REŠEVANJE PROBLEMOV S HEVRISTIČNIM PREISKOVANJEM

LJUPČO TODOROVSKI

Povzetek. V nadaljevanju podajam gradivo za predavanje na temo hevrističnega preiskovanja. Cilj predavanja je spoznati preiskovalne algoritme, ki so splošni pristop k reševanju problemov. Za reševanje problema s preiskovalnim algoritmom moramo najprej definirati prostor možnih stanj oziroma prostor rešitev. Ko imamo prostor možnih stanj, ga lahko preiščemo z različnimi preiskovalnimi algoritmi, ki jih lahko razvrstimo v skupini neinformiranih in informiranih algoritmov. V slednjo skupino sodi algoritem hevrističnega preiskovanja A^*, ki uporablja hevristične funkcije za izboljšanje učinkovitosti preiskovanja. Gradivo ilustrira predstavljene koncepte na zgledih, kakšni izmed zgledov so pripravljeni v obliki videa, dostopnega v spletni učilnici. Med predavanjem bomo reševali vaje iz tega gradiva. Zadnje poglavje teh gradiv podaja nabor nalog za nadaljnje, samostojno delo študentov.

1. Reševanje problemov s preiskovanjem

Preiskovalni algoritmi ponujajo splošni pristop k reševanju problemov. Ideja tega pristopa je, da do rešitve problema pridemo s splošnim algoritmom preiskovanja prostora možnih (delnih) rešitev. Torej zato, da problem lahko rešujemo s preiskovalnim algoritmom, moramo najprej definirati prostor možnih rešitev. Podajmo formalno definicijo, nato pa si oglejmo zgled in rešimo nekaj vaj, preden spoznamo preiskovalne algoritme.

Definicija 1.1. *Prostor stanj* $\mathcal{S}$ je četverica $\langle S, P, s_0, S_k\rangle$. S je števna *množica stanj*, ki jih imenujemo tudi delne ali možne rešitve problema. *Funkcija prehodov* med stanji $P : S \to 2^{S\times\mathbb{R}}$ za podano stanje s določi množico dvojic $\langle t, c\rangle$, kjer t označuje naslednje stanje prehoda, c pa njegovo ceno. Stanje $s_0 \in S$ je *začetno stanje*. $S_k \subseteq S$ je *množica končnih stanj*, ki ustrezajo rešitvi problema.

Prostor stanj si lahko poenostavljeno predstavljamo kot usmerjen graf, ker vozlišča ustrezajo stanjem iz S, povezave pa določa funkcija prehodov med stanji P tako, da sta s in t povezani, če $\exists c \in \mathbb{R} : \langle t, c\rangle \in P(s)$. Ceno c priredimo povezavi med vozlišči s in t.

Na začetku reševanja problema smo v začetnem stanju s_0. Rešitev problema je pot v grafu, ki povezuje vozlišče s_0 z enem izmed končnih vozlišč $s_k \in k$. Če seštejemo cene povezav na poti iz s_0 v s_k dobimo ceno poti oziroma rešitve problema. Rešitev je optimalna, če ne obstaja rešitev, t.j. pot iz s_0 v enega izmed končnih stanj $s_k \in S_k$ z nižjo ceno. Podajmo zdaj formalno definicijo.

Definicija 1.2. *Rešitev* r v prostoru stanj $\mathcal{S} = \langle S, P, s_0, S_k\rangle$ je zaporedje stanj $r = s_0, s_1, \ldots, s_n \in S_k$, kjer za vse dvojice zaporednih členov s_{i-1} in s_i velja $\langle s_i, c_i\rangle \in P(s_{i-1})$. *Cena rešitve* je enaka vsoti prehodov v zaporedju stanj, $\text{cena}(r) = \sum_{i=1}^{n} c_i$.

Optimalna rešitev v prostoru stanj je tista rešitev r, ki ima najmanjšo možno ceno, t.j., ne obstaja rešitev q s ceno $\text{cena}(q) < \text{cena}(r)$.

Zaželena lastnost funkcije prehodov prostora stanj je, da lahko iz začetnega stanja s_0 lahko z zaporedjem prehodov med stanji pridemo v katerokoli stanje $s \in S$. V poenostavljeni predstavitvi prostora stanj z grafom to pomeni, da v grafu obstaja pot iz s_0 do kateregakoli vozlišča s. Lastnost dosegljivosti je pomembna, saj nam zagotavlja, da če začnemo v začetnem stanju, lahko z večkratno uporabo funkcije prehodov, lahko pridemo v katerokoli stanje prostora.

Definicija 1.3. Funkcija prehodov zagotavlja *dosegljivost* v prostoru stanj, če za vsako stanje $s \in S$ obstaja zaporedje stanj $s_0, s_1, \ldots, s_n = s$, ker za vse dvojice zaporednih členov s_{i-1} in s_i velja $\langle s_i, c_i \rangle \in P(s_{i-1})$.

Zgled 1.4. *Linearna diofantska enačba* dveh spremenljivk x in y je enačba oblike $a_x x + a_y y = a$, kjer so a_x, a_y in a celoštevilčni konstantni parametri, spremenljivki x in y pa zavzameta celoštevilčne vrednosti. Primer take diofantske enačbe za $a_x = 17$, $a_y = 29$ in $a = 63$ je $17x + 29y = 63$.

Linearne diofantske enačbe lahko rešujemo s preiskovanjem enostavnega prostora stanj, ki ga definiramo takole. Stanja so dvojice $\langle x, y \rangle \in \mathbb{Z} \times \mathbb{Z}$. Začetno stanje je $\langle 0, 0 \rangle$, množico končnih stanj pa definiramo kot $S_k = \{\langle x, y \rangle : a_x x + a_y y = a\}$. Funkcijo prehodov lahko definiramo s predpisom $P(\langle x, y \rangle) = \{\langle \langle x+1, y \rangle, 1 \rangle, \langle \langle x, y+1 \rangle, 1 \rangle, \langle \langle x-1, y \rangle, 1 \rangle, \langle \langle x, y-1 \rangle, 1 \rangle\}$.

Funkcija prehodov določa enake cene vseh prehodov, kar uporabimo vsakič, ko so prehodi enakovredni ali enako zahtevni. V primeru reševanja diofantske enačbe podaja vsako končno stanje rešitev. Torej nas ne zanima pot iz začetnega v končno stanje, saj vsa informacija o rešitvi enačbe (vrednosti x in y) je zapisana v opisu (končnega) stanja. Definicija končnih stanj nam zagotavlja, da le te vedno ustrezajo rešitvam enačbe.

Vaja 1.5. Dokaži, da zgoraj definirana funkcija prehodov v prostoru stanj za reševanje diofantske enačbe zagotavlja dosegljivost.

Vaja 1.6. V prostoru stanj za reševanje linearnih diofantskih enačb izpelji formulo, ki bi vsoto cen prehodov iz začetnega stanja $s_0 = \langle 0, 0 \rangle$ v stanje $s = (x, y)$, izračunala na osnovi vrednosti x in y.

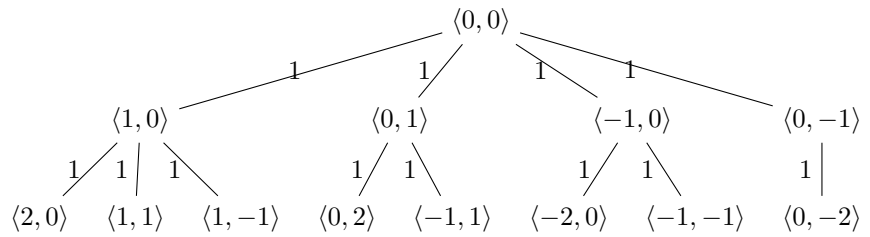
Vaja 1.7. Posploši definicijo prostora stanj za reševanje linearne diofantske enačbe z dvema neznankama, tako da lahko rešujemo enačbe s poljubnim številom neznank.

Na koncu bomo definirali še dve meri kompleksnosti prostora stanj, ki jih bomo pozneje uporabili pri analizi časovne in prostorske zahtevnosti preiskovalnih algoritmov.

Definicija 1.8. *Stopnja razvejanja* b za prostor stanj $\mathcal{S} = \langle S, P, s_0, S_k \rangle$ je maksimalno število možnih prehodov iz kateregakoli stanje $s \in S$: $b(\mathcal{S}) = \max_{s \in S} |P(s)|$. *Globina* d prostora stanj je najmanjše število prehodov potrebnih, da iz stanja s_0 pridemo v eno izmed končnih stanj iz S_k.

Vaja 1.9. Kakšna je vrednost stopnje razvejanja prostora stanj za reševanje linearnih diofantskih enačb? Kakšna je globina d tega prostora stanj? Ali lahko d ocenimo na osnovi konstantnih parametrov diofantske enačbe?

Vaja 1.10. Ali lahko definirani prostor stanj uporabimo za reševanje nelinearnih diofantskih enačb?



SLIKA 1. Primer preiskovalnega drevesa v prostoru stanj za reševanje diofantskih enačb. Pozor: v prikazu drevesa, zaradi preglednosti, manjkajo vozlišča, ki se ponavljajo. Tako manjka otrok $\langle 1, 1\rangle$ vozlišča $\langle 0, 1\rangle$, otrok $\langle -1, 1\rangle$ vozlišča $\langle -1, 0\rangle$ ter otroka $\langle 1, -1\rangle$ in $\langle -1, -1\rangle$ vozlišča $\langle 0, -1\rangle$.

2. Preiskovalni algoritmi

Osnovna podatkovna struktura, ki jo uporabljajo preiskovalni algoritmi je preiskovalno drevo.

Definicija 2.1. *Preiskovalno drevo* v prostoru stanj $\mathcal{S} = \langle S, P, s_0, S_k\rangle$ je drevo v katerem:

- so vozlišča stanja iz S,
- je korensko vozlišče s_0,
- povezave, ki izhajajo iz vozlišča s ustrezajo prehodom iz $P(s)$, torej otroci s so stanja iz množice $\{t : \exists c_t : \langle t, c_t\rangle \in P(s)\}$, cena povezave v drevesu $\langle s, t\rangle$ je enaka ceni tega prehoda c_t,
- vozlišča na veji od korena do posameznega lista ustrezajo različnim stanjem.

Preiskovalna fronta je množica končnih vozlišč (listov) v preiskovalnem drevesu. *Globina* vozlišča s v preiskovalnem drevesu je število povezav na veji od s do korenskega vozlišča s_0. *Funkcija* $g : S \to \mathbb{R}$ za podano vozlišče s v preiskovalnem drevesu vrne vsoto cen vseh prehodov na veji od s do korenskega vozlišča s_0.

Ilustrirajmo definicijo na primeru preiskovalnega drevesa v prostoru stanj iz Primera 1.4.

Zgled 2.2. Slika 1 prikazuje primer preiskovalnega drevesa v prostoru stanj za reševanje linearnih diofantskih enačb. Korensko vozlišče je $s_0 = \langle 0, 0\rangle$, iz korenskega vozlišča izhajajo štiri veje, ki ustrezajo prehodom iz množice $P(s_0) = \{\langle\langle 1, 0\rangle, 1\rangle, \langle\langle 0, 1\rangle, 1\rangle, \langle\langle -1, 0\rangle, 1\rangle, \langle\langle 0, -1\rangle, 1\rangle\}$.

Veje iz vozlišča $\langle 1, 0\rangle$ ustrezajo prehodom iz množice $P(\langle 1, 0\rangle) = \{\langle\langle 2, 0\rangle, 1\rangle, \langle\langle 1, 1\rangle, 1\rangle, \langle\langle 0, 0\rangle, 1\rangle, \langle\langle 1, -1\rangle, 1\rangle\}$. A pozor: iz vozlišča $\langle 1, 0\rangle$ izhajajo zgolj tri veje, ki ustrezajo trem prehodom iz množice $P(\langle 1, 0\rangle)$. Namreč stanje $\langle 0, 0\rangle$, ki ustreza tretjemu elementu te množice je že v drevesu. Kot določa tretja alineja definicije preiskovalnega drevesa, v drevesu nastopi zgolj enkrat in ga ne podvajamo kot naslednika vozlišča $\langle 1, 0\rangle$. Podobno, vozlišči $\langle 0, 1\rangle$ in $\langle -1, 0\rangle$ imata zgolj po dva naslednika ter vozlišče $\langle 0, -1\rangle$ le enega.

Preiskovalna fronta drevesa s Slike 1 je množica osmih končnih vozlišč drevesa, $F = \{\langle 2, 0\rangle, \langle 1, 1\rangle, \langle 1, -1\rangle, \langle 0, 2\rangle, \langle -1, 1\rangle, \langle -2, 0\rangle, \langle -1, 1\rangle, \langle 0, -2\rangle\}$. Globina vozlišča $\langle 0, 0\rangle$ je 0, globina $\langle 0, 2\rangle$ pa 2. Podobno je tudi $g(\langle 0, 0\rangle) = 0$ ter $g(\langle 0, 2\rangle) = 2$.

Vaja 2.3. Določi veje in naslednike vozlišča $\langle 1, 1\rangle$ v preiskovalnem drevesu s Slike 1. Nato naredi isto za vozlišče $\langle 2, 0\rangle$. Kako bi se rezultat spremenil, če bi vozlišči obravnavali v obratnem vrstnem redu? Ali bi se spremenila tudi preiskovalna fronta?

Algoritem 1 Splošen preiskovalni algoritem

```
1:  function PREISCI(S = ⟨S, P, s_0, S_k⟩)
2:      T = preiskovalno drevo z enim vozliščem s_0, g(s_0) = 0
3:      F = {s_0}
4:      while True do
5:          if F = ∅ then
6:              return T, False
7:          end if
8:          s = izberi list iz F, F = F \ {s}
9:          if s ∈ S_k then
10:             označi s v T
11:             return T, True
12:         end if
13:         for all (t, c_t) ∈ P(s) do
14:             if t ∉ veja(s_0, s) then
15:                 dodaj vejo ⟨s, t⟩ z oznako c_t v T, g(t) = g(s) + c_t
16:                 F = F ∪ {t}
17:             end if
18:         end for
19:     end while
20: end function
```

Algoritem 1 predstavi splošen algoritem za preiskovanje podanega prostora stanj $\mathcal{S} = \langle S, P, s_0, S_k\rangle$. Na začetku (vrstici št. 2 in 3) algoritem vzpostavi najbolj enostavno preiskovalno drevo T, ki vsebuje le eno, korensko vozlišče s_0. Ker je to vozlišče hkrati tudi list, vsebuje tudi preiskovalna fronta F en element. Nato algoritem nadaljuje delo v neskončni zanki iteracij. V posamezni iteraciji algoritem obravnava izbrano stanje s iz preiskovalne fronte F (vrstica št. 8), obravnavano vozlišče algoritem takoj odstrani iz preiskovalne fronte.

Zanka se lahko konča, če je s končno stanje (vrstice 9−12) ali pa, če v preiskovalni fronti zmanjka stanj za obravnavo (vrstice $5-7$). V prvem primeru je iskanje bilo uspešno in algoritem vrne preiskovalno drevo T z označenim končnim stanjem s, v drugem pa iskanje ni uspelo, algoritem vrne zgolj preiskovalno drevo.

Za obravnavano stanje s algoritem naprej izračuna množico prehodov z uporabo klica $P(s)$ funkcije prehodov P (vrstica št. 13). Za vsak možen prehod $\langle t, c_t\rangle$ iz te množice, algoritem najprej preveri, če je stanje t že v preiskovalnem drevesu T na veji od korenskega vozlišča s_0 do vozlišča s. Če je, nadaljuje naprej z naslednjim prehodom, sicer pa algoritem doda preiskovalnemu drevesu T novi list t in ga poveže z nadrejenim vozliščem s. Za vozlišče t izračuna vrednost funkcije g z rekurzivno formulo $g(t) = g(s) + c_t$ in ga nato doda preiskovalni fronti F (vrstici $15-16$).

Bistvena lastnost preiskovalnih algoritmov, poleg običajne računske zahtevnosti, je njihova zmožnost najti katerokoli rešitev, če le ta obstaja, in zmožnost najti optimalno rešitev.

Definicija 2.4. Preiskovalni algoritem je *popoln*, če algoritem zagotavlja, da bo našel rešitev (pot iz začetnega do končnega stanja), če le ta obstaja. Preiskovalni algoritem je *optimalen*, če najde optimalno rešitev, t.j., najcenejšo pot.

Računsko zahtevnost preiskovalnih algoritmov pa ocenjujemo glede na nekaj mer kompleksnosti obravnavanega prostora stanj pa še na osnovi največje dovoljene globine preiskovanega drevesa.

Definicija 2.5. *Maksimalna globina* m je največja dovoljena globina preiskovalnega drevesa.

Kot bomo spoznali v nadaljevanju, način izbire vozlišča za razvejanje iz preiskovalne fronte F v vrstici št. 8 Algoritma 1 bistveno vpliva na računsko zahtevnost, popolnost in optimalnost preiskovalnega algoritma. Na osnovi načina izbire, preiskovalne algoritme razvrščamo v skupini neinformiranih in informiranih algoritmov.

2.1. **Neinformirani preiskovalni algoritmi.** Ti algoritmi izbirajo vozlišče za razvejanje iz preiskovalne fronte zgolj na osnovi globine vozlišča v preiskovalnem drevesu: imenujemo jih neinformirani, ker pri izbiri vozlišča za razvejanje ne uporabljajo informacije o cenah prehodov med stanji. Algoritem *preiskovanja v širino* izbere enega izmed vozlišči preiskovalne fronte, ki imajo najmanjšo globino. Algoritem *preiskovanja v globino* izbere enega izmed vozlišči z največjo globino.

Zgled 2.6. Video v spletni učilnici ponazori delovanje algoritmov preiskovanja v širino in globino na primeru prostora stanj za reševanje linearnih diofantskih enačb.

Algoritem preiskovanja v širino je popoln, saj sistematično preiskuje prostor stanj in bo našel rešitev oziroma končno stanje, ki je najbližje (merjeno v številu potrebnih prehodov) začetnemu stanju s_0. Najkrajša pot pa ni nujno najcenejša, zato algoritem preiskovanja v širino ni optimalen. Prostorska in časovna zahtevnost algoritma sta $O(b^d)$: to je namreč ocena števila stanj, ki jih algoritem mora obravnavati zato, da doseže najbližje končno stanje.

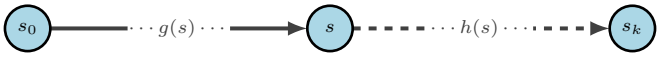
Algoritem preiskovanja v globino ni niti popoln, saj v primeru neskončnega prostora stanj lahko zabrede v neskončno globoko vejo preiskovalnega drevesa, ki ne vsebuje končnega stanja. Časovna zahtevnost algoritma je še vedno eksponentna, tokrat glede maksimalne globine preiskovalnega drevesa m, torej $O(b^m)$. Velika prednost preiskovanja v globino je njegova nizka prostorska zahtevnost $O(bm)$.

Vaja 2.7. Na osnovi ponazoritve delovanja algoritmov preiskovanja iz prejšnjega zgleda premislite in odgovorite na naslednja dva vprašanja. Zakaj je prostorska zahtevnost preiskovanja v globino manjša od prostorske zahtevnosti preiskovanja v širino? Zakaj imata oba algoritma enako časovno zahtevnost?

Oba algoritma torej imata nedopustno visoko časovno zahtevnost, ki je eksponentno narašča z naraščanjem globine d najbolj plitvega končnega stanja. Za reševanje problemov s preiskovanjem rabimo bolj učinkovite preiskovalne algoritme.

2.2. **Informirani preiskovalni algoritmi.** Algoritmi iz te skupine upoštevajo informacijo o cenah prehodov med stanji pri izbiri vozlišča za razvejanje.

Preiskovalni algoritem najprej najboljši izbere tisto stanje s iz preiskovalne fronte, ki ima najnižjo vrednost $g(s)$. Teo stanje dobi pridevnik *najboljše*, ker ima v trenutku izbire najnižjo skupno ceno prehodov na veji od začetnega stanja. Algoritem pogosto obravnavamo kot predstavnika širšega razreda *požrešnih algoritmov*, ki pri izbirah sprejmejo optimalno določitev na osnovi trenutnega, lokalnega stanja.



Slika 2. Osnova za izbiro vozlišča za razvejanje v preiskovalnem algoritmu A^*.

Vaja 2.8. V prostoru stanj za reševanje linearnih diofantskih enačb so vse cene prehodov enake 1. Kaj lahko v tem primeru povemo o razmerju med preiskovalnim algoritmom najprej najboljši in preiskovanjem v širino?

Algoritem najprej najboljši je popoln, saj podobno kot preiskovanje v širino sistematično preiskuje prostor stanj in bo našel tako rešitev oziroma končno stanje, ki je najcenejše dosegljivo iz začetnega stanja s_0. Zato je algoritem najprej najboljši tudi optimalen. Prostorska in časovna zahtevnost algoritma sta, zaradi sistematičnosti obravnave vozlišč preiskovalnega drevesa, še vedno eksponentno časovno in prostorsko zahtevnost $O(b^d)$.

3. Hevristična funkcija in algoritem A^*

V skupino informiranih preiskovalnih algoritmov sodi tudi hevristično preiskovanje. Ključen pojem, ki ga potrebujemo za definicijo tega algoritma je hevristika oziroma hevristična funkcija. Neformalno, hevristika je tehnika reševanja problemov, ki pogosto sloni na »pravilu palca« (angl. *rule of thumb*) in nam v računalništvu pomaga algoritmično reševati probleme, za katere bi tradicionalni algoritmi brez hevristike potrebovali nedopustno veliko časa. Pri preiskovalnih algoritmih hevristiko implementiramo v obliki hevristične funkcije, ki jo uporabljamo pri izbiri vozlišča za razvejanje iz preiskovalne fronte.

Definicija 3.1. *Hevristična funkcija* $h : S \to \mathbb{R}$ v prostoru stanj $\mathcal{S} = \langle S, P, s_0, S_k \rangle$ je funkcija, ki za podano stanje s *izračuna oceno cene* najcenejše poti od stanja s do enega izmed končnih stanj $s_k \in S_k$.

Zgled 3.2. Hevristična funkcija za reševanje linearnih diofantskih enačb bi lahko definirali kot $h(\langle x, y \rangle) = |a - (a_x x + a_y y)|$. Funkcija oceni kakšna je »napaka« dvojice $\langle x, y \rangle$ glede na podano diofantsko enačbo $a_x x + a_y y = a$.

Preiskovalni algoritem A^* oziroma A *zvezda* izbere tisto stanje $s \in F$, ki ima minimalno vrednost seštevka $g(s) + h(s)$. Slednji predstavlja oceno cene najcenejše poti iz začetnega stanja v končno stanje, ki gre skozi vozlišče s. Prvi člen seštevka $g(s)$ namreč poda *točno* ceno poti iz začetnega stanja s_0 v stanje s (glej Definicijo 2.1), drugi, $h(s)$ pa oceno minimalne cene poti iz s v enega izmed končnih stanj. Situacijo ponazori Slika 2: polna puščica ustreza točni ceni poti, ki jo že poznamo, črtkana puščica pa poti do končnega vozišča, ki jo še ne poznamo.

Ali je algoritem A^* popoln oziroma optimalen? Odgovor na to vprašanje je odvisen od lastnosti hevristične funkcije h. Naslednji dve definiciji se nanašata na lastnost funkcije h, ki zagotavlja popolnost in optimalnost algoritma A^*.

Definicija 3.3. *Vsevedna funkcija* $h^* : S \to \mathbb{R}$ v prostoru stanj $\mathcal{S} = \langle S, P, s_0, S_k \rangle$ je funkcija, ki za podano stanje s *izračuna ceno* najcenejše poti od stanja s do enega izmed končnih stanj $s_k \in S_k$.

Vsevedne funkcije za podan prostor stanje nikoli zares ne poznamo. Če bi jo namreč res lahko učinkovito izračunali, pomeni, da problem lahko učinkovito rešimo brez preiskovalnih algoritmov. Zato je hevristična funkcija h običajno le približek vsevedne funkcije h^*. Naslednja definicija definira razmerje med hevristično in vsevedno funkcijo, ki zagotavlja optimalnost in popolnost algoritma A^*.

Definicija 3.4. Hevristična funkcija h v prostoru stanj $\mathcal{S} = \langle S, P, s_0, S_k \rangle$ je *sprejemljiva*, če njena vrednost nikoli ne presega vrednosti vsevedne funkcije h^*, torej, če velja $\forall s \in S : h(s) \leq h^*(s)$.

Vaja 3.5. Pokaži, da hevristična funkcija h iz Primera 3.2 ni sprejemljiva: poišči protiprimer stanja s, za katerega lahko dokažeš, da velja $h(s) > h^*(s)$. Popravi funkcijo h tako, da bo sprejemljiva.

Izrek 3.6. *Algoritem A^* je popoln in optimalen, če uporablja* sprejemljivo *hevristično funkcijo.*

Dokaz. Označimo s $p = s_0, s_1, \ldots, s_k \in S_K$ pot oziroma rešitev, ki jo je našel preiskovalni algoritem A^*. Iz delovanja Algoritma 1 vemo, da je v trenutku, ko je preiskovalni algoritem odločil, da je pot p rešitev, izbral vozlišče s_k s konca te poti za razvejanje (glej vrstice $9 - 12$ Algoritma 1). To pomeni, da je v tem trenutku veljalo, da je vrednost $g(s_k) + h(s_k)$ *minimalna* vrednost seštevka $g(s) + h(s)$ za stanja iz preiskovalne fronte F, torej velja

$$\forall s \in F : g(s) + h(s) \geq g(s_k) + h(s_k). \tag{1}$$

Nadalje, ker je h sprejemljiva lahko sklepamo, da je njena vrednost v vseh končnih vozliščih enaka nič, torej je $h(s_k) = 0$ in $g(s_k) + h(s_k) = g(s_k)$, ker je obenem cena poti p, torej velja $\text{cena}(p) = g(s_k)$.

Denimo, da algoritem A^* ni optimalen, zatorej v prostoru stanj obstaja alternativna pot p', ki je cenejša od rešitve p. Predpostavimo torej, da velja $\text{cena}(p) > \text{cena}(p')$ oziroma $g(s_k) > \text{cena}(p')$.

Naj gre pot p' skozi vozlišče $s' \neq s_k$ iz zgoraj omenjene preiskovalne fronte F. Ker je $s' \in F$, iz Enačbe 1 lahko sklepamo, da za s' velja

$$g(s') + h(s') \geq g(s_k) + h(s_k) = g(s_k). \tag{2}$$

Kaj lahko povemo o ceni poti p' glede na to, da gre skozi vozlišče s'? Cena poti p' je enaka seštevku $g(s') + h^*(s')$. A ker je h sprejemljiva funkcija velja $\text{cena}(p') = g(s') + h^*(s') \geq g(s') + h(s')$. Če pri tem upoštevamo Enačbo 2, dobimo $\text{cena}(p') \geq g(s_k) = \text{cena}(p)$. Kar je v protislovju s predpostavko, da je pot p' cenejša od p. □

Očitno sta časovna in prostorska zahtevnost algoritma A^* odvisni od hevristične funkcije h. Izkaže se pravzaprav, da sta odvisni od relativne napake ocene, ki jo izračuna hevristična funkcija, ki jo definiramo kot $\epsilon = \max_{s \in S \setminus S_k} (h^*(s) - h(s))/h^*(s)$. Pri tako definirani napaki hevristične funkcije, lahko pokažemo, da ima algoritem A^* časovna zahtevnost $O(b^{\epsilon d})$. Ta je še vedno eksponentna, a vidimo, da lahko hevristična funkcija, ki je dober približek vsevedne, lahko mnogokratno izboljša ;;časovno učinkovitost preiskovanja. Izpeljava in dokaz navedene časovne zahtevnosti je izven dosega tega predavanja.

4. Naloge za nadaljnji premislek in delo

Naloga 4.1. Preglej programsko kodo, ki implementira preiskovalne algoritme predstavljene na predavanju. Ugotovi kako to kodo uporabimo za reševanje linearnih diofantskih enačb.

Naloga 4.2. Na šahovnico velikosti 8×8 je treba postaviti osem kraljic tako, da nobena kraljica ne napada nobene druge.

Definiraj prostor stanj, ki omogoča reševanje tega problema.

Naloga 4.3. Denimo, da imamo pred seboj mapo dvanajstih glavnih mest statističnih regij v Sloveniji. Na mapi so narisane povezave med tistimi mesti, kjer obstaja direktna cestna povezava. Povezave so označene z dolžino te cestne povezave v kilometrih. S pomočjo take mape hočemo poiskati najkrajšo pot iz Slovenj Gradca v Novo Gorico.

Definiraj prostor stanj, ki bi omogočal iskanje najkrajše poti. Definiraj sprejemljivo hevristično funkcijo za reševanje tega problema.

Namig: programska koda, dostopna v spletni učilnici, vključuje kodo za reševanje tega problema na primeru mape Romunije.

Naloga 4.4. Imamo na voljo nekaj posod različnih kapacitet, največja posoda je na začetku polna vode. Naloga je odmeriti vnaprej podano količino vode v eni (katerikoli) posodi s prelivanjem vode med posodami. Veljajo naslednja pravila za prelivanje vode:

- Vodo lahko le prelivamo iz ene posode v drugo, ne smemo jo izlivati ven iz posod, prav tako v posode ne smemo dodajati vodo iz zunanjih virov;
- Vodo iz posode A v B lahko prelijemo le, če A ni prazna in B ni polna;
- Iz posode A v posodo B lahko prelijemo toliko vode, kot omogoča kapaciteta posode B: posodo B napolnimo z vodo in/ali posodo A izpraznimo (lahko se zgodi oboje hkrati);
- Cena prelivanja ustreza količini prelite vode.

Konkreten primer take naloge vključuje tri posode kapacitet 3, 5 in 8 litrov. Na začetku je največja posoda kapacitete 8 litrov polna vode. Cilj je odmeriti 4 litre vode v eni izmed podanih posod.

Definiraj prostor stanj za konkretno nalogo treh posod. Definiraj prostor stanj za poljubno nalogo tega tipa, ki vključuje poljubno število posod in podano ciljno količino vode, ko jo moramo odmeriti. Programski kodi, ki implementira preiskovalne algoritme, dodaj kodo, ki implementira definiran prostor stanj in s pomočjo preiskovanja reši konkreten primer naloge s tremi posodami.

Literatura

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.

Ljupčo Todorovski, Univerza v Ljubljani, Fakulteta za matematiko in fiziko, Institut Jožef Stefan, Odsek za tehnologije znanja (E8)