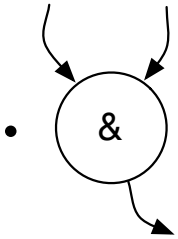
- 07/10/02
  - Co budeme dělat
    - Grafové algoritmy, toky
    - FFT
    - řetězce
    - geometrie
    - paralelní
    - pravěpodobnostní
    - aproximační (ne heuristika, ale dokážeme si, jak moc špatné řešení umíme apod.)
    - trochu kryptografie
    - trochu NP-úplnosti
    - (možná něco navíc? hehe…)
  - Cvičení
    - Doporučuje se tam chodit kvůli mechanickým důkazům:/
    - Zápočet od cvičícího za napsání zápočtové práce.
      - Vypracování (naprogramování) algoritmu a napsání krátkého textu o něm.
  - Doporučená literatura
    - mj.ucw.cz—ads2
    - CLRS
    - Demel: Grafy a jejich aplikace
    - Kučera a Nešetřil: Algebraické metody diskrétní matematiky
      - Paralelní, FFT
    - Kuchařka KSP – hlavně základní věci a řetězce (link z webu)
    - M. M.: Krajinou grafových algoritmů (1. kapitola, najdeme ji i na webu **mj**)
  - Paralelní algoritmy
    - hradlová síť
      - booleovské obvody
      - kombinační obvody
    - hradlo deterministicky přiřazuje k-tici vstupů výstup (booleovské: bool. operátory, komb.: kombinace písmen z abecedy –> písmeno z abecedy)
    - 

    - vstupní a výstupní porty (nemají žádné vstupy/výstupy – prázdná kolečka)
    - budeme předpokládat max. 2 vstupy na hradlo (což, jak se ukáže, stačí)
    - Definice:
      - Booleovský obvod je acyklický orientovaný graf s vrcholy $V = I \cup O \cup H$
      - $\forall v \in I$: deg^+(v) = 0
      - $\forall v \in O$: deg^-(v) = 0, deg^+(v) = 1
      - $\forall v \in H: \exists$ f(v): {0, 1}^a(v)
        - f: funkce vykonávaná
        - a(v): arita
        - deg^+(v)=a(v), hrany vstupující do v jsou očíslovány 1, …, a(v)
        - Navíc $\forall v \in H$ $a(v) \leq 2$. (Lze i $0 \rightarrow$ konstanta)
    - Definice:

        - *Výpočet* sítě probíhá po taktech. V 0. taktu jsou definované právě všechny vstupy. V i-tém taktu vydají výsledek, která měla def. vstupy v (i-1)-tém taktu, když jsou def. hodnoty všech hradel a portů, síť se zastaví a vydá výsledek.
    - Definice:
        - i-tá *vrstva* = vrcholy v takové, že max d(w,v) = i, w ∈ I
    - Definice:
        - *Časová složitost sítě* = # vrstev
        - *Prostorová složitost sítě* = # hradel
    - Podmínka: Vygenerovatelné v O(n^2) vzhledem k velikosti vstupů. (Jinak je to moc ďábelské…)
    - Vztah k logickým formulím
        - Vždy lze udělat síť odpovídající logické formuli, která má časovou složitost odpovídající počtu závorek (i implicitních) a prostorovou odpovídající počtu spojek.
        - Platí to i opačně (ale asi by musela být formule pro každý výstup, nebo by se muselo „předávat“ číslo výstupního portu).
    - Sčítačka
        - sčítám x_(n-1)…x_0 + y_(n-1)…y_0 = z_(n-1)…z_0
        - přenos z i-tého řádu: c_i (vč. z minus první pozice = 0)
        - z_i = xi XOR yi XOR c_(i-1)
        - c_i = (x_i & y_i) ∨ ((x_i ∨ y_i) & c_(i-1))
        - Sakra: Lineární časová složitost.
        - Podíváme se na bloky (k-tice míst) vstupu, a co dělají s přenosy:
            - vždy 0
            - vždy 1
            - = vstupní přenos
            - (= ! vstupní přenost (nejde))
        - Skládání
            - p q b (složený)
            - 0 * 0
            - 1 * 1
            - → 0 1
            - → 1 1
            - → → →
        - Jednoduché bloky
            - 0/0 0
            - 1/1 1
            - 1/0 →
            - 0/1 →
        - Pozorování: Skládání bloků je asociativní. Proto můžeme přenosy spočítat v O(log n).
        - (a,x):
            - (1, *) = → (kopírování)
            - (0, 0) = 0
            - (0, 1) = 1
            - (a, x) fň (b, y) = (c, z)
            - c = & b
            - z = (¬a & x) ∨ (a&y)
        - Ouha! Potřebujeme ale O(n^2) hradel. Ale nic se neděje, během posledních pět minut si vysvětlíme jak totéž provést s pouhopouhými O(n) hradly!
        - Jak na to:
            - V prvních O(log n) hladinách O(n) hradel. Pro bloky velikosti 2^k (∀ k ∈N) na pozcii dělitelné 2^k víme chování. Stačí to zahustit v O(log n) hladinách s O(n) hradly a pak posčítat zbytek. Tradá.
        - Cvičení do tramvaje: Namalovat schéma pro např. 8 míst. ='
- Přednášející: Martin Mareš mj@ucw.cz
- 07/10/09
    - Toky v sítích

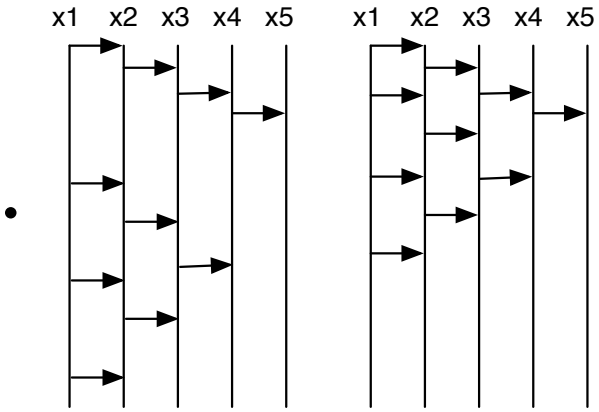
- Definice: *Síť* je čtveřice (G, z, s, c), G orientovaný graf, z, s (resp. s, t) jsou vrcholy zdroj/stok, c: E(G) → **R**_0^+
- Definice: *Tok* je funkce E(G) → **R** taková, že
  - 1)
  - 2)
- Věta: Každá síť má maximální tok.
- Definice:
  - *Řez* R v síti (G, z, s, c) je R ⊆ E(G), taková, že neexistuje cesta ze z do s, která by přes něj nevedla.
  - Kapacita řezu c(R) – součet kapacit hran v řezu.
- Hlavní věta o tocích
  - f: tok
  - max W(f) = min c(R)
- Lemma
  - A⊆V(G), z ∈A, s∉A, f je tok, potom: w(f) = f(A, V - A) - f(V - A, A).
  - Důkaz
    - $\sum\_(u,v) \in E$ f(u,v) - $\sum_{(v,u) \in E}$ f(v,u) = 0, ∀u∈A, u≠z,s (Kirchhofovy zákony)
- NIC NECHÁPU!
- KONEC

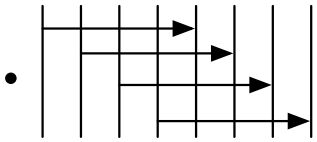
- 07/10/23
  - Definice. Funkce f: E → $\mathbf{R}_0^+$ je *vlna* v síti (V,E,z,s,c) ⇔ ∀e∈E f(e) ≤ c(e) & ∀v≠z,s $f^{\Delta}(v) \geq 0$.
  - f^Δ(v) je *přebytek* ve v
  - f^Δ(v) = - $\sum_{u:\ uv\ \in E}$ f(uv) - $\sum_{u:\ vu\ \in\ E}$ f(vu)
  - Vrcholům přiřadíme výšky h: V→**N**
  - Operace:
    - *Převedení přebytku*:
      - Předpoklady
        - u: $f^{\Delta}(u) > 0$
        - v: h(u) > h(v)
        - r(uv) > 0
      - Převedeme tok velikosti δ := min ($f^{\Delta}(u)$, r(uv)) z *u* do *v*. fΔ(u) klesne o δ, $f^{\Delta}(v)$ stoupne o δ
    - *Zvednutí vrcholu* u*: h*(*u*)++
  - Poznámka: Přebytky *z* a *s* nás nezajímají.
  - Algoritmus:
    - h(* = všechny vrcholy) := 0, h(z) := N (počet vrcholů)
    - f(*) := 0, ∀ u: zu∈ E f(zu) := c(zu)
    - dokud ∃u ≠z,s; $f^{\Delta}(u) > 0$:
      - pokud ∃uv ∈E r(uv) > 0 & h(u) > h(v)
        - převedeme po uv
      - jinak
        - zvedneme u
    - vrátíme f jako výsledek
  - Invariant A: f je vlna, ∀v: h(v) neklesá, h(z) = N, h(s) = 0
  - Invariant S (spád): ∀uv∈E r(uv) > 0 h(u) ≤ h(v) + 1
  - Lemma K (korektnost): Když se algoritmus zastaví, f je max. tok.
  - Dk:
    - 1) f je tok
    - 2) ∃ nenasycená cesta P ze z do s
      - P má spád N a ≤ N-1 hran ⇒ ∃ e∈P se spádem ≥ 2 ⇒ spor s invariantem S
  - Invariant C (cesta domů, resp. do zdroje): Je-li v ≠z,s $f^{\Delta}(v) > 0$, pak ∃ nenasycená cesta v→z.
    - Dk:
      - Mějme v: $f^{\Delta}(v) > 0$

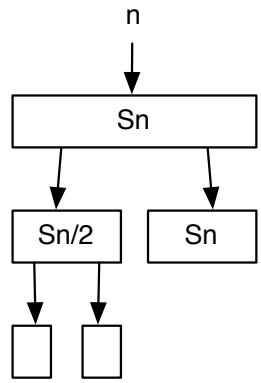
- A := {u | ∃ nenasycená cesta v → u}
- Pokud a ∈ A, b ∈ V - A, ba ∈ E ⇒ f(ba) = 0 /* jinak by r(ab) > 0 a tedy b∈A */
- $\sum_{u \in A} f^{\Delta}(u) = \sum_{ab \in E \cap ((V-A)\times A)} f(ab) - \sum_{ba \in E \cap (A\times(V-A))} f(ba) \leq 0$ (lze snadno rozmyslet proč ≤ 0)
  - ⇒ ∃u∈A: $f^{\Delta}(u) < 0$ ⇒ u (nutně) = z (jinak by to nebyla vlna) ⇒ z∈ A

- Invariant V (výška): ∀v: h(v) ≤ 2N
  - Dk (sporem): Zvedneme v: h(v) ≥ 2N, tzn. $f^{\Delta}(v) > 0$ ⇒ (podle invariantu X) ∃ cesta v→z nenasycená se spádem ≥ N – Blé! Spor!
- Lemma o zvedání: #zvednutí ≤ $2N^2$
  - Lemma S (sytá převedení): #SP ≤ 2NM, resp. NM (pokud předpokládáme symetrii grafu, tedy zpětné hrany)
    - uv ∈ E
    - h(u) > h(v): r(uv) = 0
    - (≥ 2 zvednutí v)
    - h(v) > h(u): r(uv) > 0
    - (≥ 2 zvednutí u)
    - h(u) > h: (?)
  - Lemma N (nenasycená převedení): #NP = $0(N^2M)$
    - Důkaz (*archetypální potenciálový důkaz)*
      - Pomocí potenciálu *fí* = $\sum_{v:f^{\Delta}(v) > 0,\ v\neq z,s} h(v)$.
        - fí ≥ 0
        - na počátku fí = 0
        - zvednutí zvýší fí o 1 × ≤ $2N^2$ → ≤ $2N^2$
        - SP zvýší fí o ≤ 2N × ≤ NM → ≤ $2N^2M$
        - NP sníží fí o ≥ 1 ⇒ všech NP ≤ $2N^2 + 2N^2N = O(N^2M)$
- Věta: Golbdbergův algoritmus najde maximální tok v čase $O(N^2M) = 2N^2O(N)+O(NM)+O(N^2M)$
- Implementace:
  - P := seznam všech v ∈ V, v≠z,s: $f^{\Delta}(v) > 0$
    - Při všech změnách toku dokážeme v O(1) aktualizovat seznam.
    - Díky seznamu dokážeme najít vrchol s kladným přebytkem v O(1)
  - ∀v L(v) := seznam všech vu ∈ E: r(vu) > 0 & h(u) < h(v)
    - Převedení v O(1).
    - Zvednutí v O(N)
- Vylepšení algoritmu
  - Místo tohohle: dokud ∃u ≠z,s; $f^{\Delta}(u) > 0$:
  - Tohle: dokud ∃u ≠z,s; $f^{\Delta}(u) > 0$, h(u) max:
  - Pak se to změní takhle:
    - Lemma N': #NP = $O(N^2\sqrt{M})$
      - Poznámka: K tomu bude potřeba ještě šílený důkaz. Bude potřeba jinou implementaci (jiné datové struktury?).
    - Pak bude časová složitost alogoritmu' $O(N^3)$.
    - 

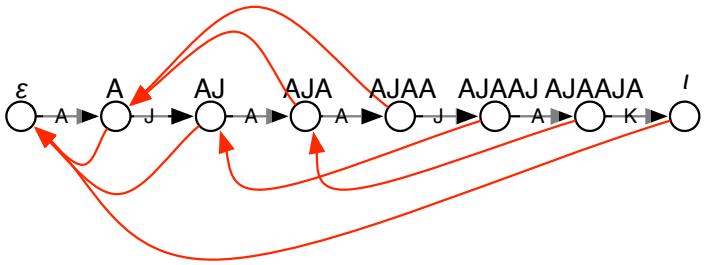
- 07/10/30
  - 
  - Lemma N': V algoritmu G' je #NP = $O(N^3)$
    - Dk: H := max { h(v) | v ≠ z,s & $f^{\Delta}(v) > 0$}
    - Běh G' rozdělíme na fáze, fáze končí změnou H
    - #NP v 1 fázi ≤ velikost nejvyšší hladiny na začátku fáze (pže nenasycené převedení vynuluje přebytek, čímž vrchol „vypadne“ a už se jím ve fázi nezabýváme)

- Úvaha: H jako potenciál, na zač. 0 a nikdy neklesne
- # fází … $\leq 4N^2$:
  - fáze končí zvýšením H (H++): $\leq 2N^2$
  - fáze končí snížením H (H+=n): $\leq 2N^2$
- Lemma N'': #NP = $O(N^2\sqrt{M})$
  - Dk:
    - Zvolíme $K \in \mathbf{N}$
    - fáze:
      - laciné: #NP ≤ K, ve všech laciných #NP $\leq 4N^2K$
      - drahé: #NP > K
    - Potenciál $\Psi = \sum_{v \neq z, s\ f\Delta(v) > 0} p(v)$, p(v) = #u, h(u) ≤ h(v)
    - Na začátku $\Psi \leq N^2/K$, stále $\Psi \geq 0$
    - *..... (nezapisuju, viz zápisky v TeXu) .....*
- Zpátky k sítím
  - Komparátorová síť je kombinační obvod, jehož hradla jsou komarátory:
  - in: **a,b** –> **[ ]** –> out: **min, max**
  - A výstupy komparátoru se nevětví
  - Bubble Sort (a BS paralelně)
  - 

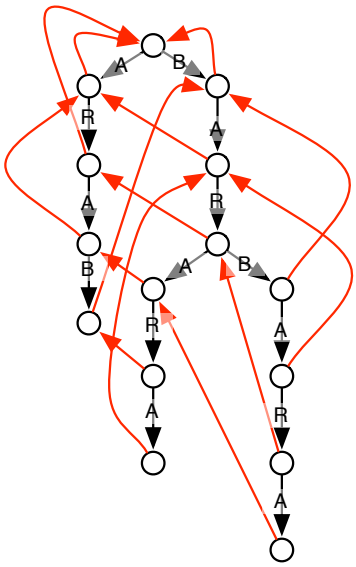
    - Definice: Posloupnost $x_0 \ldots x_{n-1}$ je
      - čistě bitonická: x0 ≤ … ≤ xj ≥ … ≥ xn-1
      - je bitonická: je čistě bitonická, když ji zrotuju o nějakých k pozic (posloupnost „v kruhu“)
    - Def: Separátor S_n
    - 
      - 1 hladina: O(n) hradel
      - $(y_i, y_{i+n/2}) = cmp(x_i, x_{i+n/2})$
    - Lemma: Pokud vstup SN je bit. x0 … xn-1, pak výstup y0 … yn-1 splňuje
      - $y_0 \ldots y_{n/2-1}$ a $y_{n/2} \ldots y_{n-1}$ jsou bitonické
      - $\forall i, j \leq n/2\ y_i \leq y_{j+n/2}$
    - Def. Bitonická třídička Bn

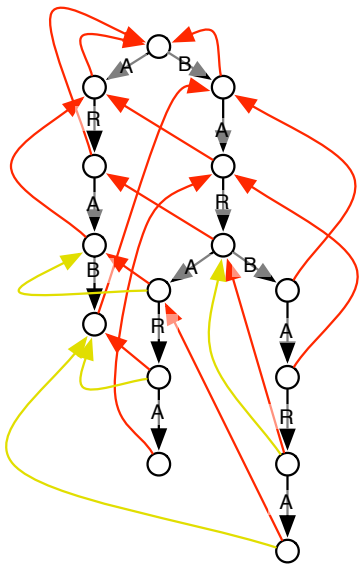


- O(log n)) hladin
  O(n log n) hradel
  - Dk:
    - $x_0 \dots x_{n-1}$ čistě bitonická
      - j = pozice maxima, búno j < n/2 (jinak zrcadlově)
      - $k := \min \{ i : x_i \geq x_{i+n/2} \}$
    - *ATD. – nepsal (a nekreslil) jsem vše, zkusit nakreslit doma*
    - mergesort
- 07/11/06
  - Hledání
    - Příklady
      - JEHLA v DRAKJEHLADOVÝ, VKUPCEJEJEHLA
      - KOKOS v ČLÁNEKOKOKOSU
    - Chci
      - Nezačínat při neúspěchu od začátku. / Porovnávat řetězce pro všechny pozice.
      - Nepřijít o skutečný výskyt.
    - Problém
      - Jak moc se vracet, co zahazovat?
    - Časová složitost pro jehlu s neopakujícími se znaky
      - O(S)
    - Časová složitost při triviálním obecném vyhledávání
      - (nejhůře) O(JS), J = délka jehly, S = délka sena
    - Příklady
      - AJAAJAK (Ája a jak) v AJAAJA (Ája a já)
        - AJA(AJA + (přijde) A
        - AJA(AJA + (přijde) J
  - Řetězce obecně
    - **Σ** *abeceda* (binární, latinská abeceda pár ASCII kódů, v extrémním případě slova jazyka, ale to neuvažujeme)
    - **Σ*** množina *slov* (*řetězců*) nad abecedou Σ
    - **ε** prázdné slovo
    - **|α|** délka slova α
    - **αβ** zřetězení
      - αε = εα = α
      - |αβ| = |α| + |β|
    - **α[i]** i-té písmeno (programátorské číslování od 0)
    - **α[i:j]** podslovo z písmen *i* až *j-1*
      - **α[:j]** = α[0:j] prefix
      - **α[i:]** = α[i:|α|] sufix
      - Každé slovo je svým prefixem/sufixem. *Vlastní* *fix, znamená, že nejde o totéž slovo.
  - Problém
    - in
      - *ι* jehla, J = |ι|
      - *σ* seno, S = |σ|
    - out

    - {i | σ[i:i+J] = ι }
- Vyhledávací automat (Knuth, Morris, Pratt) – KMP
    - orientovaný graf: stavy = vrcholy = prefixy slova *ι*
    - dopředné hrany: α → εx d(α,x)
    - zpětné hrany: α → nejdelší vlastní sufix α, který je stavem (tedy prefixem *ι*)
    - 

    - Algoritmus vyhledávání
        - Snažím se prodlužovat slovo podle stavových přechodů dokud buď nenajdu slovo nebo nemám kam dál – v tom případě se pustím po zpětné hraně.
        - α := ε
        - pro c ∈ *σ* postupně:
            - dokud ∃ d(α,c) & α ≠ ε:
                - α := z(α)
            - pokud ∃ d(α,c) ⇒
                - α := d(α,c)
            - pokud α = *ι* => hotovo
    - Praktická modifikace
        - Stavy se označí délkami prefixů.
    - Algoritmus vyhledávání 2:
        - k := 0
        - Atd.
    - Korektnost
        - Invariant: stav po přečtení vstupu β: α(β) = nejdelší sufix β, který je sufixem *ι* ⇒ korektnost KMP
    - Lemma: Vyhledávání doběhne v čase O(S).
        - Kroků dopředu je max. S. Kroků zpět je max. tolik, kolik dopředu, tedy S, tedy max 2S.
    - Konstrukce vyhledávacího automatu, resp. zpětných hran
        - Lemma: z(β) má být α(β[1:]) (β[1:], aby to nevedlo zpět do β)
        - Pustíme hotový automat na *ι*[1:] ⇒ budeme se dozvídat postupně cíle zpětných hran.
        - Fígl: Začneme s automatem bez zpětných hran, a jak je postupně „nacházíme“, tak je přidáváme do automatu.
    - Konstrukce (pořádně)
        - α : = krok(a,c) : **{**
            - dokud ∃ d(α,c) & α ≠ ε:
                - α := z(α)
            - pokud ∃ d(α,c) ⇒
                - α := d(α,c)
        - **}**
        - Algoritmus
            - sestrojíme d(opředné hrany)
            - z(ε) := ∅, z(*ι*[0] := ε
            - α := ε
            - pro i = 1 do J:
                - a := krok(α,*ι*[i])
                - z(*ι*[0:i+1]) := α
        - **Věta.** KPM hledá z v čase O(|J| + |S|)
        - Úloha k zamyšlení

        - 1 jehla : lineárně výskytů, více jehel: více výskytů (musí se vypsat, bez výpisů lze rychleji) :(– McCorasick (?)
    - Alternativní algoritmus (narychlo) **Rabin a Karp**:
        - Naivní algoritmus (samé porovnávání) zbytečně ztrácel čas – jak ho vylepšit?
        - Řešení:
            - Mějme hashovací fci: h: $\sum^J \rightarrow Z$
            - H = h(J)
            - Kontrolujeme hash aktuálního vyhledávacího okénka.
            - Je třeba zvolit hash, který dokážeme v konstantím čase změnit z h(x[i:j]) na h(x[i+1:j+1])
            - Stačí běžná hashovací fce:
                - $h(x_{0\_}x_{j-1}) = (\sum x_i p^{j-i-1}) \bmod N$
                - TODO: Proč číslování v tomto směru?
                - TODO (od Mareše): Kde je zádrhel? Fakt jsme si pomohli? (Asi ne.)
- 07/11/13
    - Zase hledání výskytů v řetězci, teď ale více výskytů
        - Slova
            - ara, bar, arab, baraba, barbara
            - 
        - krok(s, c)
            - dokud ∄f(s,c) & s ≠ kořen
                - s := z(s)
            - pokud ∃f(s,c)
                - s := f(s,c)
            - vrátíme s
        - návrh algoritmu
            - s := kořen
            - ∀c písmena σ
                - s := krok(s, c)
        - Chceme vypsat slovo končící na dané pozice.
        - Problém: třeba BARBAR – nenajde se BAR, BARAB – nenajde se ARAB
            - Řešení 2: vracení se přes zpětné hrany do kořenu? … zabere dost času, ale jde to.
        - Pozorování: Stav automatu jednoznačně určuje množinu v něm končících slov.
            - Řešení 3: předpočítání množin končících slov pro každý stav.
                - (prý) problém: Taky může zabrat až nelineárně času. TODO promyslet.
        - Řešení 4:
            - slovo(s) slovo končící v *s*, resp. index ι_i nebo 0
            - out(s) nejbližší vrchol, do nějž se z *s* můžeme dostat po zpětných hranách, a končí tam slovo (slovo(v) ≠ 0)

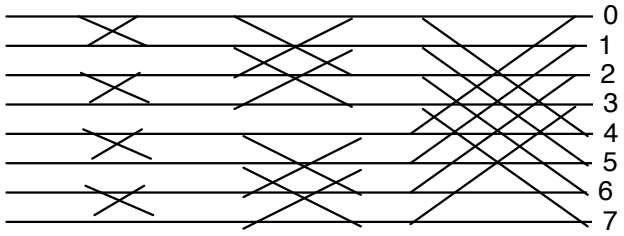
- 

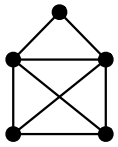
- návrh algoritmu
  - s := kořen 1
  - ∀c písmena σ 2
    - s := krok(s, c) 3
    - je-li slov(s)≠0 4
      - vypiš(slovo(s)
    - v := out(s) 5
    - dokud v ≠ 0 6
      - vypiš(slovo(v)) 7
      - v := out(v) 8
- Poznámky ke složitosti hledání a vypisování:
  - zpětná hrana aspoň o hladinu výš
  - hladina nezáporná
  - dopředná o hladinu níže (vlastně výše a naopak;)…)
  - potenciálový argument → lineární složitost O(s) kroků 2-5
  - kroky 6-8, složitost O(#výskytů)
- Konstrukce automatu (Aho, Corasicková)
  - 1. postavíme strom dopředných hran, r := kořen
  - 2. spočteme slovo(*)
    1. + 2. → O(J), J = $\sum|\iota_i|$
  - 3. spočteme z(*):
    Nemůžeme po slovech (budeme potřebovat alespoň začátky jiných slov). –> PO HLADINÁCH
    3. část trvá také O(J) viz níže
    - z(r) :=0, Q := { synové r}, ∀ v ∈ Q, z(v) := r
    - dokud Q ≠ ∅
      - u := vyber z Q
      - pro syny *v* vrcholu *u*
        - z : = krok (z(u), znak na hraně uv))
        - z(v) := z
        - Pokud slovo(z) ≠ 0
          - out(v) := z
        - Jinak
          - out(v) := out(z)
            A máme i žluté hrany.
  - Správnost
    - strom správně
    - žluté hrany jsou správně
    - funguje trik z(β) = α(β[1:])

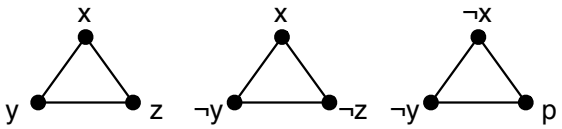
        - lze omezit hledáním všech jehel –> 3. část trvá také O(J)
      - Věta. Algoritmus Aho-Corasicková najde všechny výskyty slov $\iota_i \dots \iota_k$ ve slově σ v čase $O(\sum|\iota_i|+|\sigma| + \#\text{výskytů})$
      - Paměťová reprezentace
        - Pole ukazatelů ve vrcholech, pro velké abecedy hashovací tabulka v každém vrcholu, případně jedna tabulka s klíčem (stav, znak). Všechny možnosti zaručují konstantní čas.
- Změna tématu: Polynomy
  - Konečné polynomy nad reálnými (resp. komplexními) čísly
  - Mějme dva polynomy (předpokládejme stejný stupeň, vždycky jdou doplnit)
    - $P(x) = \sum_{j=0}^{n-1} p_j x^j$
    - $Q(x) = \sum_{j=0}^{n-1} q_j x^j$
    - Vynásobme je: $R = P \cdot Q$, $R = \sum_{j,k} p_j q_k x^{j+k}$
    - $r_l = \sum_{j=0}^{l} p_j q_{l-j}$ – skoro jako skalární součin, říká se tomu *konvoluce* (jeden vektor se otočí, částečně se překryjí/posunou a skalárně vynásobí), podobá se to taky i písemnému násobení čísel (bez přenosů)
    - $\theta(n^2)$
  - Jak násobit polynomy rychleji?
    - $\forall x\ R(x) = P(x) \cdot Q(x)$
    - Věta. Jsou-li $x_0 \dots x_k$ navzájem různá čísla a $y_0 \dots y_k$ navzájem různá čísla (reálná, komplexní, dokonce jakékoli těleso), pak $\exists!$ polynom P stupně ≤ k t.ž. $\forall j\ P(x_j) = y_j$
    - Plán:
      - k = 2n-1 (stačilo by i 2n-2, ale my chceme (proč?) liché)
      - $x_0 \dots x_k$ libovolně, ale různá
      - Spočteme
        - $P(x_0) \dots P(x_k)$
        - $Q(x_0) \dots Q(x_k)$
        - $\forall j\ y_j = P(x_j) \cdot Q(x_j)$
          - najít R stupně ≤ k: $\forall j\ R(x_j) = y_j$
      - Chceme polynom P s n koeficienty vyhodnotit v $x_0 \dots x_{n-1}$:
        - Búno $n = 2^m$
        - Rozděl a panuj:
        - $P(x_j) = p_0x^0 + p_1x^1 + p_2x^2 + \dots + p_{n-1}x^{n-1} =$
        - $= (p_0x^0 + p_2x^2 + \dots + p_{n-2}x^{n-2}) + (p_1x^1 + p_3x^3 + \dots + p_{n-1}x^{n-1}) =$
        - $= (p_0x^0 + p_2x^2 + \dots + p_{n-2}x^{n-2}) + x(p_1x^0 + p_3x^2 + \dots + p_{n-1}x^{n-2})$
        - $= S(x^2) + x \cdot L(x^2)$
        - $P(-x) = S(x^2) - x \cdot L(x^2)$
        - Polynom n koeficienty v n bodech → 2 polynomy s n/2 koeficienty s n/2 bodech.
        - Takže
          - $T(n) = 2T(n/2) + O(n)$
          - $T(n) = O(n \log n)$
          - **Ale** jak tohle udělat, abychom měli takhle spárovaná +-čísla v dalších krocích?
            - Řešení: komplexní čísla
              - n=2 1 -1
              - n=4 1 -1 i -i

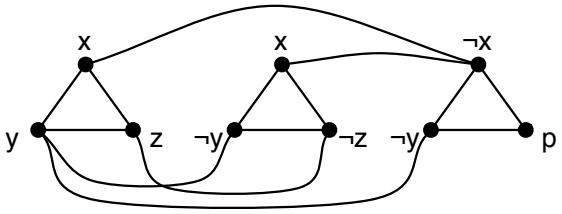
- 07/11/20
  - Jak násobit polynomy rychleji? – pokračování
    - Základní věci o komplexních číslech
      - (Opakování z mateřské školky, třeba x/y = (x*sdruž(y))/(y*sdruž(y), nebo abs(x) = sqrt(x*sdruž(x)).)
      - Goniometrický tvar:
        - $x = \cos\varphi + i \sin\varphi$ pro $|x| = 1$

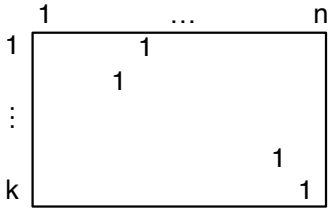
    - $x = |x|(\cos\varphi + i \sin\varphi)$, pro $\forall x$, ϕ říkáme argument
  - Exponenciální tvar
    - Eulerova formule: $e^{i\varphi} = \cos\varphi + i \sin\varphi$
    - $x = |x| e^{i \varphi(x)}$
    - Násobení
      - $xy = |x||y| e^{i(\varphi(x)+\varphi(y))}$
    - Umocňování (odmocňování)
      - $x^{\alpha} = \ldots$
    - Nejednoznačnost n-té odmocniny.
      - *n*-tých odmocnin z jedničky je *n*, platí i pro jiná čísla mimo 0.
- Segregace odmocnin z jedničky
  - Některé odmocniny jsou lepší (bohatší) než jiné.
    - např. 4. odmocniny z 1: -1 a i
      - jejich mocniny (0., 1., 2., …) dávají různý počet navzájem různých hodnot
- Definice: $x \in \mathbf{C}$ je primitivní n-tou odmocninou z 1 $\Leftrightarrow x^n = 1$, ale $x^1, x^1, \ldots, x^{n-1} \neq 1$
- Pozorování: Pokud n je sudé, pak $x^{n/2} = -1$ (…? dál už se tímhle prý zabývat nebudeme)
- Pozorování: $\omega_n = e^{2\pi i/n}$, overline{ω_n} = $e^{-2\pi i/n} = 1/\omega_n$ jsou primitivní n-té odmocniny z 1.
- Pozorování: pro $0 \leq j \leq k \leq n-1$ $x^j \neq x^k$: jinak $x^{k-j} = 1$
- Algoritmus FFT
  - Vstup: $n = 2^k$, $(p_0 \ldots p_{n-1})$ koeficienty polynomu P, ω = primitivní n-tá odmocnina 1
  - Výstup: $(p_0' \ldots p_{n-1}')$, $p_k' = P(\omega^k)$
  - 0. Pokud n = 1, vrátíme (p_0)
  - 1. $s := (p_0, p_2, \ldots p_{n-2})$, $l := (p_1, p_3, \ldots, p_{n-1})$
  - 2. $s' := FFT(s,\omega^2)$, $l' = FFT(l, \omega^2)$
  - 3. pro $0 \leq j \leq n/2$ $T(n) = 2T(n/2) + O(n) = O(n \log n)$
    - $p_j' = s_j' + \omega^j * l_j$
    - $p_{j+n/2}' = s_j' - \omega^j * l_j$
- Definice: Diskrétní Fourierova transformace (DFT)
  - je funkce $f: C^n \rightarrow C^n$, $y = f(x)$, $\forall j$ $y_j = \sum_{k=0}^{n-1} x_k \omega^{jk}$
  - $\Rightarrow$ f je lineární
  - $\Rightarrow f(x) = \Omega x$, $\Omega_{jk} = \omega^{jk}$
  - matice: $\Omega^T = \Omega$
- Lemma: součin řádků $\Omega_j \cdot \Omega_k = 0$ pro $j \neq k$, jinak = n
- Dk:
  - součin řádků $\Omega_j \cdot \Omega_k = \sum_{l=0}^{n-1} \Omega_{jl} \cdot$overline{$\Omega_{kl}$} = $\sum_{l=0}^{n-1} \omega^{jl} \cdot$overline{$\omega^{kl}$} = $\sum_{l=0}^{n-1} \omega^{(j-k)l}$ =
    - = (pro $j \neq k$) = $(\omega^{(j-k)n}-1) / (\omega^{j-k}-1) = (1-1) / (\neq 0) = 0$
    - = (pro j=k) = $\sum_{l=0}^{n-1} 1 = n$
- Důsledek:
  - Ωoverline{Ω } = $n \cdot E$ (n na diagonále, jinak nuly)
- ZBYTEK JSEM SI NEPSAL, BLÁ BLÁ BLÁ
- Aplikace:
  - Spektrální rozklad signálu (zvuku) / rozklad na sinusovky. (transformace a inverzní)
  - Ozvěna (konvoluce)
  - Komprese dat JPEG (DCT je modifikovaná DFT – jen na reálná čísla)
  - Různé problémy s polynomy
  - Nečekaně i násobení dlouhých čísel v O(n log n) nebo skoro tak (?).
- Kombinační obvod s logaritmickým počtem hladin:

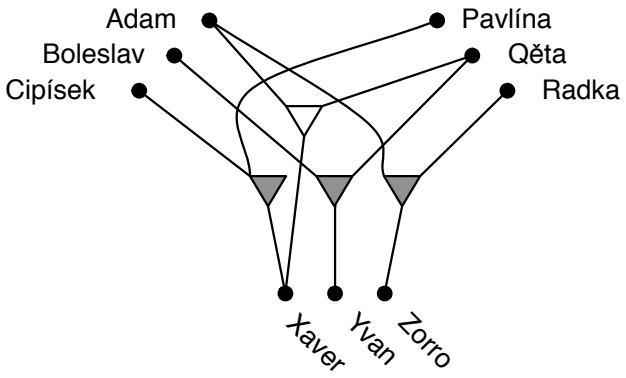
- 

    - To je ale ošklivý obrázek
- 07/11/27 (na papíře)
- 07/12/04
    - Aparát na porovnávání složitosti/obtížnosti problémů (jejich řešení), převádění problémů jeden na druhý.
    - Pro začátek: Jen rozhodovací problémy (Ano/Ne).
        - Například:
            - Existuje perfektní párování pro daný graf? (Ne: Jaké párování?)
                - To lze ale použít k nalezení toho párování:
                    - Existuje-li párování, odebereme nějakou hranu a ptáme se znova. Takhle otestujeme všechny hrany. Z nejvýše tolik dotazů, jako je hran, zjistíme párování, tzn. také v nějakém polynomiálním čase.
                    - Takové triky jdou často.
            - Existuje v síti tok velikosti ≥ k?
    - Definice: *Rozhodovací problém* je takový, jehož výstup je vždy buď *ano* nebo *ne*.
    - Definice: Jsou-li *A* a *B* rozhodovací problémy, pak říkáme, že *A* lze *redukovat* na *B* ($A \to B$) $\equiv \exists$ funkce *f* spočitatelná v polynomiálním čase taková, že $\forall x\, A(x) = B(f(x))$.
    - Pozorování: Lze-li problém redukovat na polynomiální, je také polynomiální.
    - Pozorování: Redukce je reflexivní a transitivní (jako uspořádání)
    - Problém SAT
        - „satisfiability“ – splnitelnost logických formulí
        - Např.: $(x \vee \neg y) \,\&\, (y \vee \neg x)$ je splnitelné
        - Def.: CNF: Konjunktivní normální forma: Konjunkce disjunkcí literálů, tzn. proměnných nebo jejich negací.
        - Vstup: Formule v CNF.
        - Výstup: $\exists$ dosazení 0/1 za proměnné takové, že $\phi(\,) = 1$
        - 3-SAT: SAT, ale všechny klauzule (tedy jednotlivé disjunkce obsahují ≤ 3 literály)
        - 3-SAT → SAT
            - (jasné, speciální případ)
        - SAT → 3-SAT
            - Pro příliš velkou klauzuli $(\alpha \vee \beta)$ $(|\alpha| + |\beta| \geq 4)$ přepíšeme $(a \vee x) \,\&\, (\beta \vee \neg x)$
            - Zřejmé, že to neovlivní splnitelnost (na přednášce se vysvětlovalo).
    - Nezávislá množina
        - Vstup: Neorientovaný graf G, $k \in \mathbb{N}$
        - Výstup: $\exists\, A \subseteq V(G)$: $|A| \leq k$ & $u,v \in A$, $u \neq v$, $uv \notin E(G)$
        - Např.
        - 
            - Jeden bod ze základů a vrcholek. (Nebo třeba prázdná množina.)
    - Redukce 3-SAT a nezávislé množina (oba směry)
        - 3-SAT → Nez. množina
            - U 3-SATU hledáme vlastně nekonfliktní literál a jeho hodnotu pro splnění každé klauzule.
            - Příklad
                - (Vždy můžu klauzule beze změny výsledku nafouknout na 3 literály.)
                - $(x \vee y \vee z) \,\&\, (x \vee \neg y \vee \neg z) \,\&\, (\neg x \vee \neg y \vee p)$

- 

  - Chci nezávislou množinu, jež mi vybere z každého trojúhelníku právě jeden vrchol.
  - A co konfliktní volby:
    - Nerad bych jednou vybral *x* a jednou ¬*x*.
    - Řešení: Přidám hranu mezi proměnnými a jejich negacemi.
- 

    - ∃ nezávislá množina velikosti #klauzulí?
  - Vsuvka: Problém kliky v grafu (úplný podgraf o *k* vrcholech)?
    - To je ale nezávislá množina doplňku grafu! Hotovo.
- Nezávislá množina → 3-SAT
  - Proměnné v_1 … v_n pro vrcholy
  - Pro ∀ hrany ij∈E přidáme klauzuli $\neg v_i \vee \neg v_j$
  - Jak do toho ale zakódovat počet?
- 

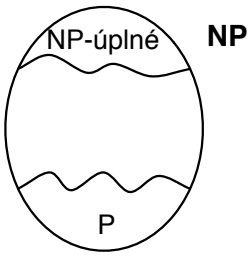
  - $X_{ij} = 1 \equiv$ i-tý prvek množiny je vrchol j
  - $\forall j, i, i', i \neq i' \quad x_{ij} \Rightarrow \neg x_{i'j}$
  - $\forall i, j, j', j \neq j' \quad x_{ij} \Rightarrow \neg x_{ij'}$
  - $\forall i \quad x_{i1} \vee x_{i2} \vee \ldots \vee x_{in}$
  - Pak dostaneme nezávislou množinu *k* nebo větší (jen implikace, viz výše)
- „3-D párování“ (matching)
- 

  - Vstup: Množina K, H, Z a množina kompatibilních trojic
  - Výstup: Perfektní podmnožina trojic
- 3-SAT → 3,3-SAT (každá proměnná v max. 3 literálech)
- Pokud se vyskytuje v k≥ 3 lit.: Nahradíme výskyty novými proměnnými x1 … xk a přidáme klauzule:
  - $x_1 \Rightarrow x_2, x_2 \Rightarrow x_2, \ldots, x_{k-1} \Rightarrow x_k, x_k \Rightarrow x_1$
- 3,3*-SAT, ze tří literálů, kde se vyskytuje proměnná aspoň jeden negativní a aspoň jeden pozitivní
- TODO: Zkusit převést na SAT 3D-párování.

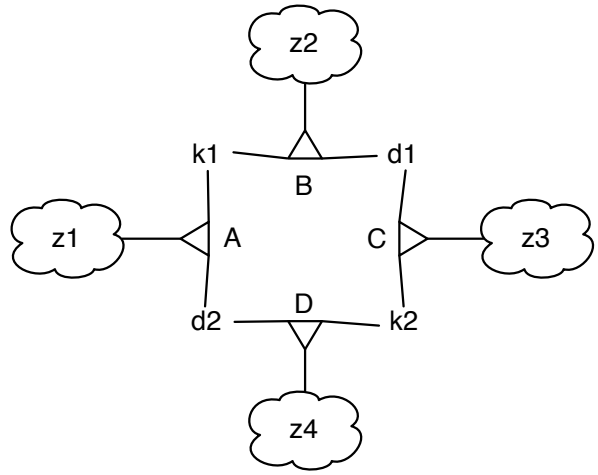
- 07/1211
  - Ke zkouškám

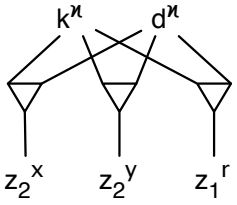
- Předtermín
  - Poslední pátek semestru 10:00 před S322
  - Nebude se zkoušet leden
  - Nebude třeba zápočet
- Zkouška (normální)
  - písemka
    - teorie
    - pár algoritmických problémů
  - následně možná ústní zkouška

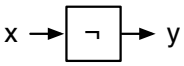
- Převádění problémů
  - Definice: P je třída (rozhodovacích) problémů, které jsou řešitelné v polynomiálním čase. ($L \in P \Leftrightarrow \exists$ polynom f $\exists$ algoritmus A tž. $\forall x\ L(x) = A(x)$ a A(x) doběhne v čase $\leq f(|x|)$
  - Definice: NP je třída (rozhodovacích) problémů, takových, že $L \in NP \Leftrightarrow \exists$ problém $K \in P$ $\exists$ polynom g: $(\forall x\ L(x) = 1 \Leftrightarrow \exists y\ |y| \leq g(|x|)$ & K(x,y)

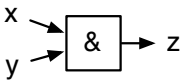
    jakoby polynomiální „s nápovědou“
  - Pozorování:
    - $SAT \in NP$
    - $P \subseteq NP$
  - Definice: Problém L je NP-těžký $\equiv \forall M \in NP\ M \rightarrow L$
    - Pokud nějaký takový je $\in P \Rightarrow P = NP$.
  - Definice: L je NP-úplný $\equiv$ L je NP-těžký & $L \in NP$
    - To jsou úplně nejtěžší problémy v NP. U těch (skoro) nemá smysl hledat polynomiální řešení.
  - Věta: (Cook) SAT je NP-úplný.
    - Převody z minulé přednášky tedy mj. plynou z této věty.
  - Větička: L je NP-úplný a $L \rightarrow M \in NP \Rightarrow$ M je také NP-úplný
    - Stačí dk. pro NP-těžkost (pak už to z toho plyne) $\forall Q \in NP: Q \rightarrow L \rightarrow M \Rightarrow Q \rightarrow M$
  - Důsledky: Nezávislá množina, klika, 3-SAT, 3,3-SAT jsou NP-úplné
  - 

    Možná, že jsou všechny množiny totožné, ale dost možná taky ne:).
  - Zoo NP-úplných problémů
    - logické: SAT, 3-SAT, 3,3-SAT
    - grafové: Nezávislá množina, kika, 3D párování, 3-barvení, hamiltonovské cesta/kružince
      - Pozn: h. cesta obsahuje všechny vrcholy (bez opakování), obdobně kružnice
    - číselné: batoh, loupežníci, Ax=b (A jsou 0/1(?), speciální případ celočíselného lineárního programování)
  - 3D-párování
    - z 3,3-SATu

- 

  - A + C x =1 zvířátka 2, 4 volná
  - B + D x = 0 zvířátka 1, 3 volná
  - (k, d – kluci, dívky)
  - Klauzule: $\varkappa$: (x ∨ y ∨ ¬ r)
  - 

  - max. 2 výskyty každého literálu ⇒ zvířátek je dost pro všechny klauzule
  - 2p-k volných zvířátek (přidáme 2p-k párů univerzálních milovníků zvířat)
  - **TODO rozmyslet doma**
  -
- Věta': Obvodový SAT je NP-úplný. (Tzn. s logickými obvody, což je zobecnění SATu.)
- Lemma: O-SAT (obvodový SAT) → 3-SAT
  - Pro ∀ hradlo zavedeme proměnnou, popisující jeho výstup
  - 

  - (x ∨ y), (¬x ∨ ¬y)
  - 

  - x & y ⇒ z (z ∨ ¬x∨¬y)
  - ¬x ⇒ ¬z (…)
  - ¬y ⇒ ¬z (…)
- L ∈ P ⇒ ∃ polynomiálně velký boolovský obvod, který počítá L
  - Odstraníme klopné obvody (paměť, tedy časový vývoj) konstrukcí kopií obvodu pro čas T_0, T_1, …
- BÚNO modifikace definice:
  - Definice: NP je třída (rozhodovacích) problémů, takových, že L ∈ NP⇔ ∃ problém K ∈P ∃ polynom g: (∀x L(x) = 1 ⇔ ∃y |y| **=** ***(místo ≤)*** g(|x|) & K(x,y)
- L ∈ NP
  - L→ O-SAT
  - vstup x
  - spočteme g(|x|) (velikost nápovědy)
  - vezmeme obvod pro K se vstupem velikosti |x| + g(|x|)
  - do obvodu dosadíme za x → splnitelnost obvodu (→ splnitelnost)
- Důležité
  - redukce

- NP-úplnost
- umět ze základního NP-ú problému (splnitelnosti) dokázat NP-úplnost jiných