Algorytm A*

Wstęp

Algorytm A* jest bardzo podobny do algorytmu Dijkstry z tą różnicą, iż wykorzystuje on heurystyką do oceny jakoś węzłów i wyboru węzła który w danej chwili należy rozwinąć.

W odróżnienbiu od algorytmu Dijkstry nie wymaga on tablicy zawierającej odległości do wszystkich węzłów w grafie, tylko przechowywane są odległości do węzłów w danej chwili odwiedzanych

Pseudoe kod:

```
G - graf
start - węzeł startowy
koniec - węzeł końcowy
Q <- kolejka protorytetowa
Q.add(start,heurystyka(start,koniec)) //do kolejki dodaj węzeł
startowy, droga wartość to priorytet - odległość od startu do
mety
C <- lista węzłów do odwiedzonych (clesedSet)
M <- mapa gdzie kluczem jest dany węzeł, a wartością para
(rodzic, odległosc)
M.put(start,[prev=null, dist=0])
while ( !isempty(Q) ){
    u <- Q.head() //Pobranie głowy kolejki
    if (u == koniec){
          break; }
     C.add(u); //dodanie węzła u do closedSet
    for (v = u.Neighbors()){ //Iterujemy po sąsiadach węzła u
        if (C.contains(v)){ //jeśli v już odwiedzony (jest w
closedSet)
            continue;}
        distU = M.get(u).dist; //odczytanie odległości od u do
     startu
        tmpDist = distU + G(u,v);  //wyznaczenie nowej
odległości G(u,v) to waga krawędzi pomiędzy u i v
        if (!Q.contains(v)){ //Jeśli kolejka nie zawiera v
            Q.add(v,tmpDist + heurystyka(v,koniec));
//Dodajemy v do kolejki, priorytet uwzględnia heurystykę
            M.put(v,[prev = u, dist = tmpDist]) //do mapy M
dodajemy dany węzeł wraz z poprzednikiem I faktyczną
odległością
        } else {
             distV = M.get(v).dist;
             if (tmpDist < distV) { //Jeśli znaleźliśmy krótsze
     dojście od startu do v to aktualizujemy priorytet kolejki
     i mapę M
```

```
                M.put(v,[prev = u, dist = tmpDist])
                Q.update(v,tmpDist + heurystyka(v,koniec));
              }
          }
      }
}
return M
```

Wynik odczytujemy analizując mapę M. Na wstępie węzeł szukany s ustawiamy na węzeł końcowy, następnie w mapie M rekurencyjnie szukamy węzła s, po jego znalezieniu odczytujemy węzeł który jest rodzicem węzła s i przypisujemy go do s. Całość powtarzamy aż dany węzeł nie będzie miał rodzica.

Zadania

1) Zaimplementuj algorytm A*

2) Rozwiąż problem Rumunii omawiany na wykładzie za pomocą algorytmu A*, jako heurystyki użyj podanej już tablicy odległości. Struktura grafu oraz dane do heurystyki dostępne są w pliku rumunia.mat w postaci zmiennej G oraz cost.

Zmienna G opisuje graf w postaci macierzy sąsiedztwa

Zmienna cost jest tablicą komórkową o dwóch kolumnach, w pierwszej znajduje się nazwa miasta (np. cost{3,1} , to miasto o indeksie 3 nosi nazwę 'Craiova') a w drugiej kolumnie znajduje się wartość heurystyki w postaci odległości Euklidesa pomiędzy danym miastem a Bukaresztem (np. cost{3,2}=160, co oznacza że odległość Euklidesa między Bukaresztem, a Craiova wynosi 160km).

3) Rozwiązać problem układanki/przesuwanki za pomocą algorytmu A*. Jako heurystykę wykorzystaj miarę oceniającą sumę odległość pomiędzy obecnym stanem każdego klocka a stanem porzadanym. Jako odległość zastosuj metrykę cityBlock – Manhattan. Porównaj czas realizacji za pomocą algorytmu A* i przeszukiwania wszerz.

Stan końcowy

1	2	3
4	5	6
7	8	

Stan bieżący

1	2	3
	5	8
6	7	4

Pole	1	2	3	4	5	6	7	8	0	h
Odległość	0	0	0	3	0	3	1	2	3	12

Dla 4 wartość odległości Manhattan = 3 ponieważ musimy wykonać 2 kroki w lewo i jeden do góry 2+1 = 3, dla ósemki aby znalazła się na swoim miejscu musimy wykonać przesunięcie o jeden krok w lewo i o jeden krok w dół, czyli w sumie 2 kroki.