

Matematická olympiáda – kategorie P

mo.mff.cuni.cz

Co se dozvíte?

- Základní informace o MO-P.
- Návodné úlohy, trénink.
- Jak by mělo vypadat řešení?

Co je MO-P?

- Programovací soutěž,
- teoretičtější zaměření, důraz na efektivitu,
- vstup a výstup v jednoduchém formátu, žádná další interakce s uživatelem.

Organizace soutěže (1)

Domácí kolo:

- Do 15. listopadu.
- 2 teoretické a 2 praktické úlohy.
 - Jedna v nestandardním výpočetním modelu.
- Odevzdávání viz mo.mff.cuni.cz.
- Typická postupová hranice: Cca 10 bodů (z 40).

Krajské kolo:

- 19. ledna.
- 4 hodiny, 4 teoretické úlohy.
- Nebo možná 2 teoretické + 2 praktické úlohy?

Ústřední kolo:

- 30 nejlepších účastníků.
- 2×5 hodin, 3 teoretické a 3 praktické úlohy
- 17.–19. března v Hradci Králové

Výběrové/přípravné soustředění:

- pro cca 10-15 nejlepších účastníků
- duben?

Mezinárodní olympiáda v programování (IOI)

- 4 nejlepší účastníci
- Německo, 12.–19.9. 2027

Středoevropská olympiáda v programování (CEOI)

- 4 nejlepší účastníci, kteří se nekvalifikovali na IOI
- Polsko, 30.8.–5.9. 2027

Evropská dívčí olympiáda v programování (EGOI)

- 4 nejlepší dívky
- Polsko, ???

Kontrolujeme vstupenky u vchodu do kina. Na vstupence je čárový kód, v němž je zakódované její unikátní číslo v rozmezí od 1 do n , kde $n \approx 10^6$. Občas se někdo pokusí podvádět a vytiskne si vstupenku s vlastním čárovým kódem, nebo něčí vstupenku zkopíruje. Proto jen některá čísla z daného rozmezí odpovídají platným vstupenkám, a chtěli bychom umět odhalit následující situace:

- Někdo nám dá vstupenku s neplatným číslem.
- Více než jeden člověk se pokusí použít vstupenku se stejným číslem.

Vstup:

- Celá čísla n ($1 \leq n \leq 10^6$), p ($0 \leq p \leq n$) a d ($0 \leq d \leq 10^6$).
- Navzájem různá celá čísla $a_1, \dots, a_p$ ($1 \leq a_i \leq n$) udávající platné vstupenky.
- Celá čísla $x_1, \dots, x_d$ udávající čísla vstupenek příchozích.

Výstup: Pro $i = 1, \dots, d$ vypište řádek obsahující

- `OK`, je-li vstupenka x_i platná a různá od všech předchozích,
- `falesna`, když vstupenka x_i není platná, či
- `duplikat`, když vstupenka x_i je platná, ale už jsme ji viděli.

Vstup:

```
100 5 5
1 24 28 90 65
1 24 26 24 26
```

Výstup:

```
OK
OK
falesna
duplikat
falesna
```

- Uložíme si seznam $A = a_1, \dots, a_p$ platných vstupenek.
- Vytvoříme si seznam V platných vstupenek, které už jsme viděli (na začátku prázdný).
- Pro $i = 1, \dots, d$:
 - Jestliže x_i není v seznamu A , vypíšeme `falesna` a pokračujeme.
 - Jinak víme, že vstupenka x_i je platná. Jestliže x_i je v seznamu V , vypíšeme `duplikat` a pokračujeme.
 - Zbývá případ, že je vstupenka platná a vidíme ji poprvé. Vložíme x_i do seznamu V a vypíšeme `OK`.

Zdrojový kód

```
vector<int> A, V;  
// Vynechávám načtení n, p, d, A  
  
for (int i = 0; i < d; i++) {  
    int x << cin;  
    bool je_v_A = false;  
    for (int a : A)  
        if (x == a) { je_v_A = true; break; }  
    if (!je_v_A) {cout << "falesna\n"; continue;}  
  
    bool je_ve_V = false;  
    for (int v : V)  
        if (x == v) { je_ve_V = true; break; }  
    if (je_ve_V) {cout << "duplikat\n"; continue;}  
  
    V.push_back (x); cout << "OK\n";  
}
```

Jak dobré je toto řešení?

- Vnější cyklus se provede $d \times$
- V každé jeho iteraci procházíme přes A (p kroků) a přes V (nanejvýš p kroků).
- Celkem se tedy každý příkaz uvnitř těchto cyklů provede nejvýše $d \cdot p \times$.

Časová složitost tedy bude nejvýše $d \cdot p \times$ nějaká těžko určitelná (relativně malá) konstanta (záležející na přesných jednotkách).

Časová složitost – O-notation

- 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 algoritmus provede nejvýše $c \cdot f(n)$ operací.
- Více viz ksp.mff.cuni.cz/encyklopedie/slozitost/

V našem případě:

$$O(d \cdot p)$$

Jak dobré/špatné je $O(d \cdot p)$?

- Můžeme mít $d, p \approx 10^6$, $d \cdot p \approx 10^{12}$.
- Běžný počítač zvládá řádově 10^9 operací za sekundu.
- Čas v řádu hodin.

- Místo seznamu A můžeme mít pole indexované od 1 do n , v němž na pozici $A[a]$ je `true`, právě když je vstupenka číslo a platná, a `false` jinak.
- Obdobně pro V .
- Přidání prvku: Změna jedné hodnoty.
- Ověření existence prvku: Test jedné hodnoty.

Zdrojový kód

```
// Načtení n, p, d
vector<bool> A(n+1, false), V(n+1, false);
// Načtení A

for (int i = 0; i < d; i++)
{
    int x << cin;
    if (!A[x])
        {cout << "falesna" << endl; continue;}
    if (V[x])
        {cout << "duplikat" << endl; continue;}
    V[x] = true; cout << "OK" << endl;
}
```

- Časová složitost $O(n + p + d)$
- $n + p + d \approx 10^6$: “na odklepnutí”

Možné komplikace

Co když jsou kódy vstupenek

- čísla z výrazně většího rozsahu,
- nebo třeba řetězce?

Různá komplikovanější, ale efektivní řešení (vyhledávací stromy, hešování, ...)

- Viz mo.mff.cuni.cz/p/knihovna.html.
- Prakticky: `map` a `unordered_map` v C++, `dictionary` v Pythonu, ...
- Je vhodné si nastudovat (a rozumět jejich časové složitosti).

Co by mělo obsahovat řešení – praktické úlohy

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

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.
 - A případně zdrojový kód „důležitých“ částí.
- Zdůvodnění jeho správnosti.
- Určení a zdůvodnění časové a paměťové složitosti.

Vyhodnocení:

- Chybné řešení: 0 bodů.
- Jinak:
 - Cca 3 body za kvalitu popisu a zdůvodnění,
 - plus 0 – 7 bodů dle efektivity.

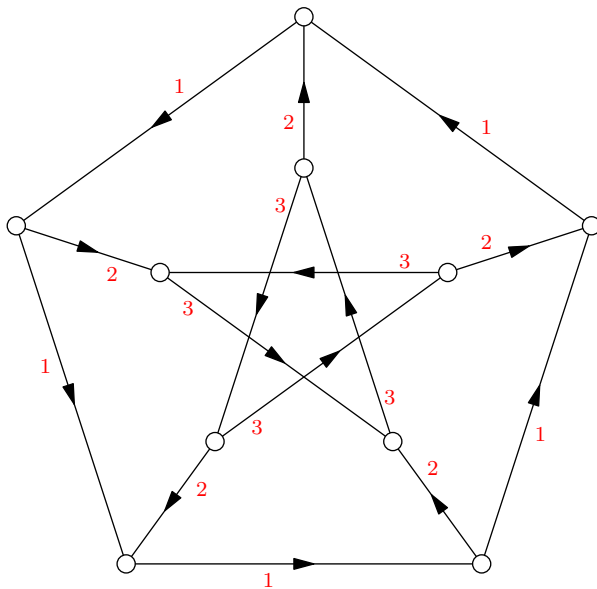
Co u teoretických úloh nedělat

- Odevzdat pouze zdrojový kód (i komentovaný).
- Přepis programu do češtiny:
 - „Ve for cyklu procházím p od 1 do $n - 1$. V každé iteraci `nejv[p][0]` nastavím na ...“
- Nevhodně pojmenované proměnné:
 - „Zavedeme si pole `pole`. Dále si zavedeme pole `pole2`.“
 - „K právě spočítanému číslu přičteme druhý z počtů určených v předminulém odstavci.“
- Pouze popis běhu algoritmu na jednom konkrétním příkladě.
- Rozebírání nedůležitých technických detailů.
- Vynechání důležitých technických detailů.

Poslední úloha:

- delší studijní text
- netradiční výpočetní prostředky
- jinak obdobně jako teoretické úlohy

- Korespondenční seminář z programování KSP:
ksp.mff.cuni.cz
 - **Encyklopedie.**
 - Začátečnická kategorie KSP-Z.
 - Soustředění.
- Úlohy z archivu (MO-P, KSP).
- Kniha Algoritmy a programovací techniky.
- Pokročilejší: kniha Průvodce labyrintem algoritmů
pruvodce.ucw.cz



Dopravní síť ve městě:

- vrcholy: křižovatky
- hrany: jednosměrné ulice s délkami

Vztahy (graf bez vah):

- vrcholy: lidé
- hrany: zná ho?

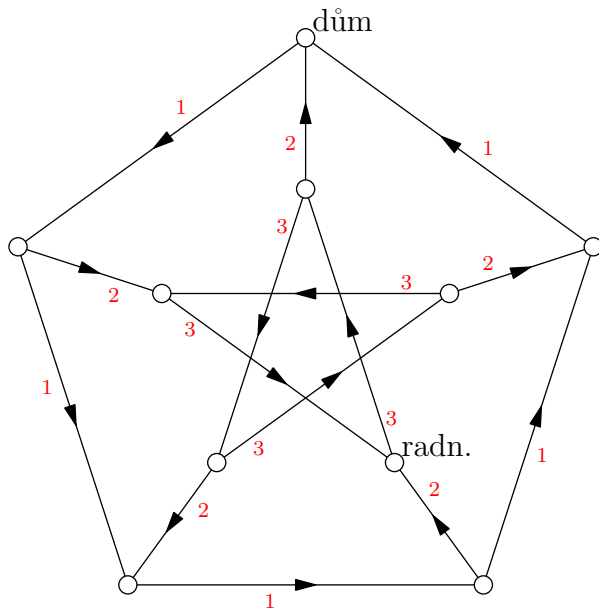
Molekuly (graf bez vah, orientací):

- vrcholy: atomy
- hrany: vazby mezi nimi

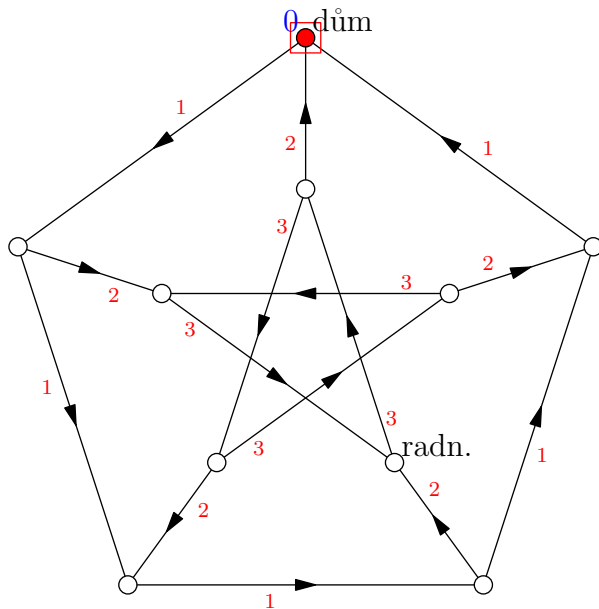
ksp.mff.cuni.cz/encyklopedie/grafy/

V Kocourkově jsou všechny ulice jednosměrné. U každé ulice víme, ze které na kterou křižovatku vede a jak je dlouhá (mimo křižovatky se ulice nepotkávají). Na jedné zadané křižovatce je váš dům, na jiné je radnice. Určete, jakou nejkratší cestou se můžeme dostat na radnici?

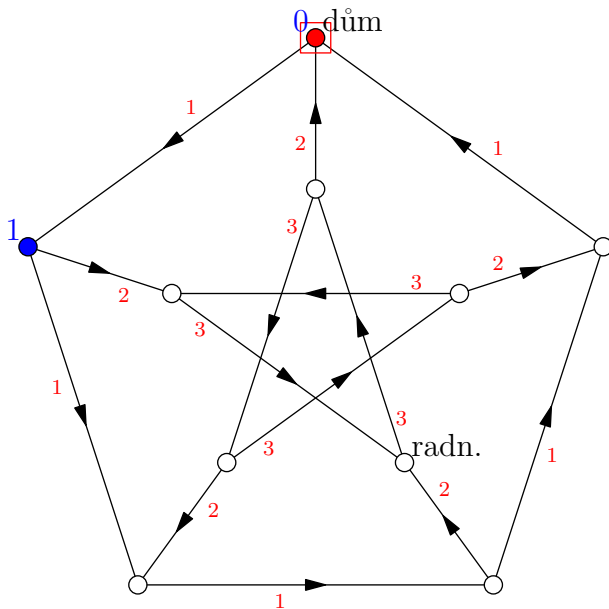
Dijkstrův algoritmus – ukázka



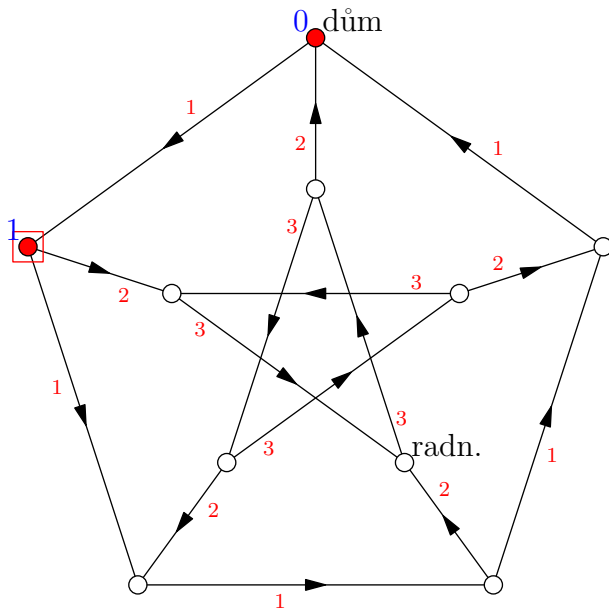
Dijkstrův algoritmus – ukázka



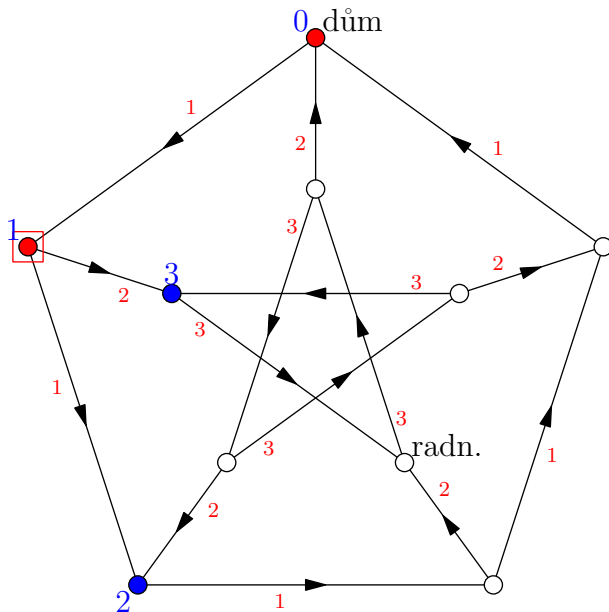
Dijkstrův algoritmus – ukázka



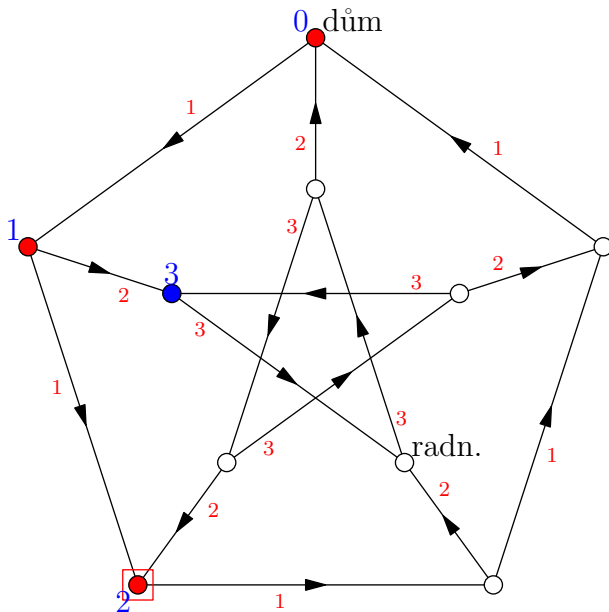
Dijkstrův algoritmus – ukázka



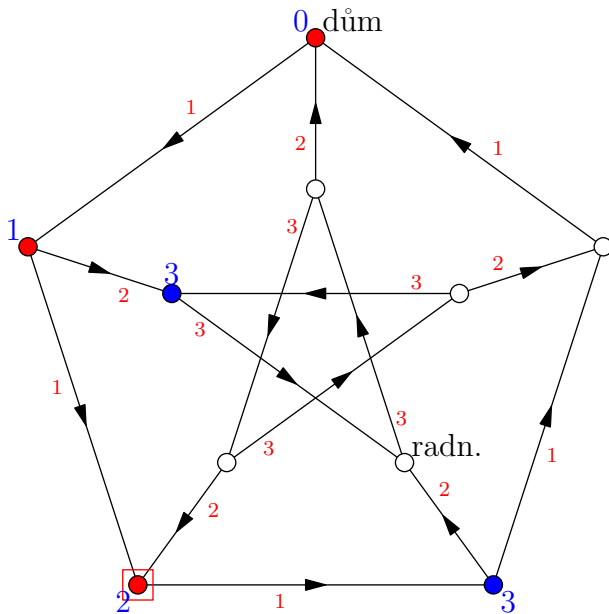
Dijkstrův algoritmus – ukázka



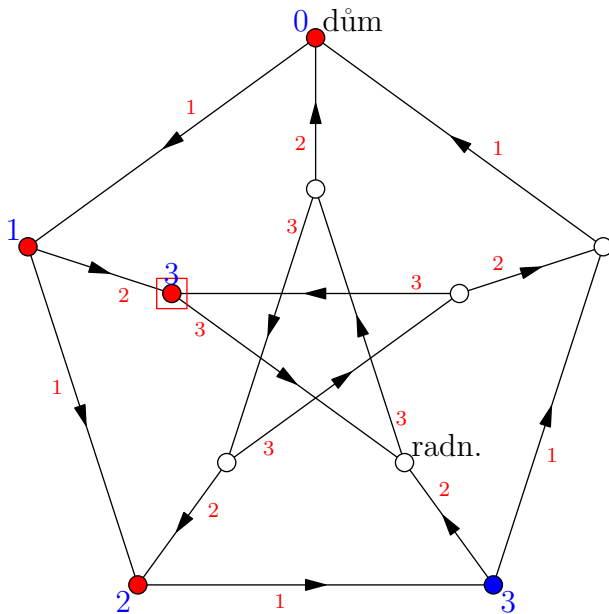
Dijkstrův algoritmus – ukázka



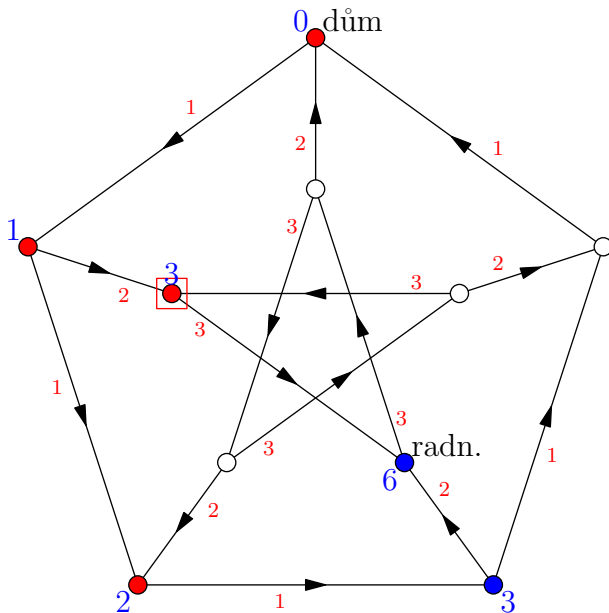
Dijkstrův algoritmus – ukázka



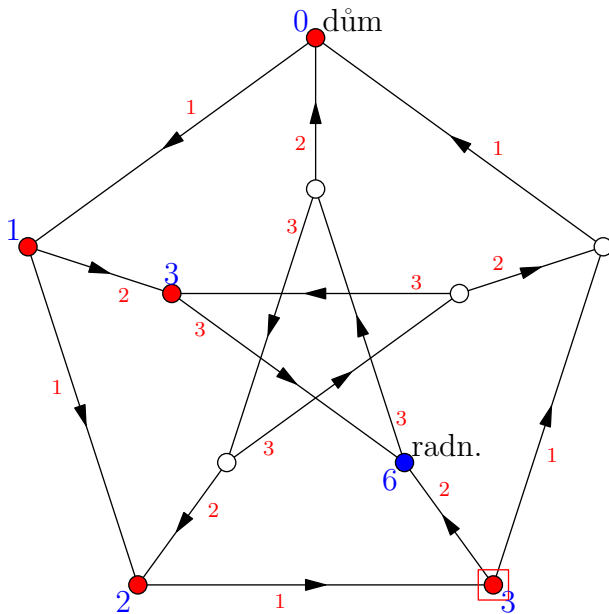
Dijkstrův algoritmus – ukázka



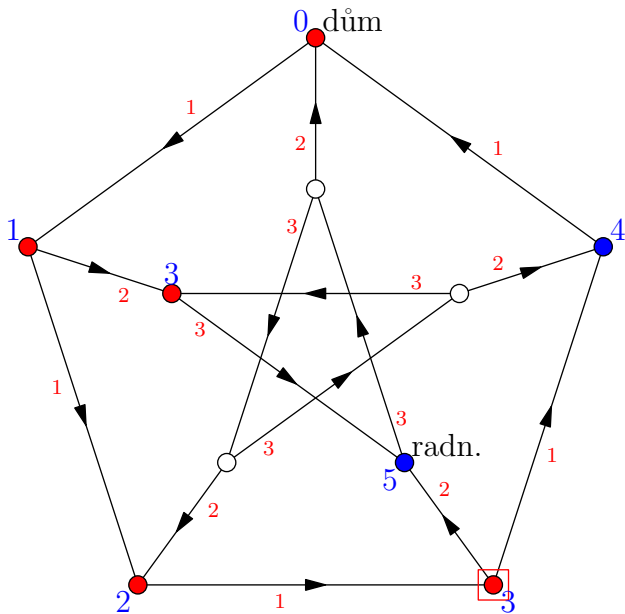
Dijkstrův algoritmus – ukázka



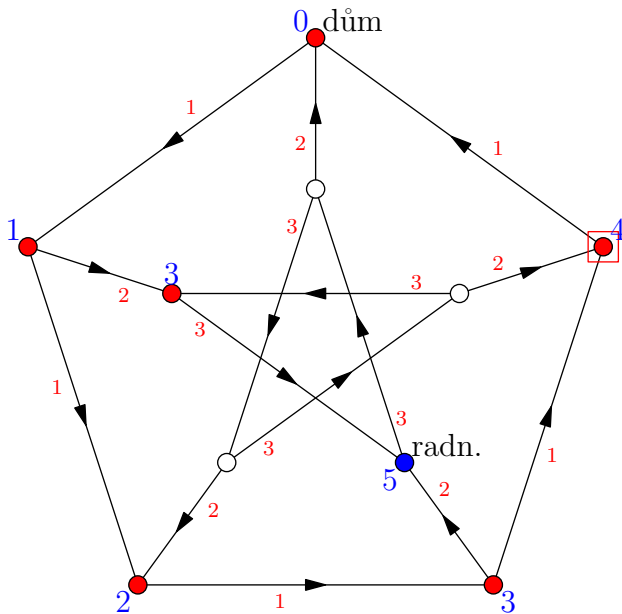
Dijkstrův algoritmus – ukázka



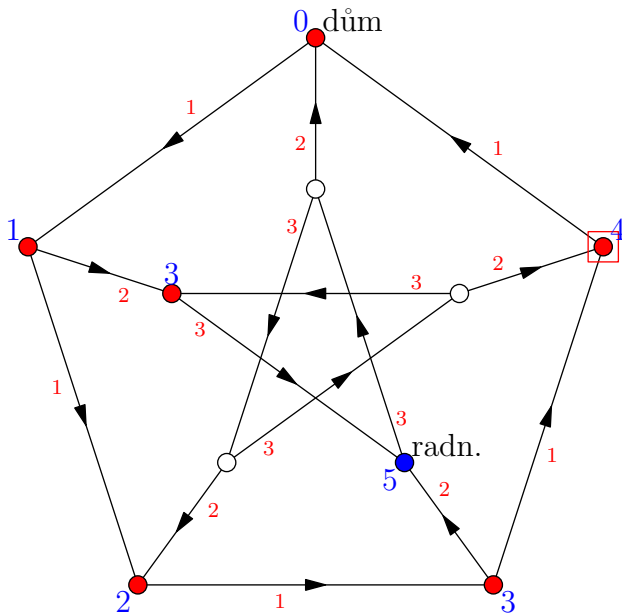
Dijkstrův algoritmus – ukázka



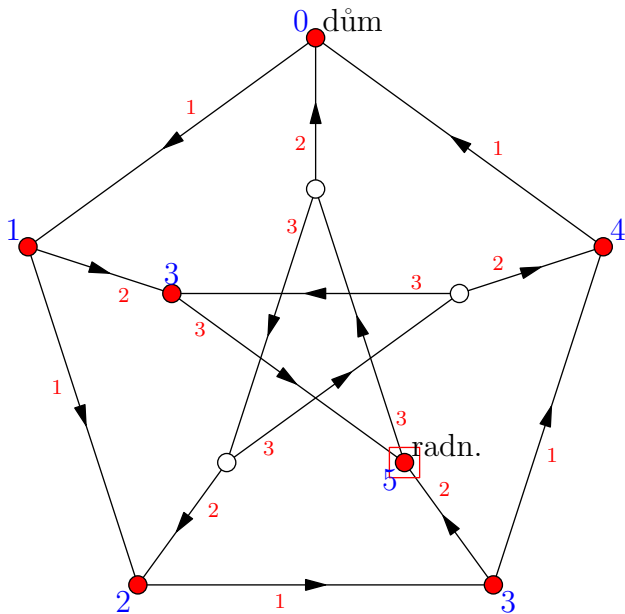
Dijkstrův algoritmus – ukázka



Dijkstrův algoritmus – ukázka



Dijkstrův algoritmus – ukázka



Seznam sousedů pro každý vrchol:

```
struct hrana
{
    int kam;
    int delka;
}
```

```
vector<hrana> graf[N];
```

Jiná možnost: matice sousednosti.

Dijkstrův algoritmus

```
vector<int> vzdal(N, INT_MAX); vzdal[odkud] = 0;
vector<bool> jisty(N, false);
int nejblizsi_nejisty (void) {
    int best = INT_MAX, best_v = -1;
    for (int v = 0; v < N; v++)
        if (!jisty[v] && vzdal[v] < best)
            { best = vzdal[v]; best_v = v; }
    return best_v;
}
while ((v = nejblizsi_nejisty ()) >= 0) {
    int aktualni = vzdal[v];
    if (v == cil) { cout << aktualni << endl; exit
    jisty[v] = true;
    for (hrana &h : graf[v])
        if (vzdal[h.kam] > aktualni + h.delka)
            vzdal[h.kam] = aktualni + h.delka;
}
```

n vrcholů, e hran:

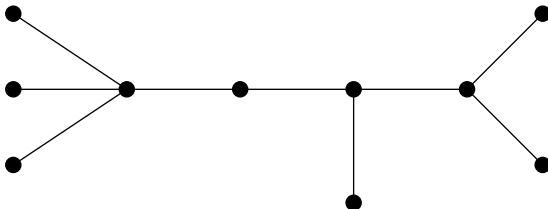
- $n \times$ hledání minima: $O(n)$
- $e \times$ vylepšení přes hranu: $O(1)$

Celkem $O(n^2 + e) = O(n^2)$.

Jiná varianta:

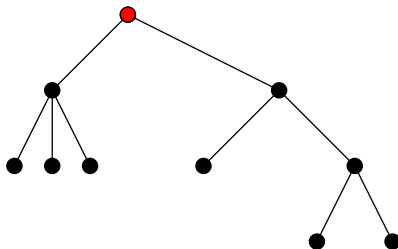
- ksp.mff.cuni.cz/encyklopedie/halda-a-cesty
- Složitost $O((n + e) \cdot \log n)$.

Souvislé grafy bez cyklů.



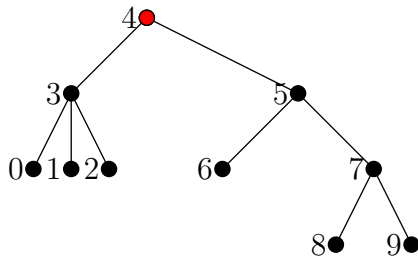
Reprezentace:

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



Reprezentace stromu

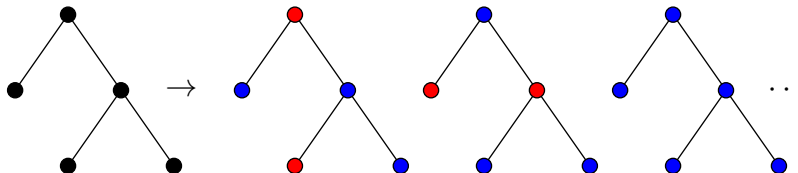
	Rodič	Děti
0	3	0,1,2
1	3	
2	3	
3	4	3,5
4	⊥	
5	4	6,7
6	5	
7	5	8,9
8	7	
9	7	



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.

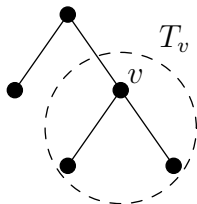
Pro zadaný strom spočítejte počet obarvení jeho vrcholů červeně a modře takových, že žádné dva červené vrcholy nesousedí.



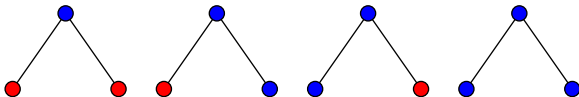
Pro každý vrchol v a podstrom T_v tvořený vrcholy pod v určíme

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

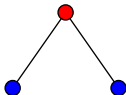
Výsledek: $m(\text{kořen}) + c(\text{kořen})$.



$$m(v) = 4$$



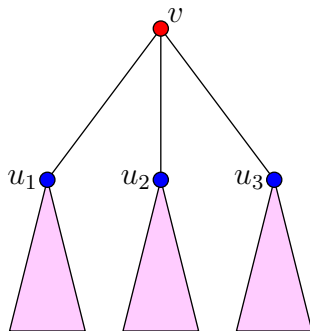
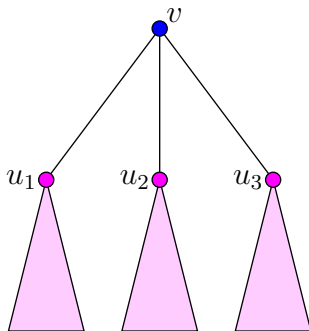
$$c(v) = 1$$



Výpočet $m(v)$ a $c(v)$

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

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



```

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: $O(\text{počet vrcholů})$.

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