INFORMATIKA

Lyžařské středisko
(Úlohy z MO kategorie P, 35. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

V jedné soutěžní úloze krajského kola 65. ročníku Matematické olympiády kategorie P (školní rok 2015/16) jsme se pokusili vyřešit problém lyžaře, který si v zimním středisku vybírá trasu s ohledem na rozmístění bufetů u sjezdovek. Jedná se zjevně o úlohu grafovou, v níž ale nestačí jenom mechanicky použít některý ze známých grafových algoritmů. Nějaké neefektivní řešení založené na zkoušení všech možností hrubou silou vymyslel při soutěži skoro každý, poměrně málo řešitelů však našlo jednoduché a efektivní řešení, které si zde v článku ukážeme. Z celkových 67 účastníků loňských krajských kol MO kategorie P řešilo tuto úlohu 61 studentů, z nich ji pouze 12 vyřešilo zcela správně a dalších 8 víceméně správně (získali aspoň 7 bodů z 10 možných).

Pro pořádek ještě poznamenejme, že všechny soutěžní úlohy 65. ročníku MO kategorie P připravili naši slovenští kolegové z Fakulty matematiky, fyziky a informatiky Univerzity Komenského v Bratislavě. Zadání úlohy uvádíme v českém překladu jenom s mírnými textovými úpravami.

* * * * * * * * * * * *

Zimní lyžařské středisko se skládá z n lokalit očíslovaných od 0 do $n-1$ a je zde také jedna sedačková lanovka. Spodní stanice lanovky je umístěna v lokalitě 0, horní stanice je v lokalitě 1. Všechny lokality mají navzájem různé nadmořské výšky, tyto výšky ale neznáme. V některých lokalitách jsou umístěny bufety, kde si lyžaři mohou zakoupit občerstvení.

V lyžařském středisku je celkem z sjezdovek. Každá sjezdovka vede směrem dolů z jedné lokality do druhé. Občas se mohou některé sjezdovky křížit, ale všechna taková křížení jsou řešena mimoúrovňově (pomocí tunelů). Přejet z jedné sjezdovky na druhou je tedy možné jedině v lokalitě, kde první sjezdovka končí a druhá začíná.

Soutěžní úloha

Úkol A (3 body)

Lyžař chce v lokalitě 0 nasednout na lanovku, nechat se vyvézt do lokality 1 a odtud sjet nějakou posloupností sjezdovek zpět do lokality 0. Cestou chce navštívit právě jeden bufet. (Může ovšem projet okolo jiných bufetů, v nichž se nezastaví.) Napište program, který určí, kolik bufetů má lyžař na výběr.

Úkol B (7 bodů)

Lyžař chce podniknout stejnou jízdu jako v úkolu A, tentokrát se ale chce cestou postupně zastavit v co nejvíce bufetech. Napište program, který určí, kolik nejvýše bufetů může lyžař při jedné své cestě navštívit.

Omezení a hodnocení

Plných 10 bodů získáte za správné řešení, které v obou úkolech efektivně vyřeší libovolný vstup s $n \leq 100\,000$, $z \leq 250\,000$.

Za libovolné polynomiální řešení dostanete až 2 body v úkolu A a až 5 bodů v úkolu B. Za libovolné správné řešení bez ohledu na jeho efektivitu můžete získat 1 bod v úkolu A a 3 body v úkolu B.

Formát vstupu a výstupu

Na prvním řádku vstupu jsou zadána čísla n, z, b – počet lokalit, sjezdovek a bufetů. Na druhém řádku vstupu jsou čísla $u_0, \ldots, u_{b-1}$ – čísla lokalit, kde jsou bufety. Všechna tato čísla jsou z rozsahu hodnot od 2 do $n-1$ a jsou navzájem různá. Zbytek vstupu tvoří z řádků, na každém z nich je popis jedné sjezdovky ve tvaru „odkud kam“. Všechny dvojice „odkud kam“ jsou navzájem různé. Sjezdovky zadané na vstupu odpovídají výše uvedenému popisu lyžařského střediska.

Na výstup vypište jedno celé číslo – řešení příslušného úkolu A nebo B. V následujícím příkladu uvádíme na výstupu řešení obou úkolů.

Příklad

Vstup:	Výstup:
11 10 7	4
2 3 4 6 7 8 10	3
1 2	
1 7	
3 5	
3 6	
4 0	
5 4	
6 0	
7 3	
8 0	
9 10	

Vysvětlení: Když se chce lyžař cestou zastavit v jednom bufetu, má na výběr čtyři možnosti: může zvolit bufet v lokalitě 3, 4, 6 nebo 7.

Při jedné cestě může lyžař navštívit nejvýše tři bufety: buď trojici bufetů v lokalitách (7, 3, 4), nebo trojici bufetů (7, 3, 6).

Úkol A

Ze zadání úlohy je zřejmé, že soustava lokalit a sjezdovek v lyžařském středisku představuje orientovaný graf. Lokality odpovídají vrcholům grafu, sjezdovky pak hranám grafu s orientací danou tím, jak se jede po sjezdovce shora dolů (tak jsou sjezdovky také zadány na vstupu). V některých lokalitách jsou umístěny bufety, což znamená, že odpovídající vrcholy grafu jsou nějak speciálně označeny (budeme jim říkat *označené vrcholy*).

Úkol A snadno vyřešíme obyčejným prohledáváním tohoto grafu. Chceme určit počet označených vrcholů (bufetů), které leží na některé orientované cestě vedoucí z vrcholu 1 (horní stanice lanovky) do vrcholu 0 (dolní stanice lanovky). Potřebujeme tedy nalézt a spočítat všechny označené vrcholy, které mají zároveň dvě vlastnosti: jsou dosažitelné z vrcholu 1 a je z nich dosažitelný vrchol 0.

Všechny vrcholy dosažitelné z vrcholu 1 najdeme tak, že z vrcholu 1 spustíme prohledávání grafu – například do hloubky nebo do šířky, to pro výsledek výpočtu není podstatné. Stejným postupem získáme i všechny vrcholy, z nichž je dosažitelný vrchol 0, jenom tentokrát budeme zkoumaný graf prohledávat od vrcholu 0 proti směru orientace hran (jako kdybychom

zjišťovali, kam všude se dostaneme, když půjdeme od dolní stanice lanovky po sjezdovkách směrem nahoru). Potom už stačí spočítat ty označené vrcholy, do nichž jsme se dostali při obou uvedených prohledáváních grafu. Při vhodné reprezentaci grafu v paměti má toto řešení časovou i paměťovou složitost $O(n + z)$.

V následující programové ukázce máme graf reprezentován pomocí seznamů následníků jednotlivých vrcholů. Tyto seznamy následníků implementujeme ve tvaru lineárních spojových seznamů. Seznamy následníků ovšem neumožňují efektivně procházet hrany proti směru jejich orientace, proto graf ukládáme ještě duplicitně jako seznamy předchůdců jednotlivých vrcholů. K procházení grafu zde použijeme prohledávání do hloubky pomocí rekurze. Pro práci s čísly vrcholů v programu používáme standardní datový typ integer. V případě větších vstupních hodnot n by bylo třeba změnit ho na longint.

```
program Lyzar1;    {řešení úkolu A}

const MaxN = 10000;          {maximální počet lokalit}
type PUzel = ^TUzel;
     TUzel = record
               Cislo: integer;
               Dalsi: PUzel
             end;

var N, Z, B: integer;      {počet lokalit, sjezdovek, bufetů}
    Bufet: array[0..MaxN-1] of boolean;        {kde je bufet}
  {Kam a Odkud vede sjezdovka}
    Kam, Odkud: array[0..MaxN-1] of PUzel;
  {dosažitelnost lokality}
    Shora, Zdola: array[0..MaxN-1] of boolean;
    i, j, k: integer;
    P: PUzel;

procedure PruchodShora(V: integer);
{prohledávání od vrcholu V ve směru sjezdovek; podle pole Kam}
var P: PUzel;
begin
P:=Kam[V];
while P <> nil do
  begin
  if not Shora[P^.Cislo] then
    begin Shora[P^.Cislo]:=true; PruchodShora(P^.Cislo) end;
  P:=P^.Dalsi
  end
end; {procedure PruchodShora}
```

```
procedure PruchodZdola(V: integer);
{prohledávání od vrcholu V proti směru sjezdovek,
tzn. podle pole Odkud}
var P: PUzel;
begin
P:=Odkud[V];
while P <> nil do
  begin
  if not Zdola[P^.Cislo] then
    begin Zdola[P^.Cislo]:=true; PruchodZdola(P^.Cislo) end;
  P:=P^.Dalsi
  end
end; {procedure PruchodZdola}

begin
readln(N, Z, B);
for i:=0 to N-1 do                          {inicializace}
  begin
  Bufet[i]:=false;                          {nemáme bufety}
  Kam[i]:=nil; Odkud[i]:=nil;               {nemáme sjezdovky}
  Shora[i]:=false; Zdola[i]:=false;         {nemáme dostupnost}
  end;
for i:=1 to B do     {načtení vstupu, kde jsou bufety}
  begin read(j); Bufet[j]:=true end;
for i:=1 to Z do     {načtení vstupu, jak vedou sjezdovky}
  begin
  read(k, j);        {sjezdovka k->j}
  new(P); P^.Cislo:=j; P^.Dalsi:=Kam[k]; Kam[k]:=P; {následník}
  new(P); P^.Cislo:=k; P^.Dalsi:=Odkud[j]; Odkud[j]:=P;
                                                    {předchůdce}
  end;

PruchodShora(1);
PruchodZdola(0);

j:=0;
for i:=0 to N-1 do
  if Bufet[i] and Shora[i] and Zdola[i] then inc(j);
writeln(j)
end.
```

Úkol B

V úkolu B vede k optimálnímu řešení jednoduché použití dynamického programování. Pracujeme se stejným orientovaným grafem jako v úloze A a chceme v něm určit, jaký maximální počet označených vrcholů leží na

nějaké orientované cestě vedoucí z vrcholu 1 do vrcholu 0. Tento hledaný počet si označíme $T[1]$. Analogicky pro každý vrchol v bude hodnota $T[v]$ znamenat maximální počet označených vrcholů, které leží na nějaké cestě z vrcholu v do vrcholu 0. Jinými slovy: kolik nejvýše bufetů můžeme navštívit při jízdě po sjezdovkách z lokality v k dolní stanici lanovky.

Jak už naznačuje zavedené označení, údaje $T[v]$ si budeme ukládat jako hodnoty prvků jednorozměrného pole T. Prvky pole T budeme počítat ve vhodném pořadí „zdola nahoru“ (od dolní stanice lanovky nahoru do kopce). Víme, že $T[0] = 0$, protože podle zadání úlohy v lokalitě 0 není bufet. Jak jsme již uvedli, řešením úlohy bude hodnota $T[1]$. Zbývá ukázat, jak spočítáme hodnotu $T[v]$ pro obecný vrchol v, $v > 0$. Z vrcholu v vede několik hran do nějakých vrcholů u_1, u_2, …, u_k. Z hodnot $T[u_1]$, $T[u_2], \ldots, T[u_k]$ tedy vezmeme maximum (to odpovídá nejvýhodnější cestě z vrcholu v do cíle) a pokud je označen vrchol v, ještě toto maximum zvýšíme o 1.

Uvedený postup výpočtu můžeme implementovat dvěma základními způsoby, jako je to ostatně u dynamického programování obvyklé: buď rekurzivně shora (s ukládáním již spočítaných hodnot pro zvýšení efektivity), nebo iteračně zdola. Ukážeme si obě tyto možnosti.

První varianta znamená zapsat výše uvedený vztah pro výpočet hodnoty $T[v]$ jako rekurzivní funkci s parametrem v. Samotné použití rekurze by ale mohlo vést k velmi neefektivnímu algoritmu s exponenciální časovou složitostí, některé hodnoty $T[v]$ bychom mohli počítat opakované mnohokrát (když do téže lokality můžeme přijet po více různých sjezdovkách). Abychom dostali efektivní algoritmus, přidáme do programu výše zmíněné pole T, v němž si budeme ukládat všechny již spočítané hodnoty. Když pak budeme v dalším výpočtu potřebovat opakovaně hodnotu $T[v]$, nebudeme ji podruhé počítat znovu, ale použijeme uloženou hodnotu. Pro každý vrchol v budeme díky tomu počítat hodnotu $T[v]$ jenom jednou.

Uvedenou úpravou dostaneme program, který nejvýše jednou zpracuje každý vrchol grafu a rovněž nejvýše jednou zpracuje každou hranu – vždy při zpracování lokality postupně projdeme všechny sjezdovky, které z ní vedou. Při vhodné reprezentaci grafu bude časová i paměťová složitost $O(n + z)$. Toto řešení je zjevně optimální, když už jenom na přečtení vstupu potřebujeme řádově stejný počet kroků výpočtu.

V následujícím programu co nejvíce dodržujeme stejné značení, jako v řešení úkolu A. Použijeme stejnou reprezentaci grafu pomocí seznamů následníků implementovaných lineárními spojovými seznamy. Tentokrát

ale nebudeme potřebovat duplicitní uložení grafu pomocí seznamů předchůdců. Pro práci s čísly lokalit v programu opět používáme datový typ integer, který by bylo třeba změnit na longint v případě větších vstupních hodnot n. Rekurzivní funkce TT zajišťuje výpočet hodnot T popsaných výše v textu.

```
program Lyzar2;
{řešení úkolu B rekurzivně}

const MaxN = 10000;      {maximální počet lokalit}
type PUzel = ^TUzel;
     TUzel = record
               Cislo: integer;
               Dalsi: PUzel
             end;

var N, Z, B: integer;       {počet lokalit, sjezdovek, bufetů}
    Bufet: array[0..MaxN-1] of boolean;  {kde je bufet}
    Kam: array[0..MaxN-1] of PUzel;      {kam vede sjezdovka}
    T: array[0..MaxN-1] of integer;
                          {max. počet bufetů na cestě do cíle}
    i, j, k: integer;
    P: PUzel;

function TT(V: integer): integer;
{rekurzivní funkce na výpočet hodnot T}
var X: integer;
    P: PUzel;
begin
X:=-maxint;
P:=Kam[V];
while P <> nil do
  begin
  if T[P^.Cislo] = -1 then T[P^.Cislo]:=TT(P^.Cislo);
  if T[P^.Cislo] > X then X:=T[P^.Cislo];
  P:=P^.Dalsi
  end;
if (X <> -maxint) and (Bufet[V]) then inc(X);
T[v]:=X;
TT:=X
end; {function TT}

begin
readln(N, Z, B);
for i:=0 to N-1 do                {inicializace}
  begin
  Bufet[i]:=false;                {nemáme bufety}
  Kam[i]:=nil;                    {nemáme sjezdovky}
```

```
  T[i]:=-1;                    {údaj zatím neznáme}
  end;
T[0]:=0;
for i:=1 to B do               {načtení vstupu, kde jsou bufety}
  begin read(j); Bufet[j]:=true end;
for i:=1 to Z do               {načtení vstupu, jak vedou sjezdovky}
  begin
  read(k, j);                  {sjezdovka k->j}
  new(P); P^.Cislo:=j; P^.Dalsi:=Kam[k]; Kam[k]:=P; {následník}
  end;
writeln(TT(1))
end.
```

Stejný postup řešení můžeme implementovat také bez použití rekurze. Začneme tím, že si všechny vrcholy grafu vhodně uspořádáme. Najdeme tzv. *topologické uspořádání* vrcholů našeho grafu, což je jedno z přípustných uspořádání lokalit podle jejich nadmořské výšky. V tomto pořadí (počínaje od nejníže položených lokalit) potom postupně každý vrchol jednou zpracujeme pomocí výše uvedeného vzorce pro výpočet hodnoty $T[v]$ a odpověď si uložíme v poli T pro další použití. Díky počátečnímu uspořádání vrcholů máme zajištěno, že při každém výpočtu $T[v]$ budeme již znát všechny hodnoty, které vzorec pro výpočet $T[v]$ používá – jsou to totiž hodnoty odpovídající níže položeným lokalitám.

Tento iterační postup zdola provádí ve skutečnosti naprosto stejný výpočet, jako předchozí rekurzivní řešení s ukládáním spočítaných hodnot do pole T. Má proto i stejnou časovou složitost $O(n + z)$. Výslednou časovou složitost nám nepokazí ani potřebné topologické uspořádání grafu, neboť i to dokážeme spočítat v čase $O(n + z)$.

Zbývá uvést, jak najdeme topologické uspořádání zadaného orientovaného grafu. Existuje více možných postupů, nejjednodušší pro nás bude použít prohledávání grafu do hloubky, s nímž jsme se již setkali v obou předchozích programech. Topologické uspořádání grafu (v opačném pořadí) je pak dáno pořadím, v němž během prohledávání grafu uzavíráme (tzn. opouštíme) jednotlivé vrcholy grafu.

Poznamenejme ještě, že orientovaný graf může mít více různých topologických uspořádání. Náš algoritmus najde nějaké jedno z nich, což pro řešení úlohy naprosto vyhovuje. Různá topologická uspořádání grafu se mohou lišit vzájemným pořadím těch vrcholů, mezi nimiž nevede žádná orientovaná cesta, a ze zadání grafu tudíž neplyne, který z vrcholů má být zpracován dříve (tzn. která z uvažovaných lokalit má nižší nadmořskou výšku).

```
program Lyzar3;
{řešení úkolu B iteračně zdola}

const MaxN = 10000;                    {maximální počet lokalit}
type PUzel = ^TUzel;
     TUzel = record
               Cislo: integer;
               Dalsi: PUzel
             end;

var N, Z, B: integer;      {počet lokalit, sjezdovek, bufetů}
    Bufet: array[0..MaxN-1] of boolean;        {kde je bufet}
    Kam: array[0..MaxN-1] of PUzel;      {kam vede sjezdovka}
    T: array[0..MaxN-1] of integer;
                          {max. počet bufetů na cestě do cíle}
    Shora: array[0..MaxN-1] of boolean; {dosažitelnost lokality}
    Poradi: array[1..MaxN] of integer;
                              {topologické uspořádání lokalit}
    vPoradi: integer;
    i, j, k: integer;
    X, V: integer;
    P: PUzel;

procedure PruchodShora(V: integer);
{prohledávání od vrcholu V ve směru sjezdovek,
 tzn. podle pole Kam}
var P: PUzel;
begin
P:=Kam[V];
while P <> nil do
  begin
  if not Shora[P^.Cislo] then
    begin Shora[P^.Cislo]:=true; PruchodShora(P^.Cislo) end;
  P:=P^.Dalsi
  end;
inc(vPoradi);
Poradi[vPoradi]:=V    {uzavíraný vrchol uložíme do pole Poradi}
end; {procedure PruchodShora}

begin
readln(N, Z, B);
for i:=0 to N-1 do                          {inicializace}
  begin
  Bufet[i]:=false;                          {nemáme bufety}
  Kam[i]:=nil;                              {nemáme sjezdovky}
  T[i]:=-1;                                 {údaj zatím neznáme}
  end;
T[0]:=0;
```

```
for i:=1 to B do                {načtení vstupu, kde jsou bufety}
  begin read(j); Bufet[j]:=true end;
for i:=1 to Z do             {načtení vstupu, jak vedou sjezdovky}
  begin
  read(k, j);                                        {sjezdovka k->j}
  new(P); P^.Cislo:=j; P^.Dalsi:=Kam[k]; Kam[k]:=P; {následník}
  end;

vPoradi:=0;
PruchodShora(1);     {topologické uspořádání lokalit}
for i:=1 to vPoradi do
  begin
  V:=Poradi[i];      {počítáme hodnotu T[V]}
  if V<>0 then continue;
  X:=-maxint;
  P:=Kam[V];
  while P <> nil do
    begin
    if T[P^.Cislo] > X then X:=T[P^.Cislo];
    P:=P^.Dalsi
    end;
  if (X <> -maxint) and (Bufet[V]) then inc(X);
  T[v]:=X;
  end;
writeln(T[1])
end.
```

Ústřední kolo 66. ročníku Matematické olympiády kategorie P

Ve dnech 29.–31. 3. 2017 se v Liberci konalo ústřední kolo 66. ročníku Matematické olympiády kategorie P (programování). Letošní ústřední kolo MO výborně připravili a organizačně zajistili pracovníci Krajské komise MO v Liberci a Technické univerzity Liberec.

K účasti v ústředním kole MO-P bylo vybráno třicet nejlepších řešitelů krajských kol, z nichž se jeden omluvil. Soutěžilo tedy 29 nejlepších mladých programátorů z celé republiky.

Vítězi se stali:

1. *Filip Bialas*, 8/8, Gymnázium Opatov, Praha 4, 56 bodů
2. *Václav Volhejn*, 8/8, Gymnázium Jana Keplera, Praha 6, 55 bodů
3. *Jan Priessnitz*, 8/8, Gymnázium tř. Kpt. Jaroše, Brno, 41 bodů
4. *Martin Kurečka*, 7/8, Gymnázium tř. Kpt. Jaroše, Brno, 37 bodů
5. *Pavel Turek*, 8/8, Gymnázium Olomouc-Hejčín, 37 bodů
6. *Richard Hladík*, 8/8, Gymnázium a OA Mariánské Lázně, 36 bodů

Úplnou výsledkovou listinu, texty soutěžních úloh i jejich vzorová řešení a další informace o 66. ročníků MO-P najdete na adrese: http://mo.mff.cuni.cz/

Podrobná zpráva o Ústředním kole 66. ročníku MO-P bude ve 4. čísle MFI.