

KMI/ALGO3 — Algoritmy 3

1. Co znamená navrhnout algoritmus

Jan Konečný

Katedra informatiky, PřF UP Olomouc

Úvod

Když jsme postaveni před novou úlohu, neměli bychom rovnou otevřít editor a začít „bastlit“ řešení.

Tím se přeskočí dvě kriticky důležité otázky:

- co přesně má program vypočítat,
- proč by zvolený postup měl fungovat.

Dnešní přednášku strávíme nad jedním příkladem — [stabilním párováním](#).

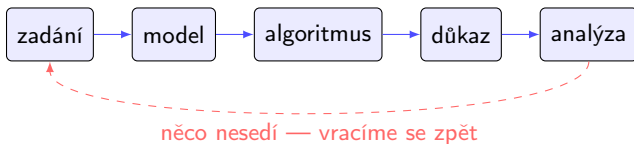
Nejde o to naučit se co nejvíc pojmenovaných algoritmů, ale projít si postup, který budeme opakovat u každé další metody.

Co všechno musíme udělat

Algoritmus je výsledek malého návrhového procesu.

Na jeho konci musíme mít prokázány tři věci:

1. algoritmus se **zastaví**;
2. vrátí výsledek, který **splňuje zadání**;
3. spotřebuje **příjemné** množství času a paměti.



Od neformální úlohy k výpočetnímu problému

Studenti si vybírají témata semestrálních projektů.

- Každý student má **ostré pořadí** (úplné uspořádání) všech projektů.
- Každý projekt (resp. jeho vedoucí) má **ostré pořadí** všech studentů.
- Každý projekt má právě jedno místo.

Co po algoritmu vlastně chceme?

Od neformální úlohy k výpočetnímu problému

Studenti si vybírají témata semestrálních projektů.

- Každý student má **ostré pořadí** (úplné uspořádání) všech projektů.
- Každý projekt (resp. jeho vedoucí) má **ostré pořadí** všech studentů.
- Každý projekt má právě jedno místo.

Co po algoritmu vlastně chceme?

- *Každý dostane první volbu?* — o některá témata je zájem několikanásobný.

Od neformální úlohy k výpočetnímu problému

Studenti si vybírají témata semestrálních projektů.

- Každý student má **ostré pořadí** (úplné uspořádání) všech projektů.
- Každý projekt (resp. jeho vedoucí) má **ostré pořadí** všech studentů.
- Každý projekt má právě jedno místo.

Co po algoritmu vlastně chceme?

- *Každý dostane první volbu?* — o některá témata je zájem několikanásobný.
- *Maximalizujeme součet bodů spokojenosti?* — do zadání vnášíme novou informaci, která tam dřív nebyla.

Od neformální úlohy k výpočetnímu problému

Studenti si vybírají témata semestrálních projektů.

- Každý student má **ostré pořadí** (úplné uspořádání) všech projektů.
- Každý projekt (resp. jeho vedoucí) má **ostré pořadí** všech studentů.
- Každý projekt má právě jedno místo.

Co po algoritmu vlastně chceme?

- *Každý dostane první volbu?* — o některá témata je zájem několikanásobný.
- *Maximalizujeme součet bodů spokojenosti?* — do zadání vnášíme novou informaci, která tam dřív nebyla.
- **Nechceme, aby existoval student a projekt, kteří by si oba polepšili, kdyby opustili přidělené partnery.**

Jedna neformální úloha připouští více různých matematických úloh/modelů.

Výpočetní problém

Definice 1 (Výpočetní problém)

Výpočetní problém je určen

- množinou přípustných **vstupů** (též **instancí**);
- pro každou instanci množinou výstupů, které považujeme za **správné**.

Algoritmus *řeší* výpočetní problém, jestliže pro každou přípustnou instanci skončí a vrátí jeden ze správných výstupů.

Tj.

- na jaká data musí být algoritmus připraven,
- co od něj požadujeme jako výstup.

Jedna *instance* je jeden konkrétní vstup.

Tři druhy odpovědí

- **Rozhodovací problém** — výstupem je pouze ANO, nebo NE.
(Je graf souvislý? Je formule splnitelná?)
- **Vyhledávací problém** — chceme nalézt objekt s danými vlastnostmi.
(Najdi stabilní párování; najdi ohodnocení, ve kterém je formule pravdivá.)
- **Optimalizační problém** — vybíráme z přípustných řešení to nejlepší podle účelové funkce.
(Najdi nejkratší cestu, minimální kostru.)

Varování

Optimalizační úlohy budou v kurzu časté, ale nejsou jediným druhem problémů. Stabilní párování hledá párování s určitou *strukturální* vlastností — ne párování s největším součtem předem daných bodů.

Algoritmus a korektnost

Algoritmus je konečný a jednoznačný popis výpočtu.

Není svázán s jazykem; zápis však musí být tak přesný, aby bylo jasné, který krok se má provést *v každé možné situaci*.

„Funguje mi to na mém příkladu“ není důkaz.

Definice 2 (Korektnost)

Algoritmus je **částečně korektní**, jestliže pokaždé, když skončí, vrátí správný výstup.

Algoritmus je **(úplně) korektní**, jestliže je částečně korektní a navíc pro každý přípustný vstup po konečně mnoha krocích skončí.

Invariant — hlavní nástroj důkazů

Definice 3 (Invariant)

Invariant algoritmu je tvrzení, které platí **před první iterací** zvoleného opakovaného kroku a jehož platnost **každá iterace zachovává**.

Invariant vymezuje, v jakých stavech se výpočet smí pohybovat: zaručuje, že se algoritmus nedostane do stavu, z něhož už správný výsledek nelze získat.

Typický důkaz invariantu má tři části

1. **Základ:** platí na začátku.
2. **Krok:** jedna iterace platnost zachová.
3. **Závěr:** invariant + podmínka ukončení $\Rightarrow$ požadovaná vlastnost výstupu.

Formalizace

Množina studentů S a množina projektů P , $|S| = |P| = n$.
Každý student i projekt má **ostré** pořadí celé druhé strany.

Definice 4 (Párování)

Párování M mezi S a P je množina dvojic $(s, p) \in S \times P$ taková, že žádný student ani projekt není ve více než jedné dvojici.

Je-li každý student i projekt právě v jedné dvojici, je párování **úplné**.
Partnera značíme $M(s)$, resp. $M(p)$.

Samotné úplné párování ale ještě není přesvědčivý výsledek.

Malá instance

student	1.	2.	3.	projekt	1.	2.	3.
<i>a</i>	<i>x</i>	<i>y</i>	<i>z</i>	<i>x</i>	<i>b</i>	<i>a</i>	<i>c</i>
<i>b</i>	<i>y</i>	<i>x</i>	<i>z</i>	<i>y</i>	<i>a</i>	<i>b</i>	<i>c</i>
<i>c</i>	<i>y</i>	<i>z</i>	<i>x</i>	<i>z</i>	<i>a</i>	<i>c</i>	<i>b</i>

Uvažme párování

$$M_0 = \{(a, z), (b, x), (c, y)\}.$$

Malá instance

student	1.	2.	3.	projekt	1.	2.	3.
<i>a</i>	<i>x</i>	<i>y</i>	<i>z</i>	<i>x</i>	<i>b</i>	<i>a</i>	<i>c</i>
<i>b</i>	<i>y</i>	<i>x</i>	<i>z</i>	<i>y</i>	<i>a</i>	<i>b</i>	<i>c</i>
<i>c</i>	<i>y</i>	<i>z</i>	<i>x</i>	<i>z</i>	<i>a</i>	<i>c</i>	<i>b</i>

Uvažme párování

$$M_0 = \{(a, z), (b, x), (c, y)\}.$$

- Student *a* by **raději** řešil *y* než přidělené *z*.
- Projekt *y* by **raději** přijal *a* než přiděleného *c*.

Dvojice (a, y) má dobrý důvod přidělení obejít.
Právě tomu chceme zabránit.

Destabilizující dvojice a stabilita

Definice 5 (Destabilizující dvojice, stabilita)

Nechť M je úplné párování. Dvojice (s, p) , která v M *není*, je **destabilizující**, jestliže

- student s dává projektu p přednost před $M(s)$ a *současně*
- projekt p dává studentu s přednost před $M(p)$.

Úplné párování je **stabilní**, jestliže nemá žádnou destabilizující dvojici.

Stabilita **neříká**, že

- všichni dostanou svou první volbu;
- jsme maximalizovali celkovou spokojenost.

Říká přesně jednu věc: *neexistuje dvojice, která by se navzájem preferovala před přidělenými partnery.*

Existuje stabilní párování vždycky?

Odpověď: **Gale** a **Shapley** (1962) konstruktivně — popsáním algoritmu.



D. Gale, L. S. Shapley.

College Admissions and the Stability of Marriage.

American Mathematical Monthly 69 (1962), 9–15.

Budeme tomu říkat **algoritmus odloženého přijetí**.

Slovo „odloženého“ je důležité: projekt může mít v daném okamžiku nejlepší dosavadní nabídku, ale až do konce si ponechává možnost vyměnit ji za lepší.

Algoritmus odloženého přijetí

Na začátku jsou všichni volní a nikdo nikomu nenavrhá spojení.

Dokud existuje volný student s , který ještě nedal návrh *všem* projektům:

1. s navrhne nejvýše postavenému projektu p , kterému ještě návrh nedal;
2. je-li p volný, **předběžně** přijme s ;
3. je-li p předběžně spojen se studentem s' , ponechá si toho z dvojice s, s' , kterému dává přednost, a druhého odmítne.

Na konci se všechna předběžná spojení prohlásí za výsledné párování.

Algoritmus odloženého přijetí

Na začátku jsou všichni volní a nikdo nikomu nenavrhl spojení.

Dokud existuje volný student s , který ještě nedal návrh *všem* projektům:

1. s navrhne nejvýše postavenému projektu p , kterému ještě návrh nedal;
2. je-li p volný, **předběžně** přijme s ;
3. je-li p předběžně spojen se studentem s' , ponechá si toho z dvojice s, s' , kterému dává přednost, a druhého odmítne.

Na konci se všechna předběžná spojení prohlásí za výsledné párování.

Pozor: student nezačíná znovu od začátku seznamu — posouvá se v něm jen dopředu (dává návrhy jen těm projektům, kterým ještě nenavrhoval)

Pseudokód

```
1 def GaleShapley( $S, P, preference$ ):  
2     všichni studenti i projekty jsou volní  
3     while existuje volný student  $s$  s neprázdným zbytkem seznamu do  
4          $p \leftarrow$  nejvýše postavený projekt, kterému  $s$  ještě nenavrl  
5         posuň ukazatel studenta  $s$  o jednu pozici  
6         if  $p$  je volný then  
7             spoj předběžně ( $s, p$ )  
8         else  
9              $s' \leftarrow$  současný předběžný partner projektu  $p$   
10            if  $p$  preferuje  $s$  před  $s'$  then  
11                zruš ( $s', p$ );  $s'$  je opět volný  
12                spoj předběžně ( $s, p$ )  
13            else  
14                 $s$  zůstává volný                                //  $p$  návrh odmítl  
15 return množina předběžných spojení
```

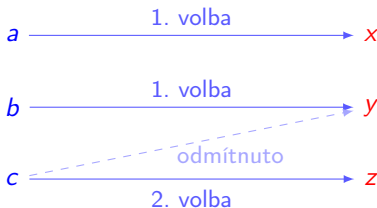
Běh na naší instanci

student	1.	2.	3.	projekt	1.	2.	3.
<i>a</i>	<i>x</i>	<i>y</i>	<i>z</i>	<i>x</i>	<i>b</i>	<i>a</i>	<i>c</i>
<i>b</i>	<i>y</i>	<i>x</i>	<i>z</i>	<i>y</i>	<i>a</i>	<i>b</i>	<i>c</i>
<i>c</i>	<i>y</i>	<i>z</i>	<i>x</i>	<i>z</i>	<i>a</i>	<i>c</i>	<i>b</i>

$a \rightarrow x, \quad b \rightarrow y, \quad c \rightarrow y.$

Projekt *y* preferuje *b* před *c*, takže si ponechá *b* a *c* odmítne.
Student *c* pokračuje druhou volbou *z*, která je volná.

Běh na naší instanci



$$M = \{(a, x), (b, y), (c, z)\}$$

Student c by sice raději řešil y , ale y dává přednost studentu b .
Dvojice (c, y) tedy není *destabilizující*.

První invariant

Lemma 6 (Projekt už nezůstane volný)

Jakmile projekt poprvé obdrží návrh, je od tohoto okamžiku až do konce algoritmu předběžně spojen s nějakým studentem.

Důkaz.

Volný projekt první návrh přijme. Při každém dalším návrhu si buď ponechá dosavadního studenta, nebo jej *okamžitě* vymění za nového.

V žádném kroku tedy nezůstane bez studenta.

První invariant

Lemma 6 (Projekt už nezůstane volný)

Jakmile projekt poprvé obdrží návrh, je od tohoto okamžiku až do konce algoritmu předběžně spojen s nějakým studentem.

Důkaz.

Volný projekt první návrh přijme. Při každém dalším návrhu si buď ponechá dosavadního studenta, nebo jej *okamžitě* vymění za nového. V žádném kroku tedy nezůstane bez studenta.

Použijeme ještě druhý invariant:

Monotonie z pohledu projektu

Projekt mění předběžného partnera pouze za studenta, kterému dává *větší* přednost. Partner projektu se tedy může jen *zlepšovat*.

Zastavení a úplnost

Věta 7 (Zastavení a úplnost)

Algoritmus odloženého přijetí skončí nejvýše po n^2 návrzích a výsledkem je úplné párování.

Kroky 1 a 2.

Žádný návrh se neopakuje. Student navrhuje vždy nejvýše postavenému projektu, kterému ještě návrh neposlal. Každá dvojice (s, p) je proto návrhem nejvýše jednou.

Počet návrhů je konečný. Dvojic je $n \cdot n = n^2$. Po nejvýše n^2 návrzích už žádný nový návrh nemůže vzniknout, takže algoritmus skončí.

Zastavení a úplnost (dokončení)

Krok 3: nikdo nezůstane volný.

Předpokládejme pro spor, že po skončení zůstal volný student s .

Algoritmus by neskončil, kdyby s ještě mohl oslovit nějaký projekt — musel tedy navrhnout *všem* n projektům.

Podle lemmatu je pak každý projekt na konci spojen s nějakým studentem. Průběžná spojení navíc v každém okamžiku tvoří párování: student pošle nový návrh jen tehdy, když je volný, takže nikdy nedrží dvě spojení současně.

Různé projekty jsou proto spojeny s různými studenty. Máme n projektů, tedy n různých spojených studentů — spojení jsou *všichni*. Spor s tím, že s zůstal volný.

Stabilita

Věta 8 (Stabilita výsledku)

Párování vytvořené algoritmem odloženého přijetí je stabilní.

Důkaz.

Pro spor: Nechť M má destabilizující dvojici (s, p) ; student s tedy preferuje p před svým partnerem $M(s)$.

Studenti oslovují projekty od nejoblíbenějšího. Než tedy s navrhl projektu $M(s)$, **musel už dříve navrhnout projektu p** . Projekt p jej buď odmítl hned, nebo přijal a odmítl později.

Projekt odmítá studenta jen tehdy, když už má návrh od někoho lepšího — a jeho partner se od té chvíle už jen zlepšuje. Na konci proto p preferuje $M(p)$ před s .

To je spor s tím, že je (s, p) destabilizující.

Jak byl ten důkaz vedený

- Nezkoušeli jsme po skončení projít všechny dvojice a ověřovat je.
- Použili jsme **historii výpočtu**: kdyby student preferoval jiný projekt, musel ho oslovit *dříve*.
- Odmítnutí nese informaci, která destabilizující dvojici vylučuje.

Algoritmus tu spolu s výsledkem vytvořil i *důvod*, proč je výsledek správný. To je u dobře navržených algoritmů běžné a budeme toho využívat i dál.

Jak byl ten důkaz vedený

- Nezkoušeli jsme po skončení projít všechny dvojice a ověřovat je.
- Použili jsme **historii výpočtu**: kdyby student preferoval jiný projekt, musel ho oslovit *dříve*.
- Odmítnutí nese informaci, která destabilizující dvojici vylučuje.

Algoritmus tu spolu s výsledkem vytvořil i *důvod*, proč je výsledek správný. To je u dobře navržených algoritmů běžné a budeme toho využívat i dál.

Algoritmus není symetrický. Zde navrhuji studenti a projekty vybírají. Při obrácených rolích dostaneme opět stabilní párování, ale obecně jiné. Navrhující strana dostává mezi všemi stabilními párováními svůj *nejlepší* možný výsledek.

Od počtu návrhů k době běhu

Horní mez n^2 na počet návrhů sama o sobě dobu běhu neurčuje. Musíme vědět, **kolik stojí jeden návrh** — a to závisí na reprezentaci dat.

- U studenta: seznam projektů + **ukazatel** na první dosud neoslovený projekt.
⇒ nalezení dalšího projektu v čase $O(1)$.
- U projektu: předpočítaná tabulka

$$\text{rank}_p(s) = \text{pozice studenta } s \text{ v pořadí projektu } p.$$

⇒ porovnání dvou studentů = porovnání dvou čísel, $O(1)$.

Od počtu návrhů k době běhu

Horní mez n^2 na počet návrhů sama o sobě dobu běhu neurčuje. Musíme vědět, **kolik stojí jeden návrh** — a to závisí na reprezentaci dat.

- U studenta: seznam projektů + **ukazatel** na první dosud neoslovený projekt.
⇒ nalezení dalšího projektu v čase $O(1)$.
- U projektu: předpočítaná tabulka

$\text{rank}_p(s)$ = pozice studenta s v pořadí projektu p .

⇒ porovnání dvou studentů = porovnání dvou čísel, $O(1)$.

Bez tabulky by jedno porovnání stálo $O(n)$ a celek $O(n^3)$.

Připomenutí: asymptotická notace

Definice 9 (O , Ω , Θ)

Nechť f, g přiřazují přirozeným číslům nezáporná reálná čísla.

- $f(n) = O(g(n))$, existují-li $c > 0$, n_0 tak, že $f(n) \leq c \cdot g(n)$ pro všechna $n \geq n_0$;
- $f(n) = \Omega(g(n))$, existují-li $c > 0$, n_0 tak, že $f(n) \geq c \cdot g(n)$ pro všechna $n \geq n_0$;
- platí-li obojí (třeba s různými konstantami), píšeme $f(n) = \Theta(g(n))$.

Připomenutí: asymptotická notace

Definice 9 (O , Ω , Θ)

Nechť f, g přiřazují přirozeným číslům nezáporná reálná čísla.

- $f(n) = O(g(n))$, existují-li $c > 0$, n_0 tak, že $f(n) \leq c \cdot g(n)$ pro všechna $n \geq n_0$;
- $f(n) = \Omega(g(n))$, existují-li $c > 0$, n_0 tak, že $f(n) \geq c \cdot g(n)$ pro všechna $n \geq n_0$;
- platí-li obojí (třeba s různými konstantami), píšeme $f(n) = \Theta(g(n))$.

Příklad. $f(n) = 3n^2 + 5n$. Pro $n \geq 5$ je $5n \leq n^2$, tedy $f(n) \leq 4n^2$ — volíme $c = 4$, $n_0 = 5$, takže $f(n) = O(n^2)$. Zároveň $f(n) \geq 3n^2$ vždy, tedy $f(n) = \Omega(n^2)$, dohromady $f(n) = \Theta(n^2)$.

V zápisu $O(n^2)$ se schovají právě ty detaily, které závisí na implementaci, a ne na myšlence algoritmu.

Složitost algoritmu odloženého přijetí

Věta 10 (Časová složitost)

Při použití ukazatelů do seznamů a tabulek pořadí lze algoritmus implementovat s časovou složitostí $O(n^2)$ a pamětovou složitostí $O(n^2)$.

Důkaz.

Vytvoření tabulek pořadí: projdeme n seznamů délky n , tedy $O(n^2)$ času i paměti. Návrhů je nejvýše n^2 a každý zpracujeme v konstantním čase. Celkem $O(n^2)$.

Složitost algoritmu odloženého přijetí

Věta 10 (Časová složitost)

Při použití ukazatelů do seznamů a tabulek pořadí lze algoritmus implementovat s časovou složitostí $O(n^2)$ a pamětovou složitostí $O(n^2)$.

Důkaz.

Vytvoření tabulek pořadí: projdeme n seznamů délky n , tedy $O(n^2)$ času i paměti. Návrhů je nejvýše n^2 a každý zpracujeme v konstantním čase. Celkem $O(n^2)$.

Je $O(n^2)$ hodně, nebo málo?

Vstup sám obsahuje $2n$ seznamů délky n , tedy řádově n^2 položek.

Algoritmus je proto **lineární ve velikosti zapsaného vstupu**.

Vůči čemu složitost měříme

Definice 11 (Velikost vstupu a časová složitost)

Velikost vstupu je počet elementárních symbolů potřebných k jeho zápisu ve zvolené reprezentaci.

Časová složitost algoritmu je funkce T , kde $T(n)$ je *největší* počet elementárních kroků, které algoritmus provede na některém vstupu velikosti n .

- Slovo *největší* je důležité — měříme **nejhorší případ**, chceme záruku pro každý vstup dané velikosti.
- Přesnou funkci T zpravidla neznáme ani nepotřebujeme; stačí asymptotický horní odhad.
- **Pozor u číselných úloh:** rozdíl mezi *hodnotou* čísla N a počtem bitů $\lceil \log_2(N + 1) \rceil$ je zásadní.

Kontrolní seznam pro návrh algoritmů

1. **Specifikace:** Jaké jsou přípustné vstupy? Co přesně musí splnit výstup?
2. **Příklady:** Jak vypadají malé instance? Kde se lze první přirozený nápad?
3. **Struktura:** Která vlastnost problému dovoluje udělat bezpečný krok nebo rozdělit práci?
4. **Algoritmus:** Je každý krok jednoznačný? Co si musí výpočet pamatovat?
5. **Správnost:** Proč skončí? Jaký invariant zachovává? Jak z něj plyne vlastnost výstupu?
6. **Složitost:** Kolikrát se opakuje hlavní krok? Kolik stojí jedna iterace při zvolené reprezentaci?
7. **Meze:** Které předpoklady důkaz používá? Co když jsou preference neúplné, obsahují remízy, nebo má projekt více míst?