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

Projekt II – temat 15.

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

**Najbardziej sensowny** -> wykorzystanie DFS (przeszukiwanie w głąb)

**Dlaczego?**

DFS (przeszukiwanie w głąb)

– Podstawowy szkielet rozwiązania: rekurencyjnie zagłębiasz się w graf, śledząc odwiedzone wierzchołki i zużyty czas. To właśnie to podejście pozwala wygenerować wszystkie możliwe proste ścieżki.

– Branch & Bound: podczas DFS możesz szybko odciąć gałęzie, gdy nawet w optymistycznym scenariuszu (wykorzystanie najkrótszych krawędzi) nie pobijesz dotychczasowego najlepszego wyniku.

BFS (przeszukiwanie wszerz)

– Nadaje się do znajdowania najkrótszej ścieżki w grafie jednowagowym lub nieważonym, albo warstwowego przeszukiwania.

– W naszym problemie jednak ważą czasy na krawędziach oraz liczy się maksymalna długość trasy (#wierzchołków), a nie minimalizacja czasu. BFS nie potrafi „przechwycić” tego typu ograniczenia budżetowego, więc nie jest tu pomocny.

Bellman–Ford

– Znajduje najkrótsze ścieżki w grafie z również ujemnymi wagami (przy braku ujemnych cykli).

– W zadaniu mamy tylko dodatnie czasy i nie potrzebujemy obsługi ujemnych odległości, ani eksploracji wszystkich ścieżek najkrótszych — więc Bellman–Ford się nie przyda.

Dijkstra

– Świetny do wyznaczania najkrótszych ścieżek od jednego źródła, gdy wagi są nieujemne.

– Może być użyty jako preprocessing:

Aby obliczyć odległość najkrótszą (najmniejszy czas) między dowolnymi dwoma komorami,

Albo by znaleźć w grafie minimalną wagę krawędzi w\_min dla heurystyki branch & bound.

– Jednak sam Dijkstra nie rozwiąże problemu maksymalizacji liczby odwiedzonych wierzchołków z budżetem — w tym celu wciąż potrzebny jest DFS z odcięciami.

Prim / Kruskal

– Algorytmy do budowy minimalnego drzewa rozpinającego (MST) — łączą wszystkie wierzchołki przy minimalnej sumie wag.

– Nas tu nie interesuje łączenie wszystkich komór w jedno drzewo o minimalnym koszcie, tylko znalezienie jednej ścieżki maksymalizującej liczbę odwiedzin przy limicie czasu.

– MST nie pomaga w orienteering.

1. **Nasza interpretacja** -> przeszukiwanie ścieżek i cofanie się w celu odkrycia nowych, pokazanie grafu i wyjaśnienie naszej interpretacji,
2. **Odcinanie ścieżek** gdy już przekroczymy limit czasowy,
3. **Stos** jako implementacja zapisu odwiedzonych punktów, ma na celu pomóc się cofać,

Tak jak myśleliśmy, trzeba zrobić tablize dwu wymiarową gdzie kolumny i wiersze to wierzchołki a wartości to czas przejścia, czyli kolumna 3 i wiersz 4 o wartości 10 to czas przejścia 10 z 3 do 4 wierzchołka

Napisz kod który odczytuje zadany czas maksymalny, oraz dane z tej tablicy:  
algorytm ma znaleźć drogę która pozwoli na odwiedzenie największej liczby wierzchołków w zadanym czasie. Do tego celu wejdź w głąb metodą DFS w losową ściężkę i zapisuj miejsca które odwiedziłeś. Gdy dojdziesz do końca lub osiągniesz limit czasu zapisz daną ścieżkę razem z jej kosztem a następnie cofnij się o jeden węzeł i przeszukaj następne aż do wykorzystania wszystkich możliwości. Ważne jest aby cofać się o jeden węzeł a następnie przechodzić wszytkie możliwe drogi z niego, następnie znowu się cofać i tak dalej