**Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera.**

# Plan pomiarów

## Generowanie grafów

Aby uniknąć generowania zbyt wielu grafów, wszystkie algorytmy zostały wykonane na tych samych grafach, nieskierowanych i na płaszczyźnie(wymagane do heurystyki A\*)

Wszystkie grafy zostały wylosowane w następujący sposób:

* + 1. Wylosowanie każdemu wierzchołka dwóch współrzędnych w przedziale [0, 8191]
    2. Dodanie krawędzi 0-1, 1-2, 1-3, … … n-0 w celu zagwarantowania spójności grafu
    3. Utworzenie zbioru wszystkich możliwych krawędzi(z wyłączeniem już dodanych)
    4. Wyliczenie na podstawie zadanej gęstości docelowej ilości krawędzi
    5. Losowe wybieranie ze zbioru wyliczonej ilości krawędzi i dodawanie ich do grafu z wagą odpowiadającą odległości między odpowiadającymi wierzchołkami

## Metoda pomiaru czasu i platforma testowa

Do pomiaru czasu użyto funkcji QueryPerformanceCounter.  
Program kompilowana w trybie realease(optymalizacja o2) w Visual Studio 2015.  
Testy przeprowadzano na laptopie z procesorem Intel Core i7-4720HQ, 6MB cache, taktowanie ograniczone do 2,6 GHz

Wszystkie pomiary wykonano według schematu:

* + 1. Dla każdego rozmiaru grafu
       1. Dla każdej gęstości grafu
          1. 100 razy

Wygenerować graf dla obu reprezentacji

Zmierzyć czas wykonania każdego z algorytmów

* + - * 1. Uśrednić czasy

# A\*, wyszukiwanie ścieżki

## Heurystyka

A\* jest algorytmem wymagającym heurystyki do optymalnego działania(w przeciwnym razie staje się wolnym Dijkstrą i w sumie) więc moja implementacja A\* przyjmuje za heurystykę dystans między wierzchołkami(punkt 1a: generowanie grafów).

## Złożoność obliczeniowa

Ponieważ A\* nie rozpatruje wszystkich wierzchołków jeśli nie musi(w instancjach testowych graf jest spójny więc droga zawsze istnieje, a heurystyka jest poprawna) złożoność obliczeniowa jest O(n) gdzie to ilość zbadanych wierzchołków. Ilość badanych wierzchołków powinna rosnąć wraz z rozmiarem grafu co widać na wykresach poniżej

## Wyniki pomiarów

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Liczba wierzchołków | Gęstość(%) | | | | |
| A\* List | 25 | 50 | 75 | 100 |
| 100 | 0,0291 | 0,0277 | 0,0287 | 0,0313 |
| 200 | 0,0687 | 0,0656 | 0,0643 | 0,0774 |
| 300 | 0,1057 | 0,1069 | 0,1090 | 0,1091 |
| 400 | 0,1297 | 0,1437 | 0,1621 | 0,1485 |
| 500 | 0,1672 | 0,1833 | 0,1678 | 0,1775 |
| Gęstość(%) | | | | |
| A\* Matrix | 25 | 50 | 75 | 100 |
| 100 | 0,0469 | 0,0452 | 0,0424 | 0,0401 |
| 200 | 0,1102 | 0,0576 | 0,0754 | 0,0854 |
| 300 | 0,1243 | 0,0975 | 0,1127 | 0,1085 |
| 400 | 0,1336 | 0,1304 | 0,1299 | 0,1425 |
| 500 | 0,1355 | 0,1481 | 0,1872 | 0,1696 |

## 

# Boruvka, szukanie MST

## Złożoność obliczeniowa

Algorytm Boruvki powinien działać ze złożonością czasową O(E log V), co ładnie widać dla grafów o mniejszych gęstości ale trudniej znaleźć dopasowanie dla większych. Prawdopodobnie jest to spowodowane sposobem implementacji łączenia podgrafów, zrealizowaną przez łączenie drzew(struktury danych), wymagające realokacji pamięci.

W przypadku formy listowej i grafu o 400 wierzchołkach, gęstości 50%, albo wylosowało się parę dziwnych grafów, albo windows uznał mój projekt może poczekać aż się zainstalują jakieś aktualizacje. Zignorowałbym tą próbkę.

## Wyniki pomiarów

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Liczba wierzchołków | Gęstość[%] | | | | |
| Boruvka List | 25 | 50 | 75 | 100 |
| 100 | 3,58 | 4,74 | 4,82 | 5,18 |
| 200 | 29,65 | 41,34 | 38,80 | 43,65 |
| 300 | 197,92 | 100,23 | 97,46 | 96,83 |
| 400 | 374,23 | 1049,79 | 216,27 | 202,43 |
| 500 | 383,15 | 363,63 | 283,82 | 234,72 |
| Gęstość[%] | | | | |
| Boruvka Matrix | 25 | 50 | 75 | 100 |
| 100 | 2,03 | 4,07 | 5,58 | 6,38 |
| 200 | 9,32 | 17,60 | 24,54 | 35,08 |
| 300 | 40,28 | 46,86 | 62,72 | 79,63 |
| 400 | 190,66 | 195,47 | 127,90 | 156,83 |
| 500 | 193,28 | 284,48 | 234,78 | 258,82 |

# Ford-Fulkerson, szukanie maksymalnego przepływu

## Złożoność obliczeniowa

Wikipedia podaje złożoność czasową metody Forda-Fulkersona jako O(Ef), gdzie f to wartość maksymalnego przepływu, co na pewno jest prawdą przy grafach nieważonych, ale wydaje mi się być mało precyzyjne, więc spróbuję sam:  
Używam algorytmu przeszukiwania wszerz(BFS) do znajdywania ścieżek w grafie rezydualnym, co ma złożoność O(V + E) co dla gęstych grafów daje O(V2) co pięknie widać na wykresie dla grafów pełnych.  
BFS wykona się maksymalnie E razy więc szacuję całość na O(E\*(V+E) ) = O(E2\*V). Kwadratowy przyrost czasu wykonania widać na większości grafów, więc chyba się zgadza.

Utworzenie grafu rezydualnego wymaga skopiowania grafu żeby nie niszczyć instancji testowej, co prawdopodobnie ma złożoność czasową O(V2) dla reprezentacji macierzowej i O(E) dla listowej ale instrukcje kopiowania są tak proste że można je pominąć.

## Wyniki pomiarów

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Liczba wierzchołków | Gęstość[%] | | | | |
| FF List | 25 | 50 | 75 | 100 |
| 100 | 67,88 | 117,38 | 119,53 | 121,72 |
| 200 | 569,30 | 930,68 | 928,55 | 988,11 |
| 300 | 2634,33 | 3739,09 | 3743,66 | 3714,19 |
| 400 | 4198,57 | 6886,71 | 9230,01 | 9645,44 |
| 500 | 8811,69 | 10605,00 | 17033,71 | 18114,96 |
| Gęstość[%] | | | | |
| FF Matrix | 25 | 50 | 75 | 100 |
| 100 | 11,14 | 35,57 | 80,33 | 122,55 |
| 200 | 95,53 | 341,67 | 685,03 | 1052,70 |
| 300 | 333,53 | 1263,37 | 2476,55 | 4244,06 |
| 400 | 974,59 | 3291,90 | 6080,16 | 8666,88 |
| 500 | 2114,18 | 6497,04 | 12089,07 | 16735,24 |

# Konkluzje

Reprezentacja macierzowa jest szybsza, chociaż dla grafów rzadkich i drzew będzie zajmować więcej pamięci niż reprezentacja listowa grafu.

A\* jest wspaniały i rozumiem dlaczego jest powszechnie wykorzystywana przy tworzeniu gier komputerowych.

Moja implementacja Boruvki jest okropna(zbiory rozłączne), ale algorytm ma duży potencjał ponieważ pozwala na wykorzystanie wielu wątków.

# Literatura

http://eduinf.waw.pl/inf/alg/001\_search/index.php

https://www.wikipedia.org/

Procedural Content Generation for C++ Game Development, Dale Green, ISBN 978-1-78588-671-3, rozdział 8