Edmonds blossom algoritem

# Povzetek

Blossom(cvet) algoritem nam pomaga poiskati največje prirejanje v grafu z uporabo t.i. skrčenja ciklov(cvetov). Pri tej temi bomo obravnali problem kako določiti največje prirejanje v grafu, ki ni nujno dvodelen, v polinomskem času. Opisane so zaporedna, vzporedna(parallel) in porazdeljena(distributed) verzija. Za primerjavo vzamemo še časovno zahtevnost vseh verzij in pogledamo v katerih primerih se posamezna najboljše obnese.

# UVOD

Blossom(cvetni) algoritem je algoritem znotraj teorije grafov, ki se uporablja za iskanje največjega prirejanja. Leta 1961 ga je odkril Jack Edmonds in je prvi algoritem, ki najde največje prirejanje v poljubnem grafu(ne samo dvodelnem – madžarska metoda). Z njim lahko rešujemo probleme kot so iskanje Hamiltonove poti/cikla, problem trgovskega potnika... V prihodnjih letih so odkrili različne izboljšave, vendar se tu omejimo na osnovno verzijo, ki deluje na neusmerjenih grafih, ki nimajo uteženih povezav.

## DEFINICIJE

**Prirejanje** v grafu G je podmnožica vseh povezav v G tako, da si nobeni 2 povezavi v podmnožici ne delita vozlišča.

**Največje(maksimum) prirejanje** **M** v grafu G je tako prirejanje, ki vsebuje maksimalno število povezav iz G. Za vsako prirejanje M' v G velja torej |**M**| >= |M'|.

Primer maks. prirejanja:

Maximum-matching-labels.svg(Vir:Wikipedia)

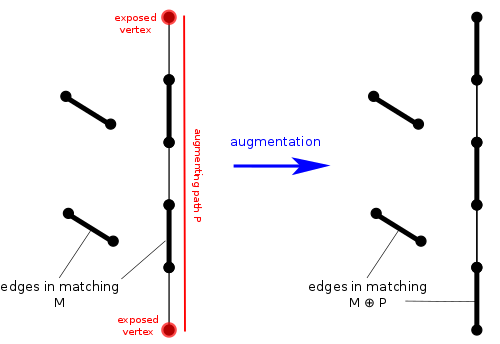
**Izpostavljeno vozlišče** je vozlišče, ki ni vsebovano v prirejanju M, je pa v grafu G. Vsa vozlišča vV(G\M) so izpostavljena vozlišča.

**Povečujoča pot** je pot lihe dolžine(liho število povezav), ki ima za začetno in končno vozlišče izpostavljeno vozlišče. Vsaka soda povezava znotraj poti mora biti vsebovana v prirejanju, vsaka liha povezava ne sme biti v prirejanju. **Povečujoča pot** je lihe dožine.

**Povečanje poti(matching augmentation)**:

Imamo povečujočo pot P, prirejanje M in graf G. Z povečanjem poti P dobimo novo prirejanje M1;





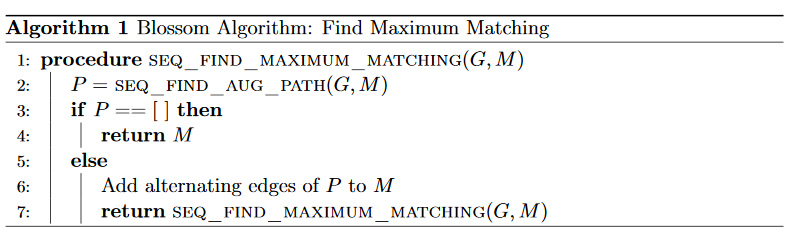
**Izrek:** Pomemben izrek pri Blossom algoritmu(BA v nadaljevanju) je, da se s povečanjem povečujoče poti P, znotraj grafa G, poveča moč M za 1.

Dokaz je očiten.

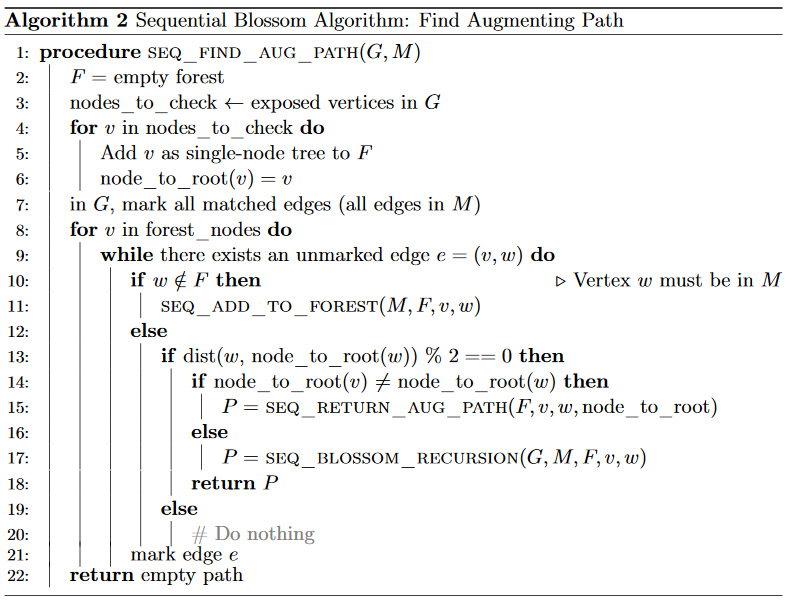
**Bergeov izrek(Theorem of correctness)** Prirejanje M na grafu G je največje, čče v G ni nobene povečujoče poti.

# Zaporedni algoritem

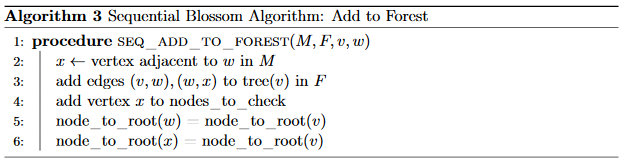
V okviru seminarske naloge si bolj podrobno pogledamo samo zaporedni algoritem, druga dva bom opisal na kratko, ker nista bila uporabljena pri predstavitvi. Predhodno definirani izreki bodo uporabljeni v tem delu opisa.

Celoten algoritem je sestavljen iz manjših delov, ki jih bom predstavil po vrsti. Najprej si poglejmo okvir(wrapper) funkcijo za celotni algoritem: 

Kot je razvidno iz kode bomo iskali povečujočo pot in jo povečali, dokler v grafu G več ne bo nobene povečujoče poti. Po Bergeovem izreku sledi, da je dano prirejanje največje. Zdaj lahko analiziramo dejanski algoritem SEQ\_FIND\_AUG\_PATH:

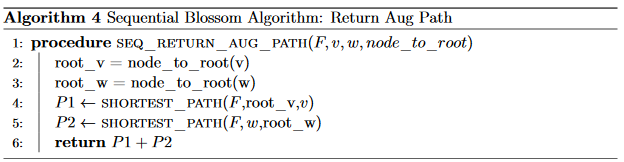


Kot parameter podamo graf G in prirejanje M(povezave). Izpostavljena vozlišča v G so koreni dreves v gozdu F. Algoritem iterira skozi ta vozlišča(korene dreves) in jim dodaja povezave ter tako gradi povečujoče poti. Iteriramo skozi vsa vozlišča, ki so sode dolžine od korena, tako zagotovimo, da je v posameznem drevesu vedno povečujoča pot.

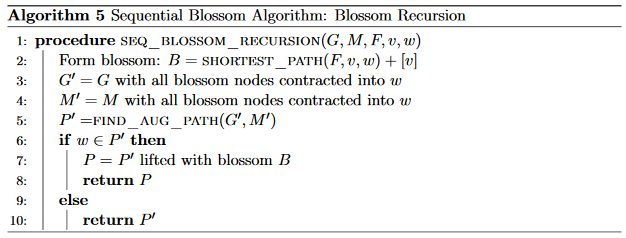


Ko se dve takšni alternirajoč poti povežeta preko proste povezave, je algoritem našel povečujočo pot. Bolj natančno, ko algoritem prispe k povezavi(*v,w*), ki povezuje dve drevesi, vzame najprej pot od korena *v* do vozlišča *v*, ji doda novo povezavo(*v,w*) in doda še pot od *w* do korena *w*. Če je deloval pravilno, bo takšna pot povečujoča.

Paziti je treba, ko je *w* za liho dolžino oddaljeno od korena. V tem primeru se nič ne zgodi in zapustimo zanko.

Včasih se zgodi, da algoritem doda novo vozlišče, s katerim se v drevesu ustvari cikel. Takšnemu ciklu(ki je alternirajoč cikel lihe dolžine) pravimo cvet(Blossom). Dokaz za obstoj je opisan v viru(https://stanford.edu/~rezab/classes/cme323/S16/projects\_reports/shoemaker\_vare.pdf - Lemma 2.5). Glavno je vedeti, da bo v tej točki algoritem pravilno zasledil cikel(cvet), če se je koda pravilno izvedla.

Sedaj se izvede rekurzivno krčenje cikla:



**Def:** Novo vozlišče **v**, ki sklene cikel imenujemo **baza cveta**.

Potem ko algoritem skrči cvet v eno vozlišče **w**, nadaljuje z rekurzijo. Ko najde povečujočo pot, se cvet razširi nazaj. Če povečujoča pot ne vsebuje vozlišča **w**, jo enostavno vrnemo tako kot je. Če pa pot vsebuje vozlišče **w**, cvet razširimo(lift) in tako dobimo pot, ki gre skozi cikel. Kako bo pot potekala skozi cikel, pa je odvisno od drugih faktorjev. Vozlišče **w** je lahko konec povečujoče poti **P'**, lahko je pa na njeni sredini. Povezavam v **w** bomo rekli **stebla(stems)**.Če hočemo razširiti-povzdigniti(lift) graf, moramo najprej razmisliti, katerim vozliščem v cvetu so **stebla** sosednja. Upoštevamo sledečo trditev;

**Trditev:** Če **w** ni konec povezave, mora biti eno od **stebel** povezano z **bazo**.

**Dokaz:** Spomnimo, da M' vsebuje natanko eno od dveh stebel, ker sta vsebovani v alternirajoči poti v G'. Nepokrito steblo ima lahko krajišče v kateremkoli vozlišču v grafu G. Pokrito steblo pa mora imeti krajišče v **bazi**, ker bi drugače imelo skupno krajišče z dvemi povezavami iz M, kar pa je protislovje, saj v tem primeru M nebi več bilo pokritje.

#####***Do konca prevedido časovne kompleksnosti, tudi dokaze. Vzporednega na kratko pa distributiranega pa samo omeni***