Slovenská komisia Matematickej olympiády
FPEDaS ŽU, Veľký diel, 010 26 Žilina

MATEMATICKÁ OLYMPIÁDA NA STREDNÝCH ŠKOLÁCH

55. ročník, školský rok 2005/2006

Riešenia úloh 2. kola kategórie P

P–II–1

Na úvod si dohodnime nasledujúce značenie:

- d_i je deň, kedy sa hrá i-ty zápas, navyše definujeme $d_0 = 0$,
- b_i je počet bodov, ktoré získame za i-ty zápas (0, 1 alebo 3).

Všimnime si nejaké (ľubovoľné) optimálne riešenie problému. Ukážeme, že existuje rovnako dobré riešenie, kde je postupnosť počtov získaných bodov neklesajúca – inými slovami, najskôr niekoľko zápasov prehráme, potom niekoľko remizujeme a všetky ostatné vyhráme.

Toto vieme dosiahnuť postupnými výmenami výsledkov zápasov. Ak pre nejaké i platí $b_{i-1} > 0$ a $b_i = 0$, tak ich môžeme vymeniť – namiesto $(i-1)$-ého zápasu vyhráme/remizujeme až i-ty, síl na to máme určite dosť a po i-tom zápase budeme na tom presne rovnako ako v pôvodnom riešení. Podobne, výsledky môžeme vymeniť aj ak $b_{i-1} = 3$ a $b_i = 1$. Tu rozlíšime dva prípady: Ak $R < V$, môžeme ich vymeniť z podobného dôvodu ako predtým, prvý zápas namiesto výhry len remizujeme (čo nás stojí nanajvýš toľko isto síl), druhý zápas vyhráme, celková „spotreba síl“ za tieto dva zápasy je rovnaká. Ak $R \geq V$, táto situácia nenastane, lebo v optimálnom riešení sa nám neoplatí remizovať.

Zamyslime sa teraz nad tým, ako overiť, či vieme vyhrať posledných niekoľko zápasov.

Označme v_i minimálnu veľkosť sily, ktorá nám tesne pred i-tym zápasom stačí na to, aby sme od tohto okamihu do konca sezóny vyhrali všetky zápasy. Hodnoty v_i spočítame od konca. Zjavne $v_N = V$. Ak vieme hodnotu v_{i+1}, hodnotu v_i spočítame nasledovne: $v_i = V + \max(0, v_{i+1} - (d_{i+1} - d_i))$. (Potrebujeme určite aspoň V na vyhranie tohto zápasu. Pred nasledujúcim zápasom musíme mať v_{i+1} sily. Do nasledujúceho zápasu jej vieme načerpať $d_{i+1} - d_i$, prípadný zvyšok si teda musíme priniesť spred i-teho zápasu.)

Na druhej strane, ľahko vieme určiť, že pred i-tym zápasom vieme mať najviac d_i sily. Vyhrať všetko od i-teho zápasu ďalej teda vieme práve vtedy, ak $d_i \geq v_i$.

Jemným zovšeobecnením tejto úvahy dostávame prvé pomerne efektívne riešenie úlohy Vyskúšame všetky možnosti pre počet výhier a. Pre každú možnosť vyššie uvedeným postupom overíme, či je prípustná. Ak áno, dopočítame, koľko najviac predchádzajúcich zápasov vieme remizovať. (Toto spravíme presne rovnakým postupom, začneme od $(N - a + 1)$-ého zápasu, pred ktorým máme mať v_{N-a+1} sily, pri výpočte sily potrebnej pred predchádzajúcimi zápasmi použijeme ten istý vzorec, len namiesto V v ňom bude R, keďže tieto zápasy chceme remizovať.) Takto získaný počet remizovaných zápasov označme b.

Vyskúšať jedno možné a vieme teda v lineárnom čase od N. Spomedzi všetkých rádovo N skúšaných možností si vyberieme samozrejme tú, kde je získaný počet bodov (teda hodnota $3a + b$) najväčší možný. Vyššie popísané riešenie má zjavne časovú zložitosť $O(N^2)$.

Teraz ukážeme, ako túto úlohu riešiť v čase lineárnom od počtu zápasov.

V prvom rade ošetríme situáciu, kedy $R \geq V$. Vtedy sa nám neoplatí remizovať, preto len ideme od konca a nájdeme maximálny možný počet výhier. V nasledujúcom texte predpokladáme, že $R < V$.

Myšlienka je nasledovná: budeme postupne zvyšovať počet vyhratých zápasov a zakaždým (šikovnejšie ako v minulom riešení) upravíme maximálny počet zápasov, ktoré predtým zvládame remizovať.

Na úvod teda nájdeme riešenie, v ktorom žiaden zápas nevyhráme, ale čo najviac posledných zápasov remizujeme, nech je ich b. Toto riešenie vieme vyššie popísaným postupom nájsť v lineárnom čase.

Teraz ideme postupne zväčšovať počet vyhratých zápasov. Počas tejto fázy si budeme pamätať prvý remizovaný zápas x a posledný remizovaný zápas y. Na začiatku je teda $x = N - b + 1$ a $y = N$.

Od tohto okamihu až do chvíle, kedy sa nám minú remizované zápasy a už žiadny nevieme vyhrať, budeme opakovať nasledujúci postup:

Všimnime si, že pri aktuálnom riešení bude naše mužstvo pred y-tym zápasom mať $d_y - R(y - x)$ sily. Ak je táto hodnota $\geq v_y$, môžeme si dovoliť y-ty zápas vyhrať bez toho, aby sme museli zmenšovať počet „bodovaných“ zápasov. V opačnom prípade musíme zmenšiť počet „bodovaných“ zápasov, a teda x-tý zápas namiesto remízy prehráme.

Malo by byť zjavné, že vyššie uvedeným postupom v okamihu, keď sme práve „vyrobili“ a-tu výhru, máme k nej najväčší možný počet remíz. Určite teda optimálne riešenie nepreskočíme.

Zostáva ukázať, že časová zložitosť tohto riešenia je naozaj lineárna. Prvú fázu (spočítanie hodnôt v_i a maximálneho počtu remíz bez výhier) vieme ľahko spraviť v lineárnom čase. Každý krok druhej fázy buď zväčší y, alebo zmenší x a prestaneme, keď $x < y$, preto je týchto krokov najviac N. Každý z nich vieme spraviť v konštantnom čase, preto celá druhá fáza beží v čase lineárnom od N, č.b.t.d.

Program uvedený nižšie implementuje tú istú myšlienku trochu ináč. Rovnako ako v prvom prezentovanom riešení začne zostrojením optimálneho riešenia, v ktorom len remizujeme. Pre každý interval si pamätá, koľko sily získanej počas neho nevyužívame. Túto silu potom používa na prerábanie remizovaných zápasov na vyhraté. Pritom platí, že vždy, keď má na výber, použije tú dostupnú silu, ktorá vznikne najneskôr – tým určite nič nepokazí. Aj implementáciou tejto myšlienky vieme dostať riešenie bežiace v lineárnom čase.

Poznámka na záver: Existujú aj iné, väčšinou pomalšie riešenia, ktoré sú založené na dynamickom programovaní. Napríklad môžeme postupne pre každý zápas (od 1 do N) a každý počet bodov (od 0 po $3N$) zistiť, či je po danom zápase možné mať daný počet bodov, a ak áno, koľko najviac sily vieme v takom prípade po jeho skončení mať. Túto informáciu spočítame tak, že vyskúšame všetky tri možnosti, ako daný zápas dopadol, a pre každú sa pozrieme na už vypočítaný výsledok pre o jedno menší počet zápasov a zodpovedajúci počet bodov. Časová aj pamäťová zložitosť tohto riešenia sú $O(N^2)$, pri šikovnej implementácii si vystačíme aj s pamäťou $O(N)$.

```
var dni:array[1..10000] of longint;
    sila:array[1..10000] of longint;
    V,R,vysl:longint;
    N,i,pV,pR:integer;

{Najde maximalny pocet najpravejsich zapasov, ak sila potrebna na jeden zapas je ,kolko'}
function pouzi(kolko:integer):integer;
var i,j,potrebujem:integer;
```

```
begin
  i:=N; {Aktualny zapas}
  j:=N; {Odkial cerpame silu}
  while (j>=1) and (i>=1) do begin
    if j>i then {Silu mozeme cerpat len zlava}
      j:=i;

    {Odcerpeme potrebnu silu pre i-ty zapas}
    potrebujem:=kolko;
    while (j>=1) and (sila[j]<potrebujem) do begin
      dec(potrebujem,sila[j]);
      sila[j]:=0;
      dec(j);
    end;

    if j>=1 then begin
      dec(sila[j],potrebujem);
      dec(i);
    end else begin
      {Co zvysilo, si odlozime na zaciatok. Urcite to pouzijeme
     len na prerobenie zapasu s cislom >=i, takze je to ok.}
      sila[1]:=kolko-potrebujem;
    end;
  end;
  pouzi:=N-i;
end;

begin
  readln(V,R,N);
  {Jednoduchy sposob zbavenia sa pripadu V<R}
  if V<R then
    R:=V;
  for i:=1 to N do
    read(dni[i]);

  sila[1]:=dni[1];
  for i:=2 to N do
    sila[i]:=dni[i]-dni[i-1];

  {1. faza}
  pR:=pouzi(R);

  {2. faza}
  pV:=pouzi(V-R);

  if pV>pR then begin
    {Ak sa nam podarilo ,,prerobit'' viac remiz, ako sme mali}
    pV:=pR;
    pR:=0;
```

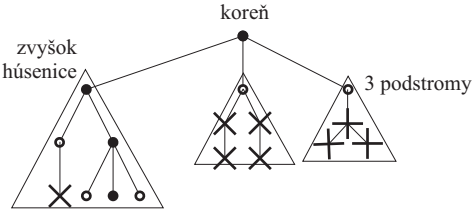
```
  end else
    pR:=pR-pV;

  vysl:=1*pR + 3*pV;

  {3. faza}
  while pR>1 do begin
    {Zrusime nalavejsiu remizu}
    dec(pR);
    sila[1]:=sila[1]+R;
    if (sila[1]>=V-R) then begin
      {Je dost sily na prerobenie}
      sila[1]:=sila[1]-(V-R);
      dec(pR);
      inc(pV);
      if 1*pR + 3*pV > vysl then
        vysl:=1*pR + 3*pV;
    end;
  end;

  writeln(vysl);
end.
```

P-II-2

Všimnime si najskôr, že počet vrcholov, ktoré musíme odstrániť, vieme vypočítať ako (počet vrcholov stromu) mínus (počet vrcholov výslednej húsenice). Môžeme teda namiesto najmenšieho počtu vrcholov na odstránenie hľadať najväčšiu húsenicu nachádzajúcu sa v strome.

Úlohu si ešte trochu zjednodušme: nebudeme hľadať hocijakú húsenicu, ale takú, ktorej telo končí v koreni (koreň je ten vrchol, z ktorého sa začíname prechádzať pri popise stromu). Pripomeňme si však, že telo húsenice je cesta a že v strome medzi každými dvoma vrcholmi vedie práve jedna cesta. Zvyšok tela potom zjavne musí byť celý obsiahnutý v niektorom podstrome. Nech sa tento zvyšok tela nachádza v ktoromkoľvek podstrome, počet vrcholov tvoriacich húsenicu vieme vyjadriť takto:

počet vrcholov húsenice
= 1 (koreň)
+ (počet podstromov − 1)
+ zvyšok húsenice



Vidíme, že ak chceme nájsť najväčšiu húsenicu, ktorej telo končí v koreni, potrebujeme nájsť čo najväčší zvyšok húsenice – t.j. čo najväčšiu húsenicu z niektorého podstromu. To nás privádza k rekurzívnemu riešeniu: Pre každý vrchol stromu (resp. podstrom s koreňom v tomto vrchole) budeme počítať veľkosť (t.j. počet vrcholov) najväčšej húsenice v podstrome, ktorá končí v tomto vrchole. Ak je vrchol list, potom tento vrchol tvorí húsenicu (najväčšiu), ktorá sa

skladá z jedného vrcholu. Pre ostatné vrcholy túto hodnotu najskôr rekurzívne vypočítame pre všetky podstromy – nájdeme v nich najväčšie húsenice. Z týchto hodnôt nájdeme maximum a (podľa vyššie uvedeného vzorca) pripočítame počet podstromov.

Všimnite si, že každý vrchol spracujeme v čase úmernom jeho stupňu, teda počtu vychádzajúcich hrán. Keďže strom má hrán $N-1$, je celkový čas potrebný na tento výpočet lineárny od N.

Ako však ukazujú už príklady v zadaní, telo húsenice v žiadnom prípade nemusí končiť práve v nami zvolenom koreni. Aby sme teda našli naozaj najväčšiu, musíme postupne ako koreň (teda jeden koniec tela) vyskúšať všetky vrcholy.[1] Tak dostávame algoritmus s časovou zložitosťou $O(N^2)$, t.j. kvadratický od počtu vrcholov stromu.

Tento algoritmus môžeme vylepšiť, ak si všimneme, ako vyzerá ľubovoľná cesta v zakorenenom strome: najskôr ide niekoľko (aj nula) vrcholov nahor (smerom ku koreňu) a potom klesá (smerom od koreňa). Každá cesta má teda práve jeden „najvyšší“ vrchol.

Upravíme teraz náš pôvodný algoritmus tak, aby pri jednom prechode stromom určite našiel najdlhšiu húsenicu. Okrem pôvodne počítanej informácie (veľkosť najväčšej húsenice idúcej v danom podstrome z jeho koreňa „dodola“) budeme pre každý vrchol počítať aj veľkosť najväčšej húsenice, ktorej telo má v danom vrchole najvyšší bod cesty. Z týchto hodnôt nám už iba stačí nájsť maximum.

Telo húsenice sa teda skladá z najvyššieho vrcholu a dvoch ciest v dvoch podstromoch. Veľkosť celej húsenice teda vieme vypočítať ako

1	za tento vrchol
+	veľkosť húsenice, ktorej telo vedie dole jedným podstromom
+	veľkosť húsenice, ktorej telo vedie dole iným podstromom
+	počet podstromov − 2 (korene ostatných podstromov tvoria nožičky)
+	1, ak tento vrchol nie je koreň (aj otec tohto vrchola je nožička)

Opäť vidíme, že najväčšiu húsenicu dostaneme vtedy, keď vyberieme dve najväčšie húsenice z dvoch rôznych podstromov. Keď už vieme pre každý podstrom veľkosť najväčšej húsenice, ktorá ide z jeho koreňa dodola, dve najväčšie z týchto húseníc vieme nájsť v čase lineárnom od počtu podstromov, a teda lineárnom od stupňa spracúvaného vrcholu.

Zopakujme si to celé. Pre každý vrchol budeme počítať:

- veľkosť najväčšej húsenice, ktorej telo končí v danom vrchole a celá sa nachádza v jeho podstrome
- veľkosť najväčšej húsenice takej, že daný vrchol je najvyšší vrchol jej tela.

Obe informácie pre konkrétny vrchol vieme vypočítať z rovnakých údajov o podstromoch vrcholu. Celý postup vieme realizovať v čase $O(N)$ – lineárnom od veľkosti stromu (stačí ho totiž raz prejsť). Strom prechádzame priamo počas čítania reťazca núl a jednotiek, ktorým je zadaný.

```
#include <cstdio>
#include <cstring>

char s[10000];  // popis stromu
int l;          // dlzka popisu
int i=0;        // pozicia v strome
int max=0;      // najvacsia dosial najdena husenica
```

[1] Resp. stačí všetky listy, ale tým si príliš nepomôžeme.

```
int dfs() {
// vrati velkost max. husenice, ktora konci v tomto vrchole
  int pp=0; // pocet podstromov;
  int h=0;  // velkost najvacsej a druhej najvacsej
  int h2=0; //  husenice konciacej v nejakom podstrome

  // ak je to list, h = 1
  if (s[i] == '0') { ++i; return 1; }

  while (s[i++] == '1') { // pre vsetky podstromy
    ++pp;                  // pocitame pocet podstromov
    int r = dfs();         // rekurzivne spocitame
                           // max. husenicu podstromu
    if (r > h) { h2=h; h=r; } // hladame dve najvacsie
    else if (r > h2) h2=r;    // husenice v podstromoch
  }

  // velkost najvacsej husenice,
  // ktorej telo konci v tomto vrchole
  int k = h + pp-1 + 1;
  // velkost najvacsej husenice, ktorej telo
  // ma v tomto vrchole svoj najvyssi bod
  int v;
  if (pp == 1) v = k;
          else { v = h + h2 + pp-2 + 1;
                 // ak nie sme v koreni, tak + 1
                 if (i < l) v = v + 1; }

  if (max < v) max = v;
  return k;
}

int main () {
  scanf („%s“, &s);
  l = strlen(s);
  dfs(); // dfs() vypocita max -- max. pocet vrcholov,
         // ktore tvoria husenicu
  if (l == 0) printf („0\n“);
          else printf („%d\n“, l/2+1-max);
  // l/2+1 je pocet vrcholov
  return 0;
}
```

P-II-3

Riešenie súťažnej úlohy

Myšací problém zjavne súvisí s tokmi, ale ako? Potrebujeme nájsť čo najviac navzájom vrcholovo disjunktných ciest z 1 do N. (Takéto cesty budeme volať *nezávislé*.) Podmienka „myš môže ísť cez každý vrchol najviac raz“ pripomína situáciu, kedy určujeme kapacitu jednotlivých hrán – tu by sme ale potrebovali obmedziť akúsi kapacitu vrchola. Ako na to?

Z grafu G na vstupe vytvoríme orientovaný graf G' takto: Pre každý vrchol v v G okrem vrchola 1 a N vytvoríme v G' dva vrcholy v^1 a v^2. Vrcholy v^1 a v^2 spojíme orientovanou hranou z v^1 do v^2 s kapacitou 1. (Prechod po tejto hrane bude zodpovedať prechodu pôvodným vrcholom. Vrchol v^1 bude akoby „vstupná časť“ a v^2 „výstupná časť“ pôvodného vrcholu v.)

Pre vrchol 1 vytvoríme iba vrchol 1^2 a pre vrchol N iba vrchol N^1, keďže nechceme vchádzať do 1 ani vychádzať z N. Každú hranu e (spájajúcu vrcholy a a b) nahradíme v G' dvomi orientovanými hranami: z a^2 do b^1 a z b^2 do a^1. Tieto hrany môžu mať ľubovoľnú kapacitu, napríklad ju tiež nastavme na 1. Výsledný graf G' dáme na vstup čiernej skrinke (vrchol 1^2 je zdroj a vrchol N^1 je ústie). Ona nám vráti nejaké číslo c_m – hodnotu maximálneho toku v G'. Tvrdíme, že počet kúskov syra, ktoré vie myška preniesť, je rovný $\lfloor c_m/2 \rfloor$. To však ešte treba dokázať.

Všimnime si, že každá hrana v G' má veľkosť jedna, teda voda buď hranou preteká v plnej kapacite alebo hranou voda nepreteká. Ukážeme, že c_m je rovné maximálnemu počtu nezávislých ciest medzi vrcholmi 1 a N, t.j. takých, že každá začína vo vrchole 1, končí vo vrchole N a každý vrchol (okrem 1 a N) sa nachádza na najviac jednej ceste. Označme maximálny počet nezávislých ciest p.

Najprv ukážeme, že $p \leq c_m$. Pozrime sa na ľubovoľných p nezávislých ciest v G. Každá cesta $w_1, w_2, \ldots, w_n$ z vrchola 1 do vrchola N v G nám určuje tok v G' prirodzeným spôsobom: Voda preteká iba hranami z ${w_i}^1$ do ${w_i}^2$ a hranami z ${w_i}^2$ do ${w_{i+1}}^1$. Cesty sú nezávislé a teda vieme toky prislúchajúce jednotlivým cestám spojiť do jedného veľkého toku tak, že vo výslednom veľkom toku bude voda prechádzať iba tými hranami, ktoré sa nachádzajú v niektorom čiastkovom toku. Teda pre každých p nezávislých ciest vieme v G' nájsť tok veľkosti c_m.

Ešte ukážeme, že $c_m \leq p$. Majme nejaký tok v G'. Urobíme presne opačnú konštrukciu ako v predchádzajúcom odstavci. Do každej dvojice vrcholov v^1 a v^2 $v \neq N$ v G' priteká a z nej odteká zjavne len jedna jednotka vody. Začnime vo vrchole 1^2 a vyberme sa nejakou hranou, ktorou z neho odteká voda. Teda keď prídeme do ľubovolného vrchola v^1, kde v nie je ústie, tak z neho vieme pokračovať do v^2 a z neho niekam ďalej. Keď prídeme do ústia, tak sme vlastne našli nejakú cestu v G z 1 do N. Keď sa vydáme na začiatku inou hranou, ktorou odteká voda, tak dostaneme nejakú ďalšiu cestu z 1 do N. Tieto cesty sú zjavne nezávislé. Teda ak zo zdroja vychádza k hrán, po ktorých tečie voda, tak vieme nájsť k nezávislých ciest v G z 1 do N. Zo zdroja vychádza práve c_m takých hrán a teda vieme nájsť aspoň c_m nezávislých ciest a teda $c_m \leq p$.

Ľahko si už môžete aj sami dokázať, že myška vie preniesť $\lfloor p/2 \rfloor$ kúskov syra, kde p je maximálny počet nezávislých ciest v G.

Hľadanie maximálneho toku

Okrem riešenia zadanej hodnoty si navyše ukážeme aj to, ako implementovať algoritmus skrytý v našej čiernej krabičke.

Začnime tým, že si definujeme jeden nový pojem: *minimálny rez*. Predstavme si, že niektoré uzly zafarbíme na čierno a ostatné na bielo tak, aby s bol čierny a t biely. Takéto ofarbenie uzlov voláme *rez*. Všimnime si teraz všetky potrubia, ktoré majú začiatok čierny a koniec biely. Číslo, ktoré dostaneme sčítaním ich kapacít, voláme *veľkosť rezu*. *Minimálny rez* je teda taký rez, ktorý má zo všetkých možných rezov najmenšiu veľkosť.

(Alebo formálne, *rez grafu* je rozdelenie všetkých vrcholov grafu do dvoch disjunktných množín A, B tak, že zdroj patrí do A a ústie do B. Veľkosť rezu je súčet kapacít všetkých hrán, ktorých začiatočný vrchol patrí do A a koncový do B. *Minimálny rez* je rez daného grafu s najmenšou možnou veľkosťou.)

Dokážeme teraz, že veľkosť minimálneho rezu je rovná veľkosti maximálneho toku. Označme veľkosť minimálneho rezu r_m a veľkosť maximálneho toku t_m. Veľkosť toku F budeme značiť t_F, veľkosť rezu R budeme značiť r_R.

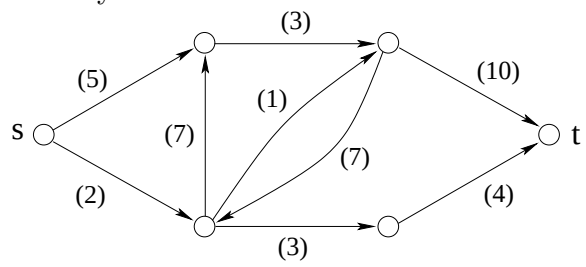
Zvoľme pevne nejaký tok F. Pre daný rez R (určený časťami A a B, pričom zdroj sa nachádza v A a ústie v B) označme o_R množstvo vody vytekajúce z časti A do časti B a p_R množstvo vody pritekajúce do A z B. Všimnime si, že $o_R - p_R$ je pre ľubovoľný rez R rovné veľkosti toku F. Tiež si všimnime, že o_R je nanajvýš rovné veľkosti rezu R (lebo každým potrubím vedúcim z A do B tečie nanajvýš toľko, koľko je jeho kapacita).

Spojením týchto pozorovaní dostávame: $t_F = o_R - p_R \leq o_R \leq r_R$, inými slovami veľkosť ľubovoľného toku je nanajvýš rovná veľkosti ľubovoľného rezu. Špeciálne teda aj veľkosť maximálneho toku je nanajvýš rovná veľkosti minimálneho rezu.

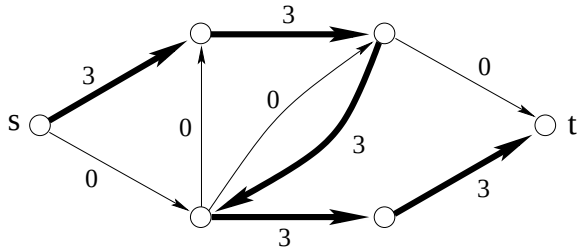
(Na túto nerovnosť nám stačí aj „sedliacky rozum“. Stačí si uvedomiť, že pre ľubovoľný tok a ľubovoľný rez každý „kúsok“ vody idúci zo zdroja do ústia musí skôr či neskôr prejsť niektorou z hrán, ktoré „tvoria rez“, teda vedú z prvej jeho množiny do druhej.)

Na dôkaz opačnej nerovnosti si zadefinujeme pojem zlepšujúca cesta a kvôli jednoznačnejšiemu popisu budeme používať terminológiu z teórie grafov (viď študijný text na konci zadania). Majme nejaký graf G a v ňom nejaký tok f. Vytvorme z grafu G graf G' tak, že v ňom ponecháme všetky vrcholy a postupne doň budeme pridávať hrany: Pre každú „nenasýtenú“ hranu e_i v G (t.j. takú, že $f(e_i) < c_i$) vedúcu z a do b dáme do grafu G' hranu z a do b s kapacitou $c_i - f(e_i)$. Označme tieto hrany ako hrany prvého druhu. Pre každú hranu e_i vedúcu z a do b, cez ktorú tečie aspoň nejaká voda (t.j. $f(e_i) > 0$), dáme do G' hranu z b do a s kapacitou $f(e_i)$. Tieto hrany označme ako hrany druhého druhu. Zlepšujúca cesta je ľubovolná cesta v grafe G' zo zdroja do ústia. Rezervou zlepšujúcej cesty C označme najmenšiu kapacitu hrany na tejto ceste. Označme ju r_c.

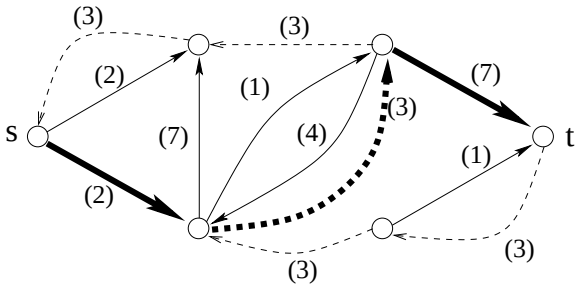
Pozrime sa na nejaký tok f pre ktorý existuje nejaká zlepšujúca cesta $C = v_1, v_2, \ldots, v_k$, kde v_1 je zdroj a v_k je ústie. Bez ujmy na všeobecnosti sa v nej neopakujú vrcholy. Nech e_i je hrana medzi v_i a v_{i+1}. Ukážeme, že vieme zvýšiť hodnotu toku f a dostať tak nejaký iný tok f', t.j. že f nie je maximálny. V toku f modifikujeme pretekajúce množstvo vody len na hranách na zlepšujúcej ceste. Na všetkých hranách e_i prvého druhu položme $f'(e_i) = f(e_i) + r_c$, kde r_c je príslušná rezerva zlepšújúcej cesty C. Na všetkých hranách druhého druhu položme $f'(e_i) = f(e_i) - r_c$. Dá sa jednoducho ukázať, že f' bude po tejto modifikácii toku f zase korektný tok.



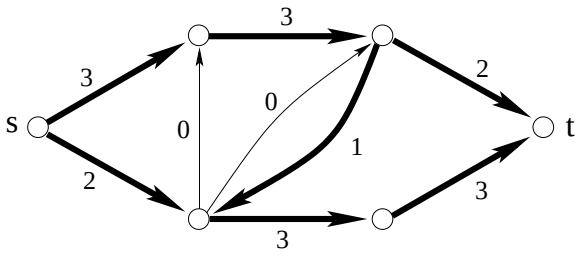
a) príklad siete potrubí



b) príklad toku veľkosti 3



c) graf G' pre zlepšujúce cesty



d) upravený tok

(Na obrázku c sú čiarkovane znázornené hrany druhého typu. Hrubo je vyznačená jedna možná zlepšujúca cesta, na obrázku d je nový tok, ktorý dostaneme z toku na obrázku b použitím zlepšujúcej cesty z obrázku c. Všimnite si, ako sa zmenil tok hranou idúcou doľava dodola.)

Ešte raz inými slovami si povedzme, čo robíme pri hľadaní zlepšujúcej cesty. Chceme našou sieťou potrubí pretlačiť ešte nejakú vodu, musíme jej teda nájsť cestu, ktorou potečie. Toto bude práve tá naša zlepšujúca cesta. Ako môže vyzerať? Hrany prvého typu sú jasné – ak máme potrubie, ktorého kapacitu sme ešte nedosiahli, môžeme ním poslať viac vody. O čom sú ale hrany druhého druhu? Predstavme si, že máme potrubie z a do b, ktorým teľie k litrov vody za sekundu. Ak teraz nejako privedieme napríklad 1 liter vody do b, môžeme spraviť to, že touto hranou pustíme z a do b len $k-1$ litrov. Tým opäť pre vrchol b platí, že doň priteká toľko, koľko odteká – zvyšuje nám ale liter vody vo vrchole a. Výsledný efekt je teda ten istý, ako keby sme daný liter vody „pretlačili proti prúdu“ potrubím z a do b.

Ukázali sme si už teda, že ak k danému toku existuje zlepšujúca cesta, vieme po nej poslať vodu a tento tok zväčšiť. To znamená, že pre maximálny tok nemôže existovať zlepšujúca cesta. Majme teraz maximálny tok f. Vytvorme preň graf G' tým istým spôsobom ako je uvedené vyššie. Vieme, že v tomto grafe nemôže existovať cesta zo zdroja do ústia. Zostrojme rez R nasledovne: do prvej množiny A dáme všetky vrcholy, ktoré sú v G' dosiahnuteľné z s, do druhej množiny B všetky ostatné vrcholy. Označme veľkosť tohto rezu r_R. Ukážeme, že každá hrana e_i z nejakého vrchola $a \in A$ do nejakého vrchola v $b \in B$ je nasýtená t.j. $f(e_i) = c_i$. V opačnom prípade totiž existuje cesta v G' z a do b (bude prvého druhu) a do komponentu A patrí aj ďalší vrchol z B. Ďalej platí, že cez žiadnu z hrán e_j z $b \in B$ do $a \in A$ netečie žiadna voda. Inak totiž z tohto istého dôvodu sa komponent A dá rozšíriť o ďalší vrchol.

Teda v toku f z komponentu A do B tečie r_R jednotiek vody a z komponentu B do A už žiadna voda netečie. Teda veľkosť toku f je nutne r_R. Ukázali sme, že veľkosť maximálneho toku c_m sa rovná veľkosti nejakého rezu, preto veľkosť maximálneho toku je väčšia alebo rovná veľkosti minimálneho rezu r_m, č.b.t.d.

A nielen to. Našli sme totiž aj algoritmus, ako (jeden možný) maximálny tok v danom grafe zostrojiť. Stačí začať s ľubovoľným tokom (napríklad prázdnym) a dookola hľadať zlepšujúce cesty a vylepšovať tak aktuálny tok, kým sa to dá. Dokázali sme totiž, že ak už neexistuje zlepšujúca cesta, je aktuálna veľkosť toku rovná veľkosti minimálneho rezu – a väčší tok už tým pádom neexistuje.

Na hľadanie zlepšujúcej cesty môžeme použiť jednoduché prehľadávanie grafu. Dá sa ukázať, že napríklad ak použijeme prehľadávanie do šírky (čiže vždy nájdeme zlepšujúcu cestu s najmenším možným počtom hrán), bude mať výsledný algoritmus časovú zložitosť $O(M^2N)$, teda bude polynomiálny.[2]

[2] N je počet vrcholov grafu (uzlov), M je počet hrán (potrubí).

P–II–4

V riešení použijeme príkazy **ForAll** a **Exists**, ktoré sme definovali v riešeniach domáceho kola. (Príkaz **ForAll**(c,N) paralelne spustí N kópii, v ktorých postupne c=0, ..., N-1; program úspešne skončí, ak všetky tieto kópie úspešne skončia. Príkaz **Exists** funguje analogicky, len stačí, aby úspešne skončila jedna ľubovoľná kópia.)

V prvom rade si uvedomme, že miest je nanajvýš $2M$ a že všetky ich čísla máme v poli C. Keď teda potrebujeme prezrieť všetky mestá, môžeme namiesto toho prezrieť všetky políčka poľa C.

Zamyslime sa, ako vieme najrýchlejšie overiť pre dve konkrétne mestá x a y, či sa dá dostať z x do y. Prvý dôležitý postreh je, že ak existuje spôsob, ako prísť z x do y, tak existuje taký spôsob, pri ktorom žiadnou cestou nejdeme dvakrát. Stačí nám teda vedieť overiť, či sa z x do y vieme dostať na najviac M „krokov“.

Toto ľahko spravíme využitím príkazu **Exists**: postupne „hádame“ cesty, ktorými ideme. Ak sa nám podarí dostať do y, zavoláme **Accept**, ak už sme išli M cestami a stále nie sme v y, zavoláme **Reject**.

Týmto dostávame triviálne riešenie v čase $O(M \log M)$: Pre každú dvojicu miest x, y paralelne overíme, či sa vieme z x dostať do y.

(Na tomto mieste by sme chceli podotknúť, že existuje riešenie v čase $O(M \log M)$, ktoré vôbec nevyužíva paralelizátor: Utriedime čísla z poľa C, prečíslujeme vrcholy číslami od 1 do $2M$, dvoma prehľadávaniami do šírky/hĺbky zistíme, či sa vieme dostať z mesta 1 do všetkých iných a či sa odvšadiaľ vieme dostať do mesta 1. Jedinou nevýhodou tohto riešenia je, že je omnoho zložitejšie a je ľahké v ňom spraviť chybu.)

Teraz si ukážeme, ako efektívnejšie overiť, či sa vieme dostať z x do y (na najviac M krokov). Trik je jednoduchý: Ak sa z x do y vieme dostať priamo, vyhrali sme. Ak nie, „uhádneme“ mesto z, cez ktoré pôjdeme (približne) v strede našej púte z x do y. Teraz potrebujeme overiť, či sme si dobre tipli – teda či sa vieme dostať z x do z (na najviac $\lceil M/2 \rceil$ krokov) aj z z do y (na najviac $\lfloor M/2 \rfloor$ krokov). Tieto dve veci ale vieme overiť paralelne!

Dostávame teda nasledujúci program:

```
{ VSTUP:
   M : longint;
   C : array[0..M-1][0..1] of longint; }


{ ForAll() paralelne spusti N kopii, v ktorych cislo=0..(N-1),
  uspesne skonci, ak vsetky uspesne skoncia }
procedure ForAll(var cislo : longint, N : longint);
var moc2, cifier, i, x : longint
begin
   { zistime, kolko ma N-1 cifier v dvojkovej sustave }
   moc2 := 1;
   cifier := 0;
   while (moc2 <= N-1) do begin moc2 := moc2 * 2; inc( cifier ); end;
   { vygenerujeme cisla od 0 do 2^cifier - 1 }
   cislo := 0;
   for i:=1 to cifier do begin Both(x); cislo := 2*cislo + x; end;
   if (cislo >= N) then Accept;
```

```
end;

{ Exists() paralelne spusti N kopii, v ktorych cislo=0..(N-1),
  uspesne skonci, ak niektora z nich uspesne skonci }
procedure Exists(var cislo : longint, N : longint);
var moc2, cifier, i, x : longint
begin
   { zistime, kolko ma N-1 cifier v dvojkovej sustave }
   moc2 := 1;
   cifier := 0;
   while (moc2 <= N-1) do begin moc2 := moc2 * 2; inc( cifier ); end;
   { vygenerujeme cisla od 0 do 2^cifier - 1 }
   cislo := 0;
   for i:=1 to cifier do begin Some(x); cislo := 2*cislo + x; end;
   if (cislo >= N) then Reject;
end;

{ Over(x,y,k) overi, ci sa z „x" da dostat do „y" na <=„k" krokov }
procedure Over(x,y,k : longint);

var priamo, pom, z : longint;

begin
   { najskor okrajove pripady }
   if (k=0 and x=y) then Accept;
   if (k=0 and x<>y) then Reject;

   { uhadneme, ci sa vieme dostat priamo, ak ano, overime }
   Some(x);
   if (x=1) then begin
      { uhadneme cislo spravnej hrany }
      Exists(pom,M);
      if (C[pom][0]=x and C[pom][1]=y) then Accept;
      Reject;
   end;

   { ak sa nedalo dostat priamo a mame len jeden krok, nejde to }
   if (k=1) then Reject;

   { uhadneme stredny vrchol, staci vybrat koniec niektorej hrany }
   Exists(pom,M);
   z := C[pom][1];

   { paralelne overime, ci existuju obe cesty polovicnej dlzky }
   Both(pom);
   if (pom=0) then
      Over(x,z,(M+1) div 2);
   else
      Over(z,y,M div 2);
```

```
end;

{ hlavny program: uspesne skoncime, ak pre kazdu dvojicu miest x, y
  over(x,y,M) uspesne skonci }

var mesto1, mesto2, i1, j1, i2, j2 : longint;

begin
   ForAll(i1,M); Both(j1);
   ForAll(i2,M); Both(j2);
   mesto1 := C[i1][j1];
   mesto2 := C[i2][j2];
   Over(mesto1,mesto2,M);
end.
```

Jeho časová zložitosť je $O(\log^2 M)$. Prečo? Pri každom rekurzívnom volaní procedúry Over sa počet krokov zmenší približne na polovicu, teda hĺbka rekurzie je $O(\log M)$. Najzložitejšia operácia pri behu procedúry Over je jedno volanie **Exists**, na ktoré potrebujeme čas $O(\log M)$.

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY
55. ROČNÍK MATEMATICKEJ OLYMPIÁDY
Riešenia 2. kola kategórie P

Vydala IUVENTA s finančnou podporou Ministerstva školstva SR
Náklad: 320 výtlačkov
Zodpovedný redaktor: M. Forišek
Sadzba programom LaTeX