12. cvičení

Geometrické datové struktury

Úloha 1 (Nejhorší případ pro k-d strom)
Mějme 2-d strom, tedy binární strom, kde rozdělujeme rovinu střídavě podle os x a y. Ukažte, že dvourozměrný intervalový dotaz (na počet bodů v obdélníku) může trvat až $\Omega(\sqrt{n})$, konkrétně najděte množinu bodů uloženou ve stromě a dotaz, který bude trvat takto dlouho, a přitom nenajde žádný bod.
Bonus: zkuste toto rozšířit na obecné k-d stromy s časovou složitostí $\Omega(n^{1-\frac{1}{k}})$.

Řešení
Vezmeme množinu $\{(i,i) : 1 \leq i \leq n\}$ a budeme se ptát na dotaz $\{0\} \times \mathbb{R}$. Pro jednoduchost mějme $n = 2^t - 1$. V každém x-vrcholu jdeme doleva, ale v každém y-vrcholu musíme jít oběma směry. Strom má hloubku $t \approx \log n$, a v každém druhém kroku zdvojnásobíme počet podstromů, na které se musíme podívat. Celková složitost tedy je $\Omega(2^{(\log n)/2}) = \Omega(\sqrt{n})$.
Rozšíření je analogické, jenom budeme mít k-tice $(i, \ldots, i)$ a dotaz bude $\{0\} \times \mathbb{R}^{k-1}$.

Úloha 2 (Hledání minima a mazání v k-d stromě)
Máme k-d strom. Navrhněte, jak na něm implementovat operaci nalezení nejmenšího prvku v dané dimenzi. (Tedy funkci FindMin(d), která nalezne prvek, jehož d-tá souřadnice je minimální.)
Dále navrhněte implementaci operace Delete. Pro jednoduchost můžete předpokládat, že žádné dva body spolu nesdílejí žádnou souřadnici.
(V obou případech neřešte časovou složitost.)

Řešení
Pro FindMin: Ve vrcholu, který řeže dimenzi rozdílnou od d-té vezmeme minimum z obou podstromů a sebe. Ve vrcholu, který řeže d-tou dimenzi, vezmeme minimum z podstromu odpovídajícímu nižší hodnotě souřadnic (a sebe).
Pro Delete: Najdeme vrchol, který mažeme, a nalezneme minimum v pravém podstromě pro souřadnici, podle které vrchol řeže. Z něj přesuneme vrchol výše na místo, které chceme smazat, a nyní chceme smazat právě vyprázdněné místo. Tím můžeme potenciálně přejít na mazání dalšího vnitřního vrcholu, ale ten bude ve stromě níže, takže po konečném počtu iterací se dostaneme do listu, který můžeme beztrestně smazat.

Úloha 3 (Nejbližší soused a k-NN)
Máme k-d strom. Navrhněte, jak na něm implementovat operaci nalezení nejbližšího souseda, a operaci nalezení m nejbližších sousedů.
Hint: pro m-NN se může hodit mít k dispozici haldu.

Řešení
Máme-li bod x, začneme jako bychom hledali x. Jakmile dorazíme do listu, zapamatujeme si poslední vrchol jako nejbližší. Budeme se v rekurzi vracet, a v každém vrcholu se podíváme, jestli vrchol není bližší, a jestli není šance, že v druhém podstromě může být bližší vrchol. (Vzdálenost nejbližšího bodu druhého obdélníka musí být menší než vzdálenost k nejbližšímu vrcholu.) Pokud šance je, podíváme se i do druhého podstromu.
Zlepšení: to samé, jenom si nejbližší vrcholy ukládám do haldy a beru vrcholy shora, abych měl aspoň nějaké omezení.

Paralelní datové struktury

Úloha 4 (BankAccountFactory)
Máme tento pseudokód v „Javě“ pro třídu, která simuluje bankovní účet. Předpokládejte, že Lock je třída, která je standardní zámek, ale tento zámek *není možné zamykat vícekrát*, v tom případě se zaseknete při čekání na sebe, až ten zámek uvolníte. Ukažte, že v tomto případě můžete vytvořit deadlock.

```
class BankAccount {
 private int balance = 0;
 private Lock lk = new Lock();
```

```
 int getBalance() {
  lk.acquire();
  int ans = balance;
  lk.release();
  return ans;
 }

 void setBalance(int x) {
  lk.acquire();
  balance = x;
  lk.release();
 }

 void withdraw(int amount) {
  lk.acquire();
  int b = getBalance();
  if(amount > b) {
   lk.release();
   throw new WithdrawTooLargeException();
  }
  setBalance(b - amount);
  lk.release();
  }
}
```

Váš kolega to ovšem vyřešil: ukažte, že funkce `newWithdraw` nahrazující `withdraw` se nemůže deadlocknout.

```
void newWithdraw(int amount) {
 lk.acquire();
 lk.release();
 int b = getBalance();
 lk.acquire();
 if(amount > b) {
  lk.release();
  throw new WithdrawTooLargeException();
 }
 lk.release();
 setBalance(b - amount);
 lk.acquire();
 lk.release();
}
```

Nakonec ukažte, že i když se tahle nová verze nemůže deadlocknout, tak bohužel není korektní: konkrétně se nějaký výběr může „ztratit“.

Řešení
První část: ve withdraw.
Druhá část: vždycky poctivě odemykáme.
Třetí část: protože odemykáme, můžeme mít dvě vlákna, která si obě nejprve načtou zůstatek, potom obě po sobě zůstatek upraví, takže to první bude schované.

Úloha 5 (Vlastně jenom code review)
Na adrese `https://deadlockempire.github.io/` najdete hru Deadlock Empire. Cílem hry je krokovat vícevláknový kód, aby došlo k nějaké nechtěné situaci jako souběžné vykonání kritické sekce, deadlock a podobně.

Řešení
Bavte se :)

Bonusové úlohy

Úloha 6 (Dosáhneme?)
Sestrojte pravděpodobnostní algoritmus[1] pro rozhodnutí dosažitelnosti (tedy odpověď na otázku „existuje cesta z u do v?“), který selže s pravděpodobností nejvýše $1/4$, a potřebuje jenom logaritmický prostor.
Technická poznámka k logaritmickému prostoru: když máme algoritmus běžící v sublineárním prostoru, počítáme jej tak, že máme vstup na „read-only“ disku, kde k němu můžeme přistupovat, ale nemůžeme jej upravovat (ale podle potřeby si jej můžeme bez problému kopírovat).[2]
Pravděpodobně se vám bude hodit následující tvrzení: když v n-vrcholovém souvislém neorientovaném grafu vyjdeme z vrcholu u, a budeme v každém kroku uniformně náhodně vybírat ze všech sousedů vrchol, kam půjdeme v dalším kroku, střední hodnota doby než navštívíme jiný vrchol v, je nejvýše $2n^3$.
Taky se může hodit připomenout si Markovovu nerovnost: Buď X nezáporná náhodná veličina. Pak $\forall \varepsilon > 0$ platí $P[X \geq \varepsilon] \leq \frac{\mathbb{E}[X]}{\varepsilon}$. Ekvivalentně pro jakékoliv $d \geq 1$, $P[X \geq d \cdot \mathbb{E}[X]] \leq \frac{1}{d}$.

Řešení
Budeme náhodně chodit $8n^3$ kroků, pokud jsme našli, tak řekneme „dosažitelný“, pokud ne, tak řekneme „nedosažitelný“. Z Markovovy nerovnosti plyne, že pravděpodobnost neúspěchu u dosažitelného vrcholu je maximálně $1/4$, a nedosažitelnost vždycky řekneme správně.

Úloha 7 (Lyžování)
Chcete začít lyžovat, ale nemáte vlastní lyže. Ve středisku ovšem máte možnost si lyže půjčit na den za 1 dolar, nebo si je koupit za C dolarů. ($C \in \mathbb{N}$)
Na druhou stranu ale máte takovou předtuchu, že vaše nemesis nějaký den brzy ráno přijde, a zlomí vám nohy, čímž ukončí vaši lyžařskou kariéru. Problém ovšem je, že nevíte, který den to bude. Máte tedy jedinou možnost: každý den ráno zjistit, jestli se vám něco nestane, a potom se rozhodnout, jestli si lyže půjčíte nebo koupíte (pokud jste si je už nekoupili, pak na nich můžete jezdit dle libosti). Na druhou stranu byste ale rádi vaše peníze využili co nejefektivněji.
Nalezněte deterministický online algoritmus, který zaplatí vždy nejvýše dvojnásobek toho, co optimum, které dopředu ví, který den d dojde k sabotáži. Zvládnete ukázat, že váš online algoritmus je co do kompetitivnosti optimální (tj. neexistuje deterministický online algoritmus, který by vždy platil méně než $2 - \frac{1}{C}$)?

Řešení
Prvních $C - 1$ dní si lyže půjčíme, a pak si je koupíme, čímž nás to bude stát maximálně $2C - 1$, a $2C - 1$ nás to bude stát v případě, kdy by bylo optimální platit C na začátku, jinak budeme optimální.
Evidentně offline optimum je: pokud $C < d$, pak kup lyže, jinak si půjčuj, tedy cena je $\min(C, d)$.
Proč jsme online optimální: označme si jako x počet dní, kdy si lyže půjčujeme. Pak naše cena je $x + C$ (pokud $x < d$), a d jinak. Nejhorší aproximační poměr máme v den, kdy jsme si lyže koupili: pak máme cenu $x + C$. Můžeme mít dvě možné ztráty: buď jsme si lyže koupili moc pozdě, pak náš poměr je $\frac{x+C}{x+1}$ (bylo by výhodnější si ještě jednou lyže půjčit). Pokud jsme si je koupili moc brzo, tak je náš poměr $\frac{x+C}{C}$. Položíme tyhle dvě hodnoty rovné, a máme $1 + \frac{x}{C} = 1 + \frac{C-1}{x+1} \rightsquigarrow C(C-1) = x(x+1) \rightsquigarrow C - 1 = x$, což je naše optimum.

[1] Pravděpodobnostní algoritmy jste asi už potkali, ale zkráceně: bude se jednat o algoritmus, který se bude moct v každém kroku rozhodnout náhodně z několika možností

[2] Technicky tuto třídu definujeme na Turingových strojích, kde máme read-only vstupní pásku, a druhou pracovní pásku. Do prostorové složitosti se nám počítá jenom pracovní páska.