Iskanje najkrajše poti

Ana Vojnović, I. gimnazija v Celju
Anja Petković, Gimnazija Bežigrad, Ljubljana
Matej Roškarič, SERŠ, Maribor
Mentor: Uroš Kuzman UL FMF

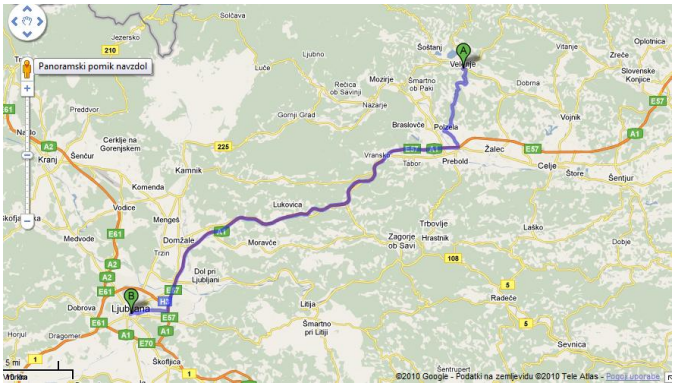
1 Uvod

V vsakdanjem življenju se velikokrat srečamo s težavo, ko potujemo iz enega v drug kraj. Za izbiro poti imamo namreč veliko možnosti, a odločiti se moramo le za eno, najbolj primerno. Pri tem upoštevamo več dejavnikov, od razdalje, časa, cene goriva, pa do udobja, povprečne hitrosti in še mnogih drugih stvari. Pogosto izbiro najboljše možne poti prepustimo kar elektronskim napravam kot je GPS, kljub temu, da nam je njih matematično ozadje nepoznano. Odločili smo se raziskati algoritme, ki stojijo za tem.

Skušajmo prevesti problem iskanja najkrajše poti v matematično obliko. Cestno omrežje lahko učinkovito ponazorimo z grafi (množico točk in daljic). Kraji postanejo vozlišča, ki jih bomo v članku označili z $V(G)$, cestno omrežje pa opišemo z množico povezavav $E(G)$, ki jim dodamo težo glede na dolžino cestne povezave med kraji ali katero izmed drugih za odločitev relevantnih količin. Poljubno povezavo $e \in E(G)$ bomo v nadaljevanju označili tudi z XY, če bo povezovala vozlišči X in Y. Z $d(e)$ oziroma $d(XY)$ bomo označili težo omenjene povezave v omrežju. Na grafu si bomo izbrali dve vozlišči, začetno S in končno C. Naš cilj pa je torej poiskati najkrajšo možno pot med njima. Po nekaj nadebudnih začetnih poizkusih se izkaže, da je problem težji, kot se zdi na začetku. Namesto preverjanja vseh možnih poti bo potrebno izbrati učinkovitejšo in bolj sistematično rešitev – z uporabo algoritmov.

2 Belman-Fordov algoritem

V tem razdelku bomo predstavili enega najosnovnejših algoritmov za iskanje najkrajše poti v grafu, ki temelji na zaporednem pregledovanju vseh povezav v grafu. Za vsako vozlišče $X \in V(G)$ bomo na posameznih korakih algoritma izračunali vrednost $d(X)$, ki bo predstavljala dolžino najkrajše trenutno najdene poti od S do X. Na začetku bomo za vrednosti nastavili $d(S) = 0$ in $d(X) = \infty$ pri vseh



Slika 1: Zemljevid poti iz Velenja do Ljubljane

ostalih vozliščih. Nato bomo vsakič, ko bomo obravnavali neko povezavo XY, preverili, ali lahko trenutni vrednosti $d(X)$ in $d(Y)$ izboljšamo. To ponazorimo s spodnjim algoritmom.

Algoritem:
$d(S) = 0$ in $d(V) = \infty$ za $V \neq S$.
Ponovi $(|V(G)| - 1)$-krat.
{
  $\forall XY \in E(G)$
  {
    Če je $d(X) > d(XY) + d(Y)$, nastavi $d(X) = d(XY) + d(Y)$.
    Če je $d(Y) > d(XY) + d(X)$, nastavi $d(Y) = d(XY) + d(X)$.
  }
}

Začni z $V = C$ in ponavljaj, dokler ne prideš do $V = S$.
{
  Poišči V-jevega soseda W, za katerega velja $d(V) = d(W) + d(VW)$. Nastavi $V = W$.
}

Zaporedje vozlišč, ki jih preteče algoritem v zadnji zanki, predstavlja najkrajšo pot od S do C, zapisano v obratnem vrstnem redu. Če smo pozorni, opazimo, da je potrebno postopek preverjanja vseh povezav ponoviti $(|V(G)| - 1)$-krat. V resnici bi lahko algoritem velikokrat zaključili že prej, v najslabšem primeru pa se lahko zgodi, da ob vsakem preverjanju povezav ponastavimo le eno vrednost $d(X)$

na najkrajši poti od začetka do cilja, le ta pa preteče vsa vozlišča v grafu (torej je dolžine $|V(G)| - 1$). Torej je število operacij, potrebnih za izvedbo algoritma, sorazmerno z produktom števila povezav in števila vozlišč. Opazimo pa lahko, da nam algoritem ob enem poišče tudi vse ostale najkrajše poti od S do vseh ostalih oglišč v grafu, kar je lahko v nekaterih primerih tudi ugodnost.

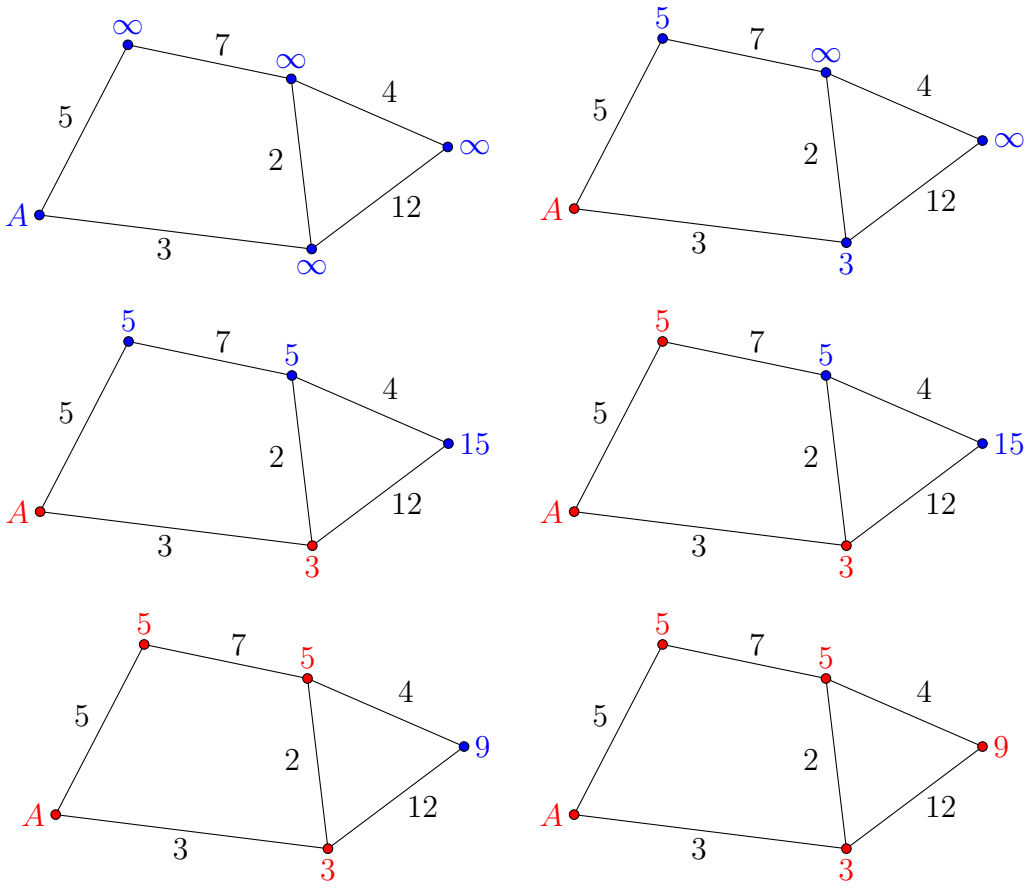
3 Dijkstrov algoritem

V nadaljevanju bomo predstavili algoritem za iskanje najkrajše poti, ki bo časovno učinkovitejši od *Belman-Fordovega*. Kot prejšnji bo tudi ta algoritem poiskal najkrajše razdalje od začetnega do vseh ostalih povezav v grafu. Osnovna operacija *Dijkstrovega algoritma* je sprostitev vozlišča, kar pomeni dokončen izračun vrednosti $d(X)$, ki kot prej ponazarja dolžino trenutne najkrajše poti od S do X. Sprostitev vozlišč začnemo pri vozlišču S, trenutno vrednost najkrajše povezave pa nastavimo za vsa njegova sosednja vozlišča. Nato v množico γ, ki predstavlja vsa sproščena vozlišča grafa G, dodamo tisto vozlišče X, čigar vrednost $d(X)$ je najmanjša med vsemi nesproščenimi vozlišči. Ob tem popravimo tudi trenutne najkrajše poti za vse njegove nesproščene sosede. Algoritem se zaključi, ko so v množici γ vsa vozlišča.

Algoritem:
$d(S) = 0$ in $d(X) = \infty$ za $X \neq S$.
Za vse sosede X vozlišča S nastavi $d(X) = d(SX)$.

Poišči vozlišče $V \in V(G) - \gamma$ z najmanjšo vrednostjo $d(V)$.
{
    Za vse sosede X vozlišča V, ki niso v γ, naredi:
    {
        Če je $d(X) > d(V) + d(XV)$ nastavi $d(X) = d(V) + d(XV)$.
        Dodaj vozlišče V v množico γ.
    }
}

Začni z $V = C$ in ponavljaj, dokler ne prideš do $V = S$.
{
    Poišči V-jevega soseda W, za katerega velja $d(V) = d(W) + d(VW)$. Nastavi $V = W$.
}



Slika 2: Primer Dijkstrovega algoritma

Opazimo, da je zadnja zanka algoritma enaka tisti v *Belman-Fordovem algoritmu*. Torej tudi tokrat konstruiramo zaporedje najkrajše poti v obratnem vrstnem redu. Vendar pa se tokrat zdi pravilnost predstavljenega algoritma nekoliko težje razumljiva, zato bomo dodali tudi dokaz.

Dokaz. Trdimo, da je v trenutku, ko sprostimo vozlišče V, vrednost $d(V)$ enaka dolžini najkrajše poti od S do V. Denimo, da to ni res in obstaja krajša pot od S do V, ki pa bo morala uporabiti vsaj vozlišče X, ki ni v γ. Tedaj obstajajo vozlišča $V_1, V_2, \ldots, V_m, m \geq 0$, da je

$$d(V) > d(X) + d(X, V_1) + d(V_1, V_2) + \ldots + d(V_m, V).$$

Vendar pa od tod sledi, da je $d(V) > d(X)$. Tako pridemo do protislovja, saj smo predpostavili, da želimo sprostiti vozlišče V. To pomeni, da ima V med vsemi vozlišči, ki niso v γ, najnižjo vrednosti $d(V)$. □

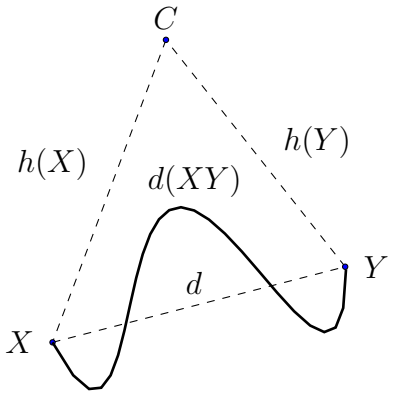
4 Algoritem A^*

V prejšnjih dveh algoritih smo ob najkrajši poti med vozlišči S in C poiskali tudi vse ostale najkrajše povezave do vozlišča S v grafu. Ta podatek je pogosto odveč. V primeru, da iščemo pot v nek določen kraj, bi želeli, da bo le-ta v algoritmu obravnavan čim prej in lahko nato algoritem zaustavimo. Za to uporabimo postopek, podoben *Dijkstrovem algoritmu*, vendar pri določanju naslednjega obravnavanega vozlišča odločamo s pomočjo *hevrističnih ocen*. Le-te predstavljajo spodnjo mejo za najkrajšo pot od trenutnega vozlišča X do cilja C. Vsota trenutne najkrajše poti $d(X)$ in hevristične ocene $h(X)$ nam tako poda oceno, koliko bi lahko znašala najkrajša pot od S do C, ki teče preko vozlišča X. V algoritmu torej sprostimo vozlišče z najnižjo vsoto.

Če pozorno spremljamo algoritem, opazimo, da zaradi spremenjenega ključa določanja vrstnega reda v primerjavi z *Dijkstrovim algoritmom* ne moremo zagotoviti, da bodo imela sproščena vozlišča že določeno najkrajšo možno pot do vozlišča S. Tako se moramo bodisi odločiti, da bomo vozlišča pregledovali večkrat ali pa se sprijaznimo, da bomo na učinkovit način našli pot, ki morebiti ne bo opitmalna. Kljub vsemu lahko v nekaterih posebnih primerih zagotovimo tudi optimalnost najdene poti. To je res, ko so hevristične ocene vozlišč *konsistentne*. To pomeni, da mora veljati

$$h(X) \leq d(XY) + h(Y)$$

za vsa sosednja si vozlišča X, Y. Dokaz te lastnosti prepuščamo bralcu, ki si lahko pomaga s tistim v primeru *Dijkstrinega algoritma*.



$d =$ zračna razdalja med X in Y

$h(X) \leq h(Y) + d \leq h(Y) + d(XY)$

Slika 3: Ponazoritev konsistentnosti hevristične ocene z uporabo zračne razdalje

Če se sedaj vrnemo k začetnemu problemu iskanja najkrajše poti med dvema krajema v omrežju cestnih povezav, je algoritem A^* prava izbira. Za hevristične ocene vozlišč lahko izberemo zračne razdalje med kraji, ki so zagotovo manjše od

dolžin cestnih povezav med njimi. Hkrati pa so, zaradi trikotniške neenakosti, te hevristične ocene tudi konsistentne.

Literatura

[1] http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[2] http://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

[3] http://en.wikipedia.org/wiki/A*_search_algorithm

[4] http://maps.google.com/