Konstrukce LCP pole

- $s(i)$: i-tý nejmenší suffix
  - tj. $s(i) = T[S[i]:]$, T je text, S je suffixové pole

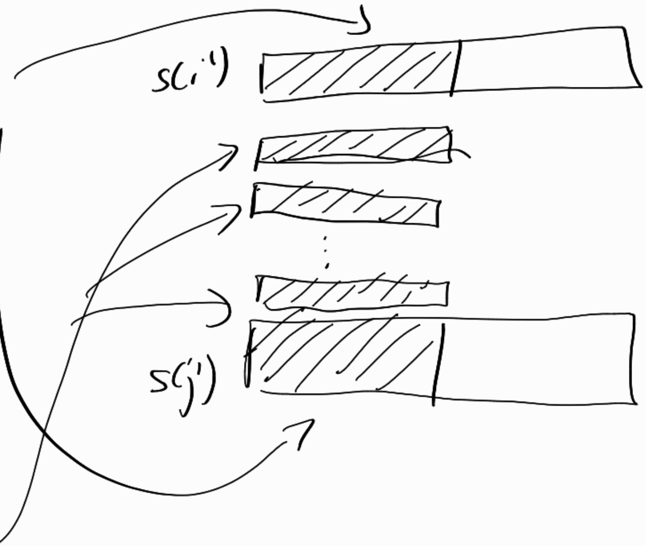
- $L[i]$: délka nejdelšího společného prefixu řetězců $s(i)$ a $s(i+1)$

- vezměme si $s(i)$ a $s(i+1)$ a předpokládejme, že $lcp(s(i), s(i+1)) = k > 0$
- pak $s(i)[1:]$ a $s(i+1)[1:]$ jsou také suffixy T !
- jelikož jsou to suffixy, pak $s(i)[1:] = s(i')$ pro nějaké i'
  $s(j)[1:] = s(j')$ — " — j'

- pokud $lcp(s(i), s(i+1)) > 0$, pak se shodují alespoň na prvním znaku

$\rightarrow$ $lcp(s(i'), s(j')) = lcp(s(i), s(i+1)) - 1 = k-1$

- všimněme si, že $lcp(s(i'), s(j')) = k-1$
  - tyhle kusy jsou stejné
  - pak jelikož máme lex. seřazené řetězce, pak i všechny řetězce mezi začínají na stejný prefix



$\rightarrow$ $lcp(s(i'), s(i'+1)) \geq k-1 \Rightarrow L[i] \geq k-1$

- budeme zpracovávat prefixy od nejdelšího po nejkratší

→ tj. začneme $T[0:]$, $T[1:]$, ...

- další suffix tedy najdeme utržením prvního znaku současného suffixu

- $R[i]$: kolikátý je v lex. pořadí suffix $T[i:]$
  - je to tedy inverze S

Alg:

1. $k \leftarrow 0$ // lcp spočítaný v předchozí iteraci
2. for $p \leftarrow 0, \dots, m-1$ // zpracováváme postupně $T[0:], T[1:], \dots$
   3. $k \leftarrow \max(k-1, 0)$ // lcp se vždy musí zkrátit alespoň o 1, neboť se zkracuje celý suffix
   4. $i \leftarrow R[p]$ // $T[p:]$ je i-tý v lex pořadí.
   5. $q \leftarrow S[i+1]$ // koukneme na další suffix v pořadí,
   6. while $(p+k < m) \wedge (q+k < m) \wedge (T[p+k] = T[q+k])$
      7. $k \leftarrow k+1$ // postupně počítáme, na kolika znacích se shodují $S[i]$ a $S[i+1]$
   8. $L[i] \leftarrow k$

Lem: Alg pracuje v čase $O(m)$.

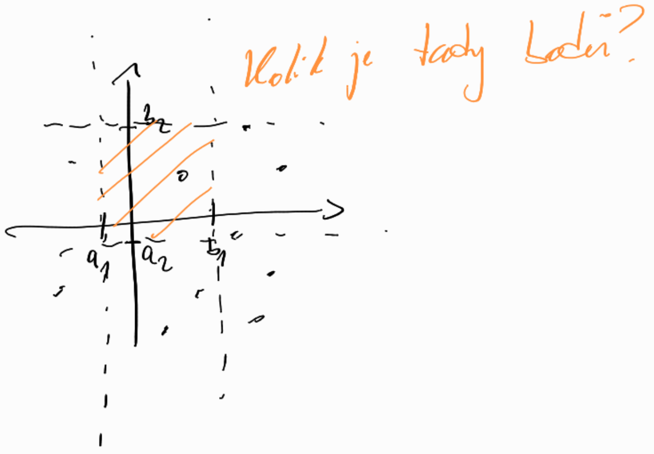
Dk:

- kromě while cyklu všechno v těle for cyklu běží konstantní
- čas ve while cyklu budeme amortizovat :)
- $k \in \{0, 1, \dots, m\}$
- k začíná na 0
- k se vždy změní aspoň jednou v každé iteraci
  - v kroku 3 se sníží o 1
  - v kroku 7 roste o 1
- k klesne nejvýše n krát, jednou za každou iteraci for cyklu
- kolikrát může k vyrůst?
- jelikož $k \leq m$ vždy!, pak může k vyrůst nejvýše $2m$ krát,

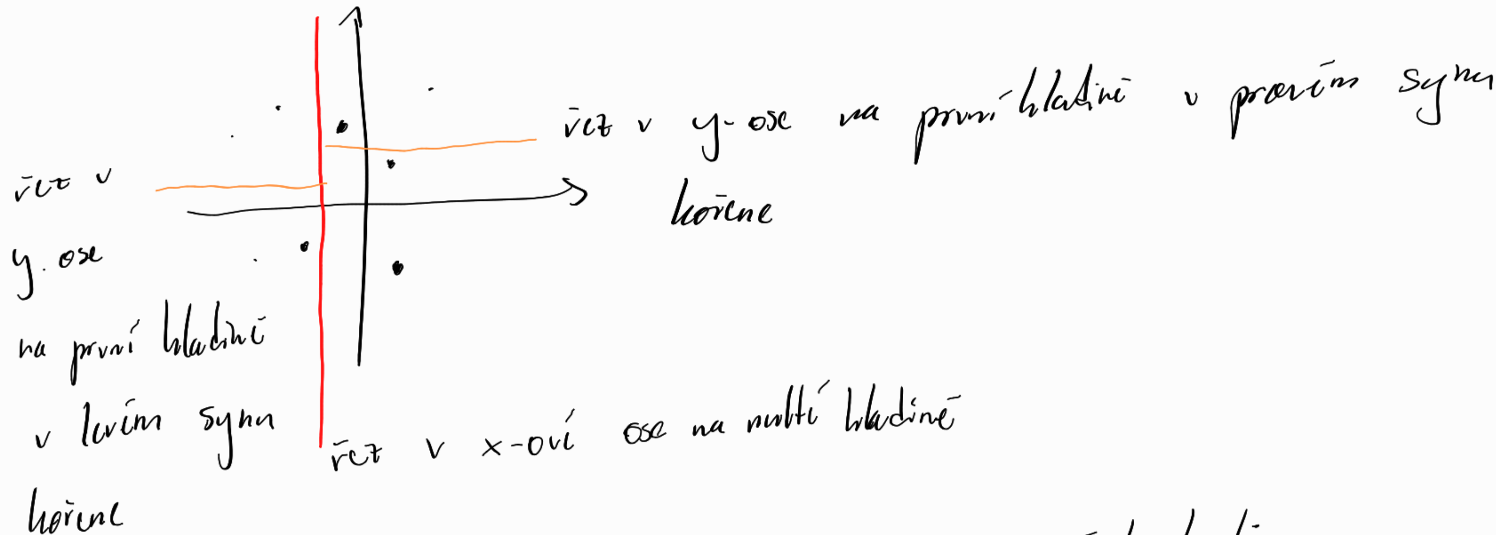
jinak $+(2m + c) - m > m$, ale $k \leq m$ vždy! ∎

K-d stromy

- jsme v prostoru $\mathbb{R}^d$ a chceme se ptát na počet/vyjmenovat/... prvky, které leží v $[a_1,b_1] \times [a_2,b_2] \times \dots \times [a_d,b_d]$



- máme binární strom, kde na $(0+i)$, $(d+i)$, $(2d+i)$, ... hladině dělíme body na „poloviny" podle i-té souřadnice



- na každé hladině se půlí počet uvažovaných bodů

$\rightarrow \log n$ výška

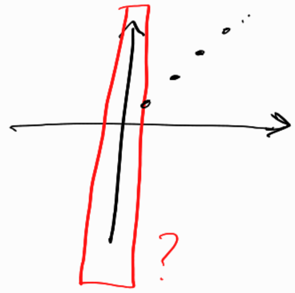
- nicméně mohou být dost pomalé

Lem: Složitost range query v k-d stromě je $\Omega(\sqrt{n})$

teď jsme v $\mathbb{R}^2$

Dk:

- uvažme kd-strom pro body $\{(i,i) : 1 \le i \le n\}$, kde $n = 2^t - 1$
- dostaneme úplný binární strom na t hladinách
- co se stane, když uděláme dotaz na interval $\{0\} \times \mathbb{R}$?



- když porovnáváme x. souřadnici, jdeme vždy doleva
- když porovnáváme y souřadnici, oba podstromy leží v R

→ počet navštívených vrcholů se zdvojnásobuje v každé druhé hladině

→ složitost $\Omega(2^{t/2}) = \Omega(\sqrt{n})$ □

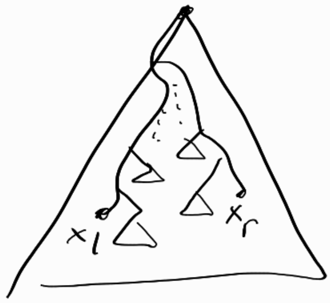
- mimochodem, tohle je worst-case
- obecně je složitost dotazu $O(n^{1-\frac{1}{d}})$
  - mimochodem je tohle optimum, pokud paměť DS je lin. vůči počtu prvků

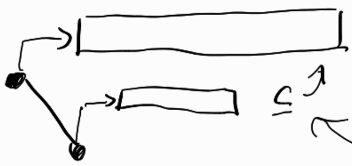
Range tree

- opět počítáme body v $\mathbb{R}^d$
- ukážeme si pro $\mathbb{R}^2$, zobecnění je ± jasné

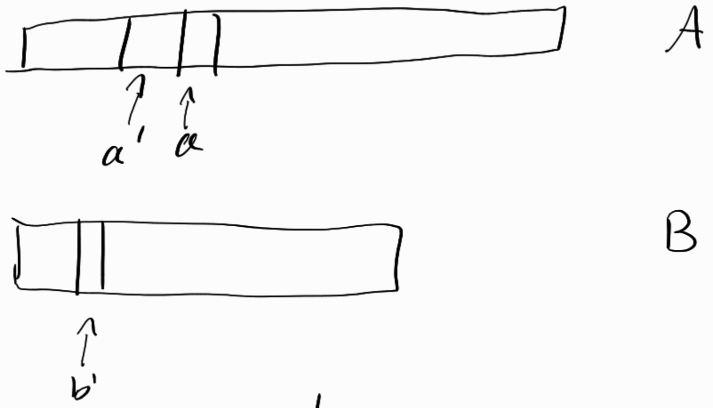
- postavíme napřed intervalový strom podle x souřadnice
- každý vrchol tohoto stromu množinu bodů, které mají x. souřadnici v nějakém intervalu
- ke každému vrcholu tedy stačí dostavět intervalový strom podle y souřadnice

- na přednášce jste viděli: $O(\lg^2 n)$ složitost počítání
- nyní si ukážeme $O(\lg n)$
  - použijeme zjednodušení fractional cascading (zlomkové kaskádování)
- prozatím si to ukážeme staticky
- tj. x souřadnice bude strom, ale y bude „pouze" seřazené pole
- při počítání hledáme vrcholy reprezentující interval $[x_l, x_r]$ a v každém poli podle y souřadnice děláme binární půlení a hledáme $[y_l, y_r]$





- zneužijeme toho, že pole dítěte je „podpolem" rodiče
- toho využijeme, abychom ušetřili potřebný počet binárních půlení
- pro každý prvek a pole podle y souřadnic si spočítáme jeho předchůdce v poli dětí
  - označme pole rodiče A, pole dítěte B
  - pro $a \in A$ hledáme největší $b \in B$ t.ž. $b \leq a$
  - to může být to a samotné, pokud $a \in B$
  - to lze dělat v lin. čase šoupáním pointerů



- nyní hledáme předchůdce pro a a víme, že předchůdce a' je b'
- $a' < a \Rightarrow$ předchůdce b pro a musí být vpravo od b'
- pointery se vždy šoupou doprava

$\rightarrow$ čas $O(|A| + |B|)$

- nyní uvažme intervalový dotaz na $[x_l, x_r] \times [y_l, y_r]$
- koukáme na nějaký vrchol, který reprezentuje podinterval $[x_l', x_r']$ t.ž. $x_l' \geq x_l$ a $x_r' \leq x_r$
- v něm bychom správně měli dělat binární půlení
- víme však odpověď na tentýž interval $[y_l', y_r']$ z rodiče

$\rightarrow$ podíváme se na předchůdce odpovědi v rodiči a tím ušetříme bin. půlení

- takže pro hledání podle y souřadnic stačí jedno bin. půlení (v kořeni stromu podle x) a pak $O(\log n)$ následování předchůdci

→ celkem $\Theta(\lg n)$