# 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.

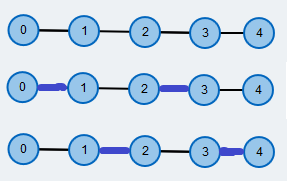
# Definicije

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

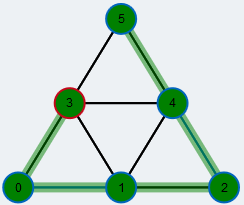
**Maksimalno prirejanje** M v grafu G je prirejanje, ki vsebuje maksimalno št povezav v G. Za vsako prirejanje M' velja |M| >= |M'|.

**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 Primer povečujoča pot ni sode dolžine

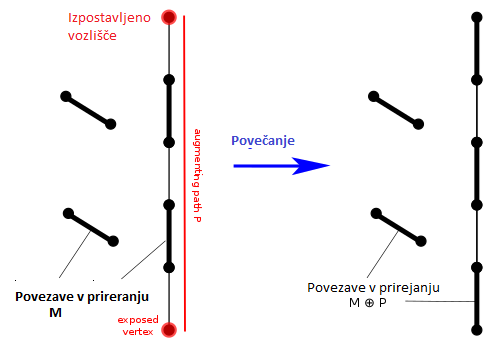


Slika Povečujoča pot

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

Denimo, da je povečujoča pot dolžine 1. Z dodajanjem poti v prirejanje povečamo prirejanje M za 1.

Denimo da imamo povečujočo pot dolžine 2k-1 in denimo da imamo povečujočo pot dolžine 2k+1. Povečujoča pot ima 2k+1 povezav, ki alternirajo v prirejanu/niso v prirejanju, vemo, da končna vozlišča niso v prirejanju. Pustimo končni vozlišči in njuna soseda, imamo 2k-2 vozlišč, ki določajo povečujočo pot dolžine 2k-3, ta povečujoča pot izpusti 2 povezavi, ki nista v prirejanju in 2 povezavi, ki sta v prirejanju v originalni povečujoči poti. Naj bo M' prirejanje M brez 2 povezav, ki sta v prirejanu. Potem po indukcijski predpostavki z zamenjavo povezav v 2k-3 povečujoči poti v M' ohranjamo M' kot prirejanje in povečamo prirejanje za 1. Naj bo M'\* spremenjeno prirejanje, za katerega velja |M'\*| = |M|-1. Naj bo |M\*|= |M'\*| + povezave(v1,v2) in v(2k+1),v(2k+2)). Potem |M\*| = |M'| + 2 = |M| + 1. M^ je še vedno prirejanje, zato lahko zaključimo, da to drži za povečujoče poti vseh lihih dolžin.

# 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

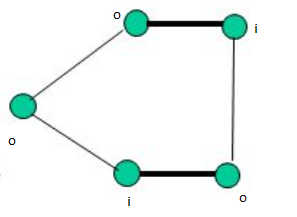
**Blossom** je ciklel dolžine 2k+1, k povezav pripada prirejanu M.



Slika 3 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 4 Zaznavanje Blossoma.

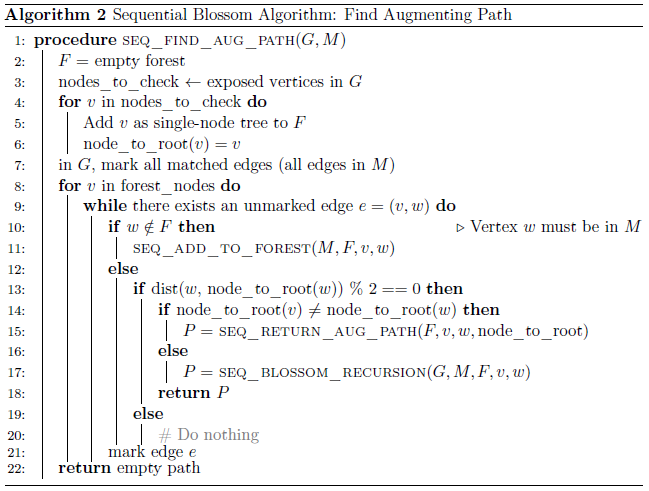
# Algoritem

V vsakem koraku iteracije algoritem najde:

Povečujočo pot

Blossom

Zaključi, da ni povečujočih poti



Slika Zaporedni algoritem [1]

# Časovna zahtevnost zaporednega algoritma

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

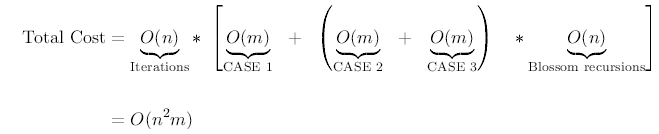
Blossomova rekurzija

Vsa vozlišča v blossomu stisnemo v eno vozlišče.

Ker gremo čez vse povezave in vozlišča to prinese O(E + V ) = O(m). Po zgornji predpostavki, metoda naredi O(n) rekurzivnih klicov.

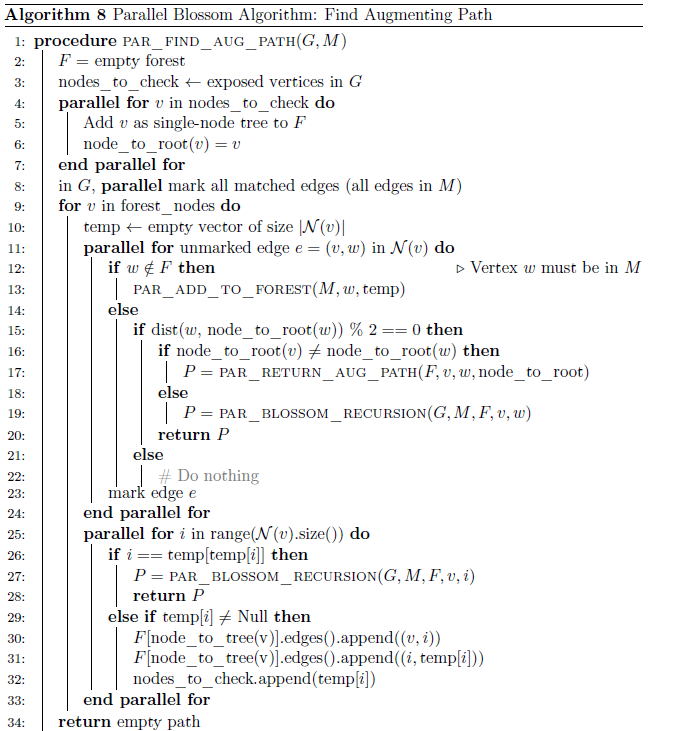
Dodaj v gozd: Dodajanje povezav v gozd je O(1) operacija, največ m povezav moramo dodati, zato je O(m)

Vrni povečujočo pot: Ker iščemo najkrajšo pot na drevesih v gozdu, naredimo DFS (pregled v globino), to zahteva O(m) operacij.



Slika Analiza zaporednega algoritma [1]

## Vzporedni algoritem



# Časovna zahtevnost vzporednega algoritma

Dodaj v gozd O(1)

Zbiranje povezav, ki jih želimo dodati v začasno tabelo O(1).

Post-procesiranje O(1)

Blossomova rekurzija:

Vsaka blossomova rekurzija zahteva O(1) korakov, v najslabšem deg(v) <= O(n)

Vrni povečujočo pot, kot pri zaporednem algoritmu 2x pokličemo funkcijo najdi\_najkrajšo\_pot in izvedemo pregled v globino, ki zahteva O(n) operacij.

Časovna zahtevnost O(n^3)

Ni odvisna od števila povezav



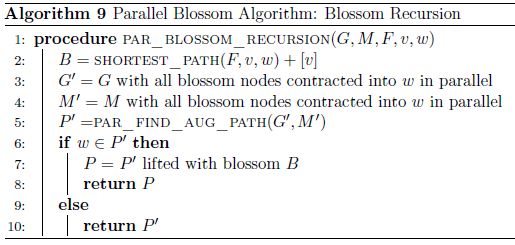
Če ne najdemo v algoritmu blossomov ali povečujoče poti v glavni pararelni for zanki (vrstice 11-24) v algoritmu, moramo post-procesirati dodati nove povezave v gozd in preveriti, če lahko dobimo kakšen cikel (blossom). To je enostavno, ker po izreku se lahko pojavijo le blossomi dolžine 3.

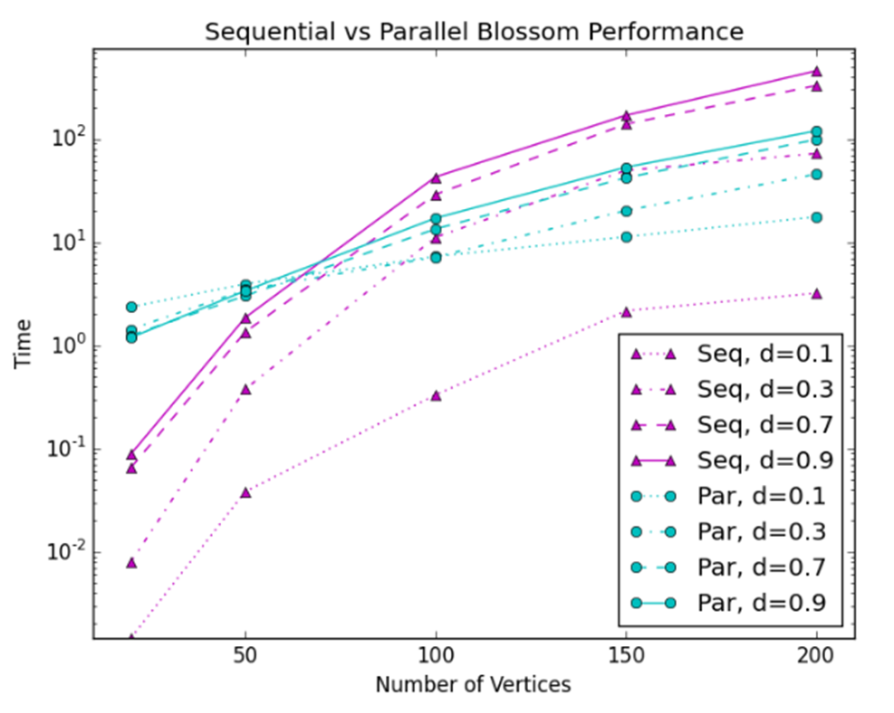
## Izrek

Denimo, da smo sprocesirali povezavo v incidentu od vozla v v vzporednem in smo našli neoznačeno povezavo (v,W), dodajanje v gozd le povezave (v,w) , po katerem ne dobimo povečujoče poti ali blossoma. Potem po dodajnju vseh parov povezav (v,w), (w,x) kateri w ∉ F in (w,x) ϵ M, edini možni cikli, ki se pojavijo so blossomi dolžine 3.

Dokaz (stran 16) [1]

S post-procesiranjem le preverimo ali je nova povezava del 3 cikla (blossoma). To naredimo v O(1). Če kateri »procesor« najde blossom, vse ostale »procesorje« izklopimo in kličemo blossom rekurzijo (algoritem 9).





Slika 7 Časovna zahtevnost zaporednega in vzporednega algoritma log log plot [1]

Vijolično je označen čas izvajanja zaporednega algoritma, turkizno pa vzporednega (pararelnega), glede na število vozlišč, d je mera za polnost grafa.



Na sliki vidimo, da če imamo malo povezav v grafu se splača uporabiti zaporedni algoritem, če graf polnejši in če je veliko vozlišč (>70) se bolj splača uporabiti vzporedni algoritem.

# 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>

(vizualizacija)

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

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

[https://stanford.edu/~rezab/classes/cme323/S16/projects\_reports/shoemaker\_vare.pdf [1](https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/shoemaker_vare.pdf%20%5b1)]

<https://math.stackexchange.com/questions/1526372/what-is-the-definition-of-the-density-of-a-graph>