[5] *Spíchal, L.*: Modeling the cross-sectional shape of the long bones using the superellipse. Matematika, informační technologie a aplikované vědy (MITAV 2019). Brno, Univerzita obrany, 2019.

[6] *Pickover, C. A.*: The Math Book: From Pythagoras to the 57th Dimension, 250 Milestones in the History of Mathematics. Sterling Publishing Company, Inc., 2009.

[7] *Gridgeman, N. T.*: Lamé ovals. The Mathematical Gazette, roč. 54 (1970), č. 387, s. 31–l37.

[8] *Gardner, M.*: The "superellipse": a curve that lies between the ellipse and the rectangle. Scientific American, roč. 213 (1965), č. 3, s. 222–238.

[9] *Wicklin, R.*: The spiral of splatter. Dostupné z: https://blogs.sas.com/content/iml/2015/06/11/spiral-of-splatter.html

[10] *Matsuura, M*: Gielis' superformula and regular polygons. Journal of Geometry, roč. 106 (2015), s. 383–403.

[11] *Devlin, K*: Jazyk matematiky. Jak zviditelnit neviditelné. Dokořán, Praha, 2011.

Nejlevnější vaření
(Úlohy z MO kategorie P, 39. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

V dalším pokračování dlouhodobého seriálu článků o úlohách z Matematické olympiády kategorie P (programování) vám představíme úlohu z krajského kola předloňského 67. ročníku MO (školní rok 2017/18). Úloha na první pohled pojednává o tom, jak nejlépe zkombinovat kuchařské recepty, abychom získali co nejlevněji požadované jídlo. Při její hlubší analýze ale zjistíme, že se vlastně jedná o grafový problém a při jeho řešení skutečně použijeme modifikace obecně známých algoritmů na hledání nejkratší cesty v grafu. Jako obvykle se nejprve seznámíme s přesným zadáním.

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

Honza se rozhodl, že si uvaří večeři. Jeho kuchařské schopnosti jsou ale značně omezené. Zvládá používat jen velmi jednoduché recepty, podle nichž ze dvou jídel vznikne nějaké třetí. Navíc nemá moc peněz, a tak by byl rád, kdyby ho celé vaření stálo co nejméně.

Soutěžní úloha

Existuje n druhů jídla. Pro potřeby této úlohy si je očíslujeme od 1 do n. Honza chce připravit jednu porci jídla číslo 1. Každé jídlo lze přímo koupit v obchodě. Jedna porce jídla číslo i má v obchodě cenu c_i. Od každého druhu jídla je možné koupit libovolný počet porcí.

Honza má kuchařskou knihu, která obsahuje r jednoduchých receptů. Každý z receptů má následující tvar: „Z jedné porce jídla s_i a jedné porce jídla t_i připravíš jednu porci jídla u_i.“

Napište program, který spočítá, za jakou nejnižší cenu lze získat jednu porci jídla číslo 1.

Formát vstupu a výstupu

Na prvním řádku vstupu jsou čísla n a r: počet druhů jídla a počet receptů. Na druhém řádku je n kladných celých čísel $c_1, \ldots, c_n$: obchodní cena jedné porce pro každé jídlo. Zbytek vstupu tvoří r řádků, z nichž každý má tvar s_i, t_i, u_i a popisuje jeden recept.

Program vypíše jedno číslo: nejmenší dosažitelnou cenu jedné porce jídla číslo 1.

Omezení a hodnocení

Za libovolné správné řešení získáte alespoň 4 body. Za libovolné správné řešení, jehož časová složitost je polynomiální vzhledem k n a r, které obsahuje také korektní důkaz správnosti, získáte alespoň 7 bodů.

Vzorové řešení vyřeší efektivně libovolný vstup, pro který platí $1 \leq n \leq \leq 100\,000$, $0 \leq r \leq 100\,000$.

Příklady

Vstup:	Výstup:
5 3 47 1 10 20 4700 4 5 1 2 3 4 3 2 5	22

Kdyby Honza přímo koupil jednu porci jídla číslo 1, stálo by ho to 47. Existují ale levnější postupy. Nejlepší z nich vypadá takto: Za 1+1+10+10 koupí Honza dvě porce jídla 2 a dvě porce jídla 3. Potom z jedné dvojice jídel 2 a 3 uvaří jídlo 4 a z druhé dvojice uvaří jídlo 5. Na závěr z jedné porce jídla 4 a jedné porce jídla 5 uvaří požadované jídlo 1.

Vstup:	Výstup:
4 3 1000 1000 1000 10 2 2 1 3 3 2 4 4 3	80

Tentokrát je nejlepší jednu porci jídla číslo 1 postupně uvařit z osmi porcí jídla číslo 4.

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

Ačkoliv to není na první pohled úplně zřejmé, k řešení úlohy lze využít postupy přímo inspirované standardními grafovými algoritmy na hledání nejkratší cesty v ohodnoceném grafu. Postupně si ukážeme tři různé postupy, které se liší svou časovou efektivitou.

Začneme úplně primitivním řešením, kterým je zkoušení všech možností. Pokud bychom už znali optimální cenu jedné porce každého jídla a všechna jídla bychom si podle této ceny uspořádali vzestupně, pak k přípravě každého jídla budeme jistě používat pouze jídla umístěná před ním, tzn. jídla s nižší cenou. V takovém případě by byl výpočet jednoduchý – procházeli bychom jídla v pořadí zleva doprava a rovnou bychom počítali jejich výsledné ceny. Optimální cenou každého jídla je buď jeho kupní cena, nebo je to cena získaná podle některého receptu, v němž již známe optimální ceny obou vstupů.

Jediný problém tohoto řešení spočívá v tom, že správné pořadí jídel neznáme. Můžeme ale vyzkoušet všech $n!$ různých pořadí a pro každé z nich použít výše uvedený postup. Mezi všemi způsoby uspořádání jídel je jistě i ten správný (nemusí být jediný), pro který dostaneme optimální cenu jídla číslo 1. Výsledkem úlohy je tedy nejnižší cena jídla číslo 1 ze všech možností, které získáme pro jednotlivá pořadí jídel.

Popsané řešení je sice zaručeně správné, ale velmi neefektivní. Zkusíme proto nalézt jiný postup s polynomiální časovou složitostí. Inspirací pro nás bude Bellmanův–Fordův algoritmus na určení délky nejkratší cesty v hranově ohodnoceném grafu ze zadaného startovního vrcholu do všech

ostatních vrcholů grafu. Připomeneme si nejprve ve stručnosti a bez důkazu správnosti, jak tento algoritmus funguje. Pro každý vrchol v počítáme hodnotu h_v představující délku dosud nejkratší nalezené cesty ze startovního vrcholu do vrcholu v. Na začátku výpočtu má startovní vrchol hodnotu 0 a všechny ostatní vrcholy hodnotu ∞ (nekonečno). Hodnoty vrcholů budeme opakovaně přepočítávat a zlepšovat, tzn. snižovat, dokud to půjde. V každém kroku výpočtu projdeme všechny hrany grafu (v libovolném pořadí) a pro každou hranu (u, v) se pokusíme zlepšit hodnotu h_v tím, že bychom šli do vrcholu v z vrcholu u touto hranou. Pokud bude platit $h_u + |(u, v)| < h_v$, hodnotu h_v patřičně snížíme. Výpočet ukončíme ve chvíli, když už se v některém kroku výpočtu hodnota žádného vrcholu nezmění. V grafu bez záporných cyklů k tomu dojde nejpozději po $n - 1$ krocích, kde n je počet vrcholů grafu. Pokud se i v n-tém kroku výpočtu některá hodnota vrcholu sníží, v grafu jistě existuje záporný cyklus a úloha hledání nejkratší cesty v takovém případě nemá smysl. Časová složitost popsaného algoritmu je $\mathrm{O}(n \cdot m)$, kde n je počet vrcholů a m je počet hran grafu.

Při řešení naší úlohy o vaření budeme postupovat obdobně. Jídla zde odpovídají vrcholům grafu a jejich hodnotou bude nejlepší cena, za jakou jídlo zatím dokážeme získat. Počáteční hodnotu každého jídla nastavíme na jeho kupní cenu. V následujících krocích výpočtu budeme opakovaně procházet seznam všech receptů a pro každý z nich vždy zkontrolujeme, zda pomocí něho můžeme zlepšit (tzn. snížit) hodnotu toho jídla, které se receptem vytváří. Pro recept $(s\ t\ u)$ se tedy ptáme, zda $h_s + h_t < h_u$. Pokud ano, hodnotu h_u snížíme na $h_s + h_t$. Výpočet ukončíme, když se v některém kroku výpočtu hodnota žádného jídla nezmění. K tomu dojde nejpozději po $n - 1$ krocích.

Ukážeme, proč popsaný algoritmus skutečně funguje, a že po $n - 1$ krocích udávají hodnoty všech jídel optimální cenu, za jakou lze tato jídla získat. Představme si opět, že máme všechna jídla seřazena podle jejich optimální ceny. Hodnoty jídel jsou na počátku rovny jejich kupním cenám. Hodnota prvního, tzn. nejlevnějšího jídla se hned na začátku výpočtu rovná jeho optimální ceně, neboť nejlevnější jídlo se nikdy nevyplatí vařit podle receptu z dražších surovin. Při každém průchodu všemi recepty získáme optimální cenu alespoň jednoho dalšího jídla v našem pořadí. Zdůvodnění je podobné, jako v prvním uvedeném řešení úlohy. Při průchodu všemi recepty vyzkoušíme mimo jiné také všechny způsoby, jak lze toto jídlo uvařit z jídel, která jsou levnější a pro která už známe optimální

cenu. Po $n-1$ průchodech všemi recepty proto budeme znát optimální ceny všech jídel.

Pro některá vstupní data můžeme získat optimální cenu více jídel při jednom průchodu seznamem receptů. Výsledné ceny všech jídel v takovém případě dostaneme dokonce po méně než $n-1$ krocích výpočtu. Výpočet lze tedy v některých případech urychlit tím, že nebudeme vždy provádět plných $n-1$ průchodů seznamem receptů, ale skončíme ve chvíli, když se při některém průchodu hodnota žádného jídla nezmění. Zjevně by se totiž žádná hodnota nezměnila ani v případě, že bychom další průchody prováděli.

Popsaný algoritmus má časovou složitost v nejhorším případě $\mathrm{O}(n \cdot r)$, kde n je počet jídel a r je počet receptů. Provádí totiž nejvýše $n-1$ kroků výpočtu a každý z nich má složitost $\mathrm{O}(r)$ odpovídající průchodu seznamem všech r receptů.

```
Vareni1;  {Bellman-Ford}

const MaxN = 100000;  MaxR = 100000;
var N, R: integer;                    {počet jídel a receptů}
    H: array[1..MaxN] of integer;      {hodnoty jídel}
    S: array[1..MaxR, 1..3] of integer;   {recepty}
    i: integer;
    Zmena: boolean;

begin
readln(N, R);
for i := 1 to N do read(H[i]);
for i := 1 to R do read(S[i,1], S[i,2], S[i,3]);
Zmena := true;
while Zmena do
  begin
  Zmena := false;
  for i := 1 to R do
    if H[S[i,1]] + H[S[i,2]] < H[S[i,3]] then
      begin
      H[S[i,3]] := H[S[i,1]] + H[S[i,2]];
      Zmena := true
      end;
  end;
writeln(H[1])
end.
```

Výše uvedené řešení má již sice polynomiální časovou složitost, ale je stále ještě dost pomalé. Důvodem je skutečnost, že v každém kroku výpočtu zbytečně procházíme úplně všechny recepty, ačkoliv mnohé z nich nám v tu chvíli nijak nepomohou: u některých receptů dosud neznáme optimální ceny vstupů, jiné recepty jsme zase již s aktuálními hodnotami vstupů použili v některém z předchozích kroků. Algoritmus proto upravíme tak, abychom v každém kroku použili pouze ty recepty, které nám mohou přinést nějaký užitek, a přitom abychom nadále v každém kroku získali optimální cenu jednoho dalšího jídla.

Upravený algoritmus je založen na podobné myšlence, jako Dijkstrův algoritmus na určení délky nejkratší cesty v ohodnoceném grafu. Velmi stručně si opět připomeneme, jak Dijkstrův algoritmus pracuje a čím se liší od Bellmanova–Fordova algoritmu. Začátek je u obou těchto algoritmů stejný: pro každý uzel grafu v budeme počítat hodnotu h_v udávající délku dosud nejkratší nalezené cesty ze startovního vrcholu do vrcholu v. Stejná je rovněž inicializace těchto hodnot – startovní vrchol má počáteční hodnotu 0 a všechny ostatní vrcholy hodnotu ∞ (nekonečno). Také nyní budeme hodnoty h_v opakovaně přepočítávat a zlepšovat. V Dijkstrově algoritmu ovšem budeme navíc rozlišovat, která hodnota h_v je dočasná (tzn. možná ji v budoucnu ještě snížíme) a která už je trvalá (je už zaručeně rovna nejkratší vzdálenosti ze startovního vrcholu do vrcholu v). Na začátku výpočtu označíme všechny hodnoty jako dočasné. Dočasná hodnota h_v bude vždy určovat délku nejkratší cesty ze startovního vrcholu do vrcholu v, pokud jdeme pouze přes ty vrcholy, které již mají trvalou hodnotu.

V každém kroku výpočtu najdeme vrchol u s nejmenší dočasnou hodnotou a jeho hodnotu prohlásíme za trvalou. Pro každou hranu (u, v) vedoucí z vrcholu u do dočasného vrcholu v se potom pokusíme zlepšit hodnotu h_v tím, že bychom šli do vrcholu v z vrcholu u touto hranou. Pokud bude platit $h_u + |(u, v)| < h_v$, hodnotu h_v patřičně snížíme. Jak sami vidíte, přepočítávání hodnot vrcholů se provádí v obou algoritmech shodným způsobem, liší se pouze pořadí, v jakém tyto hodnoty přepočítáváme. V Dijkstrově algoritmu se těchto výpočtů provádí podstatně méně. Podmínkou jeho fungování ovšem je, aby ohodnocení všech hran grafu bylo nezáporné (což zpravidla bývá). V každém kroku výpočtu se jedna hodnota h_u stane trvalou, takže po n krocích výpočtu budou všechny trvalé, tzn. budeme znát nejkratší vzdálenosti ze startovního do všech ostatních vrcholů grafu.

Časová složitost závisí na způsobu uložení grafu a na konkrétní im-

plementaci algoritmu. V případě jednoduchého uložení hodnot h_v v poli dostáváme časovou složitost $O(n^2 + m)$ čili $O(n^2)$, neboť počet hran m je jistě menší než n^2. Použití haldy pro uložení počítaných dočasných hodnot vede k časové složitosti $O(n \cdot \log n + m \cdot \log n)$, což je lepší výsledek pro grafy s velkým počtem vrcholů a relativně malým počtem hran. Podrobnější popis obou algoritmů na určení nejkratší vzdálenosti v grafu včetně důkazů jejich správnosti a odvození časové složitosti najdete v každé publikaci o teorii grafů.

My se nyní ale vrátíme zpět k naší původní úloze o vaření. Podobně jako v předchozím řešení budeme postupně počítat hodnoty všech jídel. Na začátku výpočtu dokážeme každé jídlo pořídit za jeho kupní cenu – to tedy budou počáteční hodnoty všech jídel. Všechny si označíme jako dočasné, neboť možná je ještě zlepšíme. Stejně jako v Dijsktrově algoritmu bude výpočet probíhat v n krocích, přičemž v každém kroku hodnotu jednoho jídla prohlásíme za trvalou – to bude optimální cena tohoto jídla. Výpočet skončí ve chvíli, až budeme znát trvalé hodnoty všech n jídel. Během výpočtu bude stále platit, že h_v je nejnižší dosažitelná hodnota jídla v, pokud toto jídlo buď přímo koupíme, nebo ho připravíme podle receptů z těch jídel, která už mají trvalou hodnotu.

Ukážeme si, jak vypadá jeden krok výpočtu. Bude velmi podobný jako v Dijkstrově algoritmu. Najdeme jídlo s s nejmenší dočasnou hodnotou h_s a tuto hodnotu prohlásíme za trvalou. Můžeme si to dovolit, neboť za nižší cenu nedokážeme koupit ani uvařit žádné jiné jídlo s dočasnou hodnotou, které by mohlo posloužit jako surovina pro přípravu jídla s. Když jsme získali nové jídlo s trvalou hodnotou, musíme zkontrolovat a případně vylepšit dočasné hodnoty všech jídel, která můžeme z jídla s uvařit. Projdeme tedy všechny recepty typu (stu), v nichž je jídlo s jednou ze surovin, a je-li $h_s + h_t < h_u$, snížíme hodnotu h_u na $h_s + h_t$. Každý recept tedy použijeme pouze dvakrát – vždy, když jedna z jeho dvou surovin získá trvalou hodnotu.

Technicky potřebujeme vyřešit ještě dvě věci. První z nich je vhodné uložení receptů. Ke zvolenému jídlu potřebujeme projít jenom ty recepty, v nichž se toto jídlo využívá jako surovina, nechceme procházet všechny recepty. To vyřešíme snadno, stačí uložit si recepty do samostatných seznamů podle surovin. Pro každé jídlo tedy budeme mít seznam těch receptů, v nichž je toto jídlo jednou ze surovin. Každý recept bude samozřejmě uložen ve dvou takových seznamech, neboť má dvě suroviny. Druhou věcí je vhodné uložení hodnot jídel. V každém kroku výpočtu potřebujeme na-

lézt jídlo s nejmenší dočasnou hodnotou. To lze udělat jednoduše tak, že projdeme všech n jídel a porovnáme jejich hodnoty uložené v poli. Takové řešení bude mít časovou složitost $O(n^2 + r)$ neboli $O(n^2)$, neboť každý z n kroků výpočtu potřebuje čas $O(n)$ na určení jídla s nejmenší dočasnou hodnotou a každý z r receptů se během celého výpočtu použije pouze dvakrát. Jinou možností je podobně jako při implementaci Dijkstrova algoritmu použít k uložení dočasných hodnot jídel haldu. Určení jídla s nejmenší dočasnou hodnotou má pak konstantní časovou složitost, odstranění této hodnoty z haldy má složitost $O(\log n)$ a provádí se n-krát, přepočítání dočasné hodnoty každého z jídel zvládneme také v čase $O(\log n)$ a takových změn se za celý výpočet provede $O(r)$. Celková časová složitost výpočtu tak vychází na $O(n \cdot \log n + r \cdot \log n)$.

Popsaný algoritmus můžeme ještě mírně zrychlit tím, že předčasně ukončíme výpočet ve chvíli, kdy jídlo číslo 1 získá trvalou hodnotu. V tom okamžiku známe výslednou optimální cenu jídla číslo 1 a nemusíme již dopočítávat optimální ceny zbývajících jídel. Časová složitost algoritmu v nejhorším případě se tím ovšem nijak nezmění, jelikož jídlo číslo 1 může získat trvalou hodnotu až jako posledních ze všech jídel po provedení plných n kroků výpočtu. V průměru se ale doba výpočtu zkrátí.

V následující programové ukázce jsme zvolili jednodušší implementaci algoritmu bez použití haldy, tzn. řešení s časovou složitostí $O(n^2)$. Recepty ukládáme do dynamicky alokovaných lineárních spojových seznamů rozdělené podle jednotlivých surovin a výpočet ukončíme, jakmile získá jídlo číslo 1 trvalou hodnotu.

```
program Vareni2; {Dijkstra - bez haldy}

const MaxN = 100000;

type pRecept = ^Recept;
     Recept = record
                surovina, vysledek: integer;
                dalsi: pRecept
              end;

var N, R: integer;                    {počet jídel a receptů}
    H: array[1..MaxN] of integer;
                          {hodnoty jídel, záporné = trvalé}
    S: array[1..MaxN] of pRecept;
                          {recepty rozdělené podle surovin}
```

```
    S1, S2, S3: integer;
    P: pRecept;
    i, j: integer;

begin
readln(N, R);
for i := 1 to N do
  begin
  read(T, H[i]);
  S[i] := nil
  end;
for i := 1 to R do
  begin
  read(S1, S2, S3);
  new(P);
  P^.surovina := S2; P^.vysledek := S3;
  P^.dalsi := S[S1]; S[S1] := P;
  new(P);
  P^.surovina := S1; P^.vysledek := S3;
  P^.dalsi := S[S2]; S[S2] := P;
  end;

while H[1] > 0 do   {jídlo číslo 1 má dočasnou hodnotu}
  begin
  j := 1;
  for i := 2 to N do
    if (H[i] > 0) and (H[i] < H[j]) then j := i;
    {j = jídlo s nejmenší dočasnou hodnotou}
  H[j] := -H[j];  {trvalá hodnota}
  P := S[j];
  while P <> nil do
    begin
    if abs(H[j]) + abs(H[P^.surovina]) < H[P^.vysledek]
    then
      H[P^.vysledek] := abs(H[j]) + abs(H[P^.surovina]);
    P := P^.dalsi
    end
  end;

writeln(-H[1])
end.
```