Matematická olympiáda – kategorie P

Co se dozvíte?

- Základní informace o MO-P.
- Jak by mělo vypadat řešení?
- Pár užitečných algoritmů.

Co je MO-P?

- Programovací soutěž,
- zaměřená na efektivní řešení konkrétních úloh
- matematického/teoretického typu;
- vstup a výstup v jednoduchém formátu, žádná další interakce s uživatelem.

Více informací: mo.mff.cuni.cz.

Organizace soutěže

Domácí kolo:

- Do 15. listopadu.
- 2 teoretické a 2 praktické úlohy
- Odevzdává se na osmo.matematickaolympiada.cz
- Typická postupová hranice: Cca 10 bodů (z 40).

Krajské kolo:

- 17. ledna.
- 4 hodiny, 4 teoretické úlohy řešené na papíře.

Ústřední kolo:

- 30 nejlepších účastníků.
- 2×5 hodin, 3 teoretické a 3 praktické úlohy
- 22.–24. března ve Zlíně

Výběrové/přípravné soustředění:

- pro cca 10 nejlepších účastníků

Mezinárodní soutěže

Mezinárodní olympiáda v programování (IOI)

- 4 nejlepší účastníci
- 28.8.–4.9., Szeged, Maďarsko

Středoevropská olympiáda v programování (CEOI)

- 4 nejlepší nematuranti, kteří se nekvalifikovali na IOI
- 13.-19.8., Magdeburg, Německo

Co by mělo obsahovat řešení – praktické úlohy

- Zdrojový kód řešení:
  - Pascal, C, C++, Java, Python

Vyhodnocení:

- Automatické testování na konkrétních vstupech různé velikosti.
- Musí doběhnout dostatečně rychle a vrátit správnou odpověď.

body	n
2	$n \leq 10$
2	$n \leq 1\,000$
3	$n \leq 100\,000$
3	$n \leq 1\,000\,000$

Co by mělo obsahovat řešení – teoretické úlohy

- Co nejsrozumitelnější popis řešení.
- Zdůvodnění jeho správnosti.
- Určení a zdůvodnění časové a paměťové složitosti.
- Zdrojový kód řešení:
  - v libovolném programovacím jazyce nebo pseudokódu
  - je možné vynechat nedůležité části (vstup, výstup, standardní algoritmy)

Vyhodnocení:

- Chybné/neoptimální řešení: 0 bodů.
- Jinak:
  - Cca 3 body za kvalitu popisu a zdůvodnění.
  - $0 - 7$ bodů dle efektivity.

Model

Poslední teoretická úloha:

- delší studijní text
- netradiční výpočetní prostředky
- platí vše co pro ostatní teoretické úlohy
  - „časová složitost“ a „zdrojový kód“ mohou mít jiný význam, popsaný ve studijním textu

Příklad

Radek se vypravil na veletrh dortů, tvořený stánky očíslovanými 1 až n; na i-tém z nich může za a_i Kč ochutnat dort. Radek má k Kč a chce ochutnat co nejvíc dortů. Jakmile ale poprvé ochutná dort v nějakém stánku i, nedá mu to a ochutná poté dort ve vedlejší stánku $i + 1$, následně ve stánku $i + 2$ a tak dále, než mu dojdou peníze nebo než dojde na konec řady. Ve kterém stánku má Radek začít, aby ochutnal co nejvíce dortů?

Příklad vstupu:

```
7 6                    7 stánků, 6 korun
2 1 2 3 2 2 1    ceny dortů na stáncích
```

Najděte řešení, které problém efektivně vyřeší pro $n = 1\,000\,000$.

Popis řešení

Pro každý stánek z, ve kterém Radek může začít jíst dorty, si určíme číslo stánku e_z, ve kterém skončí, a budeme si také pamatovat, jakou částku m_z za dorty utratí. Pro první stánek $z = 1$ prostě projdeme následující stánky a přičítáme ceny jejich dortů, dokud by jejich cena nepřekročila k (nebo dokud nedojdeme na konec).

Předpokládejme nyní, že už máme spočítáno e_z a m_z pro nějakou hodnotu z. Ukážeme si, jak je spočítáme pro $z + 1$. Zjevně stačí snížit m_z o hodnotu dortu na stánku z a poté ji postupně zvyšovat o hodnoty dortů na stáncích $e_z + 1$, $e_z + 2$, ..., dokud bychom opět nepřekročili k nebo nedošli na konec.

Skončíme ve chvíli, kdy e_z dorazí na konec řady stánků (od té chvíle se počet snězených dortů může jen snižovat). Vrátíme maximum z hodnot $e_z - z + 1$ za celý běh programu. Samozřejmě si nemusíme ukládat všechny hodnoty e_z a m_z, stačí si pamatovat pouze ty pro aktuální z a maximum počítat průběžně.

Zdůvodnění správnosti

Popsaný algoritmus zjevně zaručí, že po zpracování začátku z dorty v úseku mezi z a e_z stojí $m_z \leq k$ Kč, ale dorty v úseku mezi z a $e_z + 1$ (nebo v jakémkoliv delším úseku) stojí více než k Kč. Pro každý začátek z tedy přesně určíme počet dortů $e_z - z + 1$, které by Radek snědl, kdyby začal u stánku z. Nutně tedy mezi nimi nalezneme i optimální hodnotu.

Časová složitost – O-notace

- Určit počet provedených operací přesně je složité.
- Operace trvají různě dlouho, čas jejich provedení závisí na stavu paměti, ...

Místo toho: „Časová složitost je $O(f(n))$.“

- Existuje nějaká konstanta c taková, že pro každé $n \geq 2$ algoritmus provede nejvýše $c \cdot f(n)$ operací.
- Více viz ksp.mff.cuni.cz/kucharky/slozitost/

Určení a zdůvodnění časové a paměťové složitosti

Povšimněme si, že hodnota e_z se v průběhu programu pouze zvyšuje, může se tedy změnit pouze n-krát. Hodnotu m_z měníme pouze tehdy, když se zvýší z nebo e_z. Celkový čas strávený úpravami e_z a m_z je tedy $O(n)$.

Kromě upravování hodnot e_z a m_z pro každé z provádíme jen konstantně mnoho operací (úprava aktuálního maxima, zvýšení z, ...), ty tedy dohromady také zaberou pouze čas $O(n)$. Celková časová složitost je tedy $O(n)$.

Pamětová složitost: Kromě pole délky n obsahujícího vstup si udržujeme pouze několik proměnných konstantní velikosti, paměťová složitost je tedy $O(n)$.

Běžné složitosti

Současné počítače provádí řádově 10^9 operací za sekundu:

- $O(n)$ algoritmy bývají efektivní pro cca $n \leq 10^7$
- $O(n^2)$ algoritmus možná pro $n \leq 10\,000$
- $O(2^n)$ pro $n \leq 30$

Zdrojový kód

Přeskakuji načtení vstupu do pole `ceny`. Do `ceny`[*n*] si uložím ∞, abych nemusel zvlášť ošetřovat konec.

```
void prepocet_e_a_m (void) {
  while (m + ceny[e + 1] <= k) {
    e++; m += ceny[e];
  }
  vysledek = max (vysledek, e - z + 1);
}

z = e = m = 0;
prepocet_e_a_m ();
while (e != n - 1) {
  m -= ceny[z]; z++;
  prepocet_e_a_m ();
}
```

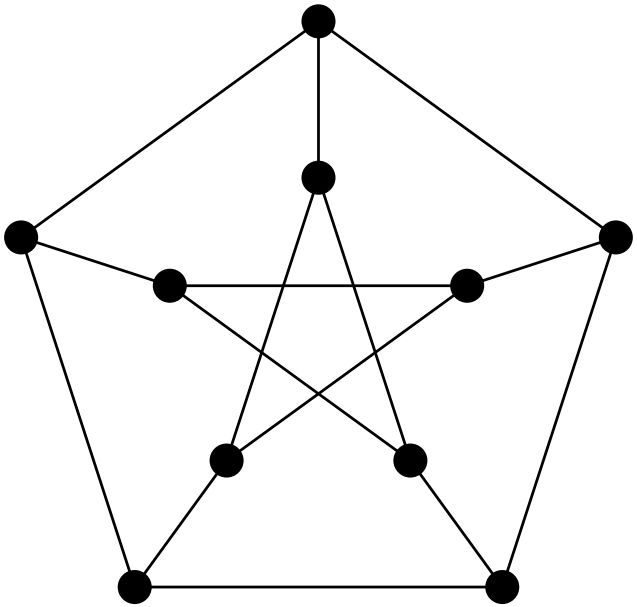
Co u teoretických úloh nedělat

- Odevzdat pouze zdrojový kód (i komentovaný).
- Přepis programu do češtiny:
  - „Nejprve nastavíme *z*, *e* a *m* na 0. Pak zvyšujeme *e* o 1 a pokaždé k *m* přičteme `ceny`[*e*], dokud *e* nebude větší než *k*. . . .“
- Nevhodně pojmenované proměnné:
  - „Zavedeme si pole `pole`. Dále si zavedeme pole `pole2`.“
  - „K právě spočítanému číslu přičteme druhý z počtů určených v předminulém odstavci.“
- Popis běhu algoritmu na jednom konkrétním příkladě.
- Rozebírání nedůležitých technických detailů.

Co naopak dělat

- Popis srozumitelný i bez nahlédnutí do programu.
- Soustředit se na vysvětlení významu, ne na technikálie.
  - „e_z bude číslo stánku, kde Radek skončí, pokud začne na stánku číslo z“
- Příklady ke složitějším definicím.
  - „Například ve vzorovém vstupu budeme mít $e_3 = 4$, jelikož dorty na stáncích 3 a 4 dohromady stojí $5 \leq k$ Kč, ale na stáncích číslo 3 až 5 by stály $7 > k$ Kč.“
- Inspirujte se autorskými řešeními.

Grafy



Příklady

Dopravní síť ve městě:

- vrcholy: křižovatky
- hrany: ulice

Vztahy:

- vrcholy: lidé
- hrany: znají se?

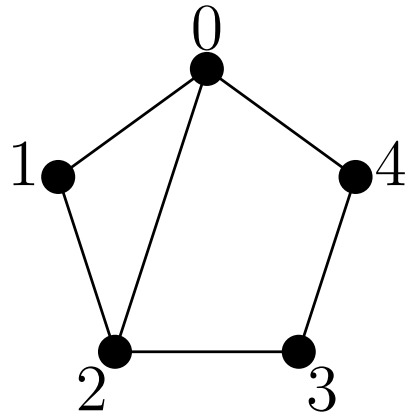
Molekuly:

- vrcholy: atomy
- hrany: vazby mezi nimi

Reprezentace grafů

Seznam sousedů:

- $0 \to 1, 2, 4$
- $1 \to 0, 2$
- $2 \to 0, 1, 3$
- $3 \to 2, 4$
- $4 \to 0, 3$

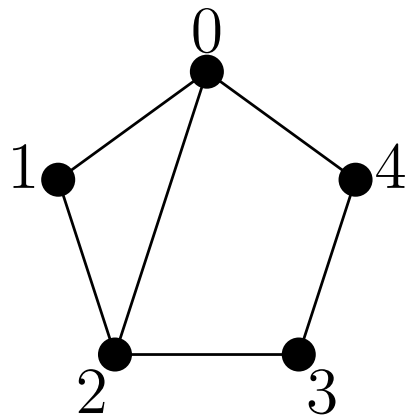


- `vector<list<int>>`
- Stupeň v: $\deg v =$ počet sousedů v.
- Sousedi vrcholu v v čase $O(\deg v)$.
- Sousedí u s v? V čase $O(\min(\deg u, \deg v))$.
- Paměťová složitost: $O(\text{vrcholů} + \text{hran})$.

Reprezentace grafů

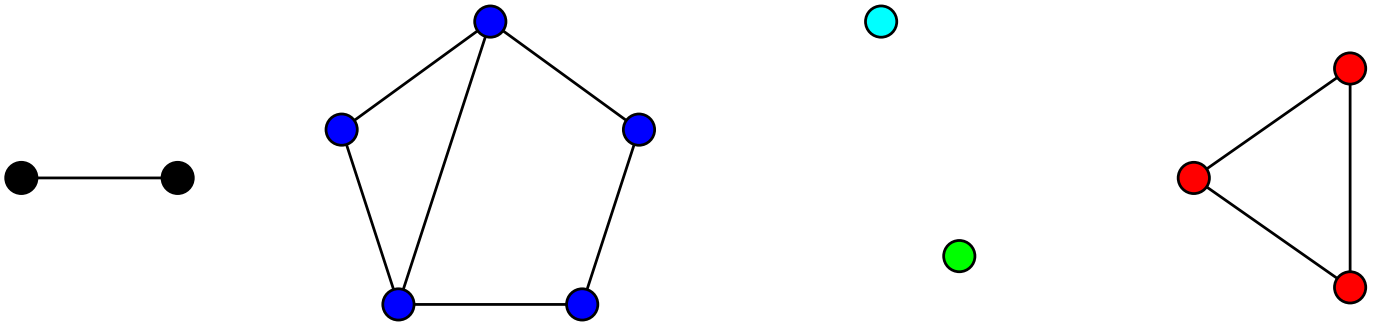
Matice sousednosti:

	0	1	2	3	4
0	0	1	1	0	1
1	1	0	1	0	0
2	1	1	0	1	0
3	0	0	1	0	1
4	1	0	0	1	0



- `bool sousedi[n][n]`
- Sousedi vrcholu v v čase $O(n)$.
- Sousedí u s v? V čase $O(1)$.
- Paměťová složitost: $O(n^2)$.

Komponenty souvislosti



- Graf je souvislý, jestliže se mezi každými dvěma vrcholy dá dostat po hranách.
- Komponenta souvislosti: Maximální souvislá část.

Jak najít komponenty souvislosti?

- Z libovolného vrcholu postupně procházím sousedy, pak sousedy sousedů, ...
- Dávám si pozor, abych nezpracovával vrchol víc než jednou.

Časová složitost: $O(\text{vrcholů} + \text{hran})$.

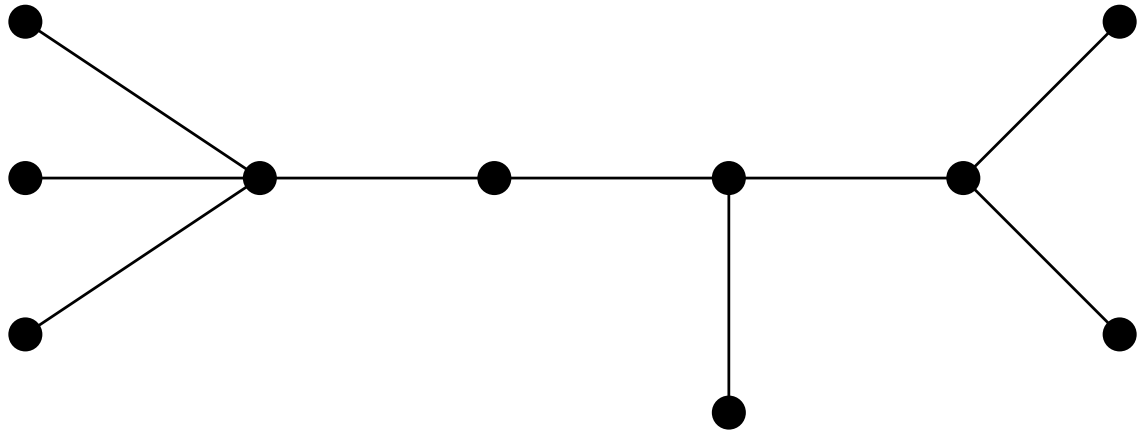
```
vector<list<int>> sousedi;
vector<int> cislo_komponenty (n, -1);
list<int> komponenta (int v, int cislo) {
  list<int> komp{v}; vector<int> fronta{v};
  cislo_komponenty[v] = cislo;
  while (!fronta.empty ()) {
    int x = fronta.back (); fronta.pop_back ();
    for (int y :  sousedi[x])
      if (cislo_komponenty[y] == -1) {
        cislo_komponenty[y] = cislo;
        komp.push_back (y); fronta.push_back (y);
      }}
  return komp;
}
int cislo = 0;
for (int v = 0 ; v < n; v++)
  if (cislo_komponenty[v] == -1)
    komponenta (v, cislo++);
```

Grafy – více informací

ksp.mff.cuni.cz/kucharky/grafy/

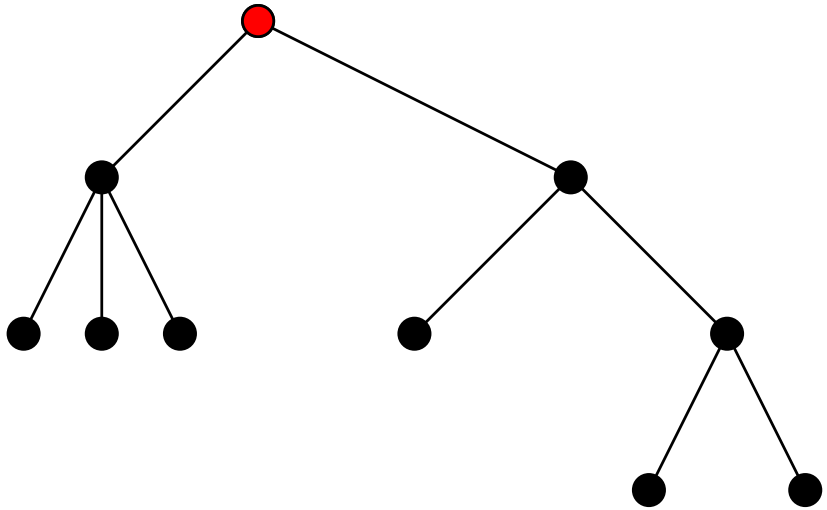
Stromy

Souvislé grafy bez cyklů.



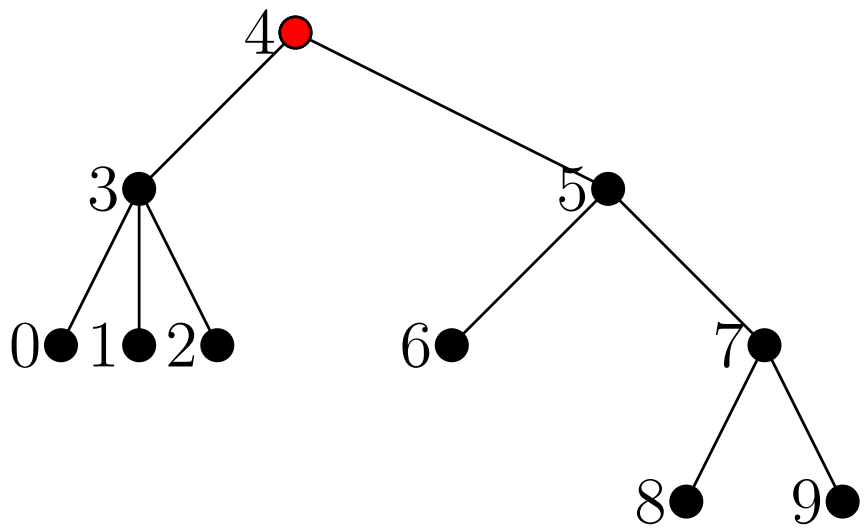
Reprezentace:

- Libovolně zakořeníme.
- Pro každý vrchol:
  - Rodič.
  - Seznam dětí.



Reprezentace stromu

	Rodič	Děti
0	3	
1	3	
2	3	
3	4	0,1,2
4	$\perp$	3,5
5	4	6,7
6	5	
7	5	8,9
8	7	
9	7	



Konstrukce reprezentace:

- Přímo ze zadání (rodič každého vrcholu).
- Průchod podobně jako hledání komponent.

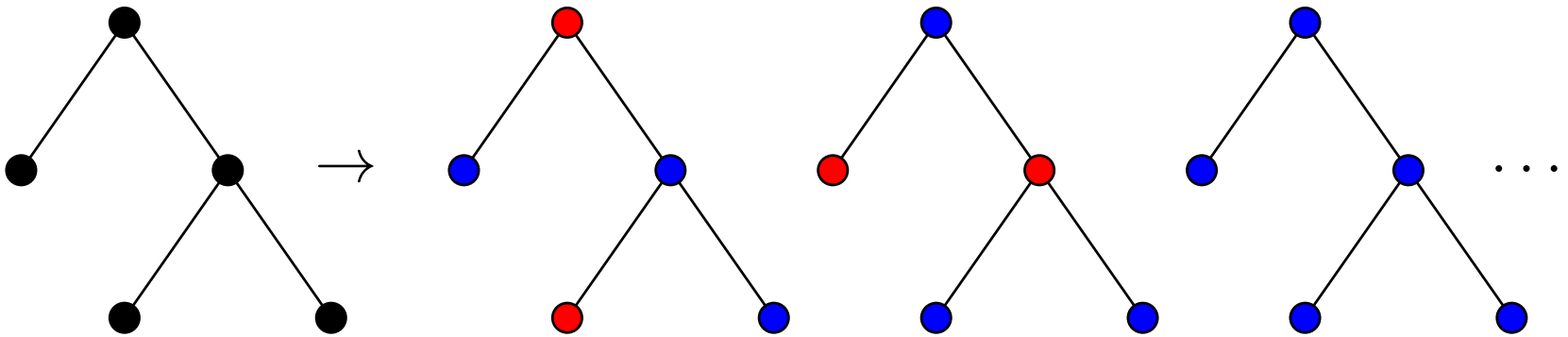
Dynamické programování na stromu

Typický postup řešení úloh na stromech:

- Úlohu vyřešíme pro podstrom pod každým vrcholem.
- Řešíme od listů / nejmenších podstromů nahoru.
- Při řešení pro daný vrchol využijeme výsledky pro děti.

Příklad

Pro zadaný strom spočítejte počet obarvení jeho vrcholů červeně a modře takových, že žádné dva červené vrcholy nesousedí.

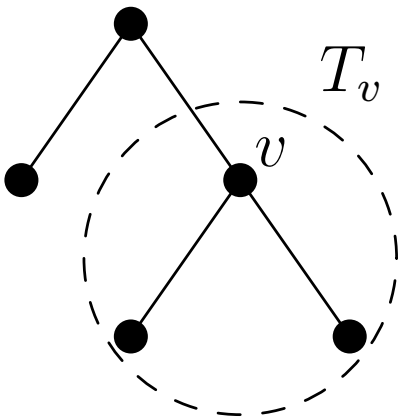


Řešení

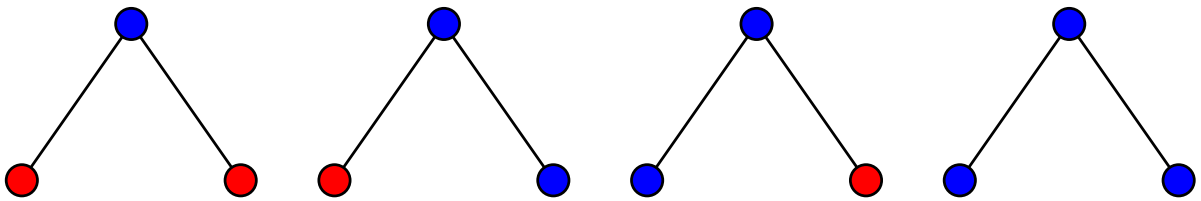
Pro každý vrchol v a podstrom T_v tvořený vrcholy pod v určíme

- $m(v) =$ počet obarvení T_v, v nichž v má modrou barvu a
- $c(v) =$ počet obarvení T_v, v nichž v má červenou barvu.

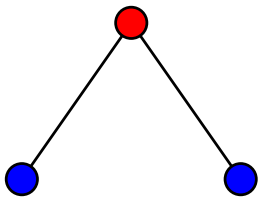
Výsledek: $m(\text{kořen}) + c(\text{kořen})$.



$m(v) = 4$



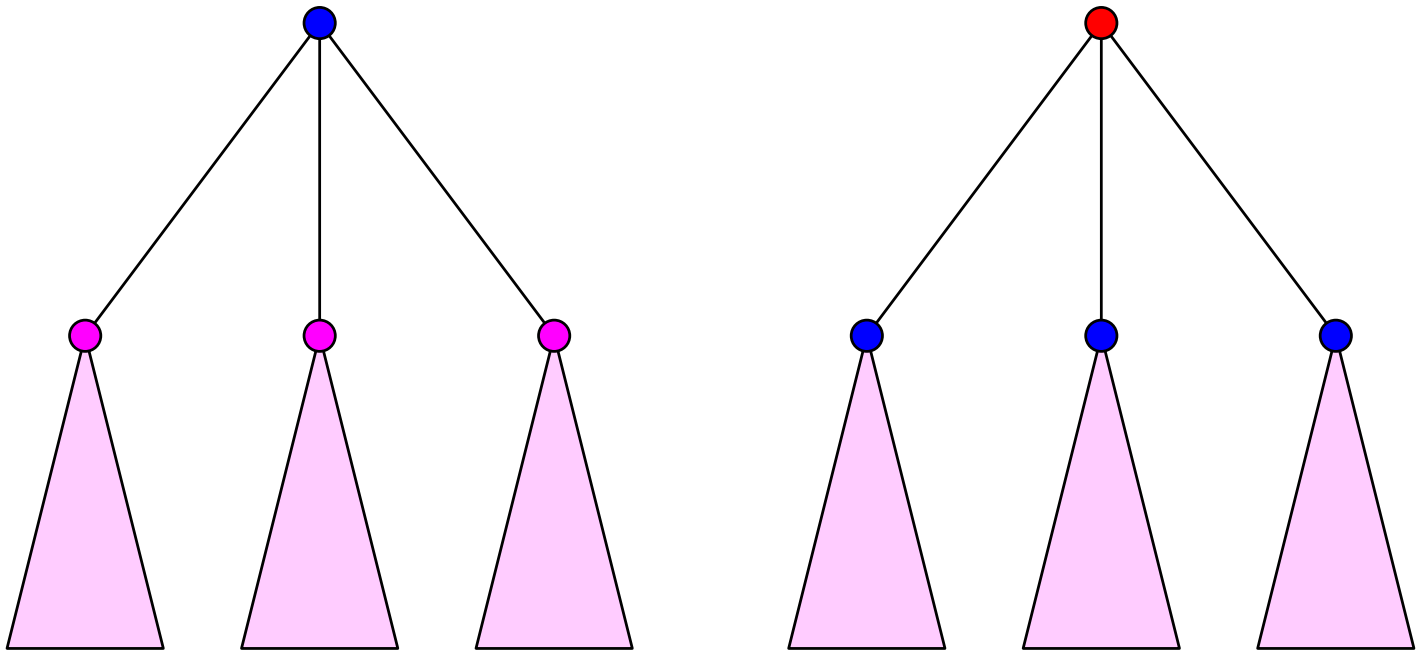
$c(v) = 1$



Výpočet $m(v)$ a $c(v)$

Pro vrchol v s dětmi $u_1, \ldots, u_k$:

- $m(v) = (m(u_1) + c(u_1)) \cdot (m(u_2) + c(u_2)) \cdots (m(u_k) + c(u_k))$
- $c(v) = m(u_1) \cdot m(u_2) \cdots m(u_k)$



```
urci_pocty (int v, int &m, int &c) {
    m = c = 1;
    for (int d : deti[v]) {
        int md, cd;
        urci_pocty (d, md, cd);
        m *= (md + cd);
        c *= md;
    }
}

int m, c;
urci_pocty (koren, m, c);
return m + c;
```

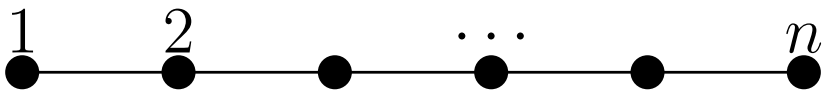
Časová složitost: O(počet vrcholů).

Dynamické programování – více informací

ksp.mff.cuni.cz/kucharky/dynamicke-programovani/

Pozor na přetečení!

Cesta s n vrcholy:



n	1	2	3	4	...	44	45
obarvení	2	3	5	8	...	1836311903	-1323752223

Typické zadání: „vypište zbytek počtu obarvení po dělení $10^9 + 7$.“

Užitečné vztahy:

- $(a + b) \bmod M = ((a \bmod M) + (b \bmod M)) \bmod M$
- $(a \cdot b) \bmod M = ((a \bmod M) \cdot (b \bmod M)) \bmod M$
- Dělení: složitější, ksp.mff.cuni.cz/kucharky/teorie-cisel/

```
#define M 1000000007

urci_pocty (uint64_t v, uint64_t &m, uint64_t &c) {
    m = c = 1;
    for (int d : deti[v]) {
        uint64_t md, cd;
        urci_pocty (d, md, cd);
        m = (m * (md + cd)) % M;
        c = (c * md) % M;
    }
}

uint64_t m, c;
urci_pocty (koren, m, c);
return m + c;
```

Další zdroje

- Kuchařky KSP: ksp.mff.cuni.cz/kucharky
- Programátorská liaheň: liahen.ksp.sk
- Průvodce labyrintem algoritmů: pruvodce.ucw.cz
- MO-P FAQ: mo.mff.cuni.cz/p/otazky_a_odpovedi.html