Slovenská komisia Matematickej olympiády
FMFI UK, Mlynská dolina, 842 48 Bratislava 4

50. ročník Matematickej olympiády, školský rok 2000/2001

Riešenia úloh celoštátneho kola kategórie P
2. súťažný deň

P-III-4 *(Formátovanie textu)*

Riešenie tejto úlohy je založené na dynamickom programovaní. Pre každé slovo si budeme pamätať, kde je najlepšie skončiť predchádzajúci riadok, ak by toto slovo bolo na konci riadku. Tiež si budeme pamätať minimálny celkový počet trestných bodov pri tomto rozdelení. Keď zrátame najlepší koniec predchádzajúceho riadku pre posledné slovo, vieme zo zapamätaných informácií ľahko zrekonštruovať rozdelenie celého textu do riadkov – z informácií pre posledné slovo vieme, ktoré slová ležia v poslednom riadku. Potom ale vieme, ktorým slovom končí predposledný riadok, preň už tiež máme zrátané optimálne rozdelenie a pokračujeme analogicky. Stačí už len správne doplniť medzery medzi vypisované slová. To urobíme tak, že keď máme umiestniť M medzier medzi $S+1$ slov, tak M mod S dvojíc slov oddelíme $(M \text{ div } S)+1$ medzerami a $S-(M \text{ mod } S)$ oddelíme M div S medzerami. Tým sa počty medzier medzi ľubovoľnými dvoma dvojicami susedných slov líšia najviac o 1 a celkový počet medzier je tiež správny. Formátovanie posledného riadku a riadku s jedným slovom sú triviálne.

A teraz už k hlavnej časti algoritmu – ako rýchlo spočítať, kde je najlepšie skončiť predchádzajúci riadok, ak bude aktuálne slovo na konci riadku? Túto informáciu budeme postupne počítať od prvého slova. Pri prvom slove nastavíme počet trestných bodov na 0. Ak máme zrátané hodnoty pre prvých N slov, začnime ju rátať pre $(N+1)$. slovo. Pôjdeme od N-tého slova smerom k začiatku textu. Pre každé slovo si zrátame počet trestných bodov za riadok, ktorý by začínal za týmto slovom a končil $(N+1)$. slovom. (Pokiaľ je $(N+1)$. slovo posledné v texte, bude ním ukončený riadok posledný, a tak hodnotiacu funkciu musíme informovať, že hodnotí posledný riadok.) K tomuto počtu trestných bodov ešte prirátame minimálny počet trestných bodov za predchádzajúci text (ten už máme zrátaný a uložený pri slove, ktoré skúšame ako koniec predposledného riadku). Z týchto hodnôt si vyberieme minimálnu, zapamätáme si ju a slovo, ktorým pri nej končil predposledný riadok. Keď už je posledný riadok pridlhý (hodnotiaca funkcia vráti ∞), vieme, že lepšie rozdelenie už nenájdeme. Môžeme teda ísť rátať hodnoty pre ďalšie slovo. Algoritmus má časovú zložitosť $O(T+W\cdot L)$, kde T je dĺžka textu, W je počet slov a L je dĺžka riadku. Pamäťová zložitosť algoritmu je $O(T)$. Správnosť algoritmu vyplýva z popisu.
Program je priamou implementáciou algoritmu:

```
#include <stdio.h>
#include <stdlib.h>

#define INPUT "format.in"
```

```
#define OUTPUT "format.out"

#define TEXTLEN 10000    /* Maximalni delka textu */
#define MAXWORDS 5000    /* Maximalni pocet slov v textu */
#define MAXLINELEN 100    /* Maximalni delka vstupni radky */
#define MAXLINES 5000    /* Maximalni pocet radek ve vystupu */

#define LINEPENALTY 10    /* Cena radky */
#define LASTLINESMALLPEN 3  /* Cena za maly posledni radek */
#define SINGLEWORDPEN 20  /* Penalta za jedno slovo na radku */
#define INFTYPEN 30000    /* Nekonecna penalta */

typedef unsigned long price_t;

int Width;      /* Sirka radku */
char Text[TEXTLEN];    /* Text nacteny ze souboru */
int Words[MAXWORDS];    /* Text rozsekany na slova */
int WCnt;      /* Pocet slov */
price_t WrapPrice[MAXWORDS];  /* Ohodnoceni textu, pokud zalomime za timto slovem */
int WrapPos[MAXWORDS];    /* Pozice, kde jsme zalomili, kdyz jsme pridavali toto slovo */

/* Ohodnoti radek */
int LinePrice(int WordLen, int Words, int Last)
{
  int Pts = Width - WordLen - Words + 1;
  int Base = LINEPENALTY;

  if (Pts < 0)
    return INFTYPEN;
  if (Last)
    return LINEPENALTY + (((WordLen+Words-1)*4 < Width) ? LASTLINESMALLPEN : 0);
  if (Words == 1)
    Base += SINGLEWORDPEN;
  return Pts * Pts + Base;
}

/* Nacte vstup a rozdeli ho na slova */
void ProcessInput(void)
{
  FILE *In;
  char Buf[MAXLINELEN];  /* Buffer na radku */
  int i, TPos = 0, WPos = 0;  /* ; Pozice v bufferu na text ; Cislo aktualniho slova */
  int SWPos = 0;    /* Pozice pocatku slova */

  if (!(In = fopen(INPUT, "r")))
  {
    puts("Can't open input file.");
    exit(1);
  }
  /* Nacteme delku radku */
  fgets(Buf, MAXLINELEN, In);
  sscanf(Buf, "%d", &Width);

  while (fgets(Buf, MAXLINELEN, In))
```

```
  {
    for (i = 0; Buf[i] != '\0'; i++, TPos++)
    {
      if (Buf[i] == '\n')
        Buf[i] = ' ';
      Text[TPos] = Buf[i];
      if (Buf[i] == ' ')
      {
        Words[WPos++] = TPos - SWPos;
        SWPos = TPos + 1;
      }
    }
  }
  fclose(In);
  WCnt = WPos;
}

void FindBestSep(void)
{
  int i, j;
  price_t Min, Price;  /* Minimalni dosazene ohodnoceni; Cena aktualniho zlomu */
  int WordsLen, MinPos;  /* Celkova delka slov na radce; Pozice minimalniho zalomeni  */

  WrapPrice[0] = 0;
  WrapPos[0] = 0;

  for (i = 0; i < WCnt; i++)  /* Postupne pridavame jednotliva slova */
  {
    Min = INFTYPEN;
    MinPos = 0;
    WordsLen = 0;
    for (j = i; j >= 0; j--)  /* Vyzkousime vsechna mozna zalomeni */
    {
      WordsLen += Words[j];
      Price = LinePrice(WordsLen, i - j + 1, i == WCnt - 1);
      if (Price == INFTYPEN)  /* Uz jsem prekrocili velikost radku? */
        break;
      Price += WrapPrice[j];
      if (Price < Min)
      {
        Min = Price;
        MinPos = j;
      }
    }
    WrapPrice[i+1] = Min;
    WrapPos[i+1] = MinPos;
  }
}

/* Vytiskne dalsi slovo */
void PrintWord(FILE *Out)
{
  static int WPos = 0;    /* Pozice slova k vypsani */
  int WStart;
```

```
  for (WStart = WPos; Text[WPos] != ' '; WPos++);
  fwrite(Text + WStart, 1, WPos - WStart, Out);
  WPos++;
}

/* Vytiskne radku */
void PrintLine(FILE *Out, int First, int Cnt)
{
  int i, j;
  int Len = 0, TotSpc;    /* Pocet znaku na radce; Celkovy pocet mezer */
  int Spc;      /* Pocet mezer v aktualni mezere */

  /* Spocteme pocet znaku ve slovech */
  for (i = 0; i < Cnt; i++)
    Len += Words[First+i];
  TotSpc = Width - Len;
  if (Cnt == 1)      /* Jedno slovo? */
  {
    for (j = 0; j < TotSpc; j++)
      fputc(' ', Out);
  }
  else
    for (i = 0; i < Cnt-1; i++)
    {
      PrintWord(Out);    /* Vytiskne dalsi slovo */
      /* Spocteme a vytiskneme potrebny pocet mezer */
      Spc = TotSpc / (Cnt - 1) + (i < TotSpc % (Cnt-1));
      for (j = 0; j < Spc; j++)
        fputc(' ', Out);
    }
  PrintWord(Out);    /* Jeste vytiskneme posledni slovo */
  fputc('\n', Out);
}

/* Vypiseme nejlepsi vysledek */
void PrintBestText(void)
{
  FILE *Out;
  int Lines = 0, ActWord;  /* Pocet radek vysledneho textu; Aktualni slovo */
  int LB[MAXLINES], i;    /* Pozice jednotlivych zalomeni */

  /* Zjistime pozice jednotlivych zalomeni */
  for (ActWord = WCnt; ActWord > 0; Lines++, ActWord = WrapPos[ActWord])
    LB[Lines] = ActWord;
  LB[Lines] = 0;
  if (!(Out = fopen(OUTPUT, "w")))
  {
    puts("Can't open output file.");
    exit(1);
  }
  /* Nechame vytisknout radku */
  for (i = Lines - 1; i > 0; i--)
    PrintLine(Out, LB[i+1], LB[i] - LB[i+1]);
```

```
  /* Ted jeste vytiskneme posledni radku */
  for (i = LB[1]; i < LB[0] - 1; i++)
  {
    PrintWord(Out);
    fputc(' ', Out);
  }
  PrintWord(Out);
  fputc('\n', Out);

  fclose(Out);
}

int main(void)
{
  ProcessInput();  /* Nacte vstup a rozdeli ho na slova */
  FindBestSep();  /* Nalezneme nejlepsi rozdeleni na radky */
  PrintBestText();  /* Vypise text podle spoctenych zalomeni */
  return 0;
}
```

P-III-5 *(Okružné trasy)*

Najskôr si rozmyslime, že platí nasledovné tvrdenie: Trasy liniek mestom sa dajú navrhnúť práve vtedy, keď z každej križovatky vychádza párny počet ulíc.

Keby sme si trasy rôznych liniek vyznačili na pláne mesta rôznymi farbami, potom by každá z nich tvorila cyklus. Každá ulica by mala jednoznačne určenú svoju farbu. Pre každú križovatku a každú farbu by platilo, že z tejto križovatky buď nevychádza žiadna ulica tejto farby, alebo z nej vychádzajú práve dve ulice tejto farby. Preto je počet ulíc vychádzajúcich z každej križovatky naozaj párny.

Teraz si naopak rozmyslime, že ak z každej križovatky vychádza párny počet ulíc, potom vieme navrhnúť trasy liniek tak, aby spĺňali požiadavky zo zadania. Postupne ofarbujeme ulice v meste tak, aby ulice rovnakej farby tvorili cyklus (teda zodpovedali nejakej linke). Ulice, ktoré sme ofarbili, už nebudeme považovať za súčasť mesta. Tým zmenšíme počet ulíc vychádzajúci z ľubovoľnej križovatky o párne číslo (o nula alebo o dve), takže počet ulíc vychádzajúcich z každej križovatky bude naďalej párny. Vyberme si nejakú križovatku v meste a označme si ju na mape (položme na ňu kamienok). Vyberme sa z tejto križovatky po ľubovoľnej (doteraz neofarbenej) ulici a položme kamienok na križovatku, na ktorú sme prišli. Z každej križovatky sa vždy dá pokračovať aspoň jednou ulicou – po jednej sme sem prišli a keďže neofarbených ulíc z nej je párny počet, sú aspoň dve a tou druhou môžeme pokračovať ďalej. Skončíme, keď by sme mali na nejakú križovatku položiť druhý kamienok – prišli sme sem druhýkrát, a teda našli cyklus. Tento cyklus ofarbíme nejakou doteraz nepoužitou farbou a prehlásime ho za novú linku. Kamienky odstránime z mapy a celý proces opakujeme, kým sa nám neminú neofarbené hrany.

Predchádzajúci dôkaz nám dáva rovno aj návod na vytvorenie algoritmu, ktorý rieši zadanú úlohu. Najskôr overíme, či z niektorej križovatky nevychádza nepárny počet ulíc. Ak taká existuje, rovno vypíšeme `Neda sa`. V opačnom prípade začneme aplikovať postup z predchádzajúceho odstavca. Kamienky samoz-

rejme nahradíme nastavovaním vhodného príznaku v programe. Keď nájdeme cyklus, príslušné ulice vymažeme z mapy a cyklus vypíšeme na výstup. Aby sme ušetrili čas, necháme kamienky na križovatkách, ktoré sú medzi križovatkou, kde sme začali a križovatkou, kde sme mali položiť druhý kamienok. Na tieto križovatky by sme totiž mohli položiť kamienky aj v meste, ktoré vzniklo vynechaním práve nájdeného cyklu. Kladenie kamienkov budeme realizovať jednoduchou rekurzívnu funkciou. Zostáva vyriešiť, ako práve ofarbené ulice rýchlo odstraňovať z mapy mesta, uloženej v pamäti.

Ulice spájajúce dve rovnaké križovatky si budeme pamätať ako jednu ulicu s uvedením počtu ulíc, ktoré spájajú tie isté dve križovatky. Ulice si uložíme do dvojrozmerného poľa. Jeden jeho index bude predstavovať číslo križovatky a druhý poradové číslo ulice vychádzajúcej z danej križovatky. Rozmery tohto poľa teda budú počet križovatiek $\times$ maximálny počet ulíc, vychádzajúcich z jednej križovatky. Pre každú križovatku máme uvedené všetky ulice, ktoré z nej vychádzajú. Pre každú si pamätáme, kam vedie, koľko ulíc tieto dve križovatky v meste spája a index do tohto poľa určujúci, kde sú informácie o tejto ulici uložené pri križovatke, ktorá je na jej opačnom konci. Príslušnú ulicu vymažeme jednoducho tak, že upravíme informácie o nej pri oboch koncových križovatkách, ktoré spája, čo vieme spraviť v konštantnom čase, lebo si pamätáme jej index pri druhej križovatke.

Pamäťové aj časové nároky nášho algoritmu sú $O(N + M)$, kde N je počet križovatiek v meste a M je počet ulíc v meste. Pamäťové nároky sa reprezentáciou len jednej z hrán spájajúcej dve konkrétne križovatky (viď predchádzajúci odsek) dajú znížiť na $O(N + H)$, kde H je maximálny počet ulíc takých, že žiadne dve nespájajú tie isté dve križovatky.

```
program okruzni; { P-III-5 }
const MAXN=120;    { Maximální hodnoty dle zadání }
      MAXD=MAXN-1;  { ... počet ulic z jedné křižovatky }
      MAXM=200;    { ... počet paralelních ulic mezi dvěma křižovatkami }
type tkriz=byte;  { typy pro data }
     tulic=byte;
     tmult=byte;
type ulice=record  { záznam o skupině paralelních ulic }
             kam:tkriz;
             ind:tulic;
      mul:tmult;
           end;
var mapa:array[1..MAXN,1..MAXD] of ulice;
      { mapa města }
    ulicdo:array[1..MAXN] of tulic;
          { počet neparalelních ulic do jedné křižovatky }
    ulicsudy:array[1..MAXN] of boolean;
          { je počet ulic vycházejících z křižovatky sudý? }
    navst:array[1..MAXN] of boolean;
          { "kamínky" při budování nové linky }
    kriz:tkriz;
    ulic:longint;
    i,j:tkriz;
    k:longint;
```

```
   vstup,vystup:text;
 procedure vybuduj(od:tkriz;var posl:tkriz);  { kamínkovací procedura }
 var pokr:tkriz;
 begin
  if navst[od] then
   begin
    posl:=od; write(vystup,od,' ')
   end
  else
   repeat
    navst[od]:=true;
    pokr:=mapa[od,ulicdo[od]].kam;
    if mapa[od,ulicdo[od]].mul>1 then
     begin
      dec(mapa[od,ulicdo[od]].mul);
      dec(mapa[pokr,mapa[od,ulicdo[od]].ind].mul);
     end
    else
     begin
      mapa[pokr,mapa[od,ulicdo[od]].ind]:=mapa[pokr,ulicdo[pokr]];
      mapa[mapa[pokr,ulicdo[pokr]].kam,mapa[pokr,ulicdo[pokr]].ind].ind:=
       mapa[od,ulicdo[od]].ind;
      dec(ulicdo[pokr]);
      dec(ulicdo[od]);
     end;
    vybuduj(pokr,posl);
    navst[od]:=false;
    if posl<>od then
     begin
      write(vystup,od,' '); exit
     end;
    writeln(vystup,od)
   until ulicdo[od]=0
 end;
begin
 assign(vstup,'okruzni.in');
 assign(vystup,'okruzni.out');
 reset(vstup); rewrite(vystup);
 { Nejprve načteme vstup ... }
 readln(vstup,kriz,ulic);
 for i:=1 to kriz do
  begin
   ulicdo[i]:=0;
   navst[i]:=false;
   ulicsudy[i]:=true;
  end;
 for k:=1 to ulic do
  begin
   readln(vstup,i,j);
   ulicsudy[i]:=not ulicsudy[i];
   ulicsudy[j]:=not ulicsudy[j];
   if (ulicdo[i]>0) and (mapa[i,ulicdo[i]].kam=j) then
    begin
     inc(mapa[i,ulicdo[i]].mul);
```

```
     inc(mapa[j,ulicdo[j]].mul);
    end
   else
    begin
     inc(ulicdo[i]);
     inc(ulicdo[j]);
     mapa[i,ulicdo[i]].kam:=j;
     mapa[i,ulicdo[i]].ind:=ulicdo[j];
     mapa[i,ulicdo[i]].mul:=1;
     mapa[j,ulicdo[j]].kam:=i;
     mapa[j,ulicdo[j]].ind:=ulicdo[i];
     mapa[j,ulicdo[j]].mul:=1;
    end
  end;
 { Má úloha řešení? }
 for i:=1 to kriz do
  if not ulicsudy[i] then
   begin
    writeln(vystup,'Nelze');
    close(vstup); close(vystup);
    halt
   end;
 { Pokud ano, tak ho nalezneme ... }
 for i:=1 to kriz do
  if ulicdo[i]>0 then
   vybuduj(i,j);
 close(vstup); close(vystup)
end.
```

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

50. ROČNÍK MATEMATICKEJ OLYMPIÁDY

Vzorové riešenia celoštátneho kola kategórie P

Vydala IUVENTA – zariadenie pre voľný čas detí, mládeže i dospelých MŠ SR
pre vnútornú potrebu Ministerstva školstva SR
Programom TEX sadzbu pripravil Michal Forišek

Autori príkladov: Jan Kára
Daniel Kráľ
Martin Mareš