Priebeh krajského kola

Krajské kolo 34. ročníka Olympiády v informatike, kategória A, sa koná 22. januára 2019 v dopoludňajších hodinách. Na riešenie úloh majú súťažiaci **4 hodiny čistého času**. Rôzne úlohy riešia súťažiaci na samostatné listy papiera. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

- Slovne popíšte algoritmus.
  Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného algoritmu/programu.
- Zdôvodnite správnosť vášho algoritmu.
- Uveďte a zdôvodnite jeho časovú a pamäťovú zložitosť.
- Podrobne uveďte dôležité časti algoritmu, ideálne vo forme programu v nejakom bežnom programovacom jazyku (napr. C++, Python, Java, Pascal).
- V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu stručne vysvetlite, ako by ste napísali program s rovnakou časovou zložitosťou bez použitia knižnice.

Hodnotenie riešení

Za každú úlohu môžete získať od 0 do 10 bodov.

Pokiaľ nie je v zadaní povedané ináč, najdôležitejšie dve kritériá hodnotenia sú v prvom rade **správnosť** a v druhom rade **efektívnosť** navrhnutého algoritmu. Na výslednom počte bodov sa môže prejaviť aj kvalita popisu riešenia a zdôvodnenie tvrdení o jeho správnosti a efektívnosti.

Efektívnosť algoritmu posudzujeme vypočítaním jeho časovej zložitosti – funkcie, ktorá hovorí, ako dlho vykonanie algoritmu trvá v závislosti od veľkosti vstupných parametrov. Nezávisí pri tom na konštantných faktoroch, len na rádovej rýchlosti rastu tejto funkcie.

V zadaní úloh uvádzame časť „Hodnotenie“, v ktorej nájdete približné limity na veľkosť vstupných údajov. Pod pojmom „efektívne vyriešiť“ chápeme to, že váš program spustený na modernom počítači by mal dať odpoveď nanajvýš do niekoľkých sekúnd.

Údaje z tejto časti zadania by mali slúžiť hlavne na to, aby ste o riešení, ktoré vymyslíte, vedeli približne povedať, koľko bodov zaň dostanete.

A-II-1 Tulipány

Evina sa nedávno presťahovala do Holandska. Aby sa tam udomácnila, rozhodla sa, že pred domom vysadí tulipány. Spravila teda jednu dlhú rovnú hriadku, tú rozdelila na n políčok a na každé políčko zasadila tulipán. Po pár dňoch však zistila, že jej niektoré sadenice zožrali slimáky, a tak má teraz viacero skupín tulipánov, oddelených od seba prázdnymi políčkami.
Evina si nenechala slimačím nájazdom pokaziť deň a jednoducho prehlásila, že z každej súvislej skupiny tulipánov bude jedna kytica.
Potom si ale uvedomila, že v každej kytici by mal byť nepárny počet kvetov. Možno teda predsa len treba aspoň na niektoré prázdne políčka vysadiť znova tulipány, a to tak, aby platilo, že každú súvislú skupinu tulipánov bude tvoriť nepárne veľa kvetov.

Súťažná úloha

Daný je popis Evininho záhonu: o každom políčku viete, či momentálne obsahuje tulipán alebo nie. Napíšte čo najefektívnejší program, ktorý zistí, koľko najmenej tulipánov ešte Evina musí vysadiť, ak chce, aby každá súvislá skupina tulipánov mala nepárnu dĺžku.
(Tulipány nesmie sadiť mimo záhonu ani na políčka, ktoré už tulipán obsahujú. Na každé prázdne políčko smie zasadiť najviac jeden tulipán. Všimnite si, že zasadenie nových tulipánov môže spojiť niekoľko pôvodných skupín tulipánov do jednej väčšej.)

Formát vstupu a výstupu

Na vstupe je popis záhonu: reťazec dĺžky n tvorený znakmi „`T`“ a „`.`“ (predstavujúcimi tulipán a voľné miesto). Je zaručené, že n je nepárne.
Na výstup vypíšte jedno prirodzené číslo: najmenší počet tulipánov, ktoré ešte treba vysadiť. (Rozmyslite si, že keďže n je nepárne, nejaké riešenie vždy bude existovať.)

Hodnotenie

Plný počet bodov dostanete za riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 100\,000$.
Za úplné riešenie, ktoré korektne a rozumne rýchlo vyrieši ľubovoľný vstup s $n \leq 20$, sa dá získať 4 body.
Existujú aj iné riešenia, ktorých časová zložitosť je horšia od prvého a lepšia od druhého z vyššie uvedených. Tieto dostanú primerane vyšší počet bodov ako 4.

Príklady

vstup	výstup
`T.TT.TT..TT`	`2`

Jediným optimálnym riešením je vysadiť tulipány ešte na piate a deviate políčko, čím vznikne záhon `T.TTTTT.TTT`*, v ktorom má každá skupina nepárny počet tulipánov.*

vstup	výstup
`.....`	`0`

A-II-2 Ohrada

Na lúke stojí n stromov. Chceme spraviť ohradu pre sliepky tak, že okolo štyroch stromov natiahneme špagát. Navyše musí platiť, že v ohrade už neleží žiaden iný strom (aby naň sliepky nemohli vyletieť a potom z jeho konárov spadnúť cez ohradu). Dá sa to? Ak áno, ako?

Súťažná úloha

Daných je n bodov v rovine, pričom môžete predpokladať, že $n \geq 4$ a že **žiadne tri body neležia na priamke**. Nájdite ľubovoľnú štvoricu spomedzi daných bodov, pre ktorú platí:

- Vybrané body tvoria vrcholy konvexného štvoruholníka.
  (Konvexný štvoruholník má všetky vnútorné uhly menšie ako 180°.)
- Tento štvoruholník neobsahuje vo vnútri žiaden iný z daných bodov.

Nezabudnite na dôkaz správnosti vášho algoritmu.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n, v každom z ďalších n riadkov sú súradnice jedného z bodov. Všetky súradnice sú celé čísla.
Na výstup vypíšte buď súradnice štvorice vybraných bodov, alebo správu o tom, že riešenie neexistuje.

Hodnotenie

Plný počet bodov dostanete za riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n \leq 100\,000$.
Za úplné riešenie, ktoré korektne a rozumne rýchlo vyrieši ľubovoľný vstup s $n \leq 20$, sa dá získať 4 body.
Existujú aj iné riešenia, ktorých časová zložitosť je horšia od prvého a lepšia od druhého z vyššie uvedených. Tieto dostanú primerane vyšší počet bodov ako 4.

Príklady

vstup

```
5
0 0
10 0
5 10
5 1
6 -1
```

výstup

```
0 0
6 -1
10 0
5 1
```

vstup

```
4
0 0
10 0
5 10
5 1
```

výstup

```
neda sa
```

A-II-3 Úrady

Horystan je krajina, ktorá celá leží na svahu obrovskej hory. V krajine je n miest. Mestá sú očíslované od 1 po n, pričom platí, že čím väčšie číslo, tým väčšia nadmorská výška.
Niektoré dvojice miest sú prepojené zjazdovkami. Po zjazdovke sa dá dostať z vyššie položeného mesta do nižšie položeného. Niektoré dvojice miest sú prepojené cestami. Po ceste sa dá chodiť oboma smermi. Konkrétna dvojica miest môže byť prepojená naraz aj cestou, aj zjazdovkou. Pre vaše pohodlie však môžete predpokladať, že každú dvojicu miest spája nanajvýš jedna cesta a nanajvýš jedna zjazdovka.
Veľký vezír Horystanu by chcel v niektorých mestách zriadiť úrady, ktoré budú kontrolovať výber daní. Úrad zriadený v meste x vie kontrolovať okrem samotného mesta x aj niektoré iné mestá, a to nasledovne:

- Vie vyslať kontrolóra na koni. Ten vie skontrolovať ľubovoľné mesto, do ktorého sa z x dá dostať postupným použitím jednej alebo viac ciest.
- Vie vyslať kontrolóra na lyžiach. Ten vie skontrolovať ľubovoľné mesto, do ktorého sa z x dá dostať postupným použitím jednej alebo viac zjazdoviek (samozrejme, len smerom dodola).

(Všimnite si, že použitie ciest a zjazdoviek nie je povolené striedať. Netrápte sa osudom kontrolórov na lyžiach po tom ako dosiahnu svoju destináciu, raz za čas ich pozbierajú helikoptéry a dovezú naspäť domov.)
Historické skúsenosti navyše ukazujú, že je zle, ak existujú dve mestá, ktoré susedia (t.j. vedie medzi nimi cesta alebo zjazdovka) a ktoré obe obsahujú úrady. Niekedy sa takéto úrady príliš hašteria o kompetencie, inokedy zase začnú spolu kuť pikle proti vezírovi. Najlepšie bude rovno to celé zakázať.

Súťažná úloha

Napíšte program, ktorý do krajiny rozmiestni úrady tak, aby platilo:

- Každé mesto v krajine musí byť kontrolované aspoň jedným úradom.
- Ak dve mestá susedia, úrad sme byť len v nanajvýš jednom z nich.

Formát vstupu a výstupu

V prvom riadku vstupu sú tri čísla: počet miest n, počet zjazdoviek z a počet ciest c. Každý z nasledujúcich z riadkov obsahuje popis jednej zjazdovky (teda čísla miest odkiaľ a kam vedie). Každý zo zvyšných c riadkov obsahuje popis jednej cesty (teda čísla miest ktoré spája).
Na výstup vypíšte buď ľubovoľný jeden zoznam čísel miest, v ktorých postaviť úrady, alebo správu o tom, že riešenie neexistuje.

Hodnotenie

Plný počet bodov dostanete za riešenie, ktoré efektívne vyrieši ľubovoľný vstup s $n, z, c \leq 100\,000$.
Za úplné riešenie, ktoré korektne a rozumne rýchlo vyrieši ľubovoľný vstup s $n \leq 20$, sa dá získať 4 body.
Existujú aj iné riešenia, ktorých časová zložitosť je horšia od prvého a lepšia od druhého z vyššie uvedených. Tieto dostanú primerane vyšší počet bodov ako 4.

Príklad

vstup

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

výstup

```
5 3
```

V krajine máme zjazdovky $2 \to 1$, $5 \to 2$, $5 \to 4$ *a cesty* $5 - 6$, $2 - 5$ *a* $3 - 4$.
Mestá 3 a 5 nesusedia, môžu teda v oboch byť úrady. Tie vedia kontrolovať každé iné mesto: napr. do miest 1, 2 a 4 pošleme z mesta 5 kontrolórov na lyžiach a do mesta 6 pošleme z mesta 5 kontrolóra na koni.

Existujú aj iné riešenia, napr. „2 3“, „2 4“, alebo „2 6 4“. Všimnite si, že nie je potrebné, aby bol počet úradov minimálny možný.

A-II-4 Online algoritmy 2

Táto súťažná úloha nadväzuje na úlohu z domáceho kola. Za samotným zadaním úlohy nájdete študijný text. Ten je totožný so študijným textom zo zadaní domáceho kola.

Zajtra začínaš brigádovať vo vedľajšom meste. Nevieš, ako dlho bude táto brigáda trvať. Začínajúc zajtrajškom, každé ráno dostaneš SMS od šéfa, a to buď s textom „aj dnes ešte príď“ alebo s textom „brigáda už skončila“. Ak brigáda ešte pokračuje, ideš na vlakovú stanicu a tam si kúpiš na ten deň spiatočný lístok. Normálny spiatočný lístok stojí 2 eurá.
Každý večer (aj dnes, pred prvým dňom tvojej brigády) sa dá na stanici zakúpiť zľavová karta. Zľavová karta stojí k eur, pričom $k \geq 1$ je nejaké konkrétne celé číslo ktoré poznáš. Vlastníci zľavovej karty smú jazdiť na polovičné lístky. Teda akonáhle si kúpiš zľavovú kartu, budeš si každé ďalšie ráno môcť kúpiť spiatočný lístok za 1 euro.
Tvojim cieľom je za celú brigádu minúť na cestovnom (teda dokopy za lístky a prípadne zľavovú kartu) čo najmenej. Pozor, brigáda môže skončiť aj úplne hneď – už zajtra ráno môžeš dostať SMS, že je koniec.
Navrhni algoritmus pre túto úlohu, pričom sa snaž, aby tvoj algoritmus bol c-kompetitívny pre nejaké c (samozrejme, čím menšie c, tým lepšie). Inými slovami, musí platiť, že tvoj algoritmus vždy zaplatí nanajvýš c-krát viac ako by zaplatil optimálne sa rozhodujúci človek, ktorý vopred vie, koľko dní bude brigáda trvať.
Na plný počet bodov treba nájsť optimálny online algoritmus a aj o ňom dokázať, že je optimálny.

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 2019