Univerza v Ljubljani

Fakulteta za matematiko in fiziko

Matematiika z računalnikom

Randomized algorithms for longest path problem in a graph

Avtor:
Tine Fabiani

Mentorja:
prof. dr. Sergi Cabello Justo
asist. mag. Gašper Domen Romih

Ljubljana, 22. maj 2024

1 Predstavitev problema iskanja najdaljše poti v grafu

Problem iskanja najdaljše poti v grafu je problem v katerem želimo poiskati najdaljšo enostavno pot v grafu, kjer so lahko povezave med vozlišči bodisi utežene bodisi neutežene. Pot v grafu je enostavna natako takrat, ko na naši poti vsako vozlišče obiščemo največ enkrat. V nasprotju z iskanjem najkrajše poti v grafu, kjer lahko za aciklične grafe rešitev najdemo v polinomskem času, je iskanje najdaljše poti v grafu *NP-hard problem*, kar pomeni, da rešitve ne znamo poiskati v polinomskem času. Iskanje najdaljše poti lahko tudi gledamo, kot generalizacijo iskanja Hamiltonove poti, kjer želimo v neusmerjenem grafu poiskati, takšno pot, da gremo skozi vsako vozlišče grafa in pritem vsako vozlišče obiščemo natako enkrat. V primeru, ko je začetno in končno vozlišče isto govorimo o Hamiltonovem ciklu.Iskanje Hamiltonove poti oziroma cikla v grafu je tako ubistvu problem iskanja najdljše poti v neuteženem grafu, tako je tudi Iskanje Hamiltonove poti *NP-hard problem*.

2 Klasični pristopi za iskanje najdaljše poti v grafu

Pri takšnih problemih vedno obstaja opcija, da poiščemo rešitev tako, da preizkusimo vse možnosti, a ker vemo, da je to NP-problem, hitro ugotovimo, da je to mogoča opcija za grafe z majhnim številom vozlišč, brž ko pa začnemo število vozlišč povečevati, pa bi takšna metoda postala hitro računališko prezahtevna.

Skozi zgodovino, so bili razviti različni algoritmi za iskanje najdaljše poti v grafu, seveda pa vsi ti algoritmi, delujejo pri določenih predpostavkah. Za bolše razumevanje problema bom v nadeljevanju naloge na kratko predstavil nekatere pristope in potrebne predpostavek za reševanje tega problema.

- **Prevedemo problem na iskanje najkrajše poti**. Ena od možnih rešitev za iskanje najdaljše poti v grafu **G** je da naredimo vse povezave negativne torej dobimo graf **-G**, nato pa z enim od poznanih algoritmov za iskanje nakrajše poti v grafu, poiščemo najkrajšo pot v grafu **-G**, ki je v originalnem grafu **G** iskana najdaljša pot. Omejitev, ki jo mora imeti graf **G**, je da nima pozitivnih ciklov graf mora biti usmerjen. Tako znamo ubistvu rešitev najti za aciklične usmerjene grafe.

- Najdaljšo pot v grafu znamo poiskati tudi v drevesnih grafih, kjer rešitev poiščemo, s pomočjo BFS-algoritmov (Breadth-first search - algoritem). Za tak problem, iskano rešitev poiščemo v linearnem času.

- Obstajo še nekateri algoritmi, ki poiščejo najdaljšo pot v grafu v polinomskem času za grafe kot so intervalni grafi, grafi krožnega loka in še nekateri drugi grafi.

- Obstajajo tudi algoritmi, ki najdaljšo pot v grafu poiščejo s pomočjo linearnega programiranja.

3 Generator grafov- Nauty geng

Za preizkušanje algoritmov potrebujemo, generirati več grafov. Za namene te seminarske neloge bom uporabil generator grafov Nauty geng.

nauty-geng je programska funkcija znotraj knjižnice nauty, ki se jo uporablja za generiranje grafov določene strukture ali lastnosti. Sam program je orodja uporaben v različnih področjih, kot so teorija grafov, kombinatorika in podatkovne strukture. nauty-geng je posebej zasnovan za generiranje neizomorfnih grafov, kar pomeni, da generira grafe, ki so različni glede na svojo strukturo, vendar ohranjajo določene lastnosti, kot so število vozlišč, povezave ali druga merila. To je koristno za raziskovanje lastnosti grafov in algoritmov, ki delujejo na njih.

Izvorna koda za nauty-geng je napisana v C-ju, a lahko danes nauty-geng uporabljmo tudi v programskih jezikih, kot so paython,MATLAB in še nekateri ostali.

4 Implementacija randomiziranega algoritma za iskanje najdaljše poti v grafu

Tako, kot nam že samo ime pove je ideja randomiziranih algoritmov za iskanje najdaljše poti v grafu v tem, da poskušamo, najdašo pot poiskati s pomočjo večkratnega naključnega poizkušanja. Takšne algoritme lahko uporabljamo tako, da so v celoti randomizirani, lahko pa jih tudi kombiniramo z že znanimi algoritmi za iskanje najdalše poti v grafu.

Za namene seminarske naloge sem se odločil, da bom implementiral algoritem iz predavanj profesorja Jianer Chen (povezava do članka: *https://people.engr.tamu.edu/j-chen3/courses/658/2016/notes/s6.pdf*). Algoritem, ki ga bom implementiral je kombinacija randomiziranega iskanja in že poznanega algoritma.

Najprej ponovimo nekaj definiciji o gragih,ki so pomebni za naš problem.Pot v grafu G je zaporedje $\{w_1, w_2, \ldots, w_k\}$ vozlišč, pri čemer za vsak i, $[w_i, w_{i+1}]$ predstavlja povezavo v G. Pot je preprosta, če se nobeno vozlišče ne ponovi v zaporedju. Če je G usmerjen graf, zahtevamo, da je $[w_i, w_{i+1}]$ usmerjena povezava od w_i do w_{i+1}, kar zapišemo kot $hw_i, w_{i+1}i$. k-pot v G je preprosta pot s k vozlišči. Če je graf G utežen, je maksimalna k-pot v G k-pot v G, katere utež je največja med vsemi k-potmi v G. Z samimi algoritmom se bom osredotočil na iskanje najdaljše poti v uteženem in neusmerjenem grafu oziroma iskanje poti dolžine k, če le ta sploh obstaja, sicer želimo, da nam algoritom, pove da mu take poti ni uspelo dobiti.

Kar že vemo je da je iskanje najdaljše poti v usmerjenem acikličnem grafu, precej enostavno rešljiv problem in ga lahko režimo s sledečim algoritmom, ki temelji na dinamičnem progamiranju.

Algoritem: DagPath

- **Vhod:** usmerjen, acikličen, in utežen graf $\hat{G}$, in celo število k;
- **Izhod:** najdaljša k-pot v grafu $\hat{G}$;

1. topološko razvrstimo vozlišča $\hat{G}$: $v_1, v_2, \ldots, v_n$;
2. **for** $i = 1$ **to** $n - 1$ **do**
   $W[i, 1] = 0; X[i, 1] = *$;
   for h $= 2$ **to** k **do** $\{W[i, h] = -\infty; X[i, h] = *\}$;
3. **for** $i = 1$ **to** $n - 1$ **do**
   for vsako povezavo v_i, v_j **do**
   for $h = 1$ **do** $k - 1$ **do**
   if $W[i, h] \neq -\infty$ and $W[i, h] + \text{wt}[v_i, v_j] > W[j, h + 1]$
   then $W[j, h + 1] = W[i, h] + \text{wt}[v_i, v_j]; X[j, h + 1] = i$;
4. Vozlišče v_t z največjo vredostjo $W[t, k]$ je zadnje vozlišče najdaljše k-poti.

Nekaj pojasil glede delovanja algoritma. Vsako vozlišče v_i je povezano z $2k$ vrednostmi $W[i, h]$ in $X[i, h]$, kjer je $1 \leq h \leq k$. $W[i, h]$ je utež oziroma dolćina največje h-poti $P(h, v_i)$, ki se konča pri v_i in smo jo doslej odkrili, medtem ko je $X[i, h]$ vozlišče tik pred v_i na poti $P(h, v_i)$. Na začetku koraka 2 vsako vozlišče v_i vidi le pot dolžine 1, ki se začne in konča pri njem samem, zato moramo nastavi $W[i, 1] = 0$. Ker ima pot dolžine 1 le eno vozlišče, pustimo $X[i, 1] = *$, kar na konec koncev niti ni pomebno. Za $h > 1$ uporabimo $W[i, h] = -\infty$, da označimo, da ni bila še odkrita nobena h-pot, ki se konča pri v_i.

Ker imamo vozlišča topološko razvrščena (torej povezave potekajo le od vozlišč z manjšim indeksom do vozlišč z večjim indeksom), lahko v zanki v 3. koraku,takoj ko dosežemo vozlišče v_i, so s tem bila s tem tudi obiskana oziroma sprocesirana vsa vozlišča na katerikoli of poti, ki se konča pri vozlišču v_i, tako je tudi $W[i, h]$ in $X[i, h]$, za $1 \leq h \leq k$, sprotno beležita informacije za maksimalno h-pot, ki se konča pri v_i, za vse $h \leq k$. Algoritem v koraku 3 nato doda pravilne informacije oziroma rešitve v vozlišču v_i na poznejša vozlišča preko povezav katera gredo iz v_i. Na koncu koraka 3 tako dobimo rešitev o maksimalni h-poti, ki se konča pri vsakem vozlišču, za vse $h \leq k$, vozlišče v_t katere $W[t, k]$ je največji je tako končno vozlišče naše maksimalne k-poti v grafu G. Našo dobljeno maksimalno k-pot $\{w_1, w_2, \ldots, w_k\}$ lahko rekonstruiramo tako, da začnemo z v_t in nato sledimo povezavam, ki jih podaja matrika $X[*, *]$ o prejšnjih vozlišč na poti: $w_k = v_t$, $w_{k-1} = X[t, k] = v_i$, $w_{k-2} = X[i, k-1] = v_j$, $w_{k-3} = X[j, k-2]$, in tako naprej.

Časovna zahtevnost algoritma DagPath je $O(k(n+m))$, kjer sta n in m število vozlišč in število povezav v grafu $\hat{G}$, oziroma.

Sedaj, ko imamo osnovo, ki jo bomo kasneje uporabili, se vrnimo k prvotnemu problemu in sicer k iskanju najdaljše poti na neusmerjenih in uteženih grafih. Na žalost ne poznamo enostavnega načina za pretvorbo neusmerjenega grafa G v neako "enakovreden"usmerjen acikličen graf, tudi ko neusmerjen graf G sam nima ciklov (npr. G je drevo). Tukaj se pokaže možnost, da bi v poštev lahko prišla prav naključnost.

Če predpostavimo, da je množica vozlišč neusmerjenega grafa G enaka $V = \{v_1, v_2, \ldots, v_n\}$. Naj bo $\pi = (v'_1, v'_2, \ldots, v'_n)$ permutacija množice $\{v_1, v_2, \ldots, v_n\}$. Če vsako povezavo $[v'_i, v'_j]$ grafa G, kjer je $i < j$, pretvorimo v usmerjeno povezavo $hv'_i, v'_j i$ od v'_i do v'_j (in ohranimo isto utež povezave), potem postane G neposredno acikličen graf $\hat{G}_\pi$. Poti v $\hat{G}_\pi$ imajo naslednje lastnosti:

- **P1.** Vsaka (usmerjena) pot v $\hat{G}_\pi$ ustreza poti enake dolžine in enake uteži v G;
- **P2.** Če postane maksimalna k-pot v G (usmerjena) pot v $\hat{G}_\pi$, potem vsaka maksimalna k-pot v usmerjenem acikličnem grafu $\hat{G}_\pi$ ustreza maksimalni k-poti v neusmerjenem grafu G.

Predpostavimo sedaj, da je $Pk = (w_1, w_1, \ldots, w_k)$ maksimalna k-pot v neusmerjenem grafu G. Pravimo, da pot Pk sledi vrstnemu redu permutacije $\pi = (v'_1, v'_2, \ldots, v'_n)$ množice vozlišč $\{v_1, v_2, \ldots, v_n\}$, če za vsak i, w_{i+1} pojavi po w_i v zaporedju vozlišč $(v'_1, v'_2, \ldots, v'_n)$ oziroma pravimo, da pot Pk sledi obratnemu vrstnemu redu π, če za vsak i, se w_{i+1} pojavi pred w_i v zaporedju π. Vidimo lahko, da če Pk maksimalna k-pot,ki sledi vrstnemu redu ali obratnemu vrstnemu redu permutacije π, potem po lastnostih P1-P2 maksimalna k-pot v usmerjenem acikličnem grafu $\hat{G}_\pi$ ustreza maksimalni k-poti v neusmerjenem grafu G. Maksimalna k-pot v usmerjenem acikličnem grafu $\hat{G}_\pi$ lahko potem učinkovito konstruiramo s pomočjo prej opisanega algoritma DagPath.

Naš problem se tako pretvori v problem problem iskanja ustrezne permutacije vozlišč v grafu G. Če naključno izberemo permutacijo vozlišč v G se moramo vprašati kakšna je verjetnost, da imamo ustrezno permutacija preko katere lahko potem poiščemo rešitev?

Zamislimo si zaporedje n pozicij. Določimo k pozicij, ki bodo pozicije za vozlišča k-poti Pk in so urejena v vrstnem redu ali pa v obratnem vrstnem redu. Obstaja $(n-k)!$ različnih permutacij za $n-k$ vozlišč, ki niso na poti Pk. Za prvotnih k pozicij obstaja $2(n-k)!$ permutacij vozlišč $\{v_1, v_2, \ldots, v_n\}$, pri katerih pot Pk sledi vrstnemu redu ali obratnemu vrstnemu redu. Ker obstaja $\binom{n}{k}$ načinov, kako izbrati k pozicij od n pozicij, sklepamo, da je skupno število permutacij, pri katerih pot Pk sledi vrstnemu redu ali obratnemu vrstnemu redu, enako

$$\binom{n}{k}(n-k)! = \frac{n!}{k!}.$$

Izmed $n!$ permutacijami vozlišč $\{v_1, v_2, \ldots, v_n\}$, jih je $\frac{n!}{k!}$ dobrih in lahko iz njih konstruiramo maksimalno k-pot neusmerjenega grafa G z algoritmom DagPath. Če torej naključno izberemo permutacijo vozlišč $\{v_1, v_2, \ldots, v_n\}$, potem je z verjetnostjo $\frac{\frac{n!}{k!}}{n!} = \frac{1}{k!}$ izbrana dobra permutacija. To nam da idejo za naslednji naključni algoritem za iskanje najdaljše poti v grafu:

Algoritem: PathPerm

- **Vhod:** neusmerjen graf G in celo število k;
- **Izhod:** najdaljša k-pot v G;
- **Koraki:**

1. **repeat** $t \cdot k!$-krat
   - 1.1 Naključno izberemo permutacijo π iz množice $\{v_1, v_2, \ldots, v_n\}$;
   - 1.2 Konstruirajmo graf $\hat{G}_\pi$;
   - 1.3 Pokličemo algoritem DagPath, da konstruira maksimalno k-pot v $\hat{G}_\pi$;
2. **if** noben klic v koraku 1.3 ne vrne k-poti
   - 2.1 **then** vrni ("ni k-poti");
   - 2.2 **else** vrni k-pot z največjo utežjo med tistimi, ki so bile konstruirane v koraku 1.3.

Opazimo lahko, da če imamo neusmerjeni graf G, ki nima k-poti, potem nobena permutacija π ne more narediti usmerjenega acikličnega grafa $\hat{G}_\pi$, da ima k-pot. Zato v tem primeru algoritem PathPerm vedno vrne korak 2.1 z ustreznim sklepom. Po drugi strani pa, če ima G k-poti, lahko izberemo katerokoli maksimalno k-pot Pk v G. Kot že vemo, naključna permutacija π ima verjetnost $\frac{1}{k!}$, da sledi vrstnemu redu ali obratnemu vrstnemu redu poti Pk. Za takšno permutacijo π bo korak 1.3 algoritma, ki uporablja DagPath na usmerjenem acikličnem grafu $\hat{G}_\pi$, vrnil maksimalno k-pot v $\hat{G}_\pi$, ki po lastnostih P1-P2 ustreza maksimalni k-poti v neusmerjenem grafu G. Verjetnost, da nobena od $t \cdot k!$ naključnih permutacij, izbranih v koraku 1.1, ni dobra, je omejena z

$$\left(1 - \frac{1}{k!}\right)^{t \cdot k!} < \frac{1}{e^t}.$$

Če torejvzamemo $t = 10$, potem algoritem PathPerm reši problem najdaljše poti z verjetnostjo vsaj 0.99999.

Za vsako fiksno konstanto t (npr. $t = 10$), je časovna zahtevnost algoritma enaka *PathPerm* $O(k(n+m)k!)$, kar je še sprejemljivo, kadar je vrednost k majhna.

Za korektnost algoritma PathPerm je treba pokazati, da če G ima k-pot, potem algoritem PathPerm vedno vrne k-pot; in če G nima k-poti, potem algoritem PathPerm vedno vrne "ni k-poti", a bomo ta del zanemarili, saj je bistveno preobsežen.

5 Testiranje algoritma

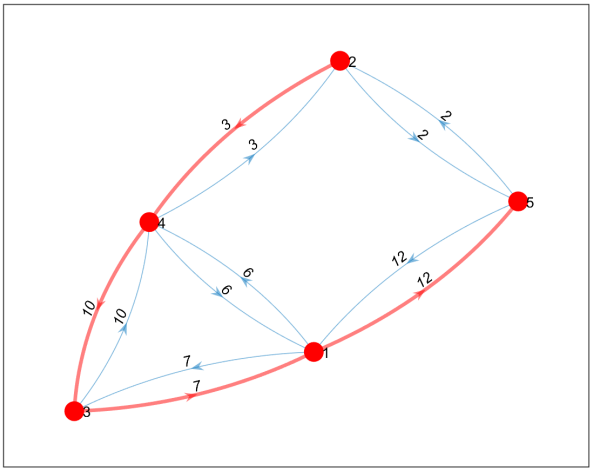
Ko sem testiral sem opazil nekaj ratlik med iskanjem najdaljše poti v uteženih in neuteženih grafih, zato si bomo oba primera pogledi in testirali posebaj

5.1 Uteženi grafi

1. Začimo najšrej z grafom, ki ima 5 vozlišč in naslednjo matriko sosednosti

$$\begin{bmatrix} 0 & 0 & 7 & 6 & 12 \\ 0 & 0 & 0 & 3 & 2 \\ 7 & 0 & 0 & 10 & 0 \\ 6 & 3 & 10 & 0 & 0 \\ 12 & 2 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

Slika grafa in njegova najdalša pot sta prikazana na naslednji sliki

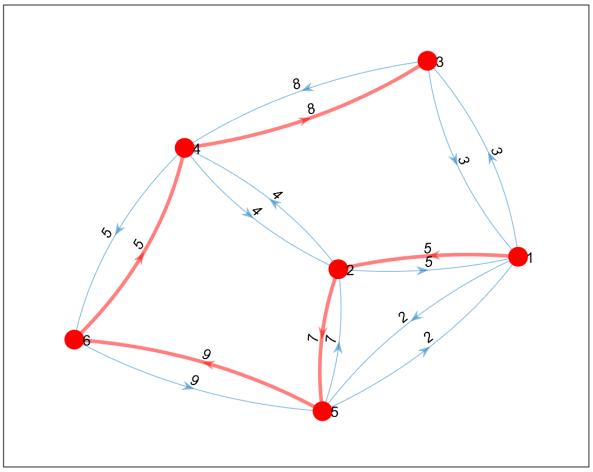


Maximalna dolžina poti je dolžine 32, rešitev pa je bila poiskana v 0,32 sekunde

2. naslednji graf ima 6 vozlišč in naslednjo matriko sosedosti

$$\begin{bmatrix} 0 & 5 & 3 & 0 & 2 & 0 \\ 5 & 0 & 0 & 4 & 7 & 0 \\ 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 4 & 8 & 0 & 0 & 5 \\ 2 & 7 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 5 & 9 & 0 \end{bmatrix} \tag{2}$$

in naslednjo sliko z rešitvijo



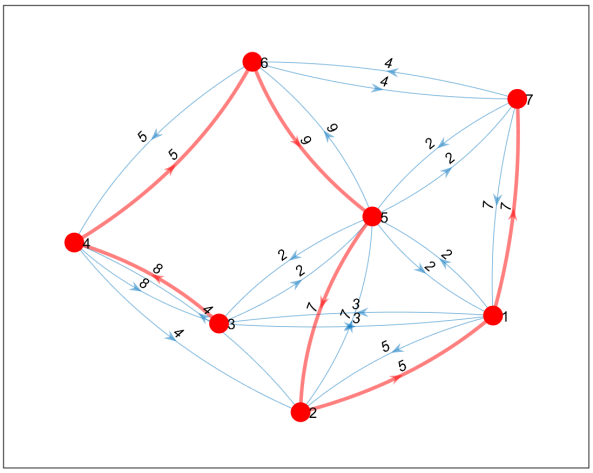
Maximalna dolžina poti je dolžine 34, rešitev pa je bila poiskana v 1,4 sekunde

3. naslednji graf ima 7 vozlišč in naslednjo matriko sosedosti

$$\begin{bmatrix} 0 & 5 & 3 & 0 & 2 & 0 & 7 \\ 5 & 0 & 0 & 4 & 7 & 0 & 0 \\ 3 & 0 & 0 & 8 & 2 & 0 & 0 \\ 0 & 4 & 8 & 0 & 0 & 5 & 0 \\ 2 & 7 & 2 & 0 & 0 & 9 & 2 \\ 0 & 0 & 0 & 5 & 9 & 0 & 4 \\ 7 & 0 & 0 & 0 & 2 & 4 & 0 \end{bmatrix} \tag{3}$$

in naslednjo sliko z rešitvijo

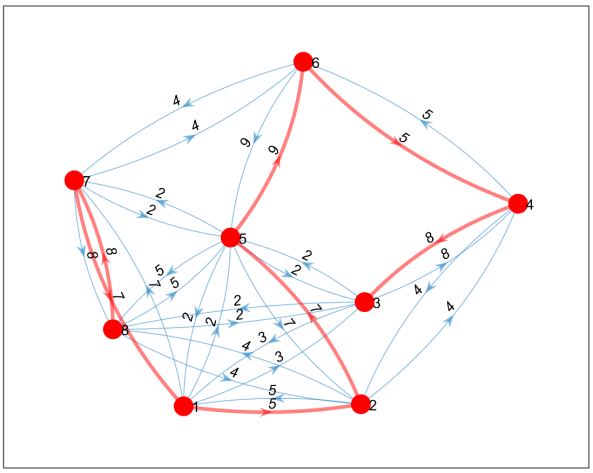
Maximalna dolžina poti je dolžine 41, rešitev pa je bila poiskana v 12,1 sekunde



4. naslednji graf ima 8 vozlišč in naslednjo matriko sosedosti

$$\begin{bmatrix} 0 & 5 & 3 & 0 & 2 & 0 & 7 & 0 \\ 5 & 0 & 0 & 4 & 7 & 0 & 0 & 4 \\ 3 & 0 & 0 & 8 & 2 & 0 & 0 & 2 \\ 0 & 4 & 8 & 0 & 0 & 5 & 0 & 0 \\ 2 & 7 & 2 & 0 & 0 & 9 & 2 & 5 \\ 0 & 0 & 0 & 5 & 9 & 0 & 4 & 0 \\ 7 & 0 & 0 & 0 & 2 & 4 & 0 & 8 \\ 0 & 4 & 2 & 0 & 5 & 0 & 8 & 0 \end{bmatrix} \tag{4}$$

in naslednjo sliko z rešitvijo



Maximalna dolžina poti je dolžine 49, rešitev pa je bila poiskana v 124,3 sekunde, kar je že kar velik skok v prierjavi z prejšnim primerom.

5.2 Neuteženi grafi

Vemo, da je vsak neutežen graf ubistvu utežen graf, kjer ima vsaka povezava utež vrednost 1. Vemo pa tudi, da se problem najdalše poti v neuteženih grafih enostavno prevede na iskanje Hamiltonove poti. Sama časovna zahtevnost iskanja najdalše poti se med uteženimi in neuteženim ubistvu ne razlikuje, saj moremo za enako velik bodisi utežen bodisi neutežen graf opraviti isto đtevilo korakov v algoritmu.

Ima pa iskanje hamiltonove poti eno prednost pred iskanjem najdalše poti v uteženm grafu, ta pa je da za hamiltonovo pot, ki je obenem tudi najdaša pot v neuteženem grafu v naprej vemo, kakšna je dolžina najdalše poti. Vemo, da je v splošnem iskanje hamiltnove poti težak problem in v splošnem pogosto sploh nevemo ali takšna pot sploh obstaja. Z majhno modifikacijo našega algoritma in sicer z dodatnim zaustavitvenim pogojem, ki ustavi algoritem takoj, ko dobi pot, ki imam maksimalno dolžino (kar je mogoče ker poznamo maksimalno dolžino v naprej, če hočemo imeti hamiltonovo pot), se algoritem zaustavi in nam vrne najdalšo pot.

Na ta način lahko morda hitreje dobimo, možno rešitev za hamiltonovo pot in s tem pokažemo obstoj le te tudi v večjih grafih. Seveda pa se je treba zavedati, da je hitrost koliko časa smo porabili za iskanje takšne poti, v primeru, da hamiltova pot obstaja, poplnoma odvisna od naključja, kdaj izberemo pravo permutacijo,ki nam vrne ustrezno rešitev.Te metode pa nemoremo uporabiti za

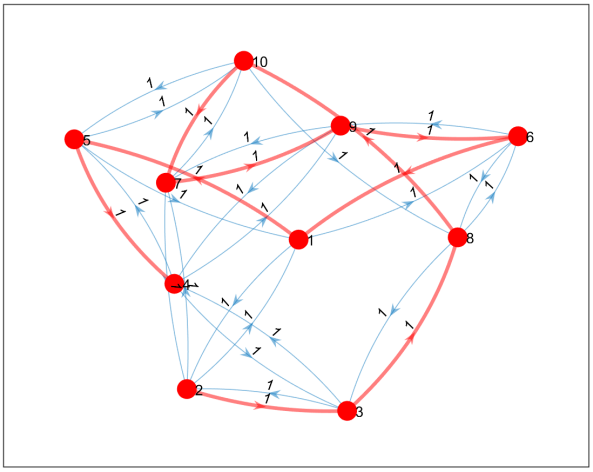
uteženе grafe saj nemoremo v naprej vedeti, koliko je dolžina najdalše poti.

Sedaj si tako poglejmo še nekaj primerov iskanja hamiltonove poti z modifiranim algoritmom.

1. začino z grafom, za katerega vemo, da hamiltonova pot obstaja in sicer Petrsonov grafom, ki ima 10 vozlišč in naslednjo matriko sosednosti.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{5}$$

   in naslednjo sliko z rešitvijo



   v primeru ko sem iskal to rešitev ja bil čas iskanj manj kot 1 sekunda pa čeprav smo iskali po grafu z 10 vozlišči. Seveda pa je čas iskanja v tem primeru čisto odvisen od naključja, kdaj izberemo pravo permutacijo.