# Blossom algoritem

# Problem

Določi največjo podmnožico povezav v grafu, tako da nobeno vozlišče ne bo dotaknjeno več kot 1x.

# Uvod

Algoritem je razvil Jack Edmonds leta 1961, objavljen je bil leta 1965.

Prvi algoritem, ki išče maksimalno prirejanje v polinomskem času in dela na vseh grafih.

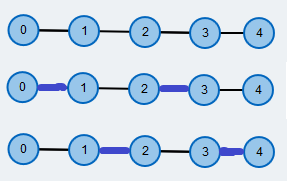
# Definicije

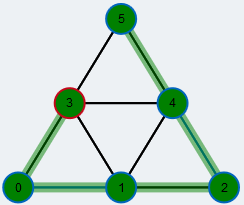
**Prirejanje** M v grafu G je podmnožica povezav v G, tako da nobeno vozlišče ni vključeno več kot 1x.

**Maksimalno prirejanje** M v grafu G je prirejanje, ki vsebuje maksimalno št povezav v G. Za vsako prirejanje M' velja |M| >= |M'|.

**Popolno prirejanje**, vsako vozlišče je dotaknjeno z prirejanjem

**Povečujoča pot** je pot, ki se začne in konča v prostem vozlišču in alternira med povezavami, ki so v prirejanju in niso. Povečujoča pot je pot lihe dolžine. Sode dolžine ne more biti, ker bi potem le zamenjali povezave, ki so v prirejanju s povezavami, ki niso in prirejanje se ne bi povečalo in ker se bodisi začetno bodisi končno vozlišče dotika prirejanja.



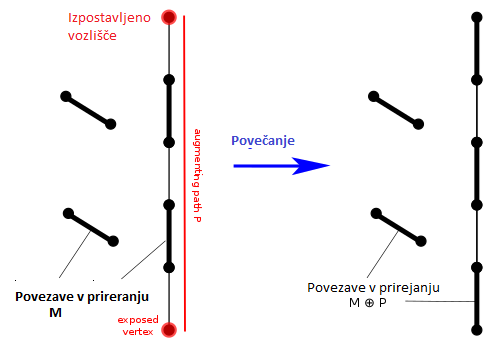


Slika 1 Povečujoča pot

Modri povezavi sta v trenutnem prirejanju, v naslednjem koraku dobimo 3 povezave v prirejanje.

Za graf G z prirejanjem M je **izpostavljeno vozlišče** v, ki ne pripada M, ampak je v G.

V(G\M) so izpostavljena vozlišča.



# Trditev

S povečujočo potjo in prirejanjem M, z odstranitvijo povezav v prirejanju in z dodajanjem, ki niso v prirejanju povečamo velikost za 1.

# Dokaz

Z indukcijo

# Algoritem

Vhod: Graf G, Prirejanje M

Izhod: Povečujoča pot, če obstaja, sicer prazna pot.

Iskanje poti za povečanje uporablja pomožno podatkovno strukturo gozd F, katerega posamezna drevesa ustrezajo določenim delom grafa G.

# Bergejeva lema

Z Bergejevo lemo je prirejanje M največje, če in le če v G ne obstaja M-širitvena pot G. Zato je bodisi prirejanje največje ali pa se lahko poveča.

Tako lahko z začetnega ujemanja izračunamo maksimalno prirejanje, tako da povečamo trenutno prirejanje z razširitvami poti, dokler jih lahko najdemo in se vrnemo, če ni več poti za povečanje.

Algoritem lahko formaliziramo na naslednji način:

VHOD: Graf G, začetno prirejanje M na G

IZHOD: Maksimalno prirejanje M\* na G

A1 najdi\_največje\_prirejanje( G, M ) : M\*

A2 P ← najdi\_povečujočo\_pot( G, M )

A3 Če P ni prazna, potem

A4 vrni najdi\_največje\_prirejanje( G, prirejanje M na P )

A5 Sicer

A6 Vrni M

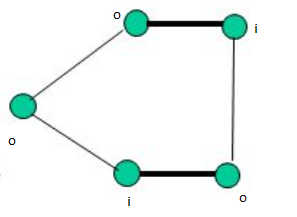
**Blossom** je ciklel dolžine 2k+1, k povezav pripada prirejanu M.



Slika Blossom dolžine 5 z 2 povezavama v prirejanju

# Zaznavanje blossomov

1. Preglej graf, začni v izpostavljenem vozlišču
2. Začetnega označi z „o“
3. Vozlišča označi z „i“ in „o“ alternajoče.
4. Če sta začetni in končni označena z „o“ je blossom.



Slika Zaznavanje Blossoma.

# Algoritem

V vsakem koraku iteracije algoritem najde:

Povečujočo pot

Blossom

Zaključi, da ni povečujočih poti

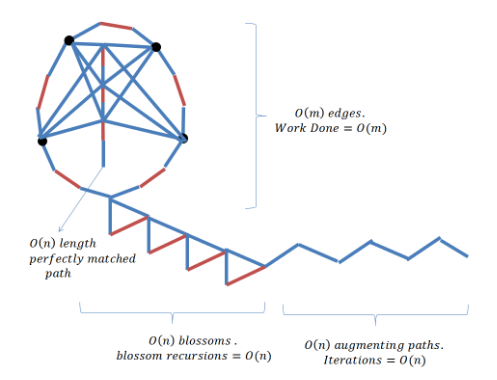
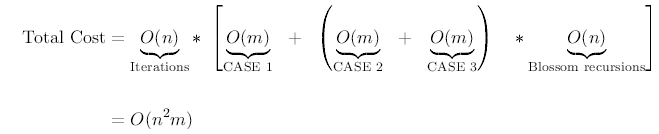
# Časovna zahtevnost zaporednega algoritma

Blossom algoritem zahteva največ klicev funkcije najdi\_povečujočo\_pot (find\_aug\_path)

Blossomova rekurzija

Dodaj v gozd

Vrni povečujočo pot

# Časovna zahtevnost vzporednega algoritma

Dodaj v gozd O(1)

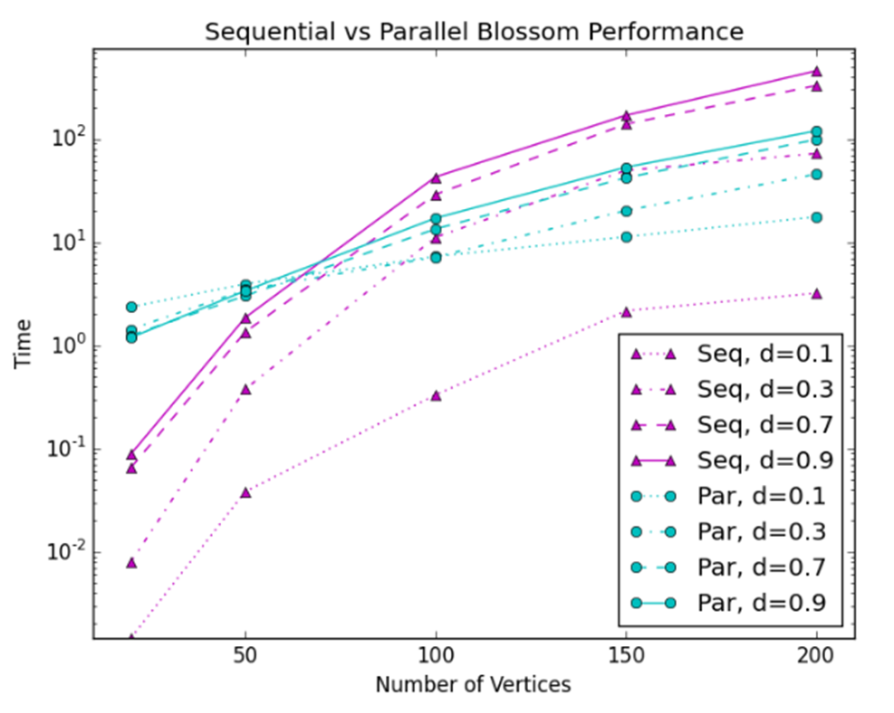
Blossomova rekurzija O(1)

Vrni povečujočo pot O(n)

Časovna zahtevnost O(n^3)

Ni odvisna od števila povezav





Slika Časovna zahtevnost zaporednega in vzporednega algoritma d mera za polnost grafa

# Izboljšave

BFS izvedemo na vseh prostih točkah – s tem lahko najdemo več disjunktnih poti hkrati.

Vseh blossomov ni treba krčiti takoj – obstajajo Blossomovi pogoji, ki določajo, ali moramo krčiti blossome ali lahko odložimo

Posebno označevanje dovoljuje hitro širjenje blossomov

Micali in Vazirani (1980) O(|E|)

# Viri

<http://www.imsc.res.in/~meena/matching/edmonds.pdf>

<https://en.wikipedia.org/wiki/Blossom_algorithm>

<https://www-m9.ma.tum.de/graph-algorithms/matchings-blossom-algorithm/index_en.html>

<http://slideplayer.com/slide/10831654/>

(vizualizacija)

<https://brilliant.org/wiki/blossom-algorithm/>

<https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/shoemaker_vare.pdf>