Problem komiwojażera-krótkodystansowca

dokumentacja powykonawcza

Autorzy: Piotr Bródka, Rafał Kobiela

Wstęp, zmiany w porównaniu do założeń

Problem składał się z dwóch podproblemów. Pierwszy polegał na przygotowaniu Minimum Bottleneck Spanning Tree (MBST) - drzewa rozpinającego, w którym waga największej krawędzi jest minimalna (wśród innych drzew rozpinających tego samego grafu).
Drugim podproblemem było wyznaczenie ścieżki Hamiltona na tym drzewie, korzystając z możliwości przeskakiwania jednego lub dwóch wierzchołków.

Druga część udała się. W pierwszej natomiast w założeniu chcieliśmy skorzystać z algorytmu Cameriniego, którego czas jest liniowy. Mimo wielu starań nie udało nam się sprawić, by zawsze działał. Naszym podejrzeniem jest to, że nie zrozumieliśmy dokładnie, czym jest $(G_A)_\eta$ (o którym mowa w artykule en.wikipedia.org/wiki/Minimum_bottleneck_spanning_tree).

Wyjściem z sytuacji było posłużenie się algorytmem Kruskala, który znajduje minimalne drzewo rozpinające - MST. Zauważmy, że każde MST jest MBST, więc algorytm daje poprawny wynik. Natomiast jego złożoność jest liniowo-logarytmiczna.

Kod źródłowy

W tej części opiszemy, co znajduje się w plikach źródłowch projektu.

1. **Camerini.cs** - Próba implementacji algorytmu Cameriniego. Jest wiele funkcji pomocniczych, które tworzą główną funkcję.
2. **TestCamerini.cs** - Testy, do funkcji pomocniczych, wykorzystywanych w Camerini.cs. Wszystkie testy działają zgodnie z naszymi założeniami.
3. **BottleneckTravellingSalesman.cs** - Implementacja chodzenia po drzewie rozpinającym (o tym w dalszej części).
4. **HamiltonPathChecker.cs -** Służy do sprawdzania, czy utworzona ścieżka Hamiltona jest poprawna oraz liczy jej koszt.
5. **GraphSerializer.cs** - tworzenie grafu z danych w pliku (a także zapis grafu do pliku).
6. **TreeGenerator.cs** - Generator losowych drzew. Służył do wygenerowania dużego zbioru przypadków testowych o rosnącej wielkości, by sprawdzić, jak rośnie czas wykonania programu wraz ze wzrostem rozmiaru zadania.

7. **FullGraphGeneator.cs** - Generator grafów pełnych o losowych wagach. Powstał po to, by nie trzeba było tworzyć ręcznie plików testowych.

Chodzenie po drzewie rozpinającym

Implementacja znajduje się w pliku BottleneckTravellingSalesman.cs i jest tam wiele komentarzy, ułatwiających zrozumienie algorytmu. Natomiast wskazane jest jeszcze coś dodać.

Szukamy pierwszego wierzchołka, na którym jest rozgałęzienie (przypadek, gdy nie ma takiego jest też rozpatrywany w algorytmie. Rozpoczynamy tworzenie ścieżki wokół niego. To znaczy we wszystkie kierunki wokół wierzchołka centralnego jest uruchamiana rekurencyjnie funkcja, wyznaczająca ścieżki.
Gdy znalezione zostaną ścieżki dla wszystkich kierunków wokół wierzchołka, metoda Bind() łączy je w jedną.
Przy chodzeniu po drzewie, gdy doszliśmy do liścia, to kończymy rekursję i dla ostatniego wierzchołka na którym nastąpiło rozgałęzienie - dodajemy do jego ścieżki Hamiltona ścieżkę, łączącą go z liściem (i powrót z liścia).

Bardziej szczegółowe komentarze znajdują się w kodzie źródłowym.

Testy poprawności działania chodzenia po drzewie rozpinającym

W pliku **HamiltonPathChecker.cs** znajduje się metoda, sprawdzająca, czy wynik algorytmu jest poprawny.
Po pierwsze - w ścieżce Hamiltona muszą znaleźć się wszystkie wierzchołki grafu.
Po drugie - dla każdych dwóch sąsiednich wierzchołków - ich odległość liczona ilością krawędzi nie może być większa niż 3.

Dalszy opis testów poprawności działania chodzenia po drzewie rozpinającym znajduje się w pliku **“Testy dla tworzenia ścieżek Hamiltona.pdf**”. Są pokazane przykłady drzew i zwracane przez nie wyniki.

Testy można uruchomić, o tym będzie w instrukcji obsługi.

Testy poprawności działania całego systemu

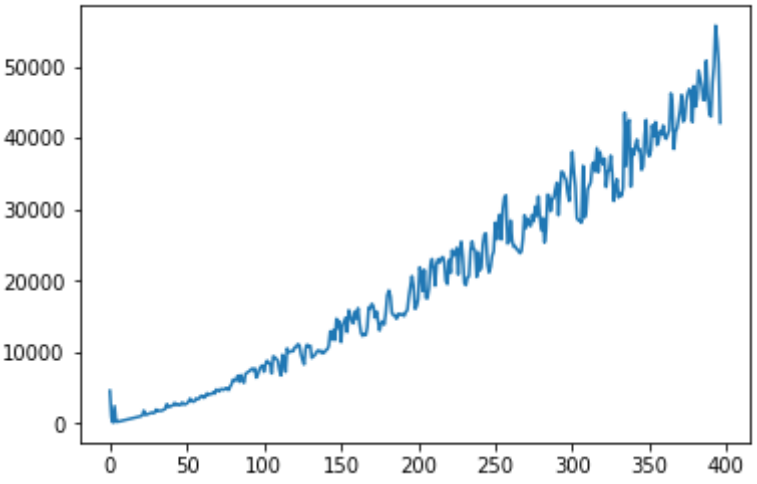
W pliku **“Testy dla całego problemu.pdf”** przedstawione są grafy, dla których testowaliśmy rozwiązanie. Zauważmy, że dla pewnych grafów nie ma możliwości wyznaczenia cyklu Hamiltona. Algorytm komiwojażera - krótkodystansowca radzi sobie z takimi przypadkami.

Testy można uruchomić, o tym będzie w instrukcji obsługi

Testy wydajnościowe, czyli sprawdzenie, jaka jest złożoność czasowa algorytmu

Ponieważ do wyznaczenia MBST korzystamy z algorytmu Kruskala, który przekracza złożoność liniową, do sprawdzenia, jak czas wykonania algorytmu zależy od rozmiaru problemu, wykorzystamy jedynie część, w której wyznaczamy sposób chodzenia po drzewie rozpinającym.

Przetestowaliśmy algorytm dla rozmiarów zadań (liczba wierzchołków) od 2 do 400. Czas wykonania liczony jest w cyklach procesora. Oto zależność:



Potwierdziliśmy nasze przypuszczenia z rozważań teoretycznych. Zależność jest liniowa.

Instrukcja obsługi

Graphviz

Graphviz jest biblioteką do wizualizacji grafów. Konieczna jest ona do tworzenia wizualizacji grafów. Plik z biblioteką jest w głównym folderze i nazywa się graphviz. Klikamy go i przechodzimy przez instalator.

Na początek

Rozpakowujemy plik projektu. Wchodzimy do BottleneckTravellingSalesman > bin > Debug. Mamy dwa foldery, które nas interesują: TREES i GRAPHS.

Testowanie poprawności działania chodzenia po drzewie rozpinającym (wymaga zmiany w kodzie)

Wchodzimy do TREES. Mamy tam w plikach tekstowych zapisane grafy. Format plików:

```
<ilość wierzchołków>
<wierzchołek startowy krawędzi> <wierzchołek końcowy krawędzi>
...
<wierzchołek startowy krawędzi> <wierzchołek końcowy krawędzi>
```

W pliku **Program.cs** w linii 23. Powinno być:

```
WHAT_TO_DO todo = WHAT_TO_DO.construct_hamiltonian_path;
```

Wybieramy za pomocą tego sposób działania programu.

Uruchamiamy program w trybie Debug.
Wówczas program odczytuje pliki w folderze TREES. Tworzy ich reprezentacje graficzne o takich samych nazwach, jak pliki tekstowe. Dodatkowo w folderze PATHS zapisywane są ścieżki dla odpowiednich drzew.

W przesłanych źródłach pliki wyjściowe już są (bo program był odpalany). Żeby zobaczyć, że program działa, można usunąć te pliki i ponownie uruchomić program.

Równocześnie na konsoli wypisywane są wyniki działania programu.

Testowanie poprawności działania całego systemu (nie wymaga zmiany w kodzie)

W pliku **Program.cs** w linii 23. Powinno być:

```
WHAT_TO_DO todo = WHAT_TO_DO.construct_MBST_and_hamiltonian_path;
```

Wybieramy za pomocą tego sposób działania programu. Jest to opcja, wybrana w skompilowanym pliku .exe.

Program odczytuje pliki w folderze GRAPHS.
Format plików:

```
<ilość wierzchołków>
<wierzchołek startowy krawędzi> <wierzchołek końcowy krawędzi>
...
<wierzchołek startowy krawędzi> <wierzchołek końcowy krawędzi>
```

W folderze MBST tworzą się drzewa rozpinające dla odpowiednich grafów. Dodatkowo w folderze PATHS zapisywane są ścieżki chodzenia po drzewach.

Równocześnie na konsoli wypisywane są wyniki działania programu. Dla grafu wywoływane jest rozwiązywanie problemu komiwojażera:

1. Metodą dokładną - Branch and Bound,
2. Metodą aproksymacyjną - komiwojażera - krótkodystansowca.

Wniosek z działania

Algorytm komiwojażera - krótkodystansowca gwarantuje nam znalezienie rozwiązania co najwyżej 3 razy droższego od optymalnego. Natomiast w praktyce - przy losowych wagach krawędzi - koszt cyklu Hamiltona w algorytmie aproksymacyjnym jest większy około 1.5 raza. Wydaje nam się, że jest dobry wynik.

Podział pracy

Dokumentacja początkowa - Piotr Bródka, Rafał Kobiela
Algorytm Cameriniego - Piotr Bródka
Algorytm chodzenia po drzewie rozpinającym - Piotr Bródka, Rafał Kobiela
Przygotowanie przypadków testowych - Rafał Kobiela
Architektura informatyczna rozwiązania - Piotr Bródka