Korespondenční Seminář z Programování

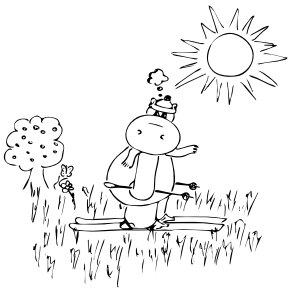
35. ročník **KSP** **Leden 2023**

Milí řešitelé, řešitelky a řešitelčata!

Dostává se k vám třetí číslo hlavní kategorie 35. ročníku KSP.

Letos se můžete těšit v každé z pěti sérií hlavní kategorie na **4 normální úlohy**, z toho alespoň jednu praktickou opendatovou. Také na **kuchařky** obsahující nějaká zajímavá informatická témata, hodící se k úlohám dané série. Občas se nám také objeví **bonusová X-ková úloha**, za kterou lze získat X-kové body. Kromě toho bude součástí sérií **seriál** na lineární algebru.

Autorská řešení úloh budeme vystavovat hned po skončení série. Pokud nás pak při opravování napadnou nějaké komentáře k řešením od vás, zveřejníme je dodatečně.



Odměny & na Matfyz bez přijímaček

Za úspěšné řešení KSP můžete být **přijati na MFF UK bez přijímacích zkoušek.** Úspěšným řešitelem se stává ten, kdo získá za celý ročník (hlavní kategorie) alespoň 50 % bodů. Za letošní rok půjde získat maximálně 300 bodů, takže hranice pro úspěšné řešitele je 150. Maturanti pozor, pokud chcete prominutí využít letos, musíte to stihnout do konce čtvrté série, pátá už bude moc pozdě. Také každému řešiteli, který v tomto ročníku z každé série dostane alespoň 5 bodů, darujeme KSP propisku, blok, nálepku na notebook a možná i další překvapení.

Termín série: **19. února 2023 ve 32:00 (tedy další ráno v 8:00)**

Odevzdávání: Přes web na adrese *https://ksp.mff.cuni.cz/h/odevzdavatko/*

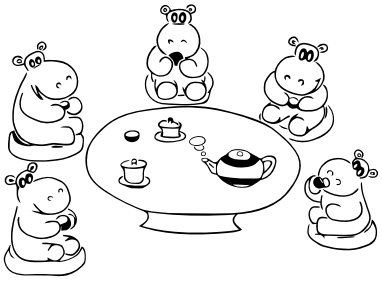
Značky úloh: Lehčí úloha (či její část) vhodná pro začátečníky; Praktická open-data úloha; Úloha, u které doporučujeme začíst se do kuchařky; Seriálová úloha

Odměna série: **Sladkou odměnu** si vysloužíte, když nám **k řešení přiložíte ručně kreslený obrázek hrošíka.** Hrošík by měl vykonávat nějakou netriviální činnost, představivosti se meze nekladou :). Pokud řešení odevzdáváte elektronicky, můžete nám oskenovaný obrázek poslat emailem na známou adresu.

Třetí série třicátého pátého ročníku KSP

35-3-1 Čajovod — 10 bodů

Ondra dostal práci v perspektivní technologické firmě. Programátoři si zde velmi potrpěli na kvalitu čaje, a tak se vedení rozhodlo postavit centrální ohřívárnu – zde se čaj připravuje. Dále bylo z ní zapotřebí čaj distribuovat čajovodem, který dostal na starosti právě Ondra.



Jak se brzy dozvěděl, čajovod sestává z uzlů a trubek mezi nimi. Každá trubka spojuje dva uzly. Poté máme (kromě obyčejných) speciální typy uzlů – ohřívárnu a místnosti. V ohřívárně se vyrábí čaj a v místnostech se čaj může spotřebovávat. V místnostech čajovod vždy končí – do každé z nich vede jen jedna trubka. Kromě toho z ohřívárny se do každého uzlu může čaj dostat právě jednou posloupností trubek a uzlů. Formálně čajovod tvoří strom, ohřívárna se nachází v kořeni a místnosti v listech.

Není to ale tak jednoduché. V místnostech buď sedí programátoři, manažeři, nebo jsou prázdné. Pokud v místnosti sedí programátoři, je zapotřebí, aby tam čaj dotekl. Naopak manažeři pijí kafe, a proto do místností s manažery téct čaj nesmí. U prázdných místností je to jedno.

Aby se tok čaje dal regulovat, na každé trubce je uzávěr, který je buď otevřený, nebo uzavřený, a podle toho skrz ni teče, nebo neteče čaj. Čaj doteče do místnosti, když všechny trubky z ohřívárny k ní jsou otevřené. Ondra potřebuje popřehazovat některé uzávěry tak, aby do místností tekl čaj, podle toho, kdo tam je. Aby se co nejméně napracoval, rád by jich změnil co nejméně. Pomůžete mu?

Toto je praktická open-data úloha. V odevzdávacím systému si necháte vygenerovat vstupy a odevzdáte příslušné výstupy. Záleží jen na vás, jak výstupy vyrobíte.

Formát vstupu: Na prvním řádku dostanete dvě čísla, počet uzlů N a počet místností M. Ohřívárna se nachází v uzlu číslo 1. Na dalších $N-1$ řádcích dostanete dvě čísla uzlů, mezi kterými vede příslušná trubka, a znak `O`, když je otevřená, jinak `Z`. Na posledních M řádcích jsou popsány místnosti. Na každém řádku je číslo uzlu a písmeno, které udává, kdo v ní sedí:

- `P` – v místnosti jsou programátoři
- `M` – v místnosti jsou manažeři
- `E` – místnost je prázdná

Formát výstupu: Vypište nejmenší počet uzávěrů, který Ondra musí přepnout, aby do místností tekl nebo netekl čaj (podle požadavků).

Lehčí varianta (za 4 body): V lichých vstupech manažeři sídlí ve vedlejší budově – v místnostech jsou buď programátoři, nebo tam není nikdo.

Ukázkový vstup:

```
8 4
1 2 O
1 3 Z
1 4 Z
2 5 O
3 6 O
4 7 O
4 8 Z
5 M
6 E
8 P
7 P
```

Ukázkový výstup:

```
3
```

35-3-2 Hroší hortenzie — 10 bodů

Lucce se na Internetu podařilo sehnat bylinný čaj Amacha z fermentovaných listů hortenzie hroší (*Hydrangea hippopotamina*). Tyto listy obsahují chemikálii zvanou *hippopodulcin*, což je látka 400–800krát sladší než sacharóza.

Lucka by ráda N svých kamarádů pohostila tímto sladkým čajem. O každém kamarádovi Lucka ví, jakou nejmenší a největší koncentraci hippopodulcinu (měřenou v mikrogramech na litr) by chtěl ve svém čaji mít. Zároveň ale chce spotřebovat co nejméně listů.

K přípravě čaje může Lucka využít některou z K mističek rozličných velikostí. Příprava čaje v i-té mističce probíhá tak, že se celá vystele listy hroší hortenzie, k čemuž je jich potřeba právě L_i. Následně se až po okraj zalije 90stupňovou vodou, čímž po minutě vznikne čaj s koncentrací C_i mikrogramů hippopodulcinu na litr. Zde pro jednoduchost předpokládáme, že listy jsou stejně velké a každý z nich vylučuje stejné množství hippopodulcinu. Z listů lze takto připravit pouze jediný nálev.

Poraďte Lucce, ve které mističce má pro každého z kamarádů připravit jeden nálev čaje tak, aby vyhověla jejich nárokům, a přitom spotřebovala celkově co nejméně listů. Mističky se dají vymývat, takže dvěma různým kamarádům může Lucka připravit čaj ve stejné nádobě.

Všechny zadané hodnoty jsou nezáporná celá čísla, přičemž koncentrace hippopodulcinu mohou být velké až 10^9 μg/l. Můžete předpokládat, že kamarádů je řádově více než mističek.

Toto je teoretická úloha. Není nutné ji programovat, odevzdává se pouze slovní popis algoritmu. Více informací zde: `http://ksp.mff.cuni.cz/viz/tinfo`

35-3-3 Prodlužovačky — 12 bodů

Kevina trápí problém. Rád si kutí a vytváří různé elektrické přístroje. Každý tento přístroj ale potřebuje být zapojený do elektřiny. Jelikož je ale Kevin nepořádný, vždy jen sáhne po nejbližší prodlužovačce, přístroj zapojí a kutí dál. Občas taky zapojí novou prodlužovačku, klidně i do jiné.

Nedávno ale zjistil, že každá prodlužovačka má určený maximální příkon, který snese. Aktuální příkon, kterému je prodlužovačka vystavena, jde určit jako součet příkonů připojených zařízení. Pro tyto potřeby se prodlužovačka, zapojená do jiné prodlužovačky, chová jako zařízení, jehož příkon se zjistí právě zmíněným způsobem.

Kevin si tedy zjistil pro každou prodlužovačku, jaký příkon snese, pro každé zařízení, jaký má příkon, a taky si udělal pořádek v tom, co je kam zapojené. Zjistil, že některé prodlužovačky jsou přetížené. Nyní by chtěl některá zařízení přepojit do zásuvek (těch má určitě dost a ty nemají omezený příkon) tak, aby žádná prodlužovačka přetížená nebyla. Prodlužek se momentálně bojí dotýkat, takže chce přepojovat pouze zařízení a nikoliv celé prodlužovačky. Jelikož se chce ale co nejdřív pustit znovu do kutění, chce přepojit co nejméně zařízení. Pomůžete mu?

Zmatek v Kevinově pokoji tvoří zakořeněný strom (tj. žádné zapojení prodlužovaček do sebe netvoří cyklus) a všechna zařízení jsou v listech tohoto stromu. Kořenem tohoto stromu je zeď, která má neomezený počet zásuvek. Pro každý list tohoto stromu (tj. pro každé zařízení) známe jeho příkon, což je kladné celé číslo. Pro každý vnitřní vrchol (tj. pro každou prodlužovačku) známe jeho kapacitu, což je nezáporné celé číslo určující maximální povolený součet příkonů listů v podstromu.

Navrhněte algoritmus, který pro zadanou soustavu prodlužovaček a zařízení vypíše, kolik nejméně a která zařízení odpojit z prodlužovaček a přepojit do zdi tak, aby nebyla pro žádnou prodlužovačku překročena kapacita.

Toto je teoretická úloha. Není nutné ji programovat, odevzdává se pouze slovní popis algoritmu. Více informací zde: `http://ksp.mff.cuni.cz/viz/tinfo`

35-3-4 Záchrana listí — 13 bodů

Honza se rozhodl založit si svou vlastní čajovnu. Vše potřebné už koupil a rozmístil. Protože místa měl málo, rozhodl se skladovat krabice s čaji na střeše své čajovny.

Aby se v krabicích s čajem dobře vyznal, tak je skladuje v řadě. Každá krabice obsahuje určité množství čaje. Kromě toho na některých krabicích jsou umístěná víka.

Honza šel na střechu sehnat čaj pro zákazníky, jenže ouha: Začalo pršet. Honza by samozřejmě chtěl zabránit tomu, aby jeho drahocenné listí zmoklo v krabicích. Všiml si, že do krabic s víkem neprší. Proto se rozhodl, že popřemisťuje

víka tak, aby hmotnost suchého čaje byla co největší. Má to jeden háček: Víka jsou přivázaná, a dají se přemístit jen do omezené vzdálenosti. Honza ale nemá čas na složité plánování, jak víka přemístit (jinak mu čaj zmokne), proto se ptá, jestli mu pomůžete.

Toto je praktická open-data úloha. V odevzdávacím systému si necháte vygenerovat vstupy a odevzdáte příslušné výstupy. Záleží jen na vás, jak výstupy vyrobíte.

Formát vstupu: Na prvním řádku najdete dvě čísla N a K – počet krabic a délku provázku. Na dalších N řádcích dostanete dvě čísla m_i a v_i, popisující i-tou krabici. Číslo m_i udává hmotnost čaje v ní a v_i, zdali má na sobě víko (`1` pokud má, jinak `0`).

Hodnota délky provázku určuje, o kolik krabic vedle se dá víko přesunout. Např. $K = 1$ znamená, že víko se dá nejdále posunout na krabice hned vedle.

Formát výstupu: Na první řádek vypište největší možné množství čaje, které má šanci Honza zachránit. Poté vypište pro každé víko index původní krabice, ke které je přivázané, a číslo krabice, na které je teď položené.

Lehčí varianta (za 7 bodů): Slibujeme, že v prvních čtyřech vstupech se dají víka přesunout nejdále na sousední krabice. Je tam tedy tedy $K = 1$.

Ukázkový vstup:

```
6 2
1 1
2 0
4 1
0 0
5 0
3 1
```

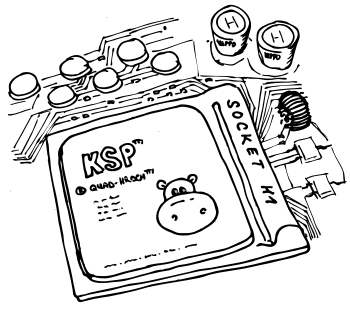
Ukázkový výstup:

```
12
3 5
6 6
1 3
```

35-3-X1 Obvody — 10 bodů

Toto je bonusová úloha pro zkušenější řešitele, těžší než ostatní úlohy v této sérii. Nezískáte za ni klasické body, nýbrž dobrý pocit, že jste zdolali něco výjimečného. Kromě toho za správné řešení dostanete speciální odměnu a body se vám započítají do samostatných výsledků KSP-X.

Jirka se snažil spravit Danovu rozbitou myšku. Proto si navrhl obvod se spoustou součástek. Chystal se obvod spájet a podíval se tedy do svého šuplíku na součástky, jenže zjistil, že mu došly veškeré rezistory.



Proto Jirka vyhrabal svoji zásobu součástek do největší krize. Tam má pro každou kladnou celočíselnou hodnotu odporu právě jeden rezistor. Ví, že do i-tého místa v obvodu potřebuje právě jeden rezistor, který může mít hodnotu odporu v jednom z M_i intervalů:

$$[\ell_1, r_1], [\ell_2, r_2], \ldots [\ell_{M_i}, r_{M_i}],$$

kde $\ell_1 \leq r_1 < \ell_2 \leq r_2 < \ldots$ a každé $\ell_j, r_j \in \mathbb{Z}^+$.

Po vás by chtěl každému místu na obvodu přiřadit jeden rezistor, který má odpor v jednom z pro něj povolených intervalů. Bohužel pro každou hodnotu odporu má rezistor pouze jeden.

Např. pro místa v obvodu s povolenými rozsahy:

$$[1, 2]$$
$$[1, 1], [3, 3]$$
$$[3, 4]$$
$$[3, 4]$$
$$[1, 1], [3, 5]$$

je jedno z možných přiřazení $2, 1, 4, 3, 5$.

Složitost algoritmu určujte vzhledem k počtu volných míst v obvodu N a celkovému počtu intervalů $M = \sum_{i=1}^{N} M_i$.

35-3-S Vektorové prostory — 15 bodů

Právě čtete třetí díl seriálu. Pokud jste předchozí díly neřešili, pro pochopení následujících odstavců je vhodné si je přinejmenším přečíst. Navíc je stále možné odevzdávat úlohy z nich za polovinu bodů.

V tomto dílu se vrátíme k již známým pojmům, jen se na ně tentokráte podíváme o trochu obecněji.

Všechny úlohy z tohoto dílu odevzdávejte dohromady v jednom zazipovaném archivu. První termín odevzdání je shodný s termínem pro třetí sérii. Poté lze odevzdávat za snížený počet bodů až do konce ročníku.

Souřadnice

Ve druhém dílu jsme zavedli pojem báze. Pro připomenutí, báze prostoru U je soubor vektorů z U, které tento prostor generují a zároveň jsou lineárně nezávislé. Také jsme potkali kanonickou bázi, tedy bázi prostoru $\mathbb{R}^d$ složenou z jednotkových vektorů.

S kanonickou bází se dobře pracuje, neboť souřadnice vektorů standardně píšeme vzhledem k ní. Můžeme je však psát i vzhledem k jiné bázi $B = \mathbf{b}_1, \ldots, \mathbf{b}_n$. Vezměme si nějaký vektor $\mathbf{v}$ a zapišme jej jako lineární kombinaci:

$$\mathbf{v} = \sum_{i=1}^{n} a_i \mathbf{b}_i$$

Poté jako *souřadnice* vektoru $\mathbf{v}$ vzhledem k bázi B nazveme koeficienty $a_1, \ldots, a_n$. Matematicky zapisujeme $[\mathbf{v}]_B = (a_1, \ldots, a_n)^\top$.

Ukažme si na příkladu. Vektor $\mathbf{v} = (5, 2)^\top$ má souřadnice vzhledem ke kanonické bázi $[\mathbf{v}]_{\text{kan}} = (5, 2)^\top$. Jeho souřadnice vzhledem k bázi $B = (1, 4)^\top, (-2, 1)^\top$ jsou $[\mathbf{v}]_B = (1, -2)^\top$.

Povšimněte si, že potřebujeme obě vlastnosti báze, aby byly souřadnice jednoznačné. Kdyby bazické vektory negenerovaly prostor, některé vektory z něj by neměly souřadnicové vyjádření. Kdyby byly lineárně závislé, jeden vektor by mohl mít více souřadnic.

Souřadnice jsme již počítali dříve, byť jsme jim samozřejmě neříkali takto, například v úloze Robot na Marsu.[1] Připomeňte si, jak výpočet probíhal.

Od té doby se však naše znalosti rozšířily. Nyní můžeme na výpočet souřadnic pohlížet také jako na zobrazení, které vektory báze B zobrazí na vektory kanonické báze. Neboť je toto zobrazení lineární, platí linearita i pro souřadnice:

$$[\mathbf{x} + \mathbf{y}]_B = [\mathbf{x}]_B + [\mathbf{y}]_B$$

[1] `http://ksp.mff.cuni.cz/viz/35-1-S`

$$[a\mathbf{x}]_B = a[\mathbf{x}]_B$$

Ještě se nám bude hodit jedno pozorování, totiž že všechny báze prostoru U mají stejný počet vektorů. Již v prvním dílu jsme nahlédli, že mezi lineárně závislými vektory jsou některé nadbytečné, i po jejich odebrání budou ty zbylé generovat stejný prostor. Obecně platí (byť to nebudeme dokazovat), že počet generátorů prostoru U bude vždy větší nebo roven počtu lineárně nezávislých vektorů v U. Máme-li dvě báze B_1 a B_2, můžeme tento fakt použít v obou směrech. Nejprve budeme B_1 považovat za generátory a B_2 za nezávislý soubor, čímž dostaneme nerovnost $|B_1| \geq |B_2|$. Prohozením získáme opačnou nerovnost, spojením obou nerovností kýženou rovnost.

Jelikož jsou všechny báze prostoru stejně veliké, můžeme definovat *dimenzi* prostoru jako velikost libovolné z bází.

Obecné prostory

Souřadnice nám nyní umožní zobecnit myšlenku vektorových prostorů. Dosud to pro nás byl pouze pojem popisující množiny $\mathbb{R}^d$, takzvané *aritmetické* prostory. Pomocí souřadnic budeme schopni převést na aritmetické prostory i jiné množiny.

Tuto myšlenku si ukažme na již známém lineárním obalu. Zvolme dva trojrozměrné vektory, například $\mathbf{u}_1 = (2,2,2)^\top$, $\mathbf{u}_2 = (3,4,5)^\top$, a zaveďme lineární obal $U = \operatorname{span}\{\mathbf{u}_1, \mathbf{u}_2\}$. Množina U má podobu roviny, nejedná se však o prostor $\mathbb{R}^2$ (třeba $\mathbf{u}_1$ v $\mathbb{R}^2$ neleží). Přesto s ním úzce souvisí.

Daná souvislost vynikne, pokud prostor U popíšeme pomocí báze $B = \mathbf{u}_1, \mathbf{u}_2$. Souřadnice jakéhokoli vektoru z U vzhledem k bázi B mají podobu dvourozměrného vektoru, tedy vektoru z $\mathbb{R}^2$. Dokud tedy pracujeme se souřadnicemi, oba prostory jsou totožné; rozdíl tkví pouze v použité bázi. Poznatky, které jsme získali pro aritmetické prostory, můžeme převést do obecných prostorů:

- Chceme-li ověřit nezávislost vektorů $\mathbf{x}, \mathbf{y} \in U$, stačí ověřit nezávislost jejich souřadnic $[\mathbf{x}]_B, [\mathbf{y}]_B$.
- Obdobně, chceme-li vyjádřit vektor $\mathbf{v} \in U$ jako lineární kombinaci jiných vektorů, stačí najít koeficienty, které vyjádří $[\mathbf{v}]_B$ pomocí souřadnic daných vektorů.

Zmíněná tvrzení platí díky linearitě souřadnic.

Prostory funkcí

Proč se však omezovat na prostory aritmetických vektorů? Když už umíme reprezentovat prostory pomocí souřadnic, můžeme si dovolit exotičtější prostory.

Vektorovým prostorem může být například množina Q kvadratických funkcí $\mathbf{q}(x) = ax^2 + bx + c$. Slovo *vektor* se v kontextu prostorů používá jako obecný termín pro jednotlivé prvky množiny, v tomto případě pro konkrétní funkce. Jeden možný vektor by tedy byl $\mathbf{q}_1(x) = 3x^2 + 5$. Po vektorech chceme, abychom je mezi sebou mohli sčítat a násobit skalárem, což kvadratické funkce splňují – sečteme odpovídající koeficienty, případně je všechny vynásobíme příslušným skalárem. Aby se jednalo o vektorový prostor, ještě musí být splněno pár dalších podmínek, jak si ukážeme za chvíli.

Jedna možná báze tohoto prostoru je $B = 1, x, x^2$. Souřadnicemi vektoru by byly jednoduše koeficienty $(c, b, a)^\top$.

Pro různá použití se však hodí různé báze. Mohlo by být výhodnější reprezentovat funkce pomocí jejich funkčních hodnot v bodech $x = -1, 0, 1$ (tři body určují kvadratickou funkci jednoznačně). Hledáme tedy bázi $C = \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$, aby souřadnice vzhledem k C byly $[\mathbf{q}]_C = (\mathbf{q}(-1), \mathbf{q}(0), \mathbf{q}(1))^\top$.

Jak takovou bázi najít? Inu, musí platit $[\mathbf{c}_1]_C = (1,0,0)^\top$. Dostáváme soustavu tří rovnic pro neznámé a, b, c:

$$\begin{aligned} \mathbf{q}(-1) &= a - b + c = 1 \\ \mathbf{q}(0) &= c = 0 \\ \mathbf{q}(1) &= a + b + c = 0 \end{aligned}$$

Řešení je $a = 1/2, b = -1/2, c = 0$, takže $\mathbf{c}_1 = 1/2x^2 - 1/2x$. Obdobně pro $\mathbf{c}_2$ a $\mathbf{c}_3$.

$$\begin{aligned} \mathbf{c}_1(x) &= \tfrac{1}{2}x(x-1) \\ \mathbf{c}_2(x) &= -(x+1)(x-1) \\ \mathbf{c}_3(x) &= \tfrac{1}{2}(x+1)x \end{aligned}$$

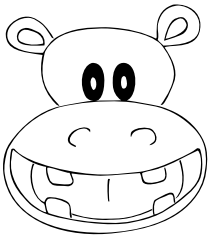
Tuto myšlenku lze rozšířit. Máme $n+1$ bodů v rovině (x_i, y_i) a hledáme polynom n-tého stupně, který těmito body prochází. Řešení nabízí *Lagrangeův interpolanční polynom*. Zvolíme bázi $L = \mathbf{p}_0, \ldots, \mathbf{p}_n$, kde

$$\mathbf{p}_i = \prod_{\substack{0 \leq j \leq n \\ j \neq i}} \frac{x - x_j}{x_i - x_j}.$$

Jedná se o stejné polynomy, jaké jsme dostali pro kvadratické funkce. Po polynomu $\mathbf{p}_i$ chceme, aby měl v bodě x_i hodnotu 1, v ostatních x_j hodnotu 0. Nulové hodnoty zajistíme výrazem $x - x_j$, který navíc vhodně přenásobíme, aby byl celý výraz v bodě x_i roven 1. Polynom se souřadnicemi $(y_0, \ldots, y_n)^\top$ vzhledem k bázi L tedy prochází všemi body. Převedením do báze $1, x, \ldots, x^n$ získáme jeho koeficienty.

Síla abstrakce

Prostor funkcí nebyl dosti exotický? Tak co množina smějících se a mračících se hrochů?



Dokonce i to může být vektorový prostor, jen bychom museli zadefinovat, jak se bude chovat sčítání hrochů a jejich násobení skalárem. Nejspíše bychom chtěli sčítat jejich nálady, přičtením smějícího se hrocha by se úsměv zvětšil, přičtením mračícího se hrocha zase zmenšil. Nepůjde využít standardní sčítání a násobení, vždyť se nejedná o čísla, museli bychom si tak vymyslet vlastní operace. Jak se dozvíme za chvíli, tyto operace však musí splňovat určitá pravidla.

Pomalu přichází čas uvést si přesnou definici vektorového prostoru. Co vlastně od takové definice očekáváme? Na jednu stranu bychom chtěli, aby pro každý vektorový prostor platily poznatky, které jsme učinili o aritmetických prostorech. Na druhou stranu bychom rádi umožnili, aby si každý mohl vymyslet svůj vlastní vektorový prostor, s vlastním sčítáním a násobením, třeba smějící se hrochy.

Nemůžeme tedy popsat vektorový prostor přímo. Místo toho ponecháme definici plně abstraktní, pouze vyjmenujeme vlastnosti, které musí splňovat. Tyto vlastnosti se nazývají *axiomy* a jsou to jediné, z čeho můžeme chování vektorových prostorů odvozovat.

Pro vyřešení úloh není nutné axiomy znát, ba ani pochopit. Dají vám však malý náhled do světa matematiky za hranicí středoškolského učiva.

Vektorový prostor je množina V, kde platí:

1) Asociativita sčítání: $\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$
2) Komutativita sčítání: $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$
3) Existence nuly: $\exists \mathbf{0}\, \forall \mathbf{x}\colon \mathbf{0} + \mathbf{x} = \mathbf{x} + \mathbf{0} = \mathbf{x}$
4) Existence opačného prvku: $\forall \mathbf{x}\, \exists \mathbf{x}'\colon \mathbf{x} + \mathbf{x}' = \mathbf{x}' + \mathbf{x} = \mathbf{0}$
5) Asociativita násobení: $a(b\mathbf{x}) = (ab)\mathbf{x}$
6) Existence jedničky: $1\mathbf{x} = \mathbf{x}$
7) Distributivita poprvé: $a(\mathbf{x} + \mathbf{y}) = a\mathbf{x} + a\mathbf{y}$
8) Distributivita podruhé: $(a + b)\mathbf{x} = a\mathbf{x} + b\mathbf{x}$

Není-li uvedeno jinak, dané vlastnosti musí platit pro všechny vektory $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$ a všechny skaláry a, b.

Standardní sčítání a násobení všechny tyto axiomy splňuje. V prostoru hrochů by například pravidlo 3 znamenalo, že musí existovat hroch s neutrálním výrazem, který se ani nesměje, ani nemračí, a jeho přičtení k jinému hrochovi nezpůsobí žádnou změnu. Pravidlo 4 říká, že hrochy lze rozdělit do dvojic s opačnou náladou, jen chudák neutrální hroch bude ve dvojici sám se sebou. Poslední 4 pravidla předurčují, že násobení se musí chovat jako opakované sčítání. Například z pravidel 8 a 6 vyplývá:

$$2\mathbf{x} = (1 + 1)\mathbf{x} = 1\mathbf{x} + 1\mathbf{x} = \mathbf{x} + \mathbf{x}$$

Aby toho nebylo málo, lze přidat ještě jedno zobecnění. Místo skalárů z $\mathbb{R}$ můžeme vektorový prostor definovat nad jakýmkoli *tělesem*, například nad $\mathbb{Q}$ nebo nad $\mathbb{C}$. (Pozor, obory $\mathbb{N}$ a $\mathbb{Z}$ tělesa nejsou!) V informatice se hodí těleso $\mathbb{Z}_2$, které má prvky 0 a 1 a sčítání se tam chová jako binární XOR.

Zobrazení mezi prostory

Zobrazení jsme doteď vnímali jako transformace obrázků, tedy transformace v $\mathbb{R}^d$. Nic však nebrání tomu, aby zobrazení jako vstup dostalo vektor z prostoru U a jako výstup vydalo vektor z prostoru V. Říkáme, že zobrazení f je z U do V, matematicky $f\colon U \to V$.

Menší zádrhel přijde ve chvíli, kdy budeme chtít zobrazení popsat maticí. Jak jsme odvodili v minulém dílu, ve sloupcích matice jsou obrazy jednotkových vektorů. Jednotkových proto, že jsme pracovali v kanonické bázi. Dokonce i obrazy samotné jsme zapsali vzhledem k ní. Jenže v obecných prostorech žádná kanonická báze existovat nemusí.

Musíme si tedy vybrat bázi, kterou bude matice používat. A to hned dvě, jednu pro prostor U a jednu pro prostor V. Matice tak vlastně bude zobrazení popisovat v řeči souřadnic. Pro bázi B_U prostoru U a bázi B_V prostoru V budeme matici zobrazení f značit ${}_{B_V}[f]_{B_U}$.

Ukažme si příklad na prostoru kvadratických funkcí Q zavedeném výše. Nejprve uvážíme zobrazení $f\colon Q \to \mathbb{R}$, které vyhodnotí funkci v bodě $x = 3$. Prostor Q má dimenzi 3, prostor $\mathbb{R}$ dimenzi 1, matice tak bude mít rozměry 3×1. Použijeme-li bázi $B = 1, x, x^2$, dostaneme matici:

$${}_{\text{kan}}[f]_B = \begin{pmatrix} 1 & 3 & 9 \end{pmatrix}$$

A obraz vektoru $\mathbf{q}_1(x) = 3x^2 + 5$ vypočteme následovně:

$$[f(\mathbf{q}_1)]_{\text{kan}} = {}_{\text{kan}}[f]_B \cdot [\mathbf{q}_1]_B = \begin{pmatrix} 1 & 3 & 9 \end{pmatrix} \begin{pmatrix} 5 \\ 0 \\ 3 \end{pmatrix} = (32)$$

Povšimněte si, že v dolních indexech jsou stejné báze u sebe, jedině tak jsou vektory s maticí kompatibilní.

Jako druhý příklad použijeme zobrazení $g\colon Q \to Q$, které funkci zrdcadlově převrátí podél osy y. To bude naopak jednodušší v bázi C, pouze se prohodí funkční hodnoty v bodech $x = -1$ a $x = 1$:

$${}_C[g]_C = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Přechod mezi bázemi

Matice výše pracuje s bází C. Pokud bychom měli vektor v bázi B, museli bychom jej přeložit. Matice, která to zajistí, se nazývá *matice přechodu* a značí se ${}_C[\mathrm{id}]_B$. Ve značení využíváme *identitu* – zobrazení, které nic nemění. Podobu této matice jsme již naznačili v sekci o souřadnicích, v jejích sloupcích budou vektory báze B zapsané v bázi C.

$${}_C[\mathrm{id}]_B = \begin{pmatrix} | & | & | \\ [\mathbf{b}_1]_C & [\mathbf{b}_2]_C & [\mathbf{b}_3]_C \\ | & | & | \end{pmatrix}$$

Složením zobrazení dokonce zvládneme spočítat matici zobrazení g, která pracuje rovnou v bázi B:

$${}_B[g]_B = {}_B[\mathrm{id}]_C \cdot {}_C[g]_C \cdot {}_C[\mathrm{id}]_B$$

Úkol 1 – Matice přechodu [7b]:

Sestavte obě matice přechodu, z báze B do C (popsaných výše) a zpět. Poté pomocí nich spočítejte ${}_B[g]_B$. Popište, jak by ji šlo přímo odvodit interpretací zobrazení g, podobně jako jsme odvodili ${}_C[g]_C$.

Maticové prostory

Hezké vektorové prostory vychází z matic. Pro $\mathbf{A} \in \mathbb{R}^{m \times n}$ definujeme *sloupcový prostor* $\mathcal{S}(\mathbf{A})$ jako lineární obal sloupců matice, tedy $\operatorname{span}\{(\mathbf{A})_{*1}, \ldots, (\mathbf{A})_{*n}\}$. Pokud maticí $\mathbf{A}$ popíšeme zobrazení, pak sloupcový prostor odpovídá oboru hodnot. Proč tomu tak je? Obor hodnot je množina výrazů $\mathbf{Ax}$ pro všechny $\mathbf{x} \in \mathbb{R}^n$. Složky vektoru $\mathbf{x}$ se při násobení s maticí chovají jako koeficienty, kterými se násobí sloupce $\mathbf{A}$, vytváří se tedy jejich lineární kombinace.

Podobným způsobem zavedeme *řádkový prostor* $\mathcal{R}(\mathbf{A})$ jako lineární obal řádků, tedy $\operatorname{span}\{(\mathbf{A})_{1*}, \ldots, (\mathbf{A})_{m*}\}$. Jeho význam nahlédneme zanedlouho.

Posledním prostorem je množina všech $\mathbf{x} \in \mathbb{R}^n$, pro která platí $\mathbf{Ax} = \mathbf{0}$. Tento prostor je nazývá *jádro matice* a značí se $\ker(\mathbf{A})$. Zde již nemusí být na první pohled zřejmé, že se jedná o vektorový prostor. Jedná se o podmnožinu prostoru $\mathbb{R}^n$, což zajistí většinu vlastností, které od prostoru požadujeme. Ještě je potřeba ukázat, že tato podmnožina je neprázdná a že je uzavřená na součet a násobek skalárem. Přinejmenším platí $\mathbf{0} \in \ker(\mathbf{A})$, protože součin $\mathbf{A0}$ určitě bude roven nule. Pokud $\mathbf{x}, \mathbf{y} \in \ker(\mathbf{A})$, poté jejich pro jejich součet platí $\mathbf{A}(\mathbf{x} + \mathbf{y}) = \mathbf{Ax} + \mathbf{Ay} = \mathbf{0} + \mathbf{0} = \mathbf{0}$, ten tedy v jádru leží. Pro násobek skalárem obdobně.

Jádro má taktéž význam ve spojitosti s lineárními zobrazeními. Jádro obsahuje jen a pouze nulový vektor právě tehdy, když je zobrazení *prosté*, tedy když se žádné dva vektory nezobrazí na stejný obraz. Nahlédněme odůvodnění alespoň implikace zprava doleva: Pro každé lineární zobrazení platí $f(\mathbf{0}) = \mathbf{0}$ a prosté zobrazení již žádný další prvek na $\mathbf{0}$ zobrazit nemůže.

Bez důkazu uvedeme následující větu ($\dim V$ značí dimenzi vektorového prostoru V):

$$\dim \mathcal{S}(\mathbf{A}) = \dim \mathcal{R}(\mathbf{A}) = n - \dim \ker(\mathbf{A})$$

Co nám věta říká? Uvažme výraz $\mathbf{Ax}$, kde $\mathbf{x} \in \mathbb{R}^n$. Pokusme se sestavit nějakou bázi prostoru $\mathbb{R}^n$. Za část bazických vektorů, konkrétně za $k = \dim \ker(\mathbf{A})$ z nich, můžeme zvolit bazické vektory jádra. Ještě potřebujeme dalších $n - k$, dle věty výše je tento počet roven $r = \dim \mathcal{R}(\mathbf{A})$. Bázi tedy doplníme bazickými vektory řádkového prostoru. Poznamenejme, že takové doplnění neporuší lineární nezávislost a tudíž jej lze provést, odůvodnění uvidíme ve čtvrtém dílu.

V důsledku lze každý vektor $\mathbf{x} \in \mathbb{R}^n$ jednoznačně rozložit jako $\mathbf{x} = \mathbf{x}_R + \mathbf{x}_K$, kde $\mathbf{x}_R \in \mathcal{R}(\mathbf{A})$ a $\mathbf{x}_K \in \ker(\mathbf{A})$.

Navíc, díky shodné dimenzi řádkového a sloupcového prostoru existuje mezi vektory těchto prostorů párování neboli *bijekce.* V rovnici $\mathbf{Ax}_R = \mathbf{b}$ tedy pro každé $\mathbf{x}_R$ existuje jediné $\mathbf{b}$ a naopak.

Úkol 2 – **Soustavy pomocí maticových prostorů** [5b]:

Pro danou matici $\mathbf{A} \in \mathbb{R}^{m \times n}$ známe $\mathcal{S}(\mathbf{A}), \mathcal{R}(\mathbf{A}), \ker(\mathbf{A})$. Všechny máme reprezentované pomocí jejich bází:

- Báze $\mathcal{S}(\mathbf{A})$ sestává z r vektorů $\mathbf{s}_1, \ldots, \mathbf{s}_r$,
- báze $\mathcal{R}(\mathbf{A})$ sestává z r vektorů $\mathbf{r}_1, \ldots, \mathbf{r}_r$,
- báze $\ker(\mathbf{A})$ sestává z k vektorů $\mathbf{k}_1, \ldots, \mathbf{k}_k$.

Nově dostaneme zadané $\mathbf{b}$.

1) Rozhodněte, kolik řešení má soustava $\mathbf{Ax} = \mathbf{b}$.
2) Znáte-li jedno řešení $\mathbf{x}_0$, vyjádřete množinu všech řešení.

Porovnejte časovou složitost vašeho přístupu s přímočarým hledáním všech řešení pomocí Gaussovy eliminace. Ve kterých případech je váš přístup výhodnější?

Prostory posloupností

Na závěr se podívejme na posloupnosti zadané rekurentním vztahem. V jedné z úloh minulého dílu jsme zjistili, jak rychle spočítat člen Fibonacciho posloupnosti. Pomocí vektorových prostorů dokážeme pro podobné posloupnosti odvodit explicitní vzorec.

Uvažme posloupnost $(s_0, s_1, s_2, \ldots)$ zadanou rekurentním vztahem $s_{n+2} = 2s_{n+1} + 3s_n$. Takových posloupností je více, označme jejich množinu S. Když bychom určili první členy s_0, s_1, získali bychom konkrétní posloupnost. Množina S tvoří vektorový prostor, sčítáme a násobíme po členech. Podobně jako dříve, použitím standardního sčítání a násobení splníme většinu axiomů. Nulová posloupnost do prostoru patří, uzavřenost na operace také platí.

Konkrétní posloupnost je určena dvěma prvními členy, prostor S má tedy dimenzi 2. Pokusme se najít jeho bázi. Rekurentní posloupnosti většinou rostou exponenciálně, zkusme tedy zjistit, jestli není exponenciální posloupnost α^n součástí prostoru S pro vhodné α. Pro tuto posloupnost platí $s_{n+1} = \alpha s_n$, což dosadíme do rekurentního vztahu:

$$\begin{aligned} s_{n+2} &= 2s_{n+1} + 3s_n \\ \alpha^2 s_n &= 2\alpha s_n + 3s_n \\ \alpha^2 - 2\alpha - 3 &= 0 \\ (\alpha - 3)(\alpha + 1) &= 0 \end{aligned}$$

Našli jsme dva bazické vektory: $\mathbf{b}_1 = 3^n, \mathbf{b}_2 = (-1)^n$. Každá posloupnost prostoru S jde vyjádřit jako jejich lineární kombinace.

Úkol 3 – **Rekurence** [3b]:

Najděte explicitní vzorec pro člen s_n posloupnosti zadané rekurentním vztahem výše a prvními členy $s_0 = 0, s_1 = 1$, tedy $0, 1, 2, 7, 20, 61, 182, 547, \ldots$

David Klement

Recepty z programátorské kuchařky: Intervalové stromy

Představme si, že máme posloupnost celých čísel

$$p_0, p_1, \ldots p_{N-1},$$

se kterou budeme průběžně provádět tyto dvě operace:

1. změna jednoho čísla v posloupnosti
2. zjištění součtu čísel na nějakém intervalu $[a, b]$, tedy $p_a + p_{a+1} + \ldots + p_b$

Nejdříve se ale zkusíme zamyslet, jak bychom úlohu řešili, kdybychom měli jen druhou operaci, tj. dotazy na součty na konkrétních intervalech. K řešení využijeme pole *prefixových součtů*.

Pole prefixových součtů je pole délky $N + 1$, ve kterém na indexu i leží součet prvků posloupnosti od indexu 0 až do indexu $i - 1$. Tedy

$$pref[i] = p[0] + \ldots + p[i-1], \quad pref[0] = 0$$

Není těžké si rozmyslet, že toto pole dokážeme jednoduše spočítat v čase $\mathcal{O}(N)$.

Nyní, když už známe všechny prefixové součty posloupnosti, umíme snadno spočítat součet na libovolném intervalu $[a, b]$:

$$s[a, b] = pref[b + 1] - pref[a]$$

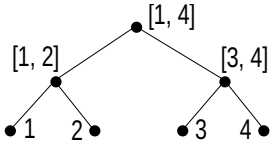
Každý dotaz dokážeme zodpovědět v konstantním čase. Celý algoritmus má tedy složitost $\mathcal{O}(N + D)$, kde N je délka posloupnosti a D je počet dotazů.

Když si do úlohy přidáme i operaci č. 1 (změna čísla v posloupnosti), tak se nám pokazí časová složitost. S prefixovými součty stále dokážeme dotaz č. 2 provádět v konstantním čase, ale při operaci č. 1 se nám může stát, že musíme změnit až všechny prefixové součty, takže složitost této operace je $\mathcal{O}(N)$ a celková složitost pro Z změn a D dotazů je v nejhorším případě $\mathcal{O}(NZ + D)$.

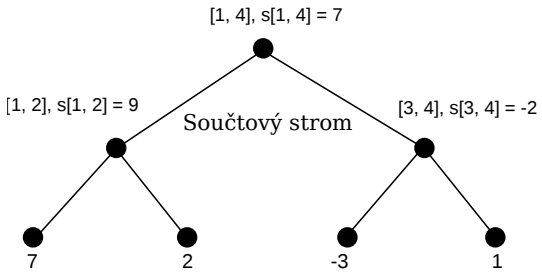
S touto složitostí se samozřejmě nespokojíme a budeme se snažit, abychom výsledné intervaly uměli co nejrychleji skládat z předpočítaných hodnot a abychom při změně posloupnosti museli změnit co nejméně hodnot. K tomu se nám bude hodit datová struktura jménem intervalový strom.

Zavedení intervalového stromu

Intervalový strom je dokonale vyvážený binární strom, jehož každý list představuje nějaký interval a všechny ostatní vrcholy reprezentují interval, který vznikne složením intervalů jejich synů. Zároveň intervaly vrcholů jedné hladiny na sebe navazují (vždy směrem zleva doprava). Z toho vyplývá, že složením intervalů z vrcholů jedné hladiny dostaneme interval, který si pamatujeme v kořeni.



Intervalových stromů existuje více druhů. Obvykle je rozlišujeme podle toho, jaké informace si v nich pamatujeme. Například ve stromu pro součty si každý vrchol pamatuje součet na svém intervalu, ve stromu pro maxima si pamatuje maximum na intervalu, apod. Můžeme ale klidně mít strom, který si pamatuje, jestli celý jeho interval obsahuje jen jednu hodnotu, a pokud ano, tak jakou.



My se teď zaměříme na intervalový strom pro součty a pomocí něj vyřešíme úvodní úlohu.

Na začátku budeme chtít, aby v listech intervalového stromu byly hodnoty původní posloupnosti, přičemž první a poslední list stromu necháme volné, později uvidíme, proč. Zároveň ale chceme, aby tento strom byl dokonale vyvážený.

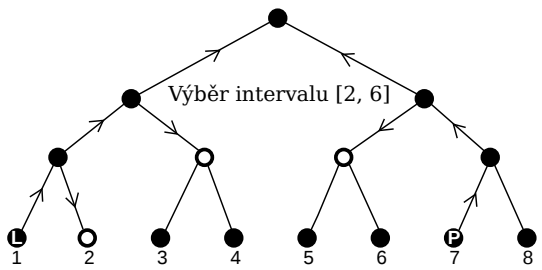
Posloupnost tedy prodloužíme tak, aby její velikost byla mocnina dvojky minus dva (na její konec přidáme nějaké prvky). Všimněte si, že tím jsme strom nezvětšili více než dvakrát a že nám nezáleží na tom, jaké prvky jsme do stromu přidali, protože s nimi nikdy nebudeme pracovat. Nyní k jednotlivým operacím.

Změnu čísla v posloupnosti uděláme jednoduše. Zjistíme, o kolik se hodnota prvku posloupnosti změní, najdeme odpovídající list a k tomuto listu a ke všem jeho předkům přičteme daný rozdíl. Tím jsme upravili všechny intervaly, do kterých tento prvek patří.

Nyní se podívejme, jak ze stromu zjistíme součet na nějakém intervalu $[a, b]$. Jinými slovy: potřebujeme ze stromu vybrat takové vrcholy, aby sjednocení jejich intervalů byl náš dotazovaný interval, a zároveň chceme, aby těchto vrcholů bylo co nejméně.

Součet intervalu $[a, b]$ zjistíme tak, že si ve stromu najdeme listy reprezentující pozice $a-1$ a $b+1$ posloupnosti a jejich nejbližšího společného předka p. Nyní budeme postupovat z listu od $a-1$ až do p a vždy, když do nějakého vrcholu přijdeme z levého syna, tak do výsledku přidáme interval pravého syna. Stejně tak postupujeme od $b+1$ k p, a pokud do vrcholu přijdeme z pravého syna, tak přidáme jeho levého syna.

Všimněte si, že při takovémto průchodu složíme celý interval. Vše je vidět na nasledujícím obrázku:



Způsobů, jak pracovat s intervalovým stromem a zjišťovat z něj informace, je více. Toto byl jeden z nich.

Změna prvku posloupnosti má časovou složitost $\mathcal{O}(\log N)$, protože jsme na každé hladině změnili pouze jeden interval a strom má $\mathcal{O}(\log N)$ hladin. Zjištění součtu na intervalu má také složitost $\mathcal{O}(\log N)$, jelikož jsme do výsledku přidali maximálně $2 \log N$ intervalů: nejvýše $\log N$ při cestě z listu $a-1$ a $\log N$ při cestě z $b+1$.

Implementace intervalového stromu

Při implementaci intervalového stromu využijeme jeho dokonalé vyváženosti a budeme jej implementovat v poli (stejně jako se do pole ukládá halda). Kořen stromu bude v poli na indexu 1, vrcholy z druhé hladiny budou mít postupně indexy 2 a 3, ..., až listy budou mít indexy $N, \ldots, 2N-1$. V této reprezentaci platí pro vrchol s indexem i následující pravidla:

1. $2i$ a $2i+1$ jsou jeho synové.
2. $\lfloor i/2 \rfloor$ je jeho předek (pro $i > 1$).
3. Pokud je i sudé, tak je vrchol levým synem, jinak pravým.
4. Pro sudé i je $i+1$ pravý bratr, pro liché i je $i-1$ levý bratr.

Nyní víme vše potřebné, tak se podívejme na samotnou implementaci v jazyce C:

```
int N = 100;    // velikost posloupnosti
int posl[100];  // posloupnost
int *strom;     // intervalový strom

// Deklarace funkcí
void inic(int N);
void pricti(int index, int hodnota);
int soucet(int A, int B);

// Inicializace intervalového stromu
// Pozor: prvky posloupnosti indexujeme 1, ..., N
void inic(int N) {
  // Najdeme nejbližší vyšší mocninu dvojky
  int listy = 1;
  while (listy<N+2) listy = listy*2;
  // Pro strom potřebujeme 2*(počet listů) vrcholů
  // (nepoužíváme strom[0])
  strom = (int*)malloc(sizeof(int)*2*listy);
  N = listy;
  for (int i=0; i<2*listy; i++) strom[i] = 0;
  // Na příslušná místa přičteme hodnoty posloupnosti
  for (int i=0; i<N; i++)
    pricti(i, posl[i]);
}

// Přičtení hodnoty na dané místo posloupnosti
void pricti(int index, int hodnota) {
  int k = N + index;
  while(k>0) {
    strom[k] = strom[k] + hodnota;
    k = k/2;
  }
}

// Zjištění součtu na intervalu
int soucet(int A, int B) {
  int souc = 0;
  int a = N + A - 1;
  int b = N + B + 1;
  while (a!=b) {
    // Pokud je a levý syn, tak přičti pravého bratra
    if (a%2==0) souc = souc + strom[a+1];
    // Pokud je b pravý syn, tak přičti levého bratra
    if (b%2==1) souc = souc + strom[b-1];
    // Přesun na otce
    a = a/2; b = b/2;
  }
```

```
  // Navíc jsme přičetli syny společného předka.
  souc = souc - strom[2*a] - strom[2*a+1];
  return souc;
}
```

V této implementaci jsme strom upravovali zdola směrem nahoru. Existuje ještě rekurzivní implementace, v níž se strom upravuje od kořene směrem dolů, ale tu si zde ukazovat nebudeme.

Cvičení

- Naprogramujte rekurzivní implementaci operací (strom se prochází shora dolů).
- Jak by vypadala implementace intervalového stromu pro maxima?

Použití intervalového stromu

Intervalový strom je silný nástroj, kterým se dá vyřešit spousta úloh. Ale než ho začnete používat, tak si vždy rozmyslete, zda úlohu nelze řešit elegantněji bez intervalového stromu. Ne všechny druhy intervalových stromů se dobře implementují.

Intervalový strom obvykle použijeme, pokud potřebujme průběžně zjišťovat informace o intervalech a zároveň je i měnit. Pokud používáme jen jednu z těchto operací (a tu druhou jen zřídka), existuje často lepší řešení než intervalový strom – viz úvodní příklad.

Fenwickův strom

Fenwickův strom, někdy také nazývaný *finský strom,* je způsob, jak strom reprezentovat v poli. Jeho používání je podobné jako používání intervalového stromu pro součty. Rozdíl je jen v implementaci daných funkcí. My si Fenwickův strom opět ukážeme na úvodním příkladu. Zase tedy budeme potřebovat funkci pro změnu hodnoty v posloupnosti a funkci pro zjištění součtu na intervalu. (Ve skutečnosti zjistíme dva prefixové součty a z nich pak spočítáme výsledný interval.)

Fenwickův strom je poněkud magická datová struktura. Abychom si tuto magii mohli užít, zvolíme trochu netradiční způsob vysvětlování a nejdříve si ukážeme, jak se Fenwickův strom implementuje, a teprve pak si vysvětlíme, jak to všechno funguje.

Fenwickův strom bude pole velikosti $N+1$, kde index 0 nebudeme používat. Používat budeme pouze prvky $1, \ldots, N$, které všechny na začátku nastavíme na 0. Pokud v posloupnosti změníme hodnotu, stejně jako u intervalového stromu, ve Fenwickově stromu na některá místa přičteme rozdíl oproti předchozí hodnotě.

```
void pricti(unsigned int index, int rozdil) {
  while (index<=N) {
    strom[index] += rozdil;
    index = index + (index & -index);
    // "&" značí bitový AND
  }
}
```

A zde je funkce pro zjištění prefixového součtu:

```
int pref_soucet(unsigned int index) {
  int soucet = 0;
  while (index>0) {
    soucet = soucet + strom[index];
    index = index & (index-1);
  }
  return soucet;
}
```

Toť celá implementace. No, nevypadá na první pohled magicky? Pokud chcete vědět, jak tohle celé funguje, tak čtěte dál.

Ve Fenwickově stromu je na indexu 1 uložen první prvek, na indexu 2 součet prvního a druhého, na indexu 3 třetí prvek, na indexu 4 součet prvních čtyř, ... na indexu N je uložen součet posledních 2^K hodnot, kde K je pozice prvního jedničkového bitu v binárním zápisu čísla N. Ve stromu máme tedy uloženou takovou pravidelnou strukturu intervalů.

Nyní se podíváme, co dělají naše magické funkce na posouvání ve stromu a pak najednou bude všechno jasné. Ve výrazu `index & (index-1)` z funkce `pref_soucet()` se neděje nic jiného, než že se vynuluje nejpravější jedničkový bit v indexu. Tím se dostaneme na první interval, který jsme ještě nepřičetli. V momentě, kdy se dostaneme na index 0, tak už máme dotazovaný interval kompletní a výpočet můžeme ukončit.

Výraz `index + (index & -index)` dělá to, že se v pomyslném stromu intervalů posune o úroveň výš.[2] Pokud jsme tedy v intervalu o velikosti 2, tak se dostaneme do intervalu velikosti 4, který daný interval obsahuje (tento interval je jednoznačný). Samotný výpočet dělá to, že v čísle `index` vezme nejpravější jedničku a znova ji přičte.

Fenwickův strom se používá hlavně kvůli jednoduchosti jeho naprogramování a také kvůli efektivitě samotného výpočtu a nevelké náročnosti na paměť. Při jeho implementaci doporučujeme dávat si pozor na správnost bitových funkcí.

Cvičení

- Rozmyslete si, že oba magické výpočty opravdu dělají to, co mají, a také, proč vše vlastně funguje.

Karel Tesař

[2] Magická operace `index & -index` funguje jen v případě, že se jako reprezentace záporného čísla používá tzv. dvojkový doplněk: `-k == ~k + 1`, neboli všechny bity čísla se znegují a pak se přičte jednička.

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.
Realizace projektu byla podpořena Ministerstvem školství, mládeže a tělovýchovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Organizátoři a kontakty: *https://ksp.mff.cuni.cz/kontakty/*