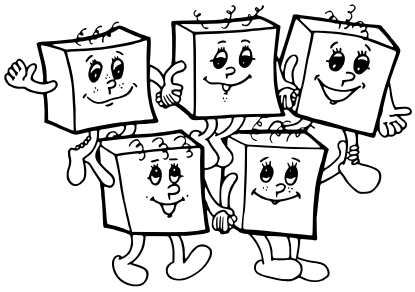
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



dvadsiaty ôsmy ročník

školský rok 2012/2013

zadania celoštátneho kola, deň 1
kategória A

Priebeh celoštátneho kola

Celoštátne kolo 28. ročníka Olympiády v informatike, kategórie A, sa koná v dňoch 20. – 23. marca 2013. Na riešenie úloh prvého, teoretického dňa majú súťažiaci 4,5 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 Pascale alebo C/C++.
- 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 prvého (teoretického) dňa

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 úlohy môžu byť uvedené limity na veľkosť premenných. Tieto môžete použiť na odhad toho, ako dobré vaše riešenie je. Na počítači, ktorý vykoná miliardu inštrukcií za sekundu, vyrieši vzorové riešenie ľubovoľný povolený vstup nanajvýš za niekoľko sekúnd.

A-III-1 Vodovody

Kráľ Hurim bol veľmi spokojný s cestou po Lineárnom kráľovstve – vďaka vašim radám sa dobre napapkal a pribral 4.7 kg. Následne sa rozhodol, že za odmenu všetkým obyvateľom zavedie pitnú vodu.

Súťažná úloha

Pripomeňme si, že Lineárne kráľovstvo je tvorené jedinou cestou. Na tej je n miest, ktoré sú postupne (v poradí, v akom na ceste ležia) očíslované od 1 po n.

Každé mesto i má svoju produkciu vody p_i. Kladná produkcia znamená, že vodné zdroje v meste produkujú viac vody ako mesto stihne spotrebovať. Samozrejme, niektoré mestá môžu mať produkciu zápornú – nemajú dosť pitnej vody.

Ak máme v nejakom meste prebytok vody, môžeme túto prepraviť potrubiami do susedných miest. Každé potrubie musí spájať dve bezprostredne susediace mestá. Pre každú dvojicu miest $(i, i+1)$ poznáme cenu c_i postavenia potrubia medzi nimi. Voda môže postupne putovať viacerými nadväzujúcimi potrubiami a ľubovoľne sa deliť medzi mestá. Kapacitu potrubí považujte za nekonečnú.

Vašou úlohou je nájsť *najlacnejšiu* množinu potrubí takú, že keď ich postavíme, každé mesto dostane dostatok vody. Teda pre každú skupinu miest prepojených potrubiami musí platiť, že súčet produkcie miest v nej je nezáporný.

Formát vstupu a výstupu

V prvom riadku vstupu je počet miest n.

V druhom riadku je n celých čísel $p_1, \ldots, p_n$: pre každé mesto jeho produkcia. Súčet všetkých produkcií bude vždy nezáporný. (Vždy teda bude existovať nejaké prípustné riešenie.)

V treťom riadku je $n-1$ kladných celých čísel $c_1, \ldots, c_{n-1}$: pre každú dvojicu susedných miest cena ich prepojenia.

Môžete predpokladať, že všetky hodnoty p_i a c_i aj ich súčty sa pohodlne zmestia do bežných celočíselných premenných. (Formálne môžete napr. predpokladať, že $\sum_i |p_i|$ aj $\sum_i c_i$ sú menšie ako 10^{18}.)

Na výstup vypíšte jediný riadok a v ňom najmenšiu celkovú cenu, ktorú treba zaplatiť za postavenie potrubí.

Hodnotenie

- Najviac 2 body môžete získať za riešenie, ktoré efektívne vyrieši vstupy s $n \leq 20$.
- Najviac 6 bodov môžete získať za riešenie, ktoré efektívne vyrieši vstupy s $n \leq 1000$.
- Plných 10 bodov môžete získať za riešenie, ktoré efektívne vyrieši vstupy s $n \leq 1\,000\,000$.

Príklad

vstup

```
5
-2 10 -2 -5 5
2 2 3 4
```

výstup

```
7
```

Najlacnejšie je postaviť prvé tri potrubia, čím prepojíme mestá 1, 2, 3 a 4. Prebytok v meste 2 prerozdelíme ostatným trom mestám (a ešte nám trocha ostane). Mesto 5, ktoré ostalo izolované, má vody dosť. Cena tohto riešenia je 2+2+3 = 7.

Iným korektným (ale drahším) riešením je postaviť potrubia 1-2, 2-3 a 4-5.

A-III-2 Dievka na vydaj

Princezná Baška už je rúča deva. V poslednom čase sa ale zbláznila do seriálu My Little Pony. To sa samozrejme u dospelej následníčky trónu nehodí – ale s Baškou nebola reč. Až keď nezabrali prosby, sľuby ani vyhrážky, pochopil jej kráľovský otec, že tento problém má jediné riešenie: nájdu jej vhodného ženícha, vydajú ju za neho – no a nech si tie prekliate poníky rieši on!

Ako áno, ako nie, keď sa rozkríklo, že budú Bašku vydávať, prihlásilo sa obrovské, prakticky nekonečné množstvo pytačov. Kráľ by chcel Baške vybrať nejakého slušného manžela, nech dievka nehladuje. Je si ale vedomý, že medzi najbohatších pytačov by tiež patriť nemusel – tí sú totiž vplyvní, majú veľké armády a bohvie, čo by spravili po odhalení Baškinej mánie. Kráľ sa preto rozhodol, že Bašku vydá za k-teho najbohatšieho pytača.

Je známe, že všetci pytači majú navzájom rôzne bohatstvo, takže k-ty najbohatší pytač je jednoznačne určený. Ktorý to ale je? Ako ho v tom dave nájsť? Kráľ zavolal svojich n radcov (pričom n je omnoho menšie ako k), všetkých pytačov rozdelil do n skupín a každú skupinu dal na starosť jednému z radcov.

Každý radca si o pytačoch zo svojej skupiny zistil, kto je ako bohatý, a usporiadal svoju skupinu podľa bohatstva, začínajúc najbohatším pytačom v nej. Bohužiaľ, v tejto situácii ešte stále nebolo jasné, ako nájsť k-teho najbohatšieho zo všetkých pytačov. A čo je horšie, kráľ už nemá žiadnych ďalších radcov, ktorí by mu s tým pomohli. Bude to teda na vás.

Súťažná úloha

Napíšte program, ktorý bude fungovať nasledovne:

Na začiatku načíta počet radcov n a číslo k udávajúce, koľkého najbohatšieho pytača hľadáme. Radcovia sú očíslovaní od 1 po n. V skupine každého radcu sú pytači očíslovaní vzostupne začínajúc od 1 (pričom číslo 1 má najbohatší v skupine). V každej skupine je aspoň k pytačov.

Následne bude váš program klásť otázky radcom. Otázku položíte zavolaním funkcie `bohatsi(r1,i1,r2,i2)`. Tá v konštantnom čase vráti `true` (pravda) alebo `false` (nepravda) podľa toho, či je i_1-ty pytač v skupine radcu r_1 bohatší ako i_2-ty pytač v skupine radcu r_2.

Keď už si je váš program istý, vypíše, ktorý pytač je ten, ktorého hľadáme, a skončí. Presnejšie, váš program má vypísať dve čísla r a i: číslo radcu a poradové číslo pytača v jeho skupine.

Hodnotenie

- Najviac 2 body môžete získať za riešenie, ktoré efektívne vyrieši vstupy s $n = 2$ a $k \leq 100\,000$.
- Najviac 6 bodov môžete získať za riešenie, ktoré efektívne vyrieši vstupy s $n \leq 10\,000$ a $k \leq 100\,000$.
- Podľa konkrétnej časovej zložitosti môžete získať 7 až 10 bodov za riešenie, ktoré efektívne vyrieši vstupy s $k \leq 10^{18}$ a čo najväčším n.

Príklad

Majme $n = 3$ skupiny pytačov a nech $k = 3$, teda hľadáme tretieho najbohatšieho medzi nimi. Pre názornosť si ukážeme aj konkrétne skupiny pytačov a ich bohatstvo v eurách. (Tieto hodnoty poznajú radcovia, váš program k nim ale nemá prístup.)

skupina 1:	20, 15, 10, ...
skupina 2:	14, 7, 4, ...
skupina 3:	18, 16, 8, ...

Ak by sme v našom programe zavolali funkciu `bohatsi(1,2,3,3)`, dostali by sme odpoveď `true`. Totiž u radcu 1 má pytač 2 bohatstvo 15, u radcu 3 má pytač 3 bohatstvo 8, no a $15 > 8$.

Ak by sme zavolali funkciu `bohatsi(2,3,3,1)`, dostali by sme odpoveď `false`, lebo $4 < 18$.

Váš program by pre tento vstup postupnými volaniami funkcie `bohatsi` mal zistiť, že tretím najbohatším zo všetkých pytačov je v skupine radcu 3 pytač číslo 2 (ten s majetkom 16). Správnym výstupom programu by teda boli čísla „`3 2`“.

A-III-3 Log-space výpočty

Študijný text k tejto úlohe je uvedený nižšie. Je identický so študijným textom z domáceho kola.
Pripomíname, že pri riešení súťažných úloh nám ***nezáleží*** *na časovej, ale iba na pamäťovej zložitosti programov.*
Nezabudnite ako súčasť riešenia ***zdôvodniť****, že vami navrhnutý program je naozaj log-space.*

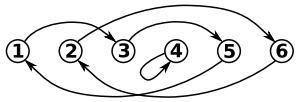
Podúloha A (4 body)

Permutáciou čísel 1 až n voláme takú postupnosť $P[1], \ldots, P[n]$, v ktorej sa každé z čísel 1 až n vyskytuje práve raz. Na permutáciu sa môžeme dívať ako na orientovaný graf, ktorého vrcholmi sú čísla 1 až n a v ktorom máme pre každé i orientovanú hranu z vrcholu i do vrcholu $P[i]$.

Z každého vrcholu teda *vychádza* práve jedna hrana. No a keďže všetky hodnoty $P[i]$ sú navzájom rôzne, znamená to, že do každého vrcholu *vchádza* práve jedna hrana. Náš graf sa teda nutne skladá z niekoľkých *cyklov*. (Niektoré z týchto cyklov môžu byť *slučky* tvorené jedinou hranou z vrcholu i rovno do neho samého.)

Napíšte log-space program, ktorý pre danú permutáciu zistí, koľko má cyklov. Vstupom je celočíselná premenná n a pole $P[1..n]$. Výstupom je celočíselná premenná c.

Príklad. Nech $n = 6$ a nech $P[1..6] = (3, 6, 5, 4, 1, 2)$. Táto permutácia má tri cykly: $(1, 3, 5)$, $(2, 6)$ a (4). Správnym výstupom je teda $c = 3$. Na nasledujúcom obrázku je graf zodpovedajúci tejto permutácii.



Podúloha B (6 bodov)

Strom je súvislý neorientovaný graf neobsahujúci kružnicu. Napíšte log-space program, ktorý pre zadaný strom a dva jeho vrcholy u a v vypočíta vzdialenosť dotyčných dvoch vrcholov – teda najmenšie číslo k také, že na to, aby sme sa dostali z u do v, stačí postupne prejsť po k hranách daného stromu.

Vstup tvoria tri celočíselné premenné (n, u, v) a dve polia (A, B). Premenná n udáva počet vrcholov stromu. Tie sú očíslované od 1 po n. Premenné u a v obsahujú dve rôzne čísla vrcholov.

Strom s n vrcholmi má presne $n - 1$ hrán. Tieto popisujú polia $A[1..n-1]$ a $B[1..n-1]$: pre každé i máme v strome hranu spájajúcu vrcholy $A[i]$ a $B[i]$.

Výstup tvorí jediná celočíselná premenná d, do ktorej máte uložiť vzdialenosť vrcholov u a v.

Príklad. Nech $n = 5$, teda máme strom s 5 vrcholmi a 4 hranami. Nech ďalej $A[1 \ldots 4] = (1, 4, 3, 3)$ a $B[1 \ldots 4] = (4, 2, 4, 5)$, teda v našom strome máme hrany 1-4, 4-2, 3-4 a 3-5.

Pre $u = 3$ a $v = 5$ by bolo správnym výstupom $d = 1$, keďže medzi vrcholmi 3 a 5 máme priamu hranu.

Pre $u = 5$ a $v = 1$ by bolo správnym výstupom $d = 3$. Najkratšia cesta z vrcholu 5 do vrcholu 1 vedie cez vrcholy 3 a 4.

Študijný text

Ak poznáme viac algoritmov, ktoré riešia tú istú úlohu, väčšinou za lepší považujeme ten, ktorý má menšiu časovú zložitosť. Pamäťovú zložitosť zväčša používame len ako dodatočné kritérium (s výnimkou situácií, kedy sú pamäťové nároky algoritmu absurdne vysoké). V úlohách, ku ktorým patrí tento študijný text, bude situácia presne opačná: zaujímať nás bude takmer výlučne pamäťová zložitosť programu. Tá bude musieť byť veľmi malá.

Presnejšie, budeme písať *log-space programy*. Pôjde o obyčajné programy vo vašom obľúbenom bežnom programovacom jazyku, len budú navyše musieť spĺňať nasledujúce obmedzenia:

- Každá použitá premenná musí byť jednoduchého *celočíselného* typu (napr. `int` v C++ či `longint` v Pascale).
- U celočíselných premenných nám nebude záležať na presnom rozsahu, nemusí vás teda napr. trápiť, či sú 32-bitové alebo 64-bitové. Namiesto toho si povolený rozsah hodnôt, ktoré sa do premenných zmestia, definujeme nasledovne: Nech n je veľkosť vstupu (teda napr. počet čísel na vstupe). Potom do premenných môžeme ukladať len hodnoty ktoré sú *polynomiálne veľké* v závislosti od n.

  Môžeme mať teda napr. premennú, ktorá bude nadobúdať hodnoty $-n \dots n$, hodnoty $-3n^5 \dots 3n^5$, čí len hodnoty $-4 \dots 7$. Nesmieme však už mať premennú nadobúdajúcu napr. hodnoty $0 \dots 2^n$.
- Pre pohodlie budeme aj typy `char` a `boolean` (resp. `bool` v C++) považovať za celočíselné, a teda povolené.
- Žiadne iné typy premenných (teda napr. ani polia alebo ukazovatele) nie sú povolené.
- Výnimku z predchádzajúcich pravidiel tvoria vstup a výstup. Vstup programu bude dostupný v nejakých špeciálnych premenných (zväčša to budú polia), ktoré môže váš program *len čítať*. A podobne výstup bude váš program ukladať do iných špeciálnych premenných, do ktorých smie *len zapisovať*.

  (Špeciálne upozorňujeme, že nesmiete výstupnú premennú zväčšiť o danú hodnotu. Teda na ňu nesmiete napr. v Pascale použiť príkaz `inc`, v C++ operátor `++` či `+=`. Všetky tieto zmeny totiž vyžadujú nie len zápis novej hodnoty, ale najskôr aj prečítanie starej – lenže výstupnú premennú čítať nesmiete.)

  Čísla na vstupe vždy budú mať veľkosť polynomiálnu od veľkosti vstupu, a teda každé z nich sa zmestí do pracovnej premennej.
- Vaše programy nesmú používať rekurziu.

Príklad 1. Ukážeme si log-space program, ktorý nájde maximum v poli čísel.

Listing programu (Pascal)

```
var n: integer;                  { vstupná premenná: počet čísel }
    A: array [1..n] of integer;  { vstupná premenná: pole čísel }
    m: integer;                  { výstupná premenná: pozícia maxima }
    i, j: integer;               { pracovné premenné }
begin
    j := 1;
    for i := 2 to n do
        if A[i] > A[j] then j := i;
    m := j;
end;
```

Jediné dve premenné sú `i` a `j` a zjavne nadobúdajú len hodnoty z rozsahu $1 \dots n$. Tým sú teda splnené všetky potrebné podmienky a teda skutočne ide o log-space program.

Príklad 2. Nasledujúci log-space program nájde v poli čísel to, ktoré sa tam vyskytuje najčastejšie.

Listing programu (Pascal)

```
var n: integer;                  { vstupná premenná: počet čísel }
    A: array [1..n] of integer;  { vstupná premenná: pole čísel }
    m: integer;                  { výstupná premenná: jedna pozícia najčastejšieho }
    i, j, c, cmax: integer;      { pracovné premenné }
begin
    cmax := 0;
    for i := 1 to n do begin
        { spočítame, koľkokrát sa vyskytuje A[i] }
        c := 0;
        for j := 1 to n do
            if A[j] = A[i] then c := c+1;
        { je to viac ako doterajšie maximum? }
        if c > cmax then begin
            cmax := c;
            m := c;
        end;
    end;
end;
```

Opäť si ľahko rozmyslíme, že hodnoty pracovných premenných nikdy neprekročia n. Taktiež všetky ostatné požiadavky na log-space program sú dodržané.

Prečo práve takéto programy?

Isto ste si už položili otázku, prečo takéto programy nazývame log-space a načo je vlastne dobré sa nimi zaoberať.

Najpresnejší spôsob merania pamäťovej zložitosti daného programu dostaneme vtedy, keď zistíme počet *bitov pamäte*, ktorú potrebuje v závislosti od veľkosti vstupu.

Existuje síce pár veľmi jednoduchých problémov, ktoré vieme riešiť napríklad len s troma bitmi pamäte, veľa ich ale nie je a nebudú nás príliš zaujímať. My sa sústredíme na programy, ktoré vstupné dáta dostanú v nejakom n-prvkovom poli. Na to, aby sme vôbec mohli k prvkom takéhoto poľa rozumne pristupovať, potrebujeme do neho vedieť *indexovať*. A na to treba mať premennú, ktorá môže nadobudnúť n rôznych hodnôt.

Jeden bit pamäte má dva rôzne stavy: 0 alebo 1. Ak máme k-bitovú premennú, tá môže nadobúdať 2^k rôznych stavov – nezávisle pre každý z k bitov máme dve možnosti. Tieto stavy si my potom rôzne interpretujeme: raz ako znaky, inokedy ako čísla, a podobne. Napríklad taká 8-bitová premenná môže mať 256 rôznych stavov. Niekedy tieto stavy môžeme považovať za čísla od 0 do 255, inokedy za čísla od -100 do 155, a ešte inokedy za znaky v 8-bitovom ASCII kódovaní.

A tú istú úvahu môžeme spraviť aj opačne. Ak potrebujeme premennú, ktorá vie mať n rôznych hodnôt, potrebujeme, aby pre jej počet bitov k platila nerovnosť $2^k \geq n$. Inými slovami, najmenšie vyhovujúce k je $\lceil \log_2 n \rceil$ (t.j. horná celá časť dvojkového logaritmu čísla n).

Dostávame teda nasledovný záver: na prácu s n-prvkovým poľom určite treba aspoň rádovo $\log_2 n$ bitov pamäte. Toto je teda v istom zmysle najmenšia prakticky zaujímavá pamäťová zložitosť programov.

No a práve takúto (asymptotickú) pamäťovú zložitosť budú mať aj všetky programy, ktoré budete písať vo svojich riešeniach. Totiž naše obmedzenie na počet a veľkosť premenných, ktoré smiete používať, zaručuje, že celkový počet bitov pamäte, ktoré použijete, bude nanajvýš rádovo logaritmický od n. Odtiaľ teda pochádza názov „log-space“ – sú to programy, ktoré používajú $O(\log n)$ bitov pamäte.

(Príklad: Ak použijete 3 premenné, z ktorých každá môže nadobúdať hodnoty od 1 po n^5, bude váš program používať $3\lceil \log_2 n^5 \rceil \approx 15 \log_2 n$ bitov pamäte.)

O zákaze používania polí a rekurzie

Teoreticky by sa do logaritmického počtu bitov pamäte nejaké maličké polia zmestili – ale chcelo by to, aby súčet veľkostí všetkých prvkov bol nanajvýš logaritmický. Teda napríklad by sme mohli mať pole obsahujúce $2 \log n$ čísel z rozsahu 0 až 7. Alebo pole obsahujúce 3 čísla z rozsahu 1 až n^2. Nič z toho vám zrejme pri súťažných úlohách nepomôže, preto sme pre jednoduchosť polia zakázali úplne. (Namiesto poľa obsahujúceho 3 čísla predsa vždy môžete použiť 3 vhodne pomenované premenné.)

Druhou konštrukciou, ktorú sme museli zakázať, je rekurzia. Treba si totiž uvedomiť, že pamäť počas behu programu nepoužívame len na premenné. Vždy, keď v programe zavoláme nejakú funkciu, musí sa vyrobiť (na tzv. zásobníku) záznam, kde si okrem iného program zapamätá správnu návratovú adresu a tiež hodnoty parametrov, s ktorými sme danú funkciu zavolali. Taktiež lokálne premenné pre danú funkciu niekam treba uložiť. A tu práve rekurzia začína robiť problémy.

Totiž ak máme program pozostávajúci z viacerých funkcií, ale bez použitia rekurzie (t.j. každá funkcia volá len tie, ktoré boli v programe uvedené pred ňou), vždy ho vieme prerobiť („dosadením“ celého tela funkcie namiesto každého jej volania) na program, ktorý počíta to isté, ale už nepoužíva funkčné volania. Môžete si rozmyslieť, že sa pri tejto zmene nijak výrazne nezmenia pamäťové nároky programu.

Akonáhle však povolíme rekurziu (funkciu, ktorá volá sama seba, prípadne viacero funkcií, ktoré sa vedia volať navzájom), mohlo by sa stať, že pri behu programu nám bude postupne vznikať viac a viac lokálnych premenných a celková pamäť ľahko narastie nad logaritmickú. Preto radšej rekurziu úplne zakazujeme. Rovnako ako pri poliach, aj tu by malo platiť, že vám toto obmedzenie nijak zásadne nesťaží riešenie súťažných úloh.

DVADSIATY ÔSMY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Vladimír Boža, Michal Forišek, Ján Hozza, Marek Špano
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2013