PAMSI 8,9

Wygenerowano przez Doxygen 1.8.6

Cz, 12 cze 2014 03:18:44

Spis treści

1 Strona główna

Laboratorium 8 i 9.

Algorytmy przeszukiwania grafu.

Autor

Jakub Chmiel 200314

2 Indeks klas

2.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

3 Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

4 Dokumentacja klas

4.1 Dokumentacja klasy AStar

Algorytm szukania A∗.

`#include <AStar.h>`

Statyczne metody publiczne

- static bool szukaj (Graf &graf, int W1, int W2, list< int > &sciezka)
  Algorytm przeszukiwania grafu A∗.

Statyczne atrybuty publiczne

- static Graf ∗ **gr** = NULL

4.1.1 Opis szczegółowy

Algorytm szukania A∗.

4.1.2 Dokumentacja funkcji składowych

4.1.2.1 bool AStar::szukaj (Graf & *graf,* int *W1,* int *W2,* list< int > & *sciezka*) [static]

Algorytm przeszukiwania grafu A∗.

Parametry

graf	graf w ktorym poszukujemy sciezki.
W1	indeks poczatkowego wierzcholka.
W2	indeks docelowego wierzcholka.
sciezka	wynikowa sciezka jesli udalo sie taka znalezc.

Zwraca

stan powodzenia funkcji.

Dokumentacja dla tej klasy została wygenerowana z plików:

- AStar.h
- AStar.cpp

4.2 Dokumentacja klasy BreadthSearch

Algorytm szukania wszerz.

`#include <BreadthSearch.h>`

Statyczne metody publiczne

- static bool szukaj (Graf &graf, int W1, int W2, list< int > &sciezka)

  Algorytm przeszukiwania grafy wszerz.

4.2.1 Opis szczegółowy

Algorytm szukania wszerz.

4.2.2 Dokumentacja funkcji składowych

4.2.2.1 bool BreadthSearch::szukaj (Graf & *graf,* int *W1,* int *W2,* list< int > & *sciezka*) [static]

Algorytm przeszukiwania grafy wszerz.

Parametry

graf	graf w ktorym poszukujemy sciezki.
W1	indeks poczatkowego wierzcholka.
W2	indeks docelowego wierzcholka.
sciezka	wynikowa sciezka jesli udalo sie taka znalezc.

Zwraca

stan powodzenia funkcji.

Dokumentacja dla tej klasy została wygenerowana z plików:

- BreadthSearch.h
- BreadthSearch.cpp

4.3 Dokumentacja klasy DepthSearch

Algorytm szukania wszerz.

`#include <DepthSearch.h>`

Statyczne metody publiczne

- static bool szukaj (Graf &graf, int W1, int W2, list< int > &sciezka)

  Algorytm przeszukiwania grafy wglab.

4.3.1 Opis szczegółowy

Algorytm szukania wszerz.

4.3.2 Dokumentacja funkcji składowych

4.3.2.1 bool DepthSearch::szukaj (Graf & *graf,* int *W1,* int *W2,* list< int > & *sciezka*) [static]

Algorytm przeszukiwania grafy wglab.

Parametry

graf	graf w ktorym poszukujemy sciezki.
W1	indeks poczatkowego wierzcholka.
W2	indeks docelowego wierzcholka.
sciezka	wynikowa sciezka jesli udalo sie taka znalezc.

Zwraca

stan powodzenia funkcji.

Dokumentacja dla tej klasy została wygenerowana z plików:

- DepthSearch.h
- DepthSearch.cpp

4.4 Dokumentacja klasy Graf

Implementacja grafu.

`#include <Graf.h>`

Metody publiczne

- Graf (const Graf &inny)

  Zwraca wierzcholek o danym indeksie.
- Wierzcholek & wierzcholek (int W)

  Zwraca wierzcholek o podanym indeksie.
- int dodaj_wierzcholek ()

  Tworzy nowy wierzcholek bez zadnych krawedzi.
- int dodaj_wierzcholek (int x, int y)

  Tworzy nowy wierzcholek bez zadnych krawedzi, z ustawionymi wspolrzednymi.
- int ilosc_wierzcholkow ()

  Zwraca ilosc wszystkich wierzcholkow.
- bool dodaj_krawedz (int W1, int W2, TYP_KOSZTU koszt)

  Dodaje krawedz miedzy dwoma wierzcholkami.
- bool usun_wierzcholek (int W)

  Usuwa wierzcholek i krawedzie ktore z niego wychodza.
- bool usun_krawedz (int W1, int W2)

Usuwa krawedz miedzy dwoma wierzcholkami.

- bool czy_polaczone (int W1, int W2)

  Sprawdza czy istnieje krawedz miedzy dwoma wierzcholkami.
- void **sasiedztwo** (int W, vector< int > l_sasiadow, vector< int > l_kosztow)
- int znajdz_losowy_wierzcholek ()

  Zwraca indeks losowego wierzcholka z grafu.
- void reset_zmiennych_pomocniczych ()

  Resetuje zmienne pomocnicze wszystkich wierzcholkow.
- list< int > zrekonstruuj_sciezke (int W)

  Tworzy liste ktora prowadzi od podanego wierzcholka do wierzcholka bez nastepnego wyboru.
- void generuj_graf (int rozmiar)

  Czysci graf i generuje nowy o zblizonym ksztalcie do kwadratu.
- void rysuj_graf ()

  Rysuje graf na standardowym wyjsciu.
- void wspolrzedne_wierzcholka (int W)

  Zwraca tekst opisujacy wspolrzedne danego wierzcholka.
- int heurystyka (int W1, int W2)

  Oblicza heurystyke dla drogi pomiedzy dwoma wierzcholkami.

4.4.1 Opis szczegółowy

Implementacja grafu.

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 Graf::Graf (const Graf & *inny*)

Zwraca wierzcholek o danym indeksie.

Parametry

index	index wierzcholka.
wynik	znaleziony wierzcholek.

Zwraca

Sukces. Konstruktor kopiujacy.

Parametry

inny	kopiowany graf.

4.4.3 Dokumentacja funkcji składowych

4.4.3.1 bool Graf::czy_polaczone (int *W1*, int *W2*)

Sprawdza czy istnieje krawedz miedzy dwoma wierzcholkami.

Parametry

W1	indeks pierwszego wierzcholka.

W2	indeks drugiego wierzcholka.

4.4.3.2 bool Graf::dodaj_krawedz (int *W1,* int *W2,* TYP_KOSZTU *koszt*)

Dodaje krawedz miedzy dwoma wierzcholkami.

Parametry

W1	indeks pierwszego wierzcholka.
W2	indeks drugiego wierzcholka.
koszt	koszt/waga krawedzi.

Zwraca

Sukces.

4.4.3.3 int Graf::dodaj_wierzcholek ( )

Tworzy nowy wierzcholek bez zadnych krawedzi.

Zwraca

indeks dodanego wierzcholka.

4.4.3.4 int Graf::dodaj_wierzcholek (int *x,* int *y*)

Tworzy nowy wierzcholek bez zadnych krawedzi, z ustawionymi wspolrzednymi.

Zwraca

indeks dodanego wierzcholka.

4.4.3.5 void Graf::generuj_graf (int *rozmiar*)

Czysci graf i generuje nowy o zblizonym ksztalcie do kwadratu.

Parametry

rozmiar	bok kwadratu.

4.4.3.6 int Graf::heurystyka (int *W1,* int *W2*)

Oblicza heurystyke dla drogi pomiedzy dwoma wierzcholkami.

Parametry

W1	wierzcholek 1.
W2	wierzcholek 2.

Zwraca

Heurystyka

4.4.3.7 bool Graf::usun_krawedz (int *W1,* int *W2*)

Usuwa krawedz miedzy dwoma wierzcholkami.

Parametry

W1	indeks pierwszego wierzcholka.
W2	indeks drugiego wierzcholka.

4.4.3.8 bool Graf::usun_wierzcholek (int *W*)

Usuwa wierzcholek i krawedzie ktore z niego wychodza.

Parametry

W	indeks wierzcholka.

Zwraca

Sukces.

4.4.3.9 Wierzcholek & Graf::wierzcholek (int *W*)

Zwraca wierzcholek o podanym indeksie.

Parametry

W	indeks wierzcholka.

4.4.3.10 void Graf::wspolrzedne_wierzcholka (int *W*)

Zwraca tekst opisujacy wspolrzedne danego wierzcholka.

Parametry

W	wierzcholek.

Zwraca

tekst postaci (x, y)

4.4.3.11 list< int > Graf::zrekonstruuj_sciezke (int *W*)

Tworzy liste ktora prowadzi od podanego wierzcholka do wierzcholka bez nastepnego wyboru.

Zwraca

lista indeksow sciezki.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Graf.h
- Graf.cpp

4.5 Dokumentacja klasy PorownajWierzcholki

Metody publiczne

- bool **operator()** (int &W1, int &W2)

Dokumentacja dla tej klasy została wygenerowana z pliku:

- AStar.h

4.6 Dokumentacja klasy Wierzcholek

Struktura wierzcholka grafu.

`#include <Wierzcholek.h>`

Metody publiczne

- **Wierzcholek** (int x, int y)
- Wierzcholek (const Wierzcholek &inny)

  Konstruktor kopiujacy.
- void dodaj_krawedz (int W, TYP_KOSZTU koszt)

  Dodaje nowa krawedz miedzy tym wierzcholkiem a podanym.
- bool usun_krawedz (int W)

  Usuwa krawedz z list tylko tego wierzchola.
- bool usun_wszystkie_krawedzie ()

  Usuwa wszystkie krawedzie miedzy tym wierzcholkiem a wszystkimi sasiadami.
- void **usun** ()
- bool **czy_istnieje** ()
- bool czy_polaczone (int W)

  Sprawdza czy wierzcholki sa polaczone.
- void **sasiedztwo** (vector< int > &l_sasiadow)
- void **sasiedztwo** (vector< int > &l_sasiadow, vector< int > &l_kosztow)

Atrybuty publiczne

- int **x**
- int **y**
- bool **odwiedzono**
- int **poprzedni_wierzcholek**
- int **koszt_sciezki**
- int **szacowany_koszt**

4.6.1 Opis szczegółowy

Struktura wierzcholka grafu.

4.6.2 Dokumentacja konstruktora i destruktora

4.6.2.1 Wierzcholek::Wierzcholek (const Wierzcholek & *inny*)

Konstruktor kopiujacy.

Parametry

inny	Kopiowany wierzcholek

4.6.3 Dokumentacja funkcji składowych

4.6.3.1 bool Wierzcholek::czy_polaczone (int *W*)

Sprawdza czy wierzcholki sa polaczone.

Parametry

W	docelowy wierzcholek.

4.6.3.2 void Wierzcholek::dodaj_krawedz (int *W*, TYP_KOSZTU *koszt*)

Dodaje nowa krawedz miedzy tym wierzcholkiem a podanym.

Parametry

W	docelowy wierzcholek.
koszt	koszt/waga krawedzi.

4.6.3.3 bool Wierzcholek::usun_krawedz (int *W*)

Usuwa krawedz z list tylko tego wierzchola.

Parametry

W	docelowy wierzcholek.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Wierzcholek.h
- Wierzcholek.cpp

5 Dokumentacja plików

5.1 Dokumentacja pliku AStar.cpp

```
#include "AStar.h"
```

5.2 Dokumentacja pliku AStar.h

```
#include <queue>
#include "Definicje.h"
#include "Graf.h"
```

Komponenty

- class AStar

  Algorytm szukania A∗.
- class PorownajWierzcholki

5.3 Dokumentacja pliku BreadthSearch.cpp

```
#include "BreadthSearch.h"
```

5.4 Dokumentacja pliku BreadthSearch.h

```
#include <queue>
```

```
#include "Definicje.h"
#include "Graf.h"
```

Komponenty

- class BreadthSearch

  Algorytm szukania wszerz.

5.5 Dokumentacja pliku Definicje.h

Plik zawiera ogolne instrukcje preprocesora wspolne dla wszystkich plikow zrodlowych.

```
#include <vector>
#include <list>
#include <iostream>
#include <fstream>
#include <time.h>
#include <stdlib.h>
```

Definicje

- #define **TYP_KOSZTU** int
- #define **BEZ_KOSZTU** 0

5.5.1 Opis szczegółowy

Plik zawiera ogolne instrukcje preprocesora wspolne dla wszystkich plikow zrodlowych.

5.6 Dokumentacja pliku DepthSearch.cpp

```
#include "DepthSearch.h"
```

5.7 Dokumentacja pliku DepthSearch.h

```
#include <stack>
#include "Definicje.h"
#include "Graf.h"
```

Komponenty

- class DepthSearch

  Algorytm szukania wszerz.

5.8 Dokumentacja pliku Graf.cpp

```
#include "Graf.h"
```

5.9 Dokumentacja pliku Graf.h

```
#include "Wierzcholek.h"
#include "Definicje.h"
```

Komponenty

- class Graf

  Implementacja grafu.

5.10 Dokumentacja pliku Grafy.cpp

```
#include "Grafy.h"
```

Funkcje

- bool zapisz_dane (const char ∗nazwa_pliku, int ∗col_rozmiar_problemu, double ∗col_czas, int rozmiar)

  zapisuje dane do pliku .csv dane zawieraja: 1 kolumna: wielkosc problemu 2 kolumna: czas potrzebny do zrealizowanego danego problemu
- void testuj_algorytm (bool(∗algorytm)(Graf &graf, int W1, int W2, list< int > &sciezka), const char ∗plik_wyjsciowy)

  wykonuje testy czasu algorytmu dla przygotowanych parametrow zmierzone czasu zapisuje do pliku
- int **main** ()

5.10.1 Dokumentacja funkcji

5.10.1.1 void testuj_algorytm (bool(∗)(Graf &graf, int W1, int W2, list< int > &sciezka) *algorytm,* const char ∗ *plik_wyjsciowy*)

wykonuje testy czasu algorytmu dla przygotowanych parametrow zmierzone czasu zapisuje do pliku

Parametry

double(∗algorytm)	funkcja z algorytmem do testowania
∗tablica	dane dla algorytmu
rozmiar	rozmiar tablicy
∗plik_wyjsciowy	nazwa pliku do zapisu zmierzonych czasow

5.10.1.2 bool zapisz_dane (const char ∗ *nazwa_pliku,* int ∗ *col_rozmiar_problemu,* double ∗ *col_czas,* int *rozmiar*)

zapisuje dane do pliku .csv dane zawieraja: 1 kolumna: wielkosc problemu 2 kolumna: czas potrzebny do zrealizowanego danego problemu

Parametry

∗nazwa_pliku	nazwa pliku do zapisu
∗col_rozmiar_problemu	tablica z 1 kolumna

∗col_czas	druga kolumna
rozmiar	rozmiar obu tablic

Zwraca

- true sukces
- false blad

5.11 Dokumentacja pliku Grafy.h

```
#include <iostream>
#include <fstream>
#include <time.h>
#include <stdlib.h>
#include "Definicje.h"
#include "Graf.h"
#include "BreadthSearch.h"
#include "DepthSearch.h"
#include "AStar.h"
```

Definicje

- #define LICZBA_POWTORZEN 100

  ilosc powtorzen pomiaru czasu dla kazdego rozmiaru problemu.
- #define LICZBA_WIELKOSCI 7

  ilosc roznych rozmiarow problemu.
- #define WIELKOSCI_PROBLEMU {100,150,200,250,300,350,400}

  tablica zawierajaca wszystkie mierzone rozmiary problemu.

Funkcje

- bool zapisz_dane (const char ∗nazwa_pliku, int ∗col_rozmiar_problemu, double ∗col_czas, int rozmiar)

  zapisuje dane do pliku .csv dane zawieraja: 1 kolumna: wielkosc problemu 2 kolumna: czas potrzebny do zrealizowanego danego problemu
- void testuj_algorytm (bool(∗algorytm)(Graf &graf, int W1, int W2, list< int > &sciezka), const char ∗plik_wyjsciowy)

  wykonuje testy czasu algorytmu dla przygotowanych parametrow zmierzone czasu zapisuje do pliku
- int **main** ()

5.11.1 Dokumentacja funkcji

5.11.1.1 void testuj_algorytm (bool(∗)(Graf &graf, int W1, int W2, list< int > &sciezka) *algorytm,* const char ∗ *plik_wyjsciowy*)

wykonuje testy czasu algorytmu dla przygotowanych parametrow zmierzone czasu zapisuje do pliku

Parametry

double(∗algorytm)	funkcja z algorytmem do testowania

∗tablica	dane dla algorytmu
rozmiar	rozmiar tablicy
∗plik_wyjsciowy	nazwa pliku do zapisu zmierzonych czasow

5.11.1.2 bool zapisz_dane (const char ∗ *nazwa_pliku,* int ∗ *col_rozmiar_problemu,* double ∗ *col_czas,* int *rozmiar*)

zapisuje dane do pliku .csv dane zawieraja: 1 kolumna: wielkosc problemu 2 kolumna: czas potrzebny do zrealizowanego danego problemu

Parametry

∗nazwa_pliku	nazwa pliku do zapisu
∗col_rozmiar_problemu	tablica z 1 kolumna
∗col_czas	druga kolumna
rozmiar	rozmiar obu tablic

Zwraca

- true sukces
- false blad

5.12 Dokumentacja pliku Wierzcholek.cpp

```
#include "Wierzcholek.h"
```

5.13 Dokumentacja pliku Wierzcholek.h

```
#include "Definicje.h"
```

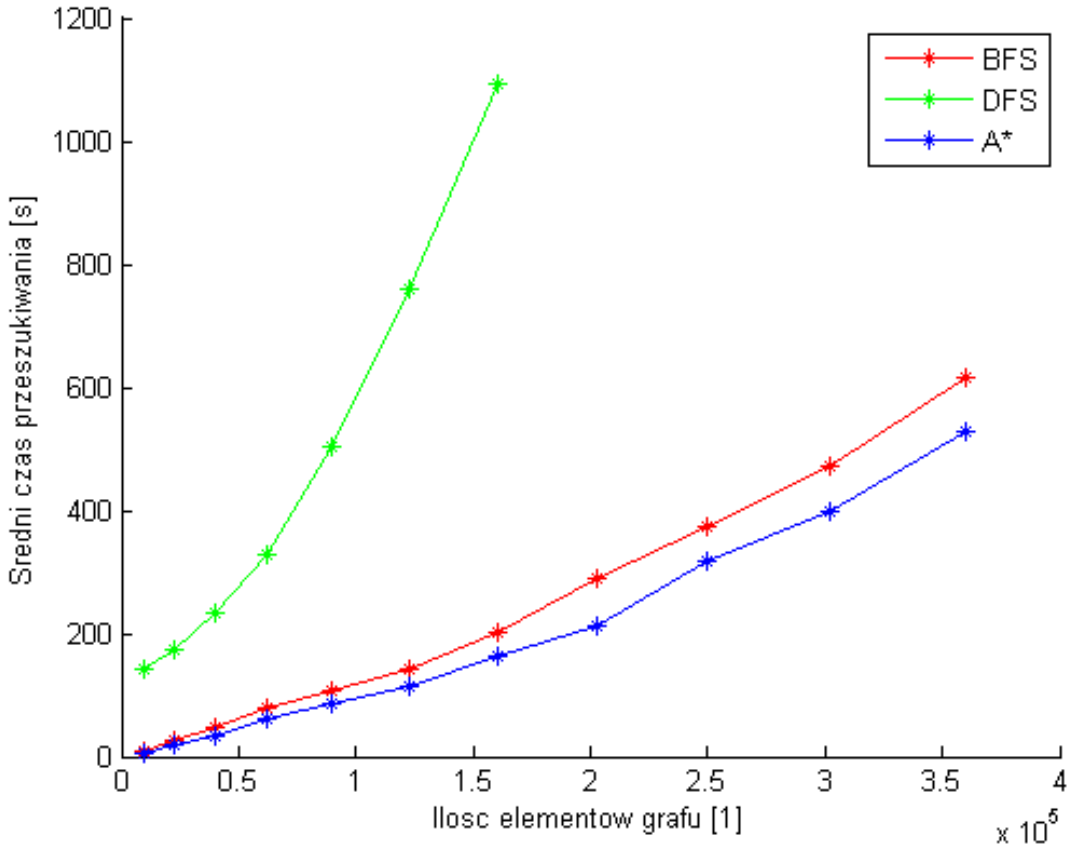
Komponenty

- class Wierzcholek

  Struktura wierzcholka grafu.

6 Sprawozdanie z wykonania programu labolatorium 8 i 9.

Pomiar zlozonosci obliczeniowej dla przeszukiwania grafu wszerz (BFS), wglab (DFS) i algorytmu A*.
Przeszukiwany graf mial postac dwuwymiarowej tablicy, gdzie każdy wierzcholek byl polaczony z 8 sasiednimi. Losowe wierzcholki byly usuwane. Pozwolilo to na zaimplementowanie heurystyki dla algorytmu A* w postaci metryki miejskiej. Pomiary czasu znajduja sie na wykresie 1.
Najgorzej wypadl algorytm DFS, z wyraznie gorszym czasem i mozliwie wykladnicza zlozonoscia. Algorytm BFS i A* maja zblizony czas. Algorytm A* powinien dzialac duzo szybciej od pozostalych. Niewielka roznica moze wynikac z czasochlonnych operacji wykonywanych w algorytmie A* (utrzymanie posortowanej kolejki) lub z nieoptymalnie wybranych punktow pomiedzy, ktorymi szukamy sciezki (BFS moze wyszukac sciezke dosc szybko jesli punkty sa blisko siebie, natomiast DFS przeszukuje wiekszosc wierzcholkow grafu niezaleznie od tego jak blisko siebie leza dane punkty).
Algorytmy BFS i DFS mozna uzyc do przeszukiwania calego grafu, nie nadaja sie dobrze do znajdowania sciezki pomiedzy dwoma wierzcholkami. Zwlaszcza algorytm DFS w danej sytuacji moze przeszukac caly graf poruszajac sie tylko wglab po jednej galezi i znajduje jedna z najdluzszych mozliwych sciezek. Algorytm A* wyszukuje najkrotsze sciezki i robi to duzo szybciej, jednak wymaga aby graf reprezentowal dane, dla ktorych mozliwe jest szacowanie odleglosci miedzy wierzcholkami.



Rysunek 1: Wykres pomiaru czasu przeszukiwania grafu dla wybranych algorytmow.

Skorowidz