**Wstęp**

Sprawozdanie dotyczy algorytmów z powracaniem, które mają na celu znaleźć rozwiązania problemów obliczeniowych. Algorytm stopniowo generuje potencjalnych kandydatów, lecz gdy stwierdzi, że znaleziony kandydat nie spełnia wszystkich wymagań, powraca do punktu, w którym może znaleźć innego kandydata i zmienia budowę rozwiązania. Przykładem praktycznym wykorzystania takiego algorytmu jest odnajdywanie drogi w labiryncie, gdy natrafimy na ślepą uliczkę, wracamy się i próbujemy innej drogi. W języku Python zaimplementowano algorytmy znajdowania cyklu Hamiltona (algorytm Robertsa-Floresa) i cyklu Eulera w grafie nieskierowanym. Dla każdego z nich dokonano pomiarów czasu działania tych algorytmów i sporządzono wykresy t(n), gdzie n to liczba wierzchołków. Wykorzystano grafy o nasyceniach 30% i 70%.

**Reprezentacja**

Uznaliśmy, że najlepszym wyborem będzie lista następników, jest ona szybka i wygodna w obsłudze. Działa ona szybciej niż inne reprezentacje, ponieważ umożliwia szybki dostęp do krawędzi wychodzących z wierzchołka. Znalezienie następników danego wierzchołka w tej reprezentacji jest nieco szybsze niż dla macierzy sąsiedztwa i znacznie szybsze niż dla tabeli krawędzi.

**Cykl Hamiltona**

Algorytm ten rozpoczynamy z wybranego wierzchołka i metodą DFS wchodzimy w głąb grafu zaznaczając wierzchołki jako odwiedzone i dodając je do listy. W momencie kiedy dotrzemy do wierzchołka, od którego zaczynaliśmy, sprawdzamy czy wszystkie wierzchołki są już odwiedzone. Jeśli tak, znaczy to że droga, którą przeszliśmy to poszukiwany cykl Hamiltona, w przeciwnym razie cofamy się w poszukiwaniu alternatywnej drogi. Algorytm wykonujemy do momentu aż znajdziemy cykl Hamiltona. Jeśli sprawdzimy wszystkie możliwe trasy a nadal nie znaleźliśmy cyklu, oznacza to że graf takowego nie posiada. Pierwszy wykres pokazuje czas znajdywania cyklu dla grafów hamiltonowskich o nasyceniu 30% i 70%. Co ciekawe, to w grafie o większym nasyceniu algorytm szybciej znajdował cykl. Spowodowane jest to tym, że taki graf miał przeważnie więcej różnych cykli niż ten o mniejszym nasyceniu, więc łatwiej było na taki „trafić”, było więcej możliwości poruszania się po grafie i większe szanse szybkiego odnalezienia któregoś z cykli. Drugi wykres obrazuje czas znajdywania cyklu w grafach niehamiltonowskich o nasyceniu 50%, gdzie algorytm zwracał „False”. Musiał przejść wszystkie możliwe łańcuchy w grafie, przez co czas działania algorytmu był spory i zbadano maksymalnie 18-wierzchołkowy graf. Oba wykresy zarówno bez jak i w skali logarytmicznej. Złożoność działania algorytmu to O(n!) w najgorszym wypadku – grafie niehamiltonowskim, gdyż algorytm musi przejść wszystkie permutacje wierzchołków a tyle ich jest. W średnim przypadku jest to O(2n), dla grafu Hamiltonowskiego, gdy algorytm szybciej znajduje rozwiązanie.

**Cykl Eulera**

Algorytm wybiera wierzchołek, z którego będzie rozpoczynał pracę, następnie przechodzimy przez graf rekurencyjną metodą DFS. Przebyte krawędzie usuwamy, a wierzchołki po zakończeniu umieszczamy na stosie. Jeżeli algorytm znajdzie cykl Eulera, to po zakończeniu na stosie znajdą się kolejne wierzchołki tego cyklu. Wykres obrazuje czas działania algorytmu dla grafów o nasyceniu 30% i 70%. Algorytm działał szybciej dla nasycenia 30%, gdyż graf taki posiada mniej krawędzi – cykl Eulera jest zauważalnie krótszy. Wykres zarówno bez jak i w skali logarytmicznej. Złożoność działania algorytmu to O(E), gdzie E to liczba krawędzi w grafie. Wynika ona z tego, że każdą krawędź przechodzimy dokładnie jeden raz.