Informácie a pravidlá

Pre koho je súťaž určená?

Do **kategórie B** sa smú zapojiť len tí žiaci základných a stredných škôl, ktorí ešte ani v tomto, ani v nasledujúcom školskom roku nebudú končiť strednú školu.
Do **kategórie A** sa môžu zapojiť všetci žiaci (základných aj) stredných škôl.

Odovzdávanie riešení domáceho kola

Riešitelia domáceho kola odovzdávajú riešenia sami, v elektronickej podobe, a to priamo na stránke olympiády: http://oi.sk/. Odovzdávanie riešení bude spustené niekedy v septembri.

Riešenia kategórie A je potrebné odovzdať najneskôr **15. novembra 2018**.
Riešenia kategórie B je potrebné odovzdať najneskôr **30. novembra 2018**.

Priebeh súťaže

Za každú úlohu domáceho kola sa dá získať od 0 do 10 bodov. Na základe bodov domáceho kola stanoví Slovenská komisia OI (SK OI) pre každú kategóriu bodovú hranicu potrebnú na postup do **krajského kola**. Očakávame, že táto hranica bude približne rovná **tretine maximálneho počtu bodov**.

V krajskom kole riešitelia riešia štyri teoretické úlohy, ktoré môžu tematicky nadväzovať na úlohy domáceho kola. V kategórii B súťaž týmto kolom končí.

V kategórii A je približne najlepších 30 riešiteľov krajského kola (podľa počtu bodov, bez ohľadu na kraj, v ktorom súťažili) pozvaných do **celoštátneho kola**. V celoštátnom kole účastníci prvý deň riešia teoretické a druhý deň praktické úlohy. Najlepší riešitelia sú vyhlásení za víťazov. Približne desať najlepších riešiteľov následne SK OI pozve na týždňové výberové sústredenie. Podľa jeho výsledkov SK OI vyberie družstvá pre Medzinárodnú olympiádu v informatike (IOI) a Stredoeurópsku olympiádu v informatike (CEOI).

Ako majú vyzerať riešenia úloh?

V praktických úlohách je vašou úlohou vytvoriť program, ktorý bude riešiť zadanú úlohu. Program musí byť v prvom rade korektný a funkčný, v druhom rade sa snažte aby bol čo najefektívnejší.

V kategórii B môžete použiť ľubovoľný programovací jazyk.

V kategórii A musíte riešenia praktických úloh písať v jednom z podporovaných jazykov (napr. C++, Pascal alebo Java). Odovzdaný program bude automaticky otestovaný na viacerých vopred pripravených testovacích vstupoch. Podľa toho, na koľko z nich dá správnu odpoveď, vám budú pridelené body. Výsledok testovania sa dozviete krátko po odovzdaní. Ak váš program nezíska plný počet bodov, budete ho môcť vylepšiť a odovzdať znova, až do uplynutia termínu na odovzdávanie.

Presný popis, ako majú vyzerať riešenia praktických úloh (napr. realizáciu vstupu a výstupu), nájdete na webstránke, kde ich budete odovzdávať.

Ak nie je v zadaní povedané ináč, riešenia teoretických úloh musia v prvom rade obsahovať **podrobný slovný popis použitého algoritmu, zdôvodnenie jeho správnosti** a diskusiu o efektivite zvoleného riešenia (t. j. posúdenie časových a pamäťových nárokov programu). Na záver riešenia uveďte program. Ak používate v programe netriviálne algoritmy alebo dátové štruktúry (napr. rôzne súčasti STL v C++), súčasťou popisu algoritmu musí byť dostatočný popis ich implementácie.

Usporiadateľ súťaže

Olympiádu v informatike (OI) vyhlasuje *Ministerstvo školstva SR* v spolupráci so *Slovenskou informatickou spoločnosťou* (odborným garantom súťaže) a *Slovenskou komisiou Olympiády v informatike*. Súťaž organizuje *Slovenská komisia OI* a v jednotlivých krajoch ju riadia *krajské komisie OI*. Na jednotlivých školách ju zaisťujú učitelia informatiky. Celoštátne kolo OI, tlač materiálov a ich distribúciu po organizačnej stránke zabezpečuje IUVENTA v tesnej súčinnosti so Slovenskou komisiou OI.

A-I-1 Rušenie ulíc

*Toto je **praktická úloha**. Pomocou webového rozhrania odovzdajte **funkčný, odladený program**.*

Nedávny výskum ukázal, že kocúrkovskí vodiči majú problémy pri prejazde križovatkami, v ktorých sa stretá nepárny počet ulíc. Kocúrkovskí radní sa preto rozhodli tento problém riešiť, a to najjednoduchším spôsobom: jednoducho zrušia niektoré ulice tak, aby ich v každej križovatke ostal párny počet. Samozrejme, výberom toho, ktoré ulice zrušiť, poverili teba.
V tejto chvíli sa zrejme tešíš, aké to budeš mať jednoduché: zrušíš všetky ulice, takže ich v každej križovatke ostane nula. Ale pomaly! Problém je v tom, že kocúrkovskí radní ti nedovolia zrušiť žiadnu z ulíc, ktoré oni sami používajú.

Formát vstupu a výstupu

V Kocúrkove je n križovatiek, sú očíslované od 1 po n. Každá ulica je obojsmerná a spája dve rôzne križovatky. Každé dve križovatky sú priamo spojené nanajvýš jednou ulicou.
V prvom riadku vstupu sú čísla n, m a k, kde m je celkový počet ulíc a k je počet dôležitých ulíc, ktoré nesmieš zrušiť. V každom z ďalších m riadkov sú čísla dvoch križovatiek spojených jednou z ulíc. Prvých k ulíc sú tie, ktoré je zakázané zrušiť.

Na výstup vypíšte niekoľko riadkov. V prvom riadku vypíšte číslo z: počet ulíc, ktoré chcete zrušiť. Následne vypíšte z riadkov a v každom z nich čísla križovatiek, ktoré spája jedna z ulíc, ktoré chcete zrušiť. (Každú ulicu vypíšte práve raz. Poradie čísel v riadku aj poradie riadkov popisujúcich ulice môže byť ľubovoľné. Ak existuje viacero správnych riešení, môžete vypísať ľubovoľné z nich.)
Ak riešenie neexistuje, vypíšte namiesto toho všetkého len jeden riadok a v ňom číslo -1.

Obmedzenia a hodnotenie

Odovzdané riešenie bude otestované na desiatich sadách vstupov, za každú je bod.
Vo všetkých sadách platí $1 \leq n$ a $0 \leq k \leq m$.

V sadách 1 a 2 sú všetky ulice dôležité ($k = m$).
V sadách 3, 4 a 5 je nanajvýš jedna dôležitá ulica ($k \leq 1$).
Vstupy v sadách 3 a 6 sú malé: $n, m \leq 20$.
Vstupy v sadách 1, 4 a 7 sú stredne veľké: platí $n \leq 2000$ a $m \leq 10\,000$.
Vstupy v ostatných sadách sú veľké: $n \leq 1\,000\,000$ a $m \leq 5\,000\,000$.

Príklady

vstup
```
5 6 1
1 2
1 3
2 3
1 4
4 5
2 5
```

výstup
```
3
1 4
5 4
5 2
```

vstup
```
3 2 2
1 2
2 3
```

výstup
```
-1
```

vstup
```
3 3 2
1 2
1 3
2 3
```

výstup
```
0
```

V prvom príklade je iným správnym riešením zrušiť obe ulice vedúce z križovatky číslo 3. Všimnite si, že nie je potrebné zrušiť najmenší možný počet ulíc.

V druhom príklade riešenie neexistuje, keďže z križovatky 1 vedie jediná ulica a nesmieme ju zrušiť.

V treťom príklade je jediným riešením nezrušiť žiadnu ulicu.

A-I-2 Cyklistický závod

*Toto je **praktická úloha**. Pomocou webového rozhrania odovzdajte **funkčný, odladený program**.*

V kopcoch nad Kocúrkovom sa nachádza kopec cestičiek pre horské bicykle. Každá cestička má vyznačený smer, v ktorom sa po nej smie ísť. (Čelné zrážky bicyklov sú fakt nepríjemné.) Na každej cestičke sú dve stopy pre bicykle, aby mohli dvaja kamaráti ísť vedľa seba – alebo spolu závodiť.
Danka a Samko sa pri pive pohádali, kto z nich je lepší cyklista. Rozhodli sa, že to vyriešia závodom. Samko vyberie nejakú uzavretú trasu (teda takú, ktorá skončí tam kde začínala) a kto ju prejde skôr, ten vyhrá.
Ráno pred súťažou si však Danka uvedomila, že v tom môže byť háčik. Bežne sa totiž stáva, že kvôli rôznym terénnym nerovnostiam je ľavá stopa na cestičke ináč dlhá ako pravá. Čo ak tá potvora Samko vyberie takú trasu, na ktorej bude jedna stopa dokopy výhodnejšia ako druhá?
Pomôžte Danke zistiť, či jej Samko môže vyviesť takúto galibu.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n a m: počet križovatiek a počet cestičiek. Križovatky sú očíslované od 1 po n. Zvyšok vstupu tvorí m riadkov, každý z nich popisuje jednu cestičku. Popis cestičky má tvar „x_i y_i l_i r_i“. Čísla x_i a y_i sú čísla križovatiek odkiaľ a kam táto cestička vedie. Čísla l_i a r_i udávajú dĺžku ľavej a pravej stopy pre túto cestičku. Pre každú cestičku platí, že $x_i \neq y_i$. Môže existovať viacero cestičiek s tou istou dvojicou (x_i, y_i). Cestičky sú očíslované od 1 po m v poradí, v ktorom sú zadané na vstupe.

Ak existuje okružná trasa, ktorej ľavá stopa má v súčte inú dĺžku ako pravá stopa, ľubovoľnú jednu takúto trasu nájdite a vypíšte. Presnejšie, vypíšte jeden riadok a v ňom postupnosť čísel cestičiek v poradí, v akom ležia na trase. Ak taká okružná trasa neexistuje, vypíšte namiesto toho jeden riadok a v ňom číslo 0.
Upozorňujeme, že trasa smie aj viackrát prechádzať tou istou cestičkou.

Obmedzenia a hodnotenie

Odovzdané riešenie bude otestované na desiatich sadách vstupov, za každú je bod.
Vo všetkých vstupoch platí $2 \leq n \leq 1\,000\,000$, $1 \leq m \leq 1\,000\,000$, $1 \leq x_i, y_i \leq n$ a $1 \leq l_i, r_i \leq 1000$.

V sadách 1, 2 a 3 platí $n, m \leq 20$.
V sadách 4 a 5 platí, že existuje nanajvýš 6 križovatiek so stupňom viac ako 2. V sade 4 navyše platí $m \leq 1000$. (Stupeň križovatky je počet vchádzajúcich plus počet vychádzajúcich cestičiek.)
V sadách 6 a 7 platí $n, m \leq 1000$.
V sadách 6 a 8 platí, že z každej križovatky sa po cestičkách dá dostať na každú inú križovatku.

Príklady

vstup

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

výstup

```
1 4 5
```

vstup

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

výstup

```
0
```

vstup

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

výstup

```
0
```

Výstup v prvom príklade popisuje okruh z križovatky 1 cez križovatky 2 a 4 späť na križovatku 1. Ľavá stopa tohto okruhu má dĺžku 1+1+1, zatiaľ čo pravá 1+1+2. Iné správne výstupy sú napr. „4 5 1“ a „1 2 3 1 4 5“.

V druhom príklade v podstate existuje len jediný okruh tvorený ulicami 1, 2 a 3. Danka a Samko na ňom môžu spokojne súťažiť, keďže obe jeho stopy sú v súčte rovnako dlhé. V treťom príklade sa nedá navrhnúť vôbec žiadna okružná trasa. Tým skôr neexistuje žiadna okružná trasa na ktorej by sa líšili dĺžky stôp.

A-I-3 Trojuholník

Toto je teoretická úloha. Pomocou webového rozhrania odovzdajte súbor vo formáte PDF, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách. Každá podúloha je hodnotená samostatne, nie je nutné ich riešiť v uvedenom poradí.

Na vstupe je n bodov v rovine (pričom $n \geq 3$). Žiadne tri z nich neležia na jednej priamke. Spomedzi všetkých trojuholníkov, ktoré majú vrcholy v zadaných bodoch, nájdite ten, ktorý má najmenší obsah.
Pri písaní svojho programu môžete predpokladať, že všetky výpočty s reálnymi číslami sú presné. Nemusíte sa teda zaoberať zaokrúhľovacími chybami, ktoré by vznikali v praxi.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n.
V každom zo zvyšných n riadkov sú dve celé čísla x_i a y_i: súradnice jedného z bodov.

Na výstup vypíšte tri riadky a v nich súradnice vrcholov trojuholníka s najmenším obsahom.

Obmedzenia a hodnotenie

Za riešenie s časovou zložitosťou kubickou od n môžete získať nanajvýš 4 body.
Za ľubovoľné riešenie s časovou zložitosťou $O(n^2 \log n)$ môžete získať 10 bodov.

Príklady

vstup

```
4
0 0
0 10
10 0
10 10
```

výstup

```
0 0
0 10
10 10
```

Všetky štyri trojuholníky majú rovnaký obsah, vypísať môžeme ľubovoľný z nich.

vstup

```
5
0 0
100 100
-100 -30
2 0
0 1
```

výstup

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

A-I-4 Online algoritmy

Toto je teoretická úloha. Pomocou webového rozhrania odovzdajte súbor vo formáte PDF, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách. K tejto úlohe patrí študijný text uvedený na nasledujúcich stranách. Odporúčame najskôr prečítať ten a až potom sa vrátiť k samotným súťažným úlohám.

Dvaja zlodeji majú vrece s lupom. Vo vreci je n predmetov. Brali ich potme, takže o nich nič nevedia. Zlodeji si tento lup teraz idú rozdeliť. Robia to tak, že vždy vytiahnu z vreca jeden predmet, pozrú sa na jeho cenu a rozhodnú sa, kto ho dostane. Zlodeji by si chceli veci rozdeliť čo najviac rovnomerne. Presnejšie, ich cieľom je, aby súčet cien na drahšej kôpke bol čo najmenší.

Podúloha A (2 body): Dokážte, že algoritmus, ktorý dá všetky predmety jednému lupičovi, je 2-kompetitívny.
Podúloha B (4 body): Nájdite 3/2-kompetitívny algoritmus pre tento problém.
Podúloha C (4 body): Dokážte, že žiadny algoritmus pre tento problém nemôže byť lepší ako 3/2-kompetitívny.

Študijný text: online algoritmy

V úlohách našej olympiády sa zväčša stretnete s úlohami, v ktorých dostanete naraz celý vstup a z neho máte vypočítať optimálny výstup. V praxi sa však občas vstup dozvedáme postupne a náš algoritmus musí už priebežne robiť nejaké rozhodnutia. Takýmto problémom hovoríme *online* problémy.

Príklad: Stránkovanie

Počítače majú len obmedzené množstvo fyzickej pamäte. Aby na nich mohlo bežať viac procesov naraz, rieši sa to tak, že majú väčšie množstvo pamäte virtuálnej. Tej si každý program alokuje, koľko potrebuje. V prípade, že programy spotrebujú virtuálnej pamäte viac ako je dostupnej fyzickej, prichádza následne k *swapovaniu*: vždy, keď nejaký program potrebuje pristupovať ku svojej pamäti, presunieme príslušný kus údajov do fyzickej pamäte. Ak je tá ale už plná, musíme najskôr uvoľniť nejaké miesto v nej.
My sa podrobnejšie pozrieme na jeden konkrétny spôsob, ako robiť takúto správu pamäte. Pôjde o tzv. *stránkovanie*. Fyzickú pamäť (napr. RAM) si rozdelíme na k rovnakých kusov nazývaných *stránky*. Hodnotu k budeme považovať za pevne zvolenú konštantu. Vo virtuálnej pamäti (napr. na pevnom disku) máme priestor pre ľubovoľne veľa stránok. Stránky vo virtuálnej pamäti si označíme prirodzenými číslami.
Naša úloha teraz bude vyzerať nasledovne: Predpokladajme, že na začiatku je celá fyzická pamäť prázdna. Postupne nám budú chodiť požiadavky. Každá požiadavka bude tvaru „virtuálnu stránku a_i treba dať do fyzickej pamäte“. Ak tam táto stránka momentálne už je, nemusíme robiť nič, ak tam nie je, musíme na ňu spraviť miesto (ak už je fyzická pamäť plná, niektorú stránku z nej odswapovať na disk) a následne ju tam nahrať. Toto je pomalé, preto by sme radi **minimalizovali počet nahratí stránky do fyzickej pamäte**.
Príklad: Uvažujme situáciu v ktorej máme len $k = 2$ stránky fyzickej pamäte. Postupne nám prídu požiadavky 7, 2, 3, 2. Jedna možnosť, ako ich môžeme spracovať, vyzerá nasledovne:

1. Potrebujeme virtuálnu stránku 7. Nahráme ju do fyzickej pamäte na prvú pozíciu.
   Fyzická pamäť teda vyzerá nasledovne: (7, nič).
2. Potrebujeme virtuálnu stránku 2. Nahráme ju na druhú pozíciu. Fyzická pamäť: (7, 2).
3. Potrebujeme virtuálnu stránku 3. Odswapujeme stránku 7 a nahráme stránku 3. Fyzická pamäť: (3, 2).
4. Potrebujeme virtuálnu stránku 2. Tú ešte stále máme vo fyzickej pamäti, nemusíme teda nič robiť.

Dokopy sme teda 3× museli nahrávať stránku do fyzickej pamäte. Všimnite si, že ak by sme v kroku 3 namiesto virtuálnej stránky 7 vyhodili virtuálnu stránku 2, museli by sme stránku do fyzickej pamäte nahrávať až 4×.

Ktorý online algoritmus je lepší?

Zamyslime sa ešte nad príkladom, ktorý sme práve videli. Keby sme vopred poznali celú postupnosť požiadaviek, je zjavné, že by sme si vybrali prvé riešenie – to, ktoré potrebovalo len trikrát nahrávať stránku. Problém je ale

v tom, že ju nevieme. Keď prišla tretia požiadavka, musíme sa hneď vtedy rozhodnúť, ktorú stránku z fyzickej pamäte vyhodiť. A nech sa rozhodneme akokoľvek, vždy bude možné, že ako ďalšia príde zrovna tá požiadavka, ktorá nám to celé pokazí. Online algoritmy teda vo všeobecnosti nebudú vedieť zaručiť, že zostroja optimálne riešenie.

Keď chceme porovnať dva online algoritmy, potrebujeme vedieť nejak odmerať, ktorý z nich je lepší. Intuitívne by sme čakali, že lepší bude ten, ktorý sa od optimálneho riešenia bude menej líšiť. Ako ale túto intuíciu zachytiť formálnou definíciou?

Podobne ako pri časovej zložitosti, aj tu sa sústredíme na najhorší prípad: bude nás zaujímať *koľkokrát horšie* môže byť riešenie vyrobené online algoritmom v porovnaní s tým, ktoré by vyrobil offline, teda „vševedúci“, algoritmus *pre ten istý vstup*.

Uvažujme optimalizačné problémy, v ktorých je naším cieľom minimalizovať hodnotu riešenia. Majme konkrétny online algoritmus. Pre daný vstup v si označme $ALG(v)$ hodnotu riešenia, ktoré vyrobí on, a $OPT(v)$ hodnotu riešenia, ktoré by vyrobil optimálny offline algoritmus. Ak existuje reálna konštanta c taká, že pre úplne všetky v platí $ALG(v) \leq c \cdot OPT(v)$, tak hovoríme, že náš online algoritmus je *c-kompetitívny*. A samozrejme, čím je c bližšie k 1, tým je ten algoritmus lepší.

Poznámka o determinizme

Vyššie uvedenú definíciu a aj analýzu konkrétnych algoritmov by nám výrazne skomplikovalo, keby sme im dovolili používať náhodné čísla. Keďže vás tým nechceme trápiť, používanie náhody pre istotu úplne zakážeme a budeme sa zaoberať len deterministickými algoritmami. Podotýkame, že taktiež je zakázané používať generátor náhodných čísel pri riešení zadaných súťažných úloh.

Algoritmus „zásobník“

Stále sa zaoberáme problémom stránkovania, ktorý sme si popísali vyššie. V tejto časti sa pozrieme na konkrétny online algoritmus. Bude veľmi jednoduchý: vždy, keď je fyzická pamäť plná a algoritmus do nej potrebuje nahrať stránku, ktorá tam nie je, tak vyhodí tú stránku, ktorá tam je najkratšie.

Je tento algoritmus c-kompetitívny pre nejaké c? Ukážeme si, že nie.

Všimnime si vstup dĺžky $n > k$ ktorý vyzerá nasledovne:

$$1,\ 2,\ \ldots,\ k-1,\ k,\ k+1,\ k,\ k+1,\ \ldots,\ k,\ k+1,\ \ldots$$

Čo spraví algoritmus „zásobník“ pre tento vstup? Prvými k požiadavkami si zaplní fyzickú pamäť a potom bude na striedačku vyhadzovať stránku k a $k+1$ a nahrádzať ju tou druhou. Dokopy teda bude nahrávať stránku do fyzickej pamäte až n-krát (čiže v úplne každom kroku).

Ako vyzerá optimálne riešenie pre tento istý vstup? Zjavne po rovnakých prvých k krokoch stačí v $(k+1)$. kroku vyhodiť z pamäte ľubovoľnú stránku inú ako k. Od tej chvíle už budeme mať vo fyzickej pamäti aj stránku k, aj stránku $k+1$, a teda už nebudeme potrebovať nikdy nič ďalšie nahrávať. Dokopy teda nahráme stránku do fyzickej pamäte len $(k+1)$-krát.

No a teda tento algoritmus nemôže byť c-kompetitívny pre žiadne reálne c, lebo čím väčší vstup zoberieme, tým väčší bude pomer $n/(k+1)$. Teda ak by napríklad niekto tvrdil, že algoritmus „zásobník“ je 1000-kompetitívny, vieme mu to vyvrátiť napríklad tak, že zoberieme vstup vyššie popísaného tvaru v ktorom $n = 1001(k+1)$.

Algoritmus „fronta“

Pozrime sa teraz na druhý veľmi jednoduchý algoritmus. Od algoritmu „zásobník“ sa líši len v tom, že vždy, keď potrebuje uvoľniť miesto vo fyzickej pamäti, vyhodí tú stránku, ktorú má vo fyzickej pamäti *najdlhšie*.

Dokážeme si, že algoritmus „fronta“ je k-kompetitívny.

Uvažujme ľubovoľný vstup. Nech optimálne riešenie nahráva nejakú stránku do pamäte presne m-krát, a nech je to po požiadavkách s indexmi $i_1, \ldots, i_m$. (Všimnite si, že nutne platí $i_1 = 1$, keďže na začiatku je pamäť prázdna a teda prvú požadovanú stránku do nej musíme nahrať.)

Rozdeľme si teraz vstup na m úsekov, pričom j-ty úsek tvoria požiadavky s číslami i_j až $i_{j+1} - 1$. (Posledný úsek je od i_m po koniec vstupu.)

Všimnime si, že úseky môžu síce byť ľubovoľne dlhé, ale v každom úseku sa môže vyskytovať len nanajvýš k rôznych stránok. Totiž keďže počas j-teho úseku optimálny algoritmus nahráva do pamäte novú stránku len v

kroku i_j, musia všetky požiadavky v tomto úseku byť len na tie stránky, ktoré má náš optimálny algoritmus práve v pamäti po spracovaní požiadavky s indexom i_j.
To ale znamená, že algoritmus „fronta“ počas spracúvania tohto úseku nahrá stránku do pamäte nanajvýš k-krát. Totiž počas spracúvania úseku stretneme požiadavky na nanajvýš k rôznych stránok, a ľahko nahliadneme, že každú z nich počas spracúvania úseku nanajvýš raz nahráme do pamäte. Prečo? Lebo na to, aby nám pri použití algoritmu „fronta“ nejaká stránka z pamäte vypadla, potrebujeme po nej nahrať do pamäte iných k stránok, a to počas tohto úseku nenastane.
Inými slovami, na ľubovoľnom vstupe, na ktorom optimálny algoritmus nahráva stránku do pamäte m-krát, nahráva algoritmus „fronta“ stránku do pamäte nanajvýš km-krát. A z toho priamo podľa definície vidíme, že algoritmus „fronta“ je k-kompetitívny.

Optimálnosť algoritmu „fronta“

V tejto časti si dokážeme, že pre online problém stránkovania nemôže existovať lepší ako k-kompetitívny algoritmus – a teda že algoritmus „fronta“ je v tomto zmysle optimálnym online algoritmom.
Uvažujme úplne ľubovoľný deterministický online algoritmus pre problém stránkovania. Začneme tým, že špeciálne preň zostrojíme vstup, ktorý ho donúti veľakrát nahrávať stránku do pamäte.
Vstup budeme zostrojovať priebežne počas behu samotného algoritmu. Prvých k požiadaviek budú stránky s číslami 1 až k. Každú ďalšiu požiadavku zvolíme nasledovne: pozrieme sa na to, ktorých k stránok má náš algoritmus práve v pamäti, a ako požiadavku mu zadáme najmenšie číslo stránky, ktorú v pamäti nemá. (Kvôli neskoršej analýze si všimnite, že toto číslo bude zjavne vždy z rozsahu 1 až $k+1$.)
Zjavne k ľubovoľnému online algoritmu a číslam n a k vieme takto zostrojiť jeden konkrétny vstup dĺžky n, pre ktorý tento online algoritmus bude musieť nahrávať stránku do pamäte v úplne každom kroku. Pre tento vstup v teda platí $ALG(v) = n$.
Čo vieme povedať o hodnote $OPT(v)$? Uvažujme nasledovný (nie nutne optimálny) offline algoritmus: Prvých k krokov má vynútených, rovnako ako v predchádzajúcom riešení musí nahrať do pamäte stránky s číslami 1 až k a zatiaľ nemá zmysel žiadnu z nich vyhadzovať. Od tohto okamihu bude náš offline algoritmus postupovať nasledovne: vždy, keď treba nejakú stránku z pamäte vyhodiť, pozrieme sa do budúcnosti na k požiadaviek začínajúc práve spracúvanou a vyhodíme z pamäte ľubovoľnú stránku, ktorá sa medzi nimi nenachádza.
Všimnime si, že v požiadavkach, ktoré spracúvame, sa používa len $k+1$ rôznych stránok. Preto vždy v okamihu, keď nejakú stránku z pamäte vyhodíme a nahradíme inou, budeme mať v pamäti všetkých k stránok iných od tej práve vyhodenej. To ale znamená, že vždy, keď náš algoritmus vyhodí z pamäte stránku, ktorú k krokov nebude potrebovať, tak má v pamäti všetky stránky, ktoré potrebovať bude. Inými slovami, vyššie popísaný algoritmus pre náš vstup v vždy po nahraní stránky do pamäte bude mať aspoň $k-1$ krokov, počas ktorých zaručene do pamäte nič nahrávať nebude.
Inými slovami, na vstupe dĺžky n tento offline algoritmus nahrá do pamäte stránku nanajvýš $(k + \lceil (n-k)/k \rceil)$-krát. A keďže optimálny algoritmus je aspoň taký dobrý ako hocijaký iný, platí $OPT(v) \leq k + \lceil (n-k)/k \rceil$.
Platí teda:

$$\frac{ALG(v)}{OPT(v)} \geq \frac{n}{k + \lceil (n-k)/k \rceil} \geq \frac{n}{(n+k^2)/k} = k \cdot \frac{n+k^2-k^2}{n+k^2} = k \cdot \left(1 - \frac{k^2}{n+k^2}\right)$$

Všimnite si, že pre dostatočne dlhé vstupy (t.j. dostatočne veľké n) bude výraz na pravej strane nadobúdať hodnoty ľubovoľne blízke k. To ale znamená, že žiaden online algoritmus pre problém stránkovania nemôže byť c-kompetitívny pre žiadne $c < k$, čo je presne to, čo sme chceli dokázať.

TRIDSIATY ŠTVRTÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2018