1 Lahenduskäik

Implemeteeritakse PriorityQueue struktuuri (frontier), kus prioriteet sõltub kasutatavast heuristilisest funktsioonist. Algoritmi alguses lisame PriorityQueuesse alguspunkti. Edaspidi lisatakse sinna current asukoha naabrid vastavalt otsingu tingimustele.

1.1 Ahne otsing

Ahne otsingu puhul sõltub prioriteetfunktsioon ainult heuristilisest funktsioonist ehk f(n) = h(n). Naaber lisatakse PriorityQueuesse, kui naaber on külastamata.

Kuna ei uuendata kaugusi või prioriteeti, siis ahne otsing on üpriski kiire, kuid ei kindlusta lühima teekonna leidmist.

1.2 A* otsing

A* otsingu korral sõltub prioriteetfunktsioon nii heuristilisest funktsioonist kui ka eelneva teekonna pikkusest (cost so far) ehk f(n) = g(n) + h(n). Naaber lisatakse PriorityQueuesse, kui temani jõutud uue teekonna pikkus on lühem eelmisest.

A* otsing on aeglasem kui ahne otsing, aga tagastab lühima tee.

2 Heuristilised funktsioonid

Kasutame kahte erinevat heuristilist funktsiooni:

1. Manhattan kauguse heuristiline funktsioon;
2. koordinaatide suurima nihke heuristiline funktsioon.

Parema tulemuse leiab nendest teine, kuid ajaliselt on efektiivsem esimene. Seega, kui algoritm leiab nii-kui-nii lühima tee (nt A* juhul kui diagonaalis liikumine pole lubatud), siis piisab Manhattani kauguse heuristilisest funktsioonist, sest see tagab õige vastuse ja teeb seda kiiremini. Kui diagonaalis liikumine on lubatud, leiab teine funktsioon mõnevõrra parema tulemuse.

Ahne algoritmi puhul on rohkem kasu teisest, mis tagab (enamjaolt) parema tulemuse ja ka parema aja.

3 Andmed

Koodifaili lõppu on lisatud näidistulemus hw2.py jooksutamisel, kus tuuakse välja erinevate kombinatsioonide leitud teekonna pikkus ja kulunud aeg.