Korešpondenčný seminár z programovania
XX. ročník, 2002/2003
Katedra vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporuje MICROSTEP-MIS spol. s r.o.

Vzorové riešenia 2. kola zimnej časti

Milí riešitelia!
Tí najlepší z vás sa už určite tešia na sústredenie. A vy ostatní by ste sa mali o to pozornejšie začítať do týchto vzorových riešení, aby ste nabudúce boli múdrejší a mohli sa aj vy týždeň ulievať zo školy :-)

Quidquid latine dictum sit, altum viditur.[1]

KSPáci

1. Opravovanie

Opravoval Braňo z Brezna
(max. 15 bodov)

Väčšina riešení bola správna, čo učiteľku Janku veľmo potešilo. Označme si počet žiakov N a poďme k bodovaniu:

- správne riešenia v časovej zložitosti $O(N)$ 15 bodov
- správne riešenia v časovej zložitosti $O(N^2)$ 13 bodov
- nesprávne riešenia max. 4 body
- chýbajúci dôkaz správnosti -2 body

Vzorové riešenie: Na začiatok si uvedomme, že ak viac ako $\lfloor \frac{N}{2} \rfloor$ žiakov písalo tú istú skupinu, tak úloha nemá riešenie. Potrebovali by sme totiž rozdeliť ich písomky zvyšným žiakom, ktorých je menej. To nejde[2]. Naopak, ak žiaden príklad nepísala nadpolovičná väčšina žiakov, tak ukážeme, že úloha riešenie má. A dokonca nejaké riešenie vieme efektívne nájsť.

Postavme žiakov do radu, každý nech má pri sebe svoju písomku. Žiakov usporiadame podľa skupiny písomky: najprv budú tí čo písali skupinu 'a', potom 'b', ... Potom sa tak ako stoja pochytajú za ruky. Navyše sa chytia aj posledný a prvý, čím utvoria kruh. Keď už stoja v kruhu, pustia sa (aby mali voľné ruky). Teraz každý žiak podá svoju písomku susedovi vľavo, pričom toto podávanie zopakujú $\lfloor \frac{N}{2} \rfloor$-krát. Tvrdíme, že teraz určite každý žiak drží v rukách písomku inej skupiny ako písal.

Tvrdenie dokážeme sporom. Očíslujme si žiakov od 0 do $N-1$ po obvode kruhu v smere, v akom si podávali písomky. Nech má niektorý žiak (BUNV nech je to ten s číslom $N-1$) teraz v rukách písomku svojej skupiny. Potom on aj žiak $N-1-\lfloor \frac{N}{2} \rfloor$ písali tú istú skupinu. Ale žiaci boli zoradení podľa skupín, preto máme dve možnosti:

- Túto skupinu písali aj všetci žiaci od $N-\lfloor \frac{N}{2} \rfloor$ po $N-2$. To je spor – je ich dokopy aspoň $\lfloor \frac{N}{2} \rfloor + 1$.
- Túto skupinu písali aj všetci žiaci od 0 do $N-2-\lfloor \frac{N}{2} \rfloor$. To je opäť spor – je ich aspoň $N+1-\lfloor \frac{N}{2} \rfloor \geq 2\lfloor \frac{N}{2} \rfloor + 1 - \lfloor \frac{N}{2} \rfloor > \lfloor \frac{N}{2} \rfloor$.

Preto naozaj každý žiak má v rukách písomku inej skupiny ako písal, q.e.d.

Pri implementácii použijeme na utriedenie countsort, lebo rozsah skupín písomiek je malý a chceme dosiahnuť lineárnu časovú zložitosť. Aby sme vedeli na konci usporiadať

[1] Voľne preložené: Čokoľvek povedané po latinsky znie múdro.

[2] Namakane sa tomu hovorí Dirichletov princíp.

žiakov späť do poradia, v akom boli na vstupe, budeme si pre každého žiaka v utriedenom poradí pamätať jeho pozíciu na vstupe (v listingu pole `inv`).

Na záver môžme pyšne zhrnúť, že časová zložitosť nášho algoritmu je $O(N)$, čo je optimálne. Už samotné vypísanie výstupu je totiž lineárne. Pamäťová zložitosť je $O(N)$.

Listing programu:

```
const maxN = 100;
var
  s,ss : string;
  count, start : array ['a'..'z'] of integer;
  inv : array [1..maxN] of integer; //tuna bol na zaciatku i-ty ziak
  c : char;
  i, max, n : integer;

begin
  readln(s); n:=length(s); ss:=s;

  for c:='a' to 'z' do count[c]:=0;
  for i:=1 to n do
    inc( count[ s[i] ] );

  start['a']:=1; max:=count['a'];
  for c:=succ('a') to 'z' do begin  //succ('a')='b'; pred('d')='c'
    start[c]:= start[pred(c)]+count[pred(c)];
    if max<count[c] then
      max:=count[c];
  end;

  if (max> (n div 2) ) then begin
    writeln('Neda sa'); halt();
  end;

  for i:=1 to n do begin
    ss[ start[ s[i] ] ]:= s[i]; //count sort
    inv[ start[ s[i] ] ]:= i;
    inc( start[ s[i] ] );
  end;

  for i:=1 to n do  //zrotuj utriedene pole a prezen cez inv
    s[ inv[i] ]:= ss[ 1+ (i-1+n div 2) mod n ];

  writeln(s);
end.
```

2. O metre II

opravoval WSX z Bystrice
(max. 13 bodov)

Pracovníci firmy Tunelár a syn boli veľmi smutní, že dostali iba 11 riešení. Jediné, čo ich potešilo bolo, že skoro všetky boli funkčné. Vaše riešenia boli aj napriek slabšiemu počtu rozmanité – od backtrackov, za ktoré bolo možné získať 7 bodov, cez inžinierske riešenia (max. 8 bodov) až po polynomiálne riešenia v čase $O(N^3 \log N)$ (12 bodov) a napokon optimálne 13-bodové v čase $O(N^3)$. Samozrejme, za nedostatočný popis alebo drobné chyby ste mohli voľačo aj stratiť.

Prejdime k samotnému optimálnemu riešeniu. Ako sme si ukázali v 1. sérii, ak máme stanice s_1, s_2, kolmicu na ich spojnicu k a orientované priamky p_1, p_2, pričom uhol kolmice leží medzi uhlami týchto dvoch priamok, potom majú stanice s_1, s_2 opačné poradie na

priamkach p_1, p_2. Takisto sme si ukázali, že bez ujmy na všeobecnosti môžeme predpokladať, že priamka prechádza bodom $[0, 0]$.

Vidíme teda, že kolmice spojníc staníc budú pre nás akési hraničné priamky, pri ktorých nastáva zmena poradia staníc. Takýchto kolmíc je polynomiálne veľa – $N(N-1)$. Pre každú kolmicu si budeme pamätať jej uhol v intervale $\langle 0, 2\pi)$ a čísla tých dvoch staníc, ktorých sa výmena týka. Predstavme si, že vieme poradie staníc pre nejaký uhol priamky, ktorý je len málo menší od uhlu nejakej kolmice. Po prechode kolmicou sa vymenia v poradí práve tie 2 stanice, ktoré máme pri danej kolmici zapísané. Túto zmenu vieme zrealizovať v konštantnom čase, no nám to bude stačiť lineárne, lebo asymptotickú zložitosť algoritmu to neovplyvní. (Počet rôznych postupností staníc môže byť až $N(N-1)$ a pre každú potrebujeme vypísať N čísel, preto len na samotný výpis potrebujeme čas $O(N^3)$.)

Problémy môžu nastať, ak by viac kolmíc bolo navzájom rovnobežných. (Prečo?) Najjednoduchšie praktické riešenie je tzv. *symbolická perturbácia* bodov – ku každej súradnici každého bodu pripočítame zanedbateľne malú náhodnú konštantu. Z pôvodne rovnobežných kolmíc stanú takmer rovnobežné, ktoré už vieme spracúvať v jednoznačnom poradí. Zároveň permutáciu vypíšeme len ak uhol, v ktorom sa nachádzame, nie je takmer nulový. (t.j. priamky boli pôvodne rovnobežné). Toto samozrejme bude prakticky fungovať, aj keď z teoretického hľadiska to nie je úplne kóšer. Samozrejme dá sa to spraviť aj korektne, ale je to zložitejšie. Detaily prenechávame na šikovného čitateľa. *(Návodné otázky: Majme 5 bodov na 1 priamke. Aké sú možné poradia? Ako ich preklopiť v lineárnom čase? Majme teraz dve rovnobežky, na každej 5 bodov. Ako vyzerajú priemety na priamky skoro kolmé na ne? Ako ich preklopiť v lineárnom čase?)*

Aby sme vedeli spracúvať kolmice zaradom, bolo by vhodné polárne (t.j. podľa uhlu) ich utriediť. Môžeme na to použiť ľubovoľné štandardné triedenie, alebo napríklad takúto fintu.[3] Máme $O(N^2)$ kolmíc, chceme ich utriediť v čase $O(N^3)$. Pole kolmíc rozdelíme na N-tice, každú z nich utriedime ľubovoľným triviálnym kvadratickým triedením. Kolmíc je spolu $N(N-1)$, teda N-tíc je $N-1$. Teraz $N(N-1)$-krát nájdeme minimálny prvok a vymažeme ho. Vieme, že minimálny prvok je vždy prvý nevymazaný v niektorej z N-tíc, teda ho vieme nájsť v čase $O(N)$. Zložitosť tohto triedenia bude teda naozaj $O(N^3)$.

Vieme síce z jednej postupnosti staníc získať nasledujúcu po prechode kolmicou, ale potrebujeme ešte získať niektorú, od ktorej začneme. Tú získame nasledovne: V polí utriedených kolmíc nájdeme prvé také dve, že interval medzi nimi je neprázdny. Zoberieme ľubovoľný uhol medzi nimi a zrotujeme všetky stanice o tento uhol doprava. Tým sa zachovajú všetky vlastnosti, lebo rotácia je zhodné zobrazenie a posunie sa aj naša priamka s nájdeným uhlom tak, že bude ležať na osi x. Teraz nám už stačí len utriediť x-ové súradnice a dostaneme danú permutáciu staníc. (Prípadne by stačilo namiesto otáčania zobrať priamo priemety na priamku určenú nájdeným uhlom a zotriediť tie.)

V tomto okamihu nám už len stačí prebehnúť po kolmiciach a zakaždým aplikovať výmenu daných dvoch staníc. Pokiaľ sa nachádzame medzi kolmicami s rôznym uhlom, zároveň aj vypíšeme dané poradie.

Pamäťová zložitosť je $O(N^2)$ kvoli počtu kolmíc. Výsledná časová zložitosť je $O(N^3)$, teda z hľadiska asymptotickej časovej zložitosti je náš algoritmus optimálny.

Listing programu:

```
Uses Math;
Const Eps=0.000001;
      Inf=1E20;
      MaxN=100;
Type Uhol=Record
```

[3] Takto sa vlastne s ľubovoľného triedenia so zložitosťou $O(N^2)$ dá vyrobiť triedenie so zložitosťou $O(N\sqrt{N})$. Dá sa podobný postup zopakovať? Aké zlepšenie dosiahneme?

```
            R   :Real;    { uhol v intervale <0,2*PI) }
            A, B:Integer; { prvky na vymenu }
          End;

Var
  I, J, K, N, M  :Integer;
  R    :Real;
  X, Y    :Array[1..MaxN] Of Real;
  Uhly, SUhly  :Array[1..MaxN*MaxN] Of Uhol;
  B    :Array[1..MaxN] Of Integer;
  Zmen    :Array[1..MaxN] Of Boolean;
  XX    :Array[1..MaxN] Of Real;

{ znormalizuj uhol -- vrati uhol v intervale <0,2*PI) }
Function NormUhol(U:Real):Real;
Begin
  While U<0 Do U:=U+2*PI;
  While U>=2*PI Do U:=U-2*PI;
  NormUhol:=U;
End;

Procedure Sort(A, B:Integer);  { tried v O((B-A)^2) }
Var I, J:Integer;
    U:Uhol;
Begin
  For I:=A To B-1 Do For J:=I+1 To B Do
    If Uhly[I].R>Uhly[J].R Then Begin
      U:=Uhly[I];
      Uhly[I]:=Uhly[J];
      Uhly[J]:=U;
    End;
End;

Procedure Vypis(X:Array Of Integer);  { vypis permutaciu }
Var I:Integer;
Begin For I:=0 To N-2 Do Write(X[I],' '); WriteLn(X[N-1]); End;

Begin
  ReadLn(N);
  Case N Of
    0:WriteLn;    { prazdna permutacia (neznamena ziadna, aj 0!=1) }
    1:WriteLn('1');  { permutacia obsahujuca stanicu 1 }
  End;
  If N<2 Then Halt;
  For I:=1 To N Do ReadLn(X[I], Y[I]);

  M:=0;
  For I:=1 To N-1 Do For J:=I+1 To N Do Begin
    Inc(M);
    Uhly[M].R:=NormUhol(ArcTan2(Y[I]-Y[J], X[I]-X[J])+PI/2);
    Uhly[M].A:=I; Uhly[M].B:=J;
    Inc(M);
    Uhly[M].R:=NormUhol(ArcTan2(Y[I]-Y[J], X[I]-X[J])-PI/2);
    Uhly[M].A:=I; Uhly[M].B:=J;
  End;
  Uhly[M+1].R:=47;

  { utried kazdu N-ticu, dokopy v case O(N^3) }
  For I:=1 To N-1 Do Sort(1+(I-1)*N, I*N);
  For I:=1 To N-1 Do B[I]:=1+(I-1)*N;  { zaciatky utriedenych casti }
```

```
  For I:=1 To N*(N-1) Do Begin
    K:=1;
    For J:=2 To N-1 Do If Uhly[B[J]].R<Uhly[B[K]].R Then K:=J;
    SUhly[I]:=Uhly[B[K]];
    Uhly[B[K]].R:=47;  { 47>2*PI, nikdy sa nepouzije }
    Inc(B[K]);    { presun sa na dalsi uhol }
  End;

  SUhly[N*(N-1)+1]:=SUhly[1];
  SUhly[N*(N-1)+1].R:=SUhly[N*(N-1)+1].R+2*PI;

  M:=1;
  While SUhly[M+1].R-SUhly[M].R<Eps Do Inc(M);  { najdeme nenulovy interval uhlov }
  R:=(SUhly[M+1].R+SUhly[M].R)/2;  { priemer uhlov }

  { otocime o uhol R doprava okolo [1,1], zaujimaju nas X-suradnice }
  For I:=1 To N Do XX[I]:=X[I]*Cos(R)+Y[I]*Sin(R);

  For I:=1 To N Do Begin
    R:=Inf;
    For J:=1 To N Do If XX[J]<R Then Begin R:=XX[J]; K:=J; End;
    XX[K]:=Inf;
    B[I]:=K;
  End;

  Vypis(B);

  For I:=1 To N Do Zmen[I]:=False;
  For I:=M+1 To N*(N-1) Do    { pre vsetky intervaly }
  Begin
    For J:=1 To N Do      { nastav priznak zmeny }
      If B[J]=SUhly[I].A Then Zmen[J]:=True Else
      If B[J]=SUhly[I].B Then Zmen[J]:=True;

    If SUhly[I+1].R-SUhly[I].R>Eps Then  { ak je interval neprazdny }
    Begin
      J:=1; K:=N;      { pretoc prvky oznacene na zmenu }
      While J<K Do Begin
        While (J<=N) And Not Zmen[J] Do Inc(J);
        While (K>=1) And Not Zmen[K] Do Dec(K);
        Zmen[J]:=False; Zmen[K]:=False;
        If J=K Then Break;
        B[J]:=B[J] Xor B[K]; B[K]:=B[J] Xor B[K]; B[J]:=B[J] Xor B[K];
      End;
      Vypis(B);        { vypis permutaciu }
    End;
  End;
End.
```

opravoval Dávidko z Bratislavy
(max. 15 bodov)

3. O cyklistike na Zahori

Potešilo ma, že veľa ľudí zvládlo napísať riešenie podobné vzorovému – tí dostali zväčša plný počet. Tiež som bol rád, že každý z vás zvládol napísať popis.

Riešenia, ktoré rátali počet trás len zvačšovaním o jedna, nemôžu fungovať rýchlejšie ako exponciálne (ak vôbec). To plynie z toho, že počet trás može byť až exponenciálne veľký od počtu miest a ciest. Skúste si za domácu úlohu nájsť cestnú sieť, kde medzi Bratislavou a

Malackami existuje až exponeciálne veľa najkratších trás. (Nie je to ťažké.) Bodovanie bolo teda takéto:

- Riešenia s časovou zložitosťou $O(N^2)$ – max. 15 bodov.
- Horšie polynomiálne – max. 12 bodov
- Exponenciálne riešenia – max. 8 bodov.
- Nefunkčné a polofunkčné riešenia – max. 4 body.

Okrem toho som stŕhal za chyby v programe a za chýbajúci listing.

Vzorové riešenie:

Vzorové riešenie je jednoduchou modifikáciou Dijkstrovho[4] algoritmu. Najprv teda vysvetlíme, čo je to za algorimus a ako funguje a potom povieme, ako ho treba modifikovať. Teda, Dijkstrov algoritmus slúži na hľadanie najkratších ciest v grafoch[5]. Funguje nasledovne: Aby sme vyrátali dĺžku najkratšej trasy z Bratislavy do Malaciek, budeme riešiť zdanlivo ťažšiu úlohu – budeme hľadať dĺžky najkratších trás z Bratislavy do všetkých miest.

Pre každé mesto si budeme pamätať dve veci: Dĺžku doteraz najkratšej nájdenej trasy doň a to, či je táto dĺžka už skutočne definitívna. (Ak sa v ďalšom texte vyskytne slovo trasa, myslí sa tým trasa z Bratislavy kamsi. Iné trasy nebudeme vôbec brať do úvahy.) Na začiatku položíme dĺžky trás nekonečno s výnimkou Bratislavy – tú položíme rovnú nule. A všetky dĺžky prehlásime za zatiaľ nedefinitívne. Zvyšok algoritmu sa skladá z N krokov. V každom z nich robíme toto:

- Vyberieme najkratšiu spomedzi nedefinitívnych trás a prehlásime ju za definitívnu. Toto bude vzdialenosť najkratšej trasy do príslušného mesta. Nazveme to zdefinitívnenie mesta. Kvôli jednoduchosti vysvetľovania, nech sme povedzme zdefinitívnili Stupavu.
- Preberáme nedefinitívne mestá susediace so Stupavou. Nech napríklad Lozorno je také mesto (t.j. ešte nie je definitívne a medzi Stupavou a Lozornom je priama cesta). Ak trasa Bratislava – Stupava – Lozorno je kratšia, ako doterajšia trasa do Lozorna, tak si zapamätáme túto namiesto povôdnej. Toto voláme skracovanie trasy (cez mesto).

Vám odporúčam si ručne algoritmus odskúšať na nejakej cestnej sieti s aspoň 10 mestami. My sa namiesto toho vrhneme do dokazovania správnosti algoritmu t.j. ideme ukázať, že v okamihu, keď prehlásime trasu za definitívnu, je skutočne najkratšia možná.

Budeme postupovať indukciou vzhľadom na počet krokov algoritmu. V prvom kroku, keď prehlasujeme Bratislavu za definitívnu, iste máme pravdu. Nech teda sme v nejakom ďalšom kroku algoritmu. Vieme už, že trasy, ktoré sú prehlásené za definitívne, sú skutočne najkratšie. A nech najkratšia nedefitívna trasa je povedzme trasa T Bratislava – Stupava a chystáme sa ju zdefinitívniť. Musíme ukázať, že je naozaj najkratšia. Sporom. Nech existuje do Stupavy nejaká iná kratšia trasa S. Vezmime na S prvé nedefinitívne mesto (idúc z Bratislavy), sú dve možnosti:

- Nie je to Stupava. Nech je to povedzme Borinka. Potom ale trasa časť trasy S Bratislava – Borinka je určite kratšia, ako trasa S a je tá podľa predpokladu kratšia než T. Ale toto je spor jak hrom, lebo náš algorimus by vybral Borinku, a nie Stupavu, no nie!?
- Je to Stupava. Potom je ale mesto tesne pred Stupavou (idúc z Bratislavy) na trase S definitívne. Z toho plynie, že sme cez neho trasu do Stupavy skracovali a teda namiesto T by sme si zapamätali S. Spor.

Zostalo nám vyriešiť, ako hľadať počet najkratších trás. Pre každé mesto si pamätáme nielen dĺžku najkratšej doteraz nájdenej trasy doň, ale aj počet trás tejto dĺžky, ktoré doň vedú. Cesta dĺžky 0 z Bratislavy do Bratislavy je určite práve jedna. Ostatné počty sú na

[4]Edsger Wybe Dijkstra 1930–2002, holandský profesor informatiky. Medzi iným vymyslel aj štrukturované programovanie t.j. používanie begin, end, for, while a repeat namiesto goto.

[5]Graf je v tomto prípade matematický model cestnej siete.

začiatku nedefininované. Tieto počty upravujeme vtedy, keď skracujeme cesty. Prestavme si, že skracujeme trasu do Lozorna cez Stupavu. Máme tri možnosti:

- Trasa je dlhšia, ako doteraz pamätaná. Vtedy nerobíme nič.
- Trasa je ostro kratšia. Vtedy ju skrátime. Navyše pre každú najkratšiu cestu do Stupavy dostávame jednu doteraz najkratšiu cestu do Lozorna. Takže položíme počet trás do Lozorna rovný počtu trás do Stupavy.
- Trasa je rovnako dlhá. Vtedy sme sa dozvedeli o ďalších doteraz najkratších trasách do Lozorna. Takže počet trás do Lozorna zvýšime o počet trás do Stupavy.

Implementácia algoritmu je priamočiara (kukni listing). Máme 4 polia: Maticu susednosti t.j. pre každú dvojicu miest dĺžku priamej cesty medzi nimi (resp. nekonečno, ak priama cesta neexistuje). Pole pre doteraz najkratšie trasy. Pole, v ktorom si pamätáme, či je trasa definitívna a pole pre počty trás.

Krokov algoritmu je N a v každom z nich treba nájsť najkratšiu nedefinitívnu trasu a zlepšiť všetky nedefinitívne trasy. Obdidvoje trvá čas $\Theta(N)$. Výsledna časová zložitosť je teda $N \cdot \Theta(N) = \Theta(N^2)$. Pamäťová zložitosť je $\Theta(N^2)$ kvôli matici susednosti. Pamäťová zložitosť by šla znížiť na $\Theta(N + M)$ pomerne ľahko použitím niečoho iného ako matice susednosti. Čo sa týka času, existuje aj asympoticky rýchlejšia implementácia Dijkstrovho algoritmu využívajúca Fibonacciho haldu pracujúca v čase $O(M + N \log N)$, žial, je trocha náročná na vysvetlenie.

Listing programu:

```
Program O_cyklistike_na_Zahori;

const MaxN = 100; { maximalny pocet vrcholov }

var N,M,x,y,d,i,j,min:integer;
    a:array[1..MaxN, 1..MaxN] of integer;  { a[i,j] je dlzka cesty medzi i<->j }
    l,p:array[1..MaxN] of integer; { dlzka a pocet ciest z BA do vrchola }
    b:array[1..MaxN] of boolean; { true, ak je vrchol definitvny }

{ skrati dlzku trasy do j cez min, a prerata pocet tras do j }
procedure skrat(min,j:integer);
begin
   if l[min]+a[min,j] = l[j] then p[j]:=p[j] + p[min];
   if l[min]+a[min,j] < l[j] then begin
      l[j]:=l[min]+a[min,j];
      p[j]:=p[min];
   end;
end;

begin
   assign(input,'prikl3.in'); reset(input);
   readln(N,M);

   { na zaciatku su vsetky hrany nekonecno }
   for i:=1 to N do for j:=1 to N do a[i,j]:=9999;

   { nacitame hrany }
   for i:=1 to M do begin readln(x,y,d); a[x,y]:=d; a[y,x]:=d; end;

   { dlzky ciest BA <-> obec[i] nastavime na nekonecno
     a nastavime, ze su nedefinitivne t.j. budu sa este menit }
   for i:=1 to N do begin l[i]:=9999; b[i]:=false; end;

   { najkrastia cesta BA<->BA je jedina a je dlzky 0 }
   p[1]:=1; l[1]:=0;
```

```
   { Dijkstrov algoritmus }
   for i:=1 to N do begin
      min:=-1;
      { najdeme nedefinitivny vrchol s najkratsou l[j] ...}
      for j:=1 to N do
      if (not b[j]) and ((min=-1) or (l[j]<l[min])) then min:=j;
      { ... a prehlasime ho za definitivny }
      b[min]:=true;
      { a skratime vsetky nedefinitivne cesty cez neho }
      for j:=1 to N do
      if (not b[j]) and (a[min,j]<9999) then skrat(min,j);
   end;

   writeln('Najkratsich ciest je ',p[N],' a maju dlzku ',l[N],'.');
end.
```

Opravoval si MišoF. z Popradu
(max. 19 bodov)

4. O Miškovi na ľade

Úlohou bolo nájsť v danej matici núl a jednotiek obdĺžnik s najväčšou plochou, ktorý je tvorený samými nulami. Aby sme nemuseli ošetrovať okrajové prípady, okolo našej matice si do poľa zapíšeme samé jednotky – riešenie úlohy sa tým nezmení, ale my nebudeme musieť rozoberať prípady, či sme na kraji alebo nie. Aby nenastali nejasnosti, našu maticu budeme označovať A, pritom vstup bude na políčkach $A[1,1]$ až $A[M,N]$, prvá súradnica je riadok, druhá stĺpec.

Najjednoduchšie riešenie je vyskúšať všetky možnosti, kde leží ľavý horný roh (tých je MN), pre každú všetky možnosti, kde je pravý dolný (tých je $O(MN)$) a pre každý obdĺžnik skontrolovať (v čase $O(MN)$), či vyhovuje. Takto dostaneme riešenie s nie práve úžasným časom $O(M^3N^3)$. Trochu si môžeme pomôcť nasledovne: Spočítajme postupne pole B, kde $B[i,j]$ bude súčet políčok v poli A v obdĺžniku $[0,0]$ až $[i,j]$. Toto pole ľahko (ako?) spočítame v čase $O(MN)$. Ak teraz chceme zistiť, či je obdĺžnik s daným ľavým horným rohom $[x_1,y_1]$ a pravým dolným rohom $[x_2,y_2]$ tvorený samými nulami, stačí nám zistiť, či je súčet príslušných prvkov poľa A nula. Ale tento súčet ľahko určíme pomocou poľa B – je to $B[x_2,y_2]-B[x_1-1,y_2]-B[x_2,y_1-1]+B[x_1-1,y_1-1]$. (Prečo? Nakreslite si obrázok...) Takto nám na skontrolovanie obdĺžnika stačí čas $O(1)$, čím sme predchádzajúce riešenie zlepšili na $O(M^2N^2)$. Stále nič moc.

Prejdime našu maticu A po riadkoch a do poľa V si spočítajme pre každé políčko $[i,j]$ počet nulových políčok bezprostredne nad ním (vrátane neho). Teda ak $A[i,j]=1$, $V[i,j]=0$, inak $V[i,j]=V[i-1,j]+1$. Keď vieme hodnoty $V[i,*]$, môžeme pomocou nich spočítať najväčší obdĺžnik, ktorý končí v i-tom riadku, nasledovne: Postupne vyskúšame všetky možnosti pre stĺpec j_1, kde začína. Pre každú posúvame posledný stĺpec j_2 od j_1 až po kraj matice, pričom si pamätáme, akú ešte najvyššiu výšku náš obdĺžnik môže mať. Na začiatku je to $V[i,j_1]$, ak posunieme pravú hranicu na políčko, nad ktorým je menej núl ako aktuálna výška obdĺžnika, musíme ho zmenšiť. Zo všetkých takto nájdených obdĺžnikov vyberieme samozrejme ten najväčší.

Časová zložitosť práve ukázaného algoritmu je $O(MN^2)$ – pre každý spodný riadok obdĺžnika vyskúšame všetky možnosti pre ľavý a pravý stĺpec, pamäťová je $O(MN)$. Za takýto algoritmus sa dalo získať 13 bodov. Prípadne 14, ak ste si uvedomili, že z matíc A a V si vždy stačí pamätať posledné dva riadky, čím zlepšíme pamäťové nároky na $O(N)$.

Aby sme mohli nájsť lepšie riešenie, musíme si o našom obdĺžniku niečo uvedomiť. Pri každej jeho strane musí ležať nejaká jednotka, inak by sme ho mohli do tejto strany predĺžiť a dostali by sme väčší. My preto budeme prezerať len obdĺžniky, ktoré majú nad hornou stranou jednotku. Presnejšie pre každé políčko $[i,j]$ zodpovieme otázku: „Aký najväčší obdĺžnik má

spodný riadok i a má v stĺpci j jednotku tesne nad hornou stranou?“ Zjavne náš najväčší obdĺžnik v niektorom riadku končí a v niektorom stĺpci má jednotku tesne nad hornou stranou, takže takto ho naozaj nájdeme. A ako teda zodpovedať nami položenú otázku?

Okrem starého známeho poľa V budeme mať ešte dve polia L a R. Pole L nám pre políčko $[i, j]$ bude hovoriť, koľko najviac políčok doľava môže siahať obdĺžnik, ktorý má pravý dolný roh $[i, j]$ a výšku $V[i, j]$. Pole R nám bude hovoriť to isté, len doprava.

Keby sme mali spočítané V, L a R, našu otázku zodpovieme ľahko – pre $[i, j]$ má obdĺžnik výšku $V[i, j]$ a aby bol najväčší, musí siahať do oboch strán najviac ako sa s tou výškou dá, čiže bude mať šírku $L[i, j] + R[i, j] - 1$. Takže jediné, čo ostáva, je vedieť efektívne spočítať hodnoty v poliach L a R. Ukážeme len L, pre R to bude zjavne fungovať rovnako, len pôjdeme sprava namiesto zľava.

Predpokladajme, že máme hodnoty $V[i-1, *]$, $L[i-1, *]$, poďme spočítať hodnoty $L[i, *]$. Prechádzajme riadok zľava doprava, pričom si budeme pamätať dĺžku k súvislého úseku núl v ňom po políčko, kde práve sme. Ak $A[i, j] = 1$, $L[i, j] = 0$. Ak $V[i, j] = 1$, $L[i, j] = k$. Inak: Zjavne $L[i, j] \leq L[i - 1, j]$ – doľava to s výškou $V[i, j]$ určite nepôjde viac ako v predchádzajúcom riadku, lebo ak sa nezasekneme skôr, zasekneme sa tam, kde predtým. Podobne zjavne $L[i, j] \leq h$ – po h políčkach sa určite zasekneme na jednotke, ktorá je v i-tom riadku. A nie je ťažké si uvedomiť, že $\min(L[i-1, j], h)$ políčok doľava siahajúci obdĺžnik výšky $V[i, j]$ naozaj žiadnu jednotku neobsahuje. Preto toto je hľadaná hodnota $L[i, j]$.

Časová aj pamäťová zložitosť práve ukázaného algoritmu je $O(MN)$. Za takýto algoritmus sa dalo získať v zadaní sľúbených 18 bodov. No a tak sa plný počet musel posunúť na 19. Na plný počet ste si opäť museli uvedomiť, že z matíc A, V, L a R si vždy stačí pamätať posledné dva riadky, čím zlepšíme pamäťové nároky na $O(N)$.

Listing programu:

```
#include <stdio.h>

int A[2][100],V[2][100],L[2][100],R[2][100],pom[100];
int M,N,row,col;
int maxS=-1,x1,y1,x2,y2;

int main(void){
  scanf("%d %d ",&M,&N);
  for (col=0;col<N;col++) V[1][col]=L[1][col]=R[1][col]=0;
  for (row=0;row<M;row++) {
    // spracuvame vstup po riadkoch
    for (col=0;col<N;col++) {
      scanf("%d ",&A[row%2][col]);
      if (A[row%2][col]) V[row%2][col]=0;
        else V[row%2][col]=V[(row+1)%2][col]+1;
    }
    // spocitame L
    pom[0]=1-A[row%2][0];
    for (col=1;col<N;col++)
      if (A[row%2][col]) pom[col]=0; else pom[col]=pom[col-1]+1;
    for (col=0;col<N;col++) {
      if (A[row%2][col]) { L[row%2][col]=0; continue; }
      if (V[row%2][col]==1) { L[row%2][col]=pom[col]; continue; }
      L[row%2][col]=L[(row+1)%2][col]<pom[col]?L[(row+1)%2][col]:pom[col];
    }
    // spocitame R
    pom[N-1]=1-A[row%2][N-1];
    for (col=N-2;col>=0;col--)
      if (A[row%2][col]) pom[col]=0; else pom[col]=pom[col+1]+1;
    for (col=0;col<N;col++) {
```

```
      if (A[row%2][col]) { R[row%2][col]=0; continue; }
      if (V[row%2][col]==1) { R[row%2][col]=pom[col]; continue; }
      R[row%2][col]=R[(row+1)%2][col]<pom[col]?R[(row+1)%2][col]:pom[col];
    }
    // pozrieme ci nemame lepsie riesenie
    for (col=0;col<N;col++) if (!A[row%2][col])
      if (V[row%2][col]*(L[row%2][col]+R[row%2][col]-1)>maxS) {
        maxS=V[row%2][col]*(L[row%2][col]+R[row%2][col]-1);
        x1=row-V[row%2][col]+1; x2=row;
        y1=col-L[row%2][col]+1; y2=col+R[row%2][col]-1;
      }
  }
  // a vypiseme vysledok
  if (maxS==-1) printf("Vsade je sneh.\n"); else
    printf("Najvacsi lad: [%d,%d]-[%d,%d]\n",x1+1,y1+1,x2+1,y2+1);
  return 0;
}
```

opravoval Martin zo Spišskej

5. O celulárnych automatoch II

(max. 15 bodov)

Takmer všetci, ktorí ste tento príklad poslali, ste ho mali správne. No aj napriek tomu som väčšine z vás nemohol dať plný počet bodov, lebo ste podcenili dôležitosť popisu vášho riešenia. Vašou úlohou nebolo iba napísať prechodovú funkciu, ale aj podrobne vysvetliť, ako váš automat funguje a aspoň sa pokúsiť dokázať, že váš automat robí naozaj to čo chcete. Takže za správne a precízne zdôvodnené riešenie ste mohli získať najviac 15 bodov. Za veľmi stručný, ba dokonca až chýbajúci popis riešenia ste mohli získať aj -4 body a za drobné chybičky v prechodovej funkcii po -1 bode.

Číslo zapísane v dvojkovej sústave budeme násobiť troma postupne, od najmenej významných bitov až po najvýznamnejšie. Teda ako prvý spracujeme najmenej významný bit. Vynásobíme ho tromi a zapamätáme si, aký bude tento bit v konečnom výsledku a aké pretečenie do vyšších rádov nastalo. Potom každú ďalšiu sekundu zoberieme najpravejší nespracovaný bit, opäť ho vynásobíme tromi, pripočítame k nemu pretečenie do vyšších rádov z násobenia jeho pravého suseda a zapamätáme si výsledok i nové pretečenie[6]. Aby sme počas výpočtu vedeli, ktorý bit bol do danej sekundy násobený ako posledný, automat nesúci tento bit prejde po vynásobnení do špeciálneho stavu Φ^{β}_{γ}, kde $\gamma \in \{0,1\}$ je výsledok a $\beta \in \{0,1,2\}$ pretečenie do vyšších rádov. V ďalšej sekunde automat prejde z tohto stavu do výsledného stavu γ. Dôležité je si uvedomiť, že v každej sekunde výpočtu bude práve jeden automat v niektorom zo stavov Φ^{β}_{γ}.

Automaty budú nadobúdať stavy z množiny $K = \{0, 1, \Phi^0_0, \Phi^1_0, \Phi^2_0, \Phi^0_1, \Phi^1_1, \Phi^2_1\}$ a ich prechodová funkcia bude:

$$\left.\begin{array}{rrcl} (1) & x\ \alpha\ \Phi^{\beta}_{\gamma} & \to & \Phi^{(3\alpha+\beta)\ div\ 2}_{(3\alpha+\beta)\ mod\ 2} \\ (2) & x\ \alpha\ * & \to & \Phi^{(3\alpha)\ div\ 2}_{(3\alpha)\ mod\ 2} \\ (3) & x\ \Phi^{\beta}_{\gamma}\ y & \to & \gamma \\ (4) & z\ \alpha\ y & \to & \alpha \end{array}\right\} \quad \begin{array}{rcl} x, y & \in & \{\$, 0, 1\} \\ \alpha, \gamma & \in & \{0, 1\} \\ \beta & \in & \{0, 1, 2\} \\ z & \in & K \end{array}$$

Aby ste videli, ako automat pracuje, zamyslite sa nad týmto príkladom výpočtu: $\$0010111* \to \$001011\Phi^1_1\$ \to \$00101\Phi^2_0 1\$ \to \$0010\Phi^2_1 01\$ \to \$001\Phi^1_0 101\$ \to \$00\Phi^2_0 0101\$ \to \$0\Phi^1_0 00101\$ \to \$\Phi^0_1 000101\$ \to \$1000101\$$.

[6] Toto pretečenie nebude nikdy vačšie než 2, rozmysli si prečo.

Dokážeme[7], že automat vypočíta torojnásobok zadaného čísla. Očíslujme si automaty číslami 1 až n tak, aby najpravejší automat mal číslo 1, jeho sused 2, až najľavejší číslo n. Pred začatím výpočtu (tj. v nultej sekunde) sú všetky automaty v stave 0 alebo 1.

Najprv ukážeme, že v i-tej sekunde ($1 \leq i \leq n$) sú všetky automaty okrem i-tého v stave 0 alebo 1 a i-tý automat v niektorom zo stavov Φ_γ^β, kde $\beta \in \{0,1\}$ a $\gamma \in \{0,1,2\}$. Toto tvrdenie dokážeme indukciou. V prvej sekunde sa môže zmeniť iba stav prvého automatu. Nech ten je α, podľa pravidla (2) sa zmení na stav $\Phi_{(3\alpha)\ mod\ 2}^{(3\alpha)\ div\ 2}$. Ostatné automaty sa podľa pravidla (4) nezmenia. Teraz predpokladajme, že dokazované tvrdenie v i-tej ($1 \leq i < n$) sekunde platí. To znamená, ze i-tý automat je v niektorom zo stavov Φ_γ^β a ostatné automaty sú v stave 0 alebo 1. Podľa pravidla (3) i-tý automat prejde do stavu γ, teda 0 alebo 1 a $(i+1)$-vý automat (taký existuje, lebo $i < n$), ak bol v i tej sekunde v stave α, prejde podľa (1) do stavu $\Phi_{(3\alpha+\beta)\ mod\ 2}^{(3\alpha+\beta)\ div\ 2}$. Ostatné automaty sa podľa pravidla (4) nemenia. Preto vsetky automaty okrem i-tého budú v $(i+1)$-vej sekunde v stave 0 alebo 1 a i-tý automat v niektorom zo stavou Φ_γ^β, kde $\beta \in \{0,1\}$ a $\gamma \in \{0,1,2\}$.

V n-tej sekunde je n-tý automat v stave Φ_γ^β a ostatné automaty sú v stave 0 alebo 1. Je zrejmé, že n-tý automat podľa pravidla (3) prejde do stavu γ a ostatné automaty podľa (4) svoj stav nezmenia. Preto v $(n+1)$-vej sekunde budu všetky automaty v stave 0 alebo 1 a celý výpočet sa ukončí.

Teraz by sme ešte potrebovali dokázať, že naozaj vypočítame trojnásobok vstupného čísla. Preto opäť indukciou podľa i ukážeme, že po i sekundách je posledných $i-1$ bitov správne vynásobených, prvých $n-i$ bitov rovnakých ako na začiatku a i-ty automat je v stave zodpovedajúcom správnemu bitu a prenosu. Keby sme chceli byť veľmi formálni, mohlo by to vyzerať napríklad takto:

Označme A_j^i stav i-tého automatu v j-tej sekunde. Označme

$$\Upsilon_i = 3 \cdot \sum_{j=i+1}^{n} A_i^j 2^{j-1} + \beta_i 2^i + \gamma_i 2^{i-1} + \sum_{j=1}^{i-1} A_i^j 2^{j-1} \quad (\text{kde } A_i^i = \Phi_{\gamma_i}^{\beta_i})$$

(Neformálne: Υ_i dostaneme tak, že sčítame číslo tvorené najnižšími i bitmi – už vynásobenými, prenos a trojnásobok čísla tvoreného $n-i$ najvyššími bitmi a i nulami – ešte nevynásobená časť.) Dokážeme indukciou podľa i, že všetky hodnoty Υ_i ($0 \leq i \leq n+1$) sú rovnaké. Pritom $\Upsilon_{n+1} = \sum_{i=1}^{n} A_{n+1}^i 2^{i-1}$ je výsledok výpočtu automatu a $\Upsilon_0 = 3 \cdot \sum_{i=1}^{n} A_0^i 2^{i-1}$ je trojnásobok pôvodného čísla.

Ako sme už zistili, v prvej sekunde sa pôvodný stav A_0^1 zmení na stav $A_1^1 = \Phi_{(3A_0^1)\ mod\ 2}^{(3A_0^1)\ div\ 2}$ a ostatné stavy sa nezmenia (tj. $A_1^i = A_0^i$, kde $1 < i \leq n$). Môžeme počítať $\Upsilon_1 = 3 \cdot \sum_{j=2}^{n} A_1^j 2^{j-1} + (3A_0^1)\ div\ 2 \cdot 2^1 + (3A_0^1)\ mod\ 2 \cdot 2^0 + \sum_{j=1}^{0} A_1^j 2^{j-1} = 3 \cdot \sum_{j=2}^{n} A_0^j 2^{j-1} + 3A_0^1 = 3 \cdot \sum_{j=1}^{n} A_0^j 2^{j-1} = \Upsilon_0$.

Rovnako v $(n+1)$-vej sekunde sa n-tý automat zmení zo stavu $A_n^n = \Phi_{\gamma_n}^{\beta_n}$.[8] na stav $A_{n+1}^n = \gamma_n$ a ostatné automaty svoj stav nezmenia (tj. $A_{n+1}^i = A_n^i$, kde $1 \leq i < n$). Preto $\Upsilon_n = 3 \cdot \sum_{j=n+1}^{n} A_n^j 2^{j-1} + \beta_n 2^n + \gamma_n 2^{n-1} + \sum_{j=1}^{n-1} A_n^j 2^{j-1} = A_{n+1}^n + \sum_{j=1}^{n-1} A_{n+1}^j 2^{j-1} = \sum_{j=1}^{n} A_{n+1}^j 2^{j-1} = \Upsilon_{n+1}$.

Nech $1 \leq i < n$, potom v $(i+1)$-vej sekunde, ako sme už ukázali, sa stav i-tého automatu zmení z $A_i^i = \Phi_{\gamma_i}^{\beta_i}$ na stav $A_{i+1}^i = \gamma_i$, stav $(i+1)$-vého automanu z A_i^{i+1} na $A_{i+1}^{i+1} = \Phi_{\gamma_{i+1}}^{\beta_{i+1}} = \Phi_{(3A_i^{i+1}+\beta_i)\ mod\ 2}^{(3A_i^{i+1}+\beta_i)\ div\ 2}$. Stavy ostatných automatov ostanú nezmenene (tj. $A_{i+1}^j = A_i^j$,

[7] Vy ste samozrejme nemuseli písať až takto podrobný dôkaz, ale niečo napísať ste predsa len mali. ;-)

[8] Podľa zadania môžeme predpokladať, že na najvyššom bite k pretečeniu nedôjde, preto $\beta_n = 0$.

kde $1 \leq j \leq n$ a $i \neq j \neq i{+}1$). A platí $\Upsilon_i = 3 \cdot \sum_{j=i+1}^{n} A_i^j 2^{j-1} + \beta_i 2^i + \gamma_i 2^{i-1} + \sum_{j=1}^{i-1} A_i^j 2^{j-1} = 3 \cdot \sum_{j=i+2}^{n} A_{i+1}^j 2^{j-1} + 3A_i^{i+1} 2^i + \beta_i 2^i + A_{i+1}^i 2^{i-1} + \sum_{j=1}^{i-1} A_{i+1}^j 2^{j-1} = 3 \cdot \sum_{j=i+2}^{n} A_{i+1}^j 2^{j-1} + (3A_i^{i+1} + \beta_i) \cdot 2^i + \sum_{j=1}^{i} A_{i+1}^j 2^{j-1} = 3 \cdot \sum_{j=i+2}^{n} A_{i+1}^j 2^{j-1} + (3A_i^{i+1} + \beta_i)\ div\ 2 \cdot 2^{i+1} + (3A_i^{i+1} + \beta_i)\ mod\ 2 \cdot 2^i + \sum_{j=1}^{i} A_{i+1}^j 2^{j-1} = 3 \cdot \sum_{j=i+2}^{n} A_{i+1}^j 2^{j-1} + \beta_{i+1} 2^{i+1} + \gamma_{i+1} 2^i + \sum_{j=1}^{i} A_{i+1}^j 2^{j-1} = \Upsilon_{i+1}$.

Dokázali sme, že výpočet sa vždy zastaví a tiež, že $\Upsilon_0 = \Upsilon_1 = \ldots = \Upsilon_n = \Upsilon_{n+1}$, výsledok výpočtu je teda rovný trojnásobku zadaného čísla. Preto automat je korektný a rieší zadaný problém.

Ako z dôkazu vyplýva, každý výpočet trvá práve $n + 1$ sekúnd, kde n je dĺžka vstupu (dĺžka binarného zápisu vstupného čísla). Teda časová zložitosť je $O(n)$.

Výsledková listina po 2. kole kategórie KSP

	Meno a priezvisko	Škola	Trieda		21	22	23	24	25	Σ
1	Závodný Jakub	Gym. Grösslingová BA	3	54	15	12	15	13	13	122
1	Šatka Milan	Gym. Liptovský Hrádok	4	52	15	6	15	19	15	122
3	Tekeľ Jakub	Gym. Jura Hronca BA	3	52	15	13	13	13	14	120
4	Jančuška Marek	Gym. Párovská Nitra	3	51	8	13	15	13	15	115
5	Štefanek Anton	Gym. Jura Hronca BA	3	36	15	13	13	18	10	105
6	Poláček Lukáš	Gym. K. Štúra Modra	3	35	13	12	15	14	13	102
7	Kováč Jakub	Gym. Jura Hronca BA	3	22	15	13	15	18	15	98
8	Burda Milan	Gym. Jura Hronca BA	3	25	15	7	13	18	14	92
9	Choma Martin	Gym. Stará Ľubovňa	4	31	15		15	16	12	89
10	Lipková Juliana	Gym. Jura Hronca BA	4	35	15	8	4	6	13	81
11	Burger Michal	Gym. Grösslingová BA	3	48	8		10		11	77
11	Repovský Michal	Gym. Komenského Trebišov	3	20	15	7	10	18	7	77
13	Smažáková Dana	Gym. Jura Hronca BA	3	28	15		12	15		70
14	Rejda Martin	Gym. Grösslingová BA	3	25	15		15	13		68
15	Lenhardt Rastislav	Gym. Jura Hronca BA	3	36	15		12	4		67
16	Štolc Miroslav	Gym. Párovská Nitra	3	17	3	8	8	13	13	62
17	Šmitala František	Gym. Cyrila a Metoda Nitra	3	19	15		8	13		55
18	Klíma Jaroslav	Gym. Jura Hronca BA	4	49						49
19	Kevický Michal	Gym. Grösslingová BA	3	23	3		8		14	48
19	Nánási Michal	Gym. Jura Hronca BA	2	0	15	11	12	10		48
19	Šufliarsky Peter	Gym. Nové Zámky	3	22	11			12	3	48
22	Trejbal Ivan	Gym. Jura Hronca BA	3	15				18	13	46
23	Šomlo Ivan	Gym. Mládežnícka Šahy	3	44						44
24	Kuchynárová Mariana	Gym. Jura Hronca BA	3	22	15		4			41
25	Kollár Juraj	Gym. Jura Hronca BA	2	11	9		8	10		38
26	Hanulová Anna	Gym. Jura Hronca BA	3	0	15		7	14		36
27	Ružička Peter	Gym. Jura Hronca BA	2	12	2		8	9		31
28	Palenčár Ján	Gym. Malá Hora Martin	3	25						25
29	Konečný Štefan	Gym. Jura Hronca BA	4	22						22
30	Čulák Radovan	Gym. Golianova Nitra	3	0	13			3		16
31	Baláž Miroslav	Gym. Jura Hronca BA	3	12						12
32	Janík Oliver	Gym. L. Stöckela Bardejov	3	8						8
32	Kotrbčík Michal	Gym. Jura Hronca BA		0			8			8
34	Beňo Michal	Gym. Kremnica	3	7						7
35	anonym2	Neznama skola		0	2					2