Datové struktury I

11. přednáška: Geometrické datové struktury

Jirka Fink

https://ktiml.mff.cuni.cz/~fink/

Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta
Univerzita Karlova v Praze

Zimní semestr 2024/25

Intervalový dotaz (Range query)

Popis problému

- Máme dánu množinu S obsahující n bodů z $\mathbb{R}^d$
- Intervalem rozumíme d-dimenzionální obdélník, např. $\langle a_1, b_1 \rangle \times \cdots \times \langle a_d, b_d \rangle$
- Operace QUERY: Najít všechny body v daném intervalu
- Operace COUNT: Určit počet bodů v daném intervalu

Aplikace

- Počítačová grafika, výpočetní geometrie
- Databázové dotazy, např. určit zaměstnance ve věku 20-35 a platem 20-30 tisíc

Intervalový dotaz v $\mathbb{R}^1$

Staticky

Body uložíme do pole

BUILD: $\mathcal{O}(n \log n)$

COUNT: $\mathcal{O}(\log n)$

QUERY: $\mathcal{O}(k + \log n)$

k je počet vyjmenovaných bodů

Dynamicky

Body uložíme do vyhledávacího stromu

BUILD: $\mathcal{O}(n \log n)$

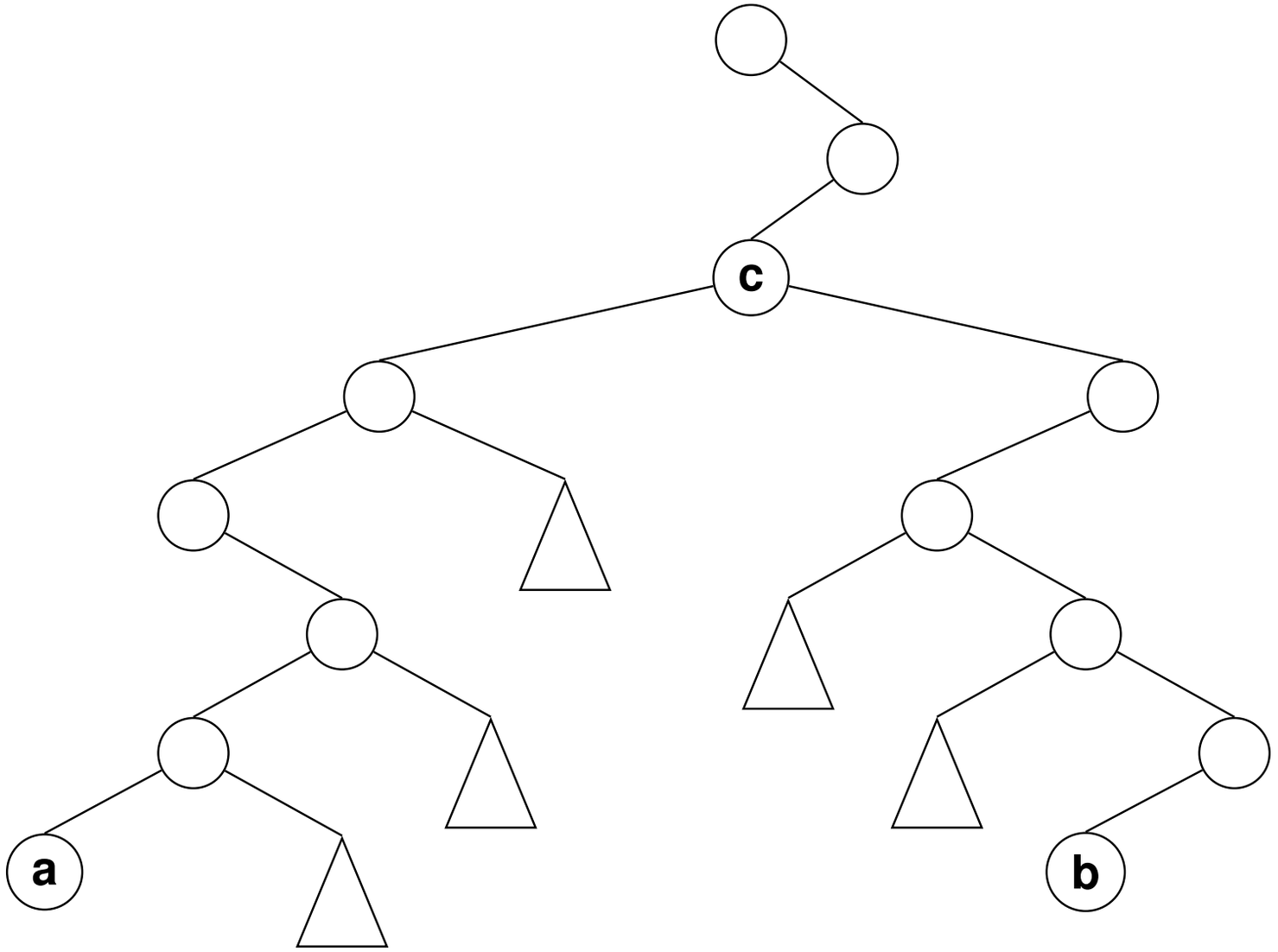
INSERT: $\mathcal{O}(\log n)$

DELETE: $\mathcal{O}(\log n)$

COUNT: $\mathcal{O}(\log n)$

QUERY: $\mathcal{O}(k + \log n)$

Intervalový dotaz v $\mathbb{R}^1$: Příklad



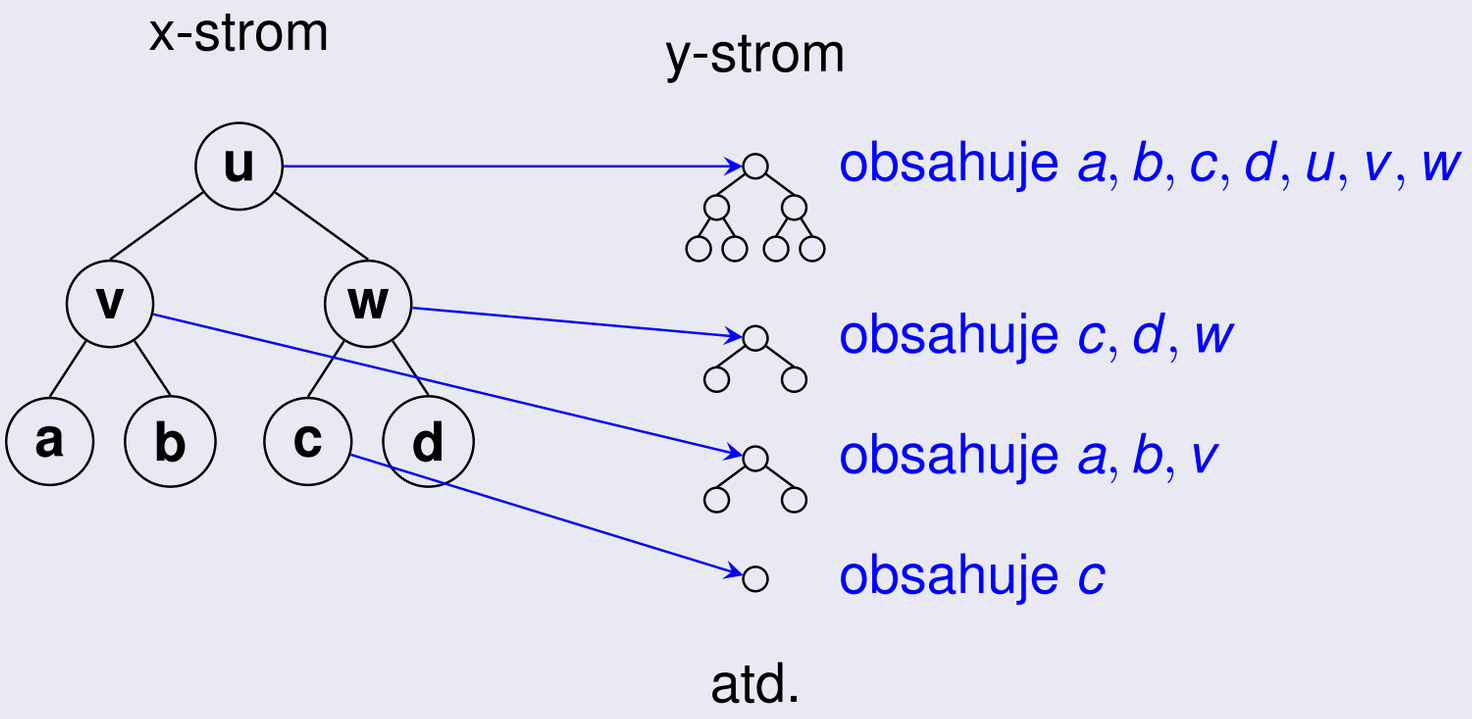
Vrchol *a* je nejmenší prvek v intervalu, *b* je největší prvek v intervalu a *c* je poslední společný vrchol na cestách z kořene do vrcholů *a* a *b*.

Intervalové stromy v $\mathbb{R}^2$ (Range tree)

Konstrukce

- Vybudujeme binární vyhledávací strom podle x-ové souřadnice bodů (x-strom)
- Nechť S_u je množina bodů v podstromu vrcholu u
- Každý vrchol u vybudujeme jeden binární vyhledávací strom podle y-ové souřadnice obsahující S_u
- V každém vrcholu je uložen jeden bod ① ②

Příklad



1. Podobně jako ve vyhledávacích stromech můžeme uvažovat dvě varianty: prvky mohou být uloženy ve všech vrcholech nebo jen v listech. V intervalových stromech předpokládáme, že v každém vrcholu je uložen jeden bod.
2. Pozor! Každý bod je uložen v několika vrcholech, takže paměťová složitost není lineární.

Intervalové stromy v $\mathbb{R}^2$: Paměť

Kde všude je uložený jeden vrchol?

Každý bod p je uložen v právě jednom vrcholu v x-stromu a dále je bod p uložen v každém y-stromu přiřazenému vrcholu na cestě z x-kořene do v.

Předpoklad

Předpokládejme, že binární vyhledávací strom použitý v intervalových stromech je vyvážený, a tedy jeho výška je $\Theta(\log n)$.

Paměťová složitost

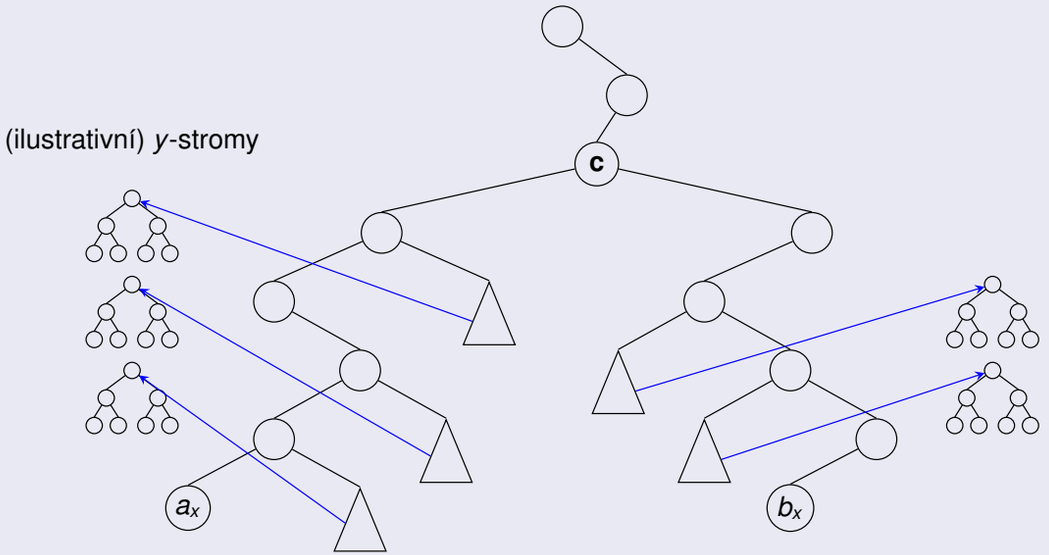
Každý bod je uložen v $\mathcal{O}(\log n)$ y-stromech a celková paměťová složitost je $\mathcal{O}(n \log n)$.

Intervalové stromy v $\mathbb{R}^2$: Dotaz na interval $\langle a_x, b_x \rangle \times \langle a_y, b_y \rangle$

Idea algoritmu vyhledávání

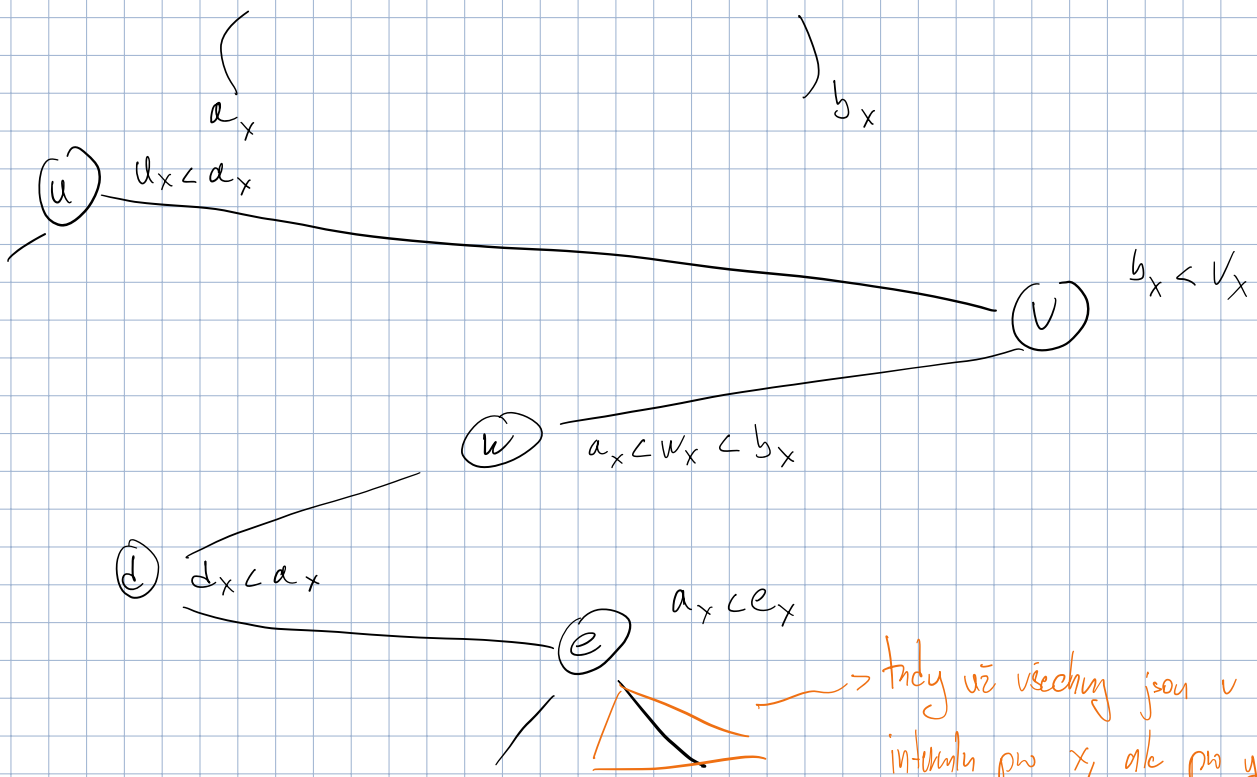
1. Najít klíče a_x a b_x v x-stromu ①
2. Určit vrcholy u x-stromu takové, že S_u obsahuje pouze body s x-ovou souřadnicí v intervalu $\langle a_x, b_x \rangle$ ②
3. V těchto vrcholech položme y-ový dotaz $\langle a_y, b_y \rangle$

Příklad



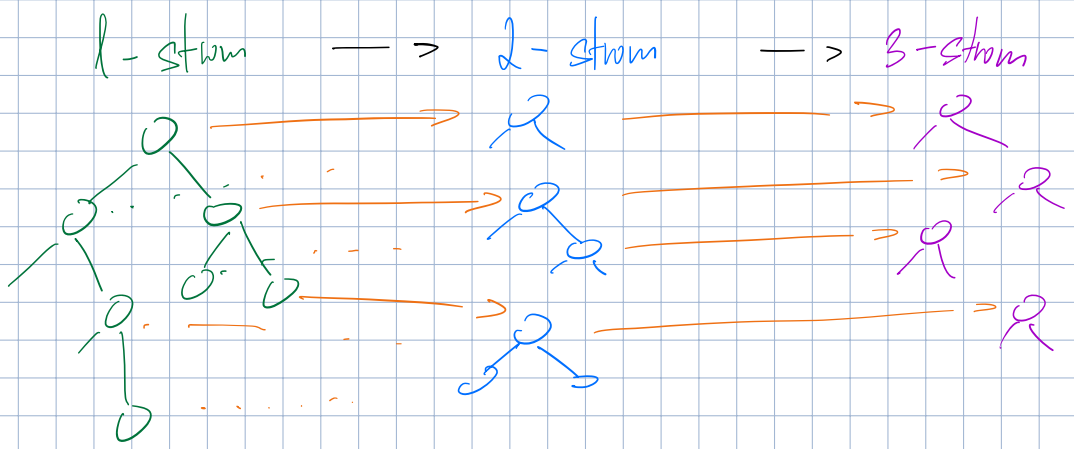
Složitost dotazu COUNT

$\mathcal{O}(\log^2 n)$ protože y-ový dotaz je volán v $\mathcal{O}(\log n)$ y-stromech ③ ④



mám logn vnitřních stromů
v každém může spravit logn kroků.
Celkem mám proto $\log^2 n$ kroků.

↳ u d-rozměrného případu jenom rozdíl větším do další dimenze



takhle se to větví do další dimenze.
Implementace je ale těžká, jde jen o dva pointery left, right navíc, které rozdělují rozsah a size
↳ lze implementovat rekurzí.

1. Přesněji: najít ze všech prvků v x-stromu dva body mající nejmenší a největší x-ovou souřadnic ležící intervalu $\langle a_x, b_x \rangle$.
2. Je zřejmé, že když vrchol tuto podmínku splňuje, tak ji splňují i synové vrcholu. Proto nás zajímají nejvýše umístěné vrcholy s touto vlastností, tj. vrcholy splňující tuto podmínky, ale jejichž otec tuto podmínku nesplňuje.
3. Všimněme si, že prohledávané y-stromy obsahují po dvou disjunktní množiny prvků.
4. V dotazu QUERY je nutné vyjmenovat všechny body, a proto složitost je $\mathcal{O}\left(k + \log^2 n\right)$, kde k je počet bodů v obdélníku.

Intervalové stromy v $\mathbb{R}^d$ (předpokládáme $d \geq 2$)

Popis

- i-strom je binární vyhledávací strom podle i-té souřadnice pro $i = 1, \ldots, d$
- Pro $i < d$ má každý vrchol u i-stromu ukazatel na $(i+1)$-strom obsahující S_u
- Intervalovým stromem rozumíme všechny výše popsané stromy

Reprezentace

Struktura vrcholu intervalového stromu obsahuje

key	nadrovina rozdělující prostor mezi syny ①
left, right	ukazatel na levého a pravého syna
tree	ukazatel na kořen $(i+1)$-stromu
size	počet bodů v podstromu (pokud potřebujeme dotaz COUNT)

Poznámka

Nechť u je vrchol i-stromu a T jsou všechny vrcholy dosažitelné opakovaným přístupem k ukazatelům left, right a tree z vrcholu u. Pak T tvoří intervalový strom na vrcholech S_u a souřadnicích $i, \ldots, d$.

1. Operace QUERY musí umět body ležící v obdélníku i vypsat, a proto potřebuje mít ve všech vrcholech uloženy všechny souřadnice bodu. Operaci QUERY stačí klíč, což v i-stromu je i-tá souřadnice bodu.

Intervalové stromy v $\mathbb{R}^d$: Struktura a paměťová složitost

V kolika vrcholech je uložený bod b?

- Existuje $\mathcal{O}(\log n)$ vrcholů u 1-stromu takových, že S_u obsahuje b
- Tedy počet 2-stromů obsahující b je $\mathcal{O}(\log n)$
- Uvažujme i-strom T obsahující bod b
- Pak v T je $\mathcal{O}(\log n)$ vrcholů w takových, že $b \in S_w$ ①
- Počet $(i+1)$-stromů přiřazených nějakému vrcholu T obsahujících b je $\mathcal{O}(\log n)$
- Každou dimenzí se počet stromů obsahujících b zvyšuje o multiplikativní faktor $\mathcal{O}(\log n)$
- Celkový počet stromů/listů obsahujících b je $\mathcal{O}\left(\log^{d-1} n\right)$
- Celková paměťová složitost je $\mathcal{O}\left(n \log^{d-1} n\right)$

Kolik má intervalový strom i-stromů?

- Počet vrcholů ve všech $(i-1)$-stromech je $\mathcal{O}\left(n \log^{i-2} n\right)$
- Každému vrcholu $(i-1)$-stromu je přiřazen jeden i-strom
- Počet i-stromů je $\mathcal{O}\left(n \log^{i-2} n\right)$ ②

1. Strom T nemusí obsahovat všechny prvky, a proto jeho výška nemusí být $\Omega(\log n)$. Dokonce většina stromů obsahuje celkem malý počet bodů.
2. Platí pro $i \geq 2$. Pro $i = 1$ máme jeden 1-strom.

Intervalové stromy v $\mathbb{R}^d$: BUILD

Algoritmus (Body M jsou v poli setříděné podle poslední souřadnice)

```
1  Procedure BUILD(množina bodů M, dimenze stromu d, aktuální souřadnice i)
2      if |M| = 1 then
3          return nový list obsahující jediný vrchol M ①
4      if i = d then                    ← když mám setříděno podle poslední souřadnice, můžu v O(n) vyhledat BVS
5          return kořen stromu vytvořený ze setříděného pole
6      v ← nový vrchol
7      v.tree ← BUILD(M, d, i + 1)
8      v.key ← medián i-tých souřadnic bodů M   ← stačí lze v lineárním čase
9      M_l, M_r ← rozděl M na body mající i-tou souřadnici menší a větší než v.key   (pivot je medián)
10     v.left ← BUILD(M_l, d, i)
11     v.right ← BUILD(M_r, d, i)
12     return v
```

Složitost jednoho volání funkce BUILD (bez rekurze)

- Pro $i = d$ je složitost $\mathcal{O}(1)$ ②
- Pro $i < d$ je složitost ~~$\mathcal{O}(|S|)$~~ ③ $O(|M|)$ → lin. čas na pivota a rozdělení

1. Zde je otázka, zda list musí mít přiřazený (triviální) strom další dimenze. Je to implementační detail, intervalový strom může fungovat v obou verzích a asymptotické složitosti se nemění.
2. Předpokládáme, že množina stromů M předávaná v rekurzi se udržuje setříděná. Čas jednoho volání funkce `BUILD` je $\mathcal{O}(1)$ pro $i = d$, protože medián leží uprostřed pole M a rozdělení M na M_l a M_r je jen otázka předání správných ukazatelů.
3. Pro $i < d$ není pole M setříděné podle aktuální souřadnice i, a proto nalezení mediánu a rozdělení pole trvá $\mathcal{O}(n_T)$. Časovou složitost vytvoření i-stromu T lze popsat rekurentní formulí $f(n) = 2f(n/2) + \mathcal{O}(n)$, jejíž řešení je $\mathcal{O}(n_T \log n_T)$.

Vytvoření d-stromů

- d-stromy vytváříme v konstantním čas na vrchol
- Počet vrcholů ve všech d-stromech je $\mathcal{O}(n \log^{d-1} n)$
- Časová složitost vytvoření všech d-stromů je $\mathcal{O}(n \log^{d-1} n)$

lip to nepůjde, protože tohle je samotná pam. složitost

Vytvoření všech i-stromů pro $i = 1, \dots, d-1$ (nepočítaje $(i+1)$-stromy)

- Počet vrcholů ve všech i-stromech je $\mathcal{O}(n \log^{i-1} n)$
- Nechť n_T je počet vrcholů v i-stromu T
- Vybudování samotného stromu T trvá $\mathcal{O}(n_T \log n_T)$
- Vybudování všech i-stromů trvá

tohle je defacto mergesort hrubého pole říká $O(n_T \log n_T)$

$\forall n_T: \log n_T \le \log n$; $n \cdot \log^{i-1} n$

$$\sum_{i\text{-strom } T} n_T \log n_T \le \log n \sum_{i\text{-strom } T} n_T = \log n \cdot n \log^{i-1} n = n \log^i n$$

Časová složitost operace BUILD

$\mathcal{O}(n \log^{d-1} n)$ → proč nám nevadí, že uvnitř musíme třídit? Sice třízení zabere $\log n$-krát víc, ale je jich $\log n$-krát méně...

Intervalové stromy v $\mathbb{R}^d$: QUERY $\langle a_1, b_1 \rangle \times \cdots \times \langle a_d, b_d \rangle$


```
1  Procedure Query(vrchol v, aktuální souřadnice i)
2      if v = NIL then
3          return
4      if v.key ≤ a_i then
5          Query(v.right, i)
6      else if v.key ≥ b_i then
7          Query(v.left, i)
8      else
9          if v.point leží v obdélníku then
10             Vypiš v.point
11         Query_left(v.left, i)
12         Query_right(v.right, i)
```

Intervalové stromy v $\mathbb{R}^d$: QUERY $\langle a_1, b_1 \rangle \times \cdots \times \langle a_d, b_d \rangle$

– jdeme od nejhlubšího společného předka

```
1  Procedure Query_left(vrchol v, aktuální souřadnice i)
2      if v = NIL then
3          return
4      if v.key < a_i then
5          Query_left(v.right, i)
6      else
7          if v.point leží v obdélníku then
8              Vypiš v.point
9          Query_left(v.left, i)
10         if i < d then
11             Query(v.right.tree, i + 1)
12         else
13             Vypiš všechny body v podstromu vrcholu v.right
```

→ pokud v této souřadnici splňuje podmínku, tak jdu na další souřadnici

Intervalové stromy v $\mathbb{R}^d$: QUERY

Složitost operace COUNT

- V každém stromě přistoupíme k nejvýše dvěma vrcholům z každé vrstvy
- Z každého navštíveného i-stromu pokračujeme do $\mathcal{O}(\log n)$ $(i+1)$-stromů
- Počet navštívených i-stromů je $\mathcal{O}\left(\log^{i-1} n\right)$
- Celková složitost je $\mathcal{O}\left(\log^{d} n\right)$

Složitost operace QUERY

- Vypsání všech bodů v podstromu trvá $\mathcal{O}(k)$, kde k je počet nalezených bodů
- Celková složitost je $\mathcal{O}\left(k + \log^{d} n\right)$

↳ query dělá to stejný jako count, jen to ještě musí vypsat

BB[α]-strom

- Binární vyhledávací strom
- Počet listů v podstromu vrcholu u označme s_u
- Podstromy obou synů každého vrcholu u mají nejvýše αs_u listů

Operace Insert (Delete je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)
- Jestliže některý vrchol u porušuje vyvažovací podmínku, tak celý jeho podstrom znovu vytvoříme operací BUILD (složitost $\mathcal{O}(s_u)$) Amortizace zajišťuje log n složitost místo n.

Amortizovaná časová složitost operací Insert a Delete

- Jestliže podstrom vrcholu u po provedení operace BUILD má s_u listů, pak další porušení vyvažovací podmínky pro vrchol u nastane nejdříve po $\Omega(s_u)$ přidání/smazání prvků v podstromu vrcholu u <= každým insertem ušetří peníze, pak s nimi zaplatí lin. rebuild
- Amortizovaný čas vyvažovaní jednoho vrcholu je $\mathcal{O}(1)$
- Při jedné operaci Insert/Delete se prvek přidá/smaže v $\mathcal{O}(\log n)$ podstromech
- Amortizovaný čas vyvažovaní při jedné operaci Insert nebo Delete je $\mathcal{O}(\log n)$

Intervalové stromy v $\mathbb{R}^d$: Dynamizace pomocí BB[α]-stromů

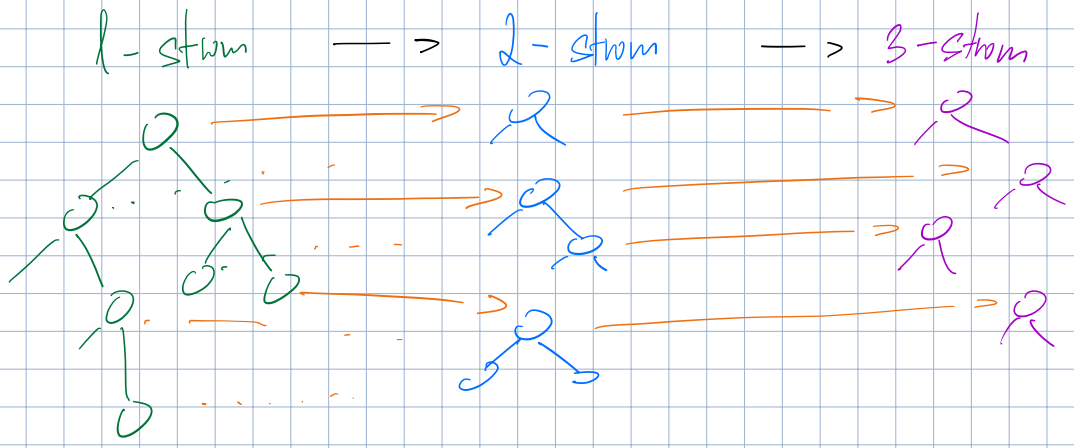
Použití BB[α]-stromů v intervalových stromech

- Binární vyhledávací stromy implementujeme pomocí BB[α]-stromů
- Vyžaduje-li BB[α]-strom vyvážení, pak přebudujeme všechny přiřazené stromy

Složitost operace Insert a Delete

- Navštívených i-stromů je $\mathcal{O}(\log^{i-1} n)$ a v každém navštívíme $\mathcal{O}(\log n)$ vrcholů
- Složitost bez přebudování je $\mathcal{O}(\log^d n)$; analyzujme přebudování
- Uvažujme libovolný vrchol u, který leží v i-stromu
- Přebudování vrcholu u trvá $\mathcal{O}(s_u \log^{d-i} s_u)$ → od i do d musím přebudovat i ostatní stromy uvnitř
- Přebudování vrcholu u může nastat po $\Omega(s_u)$ po přidání/smazání prvků v podstromu u $s_u \leq n$
- Amortizovaná cena přidání/smazání do vrcholu u je $\mathcal{O}(\log^{d-i} s_u) \leq \mathcal{O}(\log^{d-i} n)$
- Amortizovaný čas operace Insert a Delete je $\sum_{i=1}^{d} \mathcal{O}(\log^{i-1} n)\mathcal{O}(\log n)\mathcal{O}(\log^{d-i} n) = \mathcal{O}(\log^d n)$ → v každém stromu platím jinou cenu $\log^{d-i} n \cdot \log^{i-1} n = \log^{d-1} n$ ← vůbec to nezávisí na i

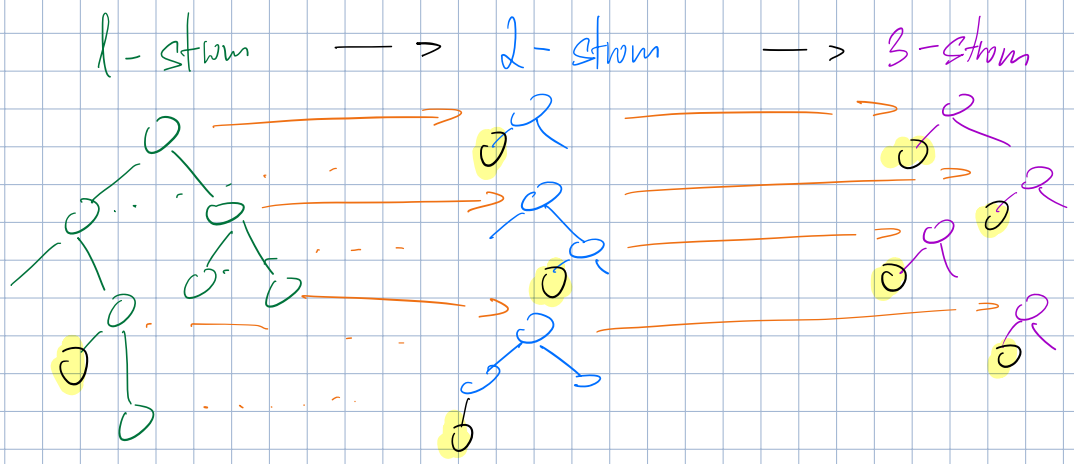
Proč nemůžu použít třeba AVL strom?



↳ Libovolná hrana by vozila pointery na další dimenzi...

Naopak BB[α] stromy se chovají zodpovědně a nepřestavují se.

Když přidávám vrchol v, tak ho musím na správném místě přidat i ve všech ostatních stromech



↳ nahoře je popsáno, jak to vyjde se složitostí...

Výsledek $O(\log^d n)$ je velmi hezký, jelikož to stále drží hodnotu $O(\log n)$ za každou dimenzi → BVS (jakožto 1-dim) je taky $O(\log n)$...

Další známá vylepšení intervalových stromů

Popsaný postup

QUERY: $\mathcal{O}(k + \log^d n)$

Paměť: $\mathcal{O}(n \log^{d-1} n)$

S použitím Fractional cascading

QUERY: $\mathcal{O}(k + \log^{d-1} n)$ / $\log n$

Paměť: $\mathcal{O}(n \log^{d-1} n)$

Chazelle, 1990

QUERY: $\mathcal{O}(k + \log^{d-1} n)$

Paměť: $\mathcal{O}\left(n \left(\frac{\log n}{\log \log n}\right)^{d-1}\right)$ → velmi malé zlepšení

Chazelle, Guibas, 1986 (pro $d \geq 3$)

QUERY: $\mathcal{O}(k + \log^{d-2} n)$ / $\log^2 n$

Paměť: $\mathcal{O}(n \log^d n)$ ↓ za cenu $*\log n$ paměti

k-d stromy

L > tohle je často na nety, místo interních stromů

L> v i-té vrstvě dělím podle mediánu i-té dimenze, když mi dojde dimenze, jdu od zase, než mám jen jednovrcholy

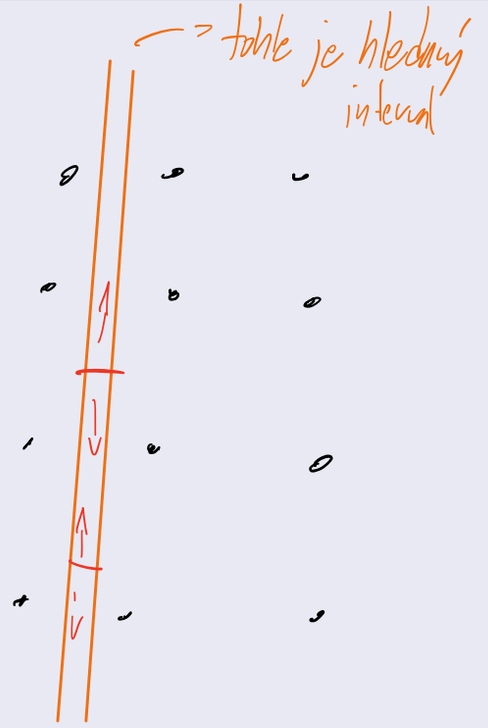
Popis

- Body uložíme do binárního stromu
- Do kořene uložíme medián podle první souřadnice
- Do levého (pravého) podstromu uložíme body mající první souřadnice menší (větší) než medián
- Vrcholy v první vrstvě pod kořenem se body rozdělují podle druhé souřadnice
- V dalších vrstvách dělíme (cyklicky) podle dalších souřadnice
- Výška stromu je $\log_2 n + \Theta(1)$
- Operace BUILD v čase $\mathcal{O}(n \log n)$
- Body je též možné ukládat jen do listů a vrcholy pak obsahují jen rozdělující nadroviny

k-d stromy: Operace QUERY, COUNT: Algoritmus

Algoritmus

```
1  Procedure QUERY(vrchol stromu v, interval R)
2      if v je list then
3          Vypiš v, pokud leží v R
4      else if rozdělující nadrovina vrcholu v protíná R then
5          QUERY(levý syn v, R)
6          QUERY(pravý syn v, R)
7      else if R je „vlevo“ od rozdělující nadroviny vrcholu v then
8          QUERY(levý syn v, R)
9      else
10         QUERY(pravý syn v, R)
```



→ můžeme se větvit často

$2 \cdot 2^{\frac{\log n}{2}} = \sqrt{n}$

v y směru půjdu vždy dvěma směry,
v x půjdu vždy jen jedním

každý druhý hook se větvím

k-d stromy: Operace QUERY COUNT: Analýza

Příklad nejhoršího případu pro $\mathbb{R}^2$

- Máme množinu bodů $S = \{(x, y); x, y \in [m]\}$, kde $n = m^2$
- Chceme najít množinu všech bodů v intervalu $\langle 1,2; 1,8\rangle \times \mathbb{R}$
- V každé vrstvě rozdělující podle y-ové souřadnice musíme prozkoumat oba podstromy
- Výška stromu je $\log_2 n + \Theta(1)$ a v polovině vrstev prozkoumáváme oba podstromy
- Celkem navštívíme $2^{\frac{1}{2}\log_2 n + \Theta(1)} = \Theta(\sqrt{n})$ listů

Příklad nejhoršího případu pro $\mathbb{R}^d$

- Mějme množinu bodů $S = [m]^d$, kde $n = m^d$
- Chceme najít množinu všech bodů v intervalu $\langle 1,2; 1,8\rangle \times \mathbb{R}^{d-1}$
- V každé vrstvě nerozdělující podle první souřadnice musíme prozkoumat oba podstromy
- V $\frac{d-1}{d}\log_2 n + \Theta(1)$ vrstvách prozkoumáváme oba podstromy
- Celkem navštívíme $2^{\frac{d-1}{d}\log_2 n + \Theta(1)} = \Theta(n^{1-\frac{1}{d}})$ listů

$[m]^d$ značí tzv. mřížové body, tj. body $\mathbb{R}^d$, jejichž každá souřadnice je celé číslo od 0 od $m-1$.

kd-stromy jsou mají nejlepší možnou časovou složitost, pokud datová struktura smí používat pouze $\mathcal{O}(n)$ paměti. Intervalové stromy umí vyhodnotit intervalový dotaz v čase $\mathcal{O}(\log^d n)$, ale potřebují $\mathcal{O}(n \log^{d-1} n)$ paměti.

Obojí má smysl, pokud je d „rozumné“; jinak je rychlejší prostě prohledávat pole.

nicméně stejně tak zabírají paměť, proto se občas můžou velmi hodit

Veselé Vánoce,

ahoj ahad. roku 2024,

snad příští rok budu v Británii!

Po Vánocích

Paralelní programování bez zámků.