1.3.9 Amortizovaná složitost. Jedná se o výpočet průměrné složitosti nejhoršího případu pro posloupnost n opakování dané instrukce. Jestliže n opakování v nejhorším případě vyžaduje čas $\mathcal{O}(T(n))$, pak jedno provedení vyžaduje čas $\mathcal{O}(T(n))/n$, a to je amortizovaná složitost jedné instrukce.

Jsou tři základní způsoby, jak amortizovanou složitost zjišťovat.

- První je tzv. *agregační* — postupuje se přímo podle předchozího odstavce.
- Druhá metoda je tzv. *účetní*. Každému provedení instrukce přiřadíme jistý kredit. Jestliže provedení instrukce nespotřebuje celý kredit, zbývající část kreditu je možno využít v dalších provedení instrukce, které jsou náročnější a na které by jejich kredit nestačil. Podmínkou ale je, aby žádná instrukce v posloupnosti nespotřebovala víc než je součet jejího kreditu a zatím nevyužitých částí kreditů.
- Třetí metoda je tzv. *potenciálová*. Označme D_i stav po provedení i-té instrukce. Máme tedy posloupnost n stavů (většinou datových struktur) $D_0, \ldots, D_{n-1}$. Každé D_i je přiřazeno nezáporné číslo, tzv. potenciál $\Phi(D_i)$. Označme ještě c_i skutečnou cenu přechodu od D_{i-1} k D_i. Pak amortizovaná cena $\hat{c}_i$ příslušná D_i je definována jako

  $$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}).$$

  Pak platí

  $$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).$$

  Odtud dostáváme podmínky na potenciály, totiž pro každé i musí platit $\Phi(D_i) \geq \Phi(D_0)$.

1.3.10 Na přednášce si ukážeme výpočet amortizované složitosti všemi třemi způsoby na příkladu následujícího pseudokódu

INCREMENT(A)
1. $i = 0$
2. `while` $i < A.length$ a $A[i] = 1$
3. $\quad A[i] := 0$
4. $\quad i := i + 1$
5. `if` $i < A.length$
6. $\quad A[i] := 1$

Kapitola 2

Časová složitost a správnost algoritmů

2.1 Časová složitost algoritmů

Výpočet časového odhadu ukážeme na příkladě Euklidova algoritmu, který pro dvě kladná nenulová přirozená čísla najde jejich největší společný dělitel.

2.1.1 Euklidův algoritmus.

Rekurzivní verze Euklidova algoritmu:

Vstup: Kladná přirozená čísla a, b.

Výstup: $\gcd(a, b)$.

EUKLID(a, b)
1. `if` $b = 0$
2. `return` a
3. `else return` EUKLID$(b, a \pmod b)$

Pro zjištění časového odhadu vycházíme z jeho rekurzivního tvaru. Nejprve dokážeme tři pomocná tvrzení.

2.1.2 Horní odhad časové složitosti Euklidova algoritmu dokazuje následující tvrzení.

Tvrzení. Označme x_k a y_k dvojici čísel $x_k > y_k$ po k-tém rekurzivním volání. Pak platí $y_{k+2} < \frac{y_k}{2}$.

Důkaz. Víme, že $y_{k+2} < y_{k+1} < y_k$. Jestliže $y_{k+1} \leq \frac{y_k}{2}$, pak $y_{k+2} < y_{k+1}$ dokazuje, že $y_{k+2} < \frac{y_k}{2}$.

Předpokládejme, že $y_{k+1} > \frac{y_k}{2}$. Pak

$$y_{k+2} < x_{k+1} - y_{k+1} = y_k - y_{k+1} < \frac{y_k}{2}.$$

2.1.3 Dolní odhad dokážeme pomocí několika lemmat.

Lemma 1: Je-li $a > b \geq 1$ a algoritmus EUKLID(a, b) potřebuje k rekurzivních volání, pak $a \geq F(k+2)$ a $b \geq F(k+1)$, kde $F(i)$ je i-tý člen Fibonacciho posloupnosti.

Připomeňme, že Fibonacciho posloupnost je posloupnost:

$$F(0) = 0,\ F(1) = 1,\ F(n) = F(n-1) + F(n-2) \text{ pro } n \geq 2.$$

Důkaz je možné vést indukcí podle počtu rekurzivních volání:

Základní krok: Pro $k = 1$ je $b \geq 1 = F(2)$ a $a > b \geq 1$, tj. $a \geq 2 = F(3)$.

Indukční krok: Předpokládejme, že tvrzení platí pro počet $k \geq 2$ rekurzivních volání. Předpokládejme, že pro dvojici a, b, $a > b$ je potřeba $k+1$ volání. Procedura EUKLID(a, b) volá proceduru EUKLID$(b, a(\bmod b))$, která potřebuje k volání. Z indukčního předpokladu víme, že $b \geq F(k+2)$ a $z = a(\bmod b) \geq F(k+1)$. Máme $z = a - qb$ pro vhodné q celé a $z < b$. Protože $z < b$, je $q \geq 1$; odtud

$$a = qb + z \geq qF(k+2) + F(k+1) \geq F(k+2) + F(k+1) = F(k+3).$$

Ukázali jsme, že $a \geq F(k+3)$ a $b \geq F(k+2)$.

Lemma 2: EUKLID$(F(k+2), F(k+1))$ potřebuje $k+1$ rekurzivních volání.

Lemma 3: Pro každé $n \geq 0$ platí $F(n+2) \geq \left(\frac{3}{2}\right)^n$.

Důkaz. Použijeme matematickou indukci.

Základní krok. Pro $n = 0$ a $n = 1$ tvrzení platí, protože $F(2) = 1 \geq \left(\frac{3}{2}\right)^0$ a $F(3) = 2 \geq \left(\frac{3}{2}\right)^1$.

Indukční krok. Předpokládejme, že platí $F(n) \geq \left(\frac{3}{2}\right)^{n-2}$ a $F(n+1) \geq \left(\frac{3}{2}\right)^{n-1}$. Pak

$$F(n+2) = F(n+1) + F(n) \geq \left(\frac{3}{2}\right)^{n-1} + \left(\frac{3}{2}\right)^{n-2} = \left(\frac{3}{2}\right)^n \left(\frac{2}{3} + \frac{4}{9}\right) \geq \left(\frac{3}{2}\right)^n.$$

Stačí si uvědomit, že $\left(\frac{2}{3} + \frac{4}{9}\right) = \frac{10}{9}$.

2.1.4 Tvrzení: Algoritmus EUKLID(a, b) vyžaduje $\mathcal{O}(\lg b)$ rekurzivních volání. Tedy jeho složitost vztažená k počtu celočíselných dělení je lineární (neboť velikost vstupu je úměrná $\lg(a + b)$).

2.2 Správnost algoritmů

2.2.1 K ověření správnosti algoritmu je třeba ověřit dvě věci

1. algoritmus se na každém vstupu zastaví,
2. algoritmus po zastavení vydá správný výstup – řešení.

Použití obou kroků si nejprve ukážeme na velmi dobře známých algoritmech.

Na přednášce ukážeme správnost některých následujících algoritmů.

2.2.2 Bublinkové třídění.

Vstup: posloupnost přirozených čísel $a[1], a[2], \ldots, a[n]$.

Výstup: posloupnost setříděná do neklesající posloupnosti.

```
begin
     for k = n step -1 to 2 do
          for j = 1 step 1 to k − 1 do
              if a[j] > a[j + 1] then
                  zaměň a[j] a a[j + 1]
end
```

2.2.3 Fakt, že se algoritmus 2.2.2 zastaví, je zaručen tím, že vnější cyklus se opakuje $(n-1)$-krát.

2.2.4 Tvrzení. Po i-tém proběhnutí vnějšího cyklu, tj. pro $k = n - i$, platí

a) $a[n-i+1], a[n-i+2], \ldots, a[n]$ jsou největší z čísel $a[1], a[2], \ldots, a[n]$

b) $a[n-i+1] \leq a[n-i+2] \leq \ldots \leq a[n]$.

Důkaz tohoto tvrzení se vede indukcí podle n počtu průchodů vnitřním cyklem.

Základní krok: Pro $i = 0$, tj. před proběhnutím vnitřního cyklu, je $n-i+1 = n+1$ a takový člen posloupnosti není. Pro $i = 1$, tj. po jednom proběhnutí vnitřního cyklu, je $a[n-1+1] = a[n]$ a je to největší prvek posloupnosti.

Indukční krok: Jestliže tvrzení platí před k-tým průchodem vnitřního cyklu, pak po jeho průchodu je $a[n-k+1] \geq a[j]$ pro $j \leq n-k$, tedy platí a) a navíc je nejmenší z $a[n-k+1]$, $a[n-k+2]$, …, $a[n]$.

2.2.5 Správnost Euklidova algoritmu 2.1.1 Protože se zbytky při dělení čísla r číslem t stále zmenšují a jsou to přirozená čísla, musí jednou nastat případ, kdy zbytek je nula. Proto se algoritmus vždy zastaví.

Uvědomte si, že nejpozději po prvním průchodu krokem 2 platí $r \geq t$.

2.2.6 Tvrzení. Dvojice čísel r, t a dvojice čísel t, z z Euklidova algoritmu 2.1.1 mají stejné společné dělitele.

2.2.7 Variant. Pro důkaz faktu, že se algoritmus na každém vstupu zastaví, je založen na nalezení tzv. *variantu*. Variant je hodnota udaná přirozeným číslem, která se během práce algoritmu snižuje až nabude nejmenší možnou hodnotu (a tím zaručuje ukončení algoritmu po konečně mnoha krocích). Poznamenejme, že někdy se též hodnota zvětšije k předem známé maximální hodnotě.

V příkladu 2.2.2 se jednalo o číslo k, v příkladu 2.1.1 se jednalo o zbytek z při dělení čísla r číslem t.

2.2.8 Invariant. *Invariant*, též *podmíněná správnost algoritmu*, je tvrzení, které

- platí před vykonáním prvního cyklu algoritmu, nebo po prvním vykonání cyklu,
- platí-li před vykonáním cyklu, platí i po jeho vykonání,
- při ukončení práce algoritmu zaručuje správnost řešení.

Pro algoritmus pro bublinkové třídění je invariantem tvrzení 2.2.4, pro Euikleidův algoritmus tvrzení 2.2.6.

2.2.9 Minimální kostra. Je dán prostý neorientovaný graf $G = (V, E)$ s množinou vrcholů V a množinou hran E. Dále je dáno ohodnoceni a hran, tj zobrazení $a\colon E \to \mathbb{N}$. Úkolem je najít kostru K grafu G takovou, že

$$\sum_{e \in K} a(e) \quad \text{je nejmenší.}$$

Ukážeme správnost jakéhokoli algoritmu založeného na následujícím schematu.

2.2.10 Obecné schema.

Vstup: souvislý neorientovaný graf $G = (V, E)$ a ohodnocení hran a.

Výstup: hrany minimální kostry K.

1. (Inicializace)
   $K := \emptyset$, $\mathcal{S} = \{\{v\} \mid v \in V\}$;
2. (Výběr hrany.)
   Dokud $\mathcal{S}$ není jednoprvková
   vybereme hranu $e \in E \setminus K$ takovou, že
   vede mezi dvěma různými množinami z $\mathcal{S}$, označme je C_1, C_2, a aspoň pro jednu z nich je nejlevnější hrana vedoucí z ní.
3. (Úpravy.)
   $K := K \cup \{e\}$;
   $\mathcal{S} := (\mathcal{S} \setminus \{C_1, C_2\}) \cup \{C_1 \cup C_2\}$.

2.2.11 Ukončení schematu pro minimální kostru (variant). Uvedené schema není algoritmus – není v něm uvedeno, jakým způsobem vybíráme hranu e v kroku 2. Jestliže však tento krok implementujeme kteroukoli metodou, která zajistí, že hranu v konečném čase najdeme, pak schema musí skončit. Ano, zpracováním každého výběru hrany v kroku 2 se zmenší počet množin v systému $\mathcal{S}$ o jednu. Protože $\mathcal{S}$ má na začátku práce schematu n množin, po $n-1$ krocích 3 bude $\mathcal{S}$ jednoprvková a schema skončí.

2.2.12 Tvrzení (invariant). Jestliže množina hran K před vykonáním kroku 2 je částí některé minimální kostry a vybereme-li hranu e podle schematu 2.2.10, pak množina hran $K \cup \{e\}$ je také částí některé minimální kostry.

Důkaz: Předpokládejme, že množina K vytvořená schematem 2.2.10 je částí minimální kostry T_{min}. Vezměme hranu e z kroku 2. Platí buď $e \in T_{min}$ nebo $e \notin T_{min}$.

První případ je jednodušší: jestliže $e \in T_{min}$, pak $K \cup \{e\} \subseteq T_{min}$ a opravdu, nová množina K je částí některé minimální kostry – totiž T_{min}.

Uvažujme tu horší variantu, totiž $e \notin T_{min}$ a předpokládejme, že hrana $e = \{u, v\}$ spojuje dvě komponenty souvislosti K, které označíme C_1 a C_2, tj. $u \in C_1$ a $v \in C_2$. Předpokládejme, že e je nejlevnější hrana vycházející ven z komponenty C_1. Protože minimální kostra T_{min} je souvislý graf, existuje cesta P v T_{min} z vrcholu u do vrcholu v. Označme e_1 hranu P, která vychází z množiny C_1.

Protože e je nejlevnější hrana vycházející z C_1 a e_1 také vychází z C_1, platí $a(e) \leq a(e_1)$.

Přidáme-li ke stromu jednu hranu, uzavřeme právě jednu kružnici; tj. $T_{min} \cup \{e\}$ obsahuje kružnici a to $P \cup \{e\}$. Proto $T = (T_{min} \cup \{e\}) \setminus \{e_1\}$ je také kostrou. Cena kostry T je $a(T_{min}) + a(e) - a(e_1)$. Protože T_{min} je minimální kostra, musí platit

$$a(T_{min}) + a(e) - a(e_1) \geq a(T_{min}), \quad \text{tj.} \quad a(e) \geq a(e_1).$$

Odtud $a(e) = a(e_1)$ a proto $a(T) = a(T_{min})$, proto T je také nějaká minimální kostra a navíc $K \cup \{e\} \subseteq T$.

2.2.13 Pozorování. Jak Kruskalův algoritmus, tak Primův algoritmus jsou zvláštní případy obecného schematu 2.2.10.