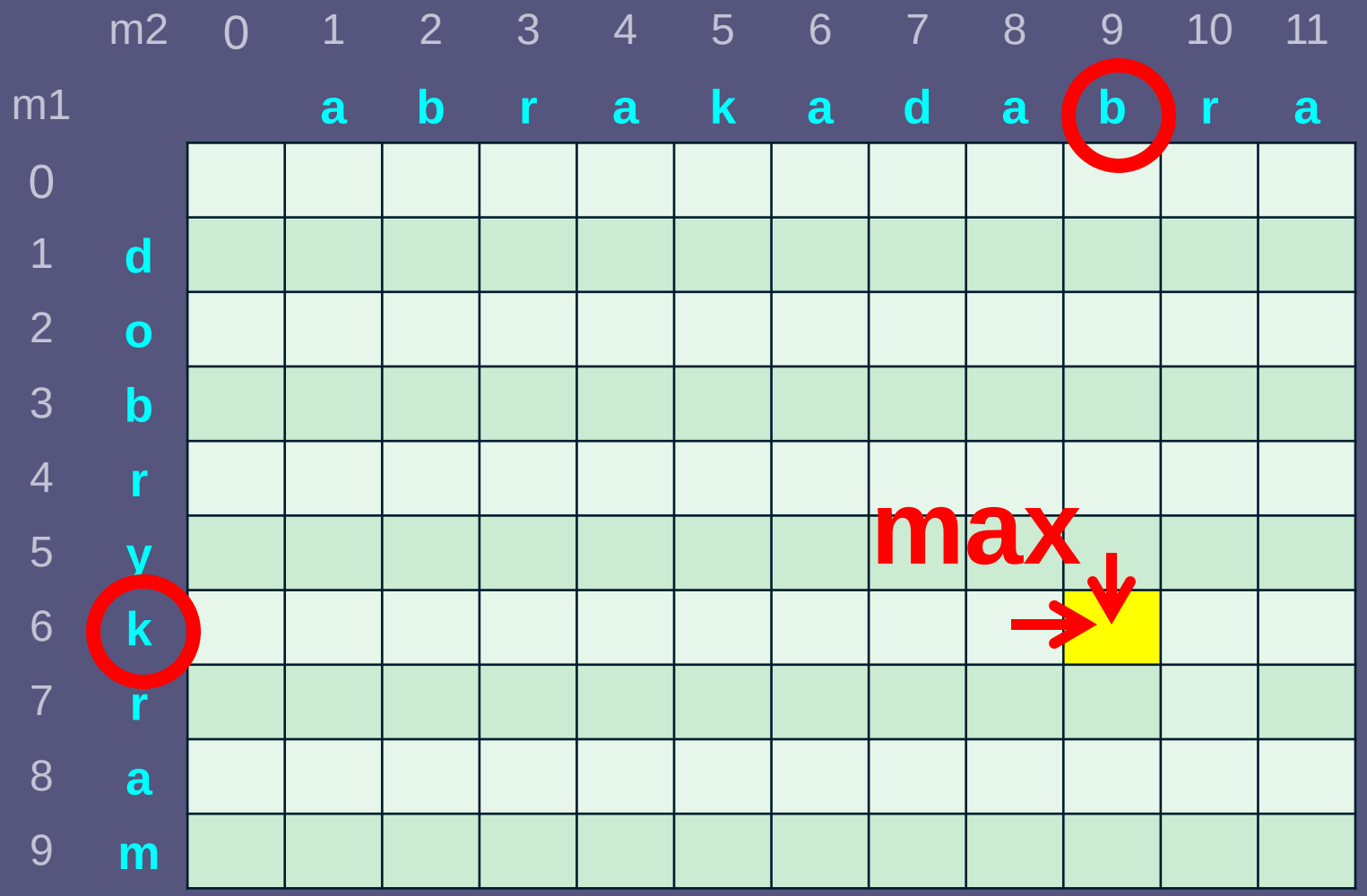
Podposloupnost dynamicky



Podposloupnost dynamicky



Podposloupnost dynamicky



Podposloupnost dynamicky

```
int best[][] = new int[len1+1][len2+1];
int commonLen (char[] s1, char[] s2) {
        for (int i = 0; i <= len1; ++i)
                best[i][0] = 0;
        for (int i = 0; i <= len2; ++i)
                best[0][i] = 0;
        for (int i = 1; i <= len1; ++i)
                for (int j = 1; j <= len2; ++j)
                        best[i][j] =
(s1[i-1] == s2[j-1]) ? 1 + best[i-1][j-1]
                : Math.max(best[i-1][j], best[i][j-1]);
        return best[len1][len2];
}
```

Jak zjistit řešení?

- Zatím máme jen délku výsledku
- Co když potřebujeme konkrétně?

1. Pamatujeme si pro každé „políčko“
2. Odvodíme na konci z matice

Podposloupnost – řešení

	m2	0	1	2	3	4	5	6	7	8	9	10	11
m1			a	b	r	a	k	a	d	a	b	r	a
0		0	0	0	0	0	0	0	0	0	0	0	0
1	d	0	0	0	0	0	0	0	1	1	1	1	1
2	o	0	0	0	0	0	0	0	1	1	1	1	1
3	b	0	0	1	1	1	1	1	1	1	2	2	2
4	r	0	0	1	2	2	2	2	2	2	2	2	2
5	y	0	0	1	2	2	2	2	2	2	2	2	2
6	k	0	0	1	2	2	3	3	3	3	3	3	3
7	r	0	0	1	2	2	3	3	3	3	3	4	4
8	a	0	1	1	2	3	3	4	4	4	4	4	5
9	m	0	1	1	2	3	3	4	4	4	4	4	5

Podposloupnost – řešení

	m2	0	1	2	3	4	5	6	7	8	9	10	11
m1			a	b	r	a	k	a	d	a	b	r	a
0		0	0	0	0	0	0	0	0	0	0	0	0
1	d	0	0	0	0	0	0	0	1	1	1	1	1
2	o	0	0	0	0	0	0	0	1	1	1	1	1
3	b	0	0	1	1	1	1	1	1	1	2	2	2
4	r	0	0	1	2	2	2	2	2	2	2	2	2
5	y	0	0	1	2	2	2	2	2	2	2	2	2
6	k	0	0	1	2	2	3	3	3	3	3	3	3
7	r	0	0	1	2	2	3	3	3	3	3	4	4
8	a	0	1	1	2	3	3	4	4	4	4	4	5
9	m	0	1	1	2	3	3	4	4	4	4	4	5

Podposloupnost – řešení

	m2	0	1	2	3	4	5	6	7	8	9	10	11
m1			a	b	r	a	k	a	d	a	b	r	a
0		0	0	0	0	0	0	0	0	0	0	0	0
1	d	0	0	0	0	0	0	0	1	1	1	1	1
2	o	0	0	0	0	0	0	0	1	1	1	1	1
3	b	0	0	1	1	1	1	1	1	1	2	2	2
4	r	0	0	1	2	2	2	2	2	2	2	2	2
5	y	0	0	1	2	2	2	2	2	2	2	2	2
6	k	0	0	1	2	2	3	3	3	3	3	3	3
7	r	0	0	1	2	2	3	3	3	3	3	4	4
8	a	0	1	1	2	3	3	4	4	4	4	4	5
9	m	0	1	1	2	3	3	4	4	4	4	4	5

Podposloupnost – řešení

	m2	0	1	2	3	4	5	6	7	8	9	10	11
m1			a	b	r	a	k	a	d	a	b	r	a
0		0	0	0	0	0	0	0	0	0	0	0	0
1	d	0	0	0	0	0	0	0	1	1	1	1	1
2	o	0	0	0	0	0	0	0	1	1	1	1	1
3	b	0	0	1	1	1	1	1	1	1	2	2	2
4	r	0	0	1	2	2	2	2	2	2	2	2	2
5	y	0	0	1	2	2	2	2	2	2	2	2	2
6	k	0	0	1	2	2	3	3	3	3	3	3	3
7	r	0	0	1	2	2	3	3	3	3	3	4	4
8	a	0	1	1	2	3	3	4	4	4	4	4	5
9	m	0	1	1	2	3	3	4	4	4	4	4	5

Podposloupnost – řešení

	m2	0	1	2	3	4	5	6	7	8	9	10	11
m1			a	b	r	a	k	a	d	a	b	r	a
0		0	0	0	0	0	0	0	0	0	0	0	0
1	d	0	0	0	0	0	0	0	1	1	1	1	1
2	o	0	0	0	0	0	0	0	1	1	1	1	1
3	b	0	0	1	1	1	1	1	1	1	2	2	2
4	r	0	0	1	2	2	2	2	2	2	2	2	2
5	y	0	0	1	2	2	2	2	2	2	2	2	2
6	k	0	0	1	2	2	3	3	3	3	3	3	3
7	r	0	0	1	2	2	3	3	3	3	3	4	4
8	a	0	1	1	2	3	3	4	4	4	4	4	5
9	m	0	1	1	2	3	3	4	4	4	4	4	5

Podposloupnost – řešení

	m2	0	1	2	3	4	5	6	7	8	9	10	11
m1			a	b	r	a	k	a	d	a	b	r	a
0		0	0	0	0	0	0	0	0	0	0	0	0
1	d	0	0	0	0	0	0	0	1	1	1	1	1
2	o	0	0	0	0	0	0	0	1	1	1	1	1
3	b	0	0	1	1	1	1	1	1	1	2	2	2
4	r	0	0	1	2	2	2	2	2	2	2	2	2
5	y	0	0	1	2	2	2	2	2	2	2	2	2
6	k	0	0	1	2	2	3	3	3	3	3	3	3
7	r	0	0	1	2	2	3	3	3	3	3	4	4
8	a	0	1	1	2	3	3	4	4	4	4	4	5
9	m	0	1	1	2	3	3	4	4	4	4	4	5

Optimalizační problémy

- Pokud stačí hodnota řešení
  - Pamatujeme pouze nejlepší číslo
- Pokud chceme kompletní řešení
  - Pamatujeme si všechny výsledky
  - Rekonstruujeme po dokončení D.P.

Jiný příklad: Hra s mincemi

- Hrají 2 hráči
- Mají před sebou řadu mincí různé hodnoty
- Tah = odebrání mince zleva nebo zprava
- Cíl: sesbírat mince s maximální hodnotou

12	30	10	6	17	42	31	15	18	26	13	4	22	8	5

Hry obecně

- „Tahové“ hry často vedou na rekurzi
- Vysoká operační složitost

```
int getBestResult(State s, Player move) {
  if (isGameOver(s)) return endResult(s);
  for (State m : getAllMoves(s)) {
      int x = reverseResult(
            getBestResult(m, other(move)));
      if (x > best) best = x;
  }
  return best;
```

Hry obecně

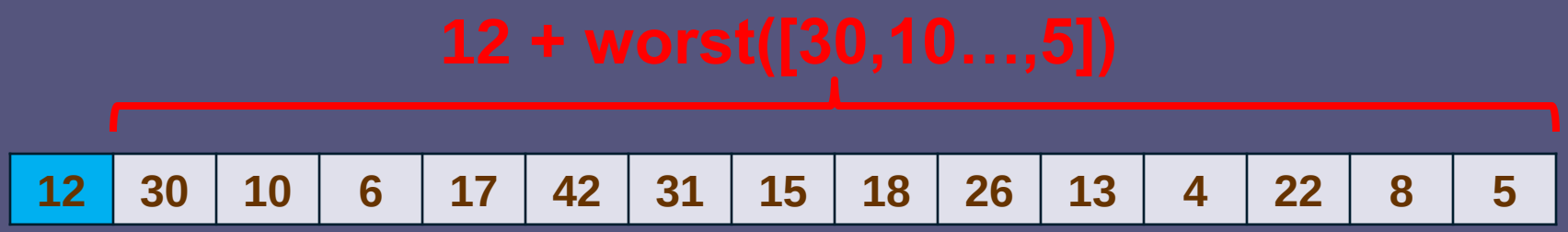
- „Tahové“ hry často vedou na rekurzi
- Vysoká operační složitost

- Pokud se stavy „spojují“, lze použít dynamické programování

Hra s mincemi

- Co kdybychom znali řešení všech menších instancí

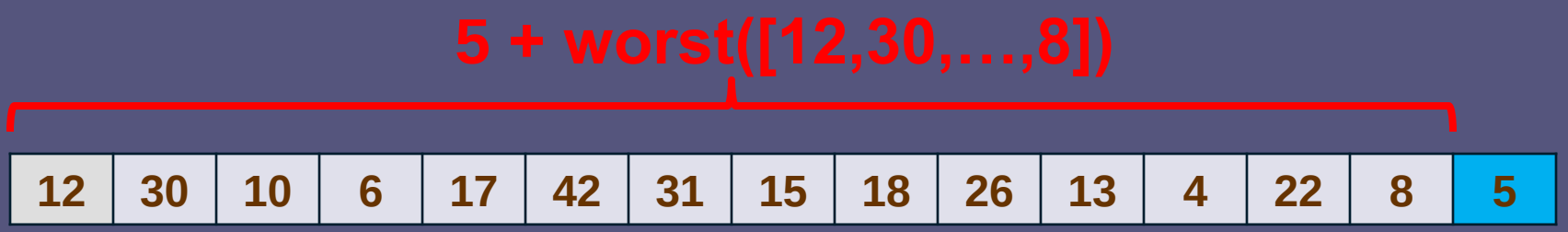
- Zkusíme odebrat obě krajní
- Použijeme lepší výsledek pro nás (tj. horší pro zbytek <= hraje druhý)



Hra s mincemi

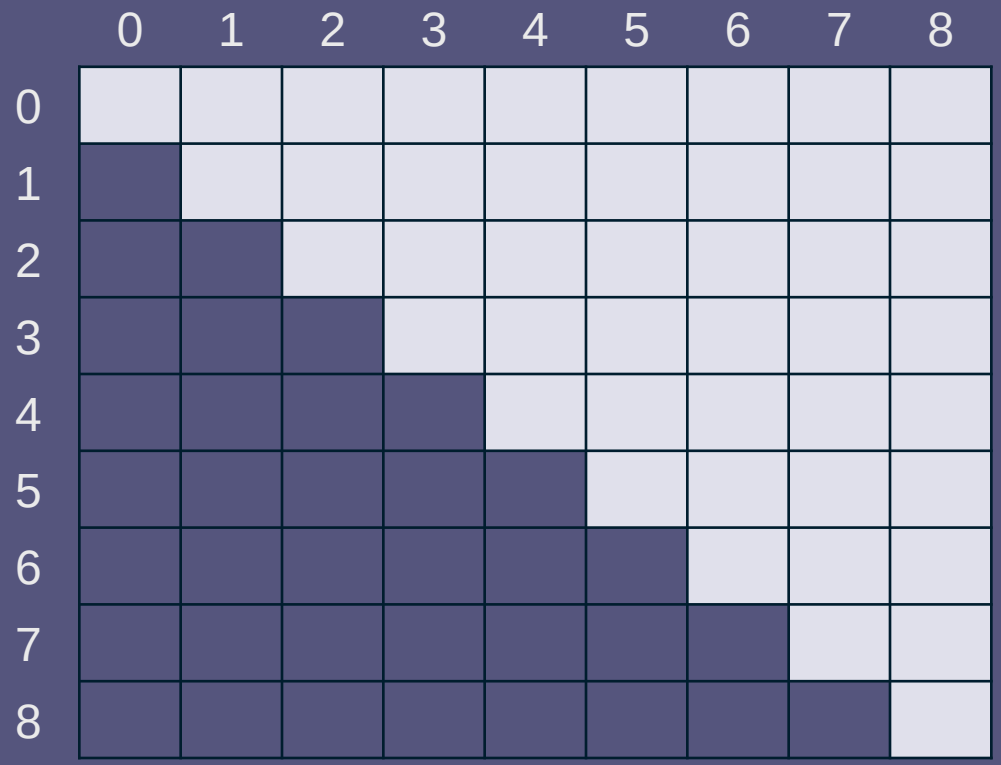
- Co kdybychom „znali“ řešení všech menších instancí

- Zkusíme odebrat obě krajní
- Použijeme lepší výsledek pro nás (tj. horší pro zbytek <= hraje druhý)



Hra s mincemi: uložení

- Nejlepší řešení pro interval **i** – **j**

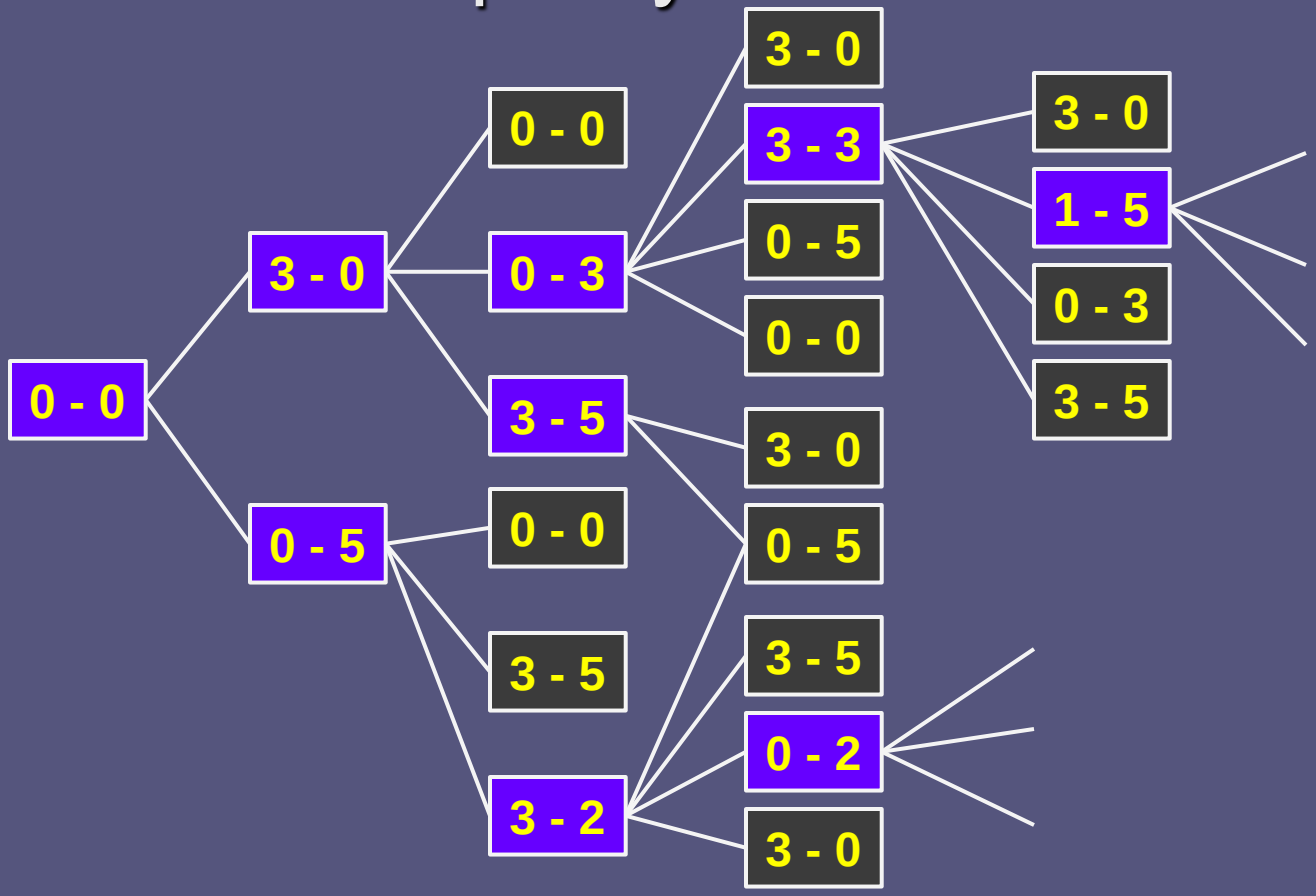


Příklad: přelévání nádob

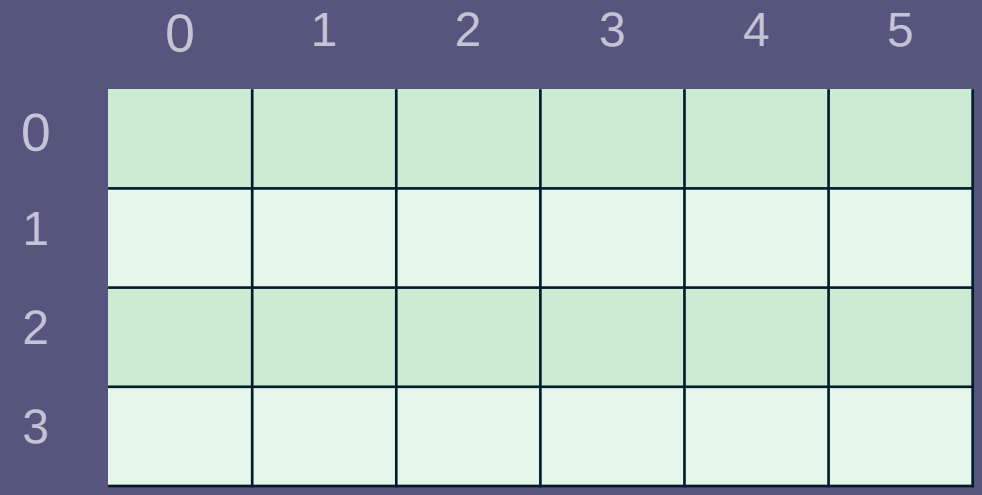
- Máme
  - 2 nádoby o objemu A a B
  - Zdroj vody
  - Kanál na vylití nepotřebné vody
- Úkolem je přesně odměřit objem C
  - Jde to vždy?

Přelévání: strom možností

- Příklad: Kapacity nádob 3 a 5 litrů



Přelévání: pamatování



- Co si pamatovat
  - Počet tahů (?)
  - Poslední tah (→ rekonstrukce)

Typické příklady D.P.

- Posloupnosti
  - Rozdělování
  - Vyhledávání
  - Porovnávání
- Kombinování prvků
- Acyklické grafy