

|  |  |
| --- | --- |
|  | **AKADEMIA GÓRNICZO-HUTNICZA** |

Raport i instrukcja z projektu

z przedmiotu

**Narzędzia komputerowe w rozwiązywaniu wybranych zagadnień matematyki wyższej i optymalizacji**

Elektronika i Telekomunikacja - Systemy Wbudowane, rok I studiów magisterskich

*Jakub Stelmach*

*Dominik Tendera*

*Jakub Ziobrowski*

14.06.2024r

Spis treści

[1. Temat projektu 3](#_Toc200835094)

[2. Wybrany algorytm 3](#_Toc200835095)

[3. Instrukcja 3](#_Toc200835096)

[4. Wnioski 4](#_Toc200835097)

# Temat projektu

Projekt nr 15

Niestety znów wykryto bazę Obcych na Ziemi! Jej kolejne komory tworzą graf z jednym tylko wejściem z zewnątrz. Niestety wszelkie metody zawiodły, pozostała już tylko jedna. Wybrany ochotnik (dlaczego to znów Ty?) musi wbiec do bazy i przebiec tyle komór ile zdoła w każdej kolejno mijanej porzucając bombę zegarową (w każdej może być tylko raz). Dostałaś mapę gniazda i znasz odległości czasowe pomiędzy komorami. Napisz algorytm, który ustali największą liczbę komór jakie zdołasz przebiegnąć w czasie T

# Wybrany algorytm

W celu optymalnego znalezienia trasy z jak największą ilością wierzchołków przy zadanym koszcie, zdecydowano się na wykorzystanie zmodyfikowanego algorytmu DFS (przeszukiwanie w głąb), śledząc przy tym dotychczas odwiedzone wierzchołki i poświęcony czas, w momencie w którym dojdziemy do końca drogi lub przekroczymy czas algorytm zapisuje daną ścieżkę i jej odpowiedni koszt, następnie cofając się do wierzchołka z którego jest następną możliwa trasa. Zadanie jest powtarzane aż do wykorzystana wszystkich możliwości. W rozwiązaniu zaimplementowano również odcinanie gałęzi, gdy nawet w optymistycznym scenariuszu nie będzie można pobić dotychczasowego najlepszego wyniku. Na rys 1. przedstawiono graf pomocniczy podczas rozwiązywania problemu.

Obraz zawierający diagram, krąg, rysowanie, linia

Zawartość wygenerowana przez AI może być niepoprawna.

Rys 1. Graf pomocniczy przy rozwiązywaniu problemu

# Implementacja

Skrypt został napisany w języku Python ze względu na prostą obsługę plików, początkowo otwiera on przygotowany opis grafu który następnie zapisuje w postaci

Funkcja find\_best\_path(filename: str) odpowiada za znalezienie optymalnej ścieżki w grafie na podstawie macierzy kosztów wczytanej z pliku wejściowego, z uwzględnieniem określonego limitu czasu. Na początku funkcja odczytuje dane z pliku, w tym limit czasu, numer wierzchołka początkowego oraz macierz kosztów, gdzie każdy wiersz odpowiada jednemu wierzchołkowi, a każda komórka zawiera koszt przejścia do innego wierzchołka. Na podstawie tej macierzy tworzona jest lista sąsiedztwa graph, w której każdemu wierzchołkowi przypisana jest lista krotek (sąsiad, koszt). Dzięki temu każdy wierzchołek może mieć wiele połączeń z innymi, z różnymi kosztami przejścia.

W celu znalezienia najlepszej ścieżki wykorzystywany jest algorytm przeszukiwania w głąb (DFS) z mechanizmem cofania (backtracking). Funkcja śledzi bieżącą ścieżkę w zmiennej path, odwiedzone wierzchołki w liście visited, oraz aktualizuje informacje o najlepszej dotychczasowej ścieżce (best\_path, best\_count i best\_cost). Dla każdego wierzchołka DFS sprawdza wszystkich sąsiadów. Jeśli sąsiad nie został jeszcze odwiedzony, a koszt dotarcia do niego nie przekracza limitu czasu, funkcja przechodzi do tego wierzchołka i kontynuuje rekurencję. Po zakończeniu eksploracji danego sąsiada następuje cofnięcie – usunięcie go z bieżącej ścieżki i oznaczenie jako nieodwiedzonego. Mechanizm backtrackingu jest kluczowy – umożliwia eksplorację wszystkich możliwych ścieżek bez powtarzania wierzchołków w jednej trasie.

Każde wywołanie DFS posiada własną pętlę for, która iteruje po sąsiadach bieżącego wierzchołka. Python zapamiętuje pozycję w tej pętli w trakcie rekurencji, więc po powrocie do wcześniejszego poziomu DFS, algorytm kontynuuje iterację od kolejnego sąsiada, zamiast zaczynać od początku. Podczas przeszukiwania, gdy zostanie odnaleziona ścieżka odwiedzająca więcej wierzchołków niż dotychczasowa najlepsza (lub taką samą liczbę przy niższym koszcie), informacje o najlepszej ścieżce są aktualizowane. Po zakończeniu działania algorytmu wypisywana jest najlepsza znaleziona ścieżka, liczba odwiedzonych wierzchołków oraz całkowity koszt czasowy.

Takie podejście pozwala na efektywne przeszukanie całego grafu z użyciem DFS, korzystając z listy sąsiedztwa i rekurencji. Dzięki ograniczeniu odwiedzin do unikalnych wierzchołków oraz ścisłej kontroli kosztu czasowego, algorytm gwarantuje poprawne i pełne znalezienie najdłuższej możliwej ścieżki w zadanym limicie.

# Instrukcja

Wersja Pythona jaką trzeba mieć zainstalowaną na komputerze to 3.8 i wyższe. W jednym folderze powinny się znaleźć plik „most\_nodes\_algorithm.exe”, który uruchomi nasz program oraz plik „graph\_values.txt” gdzie znajduje się opis naszego grafu. Format pliku .txt to:

Definicja limitu czasu, miejsce z którego zaczynamy trasę oraz tablica dwuwymiarowa, gdzie kolumny i wiersze to wierzchołki, a wartości to czas przejścia między poszczególnymi punktami. Poniżej przykładowa struktura pliku:

Defined Time: 100

Which node is the start: 1

1 2 3 4 5 6 7 8 9 10 11

1 0 34 78 0 65 12 0 45 0 19 83

2 56 0 0 23 0 88 14 0 90 0 47

3 0 42 0 51 77 0 0 0 29 65 0

4 73 0 0 0 38 0 91 27 0 33 55

5 0 19 0 0 0 60 79 0 0 0 0

6 21 0 0 0 18 0 53 92 0 0 41

7 0 38 0 0 0 0 0 0 0 96 0

8 44 0 85 0 0 0 0 0 22 0 70

9 0 0 0 0 0 33 0 0 0 0 0

10 0 71 60 0 0 0 0 93 48 0 17

11 50 0 0 84 0 0 65 0 0 0 0

Dla zobrazowania, czas który mamy do dyspozycji to 100 jednostek, zaczynamy od punktu [1, 1], gdzie dostanie się do punktu numer 3 z punktu o numerze 2 zajmie nam 42 jednostki czasu. Jako rezultat otrzymamy w oknie konsoli tak jak przedstawiono to na Rys 2. znalezioną trasę zawierającą najwięcej odwiedzonych wierzchołków oraz wykorzystany czas:

Obraz zawierający tekst, Czcionka, zrzut ekranu, czarne

Zawartość wygenerowana przez AI może być niepoprawna.

*Rys 2. Rezultat wykonania programu*

# Wnioski

Realizacja projektu pozwoliła pogłębić i zrozumieć teorię grafów oraz jej wykorzystanie w codziennych warunkach lub opisu nawet abstrakcyjnych sytuacji. Zauważono, że algorytm znalezienia najbardziej optymalnej ścieżki jest bardzo wymagający obliczeniowo przy wzroście wierzchołków i możliwych przejść oraz znacząco wzrasta poziom skomplikowania problemu.