Illia Yatskevich

Paweł Maśluch

Podstawy Sztucznej Inteligencji - Projekt 1

Algorytm A\*

Porównywaliśmy działanie algorytmu A\* z algorytmem Dijkstry i algorytmem Floyda-Warshalla. Wynikiem działania algorytmu jest tablica dwuwymiarowa zawierająca długości najkrótszych ścieżek dla wszystkich możliwych par wierzchołków (jest ich n\*n, gdzie n - liczba wierzchołków.

Spis plików:

- Astar.cpp

- Dijkstra.cpp

- FloydWarshall.cpp

- benchmark.h

- gen\_graph.cpp (jako argument podajemy liczbę wierzchołków)

- time\_test.sh (skrypt do testowania działania algorytmów z dużymi grafami pełnymi, w tym przypadku w funkcji solve() trzeba zakomentować funkcję output(), na wyjściu będzie tylko czas działania algorytmu)

- run.sh (skrypt do testowania działania algorytmów z niewielkimi algorytmami, w tym przypadku w funkcji solve() trzeba odkomentować funkcję output(), na wyjściu będzie zarówno czas, jak i tablica dwuwymiarowa, zawierająca długości najkrótszych ścieżek dla wszystkich możliwych par wierzchołków, dane trzeba ręcznie umieścić w pliku graph.txt)

**Złożoności obliczeniowe:**

1. **Algorytm Floyda-Warshalla:**

O, gdzie V – liczba wierzchołków (ze względu na potrójnie zagnieżdżoną pętlę for   
w algorytmie Floyda-Warshalla)

1. **Algorytm Dijkstry:**

Zwykły algorytm Dijkstry(najkrótsze ścieżki od ustalonego początkowego wierzchołka do wszystkich pozostałych) ma złożoność obliczeniową Żeby znaleźć najkrótsze ścieżki dla wszystkich możliwych par wierzchołków, musimy uruchomić algorytm V razy, więc złożoność będzie równa (E – liczba krawędzi grafu).   
Dla grafów pełnych , więc końcowa złożoność obliczeniowa będzie wynosić .

1. **Algorytm A\*:**

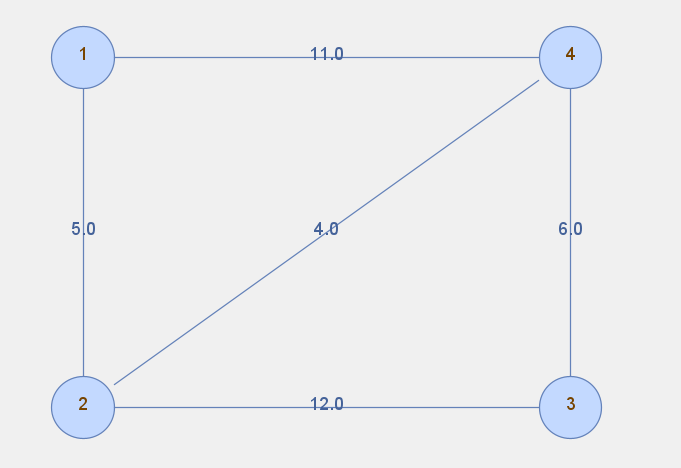
Zwykły algorytm A\*(najkrótsze ścieżki od ustalonego początkowego wierzchołka do ustalonego końcowego) ma złożoność obliczeniową Żeby znaleźć najkrótsze ścieżki dla wszystkich możliwych par wierzchołków, musimy uruchomić algorytm razy, więc złożoność czasowa będzie wynosić . Dla grafów pełnych , więc końcowa złożoność obliczeniowa będzie równa .   
Przypominamy, że E – liczba krawędzi grafu.

Testy:

W pliku gen\_graph.cpp znajduje się generator grafów pełnych o zadanej liczbie wierzchołków z losowymi wagami w przedziale [1;25]. Testy zostały przeprowadzone dla grafów o takich liczbach wierzchołków: {5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140}. W pliku nagłówkowym benchmark.h zostało zdefiniowane narzędzie do zliczania czasu działania algorytmu.

Teraz ten sam wykres, ale ze zmienioną skalą, żeby łatwiej było zobaczyć różnice w złożoności czasowej pomiędzy algorytmami.

Przykład działania programu dla grafu o 4 wierzchołkach i 5 krawędzi.



Zawartość pliku graph.txt:

4 5

1 2 5

2 4 4

4 3 6

3 2 12

1 4 11

Po uruchomieniu skryptu run.sh otrzymujemy następujące wyjście:

----------------------

Astar

0 5 15 9

5 0 10 4

15 10 0 6

9 4 6 0

A\* time: 30

----------------------

----------------------

Dijkstra

0 5 15 9

5 0 10 4

15 10 0 6

9 4 6 0

Dijkstra time: 20

----------------------

----------------------

FloydWarshall

0 5 15 9

5 0 10 4

15 10 0 6

9 4 6 0

Floyd-Warshall time: 0

----------------------

Wnioski.  
Obserwując zachowanie algorytmu A\* oraz algorytmów testowych Dijkstry oraz Floyda-Warshalla, stwierdzamy, że najlepiej do problemu znajdowania najkrótszych ścieżek między wszystkimi parami wierzchołków nadaje się algorytm Floyda-Warshalla (pamiętajmy, że analizowaliśmy przypadek grafów gęstych). Świadczą o tym także uzyskane złożoności obliczeniowe.