Laboratoria Algorytmów i Struktur Danych

Projekt 4

Grafy
Algorytmy z powracaniem

Wojciech Niedziela 160363

Jan Chojnacki XXXXXX

Godzina: **11:45** Dzień: **Piątek** Grupa: **14**

Nazwa kierunku: **Informatyka**

POLITECHNIKA POZNAŃSKA

1 | Wstęp

Celem tego sprawozdania jest zaprezentowanie operacji na grafach z algorytmami powracania, w szczególności algorytmów znajdowania cyklu Eulera i Hamiltona.

2 | Tworzenie grafu

Program obsługuje dwa tryby wprowadzenia grafu:

2.1 | -Hamilton

Opcja te generuje spójny graf nieskierowany o n wierzchołkach i podanym nasyceniu grafu. Graf jest reprezentowany jako lista sąsiedztwa. Współczynnik nasycenia jest wybierany przez użytkownika i wynosi odpowiednio 30% lub 70%. Na początku tworzony jest cykl Hamiltona, a następnie graf jest dopełniany krawędziami wg. współczynnika nasycenia. Aby zachować parzysty stopień każdego wierzchołka dodawane są krótkie cykle złożone z 3 wierzchołków.

```python
def create_hamilton_graph(self, saturation):
    cycle = self.nodes[:]
    random.shuffle(cycle)

    for i in range(len(cycle)):
        cycle[i-1].add_neighbor(cycle[i % len(cycle)])
        cycle[i % len(cycle)].add_neighbor(cycle[i-1])

    edges = len(cycle) * (len(cycle) - 1) // 2
    additional_edges = int(edges * saturation / 100) - len(cycle)
    additional_edges_needed = additional_edges

    while additional_edges_needed > 0:
        NodeA, NodeB, NodeC = random.sample(cycle, 3)
        if (NodeB not in NodeA.neighbors and NodeC not in NodeA.neighbors and
            NodeA not in NodeB.neighbors and NodeC not in NodeB.neighbors and
            NodeA not in NodeC.neighbors and NodeB not in NodeC.neighbors):
            NodeA.add_neighbor(NodeB)
            NodeB.add_neighbor(NodeA)
            NodeB.add_neighbor(NodeC)
            NodeC.add_neighbor(NodeB)
            NodeC.add_neighbor(NodeA)
            NodeA.add_neighbor(NodeC)
            additional_edges_needed -= 3
```

2.2 | -Nie-Hamilton

TODO - JANEK

3 | Operacje na grafach

3.1 | Wypisanie grafu

Program umożliwia wypisanie grafu w wybranej przez nas reprezentacji - lista sąsiedztwa.

3.2 | Znajdowanie cyklu Eulera

Program implementuje algorytm do znajdowania cyklu Eulera w grafie. Algorytm do znajdowania cyklu Eulera wykorzystany w naszym programie opiera się na algorytmie przeszukiwania grafu w głąb (DFS). Szukanie cyklu rozpoczynamy od wybrania wierzchołka startowego z niezerowym stopniem, a następnie wybieramy jedną z jego krawędzi, przechodzimy do kolejnego wierzchołka i usuwamy krawędź. Jeżeli wierzchołek nie ma nieodwiedzonych krawędzi, zdejmujemy go ze stosu i dodajemy do listy cyklu. Podaną strategię kontynuujemy do momentu, aż stos wierzchołków będzie pusty.

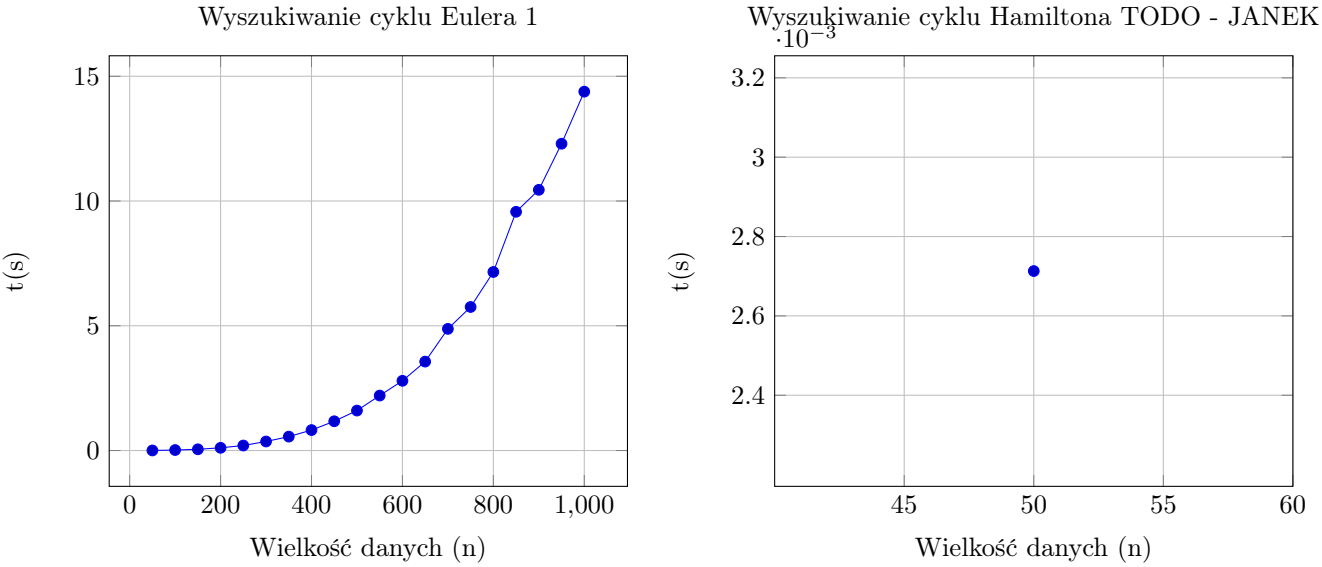
```
def DFS_Euler_iterative(self, start_node):
    eulerCycleResult = []
    visited_edges = set()
    stack = [start_node]

    while stack:
        vNode = stack[-1]
        unvisited = None
        for uNode in vNode.neighbors:
            if (vNode, uNode) not in visited_edges:
                unvisited = uNode
                break

        if unvisited is None:
            eulerCycleResult.append(stack.pop().val)
        else:
            visited_edges.add((vNode, unvisited))
            visited_edges.add((unvisited, vNode))
            self.remove_edge(vNode, unvisited)
            stack.append(unvisited)

    eulerCycleResult.reverse()
    return eulerCycleResult
```

3.3 | Algorytm znajdowania cyklu Hamiltona

TODO - JANEK

4 | wykresy

4.1 | Grafy Hamiltonowskie



Rysunek 1: Porównanie czasów wyszukiwania cykli Hamiltona i Eulera

4.2 | Grafy Nie-hamiltonowskie

TODO - JANEK