Problemi v grafih

Minimalno vpeto drevo, Barvanje grafov, Najkrajše poti, Pretoki

Tomaž Dobravec, Algoritmi in podatkovne strukture 2

Podatkovna struktura Graf

- Graf je struktura, v kateri imamo točke in povezave;
- Z grafom ponazarjamo relacije, na primer: relacija ”vodi pot”

- Če so povezave usmerjene → takrat govorimo o ”usmerjenem grafu”

- Množico točk grafa označimo z `V`, množico povezav pa z `E`;
  - z oznako `G(V,E)` opišemo graf z množico točk `V` in povezav `E`
  - velikost množice `V` (število točk) označimo z `n`
  - velikost množice `E` (število povezav) označimo z `m`

- Povezavam lahko dodamo ”uteži” → dobimo utežen graf
  - utežem rečemo tudi “cena”, “vrednost”
  - uteži predstavimo s preslikavo `w: E → R` (`w` vsaki povezavi dodeli utež)
  - z oznako `G(V,E, w)` opišemo graf z množico točk `V`, povezav `E` in utežmi `w`

- Večinoma bomo govorili o neusmerjenih grafih; če bomo govorili o usmerjenih, bomo to posebej poudarili!

Implementacija grafa

- V programih graf implementiramo z
  - z matriko
    - pri neusmerjenih grafih je simetrična
    - **Java**: dvodimenzionalna matrika (`boolean[][]` ali `double[][]`)
  - s seznamom sosed
    - **Java**: tabela tabel ali slovar seznamov

Kakovost implementacij?

- matrika zasede več prostora
- operacija "Ali je a povezan z b" je pri matriki hitrejša
- operacija: "naštej vse sosede dane točke" je pri matriki počasnejša

Podobno kot v drevesu lahko tudi v grafu izvajamo pregled v širino (BFS) in globino (DFS)

- če želimo sistematičen pregled, se moramo odločiti za nek vrstni red sosed

- ker lahko graf vsebuje cikle, je pri pregledu treba voditi evidenco o tem, katero vozlišče smo že obiskali

- **Prerez grafa** `G(V,E)` je razdelitev množice točk `V` na dve disjunktni podmnožici `S` in `T`.
  - povezave prereza so povezave med množicama S in T

Minimalno vpeto drevo

Minimalno vpeto drevo

Imamo neusmerjen uteženo graf `G(V,E,w)`.

- Vpeto drevo `G'(V', E', w)` je tak podgraf, da velja: `V'=V`, $E' \subset E$, `G'` je drevo.

- Minimalno vpeto drevo *(angl. minimum spanning tree, MST)*
  je po vsoti povezav najmanjše vpeto drevo.

Problem: V danem neusmerjenem uteženem grafu `G(V,E,w)` poišči minimalno vpeto drevo.

Primer uporabe: izgradnja ali posodobitev cestnega (telekomunikacijskega, električnega) omrežja.

Lastnosti minimalnega vpetega drevesa

- Rešitev problema ni nujno enolična.

- Rešitev je enolična v primeru različnih uteži.

- Minimalna povezava poljubnega prereza grafa `G` je del MST

Primov algoritem

- Robert C. Prim (1957)
- **Požrešen** algoritem: po vrsti dodaja ”lokalno najboljšo” točko in dobi globalno najboljše vpeto drevo.
- Delovanje algoritma:
  - drevo gradi postopoma tako, da začne s poljubno točko, nato drevo na vsakem koraku razširi
  - na vsakemu koraku izbira med povezavami, ki trenutno drevo povezujejo s preostankom grafa; med vsemi temi povezavami izbere najcenejše in jo doda drevesu
  - algoritem se izvaja, dokler drevo ne vsebuje vseh točk; opravi torej n-1 korakov

Algoritem uporablja lastnost minimalne povezave poljubnega prereza.

Primov algoritem

Psevdokoda Primovega algoritma

- ***na začetku*** *izberem poljubno r∈V in nastavim:*

  `T={}, Q=V,` *`key[v]`*`=∞ ∞ za vsak v iz V, key[r]= 0`

- **nato ponavljam**
  - `izberem točko` **`v`** `z najmanjšim` **`key[v]`**`,`
  - `točko` **`v`** `pobrišem iz` **`Q`** `in dodam v` **`T`**`,`
  - `popravim` **`key[u]`** `vsem točkam, ki so povezane z` **`v`**

  `dokler` `niso v T vse točke.`

Časovna zahtevnost?

- algoritem opravi n korakov, na vsakem doda eno vozlišče
- koliko dela opravi na posameznem koraku?
  - poišče najcenejšo točko (točko v z najmanjšim `key[v]`)
  - popravi `key[u]` vsem vseh `u`, ki so povezane z `v`

Zgornje operacije so močno odvisne od podatkovne strukture, v kateri hranimo graf, in strukture za množico `Q`.

Kruskalov algoritem

- **Požrešen** algoritem:, pregleduje povezave **po velikosti** (glede na njihovo ceno); posamezno povezavo doda v drevo, če s povezavami, ki so že v drevesu, **ne tvori cikla**.

Natančneje:

- začnemo tako, da graf razbijemo na **gozd dreves**; v vsakem drevesu je **po ena točka** grafa
- povezave grafa G pregledujem **po vrsti** (od najcenejše proti najdražji); če povezava povezuje dve drevesi, drevesi v gozdu združim v eno drevo;
- postopek **ponavljam** toliko časa, da **pregledam vse povezave** ali da v gozdu ostane eno samo drevo.

- Katero podatkovno strukturo bomo uporabili za shranjevanje vseh povezav?

- Katero strukturo pa bomo uporabili za shranjevanje gozda?

Kruskalov algoritem

Psevdokoda

Časovna zahtevnost?

Barvanje grafov

Barvanje grafov

- **Barvanje grafa** G(V,E) je preslikava, ki vsakemu vozlišču $v \in V$ dodeli barvo
- **Barvanje je pravilno**, če dve sosednji točki grafa nista iste barve.

- **`k`-barvanje:** pravilno barvanje s `k` barvami
- **Kromatično število grafa `G`**: najmanjše možno število barv pravilnega barvanja grafa `G`

- **Odločitveni problem**: ali obstaja k-barvanje grafa?
  - NP-poln problem
- **Optimizacijska varianta problema**: poišči pravilno barvanje z najmanjšim možnim številom barv.
  - NP-težak problem.
- **Uporaba**: Izvajanje opravil, razpored registrov, ...

Predstavili bomo dva pristopa: požrešni algoritem in sestopanje

Barvanje grafov s požrešnim algoritmom

Ideja algoritma:

- barve oštevilčimo (da se nanje lahko sklicujemo z indeksi 1, 2, 3,...)
- točke obiskujemo po nekem vrstnem redu;
  - točki, ki jo obiščemo, dodelimo "najmanjšo" barvo,
    ki je ne uporablja noben od njenih že pobarvanih. sosedov

Psevdokoda:

Barvanje grafov s požrešnim algoritmom

Časovna zahtevnost:

Pomembno je, kakšen **vrstni red** izberemo:

- ✧ lahko ga določimo pred izvajanjem;
- ✧ lahko ga določamo med izvajanjem, na podlagi trenutnega stanja.
- ✧ Gotovo obstaja ”optimalni” vrstni red!
- ✧ Obstajajo tudi zelo slabi vrstni redi;

Barvanje grafov s sestopanjem

- S principom **sestopanja** ugotavljam, ali se da graf pobarvati s `k` barvami.

- Sestopanje implementiram v metodi boolean `lahkoPobarvamZBarvami(G, k)`
  - metoda vrne `true`, če graf lahko pobarvam s `k` barvami in `false` sicer;

- kromatično število grafa s pomočjo metode `lahkoPobarvamZBarvami()` poiščem tako, da metodo kličem po vrsti s k=1, 2, 3, … , dokler prvič ne odgovori s `true`

Ideja delovanja metode `lahkoPobarvamZBarvami()`:

- najprej vse točke razbarvam (pobarvam z barvo 0);
- nato točke obiskujem po vrsti
  - obiskani točki povečujem barvo, dokler

  a) ne najdem pravilne barve (take, ki je ne uporablja nobena soseda)
  (v tem primeru nadaljujem z naslednjo točko)

  b) ne ugotovim, da se točke ne da pobarvati.
  (v tem primeru sestopim → točko razbarvam in grem nazaj na prejšnjo točko)

Primer barvanja z `lahkoPobarvamZBarvami()` metodo grafa pri `k=2` in `k=3`.

Barvanje grafov s sestopanjem

Psevdokoda:

Časovna zahtevnost:

Najcenejše poti v grafih

Najcenejše poti v grafih

- Imamo zemljevid s cestami in njihovimi dolžinami. Poiskati želimo (po vsoti povezav) najcenejšo pot med dvema izbranima mestoma.

  Pozor: To ni isti problem kot trgovski potnik - TP išče najcenejšo pot, ki obišče VSA mesta, kar je bistveno težji problem. Problem najcenejših poti je polinomske narave!

- Možen pristop k reševanju problema najcenejših poti: sistematično pregledujemo vse možne poti in hranimo le najboljšo.
  - vseh možnosti je zelo veliko
  - ukvarjamo se tudi s povsem ne obetavnimi možnostmi

<u>Uporabili bomo jezik teorije grafov in naslednje pojme:</u>

- Utežen graf

- Pot med točkama u in v

- Dolžina poti, cena poti

Najcenejše poti v grafih

- Množica vseh poti
- Najcenejša poti
- Cena najcenejše poti

Problem najcenejših poti v praksi nastopa v štirih različicah

- Najcenejša pot med izbranim izvorom in vsemi ostalimi vozlišči.
- Najcenejša pot med vsemi vozlišči in izbranim ciljem.
- Najcenejša pot med izbranim parom u in v.
- Najcenejša pot med vsemi pari vozlišč.

Lastnosti najcenejših poti

- Iskali bomo najcenejšo pot med vozliščem `1` in vsemi ostalimi vozlišči. Ceno najcenejše poti med vozliščem `1` in vozliščem `i` bomo na kratko označili tudi z $d(1,i)=d_i$.
- Če želimo računati najcenejše poti, potem graf ne sme vsebovati negativnih ciklov.
- Graf lahko vsebuje negativne povezave.
- Najcenejša pot nikoli ne vsebuje cikla.
- Vsaka podpot najcenejše poti je najcenejša.

Bellmanove enačbe

Najcenejšo pot d_i izračunam tako, da izračunam najcenejše poti do vseh točk `k`, iz katerih obstaja direktna povezava do točke `i`, in ceni teh poti prištejem ceno zadnje povezave $c(k,i)$.

Najcenejše poti s topološkim urejanjem

Ideja: če iz `v` ne obstaja pot do `u`, potem najcenejša pot med `1` in `u` ni odvisna od (ne poteka skozi) `v`. Zato lahko d_u izračunam pred d_v.

V tem primeru urejenost $\prec$ določa topološka ureditev (`u` "je pred" `v` tedaj in le tedaj, ko je `u` topološko pred `v`, to je takrat, ko iz `u` vodi pot do `v` (in hkrati takrat, ko iz `v` ne vodi pot do `u`) oziroma:

$$u \prec v \iff u \xrightarrow{*} v.$$

Postopek:

- Najcenejše poti računam v topološkem vrstnem redu: če je vozlišče `i` topološko pred vozliščem `j`, potem d_i izračunam pred d_j.
- Na začetku za vsak i nastavim $d_i = \infty$, začetnemu vozlišči pa $d_1 = 0$.
- Sprehodim se po vozliščih v topološkem vrstnem redu; ko pidem v vozlišče `i`, je d_i že izračunana, popravim le d_j za vse sosede, v katere lahko pridem iz `i`:

$$d_j = \min\{d_j, d_i + c_{ij}\}$$

Uporaba topološkega urejanja - algoritem, časovna zahtevnost in povzetek

Najcenejše poti s topološkim urejanjem - primer

Najcenejše poti z Dijkstrovim algoritmom

Ideja: Ob predpostavki, da cene povezav niso negativne ($c_{ij} >= 0$), velja naslednje:

če je $d_i < d_j$ potem lahko d_i izračunamo pred d_j.

- V tem primeru velja: $i \prec j \iff d_i < d_j$
- Količine d_i računamo po naraščujočih vrednostih, saj zaradi nenegativnih cen večja d_i ne more nastopati kot gradnik v manjši. Najprej izračunamo najmanjši d_i, nato drugega najmanjšega, ...

Postopek:

- Na začetku postavimo $di=\infty$ in $d1=0$ in vsa vozlišča označimo kot neobiskana.
- Nato na vsakem koraku med vsemi neobiskanimi vozlišči izberemo tisto vozlišče i, ki ima med vsemi neobiskanimi vozlišči najmanjši d_i; to vozlišče označimo kot obiskano, vsem vozliščem j, v katera lahko pridemo iz i v enem koraku, pa popravimo d_j:

$$d_j = \min\{d_j, d_i + c_{ij}\}$$

Dijkstra - algoritem, časovna zahtevnost in povzetek

Najcenejše poti z Dijkstro - primer

Najcenejše poti z Bellman-Fordovim algoritmom

Oznaka: d^h_i=najcenejša pot med 1 in i, ki vsebuje največ h korakov.

Očitno velja $d^1_i = c^1_i$ in $d_i^{n-1} = d_i$.

Ideja za algoritem: Rešitev iščemo v več korakih: najprej za vsako vozlišče i $(i=1, 2, \ldots, n)$ poiščemo najcenejšo pot dolgo največ 1 korak (d^1_i), potem najcenejšo pot dolgo največ 2 koraka (d^2_i), ..., na koncu pa najcenejšo pot dolgo največ $n-1$ korakov (d_i^{n-1}).

Formula za izračun d^h_i:

Bellman-Ford- algoritem, časovna zahtevnost, povzetek in primer

Vsi pari poti – posplošeni Bellman-Fordovov algoritem

Želja: Izračunaj vse pare najkrajših poti med vozlišči grafa.

Ideja: Postopoma računam cene najcenejših poti dolžine kvečjemu `h, (h=1, 2, ..., n-1).`

Oznake:

Formula:

Posplošeni Bellman-Ford- algoritem, časovna zahtevnost in primer

Vsi pari poti – Floyd-Warshallov algoritem

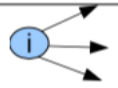
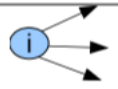
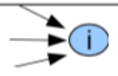
Želja: Izračunaj vse pare najkrajših poti med vozlišči grafa.

- **Ideja**: Postopoma računam cene najcenejših poti, če lahko uporabim samo prvih k vozlišč

Oznake:

Formula:

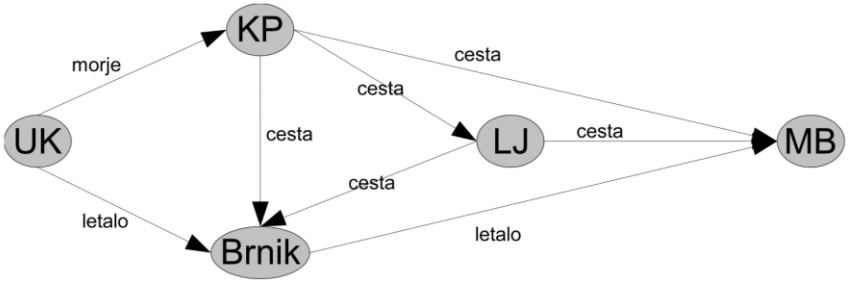
Najcenejše poti - povzetek

Problem	Algoritem	Dodatne zahteve	Tip	časovna zahtevnost
Iz enega izvora v vse točke	Topološko urejanje	nenegativni cikli		$\Theta(n+m) = O(n^2)$
Iz enega izvora v vse točke	Dijkstra	nenegativne povezave		$\Theta(n^2+m) = O(n^2)$
Iz enega izvora v vse točke	Bellman-Ford	/		$\Theta(nm) = O(n^3)$
vsi pari točk	Posplošeni Bellman-Ford	/		$\Theta(n^{2.81} log_2 n)$
vsi pari točk	Floyd-Warshall	/		$\Theta(n^3)$

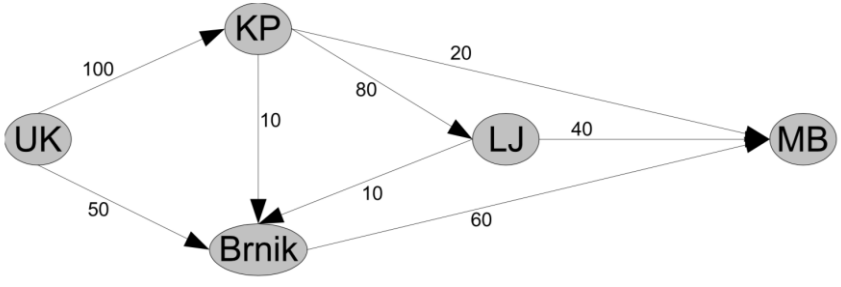
Največji pretok v omrežju

Največji pretok v omrežju

Motivacijski primer: prehrambeno podjetje s sedežem v Veliki Britaniji ima poslovalnico v Mariboru. Prevoz surovin za izdelavo prodajnih artiklov poteka po morju, zraku in po cestah.

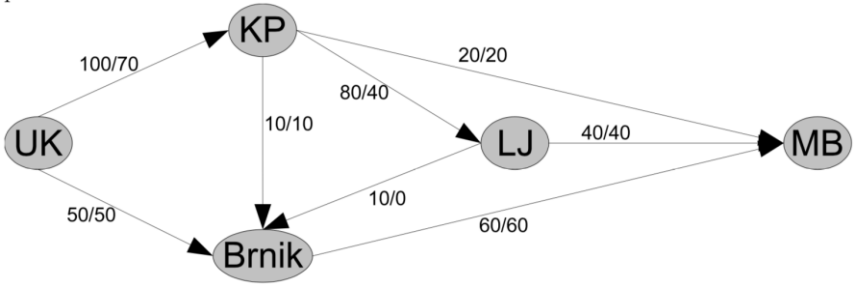


Omejitve povezav: količina tovora, ki se po povezavi lahko prepelje v danem časovnem obdobju.



Hitra analiza stanja:

- ✧ iz UK lahko po obeh poteh skupaj prepeljemo do 150 ton blaga
- ✧ v MB lahko po vseh treh potek pripeljemo do 120 tno blaga
- ✧ skupen pretok bo gotovo manjši ali enak min(150,120).



Če graf pogledamo od daleč lahko z malo truda sestavimo optimalno rešitev tega problema.

Uporabljeni pojmi in oznake:

- **Omrežje** je usmerjen graf G=<V,E> z `n` vozlišči.
- **Oznake vozlišč** - vozlišča grafa označimo s številkami od `1` do `n`.
- **Oznake povezav** - ovezavo od vozlišča `i` do vozlišča `j` označimo z `<i,j>`.
- **Kapaciteta povezave** - vsaki povezavi `<i,j>` grafa `G` je prirejena kapaciteta c_{ij} (koliko blaga lahko prepeljemo po tej povezavi).
- **Izvor/ponor** - vozlišče `1` imenujemo izvor, vozlišče `n` pa ponor; lastnost: vstopna stopnja izvora in izstopna stopnja ponora sta `0` (po domače: v izvor in iz ponora ne gre nobena povezava).
- **Pretok po povezavi** – z x_{ij} označimo pretok po povezavi (koliko blaga po tej povezavi dejansko prepeljemo).

- Lastnosti pretoka x_{ij}
  - po povezavi ne moremo prepeljati več, kot je kapaciteta te povezave
  - vse, kar v neko vmesno vozlišče `i` pride, mora to vozlišče tudi zapustiti
  - v izvor nič ne pride in iz ponora nič ne gre

- **Pretok skozi morežje** - nabor vseh x_{ij} označimo z $x=(x_{ij})$ in imenujemo pretok skozi omrežje .

- **Vrednost pretoka** je enaka vsoti vseh pretokov po povezaveh, ki izhajajo iz točke 1 (oziroma pridejo v točko n). Vrednost pretoka x označimo z $v(x)$.

- **Maksimalen pretok** skozi omrežje je tak pretok x skozi omrežje, da ne večji pretok

- **Pozitivna/negativna povezava** - naj bo P neusmerjena pot od izvora do ponora. Povezavo te poti imenujemo pozitivna povezava, če kaže v smeri poti in negativna povezava, če kaže v nasprotno smer.

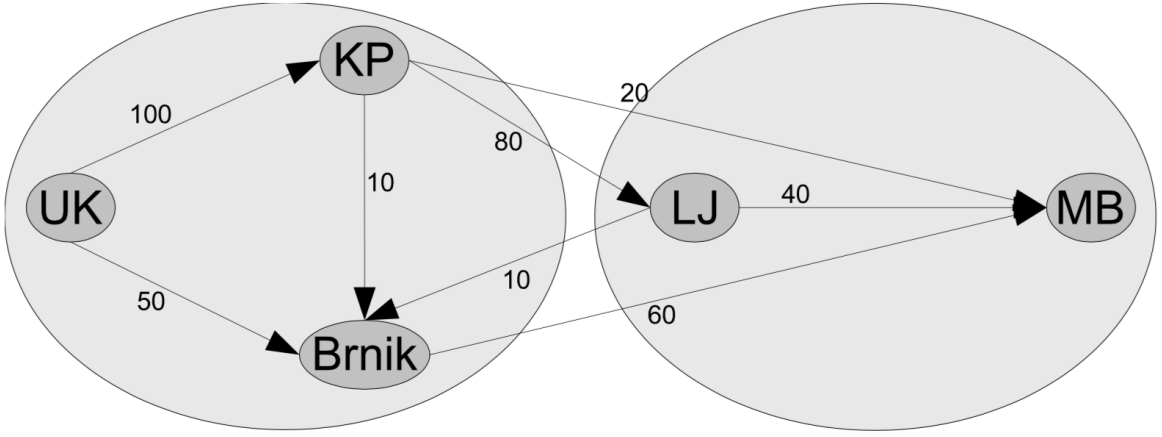
- **Nezasičene povezave** - naj bo P pot od izvora k ponoru in e povezava te poti, ki povezuje vozlišči i in j. Pravimo, da je e nezasičena, če velja:
  - e je pozitivna in $x_{ij}<c_{ij}$ ali
  - e je negativna in $x_{ij}>0$.

- **Zasičene povezave** - vsaka povezava, ki ni nezasičena, je zasičena.

- **Nezasičena pot** - pot v omrežju `G` je nezasičena, če so VSE njene povezave nezasičene. Obratno: pot je zasičena, če je vsaj ena povezava zasičena.

- **Prerez grafa** G je tak par disjunktnih množic S in T, za katere velja: $S \cup T = V$, $S \cap T = \emptyset$, $1 \in S$, $n \in T$. Prerez označimo z `<S,T>`.

- **Kapaciteta prereza** je vsota kapacitet vseh povezav med S in T, ki so usmerjene v smeri proti T. Kapaciteto prereza označimo s `c(S,T)`.



Algoritem za največji pretok v omrežju – teoretične osnove

Izrek (o zgornji vrednosti pretoka): Naj bo $x = (x_{ij})$ poljuben pretok in naj bo $< S, T >$ poljuben prerez grafa G. Potem velja

$$v(x) \leq c(S, T).$$

Posledica. Če je $x = (x_{ij})$ tak pretok, da za nek prerez velja

$$v(x) = c(S, T),$$

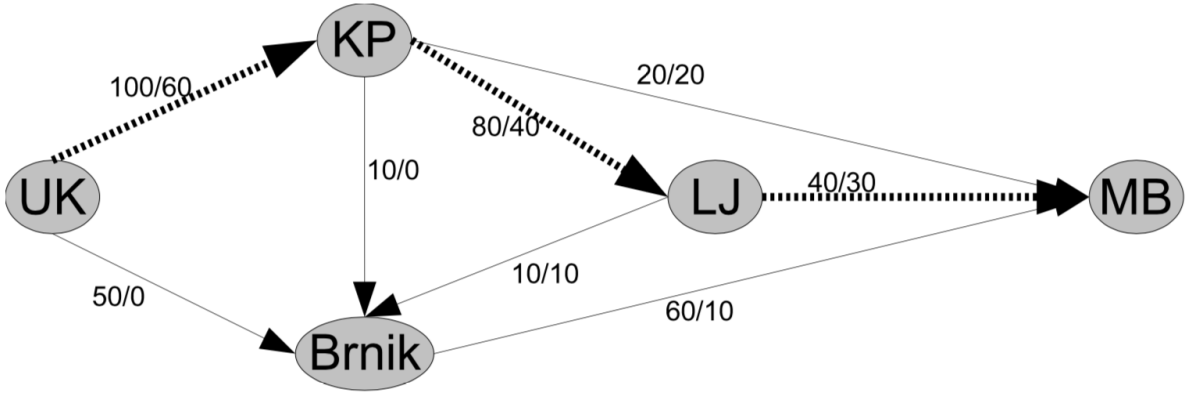
potem je pretok x maksimalen.

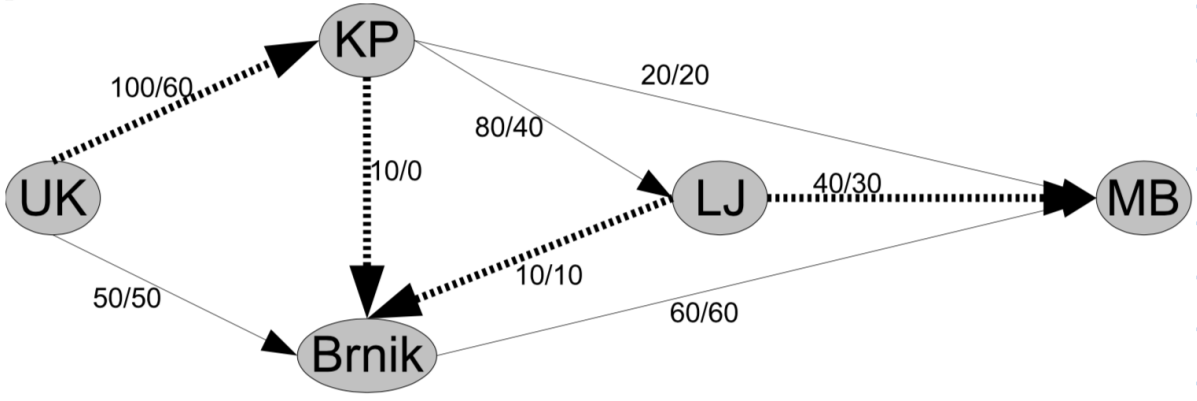
Izrek (o max pretoku in min prerezu): Vrednost maksimalnega pretoka je enaka kapaciteti minimalnega prereza.

Ideja za iskanje zgornje meje vrednosti maksimalnega pretoka: pregledujemo različne prereze in iščemo tistega, ki ima najmanjšo kapaciteto; pretok skozi omrežje gotovo ne bo večji od te vrednosti.

Izrek (o nezasičenih poteh). Nek pretok je maksimalen natanko tedaj, ko se vse poti od izvora do ponora zasičene.

Primer nazasičenih poti





Ford-Fulkersonov algoritem za iskanje največjega pretoka

Potek algoritma:

- ✧ začnemo z nekim dopustnim pretokom (lahko tudi z ničelnim pretokom);
- ✧ trenutni pretok izboljšujemo v več iteracijah - v vsaki iteraciji poiščemo nezasičeno pot in jo zasitimo;
- ✧ če nezasičene poti ni, končamo, trenutni pretok je največjo pretok za to omrežje.

Iskanje nezasičene poti v posamezni iteraciji algoritma:

Vozlišča grafa imajo tri stanja: (1) neoznačeno, (2) označeno, (3) označeno in pregledano.

```
poišči_nezasičeno_pot():
 1:  vozlišču 1 nastavi stanje "označeno", vsem ostalim pa "neoznačeno"
 2:  naj bo x poljubno označeno, a še nepregledano vozlišče
 3:  označi vse neoznačene sosede x, do katerih lahko prideš po nezasičeni povezavi
 4:  če je vozlišče n označeno
 5:     konec - našli smo nezasičeno pot
 6:  sicer:
 7:      stanje vozlišča x nastavimo na "pregledano"
 8:      če obstaja označeno in nepregledano vozlišče
 9:        nadaljejemo pri 2
10:      sicer:
11:        konec - nezasičena pot ne obstaja
```

Prikaz delovanja Ford-Fulkensonovega algoritma na primerih

Dodatek – označevanje povezav in primer delovanja algoritma