Preiskovalni algoritmi

Seminarska naloga pri predmetu Umetna inteligenca

Mentor: as. mag. Petar Vračar

Avtorja: Žan Bizjak[63170055]

Žiga Šolar[63170282]

Ljubljana, januar 2019

Kazalo vsebine

[1. Uvod 3](#_Toc535186370)

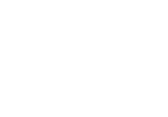
[2. Iskanje v globino 4](#_Toc535186371)

[3. Iskanje v širino 4](#_Toc535186372)

[4. A\* algoritem 5](#_Toc535186373)

[5. IDA\* 5](#_Toc535186374)

[6. Genetski algoritem 5](#_Toc535186375)



# Uvod

Seminarska naloga govori o raziskovalni nalogi, kjer je cilj implementirati in uporabiti preiskovalne alogritme za iskanje poti v labirintu. Labirinti so podani v tekstovni datoteki v obliki matrike. Predstavljeni bodo algoritmi iskanja v globino, iskanja v širino, A\* in iterativni A\*.

# Iskanje v globino

Za algoritem iskanja v globino je implementacija v Pythonu, matrika iz tekstovne datoteke je spremenjena v slovar sosedov kjer je ključ slovarja terka x in y koordinate matrike, za vrednosti pa je seznam terk koordinat sosedov. Algoritem je implementiran na podlagi sklada in množice. V množico shranjujemo že obiskanje celice, v sklad pa celice/veje drevesa. Algoritem teče dokler ne najde celice, katero podamo kot cilj.

|  |  |  |  |
| --- | --- | --- | --- |
| Labirint | Največja globina | Cena najcenejše poti | Dolžina poti |
| 1 | 5 | 100 | 27 |
| 2 | 11 | 92 | 25 |
| 3 | 16 | 164 | 43 |
| 4 | 28 | 300 | 77 |
| 5 | 5 | 57 | 27 |
| 6 | 11 | 49 | 25 |
| 7 | 16 | 124 | 43 |
| 8 | 28 | 231 | 77 |
| 9 | 5 | 57 | 27 |
| 10 | 11 | 55 | 25 |
| 11 | 16 | 99 | 43 |
| 12 | 28 | 189 | 77 |

# Iskanje v širino

Tako kot pri iskanju v globino je pri tem algoritmu matrika predstavljena v slovarju sosedov, vendar pa je za sam algoritem uporabljena vrsta in množica za obiskane celice, v objekt vertex shranjujemo trenutno celico, jo dodamo v našo množico in prevejramo, če je trenutna celica enaka našemu cilju.

|  |  |  |  |
| --- | --- | --- | --- |
| Labirint | Število obdelanih vozlišč | Cena najcenejše poti | Dolžina poti |
| 1 | 44 | 100 | 27 |
| 2 | 129 | 92 | 25 |
| 3 | 301 | 164 | 43 |
| 4 | 714 | 300 | 77 |
| 5 | 44 | 57 | 27 |
| 6 | 129 | 49 | 25 |
| 7 | 301 | 124 | 43 |
| 8 | 714 | 231 | 77 |
| 9 | 44 | 57 | 27 |
| 10 | 129 | 55 | 25 |
| 11 | 301 | 99 | 43 |
| 12 | 714 | 189 | 77 |

# A\* algoritem

A\* je hevristični preiskovalni algoritem. Je izjemno hiter, saj izkoristi veliko pomnilnika ( v primrejavi z IDA\* algoritmom opisanim kasneje) in s tem shrani vsa vozlišča katera še more pregledati in tako ne potrati časa. Algoritem lahko izboljšamo tudi tako da zraven še hranimo množico vozlišč, katera smo že preiskali in jih tako ignoriramo. Zaradi tipa naloge je bila hevristika manhattanska razdalja od vozlišča do konca. Da je bila hevristika še malo bolj natančna se je manhattanska razdalja izboljšala tako, da če je bilo med točko in končnim ciljem zid (vrednost -1) se je za enota podvojila ( namesto da hevristiki dodamo vrednost 1, dodamo 2). Iskanje najkrajšega cilja se lahko izvaja v dveh načinih. Prvi način je da za vsako novo točko dodano h končni poti izračunamo, kateri cilj je hevristično najbljiži in se prebližovati najbljižjemu. Drugi način pa je bilo »statično« iskanje za vsak cilj posebaj in nato za rešitev izbrati tisto pot, katera ima najnižjo ceno. Izkazalo se je da je tu druga rešitev boljša, saj algoritem ne more zaiti proti potencialno daljšem cilju.

# IDA\*

IDA\*, kot A\*, je hevristični algoritem. Glavna razlika je princip iskanja. Tu se shranjuje samo trenutno obiskana pot in v primeru, da pot ni optimalna, se vrne na vozlišče, kjer je pot bila še optimalna. Funkcija iskanja je rekurzivna. Zaradi ponovnega pregledovanja vseh vozlišč, je algoritem občuitno počasnejši od A\*.

# Genetski algoritem

Genetski algoritem vzame svoj navdih iz narave in jo na zelo enostaven način proba simulirati. V tem poskusu izvedbe takega algoritma se vsak gen sprehajalca smatral kot premik po labirintu. V genu so shranjene uteži, katere določijo v katero smer se bo sprehajalec obrnil iz šel. Prva generacija ima vse uteži izbrane naključno. Na samo izbiro smeri pa vpliva tudi to, kolikorat je sprehajalec že obiskal sosednje vozlišče. Če vozlišče še ni bilo obiskano, je njegova prioriteta večja. Uteži v genih so: premik v levo, desno, gor ali dol in uteži za vsako končno polje. Te uteži se upoštevajo, ko se računa hevristika za vsa končna polja in na koncu njihovo povprečje. Začetna generacija vsebuje 200 sprehajalcev. Ko vsi opravijo svoj obhod se najboljših 20 sprehajalcov raznmnoži in ustvari novo generacijo. Ta generacija nato opravlja isto nalogo. Cilj algoritma je ustvariti najboljšega sprehajalca iz podanih sprehajalcev in uteženih genov. Vsak par sprehajalcev ustvarita dva nova otroka, vsak ima približno 50% genov vsakega starša. Pri razmnožitvi je za enega od dveh otrokov tudi majhen procent mutacije. Te nam preprečijo da bi si generacije bile preveč podobne med sabo.