1.3.4 Eucleidův algoritmus. Výpočet časového odhadu ukážeme na příkladě Eucleidova algoritmu, který pro dvě kladná nenulová čísla najde jejich největší společný dělitel.

Nerekurzivní tvar Eucleidova algoritmu:

Vstup: čísla a, b, $b \neq 0$.

Výstup: $gcd(a, b)$.

1. (Inicializace.)
   $r := a$, $t := b$.

2. (Výpočet částečného podílu a zbytku.)
   `repeat while` $z > 0$
   `do` $z := r \,(\mathrm{mod} t)$;
   $r := t$;
   $t := z$;

3. (Největší společný dělitel.)
   `return(`r`)`

Je možná také rekurzivní verze Eucleidova algoritmu

EUCLEID(a, b)
1. `if` $b = 0$
2. `return` a
3. `else return` EUCLEID$(b, a(\mathrm{mod} b))$

1.3.5 Časový odhad Eucleidova algoritmu. Při zjišťování časového odhadu budeme vycházet z jeho rekurzivního tvaru.

Lemma 1: Je-li $a > b \geq 1$ a algoritmus EUCLIED(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:

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

Důkaz je možné vést indukcí podle počtu rekurzivních volání:

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

2. Indukční krok: Předpokládejme, že tvrzení platí pro počet $k - 1 \geq 1$ rekurzivních volání. Uvažujme $k \geq 2$. Procedura EUCLIED(a, b) volá proceduru EUCLIED$(b, a(\mathrm{mod} b))$, která potřebuje $k - 1$ volání. Z indukčního předpikladu víme, že $b \geq F(k+1)$ a $z = a(\mathrm{mod} b) \geq F(k)$. Máme $z = a - qb$ pro vhodné q celé a $z < b$. Protože $z < b$, je $q \geq 1$ a proto

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

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

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

1.4 Správnost algoritmů

1.4.1 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.

1.4.2 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
```

1.4.3 Fakt, že se algoritmus 1.4.2 zastaví, je zaručen tím, že vnější cyklus se opakuje $(n - 1)$-krát.

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

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

1. Základní krog: 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.

2. 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 plati a) a navic je nejmenší z $a[n - k + 1]$, $a[n - k + 2]$, $\ldots$, $a[n]$.

1.4.5 Správnost Eucleidova algoritmu 1.3.4 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 $r \geq t$.

1.4.6 Tvrzení. Dvojice čísel r, t a dvojice čísel t, z z Eukleidova algoritmu 1.3.4 mají stejné společné dělitele.

1.4.7 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).

V příkladu 1.4.2 se jednalo o číslo k, v příkladu 1.3.4 se jednalo o zbytek z při dělení čísla r číslem t.

1.4.8 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 1.4.4, pro Euikleidův algoritmus tvrzení 1.4.6.

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

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

Ukážeme správnost jakéhokoli algoritmu založeného na následujícím schematu.

1.4.10 Obecné schema.

Vstup: souvislý neorientovný 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.)
   $K := K \cup \{e\}$;
   $\mathcal{S} := (\mathcal{S} \setminus \{C_1, C_2\}) \cup \{C_1 \cup C_2\}$.

1.4.11 Ukončeni 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.

1.4.12 Tvrzení (invariant). Jestliže množina hran K před vykonáním kroku 2 je částí některé minimální kostry a vybereme-li hranu e podle schematu 1.4.10, pak množina hran $K \cup \{e\}$ je také částí některé minimální kostry.

Důkaz: Předpokládejme, že množina K vytvořená schematem 1.4.10 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 S_1 a S_2, tj. $u \in S_1$ a $v \in S_2$. Předpokládejme, že e je nejlevnější hrana vycházející ven z komponenty S_1. Protože minimální kostra T_{min} je souvislý graf, existuje cesta C v T_{min} z vrcholu u do vrcholu v. Označme e_1 hranu C, která vychází z množiny S_1.

Protože e je nejlevnější hrana vycházející z S_1 a e_1 také vychází z S_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 $C \cup \{e\}$. 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

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

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

1.4.13 Pozorování. Jak Kruskalův algorismus, tak Primův algoritmus jsou zvláštní případy obecného schematu 1.4.10.