

Obliczanie najszybszej trasy

Oskar Bartosz / Jakub Robaczewski

Sposób przechowywania danych:

Aby zaimplementować algorytm Dijkstry, stworzyliśmy własne kontenery na dane, którymi są **Graph** i **Node**. Celem klasy **Node** jest przechowywanie danych o pojedynczym elemencie w grafie, takich jak np. jego prekursor. Klasa **Graph** jest klasą symulującą działanie grafu na tablicy dwuwymiarowej (na której podawane jest 36 cyfr). Tworzy ona dokładnie taką samą tablicę, jedynie zamienia typy danych z **int** na **Node** o tej samej wartości. Posiada ona między innymi metody takie jak **findNeighbour**, która zwraca **Node** wszystkich sąsiadujących elementów.

Implementacja algorytmu:

Po podaniu do programu wartości początkowych tworzona jest tablica 6x6 wartości typu **Node** (`self.distances`). Będzie ona wypełniać obowiązki tablicy odległości i tablicy prekursorów. Następnie jej wartości uzupełniane są nieskończonością z biblioteki `math` (poza zerem, czyli wartością startową). Tworzona jest także kolejka priorytetowa, wedle której będziemy badać odległości w tablicy (`self.basic_list`).

Podczas fazy obliczania odległości, przesuwamy się zgodnie z kolejką badając sąsiadów obecnego elementu. Teraz następuje sprawdzenie warunku dla każdego z sąsiadów, które można wyrazić słowami jako:

Jeśli dotychczasowa odległość do tego elementu plus wartość tego elementu jest mniejsza niż odległość obecnie przypisana do elementu, nadpisz nową odległość.

W tym wypadku nadpisywana jest odległość od początku, oraz do **Node** którego badaliśmy dodawany jest prekursor. Następnie brana jest kolejna wartość z kolejki priorytetowej, którą sortujemy po każdym sprawdzeniu jednego **Node**. Po przejściu przez całą listę, otrzymujemy tablicę dwuwymiarową wartości które przechowują informację o odległości, oraz o węźle którym można najszybciej dojść do początku tablicy.

Wyniki:

Stworzony został także generator tablic 6x6, aby skuteczniej testować program. Zamieszczam tutaj przykładowe wyniki programu.

```
254614 .....
948141 .....
174706 ....0.
547629 ....2.
214348 .1434.
604953 .0.....
```

```
663895 .....
604996 .04...
977835 ..78..
598783 ...7..
657109 ...10.
813913 .....
```

```
792162 .....
915410 .....0
646243 .....3
469741 ....41
730638 ..063.
525542 .....
```

```
111122 .....
104122 .04...
941111 ..1...
996411 ..6...
990411 ..0...
991111 .....
```

