# Sprawozdanie z projektu w Pythonie

Bartosz Dziuba

1. Omówienie tematu: Tematem mojego projektu jest przechodzenie grafu skierowanego za pomocą algorytmu A*. Jest to klasyczny problem mający na celu znalezienie najkrótszej drogi do celu z punktu startowego zachowując optymalną złożoność obliczeniową.
2. Użyte biblioteki i ich cel:
    a. queue- biblioteka z której zaimportowałem strukturę danych priority queue, potrzebną do odpowiedniego przydziału kolejności przeszukiwania grafu.
    b. networkx- biblioteka służąca do wizualizacji grafu, potrzebna była do stworzenia graficznej reprezentacji wygenerowanego losowo grafu i zaznaczenia na nim najkrótszej ścieżki.
    c. matplotlib- wywołanie okienka na którym znajduje się graf i wynik działania programu.
    d. random- biblioteka odpowiedzialna za funkcje tworzącą liczby pseudolosowe wyznaczające pozycje nodeów grafu oraz wybierająca losowo start i koniec naszego programu.
    e. time- użyty wyłącznie w celu stworzenia ziarna do liczb pseudolosowych.

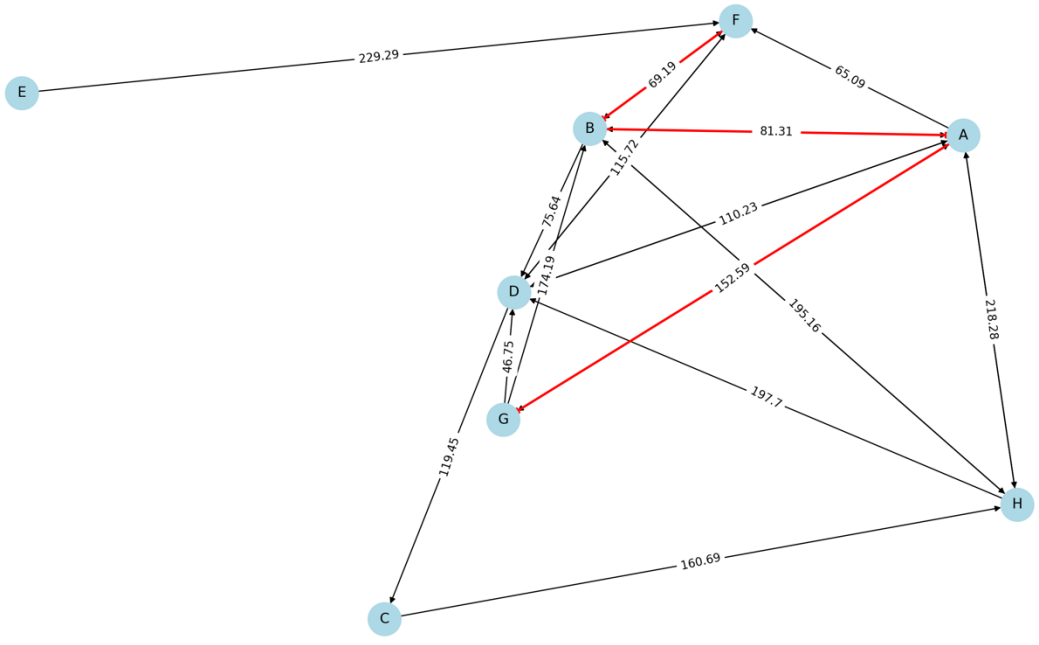
3. Struktury danych:
    - Node- węzeł naszego grafu, posiada nazwę-domyślnie jedną literę alfabetu łacińskiego, ale w programie w przypadku wywołania więcej niż 26 węzłów będą pojawiać się następne znaki ASCII. Jest porównywalna do innych węzłów pod względem pozycji , np. A(1,1)=B(1,1). Funkcja hashable musi być zaimplementowana na nazwę węzła, wynika to z nadpisania metody __eq__(self,other).
    - Graph- zawiera węzły- jako zbiór, połączenia między nimi oraz ich długości, reprezentowane jako słownik gdzie kluczem jest para uporządkowana węzłów od początku do końca połączenia, ich wartością jest długość między nimi. Graph zawiera także metodę get_neighbors(self, node), która zwraca listę wszystkich sąsiadów dla danego węzła.
    - AStarSolution- klasa zawierająca funkcję A_Star(self,heuristic). Klasa posiada atrybut start, end oraz graf.

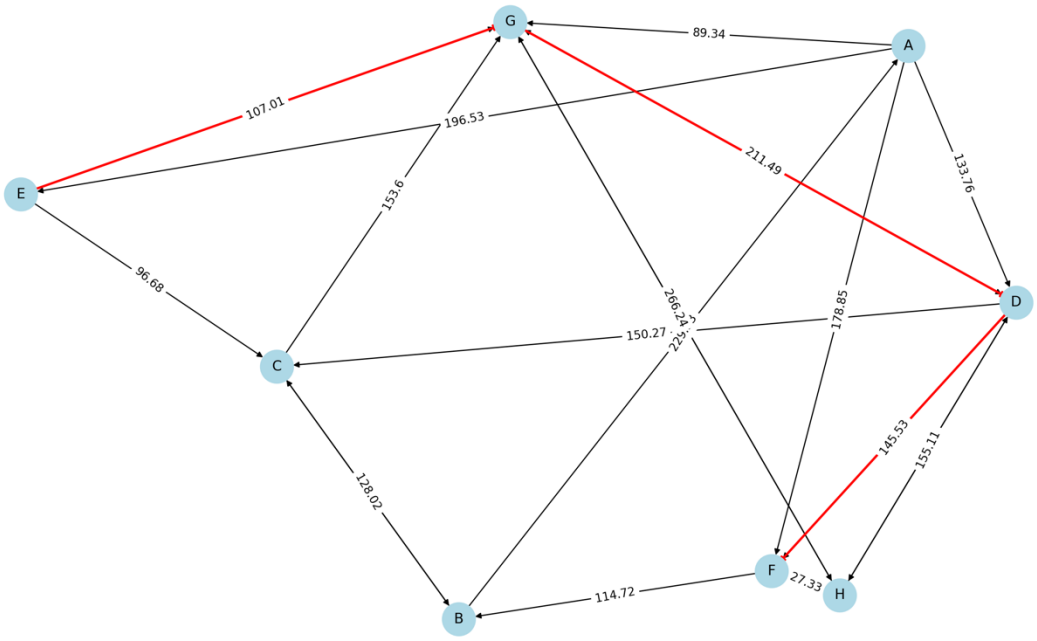
4. Działanie algorytmu: A* dla każdej iteracji pobiera sąsiadów węzła w którym się znajduje, naszym początkowym kosztem jest koszt dotarcia do obecnego miejsca, jest on już znany ale nie wystarczy do wybrania następnego kroku, więc pobiera odległość jaka jest między węzłem a sąsiadem, dodaje do niej „wagę”-stąd graf

ważony, czyli wynik funkcji heurystycznej między potencjalną przyszłą pozycją a celem i wkłada przejścia do kolejki priorytetowej. Następnie wybierana jest droga do „najtańszej” operacji, a pozycja zostaje zmieniona, tym samym następuje kolejna iteracja. Proces powtarzany jest do momentu dotarcia do celu. Lub gdy nie istnieje połączenie, tj. kolejka jest pusta.
Algorytm jest szczególnie efektywny właśnie ze względu na utylizacje dodatkowych informacji w postaci funkcji heurystycznej jak i bazowe- koszty dotarcia między węzłami.
Dla przykładu mojego programu użyłem zwyczajnej odległości Euklidowej między dwoma punktami w przestrzeni dwuwymiarowej, jako funkcji heurystycznej.

5. Działanie wizualizacji: Wybierając losowo punkty w przestrzeni dwuwymiarowej otrzymujemy zbiór naszych węzłów, między którym budujemy połączenia o koszcie który jest „zaburzoną” odległością między nimi, gdzie zaburzenie to dzielenie przez losowo wybraną liczbę od 0.4 do 0.9 w celu wydłużenia odległości połączenia na grafie. Współczynnik występowania połączeń jest liczbą w przedziale [0,1], gdyż możemy mieć graf pusty lub pełny, dla celów wizualizacji zainicjalizowałem go jako 0.63. W przypadku braku możliwości wyznaczenia ścieżki otrzymamy komunikat „Path does not exist”, stanie się tak gdy funkcja AStar zwróci None.

6. Przykładowa wizualizacja:






7. Podsumowanie: Projekt nauczył mnie działania prezentowanego przeze mnie algorytmu oraz mu pokrewnych, jak np. Dijkstra, poza tym miałem okazję popracować z wizualizacją grafów oraz tworzeniem abstrakcyjnych typów danych i manipulacji ich w algorytmie w celu znalezienia efektywnego sposobu na rozwiązanie problemu.