INFORMATIKA

Bipartitní graf
(Úlohy z MO kategorie P, 43. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

V dnešním pokračování série článků o soutěžních úlohách Matematické olympiády kategorie P (programování) si ukážeme, jak se postupy používané při řešení různých úloh občas opakují. Autoři úloh mají k dispozici jenom omezený počet standardních algoritmů, jejichž znalost mohou od řešitelů očekávat. Olympiáda přitom existuje již 36 let a v současné době zadáváme v každém ročníku 14 soutěžních úloh: po čtyřech ve školním a v krajském kole, šest v ústředním kole. Archív soutěže tak dosahuje úctyhodného rozsahu kolem 500 úloh.

Ukážeme si dvě soutěžní úlohy ze starších ročníků olympiády, které vypadají na první pohled rozdílně a týkají se každá úplně jiné problematiky. Jejich řešení ale bude v principu naprosto shodné – v obou případech budeme zjišťovat, zda je doplněk zadaného grafu bipartitní. Studenti se znalostmi základních grafových algoritmů mohli při řešení těchto úloh využít standardní algoritmus a obě úlohy vyřešili bez velkého vlastního úsilí. Správné řešení ale dokázali nalézt i ti soutěžící, kteří nebyli seznámeni zrovna s bipartitními grafy, ale ovládali obyčejné procházení grafu do hloubky nebo do šířky, což se u řešitelů olympiády předpokládá. Zbytek už nebylo těžké samostatně vymyslet.

Obě prezentované úlohy jsou již poměrně staré – první z nich pochází z krajského kola 43. ročníku MO (školní rok 1993/94) a druhá z domácího kola 47. ročníku MO (školní rok 1997/1998). Pro potřeby našeho článku jsme původní zadání úloh formulačně trochu upravili, na charakteru úloh a jejich řešení se tím ovšem nic nezměnilo.

Úloha 1

V zemi je N měst označených čísly od 1 do N. Mezi městy je vybudována silniční síť. Každá silnice spojuje vždy dvojici měst, všechny silnice jsou obousměrné. Mezi některými dvojicemi měst přímá silnice nevede, ale z každého města je možné dojet po silnicích do libovolného jiného města, třeba i více různými způsoby. Všechna případná křížení silnic mimo města jsou mimoúrovňová (pomocí mostů) a neumožňují vozidlům přejet z jedné silnice na druhou.

Napište program, který určí, zda je možné rozdělit města do dvou skupin tak, aby každá dvojice měst patřících do stejné skupiny byla spojena přímou silnicí (jinými slovy: uvnitř každé skupiny vede přímá silnice mezi každými dvěma městy). Nezáleží přitom na velikosti jednotlivých skupin, jedna ze skupin může být případně i prázdná. Každé město ale musí být do některé skupiny zařazeno.

Vstupem programu je počet měst N a dále seznam všech silnic vedoucích mezi městy. Každá silnice je zadána dvojicí čísel měst, mezi nimiž vede. Výstupem programu je jedno vyhovující rozdělení měst do dvou skupin nebo zpráva, že žádné takové rozdělení neexistuje.

Na první pohled je zřejmé, že se jedná o grafovou úlohu. Zadanou silniční síť reprezentujeme pomocí souvislého neorientovaného grafu: vrcholy grafu představují města a hrany odpovídají silnicím. Úkolem je zjistit, zda lze všechny vrcholu grafu rozdělit do dvou skupin tak, aby obě skupiny byly tzv. *klikou*. Klikou nazýváme takový podgraf zadaného grafu, který je úplný, tzn. mezi každými dvěma jeho vrcholy vede hrana.

Úlohu vyřešíme nejsnáze tak, že budeme zkoumat *doplněk* zadaného grafu. Doplňkem grafu G rozumíme graf se stejnou množinou vrcholů, ve kterém jsou vrcholy u, v spojeny hranou právě tehdy, když v původním grafu G hrana mezi vrcholy u, v nevede. V naší úloze chceme zjistit, zda lze rozdělit vrcholy doplňku zadaného grafu do dvou skupin tak, aby v rámci každé skupiny nevedla žádná hrana, neboli aby každá hrana spojovala dvojici vrcholů náležejících do různých skupin. Graf s touto vlastností nazýváme *bipartitní*. Zadání úlohy tedy můžeme přeformulovat: chceme zjistit, zda je doplněk daného grafu bipartitní.

Řešení úlohy zahájíme načtením vstupních dat, na jejichž základě si vybudujeme graf představující doplněk zkoumané silniční sítě. Uvědomte si, že ačkoliv je silniční síť podle zadání souvislá, graf představující doplněk této silniční sítě být souvislý nemusí. Souvislost původní silniční sítě při řešení úlohy vůbec nevyužijeme.

Jak jsme již uvedli, ověření bipartitnosti grafu je založeno na postupném procházení celého grafu. Je přitom jedno, zda zvolíme procházení do hloubky pomocí rekurzivní funkce, nebo průchod grafu do hloubky pomocí zásobníku a cyklu, nebo průchod do šířky pomocí fronty a cyklu. Ve všech případech dojdeme ke stejnému správnému výsledku a ve všech případech bude mít řešení i stejnou časovou složitost.

Algoritmus řešení úlohy si můžeme názorně představit tak, že při procházení grafu a rozdělování jeho vrcholů do dvou skupin budeme jednotlivé vrcholy „obarvovat“ dvěma barvami podle toho, do které skupiny byl vrchol zařazen. Vrcholům zařazeným do první skupiny přiřadíme barvu 1 a vrcholům z druhé skupiny barvu -1. Na začátku výpočtu není žádný vrchol nijak obarven, neboť dosud nebyl zařazen do žádné ze skupin (všechny vrcholy tedy mají barvu 0). Evidence obarvení jednotlivých vrcholů nám při procházení grafu poslouží k tomu, abychom nevstupovali opakovaně do vrcholů, které jsme již navštívili a obarvili dříve. Po úspěšném ukončení výpočtu nám pak obarvení vrcholů určuje hledaný výsledek, tedy jedno přípustné rozdělení vrcholů do dvou skupin.

Procházení grafu zahájíme tím, že jeden libovolný vrchol (například vrchol s číslem 1) zařadíme do skupiny 1. Vrcholy, do kterých z něj vede hrana, nesmí patřit do stejné skupiny. Zařadíme je proto do druhé skupiny a obarvíme je barvou -1. Podobně postupujeme dále. Obecně platí, že když byl nějaký vrchol zařazen do jedné ze skupin, musí být všechny jeho sousední vrcholy zařazeny do opačné skupiny. Přitom musíme průběžně kontrolovat, zda nenastal *konflikt*. Konflikt nastane tehdy, objeví-li se v grafu dva sousední vrcholy obarvené stejnou barvou. V takovém případě rozdělení vrcholů do skupin není možné, zkoumaný graf není bipartitní a výpočet ihned ukončíme. Jinak pokračujeme v obarvování vrcholů tak dlouho, dokud jsme nuceni podle uvedených pravidel obarvovat další vrcholy (vlastně jako důsledek počátečního obarvení prvního vrcholu). Tímto způsobem projdeme celou komponentu souvislosti, do níž náleží počáteční vrchol. Pokud je graf souvislý, obarvíme takto všechny jeho vrcholy a po bezkonfliktním skončení výpočtu víme, že graf je bipartitní. V opačném případě se proces zastaví, ale v grafu ještě existují neobarvené vrcholy. Budeme tedy pokračovat stejným způsobem. Jeden libovolný z nich (například ten s nejmenším číslem) můžeme obarvit libovolnou barvou (třeba 1) a od něj opět projdeme a obarvíme jeho komponentu souvislosti. Celý výpočet skončí ve chvíli, když buď jsou všechny vrcholy grafu obarveny a zkontrolovány, že jejich obarvení nezpůsobuje žádný konflikt (graf je bipartitní,

výsledné obarvení vrcholů určuje jedno možné rozdělení vrcholů do skupin), nebo když při obarvování dojde ke konfliktu (graf není bipartitní, neexistuje žádné přípustné rozdělení jeho vrcholů do dvou skupin).

Z uvedeného postupu přímo plyne správnost algoritmu. Postupem obarvování je zajištěno, že se nikdy nemohou dostat do téže skupiny dva vrcholy grafu, mezi nimiž vede hrana. Pokud tedy nalezneme bez konfliktu obarvení všech vrcholů, určuje toto obarvení správné rozdělení vrcholů do skupin a graf je proto bipartitní. Naopak, konflikt nastane jedině ve chvíli, kdy by již obarvený vrchol měl být zařazen do opačné skupiny, než kam nyní patří, tj. když rozdělení vrcholů do skupin neexistuje. Konečnost výpočtu je dána tím, že v každém kroku je zpracován jeden vrchol a žádný vrchol není zpracováván opakovaně vícekrát. Počet kroků výpočtu je tedy roven nejvýše počtu všech vrcholů N. Každý krok výpočtu představuje provedení tolika operací, kolik hran z tohoto vrcholu vede – prochází se seznam všech jeho sousedů. Během celého výpočtu se tedy projdou dohromady seznamy následníků všech vrcholů grafu. Součet jejich délek je $2M$, kde M je počet hran procházeného doplňkového grafu (každá neorientovaná hana je v seznamech následníků uložena dvakrát). Odtud plyne časová složitost uvedeného algoritmu $\mathrm{O}(N + M)$, což lze také odhadnout jako $\mathrm{O}(N^2)$.

Budeme-li chtít odvodit časovou složitost celého řešení úlohy, musíme započítat ještě náročnost vytvoření doplňkového grafu ze vstupních dat popisujících silniční síť. Zde bude záležet na tom, jakou datovou reprezentaci grafu zvolíme. Testování bipartitnosti grafu je založeno na procházení grafu do hloubky nebo do šířky, při kterém potřebujeme ke každému zpracovávanému vrcholu co nejrychleji určit jeho sousední vrcholy. Pro efektivní implementaci takového algoritmu si můžeme uložit graf buď ve formě seznamů následníků jednotlivých vrcholů, nebo pomocí matice sousednosti. Připomínáme, že všechny silnice jsou obousměrné, takže pokud mezi městy u, v nevede silnice, v případě první uvedené volby musíme vložit vrchol v do seznamu následníků vrcholu u, ale také vložit vrchol u do seznamu následníků vrcholu v. Při druhé volbě datové reprezentace bude matice sousednosti A symetrická a musíme tedy zajistit hodnotu **True** v jejích prvcích A[u][v] a také A[v][u]. Řešení s maticí sousednosti je sice paměťové náročnější, ale implementačně jednodušší. Připravíme si matici A velikosti N^2 se samými hodnotami **True**, poté budeme číst ze vstupu jednotlivé silnice a vždy při zpracování silnice (u, v) vložíme hodnotu **False** do prvků A[u][v], A[v][u]. Tím bude doplňkový graf vytvořen v čase $\mathrm{O}(N^2)$, takže i

celé řešení úlohy bude mít asymptotickou časovou složitost $O(N^2)$.

Ukážeme si nyní jednu možnou implementaci popsaného algoritmu v programovacím jazyce Python. Použijeme v ní reprezentaci doplňkového grafu pomocí matice sousednosti, tento graf budeme procházet do hloubky s použitím vlastního zásobníku a cyklu.

```
# Úloha 1 - rozdělení měst

import sys
n = int(input("Počet měst: "))
a = [[True] * (n+1) for _ in range(n+1)]  # matice sousednosti
print("Silnice:")
for line in sys.stdin:
    line = line.split()
    u, v = int(line[0]), int(line[1])
    a[u][v] = False
    a[v][u] = False
for i in range(1, n+1):
    a[i][i] = False              # máme uložen doplňkový graf

barva = [0] * (n+1)              # barvy jednotlivých vrcholů
obarven = 0                      # počet obarvených vrcholů
konflikt = False                 # nenastal konflikt
start = 1                        # výchozí vrchol pro obarvování

while not konflikt and obarven < n:
    while barva[start] != 0:
    # hledáme první neobarvený vrchol
        start += 1
    barva[start] = 1
    obarven += 1
    seznam = [start]
    while not konflikt and seznam != []:
    # jedna komponenta souvislosti
        u = seznam.pop()
        for v in range(1, n+1):
            if a[u][v]:
                if barva[v] == 0:
                # souseda obarvíme opačnou barvou
                    barva[v] = -barva[u]
                    obarven += 1
                    seznam.append(v)
                elif barva[v] == barva[u]:
                    konflikt = True

if konflikt:
    print("Rozdělení měst do skupin neexistuje")
```

```
else:
    print("Jedno přípustné rozdělení měst:")
    for v in range(1, n+1):
        if barva[v] == 1:            # první skupina měst
            print(v, end = " ")
    print()
    for v in range(1, n+1):
        if barva[v] == -1:           # druhá skupina měst
            print(v, end = " ")
    print()
```

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

Úloha 2

Na večírku se sešlo několik hostů. Víme, kteří z nich se vzájemně znají a kteří ne. Vztah „znát se“ uvažujeme zásadně jako symetrický, takže o libovolné dvojici lidí platí, že buď se oba navzájem znají, nebo ani jeden z dvojice nezná toho druhého. Hostitel chce posadit každého hosta buď v hale, nebo v obývacím pokoji. Napište program, který určí, kolika způsoby může hostitel rozmístit své hosty do obou místností tak, aby se v rámci každé místnosti všichni navzájem znali.

Vstupem programu je počet hostů N a dále seznam těch dvojic hostů, kteří se spolu znají. Jednotlivé hosty si označíme celými čísly od 1 do N, takže na vstupu bude zadán seznam dvojic čísel hostů. Výsledkem bude jedno číslo udávající počet možných rozmístění hostů.

Příklad: Je-li $N = 4$ a znají se pouze dvojice hostů (1,2) a (3,4), má hostitel dvě možnosti: buď usadí hosty 1 a 2 do haly a hosty 3 a 4 do obývacího pokoje, nebo naopak hosté 1 a 2 budou sedět v obývacím pokoji a hosté 3 a 4 v hale. Výsledkem bude tedy číslo 2.

Také tato úloha je zjevně grafová, pomocí neorientovaného grafu můžeme reprezentovat vztahy mezi pozvanými hosty na večírku. Vrcholy grafu odpovídají jednotlivým hostům, hrany propojují dvojice vrcholů odpovídající hostům, kteří se vzájemně znají. Podobně jako v první úloze popsané výše chceme rozdělit vrcholy grafu do takových dvou skupin tak, aby obě skupiny představovaly kliku v grafu, neboli aby se všichni členové skupiny vzájemně znali. Namísto nalezení jednoho konkrétního vyhovujícího rozdělení tentokrát ale chceme určit, kolik takových rozmístění hostů existuje.

Jedná se v podstatě o úplně stejnou úlohu, jakou jsme řešili v první části článku, shodný proto bude také postup řešení: opět sestrojíme doplňkový

graf a budeme zkoumat, zda je bipartitní – tedy zda lze jeho vrcholy rozdělit do dvou skupin tak, aby uvnitř žádné skupiny nevedla žádná hrana. K tomu použijeme algoritmus procházení grafu popsaný v řešení předchozí úlohy. Vrcholy grafu přitom budeme obarvovat barvami 1 a -1, což odpovídá umístění hostů do místností označených čísly 1 a -1. Prvního hosta můžeme umístit dvěma způsoby – do jedné nebo do druhé místnosti. Jakmile zvolíme jeho umístění (třeba do místnosti 1), je nezbytné usadit do místnosti -1 všechny hosty, s nimiž se první host nezná. Pokud by se mezi nimi našli dva, kteří se neznají, úloha nemá žádné řešení (takovému stavu jsme říkali konflikt). Jinak je umístíme do místnosti -1 a jsme pak nuceni umístit do místnosti 1 všechny takové hosty, s nimiž se někdo z místnosti -1 nezná. Podobně postupujeme tak dlouho, dokud je nějaké umístění hostů takto vynuceno. Jestliže jsme právě popsaným postupem rozmístili všech N hostů, jsme hotovi a výsledkem řešení úlohy jsou původní dvě možnosti, s nimiž jsme začínali u prvního hosta (všechna ostatní obarvení vrcholů byla vynucena podmínkami úlohy). Jestliže nám ovšem zůstaly nějaké neobarvené vrcholy (zatím neumístění hosté), můžeme libovolný z nich obarvit kteroukoliv barvou a začít od něj obarvovat další vrcholy jako na začátku. Pokud se nám takto K-krát stane, že barvu vrcholu můžeme zvolit libovolně, má náš doplňkový graf přesně K komponent souvislosti a existuje tedy celkem 2^K způsobů, jak lze všechny hosty rozmístit do dvou místností.

Programová ukázka se příliš neliší od řešení první úlohy. Při procházení a obarvování doplňkového grafu si pouze navíc evidujeme prozatímní počet možností, jak lze rozmístit hosty do obou místností. Po skončení průchodu pak místo vypsání jednoho možného rozdělení vrcholů grafu do dvou skupin program vypíše výslednou hodnotu počtu možných rozmístění. Pokud graf není bipartitní a při obarvování vrcholů nastane konflikt, úloha nemá řešení a program vypíše 0. Jestliže graf bipartitní je, počet přípustných rozdělení vrcholů do dvou skupin je roven hodnotě 2^(počet komponent souvislosti). Komponenty souvislosti přitom není třeba předem vyhledávat, neboť algoritmus na testování bipartitnosti si je určuje sám.

```
# Úloha 2 - rozdělení hostů

import sys
n = int(input("Počet hostů: "))
a = [[True] * (n+1) for _ in range(n+1)]    # matice sousednosti
print("Dvojice známých:")
for line in sys.stdin:
```

```
    line = line.split()
    u, v = int(line[0]), int(line[1])
    a[u][v] = False
    a[v][u] = False
for i in range(1, n+1):
    a[i][i] = False            # máme uložen doplňkový graf

barva = [0] * (n+1)            # barvy jednotlivých vrcholů
obarven = 0                    # počet obarvených vrcholů
konflikt = False               # nenastal konflikt
start = 1                      # výchozí vrchol pro obarvování
moznosti = 1                   # počet možných rozmístění hostů

while not konflikt and obarven < n:
    while barva[start] != 0:
    # hledáme první neobarvený vrchol
        start += 1
    barva[start] = 1
    obarven += 1
    moznosti *= 2
    seznam = [start]
    while not konflikt and seznam != []:
    # jedna komponenta souvislosti
        u = seznam.pop()
        for v in range(1, n+1):
            if a[u][v]:
                if barva[v] == 0:
                # souseda obarvíme opačnou barvou
                    barva[v] = -barva[u]
                    obarven += 1
                    seznam.append(v)
                elif barva[v] == barva[u]:
                    konflikt = True

if konflikt:
    print(0)                   # rozmístění hostů není možné
else:
    print(moznosti)            # výsledný počet možných rozmístění
```