AAL - Projekt

Andrzej Roguski

25 stycznia 2016

1 Treść zadania

Dany jest zbiór miast i dróg między nimi. Czas potrzebny na przebycie danej drogi zależy od pory dnia i zmienia się co 6 godzin, przy czym liczy się wyłącznie godzina wjazdu na drogę. Czas liczony jest z rozdzielczością godzinową.
Opracować algorytm wyznaczający godzinę wyjazdu oraz trasę pozwalające na jak najkrótsze przebycie drogi od miasta A do miasta B.

2 Struktury danych

Miasta i drogi w naturalny sposób mogą zostać przedstawione jako graf ważony skierowany. Odstępstwem od zwykłego grafu ważonego będzie konieczność uwzględnienia czterech wag dla każdej krawędzi, w celu odwzorowania zmienności kosztu przejścia krawędzi.

3 Analiza problemu

Problem opisany w zadaniu jest uogólnieniem zagadnienia najkrótszej ścieżki w grafie. Dodanie zmienności wag krawędzi grafu powoduje występowanie często nieoczywistych zjawisk - może na przykład okazać się, że lokalnie nieoptymalne rozwiązanie będzie zdecydowanie lepsze w kontekście globalnym (przykładowo: opłacalnym mogłoby być opóźnienie przybycia do miasta X, gdyż parę godzin później czas pokonania drogi do miasta Y będzie wielokrotnie krótszy).
Widać wyraźnie, że istotnym jest nie tylko to, jak szybko można dostać się do danego miasta, ale też sama godzina przejazdu przez miasto. Co więcej, nawet rozważanie przedziałów czasu, w których droga wychodząca z miasta ma stały czas pokonania, nie jest wystarczające; okazać się może bowiem, iż wyruszenie z kolejnego miasta godzinę później lub wcześniej znów będzie mieć znaczące konsekwencje.
Tego typu trudne do przewidzenia zależności mogą propagować poprzez całą trasę od miasta wyjściowego aż do celu podróży. Dlatego postanowiłem rozważać wszystkie możliwe godziny wyruszenia z miasta.

4 Algorytm

Algorytm bazuje na klasycznym algorytmie Dijkstry. Główną różnicą jest powtórzenie obliczania czasu dotarcia dla każdej możliwej godziny wyjazdu z miasta początkowego. Każdy wierzchołek grafu przechowuje 24-elementową tablicę zawierającą najmniejsze znane czasy dotarcia do tego wierzchołka (początkowo ustawione na $+\infty$, jeśli nie jest to sąsiad wierzchołka początkowego), wraz z informacją o tym, z którego wierzchołka i o której godzinie można w takim czasie dotrzeć.
Dla każdej godziny zgodnie z algorytmem Dijkstry wyliczane są czasy dotarcia do sąsiednich wierzchołków przez przetwarzany wierzchołek i - jeśli uzyskane czasy są lepsze - aktualizowane odpowiednie komórki tablic sąsiadów (oczywiście o indeksach zwiększonych modulo 24 o czas dojazdu do sąsiada).

Kolejność przetwarzania wierzchołków determinowana jest przez najmniejszy możliwy czas dotarcia do wierzchołka. Przetworzony wierzchołek zostaje usunięty z listy wierzchołków oczekujących. Algorytm w naturalny sposób kończy działanie, gdy skończą się wierzchołki do przetwarzania.
Po zakończeniu pracy algorytmu wystarczy wybrać najniższy możliwy czas dotarcia do wierzchołka docelowego. Dzięki przechowywaniu wraz z czasem trasy wierzchołka-poprzednika, można prosto odtworzyć ścieżkę wracając "po śladach"do wierzchołka początkowego.

5 Złożoność

Jako, że algorytm ten bazuje na algorytmie Dijkstry i wykonuje go (w przybliżeniu) stałą liczbę razy dla każdego wierzchołka, jego złożoność obliczeniowa będzie tej samej klasy co szukanie najkrótszej ścieżki w zwykłym grafie: $O((E \cdot \log_2 V)$ przy użyciu `std::priority_queue` do zaimplementowania kolejki, gdzie V to liczba wierzchołków, a E to liczba krawędzi. W zależności od implementacji i dodatkowych sprawdzeń usprawniających algorytm powinien złożoność obliczeniową około $O(24 \cdot Dijkstra)$. Złożoność pamięciowa jest liniowa, w okolicach $O(74V + 4E)$.