Jakub Szachwicz

Raport z rozwiązania problemu

Treść zadania:

Załóżmy że dany jest spójny graf nieskierowany. Każdy wierzchołek tego grafu posiada liczbę punktów otrzymywaną za jego odwiedzenie równą $s \in N$. 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 $z = \sum_{i}^{\square} \square s_{v_i}$, gdzie z jest i-tym wierzchołkiem odwiedzanym przez algorytm a $i \in [1,2,\ldots,N]$, gdzie Njest liczbą wierzchołków odwiedzonych w grafie.

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

Uściślenia i założenia projktowe:

- Ścieżkę rozumiemy jako ciąg par wierzchołków (krawędzi), które nie mogą się powtarzać
- Kolejne dwie krawędzie mają wspólny wierzchołek
- Wartość wierzchołka jest liczbą nieujemną

Propozycja rozwiązania:

Zmodyfikowany algorytm DFS (przeglądanie w głąb)- rekurencyjny

Modyfikacja polagała na tym, że algotytm otrzyma poza wierzchołkiem początkowym i końcowym pulę punktów do wykorzystania. Przy każdym odwiedzonym wierzchołku pula ta zmiejsza się o wartość odwiedzonego wierzchołka.

Warunki zakończenia algorytmu:

- Jeśli odwiedzony wierzchołek jest wierzchołkiem końcowym i pula punktów wyniesie zero.
- Jeśli pula punktów będzie liczbą mniejszą niż zero.
- Jeśli wierzchołek początkowy nie istnieje w grafie
- Jeśli wierzchołek końcowy nie istnieje w grafie
- Jeśli plik z grafem jest niepoprawny

W każdym innym przypadku algorytm będzie uruchamiany na każdej kolejnej krawędzi wychodzącej z wierzchołka rekurencyjnie.

Pseudokod (Python):

```
1. def DFS(vertex v, int z, vertex targetVertex, path path):
2.   z-= v.value
3.
4.   if (z == 0 && v is targetVertex): #stop clause for successful branch
5.     print path
6.     return
7.
8.   if (z < 0): #stop clause for non successful branch
9.     return
10.
11.  for u in v.neighbours:
12.    path.append(v) #add the current vertex to the path
13.    DFS(u, z, targetVertex, path) #recursively check all paths for of shorter de
   pth
```

Ostateczny wygląd kluczowego fragmentu program:

```
1. class Vertex:
2.     def __init__(self, id, value):
3.         self.id = id
4.         self.value = value
5.         self.neighbours = []
6.
7.
8. def find_path(graph, start, end, saldo, path=[]):
9.     path = path + [start]
10.    saldo = saldo - start.value
11.    if start == end and saldo == 0:
12.        return path
13.    if saldo < 0:
14.        return None
15.    for node in start.neighbours:
16.        if node not in path:
17.            newPath = find_path(graph, node, end, saldo, path)
18.            if newPath:
19.                return newPath
20.    return None
```

Linie 1.-5.:

Klasa Vertex jest klasą pomocniczą pozwalającą na przejrzysty i bezpieczne operowanie na wierzchołkach.

Każdy z wierzchołków posiada unikalne ID (w przypadku plików tesowych są to wielkie litery alfabetu łacińskiego), liczbę punktów reprezentowaną przez pole value, oraz tablicę sąsiedztwa.

Linia 8.:

Funkcja find_path to główna część algorytmu. Jej parametrami wywołania są kolejno

- graph-tablica wierzchołków w grafie,
- start- wierzchołek początkowy wykonania algorytmu (w treści zadania P)
- end- wierzchołek końcowy wykonania algorytmu (w treści zadania K)
- saldo- wartość, która pozostała nam do rozdysponowania na znalezienie ścieżki (w treści zadania z)
- path- dotychczas utworzona ścieżka powiększana przy kolejnych wywołaniach.

Linia 9.:

Do dotychaczas utworzonej ścieżki dodawany jest wierzchołek z którego rozpoczynamy wykonywanie algorytmu.

Linia 10.:

Zmniejszamy saldo o wartość wierzchołka w którym aktualnie się znajdujemy

Linie 11.-12.:

Jeśli jesteśmy w sytuacji, że odwiedzony wierzchołek jest szukanym wierzchołkiem końcowym i saldo punktów do wykorzystania jest w pełni skonsumowane kończymy wykonywanie algorytmu z sukcesem i zwracamy ścieżkę rozwiązującą problem.

Linie 13.-14.:

Saldo punktów zostało w pełni wykorzystane. Ścieżki nie znaleziono.

Linie 15.-20.:

Rekurencyjnie wywołuj funkcję dla każdego wierzchołka, który nie znalazł się w dotychczasowej ścieżce. Jeśli ścieżka została znaleziona- zwróć ją, jeśli nie- zwróć obiekt None.

Złożoność alorytmu:

- Złożoność czasowa: b^m
- Złożoność pamięciowa: $z=\sum_{i}^{n} \square s_{v_i}$

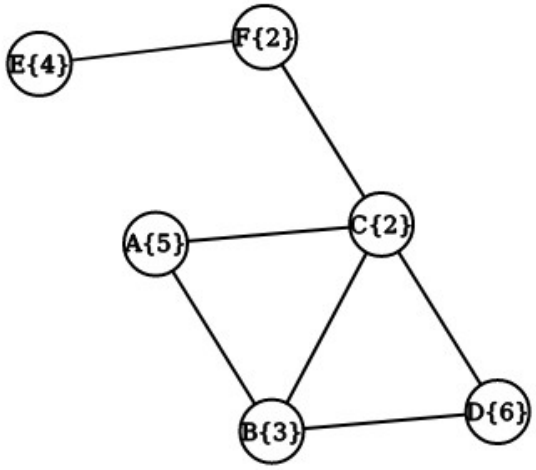
Gdzie: b- współczynnik rozgałęziania, m- maksymalna głębokość poszukiwań

Wykorzystywane technologie:

- Język programowania: Python w wersji 2.7

Przykłady grafów testowych:

Test_file_1:



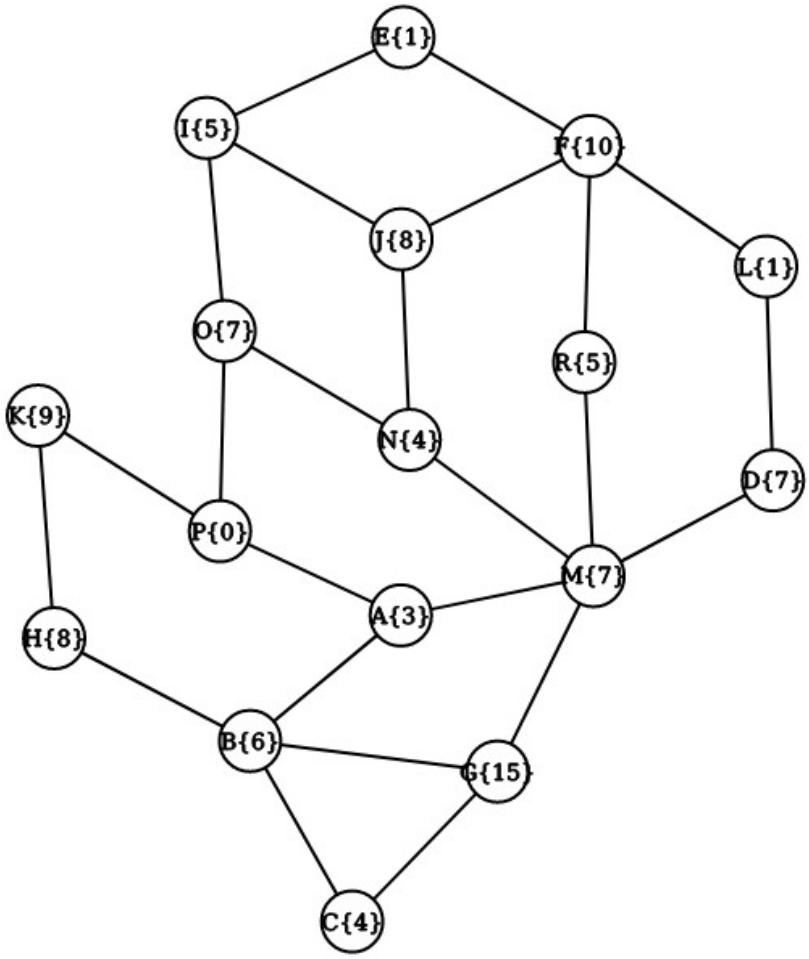
Szukamy ścieżki z wierzchołka P=A do K=C o sumie z=16

Przewidywany rezulatat: np. ścieżka A>B>D>C

Wynik działania algorytmu:

A(5) -> B(3) -> D(6) -> C(2)

test_file_2:



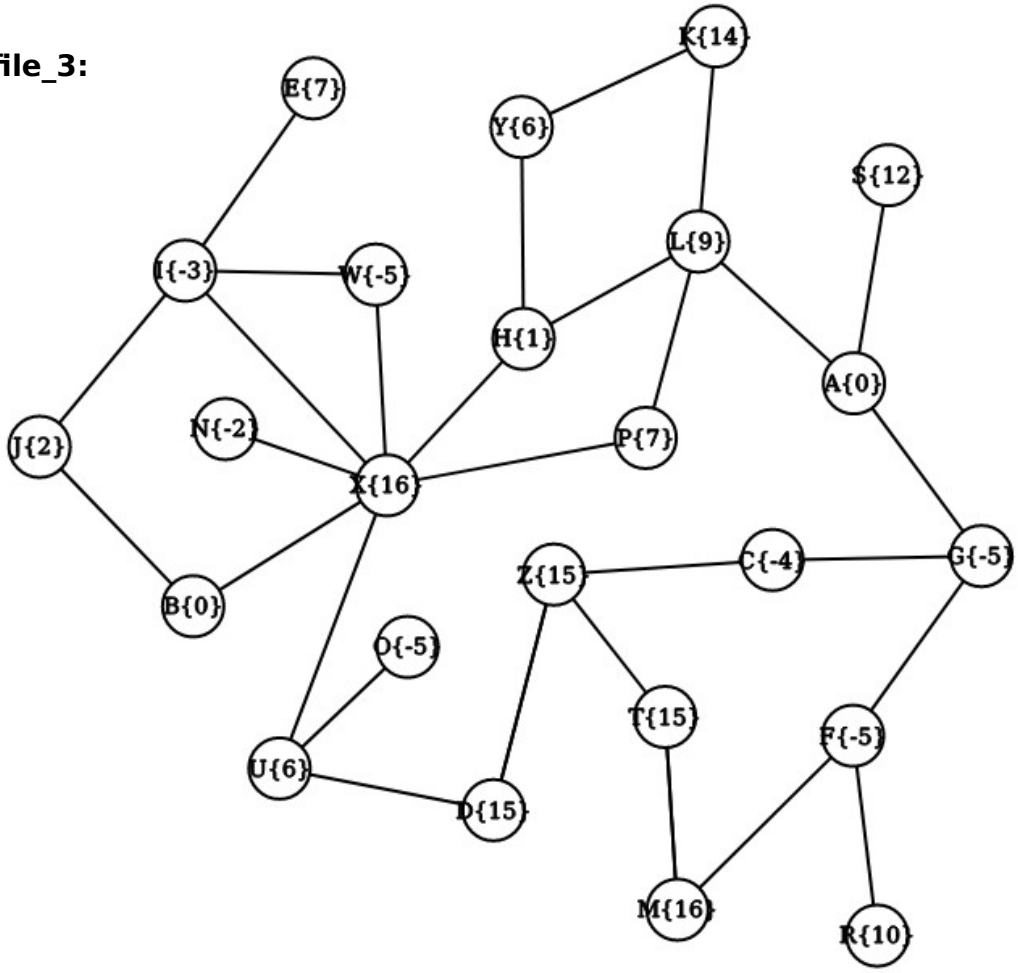
Szukamy ścieżki z wierzchołka P=A do K=E o sumie z=25

Przewidywany rezulatat: brak ścieżki

Wynik działania algorytmu:

Path doesn’t exist

test_file_3:



Szukamy ścieżki z wierzchołka P=N do K=C o sumie z=57

Przewidywany rezulatat: np. ścieżka N>X>H>Y>K>L>A>G>C

Wynik działania algorytmu:

N(2) -> X(16) -> H(1) -> Y(6) -> K(14) -> L(9) -> A(0) -> G(5) -> C(4)

Wnioski i problemy implementacyjne:

Podstawowym problemem, który pojawił się już w początkowej fazie rozwiązywania problemu, to brak jednoznacznych definicji matematycznych czym jest (a raczej czym nie jest) ścieżka w grafie. Publikacje naukowe, do których sięgałem podawały rozbieżne i wzajemnie wykluczające się) definicje:

Wikipedia (podając za źródło informacji " Thomas H. Cormen:

Wprowadzenie do algorytmów. Warszawa: Wydawnictwa

Naukowo-Techniczne, 2001, s. 114") twierdzi, że "ŚCIEŻKA PROSTA –

*ścieżka, w której **nie ma** powtarzających się wierzchołków"*

źródło: https://pl.wikipedia.org/wiki/Ścieżka_(teoria_grafów)

Z tego wynika, że w zwykłej ścieżce wierzchołki już powtarzać

się mogą, ale...

Dr hab. Łukasz Kowalik twierdzi w swojej pracy:

"Zauważmy, że w marszrucie te same wierzchołki mogą pojawiać się

*wiele razy. Jeśli wierzchołki **są parami różne**, to taka marszruta*

odpowiada ścieżce."

źródło: https://www.mimuw.edu.pl/~kowalik/papers/phd.pdf s.18

Przyjęte założenia zostały opisane we wcześniejszych punktach raportu.

Po przyjęciu założeń i konsultacjach z prowadzącym projekt najodpowiedniszym rozwiązaniem okazało się zmodyfikowanie algorytmu przeszukiwania w głąb.

Problem implementacyjny jaki się pojawił, polegał na spójnej reprezentacji grafu w pliku .txt.

Analiza wymagań oraz sam sposób działania algorytmu doprowadził mnie do poniższej konwencji:

<Nazwa wierzchołka> <Wartość> <lista sąsiadów>