Dokumentacja

Opis programu

Program używa algorytmu maksymalnego przepływu o minimalnym koszcie do wyliczenia najtańszego kosztu naprawy dróg w Shire. Świat jest reprezentowany w postaci grafu skierowanego z wieloma źródłami i ujściami. Do grafu dodawane są dwa wierzchołki: źródło i ujście. Dzięki temu możliwe jest obliczenie kosztu dla grafu z wieloma polami (źródłami) i z wieloma karczmami (ujściami). Mają one zerowy koszt, a przepływ reprezentuje produkcję pól, w przypadku źródła, albo pojemność karcz w przypadku ujścia.

Plik minCostFlow.h

```
class  minCostFlow
```

Klasa minCostFlow odpowiada za wczytywanie wejścia z pliku i obliczanie minimalnego kosztu przepływu w grafie.

Plik min_cost_input.cpp

```
void readInput(string fileName);
```

Funkcja przyjmuje na wejściu nazwę pliku z którego wczytujemy graf. Zapisuje go w postaci macierzy sąsiedztwa. Na wyjściu zwraca dwie sieci residualne w postaci macierz sąsiedztwa: kosztu i przepływu.

Plik Dijkstra.cpp

```
int searchGraphDijkstra(vector<int> dist, vector<bool> found);
```

Funkcja przyjmuje na wejściu wektor `dist` z minimalnym kosztem "podróży" do wierzchołka oraz wektor `found` odwiedzonych wierzchołków. Następnie szuka najtańszej ścieżki dla pierwszego, nie odwiedzonego wierzchołka i aktualizuje wektor `dist`. Na wyjściu zwraca numer wierzchołka z którego "przyszliśmy" do szukanego wierzchołka.

```
void getFlowDijkstra(vector<vector<int>> cost, int src);
```

Funkcja na wejściu przyjmuje macierz sąsiedztwa dla sieci residualnej kosztu oraz numer wierzchołka źródła. Następnie szuka najtańszej ścieżki z źródła do ujścia używając algorytmu Dijkstry i zapisuje ją w wektorze `path`.
`path[x]=y;`
x - wierzchołek w którym jesteśmy, y- wierzchołek z którego przyszliśmy.
Na podstawie wektora `path` oblicza koszt, który jest dodawany do kosztu całkowitego.

```
int minCostDijkstra(vector<vector<int>> cap, vector<vector<int>>
cost);
```

Na wejściu przyjmuje macierze sąsiedztwa sieci residualnej kosztu i przepływu. Funkcja w pętli `while` szuka najtańszych ścieżek, do momentu aż nie przejdziemy przez żadną nową krawędź. Na wyjściu zwraca minimalny koszt maksymalnego przepływu.

Plik BellmanFord.cpp

```
bool searchGraphBellmanFord(int src, int sink);
```

Funkcja na wejściu przyjmuje numer wierzchołka źródła i ujścia. Następnie sprawdza czy istnieje ścieżka powiększająca z źródła do ujścia. Na wyjściu zwraca `true` jeśli istnieje, w przeciwnym razie zwraca `false`.

```
output getFlowBellmanFord(int src, int sink);
```

Funkcja na wejściu przyjmuje numer wierzchołka źródła i ujścia. Szuka maksymalnego przepływu minimalnym kosztem używając algorytmu Bellmana-Forda. Na wyjściu zwraca wartość `int` zawierająca minimalny koszt.

```
int minCostBellmanFord(vector<vector<int>>, vector<vector<int>>);
```

Funkcja na wejściu przyjmuje macierze sąsiedztwa sieci residualnych kosztu i przepływu. Następnie tworzy potrzebne zmienne i wywołuje funkcję `getFlowBellmanFord`. Na wyjściu zwraca wartość minimalnego kosztu otrzymaną od wcześniej wspomnianej funkcji.

Plik main.cpp

```
int main()
```

Funkcja główna, która wywołuje funkcje szukania najtańszego, maksymalnego przepływu. Można tu zmienić zmienną `string fileName` zawierającą nazwę pliku z którego wczytujemy dane oraz włączyć lub wyłączyć debuggMode i testMode. DebuggMode wyświetla więcej informacji na wyjściu, a testMode porównuje wynik minimalnego kosztu zwrócone przez dwa różne algorytmy.