Korespondenční Seminář z Programování

38. ročník **KSP** **Březen 2026**

Milí řešitelé, řešitelky a řešitelčata!

Dostává se k vám čtvrté číslo hlavní kategorie 38. ročníku KSP.

Letos se můžete těšit v každé z pěti sérií hlavní kategorie na **4 normální úlohy**, z toho alespoň jednu praktickou opendatovou. Dále 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**, jehož díly mohou vycházet samostatně.

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.

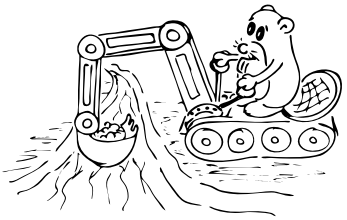
Termín série: **12. dubna 2026 ve 32:00 (tedy další ráno v 8:00)**

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
- Experimentální (neobvyklá) úloha

Čtvrtá série třicátého osmého ročníku KSP

38-4-1 Vesmírné trubky — 11 bodů



Za sedmero černými a sedmero červími děrami se nachází nadpozemská hroší civilizace. Tato civilizace je vskutku pokročilá, mnohem pokročilejší než ta naše. Teď se ale potýká s jedním velkým problémem.

V tom bájném vesmíru, kde se skýtá námi ctěná hroší civilizace, stojí též spousta magických železných trubek. Ty jsou pro hrochy posvátné a musí být ochráněny za každou cenu – hlavně nesmí nedejbože spadnout! Proto se hroši rozhodli kolem trubek postavit ochrannou zeď, a to tak, aby byla co nejmenší (to nastane tehdy, když má nejmenší obvod). Jelikož jsou ale trubky magické, mohou se znenadání objevovat nové trubky na náhodných místech. Pak musí hroši okamžitě rozšířit zeď tak, aby obklopila i novou trubku. I přes velkou pokročilost si hroší civilizace zatím neví s tímto problémem rady. Pomůžete jim?

Slibujeme, že se neobjeví dvě trubky na stejném místě. Jelikož zeď má nejmenší možný obvod, lze ji popsat pomocí pozic trubek, které leží (přesněji řečeno stojí) na její hranici. Tyto trubky nazýváme *věže zdi*. Pokud leží více trubek na stejné přímce, tak se jako věže zdi berou jenom ty krajní, které jednoznačně určují danou přímku. To znamená, že když se objeví trubka na hranici zdi, je již bezpečně uvnitř, a zeď není potřeba rozšiřovat.

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

Na vstupu dostanete nejprve N trubek, které jsou na začátku obklopeny zdí (konvexním obalem), a poté K dalších trubek, které se postupně objevují. Po každé nové trubce vypište změnu věží zdi, v případě, že nastala nějaká změna. Pokud se krajní věže zdi nezměnily (neboli do konvexního obalu nic nepřibylo a nic z něj neubylo), nic nevypisujte.

Formát vstupu: Na prvním řádku jsou čísla N a K, značící počet trubek na začátku a počet nových trubek, které se postupně objeví. Následuje $N + K$ řádků, přičemž každý obsahuje dvě čísla – souřadnice x_i a y_i i-té trubky.

Formát výstupu: Pokud po přidání trubky není potřeba zeď změnit, nic nevypisujte. Jinak vypište na jednom řádku souřadnice všech trubek, které byly buď odebrány nebo přidány jako věže zdi, v libovolném pořadí. Každou trubku vypište jako dvojici x a y oddělenou mezerou.

Ukázkový vstup:

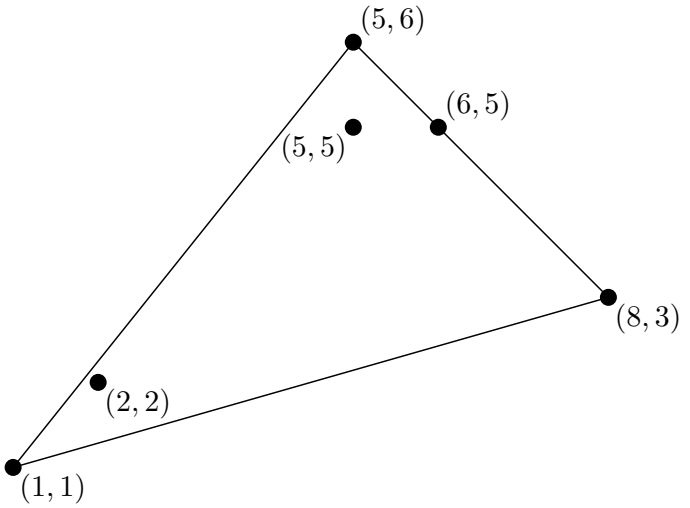
```
3 4
5 5
5 6
6 5
1 1
8 3
2 2
10 10
```

Ukázkový výstup:

```
1 1 5 5
6 5 8 3
10 10
```

Po prvních třech trubkách už zeď určují $(5,5),(5,6),(6,5)$. Po přidání trubky $(1,1)$ se zeď změní na $(1,1),(5,6),(6,5)$, tedy se přidala trubka $(1,1)$ a odebrala se trubka $(5,5)$. Pak po přidání trubky $(8,3)$ leží $(5,6),(6,5)$ a $(8,3)$ na stejné přímce na hranici zdi, takže se jako věže zdi berou jen ty krajní, tedy se zeď určí body $(1,1),(5,6)$ a $(8,3)$. Přidání trubky $(2,2)$ hranici vůbec nezmění, tedy nebudeme nic vypisovat. Dalším přidání trubky $(10,10)$ se zeď pouze zvětší (jedna přímka hranice se změnila na dvě přímky), takže vypíšeme pouze přidanou trubku $(10,10)$.

Obrázek zobrazuje situaci před přidáním bodu $(10,10)$.



38-4-2 Kdo se v tom vyzná — 8 bodů

Tato úloha je o vyznání. Hroší společnost se vyvíjí už nějaký ten čtvrtek, což je na jednu stranu super, páč jsou díky evoluci hroši takoví nadlidé. Na druhou stranu ale hroši čelí už nějakou dobu své existenční otázce. Každý se s tím popasovává různě – jeden vyznává všeplodné vejce, druhý hroch se s tím vším vypořádává díky všudypřítomnému pití, je tu skupina hrochů vyznávající kameny na cestě, jakožto tu jedinou věc, na které na světě opravdu záleží.

Nebudeme zde zkoumat, které vyznání smysl má a které ne. Jsme přeci jenom pouzí lidé a hrochů si vážíme takových, jací jsou. Co by nás ale zajímalo, je sledovat, jakou víru hroši průběžně vyznávají. Na začátku si každý hroch věří ve svou unikátní vnitřní víru. Postupně se ale víry rozpadají a jejich věřící se přelévají.

Nám přichází průběžně informace o tom, která víra se přelila do které. Budou nás v průběhu zajímat následující dotazy:

- jestli dva daní hroši mají stejné vyznání,
- kolik hrochů vyznává aktuální víru daného hrocha,
- kolik věr hroši mají.

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 čísla H a I – počet hrochů (a tedy i počáteční počet věr) a počet instrukcí, přičemž hrochy číslujeme celými čísly mezi 0 a $H-1$. Na každém z následujících I řádků bude následovat instrukce či dotaz:

- `J` A B – víra hrocha A se sloučí s vírou hrocha B (nemusí nutně platit $A \neq B$)
- `S` A B – dotaz, zda hroch A věří ve stejnou víru jako hroch B
- `V` A – dotaz na počet hrochů, kteří věří ve víru hrocha A
- `P` – dotaz na počet unikátních věr

Formát výstupu: Pro dotazy typu `S` vypište `ANO` či `NE`, a pro `V` a `P` vypište na jeden řádek číselnou odpověď.

Ukázkový vstup:

```
10 10
P
S 0 9
V 0
V 9
V 5
S 1 0
J 1 0
V 0
S 1 0
P
```

Ukázkový výstup:

```
10
NE
1
1
1
NE
2
ANO
9
```

38-4-3 Lidská pyramida — 13 bodů

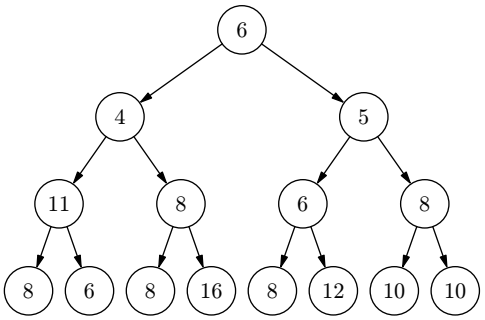
Byl krásný školní den. Zvonek na chodbě začal zvonit a Adam moc dobře věděl, co to pro třídu 7.A znamenalo. Pan učitel tělocviku mu to prozradil. Už se na to těšil poslední hodinu. Nadšeně vběhl do kabinetu, vzal pytel sýrových koulí, který si tam přesně za tímto účelem připravil, a upaloval do tělocvičny.

Zmeškal většinu vysvětlování, ale to mu vůbec nevadilo. Akorát stihnul otrávené výrazy studentů 7.A, jak se rozcházeli z úvodního pozoru. Pytel od křupek hlasitě puknul – jeden ze studentů, Petr, se na něj otočil. Při pohledu na rozzářeného Adama, křupajícího sýrové koule, natěšeného na nadcházející události, se musel Petr pousmát.

Show začíná, první řada už se rýsuje – ta jde vždycky hned. Při druhé řadě už to začíná být zajímavé. Chlapci v té první dotčeně pokřikují. Sýrové koule nikdy Adamovi nechutnaly lépe. Lidská pyramida stoupá, už se rýsuje třetí patro. Při konstrukci čtvrtého už děvčata šplhají po hlavách jako po stěně. Křupky báječně křupají. Pavel ve 2. patře naštvaně nadává – zrovna dostal nohou do ucha. Jaká báječná věc to pana tělocvikáře napadla. Teď už zbývá jen, aby Evženie usedla v pátém – nejvyšším – patře jako na trůně a jako královna svrchovaně vládla své třídě. Jak báječná zábava je pozorovat její výstup za mocí. Ve chvíli, kdy usedá, už pomalu přes otrávené pokřikování pomalu ani nejsou slyšet sýrové koule.

No a teď si představte to nadšení, když se studenti dozvěděli, že se na Roberta nedostalo a že se má prohodit s Romanem v prvním patře.

Pan tělocvikář by ocenil vaši radu, kam studenta umístit, aby skončila pyramida co nejmenší. Tato konkrétní lidská pyramida vypadá jako *úplný binární strom* – neboli strom, kde jsou všechny listy pouze na nejhlubší *hladině*, a všechny ostatní vrcholy mají právě dva syny.



Každý student má svou výšku a zároveň stojí nějak vysoko. Studenti v první řadě – v té nejhlubší hladině stromu – stojí ve výšce 0. Každý student v druhé řadě stojí na dvou studentech v první řadě, konkrétně v průměru výšek studentů pod sebou. Student v každé další řadě stojí v průměru součtů výšek hlav studentů pod ním. Intuitivně si můžete představit, že si studenti pod ním na hlavy položí prkno a on stojí v jeho prostředku.

Například na obrázku výše stojí student s výškou 11 vlevo na dvou studentech s výškami 8 a 6, tedy stojí ve výšce jejich průměru, což je $\frac{8+6}{2} = 7$. Jeho hlava se tedy nachází ve výšce $7 + 11 = 18$. Jeho soused ve vrstvě s výškou 8 stojí na 8 a 16, tedy ve výšce $\frac{8+16}{2} = 12$ a jeho hlava končí ve výšce $12 + 8 = 20$. Student, který stojí na nich, stojí ve výšce $\frac{18+20}{2} = 19$ a jeho hlava končí ve výšce $19 + 4 = 23$.

Nám budou chodit studenti a my čelíme problému ve formátu: „Přišel student s výškou V. Se studentem *na které pozici* ho máme vyměnit, aby poté byla pyramida co nejmenší?“

Vaším úkolem je po každém příchozím studentovi napsat, kam do pyramidy bude umístěn (tedy pozici studenta, kterého vymění). Vyměněný student si jde lehnout a po zbytek vyučování není k nalezení. Naopak příchozí student ho vymění a účastní se pyramidy. Pokud existuje více pozic, kde je po výměně studenta pyramida co nejnižší, zvolte tu na nejvyšší hladině. Pokud je i těch více, tak tu z nich nejvíce napravo.

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 je číslo H – výška pyramidy. Poté následuje H řádků, na i-tém z nich je 2^i čísel (řádky indexujeme od nuly) – výšky studentů v dané hladině pyramidy. Následuje číslo S – počet studentů, kteří se přijdou zapojit později. Nakonec následuje S čísel na jednom řádku s výškami příchozích studentů (v pořadí, jak přicházejí).

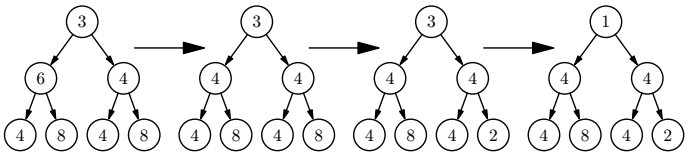
Formát výstupu: Vypište pro každého z S příchozích studentů na jeden řádek dvě čísla oddělená mezerou – hladinu, kam ho umístíte do pyramidy a index místa v dané hladině stromu – pořadí zleva. (Jako správní informatici indexujeme od nuly.)

Ukázkový vstup:

```
3
3
6 4
4 8 4 8
3
4 2 1
```

Ukázkový výstup:

```
1 0
2 3
0 0
```



38-4-4 Alokátor — 13 bodů

Kevinův operační systém je již vetchý. Jeho procesy zachází s pamětí jako bandité a šerif je v nedohlednu. Kevin se rozhodl s nimi zatočit jako s lasem a učinit paměť opět bezpečným místem.

Proto si Kevin (jako velký kluk) implementuje alokátor paměti sám. ChatGPT Kevinovi poradilo, že paměť je vlastně souvislý interval indexů od 0 do $P - 1$, kde P je velikost jeho paměti. Paměť se v průběhu alokací a dealokací fragmentuje (rozděluje) na *bloky*. Bloky můžou být obsazené či volné. Celá paměť začíná při spuštění jako jeden volný blok. Alokátor musí umět následující 3 operace:

- `alloc`(d) – alokace volného bloku o velikosti d

  Buď se obsadí nějaký volný blok o velikosti d, nebo se nějaký větší volný blok roztrhne na obsazený blok velikosti d a volný se zbytkem. Alokátor ale musí použít nejmenší dostačující blok.
- `free`(i) – uvolnění i-tého bloku

  Zde i je *pořadové číslo* bloku k uvolnění. To znamená, že chceme uvolnit i-tý blok v paměti, podle *aktuální* fragmentace paměti (číslujeme od nuly). Máte zaručeno, že i-tý blok je obsazený, tedy že uživatel nebude uvolňovat volný blok.

  Uvolňovaný blok se sloučí s volným sousedícím blokem, případně s oběma volnými sousedícími bloky, pokud je to možné. Nemůže se tedy stát, že jsou vedle sebe v paměti dva volné bloky.

  Tedy pokud máme například paměť o velikosti 10, rozdělenou na 3 obsazené bloky o velikostech 3, 5, 2, tak operace `free(1)` uvolní blok o velikosti 5. Následně by operace `free(1)` neměla smysl, jelikož je daný blok již volný. Další operace `free(2)` by uvolnila blok o velikosti 2, který by se sloučil s předchozím volným blokem a utvořil jeden blok velikosti 7. Následně by zase operace `free(2)` neměla smysl, jelikož již máme pouze 2 bloky.
- `block`(j) – zjištění indexu bloku, kde leží pozice j v paměti

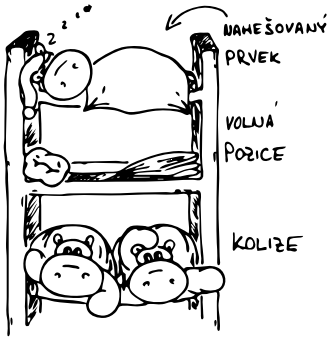
  Zajímá nás index (pořadové číslo) bloku v aktuální fragmentaci paměti, kam patří pozice j. Mějme opět fragmentaci 3, 5, 2. Dotaz `block(4)` by vrátil 1, jelikož se pozice 4 nachází v bloku o velikosti 5 s pořadovým číslem 1 (číslujeme od nuly).

Pomozte Kevinovi vymyslet, jak implementovat požadované operace. Při analýze složitosti můžete použít P – velikost paměti, K – aktuální počet bloků, či parametr d pro `alloc`.

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

38-4-S Univerzální hešování — 15 bodů

V minulém dílu jsme si hráli s hešovacími funkcemi a zjistili jsme, že je docela snadné vymyslet vstupní data, která způsobí spoustu kolizí. Intriky škodolibých uživatelů tedy mohou snadno zařídit, aby naše skvělá hešovací tabulka, z jejíž průměrně konstantní složitosti se radujeme, byla ve skutečnosti ukrutně pomalá.



Netřeba zoufat, opět nás zachrání randomizované algoritmy. Místo pevné hešovací funkce si ji totiž při každém spuštění programu zvolíme náhodně. Pak nám mohou nepřátelé předkládat jakkoliv zavilé vstupy, ale jelikož nevědí, jakou hešovací funkci jsme si vybrali, bude velmi nepravděpodobné, že vstupy budou zrovna pro tuto funkci špatné. A konečně budeme něco umět doopravdy dokázat o průměrné časové složitosti hešovacích tabulek.

Vybírat hešovací funkci úplně náhodně však nefunguje: museli bychom pro každý prvek univerza zvolit, do které přihrádky ho zahešujeme. Na to bychom potřebovali pole indexované univerzem, ovšem kdybychom si takové mohli dovolit, především bychom nepotřebovali žádné hešování. Proto budeme vybírat z nějaké omezené množiny funkcí, které umíme snadno popsat pomocí parametrů.

To vede ke konceptu *univerzálních systémů hešovacích funkcí.* Přečtěte si o nich v Průvodci labyrintem algoritmů v oddílu 11.5. Můžete vynechat části „Konstrukce ze skalárního součinu“ (ale pokud máte rádi lineární algebru, třeba z našeho seriálu, mohla by se vám tato konstrukce líbit) a „Vzorkování a k-nezávislost“. Pokud vám budou chybět znalosti dělitelnosti, kongruencí, případně konečných těles, nakoukněte do kuchařky o teorii čísel.[1]

Tabelační hešování

Prozkoumáme ještě jeden druh randomizovaných hešovacích funkcí, kterému se říká *tabelační* (tedy tabulkové) *hešování.*

Je založené na tom, že každý prvek rozdělíme na k částí, každou částí indexujeme jinou tabulku, a hodnoty přečtené z tabulek zkombinujeme funkcí XOR.

Nejprve zvolíme parametry: počet částí k, počet bitů jedné části t a počet bitů čísla přihrádky b. Univerzum je tedy tvořeno všemi kt-bitovými čísly, takže je to množina $\mathcal{U} = \{0, \ldots, 2^{kt} - 1\}$. Přihrádky jsou očíslované od 0 do $2^b - 1$.

Pořídíme si k tabulek T_0 až T_{k-1}. Každá tabulka bude indexovaná t-bitovým číslem a její položky budou b-bitová čísla. Tabulky vyplníme rovnoměrně náhodnými čísly. Chceme-li zahešovat nějaký prvek $x \in \mathcal{U}$, rozdělíme ho na k skupin po b bitech. Každou skupinou bitů zaindexujeme jednu tabulku a příslušné položky tabulek vyxorujeme.

Chceme-li například hešovat 32-bitová čísla do $1024 = 2^{10}$ přihrádek, můžeme zvolit $k = 4$, $t = 8$, $b = 10$. Pořídíme si tabulky T_0 až T_3, každou vyplníme $2^8 = 256$ náhodnými 10-bitovými čísly. Například číslo $x =$ `0x12345678` (zapsáno v 16-kové soustavě) tedy zahešujeme jako $T_3[\texttt{0x12}] \oplus T_2[\texttt{0x34}] \oplus T_1[\texttt{0x56}] \oplus T_0[\texttt{0x78}]$, kde $\oplus$ je operace XOR.

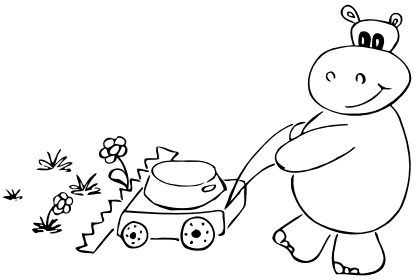
Parametry jde volit i jinak, například $k = 32$, $t = 1$, $b = 10$. Tehdy bychom si pořídili 32 tabulek, každou o dvou položkách. Při výpočtu hešovací funkce bychom ale museli rozdělit prvek na jednotlivé bity a každým indexovat jinou tabulku.

Pojďme si rozmyslet časovou a prostorovou složitost. Tabulky zaberou $k \cdot 2^t$ buněk paměti, v nichž jsou uložena b-bitová čísla. Vyplnit tabulky stojí čas $\mathcal{O}(k \cdot 2^t)$. Zahešovat jeden prvek zvládneme v čase $\mathcal{O}(k)$, pokud na rozklad prvku na části použijeme bitové posuny. V Céčku bychom to mohli zapsat třeba takto:

```
unsigned int hash(unsigned int x)
{
    unsigned int h = 0;
    for (unsigned int i=0; i<k; i++)
        h ^= T[i][(x >> (i*t)) & ((1<<t) - 1)];
    return h;
}
```

Tabelační hešování také zapadá do našeho konceptu univerzálních systémů hešovacích funkcí. Roli parametrů funkce zde hrají obsahy všech tabulek. Jen pozor na to, že na rozdíl od běžných univerzálních systémů neumíme funkci ze systému vybrat v konstantním čase. Vyhodnocení funkce ale už konstantní bude, pokud tedy parametr k zvolíme pevně.

Úkol 1 [4b]: Dokažte, že tabelační hešování je 1-univerzální (nezávisle na počtu tabulek).



Bloomovy filtry

Další kouzelnou aplikací hešování jsou Bloomovy filtry – to je randomizovaná datová struktura, která si pamatuje nějakou množinu prvků univerza. Do množiny můžeme vkládat nové prvky a ptát se, zda daný prvek v množině leží. Struktura zabírá velmi malý prostor, ale platíme za to tím, že není dokonale spolehlivá. Jak uvidíme za chvíli, tu a tam odpoví špatně.

Bloomův filtr si můžeme představit jako hešovací tabulku T s m přihrádkami, z nichž každá obsahuje jediný bit. Ten říká, zda v přihrádce leží aspoň jeden prvek. Dále si pořídíme hešovací funkci h náhodně vybranou z nějakého c-univerzálního systému.

Při vkládání prvku x spočítáme hešovací funkci $h(x)$ a nastavíme příslušný bit $T[h(x)]$ na 1. Při hledání prvku y se podíváme na $T[h(y)]$ a pokud je bit nastaven na 1, odpovíme ANO, jinak NE.

[1] `http://ksp.mff.cuni.cz/viz/kucharky/teorie-cisel`

Rozmyslete si, že odpověď NE je vždy správná. Odpověď ANO může znamenat buďto, že daný prvek y v množině leží, nebo že v ní leží jiný prvek x, který s y zkolidoval, tedy je $h(x) = h(y)$.

Pojďme spočítat pravděpodobnost kolize. Nechť struktura obsahuje m přihrádek a postupně jsme vložili prvky $x_1, \ldots, x_n$. Pravděpodobnost, že hledané y zkoliduje s konkrétním x_i, můžeme odhadnout z definice c-univerzality: $P[h(x_i) = h(y)] \leq c/m$. Jelikož $P[A \vee B] \leq P[A] + P[B]$, můžeme pravděpodobnost toho, že y zkoliduje s alespoň jedním x_i shora omezit výrazem cn/m.

Pokud bychom tedy chtěli vytvořit filtr pro n-prvkovou množinu, který se bude mýlit se zvolenou pravděpodobností p, potřebovali bychom zvolit počet přihrádek m tak, aby bylo $cn/m \leq p$, tedy $m \geq cn/p$.

Ukažme si příklad: mějme množinu $n = 10^6$ prvků z univerza 64-bitových čísel, které hešujeme funkcemi z 1-univerzálního systému. Chceme-li dosáhnout pravděpodobnosti chyby $p = 1/10$, potřebujeme volit $m \geq 10^7$. Filtr tedy spotřebuje 10 Mb = 1.25 MB paměti, zatímco obyčejné pole by zabralo 64 Mb = 8 MB paměti. Kdybychom ovšem chtěli chybu snížit na $p = 10^{-6}$, už by filtr potřeboval obřích 10^{12} Mb paměti.

Vícepásmové Bloomovy filtry

Bloomův filtr můžeme vylepšit jednoduchým trikem: pořídíme si víc filtrů s nezávislými hešovacími funkcemi. Těm budeme říkat *pásma.* Vkládat budeme každý prvek do všech pásem. Dotaz předložíme všem pásmům a ANO odpovíme jen tehdy, když se ta tom všechna pásma shodnou. Odpověď NE tedy bude spolehlivá (aspoň jedno pásmo zamítlo, což bylo spolehlivé). Odpověď ANO bude mylná jen tehdy, pokud se zmýlila všechna pásma současně.

Počet přihrádek v pásmu nastavíme tak, aby pásmo dělalo chyby s pravděpodobností 1/2: potřebujeme $m = 2cn$. Nyní zvolíme počet pásem k. Pravděpodobnost, že všechna pásma se současně zmýlí, bude $(1/2)^k$. Pokud tedy chceme, aby se celý filtr mýlil s pravděpodobnosti nejvýše zadané p, potřebujeme $(1/2)^k \leq p$, tedy $k \geq \log_{1/2} p = \log_2(1/p)$.

Pro předchozí příklad s $n = 10^6$ prvky tedy pro $p = 1/10$ dostaneme $m = 2 \cdot 10^6$ a $k = 4$. Celý filtr tudíž zabere 8 Mb. Pro $p = 10^{-6}$ pak vyjde stejné m a $k = 20$, takže celý filtr zabere 40 Mb, což je stále méně než pole.

Ještě dodejme, že místo pravděpodobnosti chyby pásma 1/2 by šla použít libovolná jiná z intervalu $(0, 1)$. To by ovlivnilo pouze konstanty, a ty nebudeme optimalizovat.

Úkol 2 [5b]: Na vstupu máme 10^9 záznamů velikosti 32 bytů. Chceme zjistit, zda jsou všechny záznamy navzájem různé. Máme ovšem k dispozici jenom 4 GB RAM. Vstup smíme číst vícekrát. Snažte se o co nejlepší průměrnou časovou složitost.

Perfektní hešování: kvadratický prostor

Bloomovými filtry ale hezké aplikace univerzálního hešování nekončí. Ukážeme, jak si pro daná data pořídit *perfektní* hešovací funkci, což je taková, která nemá žádné kolize. (Z minulého dílu seriálu víme, že zvolíme-li pevnou hešovací funkci, je značně nepravděpodobné, že bude pro daná data perfektní. Tady ale data předem známe, takže můžeme funkci volit podle nich.)

Představme si, že navzájem různé prvky $x_1, \ldots, x_n$ hešujeme do m přihrádek hešovací funkcí h vybranou náhodně z c-univerzálního systému. Označme K počet kolidujících párů, tedy dvojic (i, j) takových, že $1 \leq i < j \leq n$ a $h(x_i) = h(x_j)$. K závisí na volbě funkce h, je to tedy náhodná veličina. Pojďme spočítat její střední hodnotu.

Zavedeme indikátory kolizí: to budou náhodné veličiny $K_{i,j}$ nabývající hodnot 0 a 1 tak, že $K_{i,j} = 1$ právě tehdy, když $h(x_i) = h(x_j)$. Střední hodnotu indikátoru lze spočítat snadno: $\mathbf{E}[K_{i,j}] = 0 \cdot P[K_{i,j} = 0] + 1 \cdot P[K_{i,j} = 1]$. První člen vypadne, druhý je roven $P[h(x_i) = h(x_j)]$, což je podle definice c-univerzálního systému nejvýš c/m. Víme tedy, že $\mathbf{E}[K_{i,j}] \leq c/m$.

Teď si všimneme, že počet kolidujících párů K je roven součtu všech indikátorů $K_{i,j}$. Díky linearitě střední hodnoty (viz kuchařka) tedy platí $\mathbf{E}[K] = \sum_{i,j} \mathbf{E}[K_{i,j}] \leq \sum_{i,j} c/m$. Suma přitom sčítá přes $n(n-1)/2$ dvojic, což je menší než $n^2/2$, takže $\mathbf{E}[K] < cn^2/2m$.

Zkusíme, co se stane, zvolíme-li počet přihrádek m řádově kvadratický v počtu prvků n: konkrétně $m = cn^2$ (připomínáme, že c je konstanta z univerzality našich hešovacích funkcí). Tehdy průměrný počet kolidujících párů $\mathbf{E}[K]$ bude menší než $cn^2/2cn^2 = 1/2$.

Ukážeme, že s takhle nízkou střední hodnotou je dost pravděpodobné, že počet kolizí je nulový, a tedy funkce h je na naší množině prvků perfektní. K tomu se bude hodit tzv. *Markovova nerovnost.* Ta říká, že pro nezápornou náhodnou veličinu X a číslo a platí, že $P[X \geq a] \leq \mathbf{E}[X]/a$. Důkaz najdete například ve Wikipedii.[2]

Počítejme pomocí této nerovnosti pravděpodobnost, že naše funkce alespoň jednou zkoliduje: $P[K \geq 1] \leq \mathbf{E}[K]/1 < 1/2$. Funkce tedy bude perfektní s pravděpodobností větší než 1/2.

Nabízí se proto perfektní funkci volit tak, že vybereme náhodnou funkci (z c-univerzálního systému), ověříme, jestli je perfektní, a pokud není, postup opakujeme. Podle Lemmatu o džbánu se nám povede perfektní funkci najít v průměru po méně než dvou pokusech.

Úkol 3 [3b]: Vymyslete, jak jeden pokus provést v čase i prostoru $\mathcal{O}(n)$ v nejhorším případě.

Složíme-li vše, co jsme právě vymysleli, dohromady, dostaneme, že v průměrném čase $\mathcal{O}(n)$ sestrojíme hešovací funkci, která daných n prvků perfektně zahešuje do $\mathcal{O}(n^2)$ přihrádek.

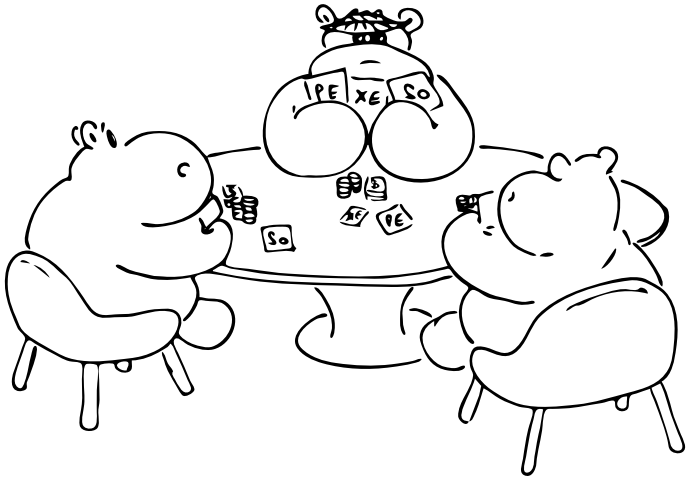
[2] `https://cs.wikipedia.org/wiki/Markovova_nerovnost`

Perfektní hešování: lineární prostor

Perfektní hešování by bylo daleko užitečnější, kdyby si vystačilo s lineárně mnoha přihrádkami. Tehdy bychom ho totiž mohli použít jako datovou strukturu pro reprezentaci statické (předem známé) množiny: do každé přihrádky bychom dali buď příslušný prvek množiny, nebo značku, že tam žádný prvek není. Pak bychom uměli testovat příslušnost prvku do množiny v konstantním čase (v nejhorším případě, žádný průměr!) – prvek bychom zahešovali, podívali bychom se do příslušné přihrádky a tam bychom buď hledaný prvek našli, nebo bychom si byli jisti, že v množině není.

Takovou datovou strukturu teď sestrojíme, ovšem naše hešování bude dvojúrovňové.

V první úrovni budeme mít *primární* hešovací funkci h vybranou náhodně z c-univerzálního systému, která zadané prvky $x_1, \ldots, x_n$ zahešuje do $m = cn$ přihrádek primární tabulky. Podle našich výpočtů bude pro průměrný počet kolidujících párů platit $\mathbf{E}[K] < cn^2/2cn = n/2$. Z Markovovy nerovnosti dostaneme $P[K \geq n] < 1/2$, takže budeme-li náhodně volit funkci h, dokud nedostaneme $K < n$, postačí nám podle Lemmatu o džbánu v průměru méně než 2 pokusy. Funkci h s méně než n kolidujícími páry tedy najdeme v průměrném čase $\mathcal{O}(n)$.



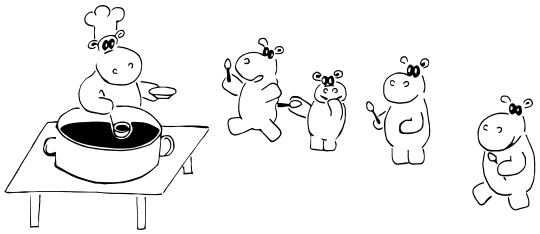
Do libovolné primární přihrádky se ovšem může zahešovat více prvků. Pro každou takovou přihrádku si pořídíme *sekundární* tabulku s další hešovací funkcí, která už pro prvky z primární přihrádky bude perfektní.

Nechť se do i-té primární přihrádky zahešuje n_i prvků. Velikost příslušné sekundární tabulky zvolíme jako $m_i = cn_i^2$, což je přesně naše nastavení pro perfektní hešování do kvadratického prostoru, takže v průměrném čase $\mathcal{O}(m_i)$ dokážeme najít funkci, která je na daných n_i prvcích perfektní.

To nám dohromady dává datovou strukturu, která umí zjišťovat přítomnost prvku v množině v konstantním čase v nejhorším případě. Není ale jasné, jak velká tato struktura bude (volba m_i kvadratického v n_i zní strašidelně). Pojďme ukázat, že to dobře dopadne.

Nejprve počítejme kolize primární funkce, konkrétně počet kolidujících párů K. V i-té přihrádce takových párů najdeme $n_i(n_i - 1)/2$. Sečtením přes všechny přihrádky dostaneme $K = \sum_i n_i(n_i - 1)/2$. Jelikož jsme primární funkci zvolili tak, aby měla $K < n$, platí $\sum_i n_i(n_i - 1)/2 < n$, tedy $\sum_i n_i(n_i - 1) < 2n$.

Teď se podívejme, kolik prostoru dohromady zabírají všechny sekundární tabulky. To je $\sum_i m_i = \sum_i cn_i^2 = c \cdot \sum_i n_i^2$. Tuto sumu můžeme napsat jako $\sum_i (n_i(n_i - 1) + n_i) = (\sum_i n_i(n_i - 1)) + (\sum_i n_i)$. První suma je podle předchozího odstavce menší než $2n$, druhá je rovná n (každý prvek padne do právě jedné primární přihrádky). Celkově je tedy $\sum_i m_i \in \mathcal{O}(n)$.



Primární tabulka zabere $\mathcal{O}(n)$ prostoru, protože každá její přihrádka si musí pamatovat odkaz na svou sekundární tabulku a parametry příslušné sekundární hešovací funkce. K tomu připočteme $\mathcal{O}(n)$ za všechny sekundární tabulky. Dohromady tedy spotřebujeme prostor $\mathcal{O}(n)$.

Ke konstrukci datové struktury potřebujeme nejprve najít primární hešovací funkci s méně než n kolizemi. To trvá průměrně $\mathcal{O}(n)$. Pak prvky v čase $\mathcal{O}(n)$ rozdělíme do primárních přihrádek a pro každou přihrádku najdeme sekundární hešovací funkci, která je perfektní. To trvá průměrně $\mathcal{O}(m_i)$, v součtu přes všechny přihrádky opět $\mathcal{O}(n)$. A nakonec v čase $\mathcal{O}(n)$ rozdělíme prvky do sekundárních přihrádek.

Naši datovou strukturu tedy dokážeme vybudovat v průměrném čase $\mathcal{O}(n)$ a pak v ní hledat ve worst-case konstantním čase.

Třídění v lineárním čase

Techniky z konstrukce perfektních hešovacích funkcí můžeme využít trochu překvapivým způsobem v následujícím algoritmu pro třídění reálných čísel. (Budeme předpokládat, že náš počítač umí provádět aritmetické operace s reálnými čísly v konstantním čase.)

Na vstupu dostaneme čísla $x_1, \ldots, x_n$ z intervalu $\langle 0, 1)$. Rozdělíme je do přihrádek očíslovaných 0 až $n - 1$, přičemž číslo x_i zařadíme do přihrádky $\lfloor x_i \cdot n \rfloor$. Pak přihrádky projdeme od první po poslední, obsah každé setřídíme libovolným kvadratickým třídicím algoritmem a vypíšeme.

Úkol 4 [3b]: Dokažte, že zvolíme-li $x_1, \ldots, x_n$ rovnoměrně náhodně z $\langle 0, 1)$, poběží náš třídicí algoritmus v průměrném čase $\mathcal{O}(n)$.

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/*