Obsah

Vyhledávání v textu

Úkol: V daném textu najít všechny výskyty zadaných textových vzorů
Konečná abeceda: $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \ldots \sigma_n\}$
Slova (v abecedě Σ): Konečná posloupnost $\sigma_1, \sigma_2, \sigma_3, \ldots \sigma_n$
Množina všech slov: Σ^*
Operace skládání: $a = \alpha_1 \alpha_2 \ldots \alpha_m$
$b = \beta_1 \beta_2 \ldots \beta_n$
à $ab = \alpha_1 \alpha_2 \ldots \alpha_m \beta_1 \beta_2 \ldots \beta_n$
Prázné slovo: e … jednotkový prvek pro skládání
$ae = ea = a \quad \forall a \in \Sigma^*$
$a \in \Sigma^*$ je předpona (prefix) slova $b \in \Sigma^*$, pokud $\exists\, u \in \Sigma^*$: $au = b$
$a \in \Sigma^*$ je vlastní předpona slova $b \in \Sigma^*$, pokud a je předpona b a zároveň $a \neq b$ (nebo $u \neq e$)
a je přípona (sufix) slova b, pokud $\exists$ u: $ua = b$
délka slova a l(a) … počet znaků

ZADÁNÍ:

- abeceda Σ
- prohledávaný text: slovo $x = \sigma_1 \sigma_2 \sigma_3 \ldots \sigma_n \in \Sigma^*$
- hledané vzory: $K = \{y_1, y_2, \ldots y_k\}$
  $y_p = \tau_{p,1} \ldots \tau_{p,l(p)}$ $l(p) \leq n$ pro p=1..k

OTÁZKA:
- najít všechny výskyty vzorů z K ve slově x, tj. dvojice <i, y_p>, kde y_p je přípona $\sigma_1 \sigma_2 \ldots \sigma_i$

Naivní algoritmus

```
for p := 1 to k do                          // přes všechny vzory
     for i := 1 to n-l(k)+1 do              // přes možné začátky
          j := 0;
          while j < l(p) and σ_{i+j} = τ_{p,1+j} do
               j := j + 1                   // ^ porovnání 1 písmena
          od
          if j = l(p) then Report(i, y_p) fi
     od                                     // ^ hlášení výsledků
od
```

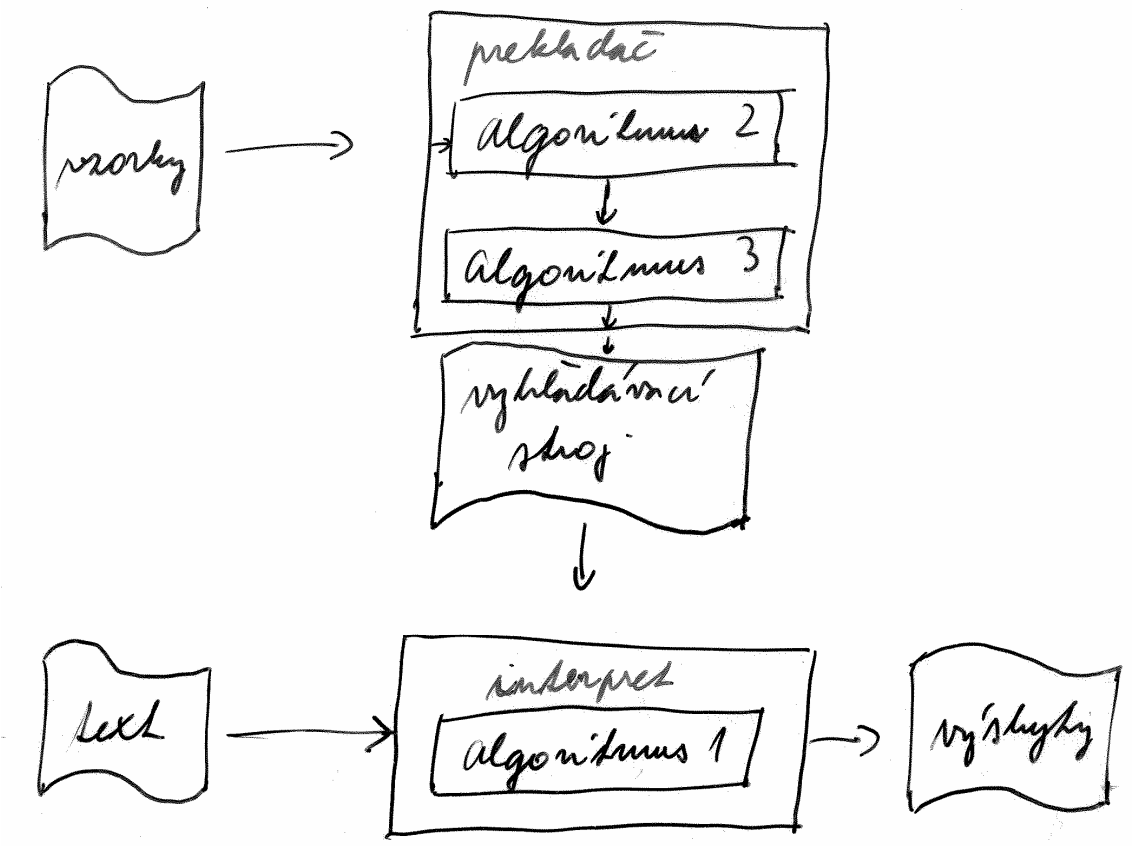
l = l(1) + l(2) + … + l(k) … součet délek vzorů
T(n, l) – počet kroků naivního algoritmu pro rozměry vstupů n, l
Pro každé p = 1..k vykonáme (n-l(p)+1) – krát vnitřní while cyklus
Vykonání while cyklu v nejhorším případě:

- 2 * l(p) + 1 skoků (podmínka)
- l(p) přiřazení (tělo)
- l(p) nepodmíněný skok (zacyklení)

$T(n, l) \geq \Sigma_{(\text{přes } p)} ((n-l(p)+1) * (4\, l(p) + 1)) = 4 * l * n - 4\, \Sigma\, l(p)^2 + 3 * l + k * n + k$
$\geq 4 * (l * n - l^2) = \Omega\,(l * n - l^2)$ - typicky n >> l

Metoda Aho – Corasicková (1975)

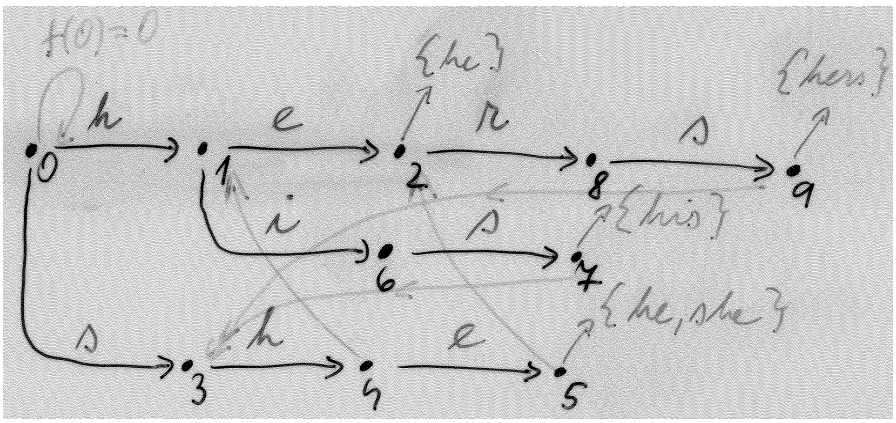
Zdroj neefektivity naivního algoritmu – vzorky se porovnávají samostatně
Idea: předzpracování (kompilace) vzorků
Překlad: O(l)
Vyhledávání: O(n)
Celkem: T(n, l) = O(n+l)



Zlepšení:

1. porovnávat naráz více vzorů
2. vstup procházet jen jednou

K = {he, she, his, hers}
Σ … latinská abeceda



Vyhledávací stroj

pro abecedu Σ a množinu slov K je čtveřice M = (Q, g, f, out), kde

- Q = {0, …, q) je množina stavů
- g: Q x Σ **à** Q ∪ {⊥} je přechodová funkce
  - g(0, σ) ≠ ⊥ pro ∀ σ ∈ Σ
- f: Q **à** Q je zpětná funkce f(0) = 0
- out: Q **à** P(K) je výstupní funkce

Zajímá nás:

- konstrukce vyhledávacího automatu
- algoritmy 2 a 3 a jejich složitost
- vlastnosti zkonstruovaného automatu
- jeho korektnost
- složitost algoritmu 1 (interpretace)

Korektnost automatu:
Pro každé $x = \sigma_1 \sigma_2 \sigma_3 \ldots \sigma_n \in \Sigma^*$ pro přečtení $\sigma_1 \ldots \sigma_i$ vypíše právě ty dvojice $<i, y_p>$, kde y_p je příponou $\sigma_1 \ldots \sigma_i$

Tři důležité vlastnosti:
(A): Když spojíme stavy s a g(s, σ) hranou označenou symbolem σ, dostaneme strom s kořenem O. Každá cesta začínající v kořenu je předponou nějakého vzoru z K. Naopak každá předpona vzorku z K popisuje cestu k nějakému stavu. Tato předpona reprezentuje příslušný stav. Hloubka d(s) stavu s je rovna délce reprezentujícího slova. Platí d(g(s, σ)) = d(s) + 1, jestliže je g(s, σ) definované a různé od 0.
(B): Stroj má vlastnost (A) a zároveň pro každý stav s reprezentovaný slovem u je stav f(s) reprezentovaný nejdelší vlastní příponou slova u, která je současně předponou nějakého vzorku z K. **è** pro s ≠ 0 platí d(f(s)) < d(s)
(C): Stroj má vlastnost (A) a zároveň pro každý stav s reprezentovaný slovem u a každý vzor y ∈ K platí, že y ∈ out(s) právě když y je příponou u.

Algoritmus 1 – interpret vyhledávacího stroje

VSTUP: text $x = \sigma_1 \sigma_2 \sigma_3 \ldots \sigma_n \in \Sigma^*$
vyhledávací stroj M = (Q, g, f out) pro Σ a $K = \{y_1, .. y_k\}$
VÝSTUP: posloupnost dvojic $<i, y_p>$, $1 \le i \le n$, $1 \le p \le k$

```
stav := 0;
for i := 1 to n do
     while g(stav, σi) = ⊥ do
          stav := f(stav)
     od;
     stav := g(stav, σi);
     for y ∈ out(stav) do
          Report(i, y)
     od;
od;
```

Příklad: u s h e r s
reprezentace: <délka_přečtené _části, stav>
<0, 0>, u, <1, 0> s, <2, 3> h <3, 4> e, <4, 5> ↑she, ↑he, <4, 2>, r, <5, 8>, s, <6, 9>, ↑hers.

Lemma 1.1.:

Pokud má stroj M = (Q, g, f, out) vlasnost (B), potom je s vyjímkou náročnosti ohlášení výsledků interpretovaný v lineárním čase, t.j. pro počet kroků T(n) algoritmu 1 (s výjimkou řádků

```
for y ∈ out(stav) do
     Report(i, y)
od;)
```

platí T(n) = O(n).

Oprávněnost vyjímky:
- ∀ korektní automat hlásí výskyty
- alternativa: závislost na počtu výskytů

Důkaz:
Algoritmus je lineární mimo tělo while – cyklu. Posloupnost stavů během výpočtu: s_0, s_1, … s_z, kde $s_0 = 0$.
Označíme pro i=0..z-1.
$G_i = \{j=0..i \mid s_{j+1} = g(s_j, \sigma)\}$ počet dopředných přechodů v hlavní cyklu.
$F_i = \{j=0..i \mid s_{j+1} = f(s_j)\}$ počet zpětných přechodů v těle while.
Zajímá nás: $|F_{z-1}|$, víme: $|G_{z-1}| = n$.
Dokážeme: $|F_i| + d(s_i) + 1 \leq |G_i|$
Indukcí **è** $|F_{z-1}| + d(s_{z-1}) + 1 \leq |G_{z-1}| = n$
$|F_{z-1}| \leq n-1$ … O(n)

Lemma 1.2:

Jestliže má M = (Q, g, f, out) vlastnost (B), potom se algoritmus 1 pro přečtení písmena σ_i nachází ve stavě reprezentovaným nejdelší příponou slova $\sigma_1..\sigma_i$, která je předponou nějakého vzorku z množiny K.

Důkaz: Indukcí podle i

- stav reprezentovaný příponou $\sigma_1..\sigma_i$
- … nejdelší

Lemma 1.3.:

Jestliže má M = (Q, g, f, out) vlasnosti (B) a (C), potom je korektní.

Důkaz:
Nechť má vlastnosti (B) a (C). Chceme dokázat, že M při interpretaci algoritmem 1 vydá po přečtení i-tého písmena slova $x = \sigma_1..\sigma_n$ právě všechny dvojice <i, y_p>, kde y_p je přípona $\sigma_1..\sigma_n$. M je po přečtení σ_i ve stavu s reprezentovaném slovem v.

a) Jestliže M vydá <i, y_p>, je $y_p \in$ out(s) a podle (C) je příponou v. Podle L1.2 je v příponou $\sigma_1..\sigma_n$, tedy dvojice <i, y_p> měla být vydaná

b) Jestliže M má vydat <i, y_p>, je y_p příponou $\sigma_1..\sigma_n$. Podle L1.2 je i příponou v a vzhledem k vlastnosti (C) bude <i, y_p> vydaná dvojice

Algoritmus 2

Vyrobí Q, g a polotovar o pro out.
VSTUP: K = {y_1,..y_p} neprázdné, z Σ
VÝSTUP: Q = {0..q}, g: Q × Σ **à** Q ∪ {⊥}
o: Q **à** P(K)

```
var q: integer;
q := 0;
for p := 1 to k do Enter(y_p) od
for all σ ∈ Σ do
     if g(0, σ) je nedefinované then
          g(0, σ) := é
     fi
od

procedure Enter(τ_1..τ_m)
{ připojení slova y = τ_1..τ_m }
begin
     s := 0; j := 1;
     while j ≤ m and g(s, τ_j) definované do
          s := g(s, τ_j); j := j + 1
     od;
     while j ≤ m do
          q := q + 1
          definuj g(s, τ_j) = q;
          s := q;
          j := j + ů
     od
     definuj o(s) = {τ_1..τ_m}
end {Enter}
```

Lemma 1.4.

Algoritmus 2 pracuje v čase O(l) a jeho výstupy mají vlastnost (A).

Algoritmus 3

Vyrobí zpětnou funkci f a výstupní funkci out z polotovaru o.
VSTUP: Q = {0..q}, g: Q × Σ **à** Q ∪ {⊥}, o: Q **à** P(K)
VÝSTUP: f: Q **à** Q, out: Q **à** P(K)

```
begin
     queue := "prázdná fronta"
     definuj f(0) := 0; out(0) := ∅
     for all σ ∈ Σ do
```

```
          s := g(0, σ);
          if s <> 0 then
               definuj f(s) := 0;
               out(s) := o(s)
               zařaď s do queue;
          fi;
     od;
     while queue <> prázdná do
          r := první prvek z queue;
          vyřaď r z queue
          for all σ ∈ Σ do
               if g(r, σ) je definované then
                    s := g(r, σ);
                    t := f(r);
                    while g(t, σ) je nedefinováno do
                         t := f(t)
                    od;
                    definuj f(s) := g(t, σ);
                    definuj out(s) := o(s) ∪ out(f(s));
                    zařaď s do queue
               fi
          od
     od
end
```

Lemma 1.5.

Algoritmus 3 pracuje v čase O(l) a jeho výstupy tvoří vyhledávací stroj korektní vzhledme k Σ a K.

Idea důkazu:
Ukážeme, že vytvořený stroj M = (Q, g, f, out) má vlastnosti (B) a (C).
- indukcí podle hloubky stavu, využijeme prohledávání do šířky

Odhad složitosti interpretace pomocí potenciálů

Idea: Hloubka aktuálního stavu je aktuální potenciál. Zpracování písmena pomocí g zvyšuje potenciál, použití f snižuje . Chceme ukázat, že celkový počet použití funkcí f a g je O(n). Analogie- s amortizovanou složitostí: v celkovém průběhu je #použití f odhadnutý O(n), tj. amortizovaně O(1) na 1 znak vstupu

Tvrzení: # použití f ≤ n

Důkaz: n … délka vstupu

= # použití g … z algoritmu

≥ celkový přírustek potenciálu … g zvyšuje hloubku nejvíc o 1, ale g(0, ?) = 0

= celkový pokles potenciálu … počáteční potenciál = 0

+ koncová hodnota

≥ celkový pokles potenciálu … koncová hloubka ≥ 0

≥ # použití f … každé použití f snižuje potenciál alespoň o 1

Rabin-Karp algoritmus

Idea: Považujeme vzor za číslo (znaky za cifry) v soustavě se základem a = |Σ|. Počítáme modulo zvolené q ∈ N (prvočíslo). Porovnáváme s podřetězci délky l (= délka vzoru) vstupu. Když se hodnota v charakteristiky vzoru a hodnota t_i charakteristiky podřetězce na pozici i nerovnají, vzor se na pozici i určitě nenachází.

Výpočet v a t_1 pomocí Hornerova schématu v čase O(l)

$v = \tau_l + a(\tau_{l-1} + a(\tau_{l-2} + \ldots + a(\tau_2 + a*\tau_1) \ldots))$

Výpočet t_{i+1} z t_i

$t_{i+1} = a(t_i - a^{l-1}*\sigma_i) + \sigma_{i+l}$

∧ ∧

vypuštění první cifry přidání poslední cifry

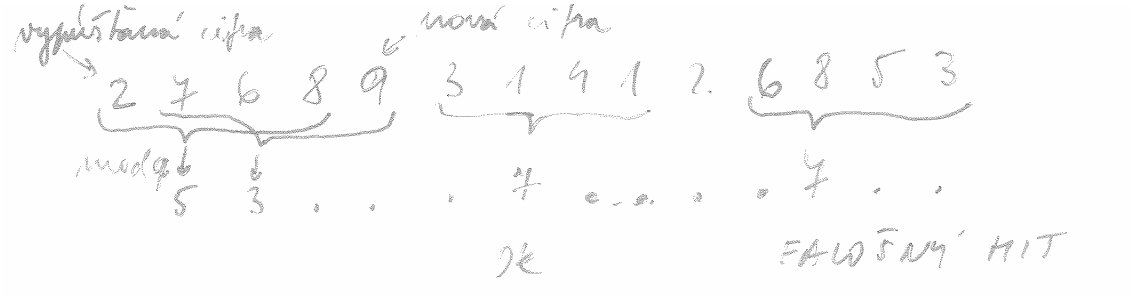
Volba q: prvočíslo; a*q se vejde do registru

è aritmetické operace v čase O(1) (ne O(l))

$t_{i+1} = (a(t_i - h*\sigma_i) + \sigma_{i+l}) \bmod q$

kde $h = a^{l-1} \bmod q$, přepočítáme v O(l)

ALE: výpočet modulo q způsobuje falešné hity, když $v = t_i$, ale $\tau_1..\tau_l \neq \sigma_i.. \sigma_{i+l-1}$




```
Rabin-Karp (σ, τ, a, q)
w ß délka textu σ
l ß délka vzoru τ          // možné rozšíření pro víc vzorů
h ß a^(l-1) mod q          // předvýpočet
v ß 0
t1 ß 0
for i ß 1 to l do
     v ß (a*v + τi) mod q
     t1 ß (a*t1 + σi) mod q
for i ß 1 to n-l+1 do
     if v = ti then
          if τ1..τl = σi..σi+l-1 then
               "nalezené na počáteční pozici" i
          end if
     end if
     if i < n-l+1 then
          ti+1 ß (a(ti - σi*h) + σi+l) mod q
     end if
```

Analýza složitosti:

nejhorší složitost: Θ ((n-l+1)*l)

průměrná složitost : O(n+l) + O(l * # OK) + O(l * # FAL)

#OK … počet nalezených pozic (predpokládáme O(1))

#FAL … počet falešných hitů za předpokladu rovnoměrného rozložení,

tj. #FAL = O(n/q)

è O(n) + O(l *(1 + n/q)), pro n ≥ m – O(n+l)

Algoritmus Knuth-Moriss-Pratt

Vyhledávání 1 vzorku (zjednodušený algoritmus A.-C.)
Idea: pozorováním vzorku vůči svým posunům spočítáme prefixovou funkci π nezávislou na zpracovávaném písmeně textu

π: {1..l} **à** {0..l-1}
π(q) = max { k: k < q a $\sigma_1..\sigma_k$ je suffix $\sigma_1..\sigma_q$}

```
KMP-vyhledávání(σ, τ)
n ß délka σ
l ß délka vzoru τ
π ß spočítej_prefix_funkci(τ)
q ß 0
for i ß 1 to n do
     while q > 0 and τ_{q+1} ≠ σ_i do q ß π(q)
     if τ_{q+1} = τ_i then q ß q+1      // neopakujeme test q písmen
     if q = l then print "našlo" i – l
     q ß π(q)

spočítej_prefix_funkci(τ)
l ß délka τ
τ(1) ß 0
k ß 0
for q ß 2 to l do
     while k > 0 and τ_{k+1} ≠ τ_q do k ß π(k)
     if τ_{k+1} = τ_q then k ß k+1
     π(q) ß k
od
reurn π
```

Pracujeme s 1 vzorkem:
à čísla stavů (indexy π) odpovídají délce úspěšně nalezeného prefixu
à hodnoty π odpovídají (zpětné funkci z A.-C. a) délce nového prefixu po neúspěšném porovnání (jednoznačného) dalšího znaku

Složitost:

spočítej_prefix_funkci:	O(l)	-- použitím potenciálu
tělo KMP	O(n)	

Souvislosti

Další metody vyhledávaní a příbuzné úlohy

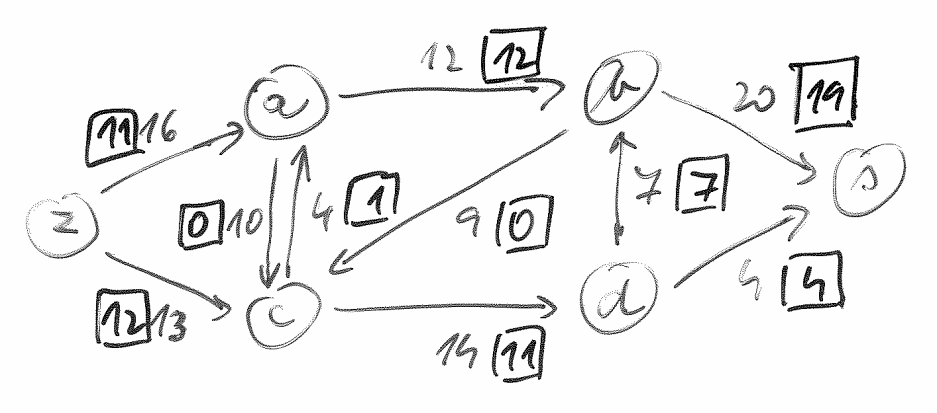
- konečné automaty (vyhledávající i nekonečné množiny vzorů, zadání např. regulárními výrazy)
- Boyer – Moore algoritmus – porovnávání vzorků od nejpravějších písmene
    - netestujeme vždy všechny písmena: čas až O(n/l)
    - opakovanému testování písmen se dá vyhnout (za paměť)
- přibližný vyhledávání: v genetických datech
- signatury: přepočítané charakteristiky umožňující rychlé vyloučení některých možností
- vyhledávání na webu: indexy slov

Poslední 2 možnosti: předzpracování textu místo vzorků

Toky v sítích

Definice: Síť S = (G, c, z, s)

G = (V, E) orientovaný graf
c: E **à** R_0^* kapacity hran
z ∈ V zdroj
s ∈ V s ≠ z, spotřebič (stok)



Tok t v síti S = (G, c, z, s) je zobrazení t: E_G **à** R splňující:

1) $0 \le t(h) \le c(h)$ pro ∀ h ∈ E
2) $\delta(t, u) = 0$ pro ∀ u ∈ V \ {z, s}

$\delta(t, u) = \Sigma_{v \in V}(t(u,v) - t(v, u))$
Značení t(h) = t(u, v) pro h = <u, v>; c(h) = c(u, v)
Definice: Velikost toku t je δ(t, z)… z je zdroj
Problém: Najít v dané síti tok o maximální velikosti
BÚNO: 1 zdroj a 1 spotřebič stačí: Máme S = (G, c, $\{z_1..z_m\}$, $\{s_1..s_n\}$)
zavedeme superzdroj a superspotřebič

Metoda: L.R.Ford, D.R.Fulkerson (1962)

inicializuj tok t na 0
while existuje zlepšující cesta P do
zlepši tna hranách cesty P
od
return t

Reziduální síť: S_t k síti S a toku t: $S_t = (G_t, c_t, z, s)$
$G_t = (V, E_t)$, $E_t = \{<u, v> \in V \times V \mid c_t(u, v) > 0\}$, kde
$c_t(u, v) = c(u, v) - t(u, v) + t(v, u)$ je reziduální kapacita hrany
Reziduální kapacita cesty P je $c_t(P) = \min \{c_t(u, v) \mid (u, v) \in P\}$
Dodefinujeme c(v, u) = 0 pro c(u, v) > 0.
BÚNO: t(u, v) = 0 anebo t(v, u) = 0

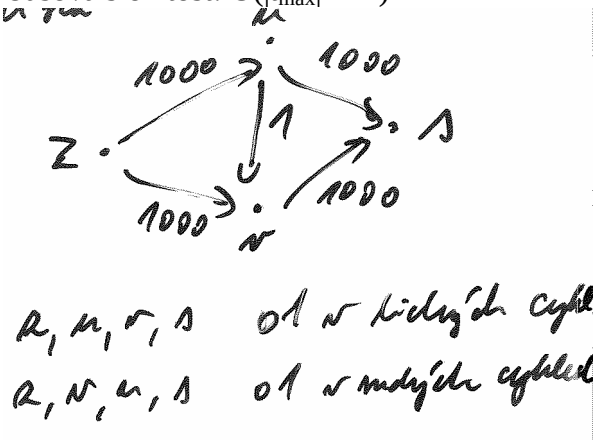
```
Algoritmus Ford-Fulkerson (G, z, s)
forall h ∈ E_G do t(h) := 0 od
while ∃ cesta z = v_0, v_1, … v_k = s taková, že
     ∀i, i=1..k platí
```

```
buď (v_{i-1}, v_i) je hrana G a t(v_{i-1}, v_i) < c(v_{i-1}, v_i)
nebo (v_i, v_{i-1}) je hrana G a t(v_i, v_{i-1}) > 0
// pokud cesta neexistuje, výpočet končí a t je max. tok
do
     urči největší Δ > 0 takže ∀i=1..k platí
          buď (v_{i-1}, v_i) je hrana a Δ ≤ c_t(v_{i-1}, v_i)
          nebo (v_i, v_{i-1}) je hrana a Δ ≤ t(v_i, v_{i-1});
     ∀i=1..k položíme
          je-li (v_{i-1}, v_i) hrana, pak
               t(v_{i-1}, v_i) := t(v_{i-1}, v_i) + Δ
          je-li (v_i, v_{i-1}) hrana, pak
               t(v_i, v_{i-1}) := t(v_i, v_{i-1}) - Δ
od
```

- algoritmus pro iracionální hodnoty c nemusí skončit
- pokud hodnoty c jsou racionální čísla **è** převedeme na celá čísla

Dále předpokládáme celočíselné c

Časová složitost: $O(|t_{max}| * m)$

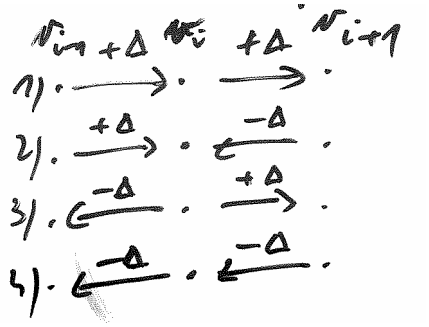


Tvrzení: t je na začátku každého cyklu tok

Důkaz: Indukcí podle počtu cyklů:

a) nulový tok t je tok
b) změny o Δ nezmění $\delta(t, v_i)$ pro vnitřní vrcholy v_i nalezené cesty $\delta(t, v_i) = 0$

4 možnosti:



Definice:

- řez $C \subseteq V(G)$, $z \in C$ & $s \notin C$
- velikost řezu (kapacita) $c(C) = \Sigma\, c(h)$ pro $h=(u, v)$, $u \in C$, $v \notin C$

Lemma: Je-li t tok a C řez, pak velikost toku je menší nebo rovna velikosti řezu C.

Důkaz: Velikost toku t je $\delta(t, z)$ a platí:

$\delta(t, z) = \delta(t, C)$ // pro $u \in C \setminus \{z\}$ je $\delta(t, u) = 0$

$= \delta^+(t, C) - \delta^-(t, C) \leq c(C) - \delta^-(t, C) \leq c(C)$ Q.E.D

$\delta^+(t, C)$ … součet toků všemi hranami h=(u, v) takových, že $u \in C$ & $v \notin C$
z definice $\leq c(C)$

$\delta^-(t, C)$ … součet toků všemi hranami h=(v, u) takových, že $v \notin C$ & $u \in C$
z definice ≥ 0

Věta: Ford-Fulkersonův algoritmus najde vždy maximální tok v síti (s celočíselnými kapacitami)
Důkaz: Nechť C je množina vrcholů, do kterých vede posloupnost
$z = v_0, v_1, \ldots v_k = v$, t.ž. $\forall i = 1..k$ platí
buď (v_{i-1}, v_i) je hrana a $t(v_{i-1}, v_i) < c(v_{i-1}, v_i)$
nebo (v_i, v_{i-1}) je hrana a $t(v_i, v_{i-1}) > 0$
Při ukončení algoritmu je C řez ($z \in C$ a $s \notin C$).
Chceme ukázat, že při ukončení algoritmu je velikost toku t rovna velikosti řezu C.
Nechť $v \in C$ a $z = v_0, v_1, \ldots v_k = v$ je posloupnost, která to dokazuje. Nechť $w \notin C$.
Pokud h = (v, w) je hrana, pak kdyby platilo $t(v, w) < c(v, w)$, potom posloupnost $v_0, v_1, .. v_k = v$, w dokazuje $w \in C$, spor **è** $t(v, w) = c(v, w)$ (1)
Pokud h = (w, v) je hrana, pak kdyby platilo $t(w, v) > 0$, potom stejná posloupnost dokazuje $w \in C$, sport => $t(w, v) = 0$ (2)
Z (1) a (2) plyne rovnost velikostí toku t a řezu C.
Tok t je největší možný, protože kdyby existoval větší tok, musel by mít větší velikost než řez C, což je podle lemmatu nemožné.

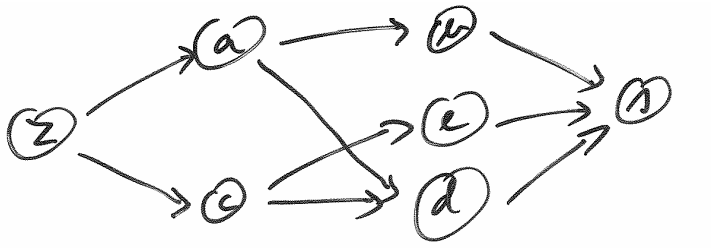
Strategie volby cesty: vždy nejkratší (prohledáváním do šířky)

- **è** EDMONDS, KARP (1972) … $O(n*m^2)$
- **è** DINIC (1970) … $O(n^2*m)$
  využívá vrstevnatou síť a nejkratších cest

Vrstvená síť

Síť je vrstevná, pokud existuje rozklad na množiny jejich vrcholů pro $X_0, X_1, \ldots X_k$ tak, že

- $X_0 = \{z\}$
- $X_k = \{s\}$
- $(u, v) \in E(G)$, $x \in X_i$, $y \in X_j$ **è** $j = i + 1$
- z každého vrcholu kromě spotřebiče hrana vychází a do každého vrcholu kromě zdroje hrana vchází



Hrana (u, v) v síti S s tokem t je nasycená, pokud $t(u, v) = c(u, v)$.
Tok v síti S je nasycený, pokud každá cesta v S ze zdroje do spotřebiče obsahuje nasycenou hranu.
Značení: d(u, v) … délka nejkratší orientované cesty z u do v, měřená počtem hran
Pozorování: ve vrstvené síti platí $\forall v$: $d(z, v) + d(v, s) = d(z, s)$

```
Algoritmus DINIC(G, z, s)
for all h ∈ E(G) do t(h) := 0; r(h) := c(h) od
while ∃ cesta P ze z do s v St do
     { síť St´ vytvoříme z St vynecháním všech hran,
     které neleží na nejkratší orientované cestě ze z do s }
     St´ := St
     for all v ∈ V(G) do urči d(z, v)
     for all v ∈ V(G) do urči d(v, s)
     for all (u, v) ∈ E(St´) do
          if d(z, u) + 1 + d(v, s) > d(z, s) then
               vynechej(u, v) z St´
     q := nasycený tok v St´
     for all (u, v) ∈ E(St´) do t(u, v) := t(u, v) + q(u, v) od
od
```

Lemma: Nechť S_{t1} a S_{t2} jsou reziduální sítě vytvořené dvěma po sobě následujícími iteracemi Dinicova algoritmu. Potom $d(S_{t1})(z, s) \leq d(S_{t2})(z, s) - 1$

Důkaz (idea):

Pro cesty obsahující hrany S_{t1} v S_{t2} platí (q je nasycený).

Pro cesty obsahující novou hranu přidanou do S_{t2}: nova hrana (v_i, v_{i-1}) vzniká kvůli úpravě toku na hraně (v_{i-1}, v_i) na nějaké minimální cestě $v_0, v_1, \ldots v_k$. Délka cesty s novou hranou je $d(S_{t1})(z, v_i) + 1 + d(S_{t2})(v_{i-1}, s) = i + 1 + k - i + 1 = k + 2$.

Přitom $d(S_{t1})(z, v_i) \leq d(S_{t2})(z, v_i)$ a $d(S_{t1})(v_{i-1}, s) \leq d(S_{t2})(v_{i-1}, s)$

Důsledek: Časová složitost Dinicova algoritmu na síti S s n vrcholy a m hranami je $O(n * f(n, m))$, kde $f(n, m)$ je čas potřebný k nalezení nasyceného toku ve vrstvené síti.

Důkaz:

Maximální délka cesty je n, tj. počet, tj. počet konstruovaných sítí je nejvýše n a časová náročnost je odhadnuta $f(n, m)$.

Určení nasyceného toku ve vrstvené síti

VSTUP: vrstvená síť S´ = (G, c, z, s)

VÝSTUP: nasycený tok g v síti S´

```
for all (u, v) ∈ E(G) do q(u, v) := 0 od
while ∃ cesta P ze zdroje z do s v S´ do
     c(P) = min {c(u, v) | u, v ∈ P}
     for all (u, v) ∈ P do q(u, v) := q(u, v) + c(P) od
     S´ := S´ - {všechny nasycené hrany cesty P}
     pročisti_síť(S´)            // uzavírání hran a vrcholů
od
```

Složitost:

1) nalezení a update cesty … O(n)
   - výběr libovolné hrany v pročištěné síti
2) každý průchod cyklem nasytí (a uzavře) aspoň 1 hranu **è** O(m) průchodů

è složitost nalezení nasyceného toku ve vrstvené síti O(m*n)

è složitost max. toku – Dinicův algoritmus $O(n^2m)$

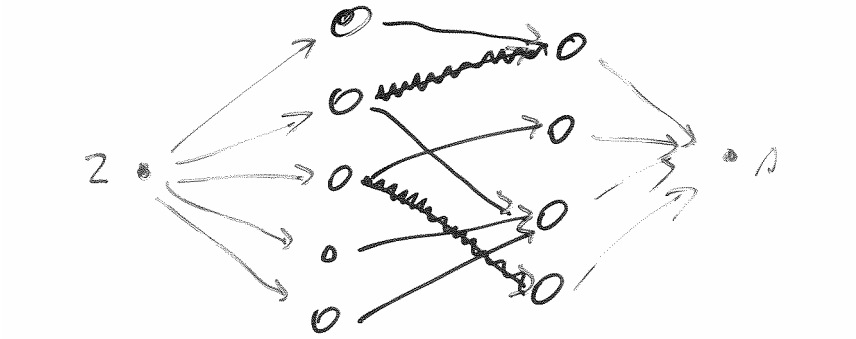
Poznámka

Algortimus Karzanov (1974),
Malmotra, Pramooh Kumar, Maheshwari (1978): $O(n^3)$

Implementace Pročisti_síť (S´):
Idea: uzavírání hran a vrcholů neležících na cestě P ze z do s, tž. c(P) > 0
Pamatuju si vstupní $d_{in}(v)$ a výstupní $d_{out}(v)$ stupně vrcholů do neuzavřených vrcholů.
Uzavírám, pokud d_(v) = 0 a dekrementuju stupeň sousedů.
è vrchol a hrana se uzavírá 1x, v konstantním čase

Aplikace: určení maximálního párování v bipartitním grafu

Párování je množina $M \subseteq E(G)$ grafu g, tž. $\forall e_1, e_2 \in M: e_1 \cap e_2 = \varnothing$
Maximální párování je párování s maximálním počtem hran.



Kapacita hran je 1.
Maximální tok je n = |V| **è** složitost Ford –Fulkersonova algoritmu je O(n*m)
Algoritmus HOPCROFT, KARP (1973)

Goldbergův algoritmus (preflow – push)

Graf G=(V, H), c: $V \times V$ **à** R^+_0, vlna/tok t: $V \times V$ **à** R^+_0
Rezerva hrany h = (u, v)
- r(u, v) = c(u, v) – t(u, v) + t(v, u)

Doplnění opačných hran (s nulovou kapacitou)
Vlna (preflow): t: $V \times V$ **à** R^+_0, $0 \le t(h) \le c(h)$
$\forall v \in V \setminus \{z\}$, v má kladný přebytek excess: V **à** R
$excess_t(v) = \Sigma_{u \to v} t(u, v) - \Sigma_{v \to u} t(v, u)$
Idea algoritmu:
- konstruujeme vlnu (ne tok po celých cestách)
- každý vrchol v má výšku H(v)
- přesouvama přebytky vrcholu po hranách s rezervoou

Jestliže vrchol v má přebytek alespoň δ, rezerva hrany (v, w) je alespoň δ, potom přenést přebytek z v do w hranou (v, w) znamená:

a) tok opačnou hranou (w, v) je 0 **è** zvýšit tok hranou (v, w) o δ
b) tok opačnou hranou (w, v) je ≥ δ **è** snížit tok hranou (w, v) o δ
c) tok opačnou hranou (w, v) je > 0 a ≤ δ **è**
snížit tok (w, v) na 0 a zvýšit tok hranou (v, w) o δ - t(w, v)

Převední způsobí:
- snížení přebytku v o δ

- zvýšení přebytku w o δ
- snížení rezervy (v, w) o δ
- zvýšení rezervy (w, v) o δ

```
Algoritmus
výška zdroje z je N = |V| (pevná)
výška ostatních vrcholů je 0
tok hran ze zdroje je rovný kapacitě hrany
tok na ostatních hraná je 0
while ∃ vrchol s kladným přebytkem, kromě s do
     v := vrchol s kladným přebytkem, v ≠ s
     if ∃ h = (v, w) s kladnou rezervou a H(v) > H(w) then
          (v, w) := hrana (vycházející z v) s kladnou rezervou
                         a splňující H(v) > H(w);
          δ := min (excess(w, z(v, w))
          převést přebytek z v do w velikosti δ
     ekse
          H(w) := H(v) + 1
     end if
end
```

3 způsoby vykonání těla
1) zvýšení vrhovou v: příkaz H(w) := H(v) + 1
2) nasycení převedení přebytku
   δ je rovná rezervě hrany (v, w) **è** nuluje rezervu hrany
3) nenasycené převedení přebytku
   δ je menší než rezerva hrany (v, w) **è** nuluje přebytek v

Označení: N = |V|, M = |H|
Poznámka: vrcholy můžou vystoupat nad N (výšku zdroje) a tak vrátit přebytek do zdroj

Lemma 1:

Po inicializaci vždy platí, že neexistuje hrana h = (v, w) taková, že H(v) > H(w) + 1 a rezerva h je nenulová.
Důkaz:
Po inicializaci podmínka platí, protože každá hrana (v, w) splňující H(v) > H(w) vede ze zdroje a má nulovou rezervu (**ç** je nasycená)
Vykonání těla while hranu porušující podmínku nevytvoří, protože

a) po zvednutí v: hrana má rezervu (předpoklad) v má přebytek (vybráním)
   è nezvedáme, ale přesouváme přebytek h
b) po převedení přebytku opačnou hranou (w, v), když r(v, w) = 0. Potom H(w) > H(v) a situace nenastane.

Věta 1:

Jestliže Goldbergův algoritmus skončí, najde největší tok. (Parciální správnost algoritmu)
Důkaz:

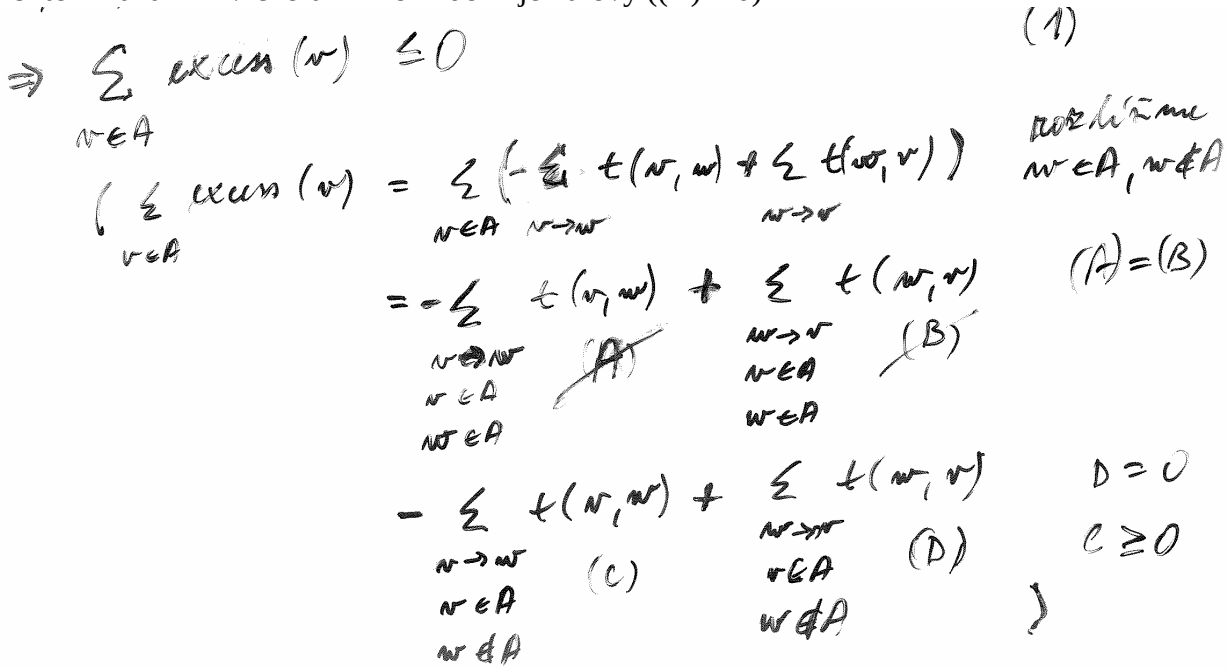
Když while cyklus skončí, potom ∀ v ∈ V, v ≠ s má nulový přebytek **è** zkonstruovaná vlna je tok.
Ještě chceme: tok je maximální

- **ç** neexistuje zlepšující cesta
- **ç** ∀ cesta má nasycenou hranu s nulovou rezervou

Nechť máme cestu ze zdroje do spotřebiče. Začíná na výšce N(=H(z)), končí v 0(=H(s)), má nejvíc N-1 hran **è** existuje hrana klesající aspoň o 2. A podle L1 má nulovou rezervu.

Lemma 2

Vždy od ukončení inicializace, jestliže má vrchol v kladný přebytek, potom z něho vede orientovaná cesta do zdroje, na které mají všechny hrany kladnou rezervu.
Důkaz:
Nechť v má kladný přebytek. Označíme A množinu vrcholů, do kterých vede z v orientovaná cesta z hran s kladnout rezervou.
Uvažujeme hranu (x, y), x∉ A, y ∈ A. Jestliže tok hranou (x, yA) je kladný, potom opačná hrana (y, x) má kladnou rezervu a x ∈ A kvůli cestě v, .. y, x – SPOR.
è tok hranami z vrcholu mimo A do A je nulový ((D) = 0)



v ∈ A, přebytek v je kladný **è** v A je vrchol se záporným přebytkem, protože podle (1) je celkový přebytek A záporný **è** zdroj je v A, protože je jediný vrchol se záporným přebytkem **è** z v do zdroje vede cesta z hran s kladnou rezervou, z definice A.

Lemma 3:

Výška vrcholu není nikdy větší než 2N.
Důkaz:
Sporem. Předpokládejme, že zdviháme vrchol v nad 2N. Potom je ve výšce 2N a má kladný přebytek. Podle lemmatu 2 existuje cesta z v do zdroje z hran s kladnou rezervou. Cesta obsahuje nejvíce N-1 hran, překonává spád (aspoň) N, proto obsahuje hranu (x, y), kde H(x) > H(y) + 1 a podle lemmatu 1 má tato hrana nulovou rezervu. Spor.

Lemma 4:

Počet zdvihnutí během celého algoritmu je nejvíc $2N^2$.
Důkaz:
Máme N vrcholů, každý zdviháme maximálně 2N-krát.

Lemma 5:

Počet nasycených převedení přebytku je za celý výpočet najvíce NM.
Důkaz:
Nechť h = (v, w) je hrana. Součet výšek v a w je mezi 0 a 4N. Jestliže dojde k nasycenému převedení přebytku, potom rezerva (v,w) klesne na 0 a platí H(v) = H(w) + ů.
Abychom znovu mohli nasyceně převádět, musí se zvýšit rezerva na neulovou hodnotu. To nastává v případě převádění přebytku opačnou hranou (w, v).
Takže musíme zvýšit w aspoň o 2 a následně zvýšit v aspoň o 2. Mezi dvěma nasycenými převedeními přebytku hranou h = (v, w) se zvýší H(v) + H(w) aspoň o 4 **è** # nasycených převedení hranou h je ≤ N. Q.E.D

Lemam 6.

Počet nenasycených převedení přebytku za celý výpočet je maximálně $2N^2 + 2N^2M$.
Důkaz:
Označíme S součet výšek vrcholů , které mají kladný přebytek mimo z a s.
Po inicializaci je S = 0, protože jediný zdroj má nenulovou výšku. Na konci výpočtu je S=0, protože „vnitřní“ vrcholy nemají přebytek.
Zvýšení vrhovou zvýší S o 1. Nasycené převedení přebytku hranou (v, w) zvýší S nejvíce o H(w) ≤ 2N (jestliže w přebytek neměl a v přebytek zůstane).
Celkový zvýšení S je nejvíce $2N^2 + 2N.NM.$ (1)
Nenasycené převedení přebytku hranou (v, w) sníží S aspoň o 1, protože výšky se nemění, sčítanec H(v) vypadne a sčítanec H(w) se případně přidá (pokud nebyl v S).
Ale H(v) = H(w) + 1 **è** S se sníží
è celkový počet nenasycených převedení je $2N^2 + 2N^2$ podle (1)

Věta 2:

Časová složitost Goldbergova algoritmu je $O(N^2M)$.

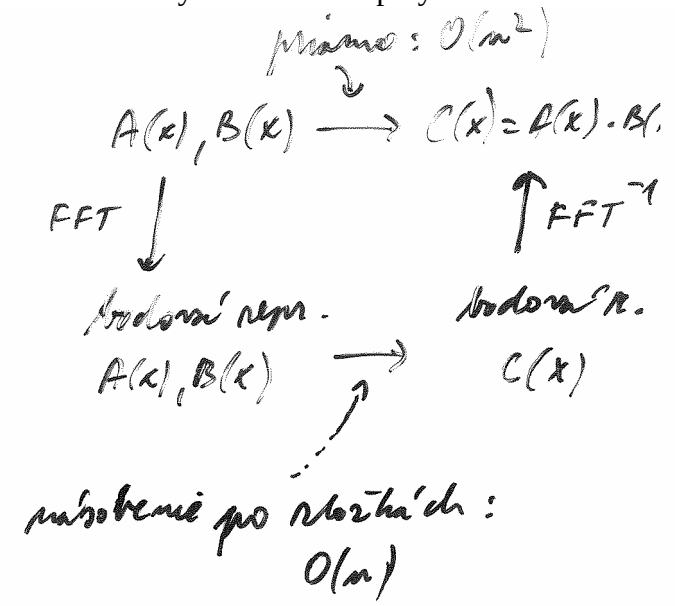
Důkaz: Z lemmat 4, 5, 6

Strategie výběru vrcholu

Vždy nejvyšší vrchol s přebytkem => # nenasycených převedení ≤ $8N^2\sqrt{M}$

Rychlá Fourierova transformace

- motivace – rychlé násobení polynomů

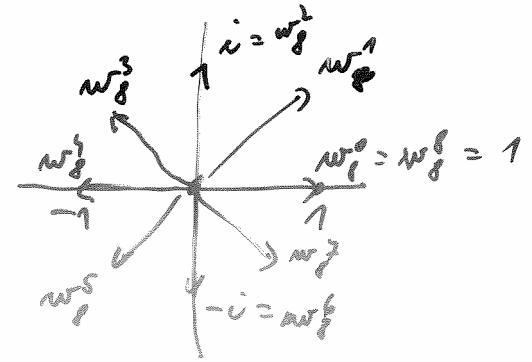


$A(x) = {}_{j=0}\Sigma^{n-1}\ a_j\ x^j$
$B(x) = {}_{j=0}\Sigma^{n-1}\ b_j\ x^j$
$C(x) = A(x) * B(x) = {}_{j=0}\Sigma^{2n-2}\ c_j\ x^j$ kde $c_j = {}_{k=0}\Sigma^{j}\ a_k\ b_{j-k}$

Vektor koeficientů $c = (c_0, c_1, c_2, \ldots c_{2n-2})$ je konvoluce vektorů $a = (a_0 .. a_{n-1})$ a $b = (b_0 .. b_{n-1})$.
Vyhodnocení polynomu v bodě x_0: Hornerovým pravidlem

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \ldots + x_0(a_{n-2} + x_0\ a_{n-1}) ..))$$

Časová složitost: O(n), pro 2n bodů – $O(n^2)$
Ale: FFT (a FFT^{-1}) v $\Theta(n \log n)$
Vhodně vybereme body, ve kterých se budou polynomy vyhodnocovat – komplexní odmocniny 1.



Využití rozděl a panuj.

Komplexní n-té odmocniny 1

Kořeny polynomu $x^n - 1$
Počet kořenů: n; hodnoty $e^{2i\pi k/n}$ pro k = 0.. n-1, kde $e^{iu} = \cos(u) + i \sin(u)$
Hodnotu $\omega_n = e^{2\pi i/n}$ nazveme základní n-tý kořen 1, ostatní kořeny jsou jeho mocniny
Platí:
- $\omega_{dn}^{dk} = \omega_n^k$ $\quad (LS = (e^{2\pi i/dn})^{dk} = (e^{2\pi i/n})^k = PS)$

- $\omega_n^{n/2} = \omega_2 = -1$
- $(\omega_n^{k+n/2})^2 = (\omega_n^k)^2$ (LS = $\omega_n^{2k+n} = \omega_n^{2k}\,\omega_n^n = \omega_n^{2k} = (\omega_n^k)^2$ = PS)
  druhé mocniny všech n různých n-tých odmocnin 1
  tovří (jen) n/2 různých n/2-tých odmocnin 1

è rekurzivní volané podproblémy vyhodnocujeme v polovičním počtu bodě

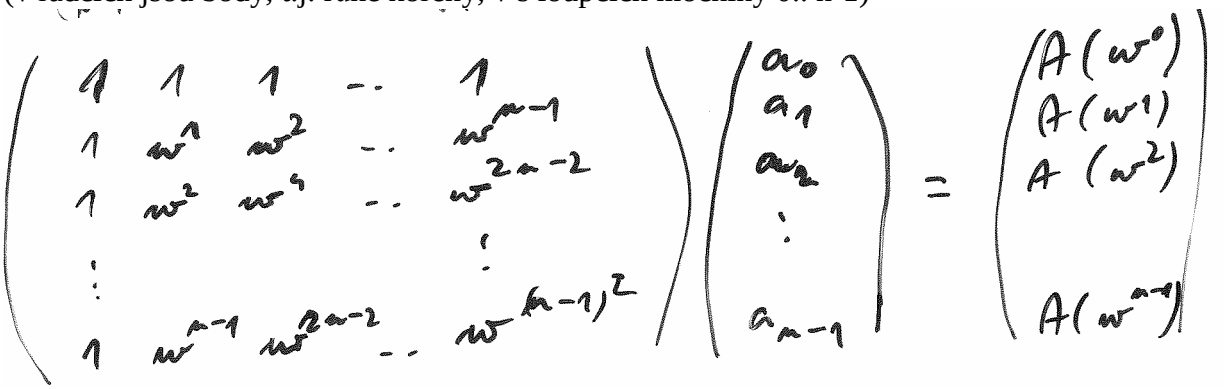
- pro $n \geq 1$ a $k \geq 0$, k není dělitelné n platí: $\sum_{j=0}^{n-1} (\omega_n^k)^j = 0$;
  LS = $((\omega_n^k)^n - 1)/(\omega_n^k - 1) = ((\omega_n^n)^k - 1)/(\omega_n^k - 1) = (1^k - 1)/(\omega_n^k - 1) = 0$ = PS

a pro k, n | k:

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = \sum_{j=0}^{n-1} 1 = n$$

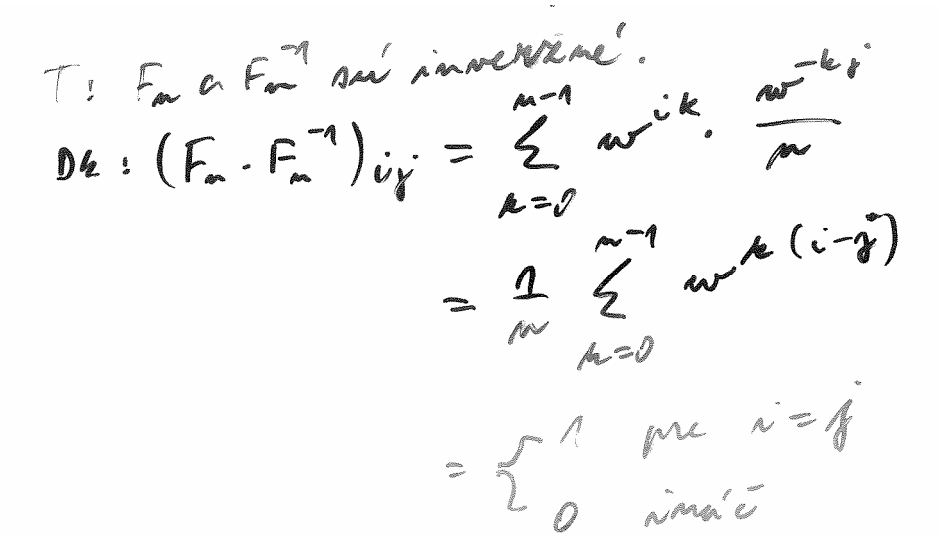
Vyhodnocujeme polynom $A(x) = a_0 + a_1 x + a_2 x^2 + \ldots a_{n-1} x^{n-1}$
v bodech $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$

Zápis pomocí Vandermondovy matice F_n rozměrů n x n, na místě (i, j) je $(\omega_n^i)^j$
(v řádcích jsou body, t.j. různé kořeny, v s loupcích mocniny 0.. n-1)



Tato lineární transformace vektoru (a0, a1 .. an-1) na (A(ω0), A(ω1), … A(ωn-1)) se nazývá diskrétní Fourierova transformace (DFT).

Matice má inverzní (uhádnutím, bez motivace, ??? a odvození)
$(F_n^{-1})_{ij} = \omega^{-ij} / n$,
t.j. inverzní matice má (až na koeficienty 1/n) stejný tvar jako DFT, ale od základního kořene $\omega^{-1} = \omega^{n-1}$



Důsledek: Inverzní FFT dokážeme spočítat ve stjeném čase jako (dopřednou) FFT.

Metoda FFT

předpokládáme $n = 2^k$
K polynomu $A(x) = a_0 + a_1x + a_2x^2 + \dots a_{n-1}x^{n-1}$
vytvoříme dvanové polynomy $A^{[0]}(x)$ a $A^{[1]}(x)$:
$A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$ (sudé koeficienty)
$A^{[0]}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$ (sudé koeficienty)
Platí $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$ (1)
takž problém vyhodnocování A(x) v bodech $\omega_n^0, \omega_n^1, \dots \omega_n^{n-1}$ se redukuje na:
1) vyhodnocení polynomů $A^{[0]}(x)$ a $A^{[1]}(x)$ stupně n/2 v bodech $(\omega_n^0)^2, (\omega_n^1)^2 \dots (\omega_n^{n-1})^2$
(jen n/2 různých hodnot)
2) výpočet hodnot polynomu A(x) z mezivýsledku podle (1)

```
rekurzivní FFT(a)
n := length(a);                  - n je mocnina 2
if n = 1 then return (a);
ωn := e^(2πi/n;
ω := 1;
a[0] := (a0, a2, .. an-2)
a[1] := (a1, a3, .. an-1)
y[0] := rekurzivní FFT(a[0])
y[1] := rekurzivní FFT(a[1])
for k := 0 to n/2 – 1 do
     yk := yk[0] + ωyk[1]        //
     yk+(n/2) := yk[0] - ωyk[1]  //   … společný podvýraz
     ω := ω * ωn    (???)        // ω je aktuální kořen, tj. ωn^k
od
return (y);
```

Zdůvodnení: vydané y je DFT vstupu a:

koncový případ: pro vector délky 1 vracíme $y_0 = a_0$
$y_0 = a_0\ \omega_1^0 = a_0 * 1 = a_0$ Q.E.D.
spočítání hodnot v rekuzi: pro k = 0, 1, … n/2-1
z rekurze máme
$y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) = A^{[0]}(\omega_n^{2k})$
$y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) = A^{[1]}(\omega_n^{2k})$
odvodíme $y_{k+n/2}$

$y_{k+n/2} = y_k^{[0]} - \omega_n^k\, y_n^{[1]}$ $(-\omega_n^k = \omega_n^{k+n/2})$
$= y_k^{[0]} + \omega_n^{k+n/2} y_n^{[1]}$
$= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+n/2}\, A^{[1]}(\omega_n^{2k})$ $(\omega_n^n = 1)$
$= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2}\, A^{[1]}(\omega_n^{2k+n})$
$= A(\omega_n^{k+n/2})$

Složitost:

- režie: $\Theta(n)$ v každém rekurzivním volání (kde n je aktuální velikost dat)
- rekurzivní vztah: $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$

Důkaz správnosti rekurzivních algoritmů

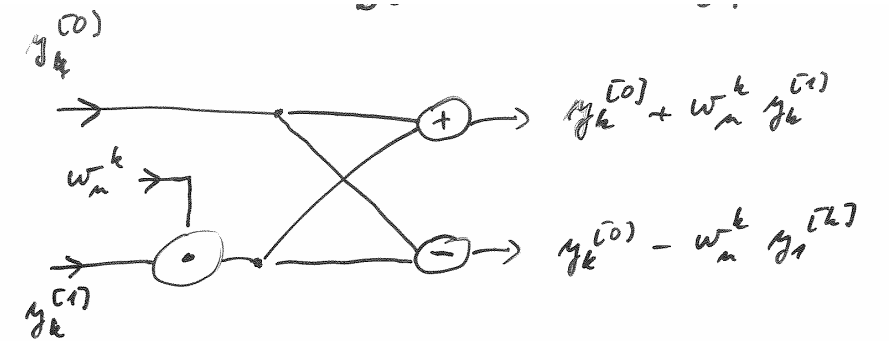
Konečnost:

- velikost vstupních dat se zmenšuje
- ukončíme pro velikost vstupu 1 případně 0

Částečná správnost:

- matematickou indukcí
- = ověříme správnost pro nedělitelné zadání
- indukční krok – z indukčního předpokladu, že algoritmus počítá správně pro data velikost n odvodíme, že počítá správně pro data velikost n+1, resp. 2n

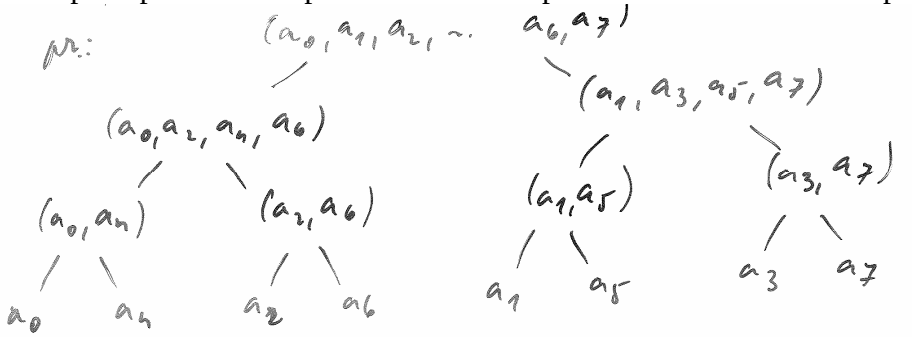
Butterfly operace



Převod rekurze na iteraci („programátorský trik“)

- menší režie
- (někdy) menší paměť (vzhledem k tabelaci) – např. dynamické programování
- složitější, tj. delší program

Idea: přeuspořádání vsuptních koeficientě podle reverzní bitového zápisu



Iterative FFT (a, A) – pseudokód

```
n := length(a)
for k := 0 to n-1 do A[rev_n(k)] := a_k   ... přeuspořádání
for s := 1 to log n do                    ... pro úrovně
     for k := 0 to n-1 step 2^s do
          skombinuj dvě 2^(s-1)-prvkové DFT
          v A[k...k+2^(s-1)-1] a A[k+2^(s-1).. k+2^s-1]
          do 2^s-prvkové DFT v A[k...k+2^s-1]
```

Aplikace FFT

- konvolute polynomů
- analýza signálu, spektrální
- násobení dlouhých čísel
- zpracování obrazu a videa (pomocí kosinovy transformace)
    - analýza, komprese (JPEG, MPEG,), syntéza (v 2D)

Násobení dlouhých čísel

Idea: číslo rozdělíme na skupiny k cifer a chápeme jako hodnotu polynomu v bodě 2^k.
Součin čísel získáme jako hodnotu součinu odpovídajících polynomů v bodů 2^k.

Hrubý odhad složitosti: $O(n \log^2 n) = O(n^{\log_2 3})$ z “rozděl a panuj”
Počítání v okruzích 2n: operace +, -, * počítáme modulo n

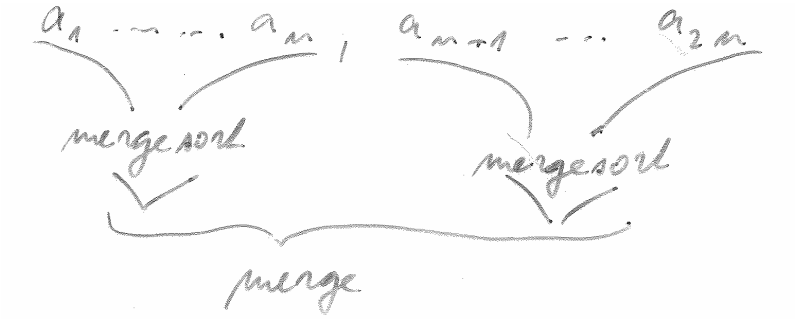
např. $2^8 = 256 \equiv -1 \bmod 257$
$\Rightarrow 2^{16} \equiv 1 \bmod 257$
$\Rightarrow 2 = \sqrt[16]{1}$ v Z_{257}

Zpracování obrazu a videa

Kosinova transformace pro n bodů se spočítá pomocí FFT v 2n bodech

Třídění a kombinační obvody

Třídění slučováním – mergesort

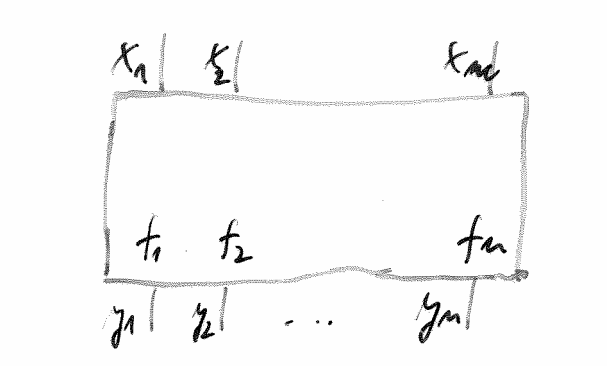



```
procedure merge(var A, B: array [1..n] of T; l,c,r: integer);
begin
     i = l; j = c + 1; k = l;
     repeat
          if i > c then                A[k] := B[j]; inc(j);
          elseif j > r then            A[k] := B[i]; inc(i);
          elseif B[i] > B[j] then      A[k] := B[j]; inc(j);
          else                         A[k] := B[i]; inc(i);
          fi;
          inc(k)
     until k >r;
end;

procedure mergesort(var A, B: array[1..n] of T; l,r: integer);
begin
     if l < r then
          c := (l+r) div 2;
          mergesort(A, B, l, c);
          mergesort(A, B, c+1, r);
          for i = l to r do B[i] := A[i] od
          merge(A, B, l, c, r);
     fi;
end
```

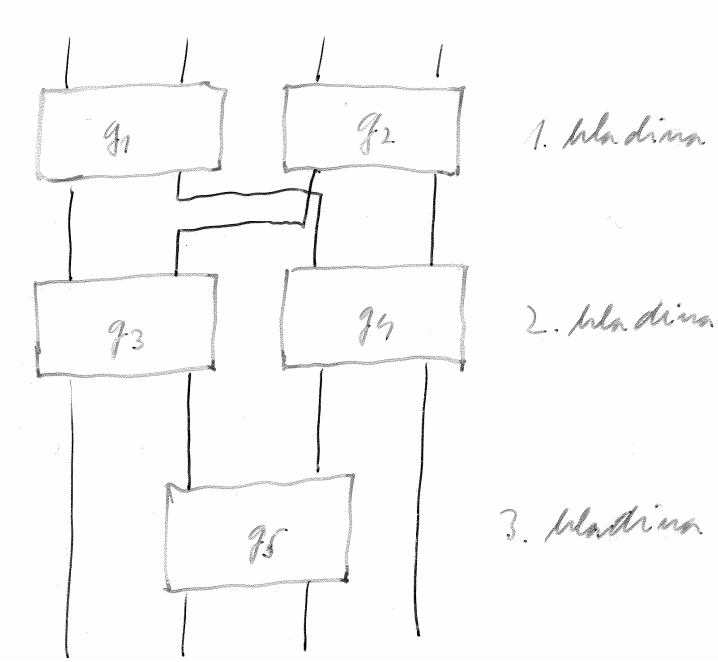
Netřídí na místě, ale s pomocným polem.
T(n) … složitost mergesort
T(n) = 2T(n/2) + k.n
=> T(n) = O(n log n)

Kombinační obvod



hradlo g
x_i i = 1 .. m = i(g) vstupy <1, g, i>
y_j j = 1 . n = o(g) výstupy <0, g, j>
$y_j = f_j(x_1 .. x_m)$ j = 1..n

kombinační obvod = (G, P)
G … množina hradel
Γ: O **à** I
O = {<0, g, j> | g∈ G, j ≤ o(g) }
I = {<1, g, i> | g ∈ G, i ≤ i(g) }
Γ … částečně prosté zobrazení
o∈ O, o∉ dom(Γ) … výstupy obvodu
i∈ I, i ∉ ???(Γ) … vstupy obvodu
prosté … BÚNO … hradlo rozdvojení, …
acyklický graf (G, E)
<g, h> ∈ E <=> ∃j ≤ o(g), ∃i ≤ i(h)
Γ (<0, g, j>) = <1, h, i>

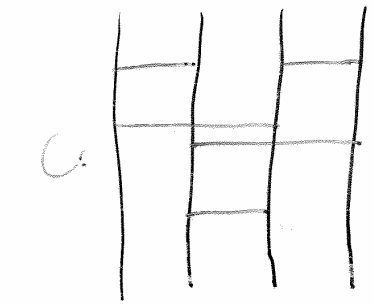


hladina i …všechn hradla, která majíé vstupy napojené na výstupy hradel hladiny j, j <≤ i

hloubka obvodu … d(c) … počet hladin
velikost obvodu … s(c) … počet hradel
stav obvodu

Tvrzení: Jestliže má každé hradlo obvodu shodný počet vstupů a výstupů, potom je počet vstupů obvodu rovný počtu výstupů.
Důkaz: Indukcí podle s(C)
Důsledek: Počet vodičů mezi hladinami je shodný a rovný počtu vstupů

Zjednodušení popisu:

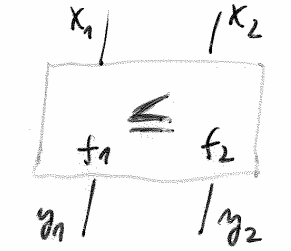


(při vhodném očíslování vodičů)

kombinační síť šířky m a hloubky k:
m vodičů, přepojené hradly v k hladinách

m = 4
k = 3 = d(c)
s(c) = 5

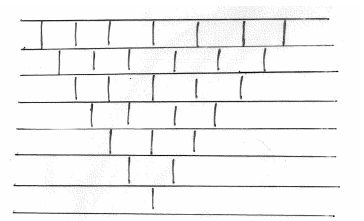
Jediný druh hradla: komparátor



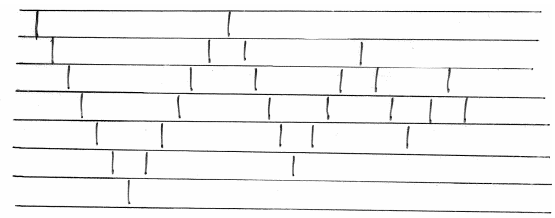
$f_1(x_1, x_2) = \min(x_1, x_2)$
$f_2(x_1, x_2) = \max(x_1, x_2)$

- transpoziční síť

popis: $\langle j, p_1, p_2 \rangle$ $1 \le j \le k, 1 \le p_1 < \le p_2 \le m$
na hladině j spojené vodiči p_1 a p_2
{<1, 1, 2>, <1, 3, 4>, <2, 1, 3>, <2, 2, 4>, <3, 2, 3>}



třídění vkládáním (insertsort), třídění vybíráním, jednostranné bublinkové třídění

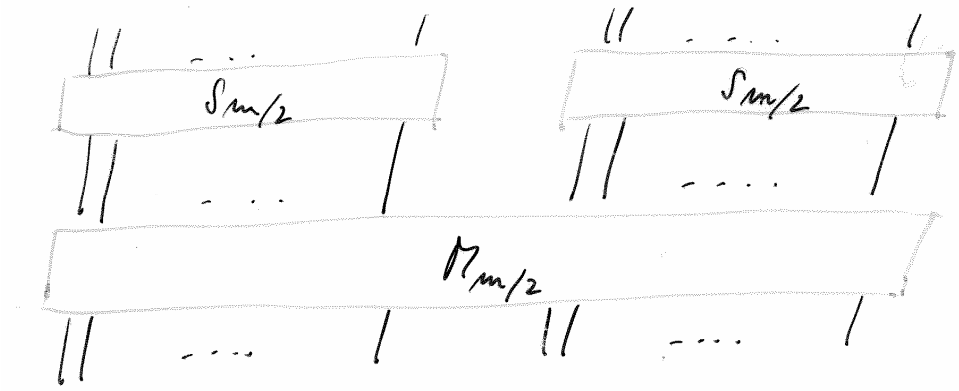


oboustranné bublinkové třídění

Třídění

„obvyklý“ předpoklad … délka posloupnosti 2^k

třídicí síť: S_m m > 1



S_1 … prázdná síť

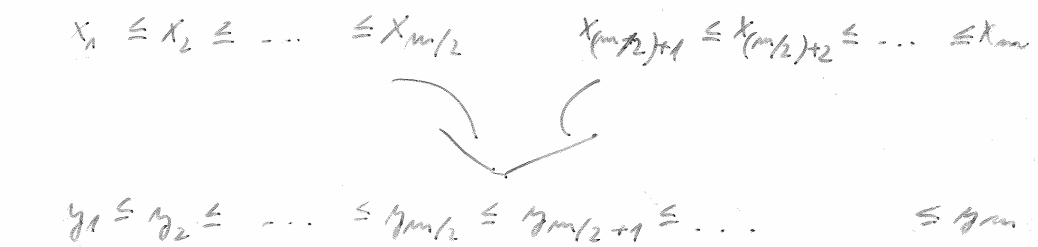
$S_m = S_{m/2}$

$\cup$ { <j, (m/2) + p_1, (m/2) + p_2>; <j, p_1, p_2> $\in$ $S_{m/2}$ }

$\cup$ { <k + j, p_1, p_2>; <j, p_1, p_2> $\in$ $M_{m/2}$ }

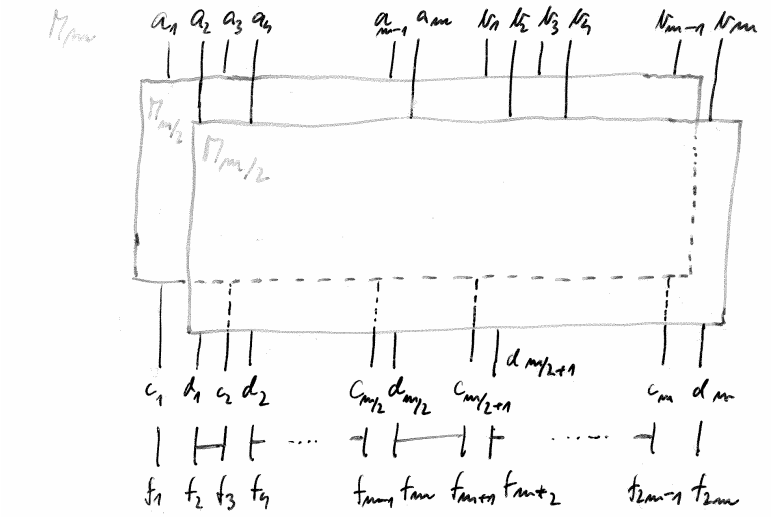
$k = d(S_{m/2})$

Slučovací síť $M_{m/2}$ šířky m



m = 1 M_1 = {<1, 1, 2> } H

m > 1



na vstupu

$a_1 \le a_2 \le \ldots \le a_{m-1} \le a_m$

$b_1 \le b_2 \le \ldots \le b_{m-1} \le b_m$

? na výstupu ?
$f_1 \le f_2 \le \ldots \le f_{m-1} \le f_m \le f_{m+1} \le ... \le f_{2m-1} \le f_{2m}$

Z indukčního předpokladu

$c_1 \le c_2 \le \ldots \le c_{m-1} \le c_m$

$d_1 \le d_2 \le \ldots \le d_{m-1} \le d_m$

Poslední hladina komparátoru:

$f_{2p} \le f_{2p+1}$ $p = 1 \ldots m-1$

zůstává ??

$f_{2p-1} \le f_{2p}$ $p = 1 \ldots m$

tj. $c_p \le d_p$?

$<c_1 \ldots c_p> = <a_1 \ldots a_{2i-1}, b_1 \ldots b_{2(p-i)-1})$ (1)

$c_p \in \{a_{2i-1}, b_{2(p-i)-1}\}$

$<d_1 \ldots d_p> = <a_2 \ldots a_{2j}, b_2 \ldots b_{2(p-j)}>$

$d_p \in \{a_{2j}, b_{2(p-j)}\}$

Rozabereme

$c_p = a_{2i-1}$

(i) $i \le j$ $c_p = a_{2i-1} \le a_{2i} \le a_{2j} \le d_p$ Q.E.D

(ii) $i > j$ $p-i < p-j$ **è** $p-1+1 \le p-j$

$b_{2(p-1)+1}$ není v (1) **è** $\ge c_p$

$c_p = a_{2i-1} \le b_2(p-i)+1 \le b_2(p-i+1) \le b_2(p-j) \le d_p$ Q.E.D.

Složitost:

hloubka:

$d(M_1) = 1$

$d(M_m) = d(M_{m/2}) + 1$ $m>1$

è $d(M_m) = \log (m) + 1$

$d(S_1) = 0$

$d(S_m) = d(S_{m/2}) + d(M_{m/2})$ $m>1$

è $d(S_m) = ½ (\log(m))(\log(m) + 1)$

velikost: odpovídá počtu porovnání

$s(M_1) = 1$

$s(M_m) = 2s (M_{m/2}) + m -1$

è $s(M_m) = m \log (m) + 1$

$s(S_1) = 0$

$s(S_m) = 2s (S_{m/2}) + s(M_{m/2})$ $m > 1$

è $s(S_m) = (m/4) \log (m)(\log(m) – 1) + m – 1$

$= O(m (\log m)^2)$

Aritmetické obvody/sítě

implementace aritmetických operací pomocí booleanovských hradel (AND, OR, NOT, XOR, NAND, NOR)

- sčítařka - vstup x, y, z, výstup s, c (suma, carry/přenos)
  $s = x \oplus y \oplus z$ (XOR)
  $c = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) = \text{majority}(x, y, z)$

úloha: sčítat dvě čísla
  $u = \sum_{i=0}^{n-1} u_i 2^i$
  $v = \sum_{i=0}^{n-1} v_i 2^i$
  $u_i, v_i \in \{0, 1\}$
  do výsledku $s = u+v = \sum_{i=0}^{n-1} s_i 2^i$ $s_i \in \{0, 1\}$

1.řešení: sčítání s přenosem
  $s_i = u_i \oplus v_i \oplus c_{i-1}$ pro i = 0, …, n-1
  $s_n = c_{n-1}$
  tedy $c_{-1}, c_0, \ldots c_{n-1}$ jsou definované
  $c_{-1} = 0$
  $c_i = \text{majority}(u_i, v_i, c_{i-1})$
hloubka obvodu $\Theta(n)$ tj. paralelný čas
velikost obvodu $\Theta(n)$

Cíl: zlepšit hloubku obvodu – carry-lookahead algoritmus
idea: vytvoříme stromovou strukturu místo lineární
ALE: nemáme (včas) vstupní data – konkrétně carry
řešení: programátorský i teoretický trik- budeme počítat namísto hodnot s funkcemi
  (možný pohled: podmíněný výpočet/odložený; rozbor případu-implicitní)+
funkcionální programování, více přechodový (i obousměrný) konečný automat

funkce pro počítání přenosu má 3 možné výstupy:
1. generuje c_i když $u_i \wedge v_i = 0$
2. přenáší c_{i-1} když $u_i \oplus v_i = 1$
2. pohlcuje c_i když $u_i = 0 \wedge v_i = 0$

DC: skládání funkcí pomocí tabulky 3x3
formálně zavedeme pomocnou proměnnou
  $g_i = u_i \wedge v_i$, $p_i = u_i \oplus v_i$
vyjádříme c_i a s_i pomocí g_i a p_i:
  $c_i = g_i \vee (p_i \wedge c_{i-1})$ i = 0, … n-1
  $s_0 = p_0$
  $s_i = p_i \oplus c_{i-1}$ i = 1, … n-1
  $s_n = c_{n-1}$
zavedení operátoru skládání Δ: hodnotám (g_1, p_1) a (g_2, p_2) přířazuje
  $(g_1, p_1) \Delta (g_2, p_2) = (g_1 \vee (p_1 \wedge g_2), p_1 \wedge p_2)$
Splněné g_i znamená generování carry, splnění p_i přenášění carry,
nesplnění ani g_i ani p_i pohlcení carry.

Lemma 1: Δ je asociativní, tj. $((g_1, p_1) \Delta (g_2, p_2)) \Delta (g_3, p_3) = (g_1, p_1) \Delta ((g_2, p_2)) \Delta (g_3, p_3))$
Důkaz:

LS $= (g_1 \vee (p_1 \wedge g_2), p_1 \wedge p_2) \Delta (g_3, p_3) =$
$= (g_1 \vee (p_1 \wedge g_2) \vee ((p_1 \wedge p_2) \wedge g_3), (p_1 \wedge p_2) \wedge p_3) =$
$= (g_1 \vee (p_1 \wedge (g_2 \vee (p_2 \wedge g_3))), p_1 \wedge (p_2 \wedge p_3) =$
$= (g_1, p_1) \Delta (g_2 \vee (p_2 \wedge g_3)), p_2 \wedge p_3)$
$= (g_1, p_1) \Delta ((g_2, p_2) \Delta (g_3, p_3))$ Q.E.D

Poznámka: Δ není komutativní.

Lemma 2: výpočet carry

pokud zavedeme

$(G_0, P_0) = (g_0, p_0)$
$(G_i, P_i) = (g_i, p_i) \Delta (G_{i-1}, P_{i-1})$ i = 1.. n-1

potom

$c_i = G_i$ pro i = 0, … n-1

Důkaz: indukcí i = 0

$c_0 = g_0 \vee (p_0 \wedge c_{i-1}) = g_0 \vee (p_0 \wedge 0) = g_0 \wedge 0 = g_0 = G_0$

jestliže platí lemma pro 0 … i-1, potom pro i máme

$(G_i, P_i) = (g_i, p_i) \Delta (G_{i-1}, P_{i-1})$
$= (g_i, p_i) \Delta (c_{i-1}, P_{i-1})$
$= (g_i \vee (p_i \wedge c_{i-1}), p_i \wedge P_{i-1})$
$= (c_i, p_i \wedge P_{i-1})$ Q.E.D.

Výpočty operátorem Δ uzávorkujeme do vyváženého stromu, konkrétně výpočet (G_{n-1}, P_{n-1}). Levý a pravý operand každého výskytu Δ můžeme počítat paralelně, protože na sobě nezávisí. Hloubka stromu, tj. počet úrovní je $\lceil \log_2 n \rceil$. Tím získáme mezivýsledky velikosti 2^i.
V druhé fázi spočítáme všechny c_i z mezivýsledků *, v počtu úrovní $\lceil \log n \rceil$, a použijeme je pro určení s_i.

Konvexní obal v rovině

Množina A v R^n je konvexní, pokud $\forall a, b, \in A$ a $\forall t, 0 \leq t \leq 1$, platí, že

$ta + (1-t)b \quad \in A$

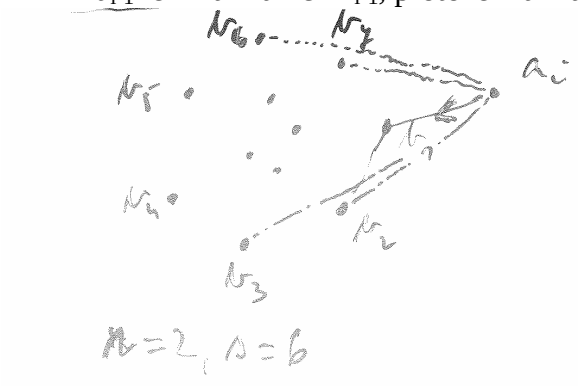
Konvexní obal množiny A je průnik všech konvexních množin v R^n, které obsahují A.

ç dobře definované, protože průnik libovolného systému konvexních množin je konvexní a celý proctor R^n je konvexní.

Úloha: najít konvexní obal konečné množiny A v R^2.
Vstup: $a_1, a_2, \ldots a_n$ jsou prvky množiny A, zadané vzestupně podle x-ové souřadnice
Výstup: body na hranici konvexního obalu, přeházené ve směru hodinových ručiček (posloupnost bodů určuje konvexní obal).

Ukážeme si lineární algoritmus (O(n)) pro konstrukci konvexního obalu (za předpokladu uspořádaného vstupu).
Idea:

Tvoříme konvexní obal množin $A_i = \{a_1, \ldots a_i\}$
Pro A_3 máme trojúhelní a_1, a_2, a_3 nebo a_1, a_3, a_2.
Krok od A_{i-1} k A_i: označíme konvexní obal A_{i-1} jako $v_1, v_2, \ldots v_k$
a_{i-1} leží na hranici A_{i-1}, protože má max. x-ovou souřadnici **è** BÚNO $b_1 = a_{i-1}$



Předpokládáme, že konstruujeme polopřímky a_ib_1, a_ib_2, … a_ib_k.

Jejich směrnice (úhel s osou x), postupně klesá, raste, klesá.

Hledáme b_r a b_s s nejmenší resp. největší směrnicí.

Konvexní obal A_i je a_i, b_r, b_{r+1}, … b_{s-1}, b_s

Hledání r: konstruujeme polopřímky z a_i do bodů $b_1, b_2, \ldots b_r, b_{r+1}$. Po zjištění, že směrnice a_ib_{r+1} stoupla oproti a_ib_r víme, že a_ib_r má minimální směrnici
Hledání s: konstruueme polopřímky z a_i do bodů (b_1), b_k, b_{k-1}, … b_s, b_{s-1}. Analogicky, až směrnice $a_i\, b_{s-1}$ klesne oproti a_ib_s, je b_s hledaný bod s maximální směrnicí a_ib_s.

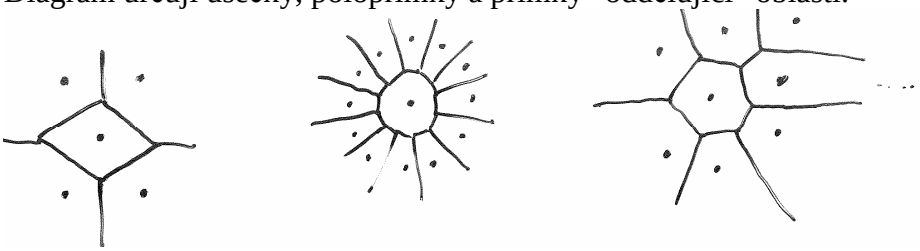
Složitost: lineární k n (poštu bodů A), ale neuvažujeme úvodní trídění podle x-ové souřadnice. Bodu a_i započítáme konstrukci polopřímky do b_r, b_{r+1}, b_s, b_{s-1} a případně a_ia_j pro $j > i$. Ostatní polopřímky z a_o započítáme příslušným bodem b_l, $1 \leq l < r$, $s < l, \leq k$. Body b_l ale vypadnou z konvexn9ch obalů A_i při průchodu od A_{i-1}, takže se jim započítá „do“ nich vedená přímka jedenkrát počas celého algoritmu. Celkem, max. 5 přímek na bod. Q.E.D

Algoritmus s počátečním tříděním má složitost O(n log n), která nejde zlepšit, protože algoritmus hledání konvexního obalu (vrácení cyklického uspořádání) je možné použít pro třídění čísel.
Pro různá reálná řísla $r_1, \ldots r_n$ uvažujeme konvexní obal množiny $\{ [r_1 – r_1^2], [r_2 – r_2^2], \ldots [r_n – r_n^2]\}$. Funkce $y = -x^2$ je konkávní, proto všechny body leží na hranici konvexního obalu a při oběhu ve směru hodinových ručiček je oběhneme v pořadí rostoucích x-ových souřadnic.

Voronoi diagram

Pracujeme v rovině, tj. R^2.
Pro bod $a \in R^2$ značíme jeho x-ovou a y-ovou souřadnici a_x a a_y.Vzdálenost dvou bodů a, $b \in R^2$ je značená $\rho(a, b) = \sqrt{((a_x-b_x)^2 + (a_y-b_y)^2)}$.
Vzdálenost bodu $a \in R^2$ a množiny $A \subseteq R^2$ značíme $\rho(a, A) = \inf_{b \in A} \rho(a, b)$
Speciálně pro A konečné: $\rho(a, A) = \min_{b \in A} \rho(a, b)$

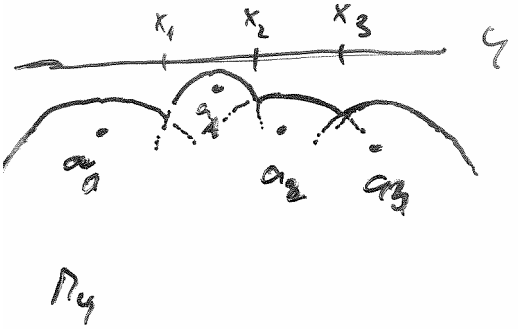
Voronoi diagram konečný množiny M bodů R^2 je zobrazení, které každému prvku a z M (nazývaného místo) přiřadí oblast A(a) bodů u v R^2, které mají k a minimální vzdálenost z celé M, tj. $\rho(u, a) = \rho(u, M)$.
Diagram určují úsečky, polopřímky a přímky "oddělující" oblasti.



Aplikace: klasifikace metodou nejbližšího souseda (1-NN, nearest neighbour)
    například: dobrý/špatný klient při daném přijmu a velikosti ???
Klasický algoritmus: pomocí dělení na poloroviny a následným sloučením
Popíšeme algoritmus pro konstrukci Voronoi diagramu: Fortune (1992)
Využívá zametací přímku (sweep line) – vodorovná,
$\exists y: a_y = y$, y je výška, označímey$_{sw}$,, roste od $-\infty$ k $+\infty$
Vždy konstruujeme (inkrementálně) část VD, kterou je možné určit ze znalosti už zametených bodů, tj. $a_y \leq y_{sw}$.
Idea: body, které jsou bližší k zametenému místu a než k zametací přímce y, se už nezmění (ve VD). Pro pevné y_{sw} a a, množina bodů stejně vzdálených od bodu a a přímky y je parabola s ohniskem a a rovnicí $y = \pi_a$.

$y = \pi_a(x)$
$\pi_a(x) = - ((x-a_x)^2 / 2(y_{sw} - a_y)) + (y_{sw} + a_y)/2$

Parabola je otevřená směrem dolů a body pod ní jsou blíže k a než k y, a tedy než k libovolnému nezametenému místu nad Y.



Označme M_y množina mís tpod Y(a na Y).
Označme U množina bodů pod hranicí a na
U je možné popsat posloupností $x_0, a_0, x_1, a_1, \ldots. x_k, a_k, x_{k+}$, takže

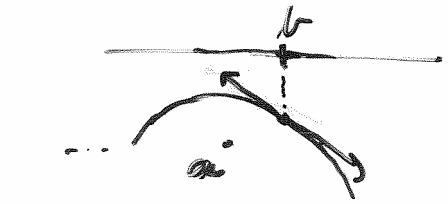
$a_i \in M_y$, $x_0 = -\infty$, $x_{k+1} = +\infty$, $x_1 \leq x_2 \leq \ldots \leq x_k$ (reálné),
$a_{i-1} \neq a_i$, pro $1 \leq i < k$, úsek hranice mezi x_i a x_{i+1} ($x_i \leq x \leq x_{i+1}$) m* nejblíže k bodu a_i
úsek před x_1 ($x \leq x_1$) nejblíže k a_0, úsek za x_k nejblíže k a_k

Konstrukce U, dávková, předpokládáme ∃ místo $a \in M_y$ pod y:
1) žádné místo $a \in M_y$ neleží na y. Nechť X je množina x-ových souřadnic přímky hraničních parabol. X je konečná. Uspořádáme do posloupnosti $x_1 \leq x_2 \leq \ldots \leq x_k$. Doplníme x_0 a x_{k+1}, doplníme a_i pro všechny (x_i, x_{i+1}), a_i jsou určená jednoznačně a jsou zhodné pro celý interval (x_i, x_{i+1}).

2) nějaké místo z M_y leží na přímce y: sestrojíme posloupnost pro množinu mist pod y, podle předpokladu je neprázdná, a následně ji rozšíříme pro $b \in$ y:
2a) jestliže $b \in (x_i, x_{i+1})$, $0 \leq i \leq k$, potom a_i v posloupnosti nahradíme a_i, b_x, b, b_x, a_i
2b) jestliže $b_k = x_i$ pro nějaké i, potom x_i nahradíme x_i, b, x_i

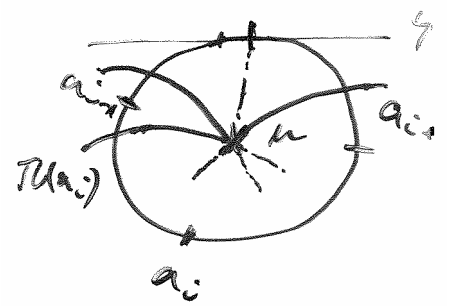
Dopočítání $x_0, \ldots x_{k+1}$ ze znalosti $a_0, \ldots a_k$ je možné.
Mimo krajních x_0 a x_{k+1} je x_i x-ovou souřadnicí průsečíku parabol mist a_{i-1} a a_i. Vybereme jeden z dvou průsečíků, ten, kde vlevo od x_i má parabola a_{i-1} vyšší hodnotu (tj. je bližší k y).
Posloupnost x_0, a_0, x_1, a_1, ... x_k, a_k, x_{k+1} nazveme rozšířená hraniční posloupnost, posloupnost $a_0, a_1, \ldots a_k$ je hraniční posloupnost. Místa se v hraniční posloupnosti vyskytují opakovaně a x-ové souřadnice netvoří obecně monotónní posloupnost.
Body VD (konce segmentů) odpovídají bodům průniku hraničních parabol v okamžiku, kdy se hraniční posloupnost mění (x-ové souřadnice odpovídají číslám z rozšířené hraniční posloupnosti).
Rozšířená hraniční posloupnost se mění se změnou y. Hraniční posloupnost se mění z dvou důvodů.
1) místní událost: přímka Y narazí na nové místo
$b \in M$,platí $b_y = y_{sw}$
2) kružnicová událost: dvě čísla x_i a x_{i+1} v rozšířené hraniční posloupnosti se při pohybu y nahoru ztotožní, následně interval (x_i, x_{i+1}) zmizí a pod podposloupnost
a_{i-1}, x_i, a_i, x_{i+1}, a_{i+1} se změní na a_{i-1}, x_i, a_{i+1}.

ad 1) místní událost: $b \in$ y (& $b \in M$) **è** $b_y = y_{sw}$
nova hraniční posloupnost se tvoří analogicky jako při dávkové konstrukci U.
1a) $b_x \in (x_i, x_{i+1})$ pro nějaké i: a_i ~> a_i, b_x, b, b_x, a_i
z bodu $[b_x, \pi(b_x)]$ se začnou vytvářet 2 nové segmenty v opačném směru, které leží na ose úsečky ab (doprostředka
$\pi(x) = \max_{a \in My} \pi_a(x)$ je hrana U



1b) $b_x = x_i$ pro nějaké i: x_0 ~> x_i, b, x_i
Segment kreslený bodem x_i končí v $[x_i, \pi(x_i)]$ a začnou se vytvářet 2 nové segmenty z tohoto bodu, které mají směr daný osami úseček a_{i-1}b a ba_i

ad 2) bod u = $[x_i, \pi(x_i)]$ patří 3 parabolám, pro a_{i-1}, a_i, a_{i+1}.



Bod u je střed kružnice procházející a_{i-1}, a_i, a_{i+1}, y je tečna kružnice v nejvyšším bodě; bod dotyku $[x_i, y_{sw}]$ je od u vzdálený stejně jako $a_{i\pm1}$. Okamžik události (tj. y_{sw}) je možné přepočítat v momentě, kdy a_{i-1}, a_i, a_{i+1} se ocitli vedle sebe v hraniční posloupnosti .(A)
Segmenty: kreslené body $[x_i, \pi(x_i)]$ a $[x_{i+1}, \pi(x_{i+1})]$ a $[x_{i+1}, \pi(x_{i+1})]$ končí v u a zároveň v u začíná segment se směrem úsečky $a_{i-1}a_{i+1}$

A) použité pro dokazování složitosti algoritmu

Datové struktury:

1) seznam událostí. Tvořený M na začátku a rozšiřovaný o kružnicové udaálosti. Utříděný podle y-ových souřadnic
2) Seznam polopřímek (a vytvořených segmentů)
3) Hraniční posloupnost

Místa se v hraniční posloupnosti mohou opakovat => posloupnost reprezentujeme objekty P_i, položka P_i obsahuje místo z M.
Každému prvku P hraniční posloupnosti (tj. oblouku paraboly) může být přiřazená kružnicová událost P.cEvent, která tento oblouk zaniká.
Polopřímky p_1,… p_i; p_i je kreslená průsečíkem parabol s ohnisky P_{i-1}.site a P_i.site

Algoritmus:

- inicializuj datové struktury
- v cycle zpracovávej události. Při zpracování události
  - najde interval nebo bod podle x-ové souřadnice
  - změní hraniční posloupnost
  - uzavře segmenty a otevře polopřímky
  - vypustí a přidá kružnicové události (pokud jsou “budoucí” a “dobře definované”)
- na konci vypíše neuzavřené segmenty
- ošetření degenerovaných případů (tj. body na kružnici)

Složitost: O(n log n), jestliže vyhledáváme a upravujeme hranovou posloupnost v O(log n)

(B) rozdíl mezi matematickým a programátorským popisem

Převoditelnost problému

Problém P … transformace vstupních dat na výstupní data

f: Σ^* **à** Θ^*

Př. násobení: $\Sigma = \{0, 1, *\}$; $\Theta = \{0, 1\}$

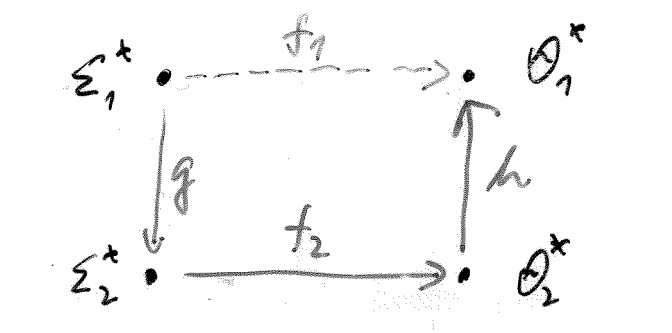
$P_1 \leq P_2$ problém P_1 je převeditelný na problém P_2

f_1(formalizace P_1) : Σ_1^* **à** Θ_1^*

f_2(formalizace P_2) : Σ_2^* **à** Θ_2^*

$\exists g: \Sigma_1^*$ **à** Σ_2^* $\exists h: \Theta_1^*$ **à** Θ_2^*

$\forall x \in \Sigma_1^*$ $f_1(x) = h(f_2(g(x)))$



Př: P ≤ P bez úhmy na obecnosti

Př: Problém hledání kostry je převeditelný na problem hledání minimální kostry

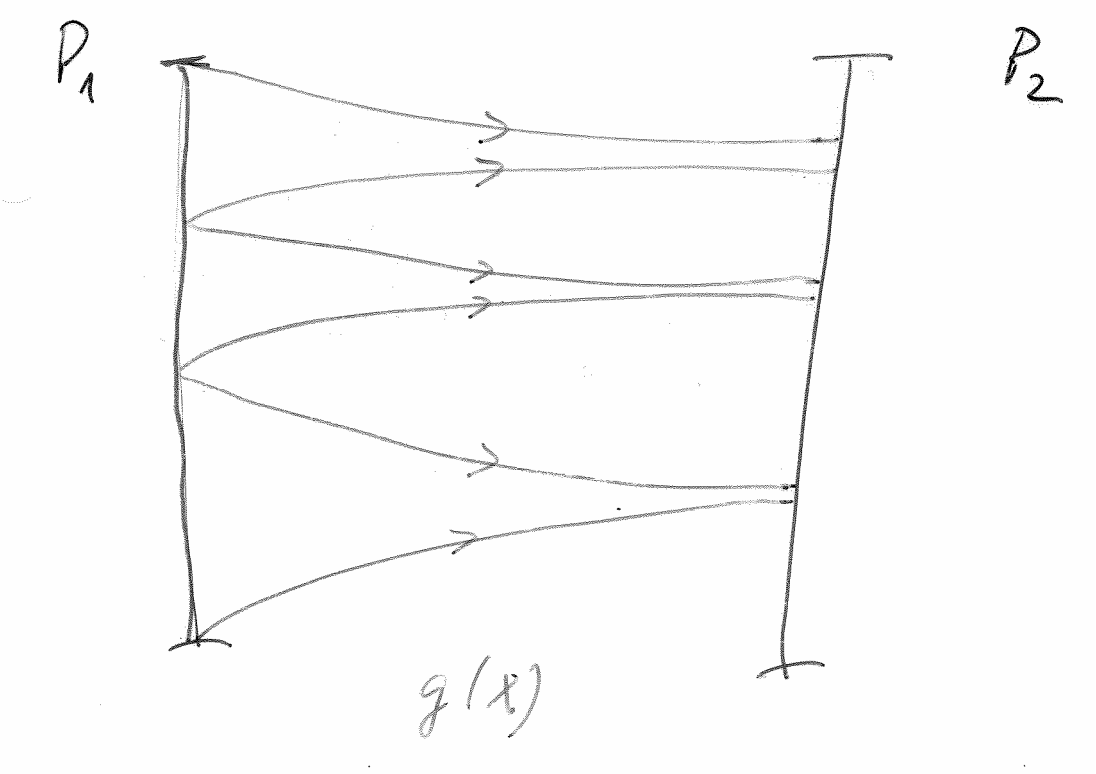
Omezíme se na rozhodovací problémy:

P: f: Σ^* **à** {ano, ne}

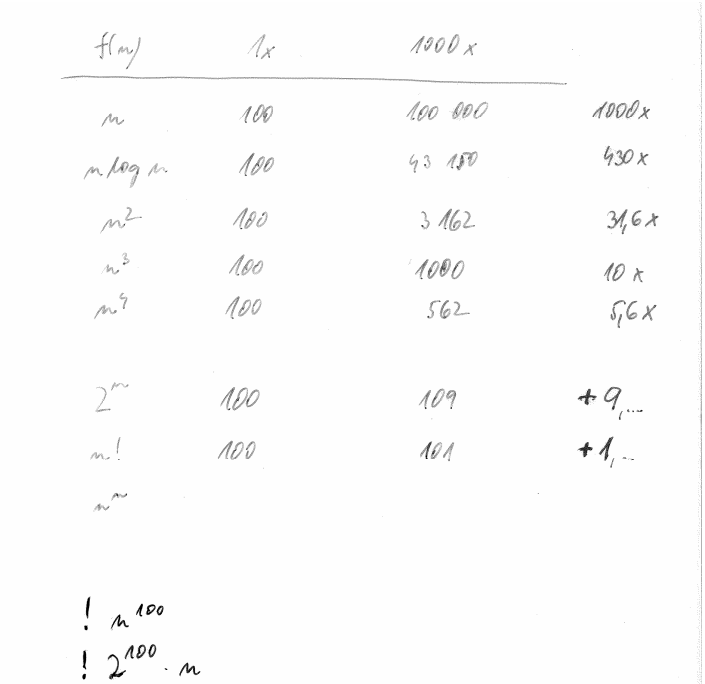
f … χ_p … $P \subseteq \Sigma^*$

$P_1 \leq P_2$ … existuje g(x), tž. x je řešení $P_1 \equiv$ g(x) je řešení P_2

Odpověď P_1 získáme složením g(x) a odpovědi P_2 na g(x)



Úvahy o složitosti



$P_1 \leq_p P_2$ P_1 je polynomiálně převeditelný na P_2
g(x) je $O(|x|^{konst})$

Složitost problému vůči třídě algoritmů
Př.Třída algoritmů s polynomiální složitostí

1.problém (P_1) – Splnitelnost formulí výrokové logiky v konjunktivě normální formy
Syntax:

Formule výrokové logiky jsou:

a) x_i (atomické formule) výrokové proměnné
b) (A).(B) (and)
c) (A) + (B) (or)
d) ($\overline{A}$) (negace – čára nahoře)

Příklad $(X_1) + ((X_2) . ((\overline{X_3}))$ nebo $X_1 + (X_2 . \overline{X_3})$

Sémantika:

pravdivostní ohodnocení
v: výrokové proměnné **à** {T, F}
pravdivostní funkce φ_v: formule **à** {T, F}

φ_v je definované rekurzivně indukcí podle struktury (zápisu) formule:

a) $\varphi_v(x_i)$:= $v(x_i)$
b) $\varphi_v(A.B)$:= $\varphi v(A) \& \varphi_v(B)$
c) $\varphi_v(A+B)$:= $\varphi v(A) \vee \varphi_v(B)$
d) $\varphi_v(\overline{A})$:= $\neg\varphi_v(A)$

Formule A je splnitelná, jestliže existuje φ_v: $\varphi_v(A) = T$
Příklad:

$(X_1 + \overline{X_2}) . (X_2 + \overline{X_3}) . (X_3 + \overline{X_1})$
splnitelná $v(X_1) = v(X_2) = v(X_3) = T$
$(X_1 . \overline{X_1})$ nesplnitelná

! rozlišujte: . + jsou formálně zápisy, reprezentace
& $\vee$ $\neg$ jsou boolovské funkce

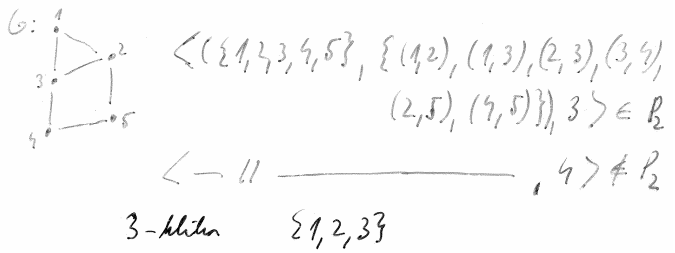
Testování splnitelnosti:
m proměnných … 2^m pravdivostních ohodnocení

Konjunktivní normální forma (K N F)

A je zápis $F_1 . F_2 . \ldots . F_p$
F_i je zápis $L_{i,1} + L_{i,2} + \ldots + L_{i,q0}$ (faktor)
$L_{i,j}$ je v tvaru x_k nebo $\overline{x_k}$
Př. $(X_1 + \overline{X_2}).(X_2 + \overline{X_3})$
$X_1 + \overline{X_2} . X_2 + \overline{X_3}$ priority ‘ ¯ ‘ > ‘+’ > ‘.’
P_1: { zápis(A) | A je splnitelná a A je v KNF)

2.problém (P_2) – problém kliky

graf G=(V, E)
k-klika … úplný podgraf na k vrcholech
P_2: {zápis (G, k) | v grafe G existuje k-kliku}



Převod:
A … $F_1.F_2. \ldots .F_p$
F_i … $L_{i,1} + L_{i,2} + \ldots + L_{i,qi}$ (faktor)
$L_{i,j}$ … X_k nebo $\overline{X_k}$ (literál)

$V = \{<i, j> \mid 1 \le i \le p, 1 \le j \le q_i\}$ odpovídající $L_{i,j}$
$(<i1, j1>, <i2, j2>) \in E \equiv i1 <> i2$ & $L_{i1,j1}$ a $L_{i2,j2}$
není vzájemná negace
mohou být současně pravdivé

A je splnitelné ≡ v(G, E) existuje p-klika
- v každém faktoru F_i je pravdivý aspoň jeden literál $L_{i,j}$ … <i, j> patří do kliky
- převod j polynomiální

Třídy problémů

P (PTIME) prob. řešitelné sekvenčním deterministickým algoritmem v $O(n^{kons})$
NP (NPTIME) prob. řešitelné sekvenčním nedeterministickým algoritmem v $O(n^{konst})$
NP úplné problémy (z NP), na které jsou převoditelné všechny problémy z NP
… a těžší

$P \subseteq NP$, otevřený problem P = NP
NP úplné $\subseteq$ NP

P … považované za efektivně řešitelné
NP … efektivně zkontrolovatelné,
efektivně řešitelné
- heuristikami,
- pro speciální data – očekávaná složitost (simplexova metoda, odvozování typů ve funkcionálních jazycích)
- pravděpodobnostní algoritmy (tradeoff – rychlost za kvalitu/správnost)

jiné NP úplné útoky:
- barvení grafu (3 barvami)
- problém obchodního cestujícího

Turungův stroj: $(\Sigma, Q, \delta, g_0, F)$
Σ … pásková abeceda
Q… množina stavů
$\delta: Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$ … přchodová funkce
$q_0 \in Q$ … počáteční stav
$F \subseteq Q$ množina koncových stavů

Nedeterministický TS: $\delta: Q \times \Sigma \rightarrow P(Q \times \Sigma \times \{L, R\})$
TS má pásku (obousměrně nekonečnou), hlava (čtoucí 1 políčko) a vnitřní stav

CNF Sat: splnitelnost formulí v KNF
KLIKA: pro daný graf G = (V, E) a číslo k, obsahuje G kliku na k vrcholech jako podgraf?

NEZÁVISLÁ MNOŽINA: pro daný graf G = (V, E) a číslo k, obsahuje G k navzájem izolovaných vrcholě?
k-obarvení – pro daný graf G = (V, E) a číslo $k \in N$, existuje obarvení G pomocí nejvíce k barev?
Problém obchodního cestujícího: pro danou množinu měst a vzdáleností najít cestu přes všechny vrcholy s celkovým minimálním součtem (není to rozhodovací formulace)

3-CNFSat: jako CNFSat ale disjunkce mají max. 3 literály
3-obarvení
2-obarvení $\in$ P … je graf bipartitní?

Batoh: pro danou množinu S předmětů, celočíselnou váhová funke w: S **à** N, zisková funkce b: S **à** , limitní váhu $W \in N$ a požadovaný zisk $B \in N$ určit, zda existuje podmnožina $S` \subseteq S$, tž. $\Sigma_{a \in S`} w(a) \leq W$ & $\Sigma_{a \in S`} b(a) \geq B$
Subret Sum: pro danou množinu S čísel z N a číslo B existuje podmnožina $S` \subseteq S$,
tž. $\Sigma_{x \in S`} = B$? (~$\lceil \log B \rceil$ bitů)
Rozdělení: Pro danou množinu S čísel, existuje $S` \subseteq S$, tž. $\Sigma_{x \in S`} x = \Sigma_{x \in S \setminus S`} x$
Platí:
Rozdělení $\subseteq_p$ SubsetSum: $B = ½ \Sigma_{x \in S} x$
SubsetSum $\subseteq_p$ Rozdělení: prvky N-B a N-(Σ-B)

Aproximační algoritmy

- pro NP-úplné úlohy chceme v polynomiálním čase „aspoň“ přibližné řešení, ale „dostatečně“ přesně.

Předpokládejme, že pracujeme na optimalizačních problémech, kde každé řešení má kladnou cenu a chceme najít skoro optimální řešení (maximum alebo minimum).

Příklad:

najít maximální kliku
najít obarvení minimálním počtem barev
najít nejkratší cestu obchodního cestujícího
najít největší součet podmnožiny nepřevyšující dané b

Definice: poměrná chyba

Nechť C* je cena optimálního řešení. Aproximační algoritmus pro problém má poměrnou chyb $\delta(n)$, jestliže pro každý vstup velikosti n cena C řešení vydaného aproximačním algoritmem splňuje $\max(C/C^*, C^*/C) \leq \delta(n)$.

Definice: analogicky, algoritmus má relativní chybu $\varepsilon(n)$, jestliže $|C-C^*|/C^* \leq \varepsilon(n)$.

Platí: $\varepsilon(n) \leq \delta(n) - 1$, pro max. problémy $\varepsilon(n) = (\delta(n)-1) / \delta(n)$

Jestliže je chyba algoritmu pevná, tj. nezávislá na n, píšeme δ a ε.

Idea: aproximační algoritmy, které dosahují postupně menší poměrnou (???) chybu při použití vice času

Definice: aproximační schéma pro optimalizační problém je aproximační algoritmus, který dostává instanci problému a $\varepsilon > 0$, a pro každé pevné ε algoritmus počítá s relativní chybou ε.

Aproximační schéma je polynomiální, pokud, pokud pro pevné $\varepsilon > 0$ algoritmus běží v polynomiálním čase vzhledem k velikosti vstupu n.

Definice: aproximační schéma je úplné polynomiální, jestliže je časová složitost polynomiální v $1/\varepsilon$ a n, kde n je velikost vstupu a ε je relativní chyba.

Ukážeme si:

- aproximační algoritmus pro vrchovolé pokrytí s poměrnou chybou 2
- aproximační algoritmus pro problem obchodního cestujícího s trojúhelníkovou nerovností s poměrnou chybou 2 (apakování)
- neexistenci polynomiálního apoximačního algoritmu pro obecný problem obchodního cestujícího
- úplné polynomiální aproximační schéma pro problem součtu podmnožiny

Příklad: Schéma polynomiálních algoritmů, které pro pevné k řeší problem k-kliky na daném grafu G.

- generuj k vložených cyklů přes vrcholy, ve vnitřním otestuj, zda jsou vrhocly různé a tvoří k-kliku
- pro pevné k je algoritmus polynominální ($O(n^k)$), ale obecně pro paremetrem dané k je úloha NP-úplná.

Vrcholové pokrytí grafu G = (V, E) je podmnožina $V` \subseteq V$, taková, že pro každou hranu (u, v) $\in$ E platí $\{u, v\} \cap V` \neq 0$ (tj. $u \in V`$ nebo $v \in V`$). Velikost vrcholového pokrytí je $|V`|$.
Problém vrcholového pokrytí je najít pro daný neorientovaný graf brcholové pokrytí minimální velikosti.

Tvrzení: Problém vrcholového pokrytí je NP-úplný.
Aproximační algoritmus pro vrchovolé pokrytí s poměrnou charakteristikou 2.
Vstup: G=(V, E)

1. C := ∅
2. E` := E;
3. while E` ≠ 0
4. zvol libovolnou hranu (u, v) ∈ E`
5. C := C ∪ {u, v}
6. vypusť z E` věechny hrany incidentní s u nebo v
7. od
8. return C

- C je vrcholové pokrytí, protože každá hrana E je pokrytá
- chceme ukázat, že algoritmus má poměrnou chybu 2: uvažuejem všechny hrany zvolené na řádku 4, označíme jako A
  Žádné dvě hrany A nesdílejí vrchol, protože hrany incidentní s vybranou hranou jsou vypuštěné na řádku 6.
  Z toho plyne, že |C| = 2|A|.
  Abychom pokryli hrany |A|, potom každé vrcholové pokrytí včeně optimálního C*, musí zahrnout aspoň jeden vrchol každé hrany A. Protože hrany A nezdílí vrcholy, platí |A| ≤ |C*| a odtuď |C| ≤ Z |C*|, Q.E.D

DC: Hladový algoritmus vybírá vrchol největšího stupně nezaručí poměrnou chybu 2.

Problém obchodního cestujícího

Daný neorientovaný graf G = (V, E) a nezáporná celočíselná cena c(u, v) pro každou hranu (u, v) ∈ E. Hledáme hamiltonský cyklus s nejmenší cenou.
Trojúhelníková nerovnost: funkce c splní trojúhelníkovou nerovnost, když pro každé vrcholy u, v, w ∈ V platí c(u,w) ≤ c(u, v) + c(v, w)
Algoritmus přibližného řešení problému obchodního cestujícího s Δ-nerovností v polynomiálním čase s poměrnou chybou 2:
1. zvolíme vrchol r ∈ V -- r je kořen
2. najdeme minimální kostru v G s cenou c
3. nechť L je seznam vrcholů v pořadí preorder při procházení kostry z vrhovou r
4. return hamiltonský cyklus H, který prochází vrcholy v pořadí daném L

Idea důkazu (opakování) … |_| značí cenu _

- kostra K je obsáhnutá v H*, proto |K*| ≤ |H*|
- úplná cesta se zaznamenáváním vrcholů pro
- vypuštěním vrcholů z úplné cesty cenu cesty zkracujeme, ale platí Δ nerovnost

Poznámka: Získaná kružnice je horní odhad optima, který můžeme vylepšovat lokálně: odstranění křížení, přeuspořádání k-tic vrcholů.

Věta: Když P ≠ NP a ρ ≥ 1, potom neexistuje polynomiální aproximační algoritmus pro problem obchodního cestujícího s poměrnou chybou ρ.
Důkaz: Sporem; ukážeme, že jestliže existuje algoritmus A z věty, potom se dá použít na řešení problému hamiltonské kružnice, který je NP.
Nechť G = (V, E) je instance problému hamiltonské kružnice. Transponujeme G na instance POC takto: G´ = (V, E´) je úplný graf na V, tj. E`= {(u, v), u, v ∈ V & u ≠ v}
c(u, v) = 1 jestliže (u, v) ∈ E
= ρ |V| + 1 jinak

Vytvoření G`a c je polynomiální v |V| a |E|. Uvažujeme o instanci POC (G´, c). Jestliže původní G má hamiltonský cyklus H, potom všechny hrany H mají cenu 1 a (G`, c) obsahuje cyklus ceny |V|. Jestliže G nemá hamiltonský cyklus, potom každý cyklus v G` obsahuje hranu mimo E a cena cyklu bude aspoň $(\rho.|V| + 1) + (|V| - 1) > \rho.|V|$
Protože hrany mimo E jsou drahé, je velký rozdíl mezi cenou hamiltonského cyklu v G (cena |V|) a libovolného jiného cyklu (cena aspoň $\rho.|V|$).
Aproximační algoritmus A musí vrátit hamiltonský cyklus, ale v G existuje, protože při požadované chybě ρ nezná jinou možnost.
Jestliže hamiltonský cyklus neexistuje, algoritmus vrátí cyklus ceny aspoň $\rho.|V|$.
Teda, pomocí A jsme vyřešili problem hamiltonské kružnice.

Úplné polynomiální aproximační schéma pro součet podmnožiny

Problém: (S, t), kde S je $\{a_1, a_2, \dots a_n\}$, množina kladných přirozených čísel a t je kladné přirozené číslo.
Rozhodovací verze: existuje podmnožina $S` \subseteq S$, tž. $\Sigma_{a_i \in S} a_i = t$
Optimalizační verze: hledáme podmnožinu $S` \subseteq S$, jejíž součet je co největší, ale nepřevyšuje t.

Značení: $S .+. x = \{s + x, s \in S\}$pro množinu S i seznam S

```
Algoritmus: Součet_podmnožiny_přesně(S, t)
  1. u := (S)
  2. L0 := <0>
  3. for i := 1 to n do
  4.      Li := merge_list(Li-1, Li-1 .+. ai)
  5.      vypusť z Li všechny prvky větší než t
  6. return maximum z Li
```

merge_list sloučí uspořádané seznamy do uspořádaného seznamu
délka L_i je až 2^i, tj. algoritmus je exponenciální (v obecnosti)

Aproximační schéma:
Idea:

- každý seznam L_i po vytvoření „zkrátíme“.
- používáme parametr δ, $0 \le \delta < 1$
- zkrátit seznam L znamená vypustit co nejvíc prvků L, tak že pro každý vypuštěný prvek y zůstal v seznamu L` prvek z ≤ y, takový, že $(y-z)/y \le \delta$, tj. $(1-\delta)z \le y \le z$
  Prvek z je reprezentant y s „dostatečně malou chybou“.

```
Algoritmus: součet_podmnožiny_aprox(S, t, ε)
  1. n := |S|
  2. L0 := <0>
  3. for i := 1 to n do
  4.      Li := merge_lists(Li-1, Li-1 .+. ai)
  5.      Li := zkrať(Li, ε/n)
  6.      odstraň z Li všechny prvky větší než t
  7. nechť z je největší hodnota v Ln
  8. return z
```

- prvky L_i jsou součty podmnožin
- chceme : $C^*(1-\varepsilon) \le C$ pro cenu C nalezeného v C* ?????

- v každém kroku ??????? chybu ε/n, indukcí podle i se dá dokázat, že pro každý prvek $y \leq t$ z nezkrácené verze existuje $z \in L_i$ tž. $(1-\varepsilon/n)^i y \leq z \leq y$
- pro optimum y* existuje $z \in L_n$ tž. $(1-\varepsilon/n)^n y^* \leq z \leq y^*$ protože $1-\varepsilon \leq (1-\varepsilon/n)^n$ **è** $(1-\varepsilon)y^* \leq z$
- ještě úplná polynomiálnost - Idea: relativní chyba ε/n rozsah 1.. t rozdělí na polynomiální úseky, v každém je ≤ 2 reprezentantů.

Šifrování

Rozšířený euklidův algoritmus

VSTUP: a, b {a>0 & b>0}
VÝSTUP: d, x, y {d=(a, b)=a*x + b*y}
d největší společný dělitel, x, y jsou konstruktivně potvrzení, že d je lineární kombinace a, b

```
procedure Euklid(a, b: integer; var d, x, y: integer);
var d1, x1, y1: integer;
{ a > 0 & b > 0 }
begin
     d = 0; x = 1; y = 0;          {r je dělitel a, b
     d1 = b; x1 = 0; y1 = 1;            <=> r je dělitel d, d1}
     repeat { ∀r( r|a & r|b <=> r|d & r|d1) &
               d = a*x + b*y  &
               d1 = a*x1 + b*y1 }
          if d > d1 then
               x := x – x1 * (d div d1);
               y := y – y1 * (d div d1);
               d := d mod d1;
          else
               x1 := x1 – x(d1 div d);
               y1 := y1 – y(d1 div d);
               d1 := d1 mod d;
          fi
     until d = 0 or d1 = 0
     if d = 0 then d := d1; x := x1; y := y1; fi
end
```

Podmínka: P(a, b, d, x, y, d1, x1, y1) ≡
{ ∀r(r|a & r|b <=> r|d & r|d1) &
d = a*x + b*y & d1 = a*x1 + b*y1 }

```
if d > d1 then
      x̄ := x – x1*(d div d1);
      ȳ := y – y1*(d div d1);
      d̄ := d mod d1;
x̄, d̄, d̄ – nové hodnoty proměnných x, y, d
```

Chceme ukázat P(a, b, $\bar{d}$, $\bar{x}$, $\bar{y}$, d1, x1, y1)
Nechť s = d div 1; t = d mod d1; d = s*d1 + t
(1) ? r | $\bar{d}$ & r | d1 <=> r | d & r | d1 ∀r
<= z dělitelnosti ∃ k, k1 r*k = s*r*k1 + $\bar{d}$
r(k – s * k1) = $\bar{d}$

„->“ ∃k1, $\bar{k}$ d= s * r * k1 + r * $\bar{k}$ = r*(…)
(2)? $\bar{d}$ = a* $\bar{x}$ + b * $\bar{x}$
t = a*(x – x1 * s) + b(y – y1 * s)
t = a*x + b*x – (a*x1*s + b*y1*s)

```
s*(a*x1 + b*y1) + t = d
s*d1 + t = d

↓ odvození          ↑ důkaz
```

Relace kongruence modulo m

a≡b (mod m) <=> m | a-b
≡ (mod m) je ekvivalentní; zachovává +, *
Z_m = Z / ≡ (mod m) m tříd rozkladů

reprezentanti tříd: {0, …, m-1}
$<a>_m + <b>_m = <a+b>_m$
$<a>_m * <b>_m = <a*b>_m$
$<a>_m$ = a mod m
$<0>_m$ … nula; $<1>_m$ … jednotka
--- okruh zbytkových tříd modulo m

Řešitelnost lineárních rovnic v Z_m
(1) $<a>_m * <x>_m = <b>_m$ $<a>_m \neq <0>_m$
eq. a*x ≡ b (mod m); m nedělí a lin. kongruence
eq. a*x + m*y = b; m nedělí a lin. ??????? rovnice

a) $<b>_m = <0>_m$ (tj.: m|b)
(1) má nenulové řešení <=> d = (a, m) > 1
x= m/d (<=> a je dělitel nuly)
b) $<b>_m \neq <0>_m$
(1) je řešitelná <=> (a, m) | b
speciálně $<a>_m * <x>_m = <1>_m$ <=> (a, m) = 1
$<x>_m$ … jediné … $<a>_m^{-1}$ … vypočítatelné EA

Kryptografie

- přenášíme řetězec bitů … číslo ∈ {0, … M}
M … řádově stovky bitů …. $M \approx 2^{100}$
šifra e: {0..M} **à** {0..N}
klíč d: {0..N} **à** {0..M}
platí: d(e(m)) = m ∀m ∈ {0..M}

Komutující šifry:

e_A, e_B na {0..K} komutují:
∀ m ≤ k je $e_A(m)$, $e_B(m) \leq M$ a platí
$e_A(e_B(m)) = e_B(e_A(m))$

Protokol:

A: zamkne zprávu m a pošle $e_A(m)$
B: přidá svůj zámek a vrátí $e_B(e_A(m))$

A: odemkne svůj zámek: $d_A(e_B(e_A(m))) = d_A(e_A(e_B(m))) = e_B(m)$
B: odemkne d_B: $d_B(e_B(m)) = m$
Při každém přnose je správa chráněná jedním zámkem

Poker po telefonu:

A: (míchá karty) zašifruje čísla kongruentní s 1, …, Ω a pošle $e_A(\overline{1}),\ldots e_A(\overline{\Omega})$ v náhodném pořadí
B: (rozdává pro A) vybere 5 karet $e_A(a_1)$, …$e_A(a_5)$ a pošle je
A: (má svoje karty) dešifruje $d_A(e_A(a_i)) = a_i$
B: (rozdává pro sebe) vybere $e_A(b_1)$ … $e_A(b_5)$, zašifruje a pošle $e_B(e_A(b_1))$, … $e_B(e_A(b_5))$
A: dešifruje karty pro B: $e_B(b_1)$, … $e_B(b_5)$
B: dešifruje a pozná svoje: $d_B(e_A(b_i)) = b_i$

Jestliže je možné šifry e_A, e_B zveřejnit, příčem klíče (inverzní k šífrám) ostatní "těžko" spočitatelné, máme možnost kontroly

…
A: (vyhledá svoje karty) pošle a_1, … a_5
B: (kontroluje) zná e_A, ale ne d_A
počítá $e_A(a_1)$, … $e_A(a_5)$ a porovná s rozdanými.
A, B, znají jen "svou" část informace

Veřejné kryptografické systémy

Zobrazení:
e: {0..M} **à** {0..N}
d: {0..N} **à** {0..M}
tvoří Diffie-Hellmannův pár, jestliže
a) d je zleva inverzní k e {d(e(m)) = m}
b) d, e jsou efektivně vyčíslitelné, ale z algoritmu realizujícího e není možné efektivně odvodit algoritmus realizující d
„padací dvířka" – příklad: telefonní seznam

Každý účastník svůj D-H pár e, d Šifry e veřejné, klíče d tajné.
Navíc, když
a´) d_A oboustranně inverzní k e_A
možnost ověřitelných elektronických dopisů

Protokol

A: vyhledá e_B pro adresáta B a pošle
e_B("Správa. Zdraví A. d_A (podpis)")
B: dešifruje a čte
„Správa. Zdraví A. bla bla ba"
najde šifru pro A a ověří: e_A(bla bla bla) = podpis

- šifry nemusí komutovat
- vystřihnutí podpisu a poslání falešně zprávy
(B **à**)C: e_c("Text. Od A. d_A(podpis)")

řešení: jiné kódování: A **à** B: e_B(„od A. d_a(A pro B, ZPRÁVA, 19.5.1994, PODPIS)“)

Šifrování založené na problému batohu

Problém batohu:
Je dané n, a_1, … a_n (váha předmětu)
a číslo b (váha batohu)
Máme určit x_1, … x_n; x_i = 0, 1, pro i = 1..n, tž. $b = a_1x_1 + \ldots + a_nx_n$

Jednoduché pro superrostoucí posloupnost:
$_{k=1}\Sigma^{i}\ a_k < a_{i+1}$ pro $\forall i,\ 1 \le i \le n-1$
Příklad: Výstuní lineární konverze: $a_i = 2^{i-1}$

Konstrukce D-H páru

Zvolíme $\{c_i\}_{i=1..m}$ superrostoucí, modul $r \ge 2*c_n$ a číslo s neousdělné s r. Spočítáme $t=<s>_r^{-1}$.
Zveřejníme $b_i = <s.c_i>_r$ a předpis šifry e(x): číslo x s binárním zápisem $x_n\ x_{n-1}$ .. x_1 se šifruje
$e(x) = x_1 * b_1 + \ldots + x_n * b_n$

Dešifrování: y **à** $<t.y>_r$ převede těžký problém batohu s koeficienty $\{b_i\}$ na jednoduchý s koeficienty$\{c_i\}$
$<t.e(x)>_r = <t.\ _{i=1}\Sigma^{n}\ x_i\ b_i>_r = <_{i=1}\Sigma^{n}\ x_i\ t\ b_i>_r = <_{i=1}\Sigma^{n}\ x_i\ t\ s\ c_i>_r = <_{i=1}\ \Sigma^{\ n}\ x_i\ c_i>_r$

Šifra RSA (Rivest, Shamir, Adleman)

komutující šifra s oboustranným inverzním klíčem
Věta (Malá Fermatova): Jestliže m je prvočíslo a a není násobkem m (m nedělí a),
potom $a^{m-1} \equiv 1 \pmod m$
Důkaz: Eulerova funkce Φ(n) je pro n>1 počet kladných celých čísel menších než n, nesoudělných s n.

Věta:
Jestliže n je prvočíslo, potom Φ(n) = n-1
Jestliže n = p.q, kde p, q jsou různá prvočísla, potom Φ(n) = (p-1)(q-1)

Věta (Eulerova věta): pro a, n nesoudělná (tj: (a, n) = 1) platí $a^{\Phi(n)} \equiv 1 \pmod n$
Důsledek: Jestliže (a, n) = 1, potom $<a>^{-1}_n = <a^{\Phi(n)-1}>_n$
(jestliže znám rozklad n na prvočísla, rychle spočítám Φ(n) a následně inv. prvek)

RSA: Volíme velké prvočísla p, q. Spočítáme n = p*q a hodnoty Eulerovy funkce
r = (p-1)(q-1).
Volíme velká číslo k nesoudělné s r a spočítáme inverzní prvek $g = <k>^{-1}_r$
Veřejná část: k, n; tajná část g, p, q
Šifrování: M ≤ n-1: $e(M) = <M^k>n = H$
Dešifrování: $d(H) = <H^g>_m$
Správnost dešifrování: $d(e(M)) = <H^g>_m = <M^{kg}>_n = <M^{cr+1}>_n = <M*M^{c\Phi(n)}>_n$
$= <M*1>_n = <M>_n = M$
$cr+1 = gk \equiv 1 \bmod r$

Příklad: p = 47, q = 71 **è** n = p*q = 3337, r = (p-1)(q-1)=3320, volíme k = 79 **è**
g = $<79>^{-1}_{3320}$ = 1019
Zpráva M = 688 e(M) = $<M^k>_m$ = $<688^{79}>_{3337}$ = 1570 = H

Bezpečnost šifrování – praktické poznámky

- problém, na kterém je šifra postavená (např. faktorizace, problém batohu) musí být těžko řešitelná i v průměrném případ (ne jen v nejhorším – ne všechny NP úplné problémy vyhovují)
- nemá existovat „rychlý“ algoritmus na „dostatečně“ přibližné řešní, které umožní získat přibližné znění zprávy
- dostatečná délka klíče, resp. prvočísla
  faktorizace:
  1975 … 39 (decimální) cifer
  1985 … 65
  1994 … 130
  (pomocí pravděpodobnostního algoritmu)
- uhádnutí části zprávy (pomocí externích zdrojů a souvislostí) pomáhá rozluštění zbytku – pokud: dělitelnost (mod Ω)
- spolehlivost lidí, …

jiné požadavky na protokoly
- ověřitelnost odesilatele (kdo to poslal?)
- ověřitelnost příjemce (komu patří tento veřejný klíč)
- potvrzení doručení – do určitého času
- zachování pořadí zpráv
- kontrola autenticity textu (dostali jsme, co bylo poslené?)

elektronické peníze:
- bezpečnost: nemožnost kopírovaní, změn, opakovaného utracení
- soukromí, anonymita, ne trasovatelnost
- (-off-line platby, transitivita, dělitelnost)