INFORMATIKA

Vesmírné cestování
(Úlohy z MO – kategorie P, 29. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

Seznámíme vás s další zajímavou úlohou z Matematické olympiády – kategorie P, tentokrát z domácího kola 55. ročníku soutěže (školní rok 2005/2006). Všechny soutěžní úlohy tohoto ročníku připravili pracovníci Fakulty matematiky, fyzika a informatiky Univerzity Komenského v Bratislavě, kteří se starají o programátorskou olympiádu středoškoláků na Slovensku. Budeme se věnovat jedné zdánlivě běžné úloze z teorie grafů – hledání nejkratší cesty v ohodnoceném orientovaném grafu. Podstatnou komplikací oproti běžným „školním“ úlohám však bude skutečnost, že hrany našeho grafu mohou být ohodnoceny i zápornými čísly. Nejprve se jako obvykle podíváme na úplné zadání úlohy, které si pro potřeby našeho článku trochu upravíme a zkrátíme, aniž bychom tím ovšem změnili smysl původní úlohy.

⋆ ⋆ ⋆

Vědcům se konečně podařilo vymyslet efektivní způsob cestování v časoprostoru. Jejich testovací středisko se skládá z několika lokalit, v každé lokalitě je umístěno několik teleportů. Když vstoupíme do takového teleportu, přemístí nás na stanovenou lokalitu a zároveň nás přemístí také v čase o stanovený počet minut (buď dopředu nebo dozadu). Cílová lokalita i časový posun jsou pro každý teleport pevně nastaveny. Vědci by chtěli zjistit, jak je cestování pomocí teleportů výhodné. Právě se nacházejí u centrálního počítače a chtěli by se jít nasvačit do bufetu. A protože čas jsou peníze, chtěli by být v bufetu co nejdříve. Pohybovat se v čase a prostoru samozřejmě chtějí jen pomocí již postavených teleportů.

Soutěžní úloha:

Program dostane na vstupu počet lokalit N, které budeme označovat čísly od 1 do N. Centrální počítač je umístěn v lokalitě číslo 1, bufet má číslo N. Následuje celkový počet postavených teleportů M a seznam těchto teleportů. Pro každý teleport je určena počáteční lokalita, koncová lokalita a změna času v minutách, jež nastane při průchodu tímto teleportem (kladné číslo znamená posun do budoucnosti, záporné do minulosti a 0 znamená, že se v koncové lokalitě ocitneme ve stejném čase, v jakém jsme nastoupili do teleportu). Každý teleport lze použít jen tím směrem, který je uveden na vstupu. Mezi dvěma lokalitami může být vybudováno více teleportů. Dokonce může existovat i teleport, který nás přesune pouze v čase (tedy počáteční a koncová lokalita jsou u něj totožné). Program má určit čas, kdy nejdříve se můžeme dostat do lokality N, jestliže se v lokalitě 1 nacházíme v čase 0. Pokud tam dokážeme být libovolně brzo (tzn. můžeme pomocí teleportů cestovat neomezeně do minulosti), nebo pokud se tam vůbec nemůžeme dostat, program o tom vydá příslušnou zprávu.

Formát vstupu:

První řádek vstupního souboru `teleport.in` obsahuje dvě čísla N a M ($2 \leq N \leq 1.000$, $0 \leq M \leq 50.000$) oddělená mezerou. Následuje M řádků, na každém z nich jsou tři čísla A_i, B_i, T_i ($1 \leq A_i$, $B_i \leq N$, $|Ti| \leq 10.000$) popisující teleport z lokality A_i do lokality B_i se změnou času T_i minut.

Formát výstupu:

Jediný řádek výstupního souboru `teleport.out` bude obsahovat zprávu „Vedci umrou hlady“, jestliže se od centrálního počítače pomocí teleportů vůbec nedá dostat do bufetu, resp. zprávu „Vedci poznaji vznik vesmiru“, jestliže můžeme cestovat do nekonečna do minulosti. Jinak bude výstup obsahovat jedno celé číslo představující čas v minutách, kdy nejdříve se vědci dokážou dostat do bufetu.

Příklady:

`teleport.in`

```
3 4
1 2 5
2 3 -7
1 3 -1
```

```
1 3 16

teleport.out
-2
```

Prvním teleportem se vědci dostanou do lokality 2 *v čase* 5*, odtud druhým do lokality* 3 *v čase* $5 + (-7) = -2$*. Ostatní možnosti jsou horší.*

```
teleport.in
2 2
1 1 -1
1 2 0

teleport.out
Vedci poznaji vznik vesmiru
```

Dříve než se vědci druhým teleportem přesunou do bufetu, mohou prvním odcestovat libovolně daleko do minulosti.

```
teleport.in
4 3
1 2 -1
2 3 0
4 3 10

teleport.out
Vedci umrou hlady
```

Poslední teleport nemohou vědci použít na přesun z lokality 3 *do lokality* 4*, jedině naopak.*

⋆ ⋆ ⋆

Jistě vás hned napadne, že situaci ze zadání úlohy si můžeme představit jako ohodnocený orientovaný graf. Vrcholy grafu budou představovat lokality, hrany reprezentují teleporty mezi lokalitami. Každá hrana je ohodnocena číslem, které určuje posun v čase při průchodu danou hranou. Toto ohodnocená budeme nazývat „délka hrany“, jak je v grafové terminologii obvyklé, i když v našem případě nepůjde o délku v obvyklém slova smyslu. Úkolem je najít sled vedoucí z vrcholu 1 do vrcholu N s nejmenším součtem ohodnocení hran (nazveme ho nejkratší), případně

vypsat zprávu, že takový sled neexistuje. Pro úplnost dodejme, že v teorii grafů sledem rozumíme posloupnost vrcholů $v_1, \ldots, v_k$ takovou, že mezi vrcholy v_i a v_{i+1} vede hrana.

V úvodu si můžeme úlohu trochu zjednodušit. Uvědomte si, že pokud mezi dvěma vrcholy vede více hran stejným směrem, stačí uvažovat jenom tu z nich, která má nejmenší délku. Jinak bychom totiž dokázali nalezený sled zkrátit výměnou delší hrany za kratší. Případných násobných hran se zbavíme nejraději hned při načítání vstupních dat, abychom nemuseli ani řešit jejich uložení v datové struktuře.

Kdyby byly všechny hrany grafu ohodnoceny nezáporně, byla by naše úloha velmi snadná. V takovém případě se při hledání nejkratšího sledu jistě nevyplatí vracet se do vrcholu, který jsme už jednou navštívili. Sled, ve kterém se žádné vrcholy neopakují, nazýváme v grafové terminologii cesta, a na nalezení nejkratší cesty v ohodnocením grafu máme k dispozici standardní algoritmy, které najdete v každé základní učebnici teorie grafů. Nejznámější z nich je Dijkstrův algoritmus. Jeho použitelnost je však omezena právě jen na grafy s nezáporným ohodnocením hran.

Nadále tedy budeme počítat s tím, že náš graf může obsahovat i hrany se zápornou délkou. Mohou nastat dvě situace, kdy nejkratší sled z vrcholu 1 do vrcholu N neexistuje. Buď se z vrcholu 1 do vrcholu N po hranách grafu nemůžeme vůbec dostat, nebo existuje sled z vrcholu 1 do vrcholu N takový, že obsahuje cyklus se záporným součtem délek hran. Po takovém cyklu pak můžeme chodit pořád dokola a stále snižovat celkovou délku sledu. Proto ani neexistuje nejkratší sled – ke každému sledu totiž dokážeme nalézt sled kratší.

Ukážeme si dva různé algoritmy, které efektivně řeší naši úlohu. V obou případech se jedná o standardní algoritmy, i když trochu méně známé, než již zmíněný algoritmus Dijkstrův. První řešení využívá **Floydův-Warshallův algoritmus**. Tento algoritmus je založen na myšlence dynamického programování a slouží k výpočtu nejkratších vzdáleností mezi všemi dvojicemi vrcholů v grafu. Výhodou pro nás je, že mu nevadí ani záporná ohodnocení hran. Graf si uložíme do dvojrozměrného pole G, přičemž výchozí hodnota $G[i, j]$ bude rovna délce hrany z vrcholu i do vrcholu j (nebo nekonečno, jestliže taková hrana neexistuje). Algoritmus vypadá následovně:

```
for k:=1 to N do
   for i:=1 to N do
      for j:=1 to N do
```

```
if G[i, j] > G[i, k] + G[k, j] then
   G[i, j] := G[i, k] + G[k, j];
```

Po skončení výpočtu je hodnotou $G[i, j]$ délka nejkratšího sledu z i do j (případně nekonečno, pokud žádný takový sled neexistuje). Navíc platí, že je-li $G[i, i]$ záporné pro určité i, pak vrchol i leží na nějakém záporném cyklu. Pokud tento vrchol leží na nějakém sledu z 1 do N (tedy $G[1, i]$ ani $G[i, N]$ nejsou rovny nekonečnu), potom existuje sled z 1 do N s libovolně nízkým ohodnocením.

Jak jsme již uvedli, popsaný algoritmus je založen na principu dynamického programování. Dynamika zde probíhá přes postupně rostoucí množinu vrcholů, které smíme použít v hledaném sledu minimální délky. Po k-tém průchodu vnějšího cyklu udávají hodnoty $G[i, j]$ délku nejkratšího sledu z i do j takového, že tento sled vede pouze přes (ne nutně všechny) vrcholy z množiny $\{1, \ldots, k\}$. V každém kroku výpočtu musíme přepočítat všech N^2 hodnot $G[i, j]$, i když nakonec po posledním N-tém průchodu nás bude v této úloze zajímat pouze jediná z nich, a to $G[1, N]$.

Ze zápisu algoritmu je zjevné, že časová složitost tohoto algoritmu je $O(N^3)$. Výpočet je totiž tvořen třemi do sebe vnořenými cykly, z nichž každý se vykoná N-krát. Přitom tělo vnitřního cyklu má již konstantní časové nároky. Paměťová složitost je $O(N^2)$, neboť vystačíme s jedním polem G velikosti $N \times N$ na uložení výchozího grafu, ukládání všech mezivýsledků i na uložení výsledných nejkratších délek.

Ukážeme si ještě druhé řešení úlohy, které využívá **Bellmanův-Fordův algoritmus**. Jedná se o standardní grafový algoritmus, který podobně jako Dijkstrův algoritmus počítá nejkratší vzdálenosti ze zvoleného výchozího vrcholu do cílového vrcholu grafu (nebo do všech ostatních vrcholů grafu). Bellmanův-Fordův algoritmus je o něco pomalejší než algoritmus Dijkstrův, ale zato ho lze použít i na grafy se záporným ohodnocením hran. Od Dijkstrova algoritmu se liší způsobem procházení grafu.

Pro potřeby tohoto algoritmu si budeme orientované hrany zkoumaného grafu uchovávat ve třech polích, kde $a[i]$ je začátek, $b[i]$ je konec a $t[i]$ je délka i-té hrany. Na pořadí hran nezáleží, budeme tedy pracovat s takovým pořadím, v jakém jsou hrany zadány na vstupu.

Rovněž tento algoritmus je založen na principu dynamického programování. Dynamika je v této úloze vedena přes počet hran, jimiž je hledaný sled minimální délky tvořen. Nechť $D[l, i]$ je délka takového nejkratšího sledu z vrcholu 1 do vrcholu i, který používá právě l hran. Je zřejmé, že $D[0, i]$ je rovno nekonečnu pro všechna i kromě 1, zatímco $D[0, 1] = 0$.

Předpokládejme, že známe D$[l-1,i]$ pro všechna i a pro nějaké $l>0$. Snadno potom spočítáme D$[l,i]$ pro libovolné i. V nejkratším sledu tvořeném l hranami je nějaká hrana poslední a zbytek je nejkratší sled používající $l-1$ hran, který končí v počátečním vrcholu l-té hrany. Stačí tedy vyzkoušet všechny možnosti pro tuto poslední hranu. Tím dostáváme vztah pro výpočet hodnot D$[l,i]$:

$$\mathrm{D}[l,i] = \min(1 \le k \le M)\{\mathrm{D}[l-1, \mathrm{a}[k]] + \mathrm{t}[k], \quad \text{kde } \mathrm{b}[k] = i\}$$

Při zvolené reprezentaci grafu se výpočet nejsnáze provádí tak, že projdeme postupně všechny hrany a počítáme jednotlivá minima pro všechny vrcholy současně. Víme, že pokud existuje nejkratší sled z 1 do N, bude mít nejvýše $N-1$ hran, neboť neobsahuje cyklus. Výsledkem je tedy minimum z hodnot D$[l,N]$ pro $l = 1,\ldots,N$–1. Je-li tato hodnota rovna nekonečnu, potom žádný sled neexistuje.

Ještě potřebujeme ověřit, zda se nemůžeme dostat do záporného cyklu. Takový cyklus může obsahovat nejvýše N hran, jak si můžeme ukázat třeba na příkladu grafu s orientovanými ohodnocenými hranami $(1,2,1)$, $(2,3,1),\ldots,(N$–$1,N,1),(N,1,$–$N)$. Jestliže tedy existuje sled obsahující záporný cyklus, pak existuje i sled délky nejvýše $2N$–1, který tento cyklus obsahuje. Popsanou úpravu vzdáleností D[l, i] proto spustíme ještě N krát. Když minimum z D$[l,N]$ pro $N \le l \le 2N$–1 je menší než výsledek, který jsme našli předtím, potom skutečně existuje sled z 1 do N, který obsahuje záporný cyklus.

Na závěr můžeme provést ještě jedno implementační zjednodušení. Uvědomte si, že nás nezajímají přesné délky sledů s právě l hranami. Proto nám stačí použít v programu pouze jednorozměrné pole D$[i]$ a výše uvedené úpravy provádět jen na něm. Tímto zjednodušením výsledek výpočtu nezměníme, i když hodnoty získané po jednotlivých iteracích mohou být odlišné od původního řešení.

Časová složitost tohoto algoritmu je O$(N.M)$, paměťová O$(N+M)$.

```
program Vesmirne_cestovani;
const MaxN = 1000;   {maximální počet vrcholů}
      MaxM = 50000;  {maximální počet hran}
      Nekonecno = 2000000000;
var N, M: integer;   {skutečný počet vrcholů a hran}
    f: text;
    a: array[1..MaxM] of integer;
```

```
    b: array[1..MaxM] of integer;
    t: array[1..MaxM] of integer;
    D: array[1..MaxN] of longint;
    i, l, k: integer;
    vysledek: longint;

begin
  {Načtení vstupních dat:}
  assign(f, 'teleport.in');
  reset(f);
  readln(f, N, M);
  for i:=1 to M do
    readln(f, a[i], b[i], t[i]);
  close(f);

  {Inicialiazce:}
  assign(f, 'teleport.out');
  rewrite(f);
  D[1]:=0;
  for i:=2 to N do
     D[i]:=Nekonecno;

  {Vlastní výpočet cest:}
  for l:=1 to N-1 do
     for k:=1 to M do
        if D[b[k]] > D[a[k]] + t[k] then
           D[b[k]]:=D[a[k]] + t[k];
  vysledek:=D[N];

  if vysledek = Nekonecno then
     writeln(f, 'Vedci umrou hlady')

  else
     begin  {ověření existence záporných cyklů:}
     for l:=1 to N do
        for k:=1 to M do
           if D[b[k]] > D[a[k]] + t[k] then
              D[b[k]]:=D[a[k]] + t[k];
```

```
      if D[N] < vysledek then
         writeln(f, 'Vedci poznaji vznik vesmiru')
      else
         writeln(f, vysledek);
      end;

  close(f);
end.
```

Wolfram|Alpha

LUKÁŠ HONZÍK

Fakulta pedagogická ZČU v Plzni

Nejen vyučující matematiky na středních a vysokých školách mají s názvem společnosti Wolfram Research, Inc. spojený především její nejznámější produkt, kterým je výpočetní program Wolfram Mathematica využitelný v mnoha technických oblastech. Mathematica se v loňském roce dočkala své již osmé verze a po dlouhou dobu zůstává „vlajkovou lodí" společnosti.

Méně známou skutečností je pak fakt, že společnost již nějaký čas provozuje a dále vyvíjí online webovou službu Wolfram|Alpha (někdy též psáno jako Wolfram Alpha nebo WolframAlpha), kterou zřejmě nejlépe definuje její anglický podtitul „*Computational Knowledge Engine*". Toto slovní spojení lze do češtiny přeložit přibližně jako výpočetní vědomostní prostředek, či nástroj. Uživatelé jej najdou prostřednictvím webového prohlížeče na adrese www.wolframalpha.com a mohou pomocí něj provádět matematické výpočty, vyhledávat a zjišťovat informace.

Vývoj a prostředí

Duchovním otcem systému je britský vědec Stephen Wolfram, hlavní vývojář již zmíněného softwaru Mathematica. První oznámení o rozběhnutí celého projektu učinil Wolfram v březnu 2009 (v návaznosti na svou