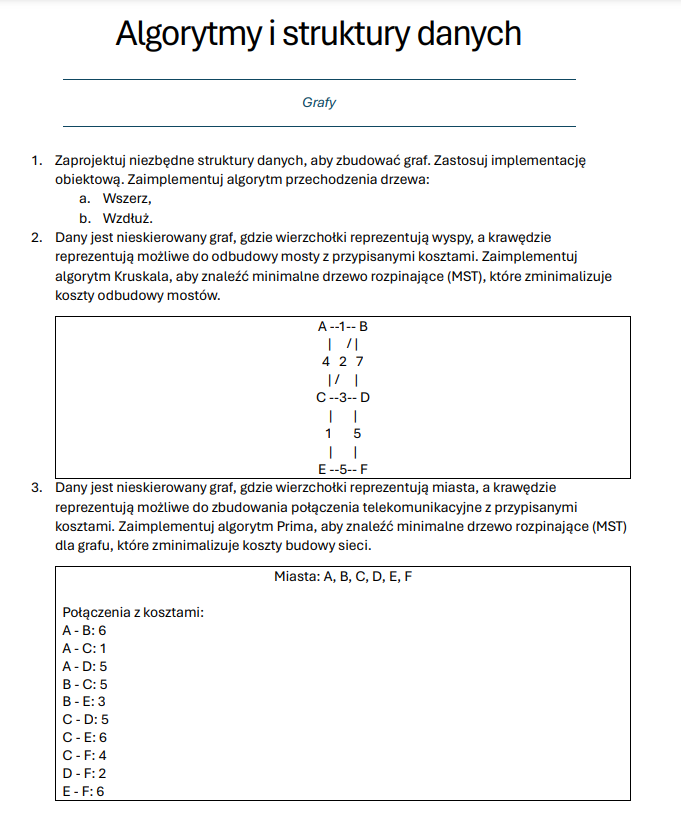
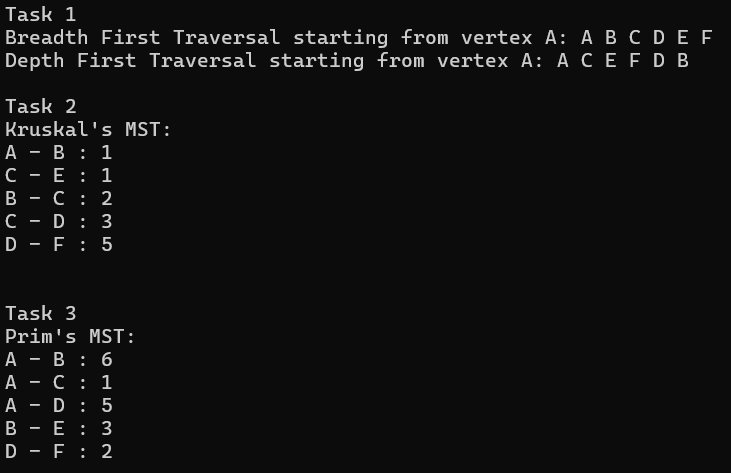
# Sprawozdanie, Zajęcia nr 5 – Grafy.

## Krystian Kostrzewa, 418845. WIMiIP, Informatyka Techniczna, sn. Data zajęć: 25.05.2024

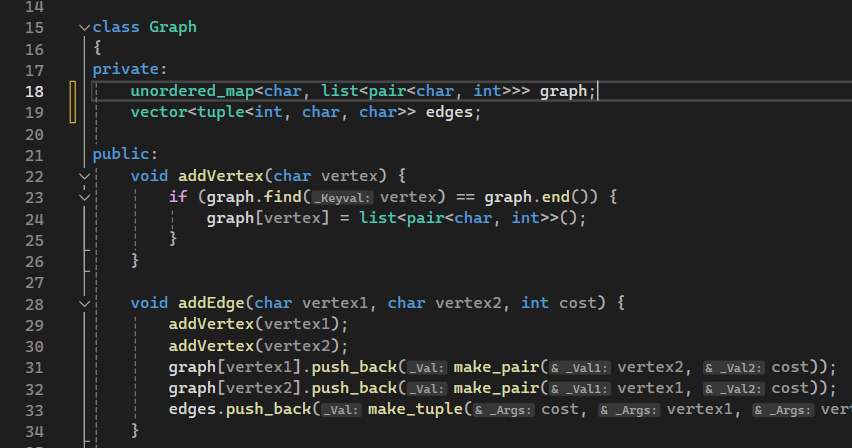
**Polecenia:**



**Wynik w konsoli:**

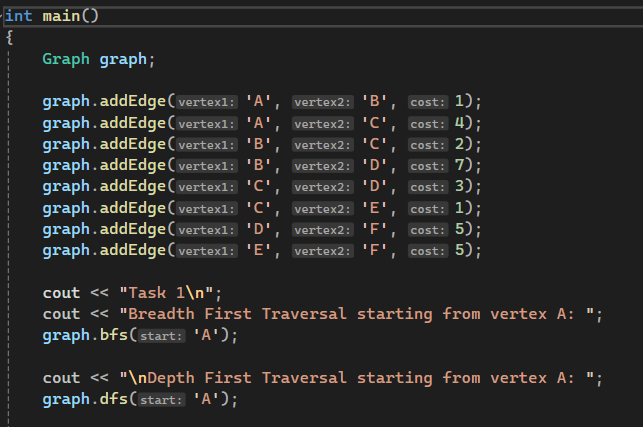


**Kod i opis rozwiązania:**



Bazę do wykonania poleceń stanowi obiekt klasy Graph. Przechowuje on początek i koniec możliwych krawędzi oraz wartość. Klasa posiada również metody *addVertex* – dodającą wierzchołek oraz *addEdge* – dodającą połączenie między wierzchołkami.

Rozwiązanie zadania 1:

****

Fragment kodu na początku funkcji *main* tworzy i uzupełnia graf, który zostanie wykorzystany do wykonania zadań 1 i 2. Potem następuje wywołanie funkcji przechodzących przez drzewa wzdłuż i wszerz.



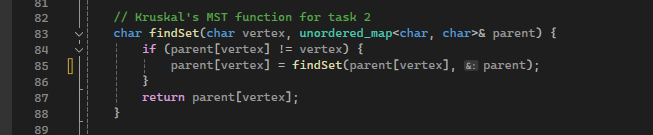
Funkcja *bfs* realizuje przeszukiwanie grafu wszerz. Na początku, tworzony jest zbiór *visited* do śledzenia odwiedzonych węzłów oraz kolejka *q* do przetwarzania węzłów. Funkcja rozpoczyna od dodania węzła startowego do kolejki. W pętli, funkcja pobiera pierwszy węzeł z kolejki, a jeśli nie był on jeszcze odwiedzony, wypisuje go na ekranie i oznacza jako odwiedzony. Następnie, dla każdego sąsiada tego węzła, sprawdza, czy sąsiad był odwiedzony. Jeśli nie, dodaje go do kolejki do przetworzenia w kolejnych iteracjach. Pętla działa do momentu, aż kolejka stanie się pusta, co oznacza, że wszystkie osiągalne węzły zostały odwiedzone i przetworzone. 

Funkcja *dfs* realizuje przeszukiwanie grafu wzdłuż . Na początku, tworzony jest zbiór *visited* do śledzenia odwiedzonych węzłów oraz stos *s* do przetwarzania węzłów. Funkcja rozpoczyna od dodania węzła startowego na stos. W pętli, funkcja pobiera węzeł ze szczytu stosu, a jeśli nie był on jeszcze odwiedzony, wypisuje go na ekranie i oznacza jako odwiedzony. Następnie, dla każdego sąsiada tego węzła, sprawdza, czy sąsiad był odwiedzony. Jeśli nie, dodaje go na stos do przetworzenia w kolejnych iteracjach. Pętla działa do momentu, aż stos stanie się pusty, co oznacza, że wszystkie osiągalne węzły zostały odwiedzone i przetworzone.

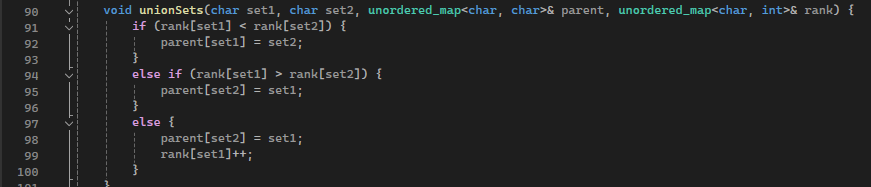
Rozwiązanie zadania 2:



W funkcji main odbywa się proste wywołanie funkcji.



Funkcja *findSet* znajduje reprezentanta (korzeń) zbioru, do którego należy dany węzeł *vertex*. Działa to w sposób rekurencyjny: jeśli rodzicem węzła nie jest on sam, funkcja wywołuje się rekurencyjnie, aż dotrze do korzenia. Aby przyspieszyć przyszłe operacje, stosuje optymalizację zwaną kompresją ścieżki– ustawia rodzica wszystkich węzłów na ścieżce do korzenia bezpośrednio na korzeń.



Funkcja *unionSets* łączy dwa zbiory *set1* i *set2*. Sprawdza ich rankingi, aby zminimalizować wysokość drzewa reprezentującego zbiór:

- Jeśli *set1* ma niższy ranking, staje się podzbiorem *set2*.

- Jeśli *set1* ma wyższy ranking, *set2* staje się podzbiorem *set1.*

- Jeśli rankingi są równe, *set2* jest dołączany do *set1*, a ranking *set1*  jest zwiększany o 1.



Funkcja *kruskalMST* implementuje algorytm Kruskala do znajdowania minimalnego drzewa rozpinającego . Działa w następujących krokach:

1. Inicjuje mapy *paren*t i *rank* dla wszystkich węzłów, ustawiając każdy węzeł jako swój własny rodzic i ustalając rankingi na 0.

2. Sortuje krawędzie grafu według kosztów.

3. Iteruje przez każdą krawędź:

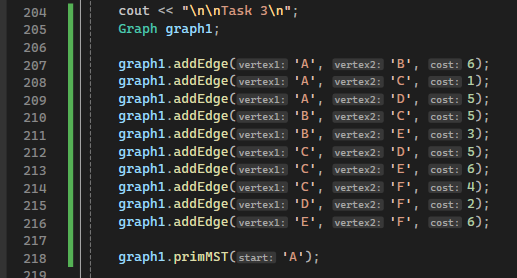
- Pobiera koszt oraz węzły *vertex1* i *vertex2*.

- Znajduje zestawy (*set1* i *set2*) do których należą węzły za pomocą *findSet*.

- Jeśli zestawy są różne, dodaje krawędź do *mst* i łączy zestawy za pomocą *unionSet*`.

4. Na końcu wypisuje krawędzie tworzące MST.

Rozwiązanie zadania 3:



W funkcji *main* następuje uzupełnienie nowego grafu odpowiadającego temu z polecenia, a następnie wywołanie funkcji *primMST*.



Funkcja *primMST* znajduje minimalne drzewo rozpinające w grafie, inicjalizując struktury *key*, *parent* oraz *inMST* i ustawia klucz startowego wierzchołka na 0. Priorytetowa kolejka (*pq*) przechowuje wierzchołki do przetworzenia, uporządkowane według wartości kluczy. W pętli, najpierw wybierany jest wierzchołek o najmniejszym kluczu, który dodawany jest do *inMST*. Dla każdego sąsiada tego wierzchołka, jeśli nie jest on w MST i krawędź do niego ma mniejszy koszt niż obecnie zapisany w *key*, to aktualizowane są *key*, *parent* oraz *pq*. Po zakończeniu pętli, MST jest wyświetlane poprzez wypisanie krawędzi i ich kosztów na podstawie *parent* i *key*.

Wnioski:

Przeprowadzone zadania umożliwiły zapoznanie się z różnymi metodami przeszukiwania grafów oraz implementacją algorytmów do znajdowania minimalnego drzewa rozpinającego. Funkcje BFS i DFS umożliwiają pełne przeszukiwanie grafu, natomiast funkcje *kruskalMST* i *primMST* poprawnie znajdują minimalne drzewa rozpinające, minimalizując koszty odbudowy mostów i budowy sieci.