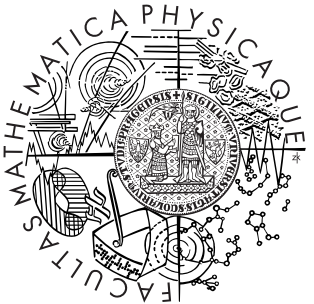
Algoritmy a datové struktury 1

2/2 Z+Zk, NTIN060

pomocné slajdy ke grafovým algoritmům

Pavel Veselý (IUUK)



vesely+ads1@iuuk.mff.cuni.cz

https://iuuk.mff.cuni.cz/∼vesely/vyuka/ls24/ads1.html

Prohledávání do hloubky (DFS) i pro nesouvislé grafy — pseudokód

```
function DFSMAIN(G = (V, E))
    ∀v ∈ V : stav(v) ← (N)    ▷ Výchozí stav je (N)=nenalezený

    for all v ∈ V : stav(v) = (N) do ▷ projdeme i nesouvislý graf
        DFS-REC(v)
    end for
end function
function DFS-REC(v)
    stav(v) ← (O)                          ▷ Otevřeme vrchol

    for all ∀vw ∈ E : do
        if stav(w) = (N) then
            DFS-REC(w)
        end if
    end for
    stav(v) ← (Z)                          ▷ Zavřeme vrchol

end function
```

Prohledávání do hloubky (DFS) i pro nesouvislé grafy — pseudokód

```
function DFSMain(G = (V, E))
    ∀v ∈ V : stav(v) ← (N)    ▷ Výchozí stav je (N)=nenalezený
    ∀v ∈ V : in(v), out(v) ← ∞
    T ← 0
    for all v ∈ V : stav(v) = (N) do ▷ projdeme i nesouvislý graf
        DFS-rec(v)
    end for
end function
function DFS-rec(v)
    stav(v) ← (O)                          ▷ Otevřeme vrchol
    T ← T + 1, in(v) ← T
    for all ∀vw ∈ E : do
        if stav(w) = (N) then
            DFS-rec(w)
        end if
    end for
    stav(v) ← (Z)                          ▷ Zavřeme vrchol
    T ← T + 1, out(v) ← T
end function
```

Mosty: opakování

Hrana $e \in G$ je most, pokud $G - e$ má více komponent než G

Mosty: opakování

Hrana $e \in G$ je most, pokud $G - e$ má více komponent než G

- Lemma: e *není* most právě tehdy, když e leží na kružnici
- Pozorování: zpětné hrany nemohou být mosty

Mosty: opakování

Hrana $e \in G$ je most, pokud $G - e$ má více komponent než G

- Lemma: e *není* most právě tehdy, když e leží na kružnici
- Pozorování: zpětné hrany nemohou být mosty

Artikulace

Vrchol $v \in G$ je artikulace, pokud $G - v$ má více komponent než G

Mosty: opakování

Hrana $e \in G$ je most, pokud $G - e$ má více komponent než G

- Lemma: e *není* most právě tehdy, když e leží na kružnici
- Pozorování: zpětné hrany nemohou být mosty

Artikulace

Vrchol $v \in G$ je artikulace, pokud $G - v$ má více komponent než G

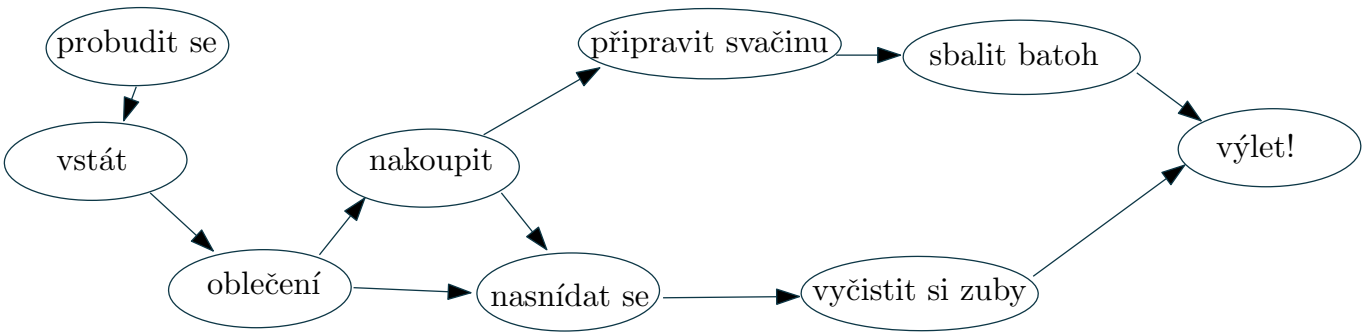
- Lemma: v *není* artikulace právě tehdy, když pro každé dva jeho různé sousedy x a y existuje kružnice obsahující hrany vx a vy

Acyklické orientované grafy

Directed acyclic graphs (**DAG**)

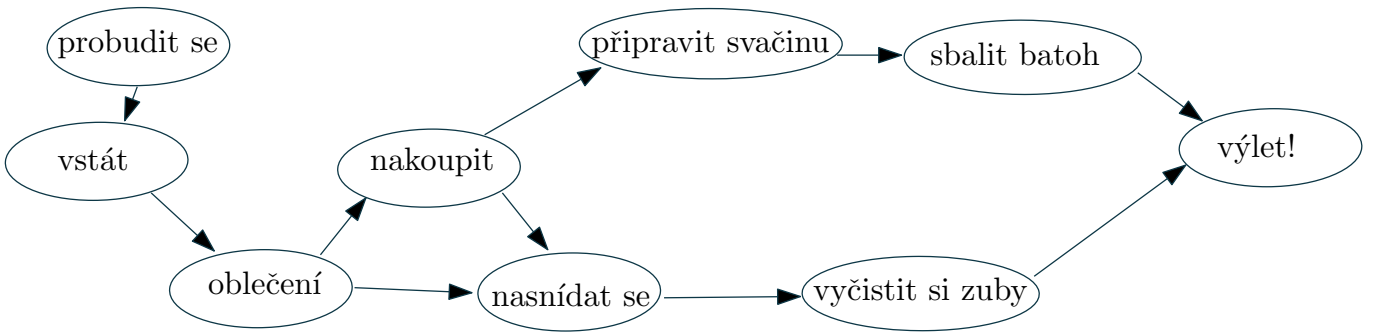
Acyklické orientované grafy

Directed acyclic graphs (**DAG**)



Acyklické orientované grafy

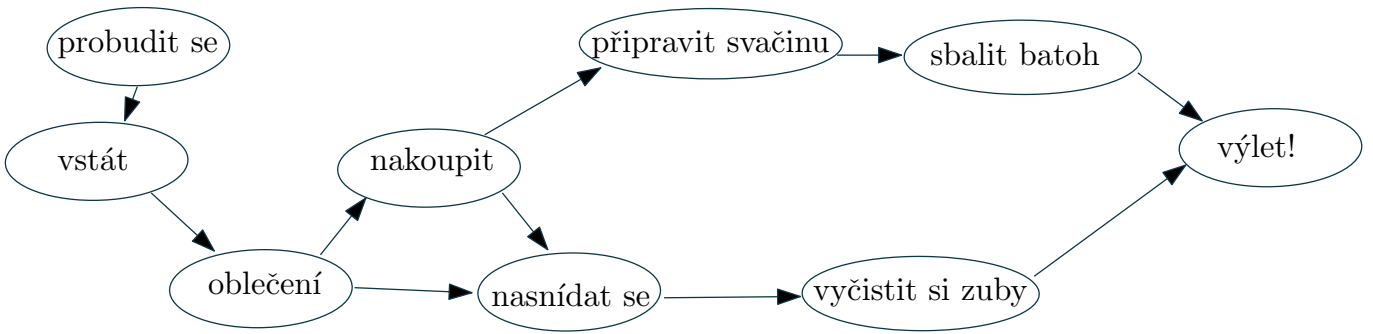
Directed acyclic graphs (**DAG**)



- Plánování pomocí topologického pořadí (uspořádání)

Acyklické orientované grafy

Directed acyclic graphs (**DAG**)



- Plánování pomocí topologického pořadí (uspořádání)
- Reálná aplikace: Make a Snakemake

Hledání komponent silné souvislosti (KSS)

Algoritmus Kosaraju-Sharir ('78 a '81)

Hledání komponent silné souvislosti (KSS)

Algoritmus Kosaraju-Sharir ('78 a '81)

1. pokus o algoritmus:

function $\text{KSSNAIVE}(G = (V, E))$
  $\forall v \in V$: komp(v) $\leftarrow$ nedef.
  while G není prázdný **do**
    $\text{DFS}(G^T)$
    $w \leftarrow$ vrchol s max. outem v G^T $\triangleright$ w leží ve zdrojové KSS v G^T, tedy stokové pro G
    $\text{DFS-RECURSIVE}(G, w)$
      projde jen KSS, v níž leží w; navštíveným vrcholům v přiřadíme komp(v)$\leftarrow w$
    Smažeme z G vrcholy KSS, v níž leží w
  end while
end function

Jakou má časovou složitost?

Hledání komponent silné souvislosti (KSS)

Algoritmus Kosaraju-Sharir ('78 a '81)

1. pokus o algoritmus:

function $\mathrm{KSS}_{\mathrm{NAIVE}}(G = (V, E))$
  $\forall v \in V$: komp(v) $\leftarrow$ nedef.
  while G není prázdný **do**
    $\mathrm{DFS}(G^T)$
    $w \leftarrow$ vrchol s max. outem v G^T $\triangleright$ w leží ve zdrojové KSS v G^T, tedy stokové pro G
    $\mathrm{DFS\text{-}RECURSIVE}(G, w)$
      projde jen KSS, v níž leží w; navštíveným vrcholům v přiřadíme komp(v)$\leftarrow w$
    Smažeme z G vrcholy KSS, v níž leží w
  end while
end function

Jakou má časovou složitost?

Pojďme to zlepšit na lineární složitost!

Hledání komponent silné souvislosti (KSS)

Tarjanův algoritmus ('72)

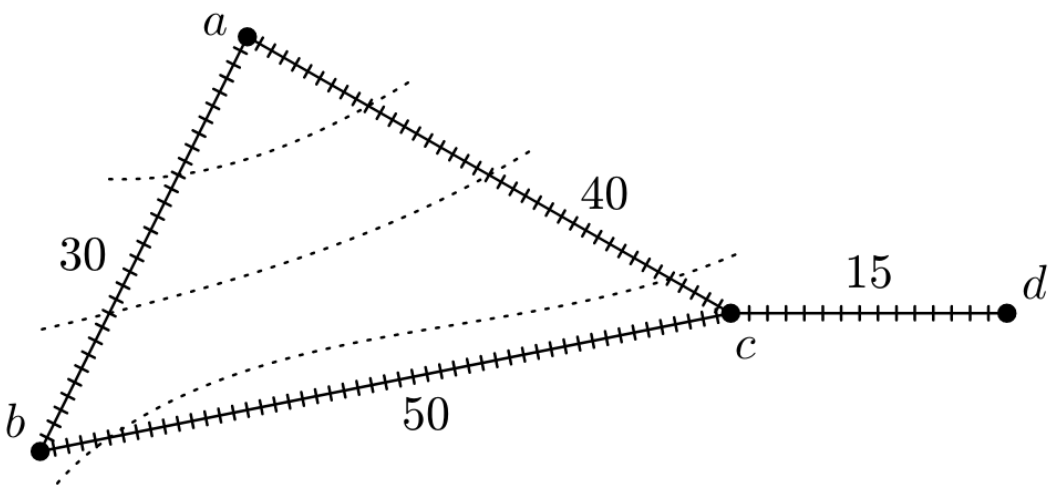
- jen jedno spuštění DFS
- Lemma: každá KSS indukuje v DFS stromu *slabě souvislý* podgraf
  - (důkaz v učebnici)

Hledání nejkratších cest z v_0 pro nezáporné délky

Dijkstrův algoritmus ('56)

Hledání nejkratších cest z v_0 pro nezáporné délky

Dijkstrův algoritmus ('56)



Hledání nejkratších cest z v_0 pro nezáporné délky

Dijkstrův algoritmus ('56)

function DIJKSTRA($G = (V, E)$, délky hran $\ell : E \to \mathbb{R}_0^+$, počáteční vrchol v_0)
  $\forall v \in V$: stav(v) $\leftarrow$ (N) ▷ Výchozí stav je (N)=nenalezený
  $\forall v \in V$: $h(v) \leftarrow \infty$ ▷ Ohodnocení („budík"); na konci vzdálenost z v_0
  stav(v_0) $\leftarrow$ (O), $h(v_0) \leftarrow 0$
  while existuje otevřený vrchol **do**
    $v \leftarrow$ otevřený vrchol s nejmenším $h(v)$ ▷ ExtractMin
    for all $vw \in E$ **do**
      if $h(w) > h(v) + \ell(vw)$ **then** ▷ Relaxace hrany
        $h(w) \leftarrow h(v) + \ell(vw)$ ▷ Insert nebo DescreaseKey
        stav(w) $\leftarrow$ (O)
      end if
    end for
    stav(v) $\leftarrow$ (Z) ▷ Zavíráme v
  end while
end function

Hledání nejkratších cest z v_0 pro nezáporné délky

Dijkstrův algoritmus ('56)

function DIJKSTRA($G = (V, E)$, délky hran $\ell : E \to \mathbb{R}_0^+$, počáteční vrchol v_0)
  $\forall v \in V$: stav(v) $\leftarrow$ (N) ▷ Výchozí stav je (N)=nenalezený
  $\forall v \in V$: $h(v) \leftarrow \infty$ ▷ Ohodnocení („budík"); na konci vzdálenost z v_0
  stav(v_0) $\leftarrow$ (O), $h(v_0) \leftarrow 0$
  while existuje otevřený vrchol **do**
    $v \leftarrow$ otevřený vrchol s nejmenším $h(v)$ ▷ ExtractMin
    for all $vw \in E$ **do**
      if $h(w) > h(v) + \ell(vw)$ **then** ▷ Relaxace hrany
        $h(w) \leftarrow h(v) + \ell(vw)$ ▷ Insert nebo DescreaseKey
        stav(w) $\leftarrow$ (O)
      end if
    end for
    stav(v) $\leftarrow$ (Z) ▷ Zavíráme v
  end while
end function

TODO list:

1. **časová složitost** (za předpokladu, že každý vrchol uzavřen jen jednou)
2. **správnost** pro libovolné nezáporné délky hran
3. co s grafy se **zápornými délkami**, ale bez záporných cyklů?

Hledání nejkratších cest z v_0: relaxační meta-algoritmus

function RelaxMetaAlg($G = (V, E)$, délky hran $\ell : E \to \mathbb{R}$, počáteční vrchol v_0)
  $\forall v \in V$: stav$(v) \leftarrow$ (N) ▷ Výchozí stav je (N)=nenalezený
  $\forall v \in V$: $h(v) \leftarrow \infty$ ▷ Ohodnocení („budík“); na konci vzdálenost z v_0
  stav$(v_0) \leftarrow$ (O), $h(v_0) \leftarrow 0$
  while existuje otevřený vrchol **do**
    $v \leftarrow$ **nějaký** otevřený vrchol
    for all $vw \in E$ **do**
      if $h(w) > h(v) + \ell(vw)$ **then** ▷ Relaxace hrany
        $h(w) \leftarrow h(v) + \ell(vw)$
        stav$(w) \leftarrow$ (O)
      end if
    end for
    stav$(v) \leftarrow$ (Z) ▷ Zavíráme v
  end while
end function

Hledání nejkratších cest z v_0

Bellmanův-Fordův algoritmus ('58 a '56; Shimbel '55)

- pro grafy bez záporných cyklů (ale záporné hrany jinak nevadí)

Hledání nejkratších cest z v_0

Bellmanův-Fordův algoritmus ('58 a '56; Shimbel '55)

- pro grafy bez záporných cyklů (ale záporné hrany jinak nevadí)

function BellmanFordAlg($G = (V, E)$, délky hran $\ell : E \to \mathbb{R}$, počáteční vrchol v_0)
  $\forall v \in V$: stav(v) $\leftarrow$ (N) ▷ Výchozí stav je (N)=nenalezený
  $\forall v \in V$: $h(v) \leftarrow \infty$ ▷ Ohodnocení („budík"); na konci vzdálenost z v_0
  stav(v_0) $\leftarrow$ (O), $h(v_0) \leftarrow 0$
  while existuje otevřený vrchol **do**
    $v \leftarrow$ **nejdéle** otevřený vrchol
    for all $vw \in E$ **do**
      if $h(w) > h(v) + \ell(vw)$ **then** ▷ Relaxace hrany
        $h(w) \leftarrow h(v) + \ell(vw)$
        stav(w) $\leftarrow$ (O)
      end if
    end for
    stav(v) $\leftarrow$ (Z) ▷ Zavíráme v
  end while
end function

Hledání nejkratších cest z v_0

Bellmanův-Fordův algoritmus ('58 a '56; Shimbel '55)

- pro grafy bez záporných cyklů (ale záporné hrany jinak nevadí)
- Časová složitost $O(n \cdot m)$ — umí se lepší algoritmus?

Hledání nejkratších cest z v_0

Bellmanův-Fordův algoritmus ('58 a '56; Shimbel '55)

- pro grafy bez záporných cyklů (ale záporné hrany jinak nevadí)
- Časová složitost $O(n \cdot m)$ — umí se lepší algoritmus? Ano! (pro celočíselné délky)
  - Článek z konference *FOCS 2022*:

Negative-Weight Single-Source Shortest Paths in Near-linear Time

Aaron Bernstein* Danupon Nanongkai† Christian Wulff-Nilsen‡

Abstract

We present a randomized algorithm that computes single-source shortest paths (SSSP) in $O(m \log^8(n) \log W)$ time when edge weights are integral and can be negative.[1] This essentially resolves the classic negative-weight SSSP problem. The previous bounds are $\tilde{O}((m+n^{1.5}) \log W)$ [BLNPSSSW FOCS'20] and $m^{4/3+o(1)} \log W$ [AMV FOCS'20]. Near-linear time algorithms were known previously only for the special case of planar directed graphs [Fakcharoenphol and Rao FOCS'01].

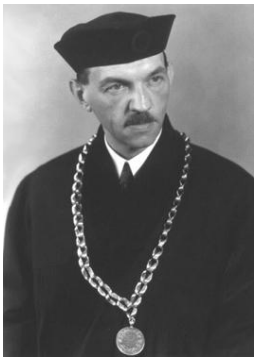
In contrast to all recent developments that rely on sophisticated continuous optimization methods and dynamic algorithms, our algorithm is simple: it requires only a simple graph decomposition and elementary combinatorial tools. In fact, ours is the first combinatorial algorithm for negative-weight SSSP to break through the classic $\tilde{O}(m\sqrt{n} \log W)$ bound from over three decades ago [Gabow and Tarjan SICOMP'89].

Hledání nejkratších cest mezi každou dvojicí vrcholů

Floydův-Warshallův algoritmus ('62; Roy '59)

- pro grafy bez záporných cyklů (ale záporné hrany jinak nevadí)
- bude v rámci dynamického programování

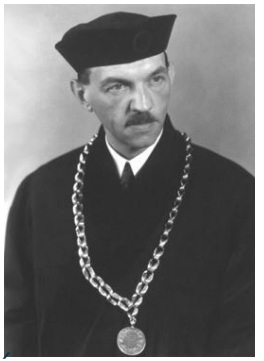
Hledání minimální kostry — 3 základní hladové algoritmy



1. Jarníkův algoritmus [Jarník '30, Prim '57, Dijkstra '59]

- staví min. kostru z jednoho vrcholu

Hledání minimální kostry — 3 základní hladové algoritmy

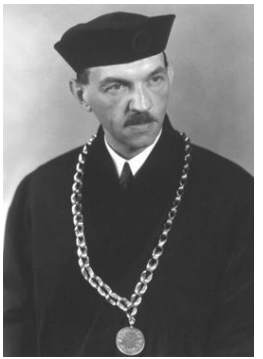


1. Jarníkův algoritmus [Jarník '30, Prim '57, Dijkstra '59]

- staví min. kostru z jednoho vrcholu
- Implementace Jarníkova algoritmu à la Dijkstra

function JARNÍK($G = (V, E)$, váhy hran $w : E \to \mathbb{R}$)
  $v_0 \leftarrow$ libovolný vrchol
  $\forall v \in V$: stav$(v) \leftarrow$ (N) ▷ Výchozí stav je (N)=nenalezený
  $\forall v \in V : h(v) \leftarrow \infty, p(v) \leftarrow$ nedef. ▷ Ohodnocení a předchůdce
  stav$(v_0) \leftarrow$ (O), $h(v_0) \leftarrow 0$
  $T \leftarrow (\{v_0\}, \emptyset)$ ▷ strom jen v_0 bez hran
  while existuje otevřený vrchol **do**
    $v \leftarrow$ otevřený vrchol s nejmenším $h(v)$ ▷ ExtractMin
    Přijde v do T a pokud je $p(v)$ definováno, pak i hranu $\{v, p(v)\}$
    for all $vu \in E$ **do**
      if $h(u) > w(vu)$ a stav$(u) \neq$ (Z) **then**
        $h(u) \leftarrow w(vu)$ ▷ Insert nebo DescreaseKey
        $p(u) \leftarrow v$ ▷ předchůdce
        stav$(u) \leftarrow$ (O)
      end if
    end for
    stav$(v) \leftarrow$ (Z) ▷ Zavíráme v
  end while

Hledání minimální kostry — 3 základní hladové algoritmy



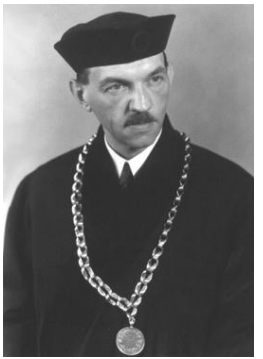
1. Jarníkův algoritmus [Jarník '30, Prim '57, Dijkstra '59]

- staví min. kostru z jednoho vrcholu

2. Borůvkův algoritmus [Borůvka '26], znovu objeven třikrát

- Borůvkova motivace: konstrukce efektivní elektrické sítě na Moravě
- staví min. kostru paralelně ze všech vrcholů
  - „paralelní Jarník“

Hledání minimální kostry — 3 základní hladové algoritmy



1. Jarníkův algoritmus [Jarník '30, Prim '57, Dijkstra '59]

- staví min. kostru z jednoho vrcholu

2. Borůvkův algoritmus [Borůvka '26], znovu objeven třikrát

- Borůvkova motivace: konstrukce efektivní elektrické sítě na Moravě
- staví min. kostru paralelně ze všech vrcholů
  - „paralelní Jarník“

3. Kruskalův algoritmus [Kruskal '56]

- přidává hrany od nejlehčí po nejtěžší, pokud přidání hrany nevytvoří cyklus

Minimální kostry — co se umí?

- Algoritmus s lineární časovou složitostí pro
  - celočíselné váhy
  - „husté“ grafy, stačí $m/n \geq \log\log\log n$

Minimální kostry — co se umí?

- Algoritmus s lineární časovou složitostí pro
  - celočíselné váhy
  - „husté“ grafy, stačí $m/n \geq \log\log\log n$

- Algoritmus s „průměrně“ lineární časovou složitostí [Karger, Klein, Tarjan '95]
  - Střední hodnota časové složitosti je $O(m)$

Minimální kostry — co se umí?

- Algoritmus s lineární časovou složitostí pro
  - celočíselné váhy
  - „husté“ grafy, stačí $m/n \geq \log\log\log n$
- Algoritmus s „průměrně“ lineární časovou složitostí [Karger, Klein, Tarjan '95]
  - Střední hodnota časové složitosti je $O(m)$
- Optimální deterministický algoritmus v porovnávacím modelu
  - [Pettie & Ramachandran '02]
  - Neznáme však jeho časovou složitost!
  - Nejlepší horní odhad je $O(m\alpha(m, n))$ [Chazelle '00]
  - α je inverze Ackermannovy funkce
  - $\alpha(m, n) \leq 4$ pro všechny *praktické* hodnoty m, n