Párenie v bipartitných grafoch

Máme graf G, ktorého vrcholy sa dajú rozdeliť na dve partície A a B, je teda bipartitný. Naším cieľom je nájsť čo najväčšie párenie v tomto grafe, teda čo najväčšiu množinu nezávislých hrán – hrán, ktoré nemajú spoločný vrchol.

Zoberme si ľubovoľné párenie M obsahujúce nejaké hrany. **Striedavá cesta** vzhľadom na párenie M nazveme takú cestu, ktorá začína v nespárenom vrchole v A a striedavo ide cez hrany nepatriace páreniu a tie do párenia patriace, teda ležiace v M a táto cesta končí v komponente B. Ak je naviac vrchol v B, v ktorom končí striedavá cesta, nespárený, takúto cestu voláme **zlepšujúca cesta**. Dôvod tohto označenia je jasný. Ak zoberieme ľubovoľnú zlepšujúcu cestu, hrany na tejto ceste, ktoré patria do M odtiaľ vyberieme a vložíme tam hrany, ktoré do M nepatrili, dostaneme párenie M', ktoré je o jednu hranu väčšie ako M.

To nás privádza na myšlienku jednoduchého greedy algoritmu na hľadanie maximálneho párenia. Začneme s prázdnym párením a postupne budeme hľadať zlepšujúce cesty. Vždy keď nájdeme nejakú zlepšujúcu cestu, vymeníme hrany na nej, čím zväčším naše párenie. V okamihu keď nenájdeme už žiadnu zlepšujúcu cestu vyhlásime párenie za maximálne.

Je však tento algoritmus správny? Nemôže sa nám stať, že sa dostaneme do lokálneho maxima, v ktorom už neexistuje zlepšujúca cesta, ale zároveň toto párenie nie je maximálne? Aby sme mohli takýto algoritmus použiť, musíme si dokázať nasledujúcu vetu.

Veta 1: Párenie M je maximálne práve vtedy, keď neexistuje zlepšujúca cesta vzhľadom na M.

Dôkaz:

Keďže dokazujeme ekvivalenciu, potrebujeme dokázať dve implikácie. Ako to častokrát býva, jedna z týchto implikácií je jednoduchšia ako tá druhá. V tomto prípade to bude implikácia zľava doprava.

Majme maximálne párenie M. Je jasné, že takéto párenie neobsahuje zlepšujúcu cestu. Ak by obsahovalo, tak viem toto párenie zväčšiť, čo je spor s tým, že bolo maximálne.

Pred sebou máme už len tú zaujímavú implikáciu, ktorú naozaj potrebujeme na funkčnosť nášeho algoritmu. Zoberme si ľubovoľné maximálne párenie M a ľubovoľné párenie M', v ktorom neexistuje zlepšujúca cesta. Ak dokážeme, že $|M| = |M'|$, tak sme dokázali celú implikáciu. Zoberme si graf G, v ktorom sme našli tieto dve párenia a označme hrany oboch párení M a M'.

Ak je nejaká hrana v G neoznačená oboma páreniami, môžeme ju ignorovať, lebo nijak nezasahuje do veľkosti týchto množín. Takisto môžeme ignorovať všetky hrany, ktoré sa nachádzajú aj v M aj v M', lebo ich zanedbanie nám odstráni rovnako veľa hrán v oboch páreniach. Ostali nám teda hrany, ktoré ležia v práve jednom z párení M a M'.

Uvedomme si, že graf, ktorý nám zostal má maximálny stupeň 2, lebo z každého vrcholu môže vychádzať najviac jedna hrana patriaca M a najviac jedna hrana patriaca M'. Ako teda vyzerajú komponenty tohto grafu. Môžu tam byť izolované vrcholy, cesty a kružnice. Izolované vrcholy nám nevadia, lebo neovplyvňujú veľkosti párení. Kružnice v tomto grafe musia byť párne, lebo sa na nich striedavo nachádzajú hrany z M a z M' a nemôžu byť vedľa seba dve hrany z toho istého párenia – potom by to nebolo párenie. No a párna kružnica má rovnako veľa hrán v M ako v M'. Ostávajú nám teda cesty.

Párne cesty sú v poriadku, lebo opäť pridajú rovnako veľa hrán do oboch párení. A nepárne cesty tu existovať nemôžu, lebo by boli zlepšujúcimi cestami pre jedno (to minoritné) z párení. A to je spor, lebo párenie M' neobsahuje zlepšujúce cesty a párenie M je maximálne, preto tiež neobsahuje zlpšujúce cesty.

Tým pádom majú párenia M a M' rovnako veľa hrán a teda ak máme párenie bez zlepšujúcej cesty, tak toto párenie je maximálne.

Z platnosti predchádzajúcej vety platí, že vyššie spomenutý algoritmus je korektný. Naším cieľom však je ho aj naozaj implementovať, preto sa v ďalšej časti budeme zaoberať tým, ako takýto algoritmus reálne naprogramovať.

Algoritmus na hľadanie najväčšieho párenia v bipartitných grafoch

Najdôležitejšia časť tohto algoritmu bude hľadanie zlepšujúcej cesty. Na začiatku teda máme nejaký nespárený vrchol v A a z neho budeme hľadať postupnosť neoznačených a označených hrán, ktoré skončia v nespárenom vrchole v B.

Na pamätanie dôležitých informácií nám poslúži niekoľko polí. Zaprvé si budeme potrebovať zapamätať samotný graf G. Ten budeme mať uložený ako zoznam susedov v poli G (v C++ sa najlepšie na toto hodí `vector<vector<int> >`). Zadruhé musíme vedieť, ktorý vrchol je spárený s ktorý iným vrcholom. Na to nám poslúži pole $sparene[]$. Na i-tom mieste tohto poľa bude -1, ak vrchol i nie je spárený a číslo vrchola, s ktorým je spárený v opačnom prípade. Naviac ešte podotknem, že vrcholy si číslujeme od 0 po $n-1$.

Majme nespárený vrchol $v \in A$. Všetky hrany, ktoré z neho vedú sú nespárené, postupne sa teda pozrime na všetky z nich. Dostaneme sa do suseda vrchola v, ktorého si označíme w. Máme dve možnosti. Vrchol w je nespárený. V tom prípade sme našli zlepšujúcu cestu, upravíme si pole $sparene[]$ a ukončíme hľadanie. Ak je však vrchol w spárený, pri hľadaní zlepšujúcej cesty musíme teraz prejsť po tejto spárenej hrane do vrcholu $sparene[w]$. V tomto vrchole sme však vo veľmi podobnej situácii ako sme boli na začiatku s vrcholom v. Sme v komponente A a hľadáme zlepšujúcu cestu z tohto vrchola.

To nás vedie k myšlienke rekurzívnej funkcie. Jej úloha bude nasledovná: *Existuje striedavá cesta z vrchola v, ktorá končí v nespárenom vrchole množiny B?* Vracať bude samozrejme jednu booleanovú hodnotu `true` alebo `false` podľa toho, aká je odpoveď na túto otázku.

Algoritmus teda vyzerá takto: zavoláme rekurzívnu funkciu na vrchol v. Vnútri nej sa pozeráme postupne po všetkých jeho susedoch w. Ak je w nespárené, upravíme pole $sparene[]$ a vrátime hodnotu `true` a ukončíme algoritmus. Ak je w spárené, zavoláme sa rekurzívne na vrchol $sparene[w]$. Ak mi toto rekurzíve volanie vráti `true`, opäť môžeme upraviť pole $sparene[]$ (o všetko ostatné sa postarala rekurzia) a vrátiť hodnotu `true`. Ak sa nám nepodarilo vrátiť `true` pre žiadneho suseda v, smutne skonštatujeme, že z vrchola v nevedia zlepšujúca cesta a vrátime `false`.

Naviac si uvedomme, že sa nám neoplatí volať túto rekurziu dvakrát na ten istý vrchol. Ak mi totiž rekurzia v tomto vrchole vrátila `true`, tak sa algoritmus hneď ukončí a teda druhýkrát sa zavolať nemôže, a ak mi vrátila `false`, tak mi túto hodnotu vráti aj teraz, lebo sa zatiaľ nič nezmenilo. Preto si budeme ešte v poli $T[]$ pamätať, z ktorých vrcholov sme už volali rekurzívnu funkciu. Celý algoritmus je vlastne len upravené prehľadávanie do hĺbky (DFS).

Náš algoritmus teda opakovane spúšťa túto funkciu a snaží sa hľadať zlepšujúce cesty. Pritom si dáva pozor, aby volanie tejto funkcie začínalo vždy v nespárenom vrchole v A. Koľkokrát to však opakovať? A z ktorých vrcholov spúšťať túto rekurziu? Keby sme to robili bez rozmyslu, zabralo by to príliš veľa času.

K tomu pomôže nasledovná úvaha. Predstavme si, že máme iba komponent B a A je zatiaľ prázdne. Naše párenie je zatiaľ prázdne, tým pádom však aj maximálne, keďže graf neobsahuje žiadnu hranu. Pridajme teraz niekoľko vrcholov z A aj so všetkými hranami, ktoré z nich vedú do B a predpokladajme, že máme nájdené maximálne párenie v tejto časti grafu.

Pridajme nový vrchol v. Maximálne párenie v tomto novom, o jedna väčšom grafe, nám mohlo narásť najviac o 1. Ak by totiž narástlo o viac, po odstránení tohto vrcholu by sme odstránili najviac jednu spárenú hranu, ale tým pádom nami nájdené párenie nebolo najväčšie. Ak narástlo o 1, tak to znamená, že tam pribudla jedna zlepšujúca cesta, ale táto môže viesť iba z vrchola v, lebo jeho pridanie spôsobilo vznik tejto zlepšujúcej cesty – bez neho tam predsa nemohla byť. To znamená, že nám stačí jedno rekurzívne volanie z vrchola v a po ňom budeme mať maximálne párenie na tomto o jedna väčšom grafe. Túto myšlienku postupne aplikujeme ďalej, pridávaním ďalších vrcholov.

Čo je super na tejto myšlienke je to, že nemusíme implementovať to, že nejaká časť grafu sa ešte nepridala. Ak som nehľadal z nejakých vrcholov zlepšujúcu cestu, tak tieto vrcholy nemajú ako zasiahnuť do aktuálneho hľadania, preto to pre algoritmus naozaj vyzerá, akoby som vrcholy postupne pridával.

Všetky tieto poznatky nás dovedú k veľmi jednoduchej implementácii. Jej časová zložitosť je rádovo $O(n(n+m))$, teda $O(nm)$. Rekurzívna funkcia má zložitosť $O(n+m)$ (lebo DFS) a voláme ju najviac n krát.

Listing programu (C++)

```
int n; //pocet vrcholov
vector<vector<int> > G;
vector<int> A,B; //zoznam vrcholov v prislusnych komponentoch
vector<bool> T;
vector<int> sparene;

bool zlepsujuca_cesta(int v) {
    T[v]=true;
    for(int i=0; i<G[v].size(); i++) {
        int w=G[v][i];
        if(sparene[w]==-1) {sparene[w]=v; sparene[v]=w; return true;}
        if(!T[sparene[w]] && zlepsujuca_cesta(sparene[w])) {sparene[w]=v; sparene[v]=w; return true;}
    }
    return false;
}

void maximalne_parenie() {
    sparene.resize(n,-1);
    for(int i=0; i<A.size(); i++) {
        int v=A[i];
        T.clear(); T.resize(n,false); //nezabudnut nanovo vymazat pole T
        zlepsujuca_cesta(v);
    }
}
```

Najmenšie vrcholové pokrytie a najväčšia nezávislá množina

Problém vrcholového pokrytia je problém hľadania takej množiny vrcholov, že každá hrana susedí s aspoň jedným vybraným vrcholom. A nezávislá množina je množina vrcholov, kde nevedie hrana medzi dvoma vybranými vrcholmi.

Dôležité pozorovanie je, že tieto dva problémy sú na seba duálne. Každé vrcholové pokrytie C určuje nezávislú množinu $I = V(G) - C$. Existencia hrany v nezávislej množine by totiž znamenala, že táto hrana nebola pokrytá príslušným vrcholovým pokrytím a naopak. Minimalizácia vrcholového pokrytia teda vedia k maximalizácii nezávislej množiny a preto problém riešiaci jeden z týchto problémov rieši v zápätí oba.

Na prednáške sme si spomínali Konigovu vetu, ktorá hovorí o tom, že veľkosť najväčšieho párenia je rovnaká ako veľkosť najmenšieho vrcholového pokrytia. Celý dôkaz tohto tvrdenia môžete nájsť v Diestelovej Graph Theory a tu ho nebudem odvádzať.

V rámci toho dôkazu sa však vyskytla nasledovná konštrukciu množiny U. Zobrali sme si maximálne párenie M. Pre každú hranu $ab \in M$ sme do U zaradili práve jeden vrchol. Vrchol b sme do U zaradili vtedy, ak existovala striedavá cesta vzhľadom na M, ktorá končila vo vrchole b (Uvedomte si, že striedavé cesty nemusia byť čo najdlhšie. Aj jednohranová cesta je striedavá cesta). V opačnom prípade sme do U zaradili vrchol a.

Následne sme pomocou tejto konštrukcie ukázali, že U je vrcholové pokrytie a to dokonca najmenšie možné. Zaujímavé na tomto dôkaze ale je, že je to dôkaz konštrukčný. Priamo nám dáva návod, akým sa dá zostrojiť najmenšie vrcholové pokrytie. Našou úlohou je to už len premeniť na algoritmus.

Predpokladajme, že máme nájdené maximálne párenie M. Cieľom bude nájsť všetky vrcholy, cez ktoré prechádza nejaká striedavá cesta, aby sme potom vedeli rozhodovať, ktorý vrchol z párenia pridať do množiny U.

Striedavé cesty začínajú v nespárených vrcholoch množiny A. Odtiaľ ideme nespárenými hranami (to znamená všetkými) do komponentu B. Aby sme zachovali striedavosť, z komponentu B musíme odísť po spárenej hrane, čím sa dostaneme opäť do komponentu A. Odtiaľ musíme ísť nespárenou hranou do B atď. Opäť je jasné, že sa nám neoplatí hľadať z jedného vrcholu viac ako raz. Tým pádom vieme tento postup implementovať či už pomocou DFS alebo BFS v čase $O(n + m)$. Teraz už len zoberieme všetky spárené hrany a do najmenšieho vrcholového pokrytia pridáme správne vrcholy, podľa toho čo nám vravela Konigova veta.

Ide to však aj o chlp jednoduchšie. Označujme si všetky vrcholy, ktorými prechádzal náš vyššie spomenutý algoritmus na hľadanie striedavých ciest. Bez ohľadu na to, v ktorom komponente tieto vrcholy ležia. Následne platí, že do najmenšieho vrcholového pokrytia patria všetky neoznačené (nenavštívené) vrcholy komponentu A a všetky označené (navštívené) vrcholy komponentu B. Rozmyslite si prečo.[1]

[1] Implementáciu naschvál vynechávam. Však niečo na tú úlohu musíte spraviť aj vlastné :P A dúfam, že slovný popis bol dostatočný, aby ste to zvládli.