# ALGORYTM DIJKSTRY

## Michał Ślusarczyk – opis klas

## Klasy:

**Klasy Główne:**

- **Parameters** – informacje zebrane od użytkownika oraz wywnioskowane z konsoli, nie posiada metod, jedynie argumenty.

Atrybuty:
- string paths[3] – ścieżki do plików,

- bool QGraph – czy podana ścieżka do grafu?
- bool Qgrapho - czy chcesz sam opisać graf?

- bool QIO - czy podana ścieżka do danych?
- bool QIOo - czy chcesz wpisać dane sam?

- bool Qsave - czy podana ścieżka do zapis?
- bool Qsaveg - czy chcesz zapisać graf?
  bool Qsaved - czy chcesz zapisać dane?
- bool Qsaves - czy chcesz zapisać rozwiązanie?

- bool Qshowg - czy wyświetlić graf?
- bool Qshowd - czy wyświetlić dane?
- bool Qshows - czy wyświetlić rozwiązanie?

- unsigned long int quantity - liczba wierzchołków w grafie,
-PTops* atops – lista wszystkich wierzchołków w grafie,

Metody:
-konstruktor(wypełnia wartościami domyślnymi)
-destruktor domyślny,

-**Management** – Główny klasa, za jego pomocą obsługiwany jest program oraz określane są kolejne kroki na podstawie dostarczonych argumentów oraz wiadomości od użytkownika. Posiada metody do obsługi programu korzysta z parametrów z klasy Parameters. Jest z nią wzajemnie zaprzyjaźniona

Atrybuty:
-Parameters* P - parametry,
-Graph* G – graf,
-LPTops* D - lista danych,
-Solution* S – rozwiązanie,

Metody:
-Konstruktor – uruchamia kolejne funkcje,
-Calibration – wypełnia Parameters,
-Ask – zadaje pytania użytkownikowi i uzupełnia Parameters,
-Load1 – wczytaj graf z pliku,
-CreateGraph - stwórz graf z użytkownikiem,
-Load2 - wczytaj dane z pliku,
-LoadData – pobierz dane od użytkownika,
-PrepareSolution - przygotuj S pod rozwiązanie,
-Dijkstra – wykonaj algorytm Dijkstry,
-FillSolution - uzupełnij rozwiązanie (możliwe że ta funkcjonalność zostanie połączona z Dijkstrą),
-Save - zapisz do pliku,
-Show - wyświetlanie rozwiązania w konsoli,
-Destruktor - zwalnia niepotrzebną pamięć i „sprząta program”,

**Klasy do obsługi pamięci dynamicznej (listy):**

-**Top** (wirtualny) – klasa wirtualna dla pierwotnego i sąsiedniego,

-**Primal** – dziedziczy po Top (zawiera numer danego wierzchołka),
Atrybuty:
-unsigned long int number – numer wierzchołka,

-**Borderer** - dziedziczy po Primal i Top (poszerzony o metrykę względem pierwotnego),

Atrybuty:
-unsigned double metrics – metryka czyli odległość,

-**PTops** - zawiera wskaźnik Primal (poszerzony o wskaźnik na kolejny element),

Atrybuty
-Primal* Ptop,
-PTops* pNext,

-**BTops** – zawiera wskaźnik Borderer (poszerzony o wskaźnik na kolejny element),

Atrybuty:
-Borderer* Btop,
-BTops* pNext,

-**LPTops –** zawiera wskaźnik PTops (poszerzony o wskaźnik na kolejny element),

Atrybuty:
-PTops* LPtop,
-LPTops* pNext;

-**Net** – zawiera wskaźniki Primal i Btops (poszerzony o wskaźnik na kolejny element),

Atrybuty:
-Primal* top,
-BTops* borderers,
-Net* pNext.

-**Graph** – zawiera wskaźnik Net (poszerzony o wskaźnik na samego siebie),

Atrybuty:
-Net* net,
-Graph* pNext,

-**Solution** – zawiera informacje o wierzchołku z którego wychodzi, do którego zmierza, pokonaną długość i listę przez która przechodzi (poszerzony o wskaźnik na samego siebie),

Atrybuty:
-Primal* start;
-Borderer* end;
-PTops* way;
-Solution* pNext,

Klasy do obsługi pamięci dynamicznej będą posiadały konstruktory i destruktory, metody do operowania pamięcią oraz przeciążone operatory do szybkiego zapisu i odczytu tworzone w miarę potrzeb wraz z postępem pisania programu.

Moje uwagi/pytania:
1. Nie wiem czy nie powinienem spróbować zredukować liczbę klas związanych z pamięcią dynamiczną, chociaż uważam że każda z nich ma zastosowanie oraz w tej ilości są bardziej przejrzyste.
2. Zastanawiam się również czy jeśli mogę to powinienem się starać się alokować pamięć dynamicznie, czy też korzystać z obiektów statycznych. Jaka jest zależność określająca kiedy jak postąpić? W powyższej rozpisce postanowiłem że jeśli jest możliwe to zaalokuje pamięć dynamicznie.
3. W opisie klas mogą zachodzić niewielkie zmiany wynikające z występujących potrzeb w trakcie pisania, ale ogólny charakter powinien się nie zmienić.