AAL – zadanie 1

Dokumentacja końcowa

Tomasz Wiaderek

Treść zadania:

Załóżmy że dany jest spójny graf nieskierowany. Każdy wierzchołek tego grafu posiada liczbę punktów otrzymywaną za jego odwiedzenie równą s $\mathbb{N}$. Podany jest wierzchołek początkowy P oraz wierzchołek końcowy K. Zaproponuj algorytm, który znajdzie ścieżkę D od P do K, która składa się z wierzchołków, których suma punktów **wynosi dokładnie z**, gdzie z = SUMA(svi), gdzie vi jest i-tym wierzchołkiem odwiedzanym przez algorytm a i [1,2,..., N], gdzie N jest liczbą wierzchołków odwiedzonych w grafie.

Zaproponuj algorytm znajdujący dokładnie jedną ścieżkę D. Oceń jego złożoność czasową oraz pamięciową.

Reprezentacja danych:

Przeszukiwany graf jest reprezentowany jako klasa „Graph". Została ona zaimplementowana specjalnie na potrzeby tego projektu. Klasa „Graph" posiada wektor obiektów klasy „Vertex" (mEdges), również zaimplementowany na potrzeby projektu. Obiekt klasy „Vertex" ma reprezentować każdy wierzchołek w grafie. Klasa ta zawiera informacje na temat identyfikatora wierzchołka (mNumber), wartość tego wierzchołka (mValue) i wektor identyfikatorów wierzchołków połączonych z tym wierzchołkiem krawędzią (mNextVertices).

Algorytm został zaimplementowany jako jedna z metod klasy Graph, żeby ułatwić dostęp do prywatnych danych tej klasy.

Opis zmiennych i metod klas:

- **Klasa Graph:**
  - int mNumberOfEdges – liczba krawędzi w grafie
  - int mNumberOfVertices – liczba wierzchołków w grafie
  - vector <Vertex> mEdges – wektor wierzchołków (reprezentuje strukturę grafu)
  - void createEdge(int vertex1, int vertex2) – funkcja tworząca krawędź pomiędzy dwoma wierzchołkami o podanych identyfikatorach
  - void loadGraphFormFile(string vertexFile, string edgesFile) – funkcja wczytująca dane na temat struktury grafu z podanych plików
    - vertexFile – plik z informacjami o wierzchołkach i ich wartościach
    - edgesFile – plik z informacjami o krawędziach

- int valueOfthePath(vector<int> path) – funkcja zwracająca wartość ścieżki reprezentowanej jako wektor identyfikatorów wierzchołków tworzących daną ścieżkę
- int getValueOfVertex(int vertex) - funkcja zwracająca wartość wierzchołka o podanym identyfikatorze
- int getNumberOfNextVerticesForVertex(int vertex) – funkcja zwracająca liczbę wierzchołków połączonych z wierzchołkiem o podanym numerze
- vector<int> getNextVerticesForVertex(int vertex) – funkcja zwracająca wektor identyfikatorów wierzchołków połączonych krawędzią z wierzchołkiem o podanym identyfikatorze
- bool isVertexVisitedInPath(int vertex, vector<int> path) – funkcja określająca czy wierzchołek o podanym identyfikatorze znajduje się w podanej ścieżce
- Graph() – konstruktor domyślny
- Graph(string vertexFile, string edgesFile) – konstruktor tworzący graf przy użyciu funkcji loadGrapgFromFile
- vector<int> findPath(int src, int dst, int value) – funkcja zwracająca ścieżkę od wierzchołka o numerze src do wierzchołka o numerze dst o zadanej wartości value w postaci wektora identyfikatorów kolejnych wierzchołków
- void show() – funkcja wyświetlająca inforamcje o grafie (wierzchołki z ich wartościami i identyfikatorami wierzchołków, z którymi każdy wierzchołek jest połączony)

- **Klasa Vertex:**
  - int mNumber – numer (identyfikator) wierzchołka
  - int mValue – wartość wierzchołka
  - vector<int> mNextVertices – wektor identyfikatorów wierzchołków połączonych z danym wierzchołkiem
  - Vertex(int number, int value) – konstruktor tworzący obiekt klasy Vertex o podanym numerze wierzchołka (numer) i wartości (value)
  - Vertex(int number, int value, vector<int> nextVertices) – konstruktor tworzący obiekt klasy Vertex o podanym numerze wierzchołka (numer), wartości (value) i wektorze identyfikatorów wierzchołków połączonych krawędzią z danym wierzchołkiem (nextVertices)
  - void setNextVertices(vector<int> nextVertices) – seter dla zmiennej mNextVertices
  - void addNextVertex(int nextVertex) – funkcja dodająca identyfikator kolejnego wierzchołka, z którym dany wierzchołek jest połączony krawędzią, do wektora mNextVertices
  - int getNumber() – funkcja zwracająca numer danego wierzchołka
  - int getValue() – funkcja zwracająca wartość danego wierzchołka
  - vector<int> getNextVertices() – funkcja zwracająca wektor identyfikatorów wierzchołków połączonych krawędzią z danym wierzchołkiem
  - int getNumberOfNextVerices() – funkcja zwracająca liczbę wierzchołków, z którymi dany wierzchołek połączony jest krawędzią (rozmiar wektora mNextVertices)
  - void show() – funkcja wyświetlająca dane danego wierzchołka (numer, wartość i listę identyfikatorów z którymi połączony jest krawędzią)

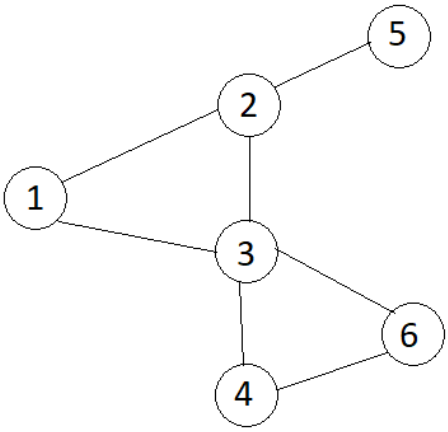
Zastosowany algorytm:

- **Założenia:**
  - Rozpatrywany graf jest spójny
  - Krawędzie są nieskierowane
  - Wartości wierzchołków są nieujemnymi liczbami całkowitymi
  - Graf nie rozpatruje ścieżek zawierających cykle (w szczególności żadnej wierzchołek nie może być odwiedzony więcej niż raz)
  - Algorytm zwraca pierwszą znalezioną ścieżkę spełniająca zadane warunki wejściowe
  - Algorytm zwraca pustą ścieżkę jeśli nie istnieje ścieżka spełniająca zadane warunki wejściowe
- **Opis:**
  1. Zastosowany algorytm działa w oparciu o algorytm „Przeszukiwania w głąb” (DFS).
  2. Złożoność czasowa: $O(n^2)$ (n -liczba wierzchołków w grafie)
  3. Złożoność pamięciowa: $O(e^2)$ (e – liczba krawędzi w grafie)

- **Kod algorytmu:**

```
vector<int> Graph::findPath(int src, int dst, int value)
{
    queue<vector<int>> pathsToAnalize;

    vector<int> path;
    path.push_back(src);

    pathsToAnalize.push(path);

    while (!pathsToAnalize.empty())
    {
        path = pathsToAnalize.front();

        pathsToAnalize.pop();

        int last = path[path.size() - 1];

        if (last == dst)
        {
            if (valueOfthePath(path) == value)
            {
                return path;
            }
            else
            {
                continue;
            }
        }
        else
        {
            if (valueOfthePath(path) > value)
            {
                continue;
            }
        }

        for (int nextVertex : getNextVerticesForVertex(last))
        {
            if (!isVertexVisitedInPath(nextVertex, path))
            {
                vector<int> newPath(path);
                newPath.push_back(nextVertex);
                pathsToAnalize.push(newPath);
            }
        }
    }
    path.clear();

    return path;
}
```

Przykład działania algorytmu:



wartości wierzchołków:

1 – 1
2 – 2
3 – 3
4 – 4
5 – 5
6 – 6

Src = 1
Dst = 6
Z = 12

0.
Queue = [{1}]
Path = {1} dl = 1

1.
Queue = [{1, 2}, {1, 3}]
Path = {1} dl = 1

2.
Queue = [{1,3}, {1,2,5}, {1,2,3}]
Path = {1,2} dl = 3

3.
Queue = [{1,2,5}, {1,2,3}, {1,3,2}, {1,3,6}, {1,3,4}]
Path = {1,3} dl = 4

4.
Queue = [{1,2,3}, {1,3,2}, {1,3,6}, {1,3,4}]
Path = {1,2,5} dl = 8

5.
Queue = [{1,3,2}, {1,3,6}, {1,3,4}, {1,2,3,6}, {1,2,3,4}]
Path = {1,2,3} dl = 6

6.
Queue = [{1,3,6}, {1,3,4}, {1,2,3,6}, {1,2,3,4}, {1,3,2,5}]
Path = {1,3,2} dl = 6

7.
Queue = [{1,3,4}, {1,2,3,6}, {1,2,3,4}, {1,3,2,5}]
Path = {1,3,**6**} dl = **10** //za krótka

8.
Queue = [{1,2,3,6}, {1,2,3,4}, {1,3,2,5}, {1,3,4,6}]
Path = {1,3,4} dl = 8

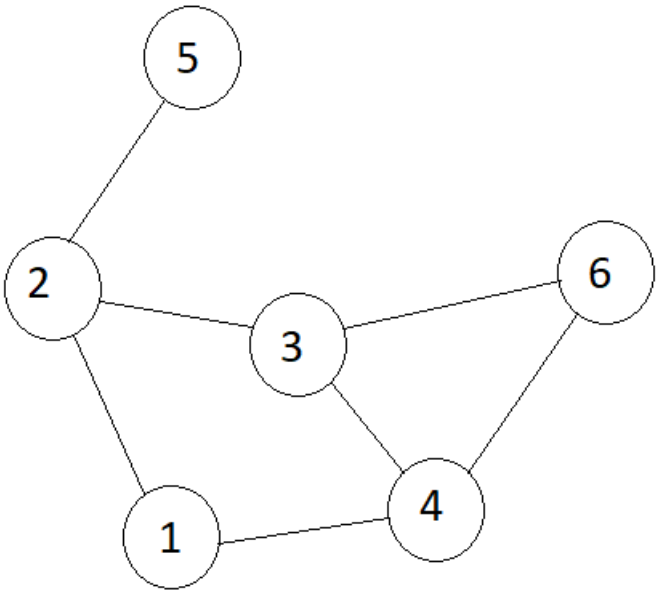
9.
Queue = [{1,2,3,4}, {1,3,2,5}, {1,3,4,6}]
Path = {1,2,3,**6**} dl = **12**
ZNALEZIONO SZUKANĄ ŚCIEŻKĘ

Wnioski:

- nie musimy rozpatrywać wszystkich ścieżek
- rozpatrujemy tylko ścieżki zaczynające się od wierzchołka startowego
- nie rozpatrujemy ścieżek które przekroczyły zadaną wartość (nie rozwijamy ich)
- algorytm DFS ma szerokie zastosowanie w rozwiązywaniu problemów, które da się reprezentować za pomocą grafu

Graficzna reprezentacja danych testowych:

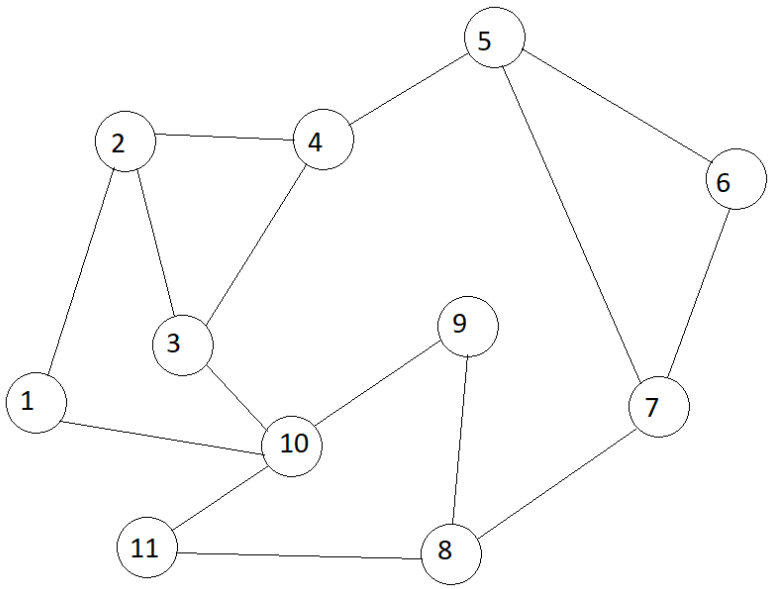
- vertices.txt + edges.txt



Wartości wierzchołków:

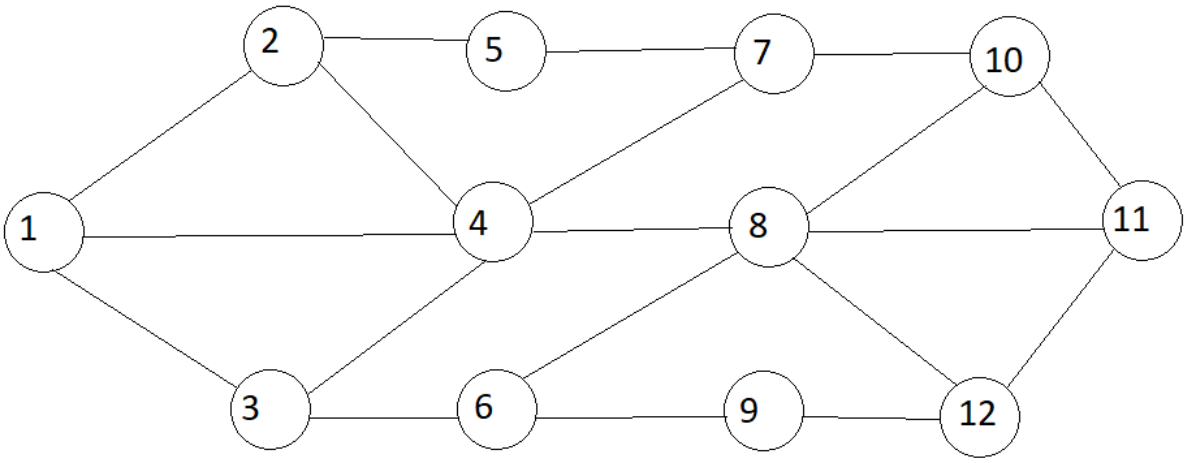
1 – 1
2 – 2
3 – 3
4 – 4
5 – 5
6 – 6

- vertices1.txt + edges1.txt



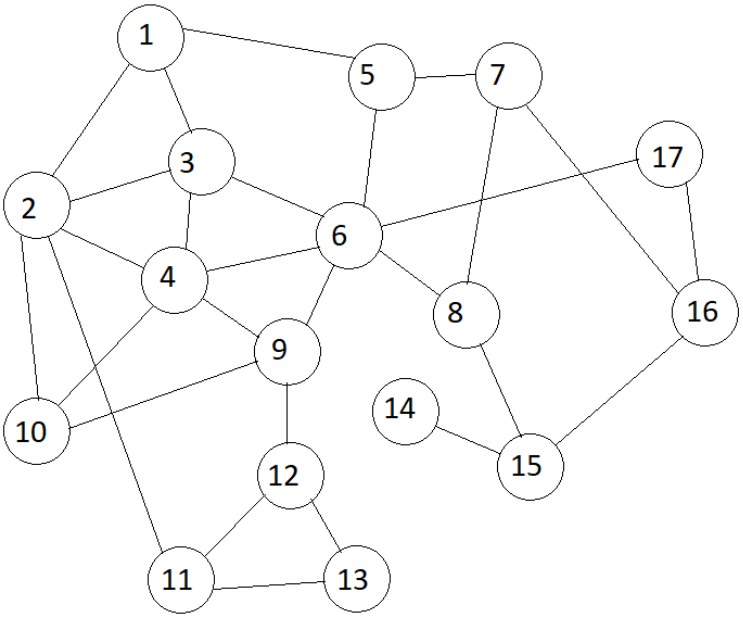
Wartości wierzchołków:
1 – 1
2 – 5
3 – 8
4 – 2
5 – 1
6 – 9
7 – 8
8 – 2
9 – 3
10 – 1
11 - 5

- vertices2.txt + edges2.txt



Wartości wierzchołków:
1 – 8
2 – 7
3 – 9
4 -10
5 – 6
6 – 2
7- 12
8 – 2
9 – 23
10 – 7
11 – 2
12 – 9

- vertices3.txt + edges3.txt



Wartości wierzchołków:
1 – 12
2 – 9
3 – 1
4 – 9
5 – 6
6 – 5
7 – 10
8 – 12
9 – 9
10 – 20
11 – 13
12 – 2
13 – 14
14 – 3
15 – 9
16 – 8
17 - 7

- **Edycja plików wejściowych:**
    - Kolejność podawanych wierzchołków i krawędzi nie ma znaczenia
    - W pliku z wierzchołkami w każdej linii podajemy <numer wierzchołka> <spacja> <wartość wierzchołka>
        - Np. 1 10
    - W pliku z krawędziami w każdej linii podajemy <numer wierzchołka> <spacja> <numer wierzchołka>
        - Np. 1 2
    - nie trzeba dublować krawędzi (wystarczy 2 3 nie trzeba podawać potem 3 2)
    - na końcu pliku warto zostawić jedną pustą linię