Priebeh krajského kola

Krajské kolo 32. ročníka Olympiády v informatike, kategória A, sa koná 17. januára 2017 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 Vzostup a pád Kocúrkova

Kocúrkovský historik, pán Rudolf, sa rozhodol, že napíše knihu o vzostupe a následnom páde jeho rodnej obce. Esejí by už mal napísaných aj na jeden a pol knihy, jeho vydavateľ sa však neustále dožaduje nejakých objektívnych dát. Rudolf si však poradí! Podarilo sa mu zistiť, že v obecnej kronike je pomerne často zaznamenaný aktuálny počet obyvateľov obce. To by v tom bol čert, keby v tých dátach nebol žiaden vzostup a pád!

Súťažná úloha

Na vstupe je daná postupnosť kladných celých čísel $p_1, \ldots, p_n$: počty obyvateľov v obci v n rôznych okamihoch. (Údaje sú v chronologickom poradí.)

Rudolf hľadá trojicu indexov $a < b < c$ takú, že platí $p_c < p_a < p_b$.
Slovne: od a-teho merania po b-te prišlo ku vzostupu, a následne od b-teho merania po c-te prišlo k ešte väčšiemu pádu.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n: počet meraní. V druhom riadku vstupu je postupnosť $p_1, \ldots, p_n$.
Na výstup vypíšte ľubovoľnú jednu trojicu indexov s hľadanou vlastnosťou, resp. text „`neexistuje`“, ak taká trojica indexov neexistuje.

Obmedzenia a hodnotenie

Na plný počet bodov je potrebné, aby vaše riešenie na bežnom počítači za sekundu zvládlo ľubovoľný vstup s $n \leq 10\,000\,000$.
Za riešenie použiteľné pre $n \leq 100\,000$ bude približne 7 bodov.
Každé korektné riešenie, bez ohľadu na časovú zložitosť, môže získať aspoň 3 body.

Príklady

vstup

```
5
10 30 50 70 20
```

výstup

```
2 4 5
```

Ešte sú dve iné správne odpovede: „2 3 5“ a „3 4 5“.

vstup

```
5
1010 1030 1050 1070 1060
```

výstup

```
neexistuje
```

A-II-2 Bicykle

Cyklistická Cestná Aliancia Kocúrkovčanov (Cyk-Cak) spúšťa dnes do prevádzky novinku: verejné bicykle. Celý systém funguje jednoducho: po meste je rozmiestnených niekoľko požičovní. V každej požičovni sú v každej chvíli k dispozícii nejaké bicykle. Keď nejaký potrebujete, jednoducho si ho vyzdvihnete,[1] nasadnete naň, odveziete sa k inej požičovni a tam ho zase vrátite.
Aktivista Michal je vášnivý cyklista. Aby podporil aktivitu Cyk-Caku, rozhodol sa, že vlastný bicykel nechá doma a bude do práce chodiť pomocou bicyklov verejných.

[1] Pri vyzdvihnutí si bicykel odomknete pomocou malej šikovnej mobilnej aplikácie. Ale o tom si rozpovieme v nejakej inej úlohe.

Požičovne bicyklov sa nachádzajú na niektorých (nie nutne všetkých) križovatkách v Kocúrkove. Na jednej z križovatiek s požičovňou bicyklov Michal býva, na inej takej križovatke pracuje.

Súťažná úloha

Kocúrkovo si môžeme predstaviť ako graf. Vrcholy grafu sú križovatky (niektoré s požičovňou, iné bez), hrany predstavujú ulice. Pre každú hranu poznáme dve čísla: čas t_p, ktorý Michal potrebuje, aby po nej prešiel pešo, a čas t_b, ktorý potrebuje na bicykli.[2] Vo všeobecnosti nemusí platiť $t_b < t_p$: napríklad takú skratku cez niekoľko poschodí obchodného domu určite prejde rýchlejšie bez bicykla ako s ním.
Na vstupe dostanete mapu Kocúrkova, teda vyššie popísaný graf, a informáciu, kde Michal býva, kde pracuje, a kde stoja jednotlivé požičovne bicyklov.
Napíšte program, ktorý nájde najkratší čas, za ktorý sa vie Michal dostať z domu do práce. Pre jednoduchosť budeme predpokladať, že vypožičanie aj vrátenie bicykla trvá nulový čas. Michal si smie cestou postupne vypožičať a zase vrátiť bicykel ľubovoľne veľakrát, naraz však pochopiteľne smie vždy mať nanajvýš jeden. Požičať si aj vrátiť bicykel môže len v požičovniach (nemôže teda napr. len tak nechať bicykel niekde opretý, keď sa mu už nehodí).

Formát vstupu

V prvom riadku vstupu sú tri prirodzené čísla: počet vrcholov n, počet hrán m a počet požičovní p.
Vrcholy (križovatky) sú očíslované od 1 po n. Michal býva na križovatke 1 a pracuje na križovatke n.
V druhom riadku vstupu je p prirodzených čísel $a_1, \ldots, a_p$: čísla vrcholov, v ktorých sú požičovne bicyklov. Vždy bude platiť $a_1 = 1$ a $a_p = n$.
Zvyšok vstupu tvorí m riadkov, každý z nich popisuje jednu hranu. Popis hrany tvoria štyri prirodzené čísla: čísla križovatiek, ktoré spája, a časy t_p a t_b pre ňu. Môžete predpokladať, že existuje aspoň jedna cesta z vrcholu 1 do vrcholu n.

Formát výstupu

Vypíšte jediný riadok a v ňom jedno prirodzené číslo: najmenší možný celkový čas za ktorý sa Michal vie dostať z domu do práce.

Obmedzenia a hodnotenie

Na plný počet bodov predpokladajte $n, m, p \leq 100\,000$.
Až 8 bodov môže získať riešenie použiteľné pre $n, m \leq 100\,000$ a $p \leq 10$.
Až 6 bodov môže získať riešenie použiteľné pre $n, m, p \leq 100\,000$ za dodatočného predpokladu, že Michal si počas optimálnej cesty požičia bicykel nanajvýš raz. Ak chcete riešiť túto zjednodušenú verziu úlohy, výrazne to v riešení vyznačte.
Rovnako 6 bodov môže získať riešenie použiteľné pre $n, p \leq 1000$.
Každé korektné riešenie bežiace v polynomiálnom čase vie získať 4 body.
Úplne každé korektné riešenie vie získať 3 body.

Príklady

vstup

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

výstup

```
7
```

Najlepší postup pre Michala je ísť z domu pešo cez križovatku 2 na križovatku 3, tam si požičať bicykel a prejsť cez križovatku 5 na križovatku 6.
Takáto cesta trvá 2+2 pešo a potom 1+2 na bicykli.

[2] Pre jednoduchosť predpokladáme, že každú ulicu sa dá prejsť pešo aj na bicykli a že čas prechodu nezávisí od smeru prechodu.

vstup

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

výstup

```
4
```

V každom vrchole je požičovňa, Michal si teda pre každú hranu môže vybrať vhodnejší dopravný prostriedok.

A-II-3 Wi-Fi

Už aj do Kocúrkova dorazil bezdrôtový internet! Teda, zatiaľ len do Rovnej ulice, ale aj to sa počíta.
V Rovnej ulici stojí n domov, sú očíslované od 1 po n v poradí, v ktorom stoja. Dom i sa nachádza x_i metrov od začiatku ulice a je vysoký y_i metrov. Na streche má každý dom svoju anténu. Problém je však v tom, že niektoré dvojice antén na seba nevidia, lebo im je v ceste niektorý iný dom. Takéto dvojice domov potom nevedia priamo komunikovať jeden s druhým.

Súťažná úloha

Pre jednoduchosť si Rovnú ulicu predstavíme ako os x. Každý dom na nej si predstavíme ako zvislú úsečku príslušnej výšky, anténu ako horný bod príslušnej úsečky. Navyše budeme predpokladať, že žiadne tri antény neležia na jednej priamke.
Napíšte program, ktorý načíta popis ulice a ku každému domu A nájde najvzdialenejší iný dom B, s ktorým A vie priamo komunikovať. (Vzdialenosť domov je rovná rozdielu ich x-ových súradníc.)

Formát vstupu a výstupu

V prvom riadku vstupu je n: počet domov.
V i-tom z nasledujúcich n riadkov je súradnica x_i a výška v_i domu číslo i.
Na výstup vypíšte postupne pre každý dom vzdialenosť (meranú vodorovne) od neho k najvzdialenejšiemu inému domu s ktorým vie priamo komunikovať.

Hodnotenie

Vzorové riešenie zvláda vstupy s $n \leq 1\,000\,000$.
Až 6 bodov môžete získať za hocijaké riešenie, ktoré v rozumnom čase spracuje vstupy s $n \leq 5\,000$.
Za akékoľvek funkčné riešenie môžete získať aspoň 3 body.

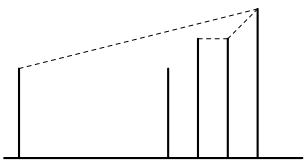
Príklady

vstup

```
5
0 3
50 3
60 4
70 4
80 5
```

výstup

```
80 50 60 10 80
```



Na obrázku je vhodne naškálovaný príklad vstupu. Čiarkované čiary ukazujú, ako komunikujú nasledovné dvojice domov: 1-5, 3-4 a 4-5. Všimnite si, že dom 4 nevie priamo komunikovať ani s domom 1, ani s domom 2.

A-II-4 Stromochod

Jednotlivé podúlohy súťažnej úlohy spolu nesúvisia, môžete ich riešiť v ľubovoľnom poradí.

Súťažná úloha, podúloha A (3 body)

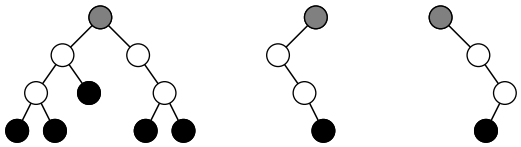
Pred spustením stromochodu vyberieme nejakú podmnožinu vrcholov stromu. Toto spravíme tak, že v každom vybratom vrchole nastavíme značku `vybraty` na `true`, zatiaľ čo v ostatných vrcholoch ju nastavíme na `false`. Množinu vrcholov voláme *nezávislá* ak platí, že žiadne dva vybrané vrcholy nie sú spojené hranou. Inými slovami, ak je nejaký vrchol u otcom vrcholu v, v nezávislej množine nesmú byť oba.
Naprogramujte stromochod tak, aby overil, či je nami vybraná množina vrcholov nezávislá. Odpoveď by stromochod mal uložiť do značky `nezavisla` v koreni.

Súťažná úloha, podúloha B (7 bodov)

Pred spustením stromochodu má každý vrchol stromu značku `vybraty` nastavenú na `false`.
Naprogramujte stromochod tak, aby vybral najväčšiu možnú nezávislú množinu vrcholov.
Teda keď stromochod ukončí svoj program, musia vrcholy, v ktorých je značka `vybraty` nastavená na `true`, tvoriť nezávislú množinu. Navyše musí platiť, že počet vybratých vrcholov je najväčší možný. Ak existuje viacero optimálnych riešení, váš program môže vyrobiť ľubovoľné z nich.

Študijný text: stromochod

V tomto ročníku OI budeme programovať **stromochod** – zariadenie určené na výpočty na binárnych stromoch. Študijný text začneme definíciou toho, čo presne sú binárne stromy.
Graf sa skladá z konečnej množiny **vrcholov** a konečnej množiny **hrán**. Každá hrana spája dva vrcholy. Každé dva vrcholy sú spojené nanajvýš jednou hranou. **Cesta** je postupnosť *navzájom rôznych* vrcholov, v ktorej sú každé dva po sebe idúce vrcholy spojené hranou.
Strom je graf, v ktorom medzi každými dvoma vrcholmi vedie práve jedna cesta. Strom môžeme **zakoreniť** – jeden z jeho vrcholov prehlásiť za **koreň.** Pre každé dva vrcholy spojené hranou následne hovoríme, že ten z nich, čo je bližšie ku koreňu, je **otec** a ten druhý je jeho **syn**. Koreň je jediný vrchol, ktorý nemá otca. Vrcholy, ktoré nemajú žiadnych synov, nazývame **listy** stromu. **Podstrom** je časť stromu, ktorá je vytvorená daným vrcholom, jeho synmi, synmi jeho synov, a tak ďalej až k listom.
Binárny strom je špeciálny zakorenený strom. Každý vrchol v binárnom strome má najviac dvoch synov. Navyše (oproti všeobecnému binárnemu stromu) rozlišujeme **ľavých** a **pravých** synov – každý vrchol má teda nanajvýš jedného ľavého a nanajvýš jedného pravého syna. (Aj ak má vrchol len jedného syna, rozlišujeme, či je ľavý alebo pravý.)



Na obrázku vpravo vidíte niekoľko rôznych binárnych stromov. Stromy kreslíme tak, aby koreň bol úplne na vrchu obrázku. Na každom obrázku je koreň vyplnený sivou a všetky listy sú vyplnené čiernou farbou.
Stromochod slúži na riešenie problémov týkajúcich sa binárnych stromov. Môžeme si ho predstaviť ako počítač, ktorý sa pohybuje po strome. V každom okamihu výpočtu se nachádza v práve jednom z vrcholov stromu.
Pamäť stromochodu je ohraničená: môže si pamätať len konečné množstvo bitov informácie, teda v každom okamihu sa môže nachádzať len v jednom z konečného počtu stavov. Okrem toho stromochod smie zaznamenávať informácie len do navštívených vrcholov. Do každého vrchola sa taktiež zmestí len konečné množstvo informácie a tieto informácie môže stromochod čítať a zapisovať vždy len v tom vrchole, v ktorom sa práve nachádza. Údaje uložené vo vrchole budeme nazývať **značky.**
Stromochod môžeme programovať v ľubovoľnom bežnom programovacom jazyku. Kvôli ohraničeniu na konečný počet stavov môžeme používať len celočíselné premenné s vopred daným rozsahom (napríklad 0..9) a booleovské premenné pre pravdivostné hodnoty. To napríklad znamená, že nie je možné používat polia ani smerníky. Z príkazov sú k dispozícii podmienky, cykly a goto. Procedúry a funkcie je možné používať, ale na ich parametre

sa vzťahujú rovnaké ohraničenia na typy a navyše je zakázaná rekurzia. Pozor, použité rozsahy premenných **nesmú závisieť** od veľkosti stromu.
K aktuálnemu vrcholu (teda k tomu, v ktorom sa práve nachádza) vie stromochod pristupovať pomocou špeciálnej premennej „`V`“. Tá sa správa ako záznam (napr. `record` v Pascale, `struct` v C) ktorý obsahuje značky uložené v aktuálnom vrchole. Množinu značiek si môžete určiť vy pri písaní programu, ale musí ich byť konštantný počet a opäť platí ohraničenie na povolené typy.
Ak chceme stromochod presunúť na ľavého syna, pravého syna, resp. otca aktuálneho vrcholu, zavoláme funkciu `krok_l`, `krok_p`, resp. `krok_o`. Táto funkcia nemá žiadne parametre ani návratovú hodnotu. Ak sa pokúsite presunúť do neexistujúceho vrcholu, program sa zastaví s chybou. Preto sa hodí vopred otestovať, či syn alebo otec aktuálneho vrcholu existuje. Na to slúžia funkcie `ex_l`, `ex_p` a `ex_o`. Tieto pomocné funkcie nemajú žiadne parametre a vracajú booleovskú hodnotu.

Výpočet stromochodu prebieha následovne. Na vstupe dostaneme nejaký strom. Stromochod umiestnime **do jeho koreňa** a všetky značky v celom strome **vynulujeme** (teda číselné premenné nastavíme na nulu a boolovské na false). Výnimkou môžu byť značky, o ktorých je výslovne povedané, že sú súčasťou vstupu. Potom sa rozbehne program. Počas behu programu sa stromochod prechádza po strome, mení svoj stav (svoje lokálne premenné) a údaje zapísané vo vrcholoch stromu (značky). Keď beh programu korektne skončí, jeho výstup si prečítame z určených značiek.

Za čas behu programu na konkrétnom strome budeme považovať počet presunov po ňom. Časová zložitosť programu je funkcia, ktorá pre n vráti maximálny čas behu dotyčného programu na n-vrcholovom strome. Pamäťovú zložitosť neposudzujeme, pretože všetky programy si pamätajú lineárne množstvo informácie.

Príklad

Ukážeme si, ako stromochod naprogramovať tak, aby navštívil všetky vrcholy stromu a do každého umiestnil značku. Strom budeme prechádzať do hĺbky: začneme v koreni, odtiaľ prejdeme do ľavého podstromu. Keď sa z neho vrátime, presunieme sa do pravého podstromu, a nakoniec sa vrátime z celého podstromu von. (Na obrázku stromu takýto prechod zodpovedá prechádzaniu vrcholmi „proti smeru hodinových ručičiek“.)
Pretože nemáme k dispozíci rekurziu, budeme si v každom vrchole pamätať značku s menom `stav`, ktorá bude vyjadrovať, koľkokrát sme už vo vrchole boli. Ak navštívime vrchol prvýkrát, prejdeme na ľavého syna; druhýkrát prejdeme na pravého a po treťom príchode do vrcholu sa už vrátime na jeho otca. Ak ľavý alebo pravý syn neexistujú, budeme se chovať tak, ako by sme sa z nich vrátili okamžite. Program môže vyzerať takto:

```
var zbytocna: 0..47;      { ukazka lokalnej premennej stromochodu }

type vrchol = record      { typ "vrchol" popisuje, ake znacky ukladame do vrcholov }
   bol_som_tu: boolean;   { tuto znacku chceme nastavit na true vo vsetkych vrcholoch }
   stav: 0..3;            { pocitadlo kolkokrat sme uz tento vrchol navstivili }
end;                      { na zaciatku je v kazdom vrchole bol_som_tu = false a stav = 0 }

begin
   zbytocna := 0;
   while zbytocna < 7 do begin { nekonecny cyklus }
      V.bol_som_tu := true;
      V.stav := V.stav + 1;
      case V.stav of
         1: if ex_l then krok_l;
         2: if ex_p then krok_p;
         3: if not ex_o then halt else krok_o;
      end;
   end;
end.
```

Časová zložitosť tohoto programu je lineárna od počtu vrcholov, pretože každý vrchol navštívime najviac trikrát.

TRIDSIATY DRUHÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2017