Projektowanie Efektywnych Algorytmów

Projekt

19/10/2021

252712 Jarosław Kopaczewski

(1) Brute force

Spis treści	Strona

1. Sformułowanie zadania

Zadanie polega na implementacji oraz przeprowadzeniu testów efektywności algorytmu przeglądu zupełnego rozwiązującego problem komiwojażera.

Rozwiązanie problemu komiwojażera sprowadza się do znalezienia w zadanym Grafie pełnym cyklu Hamiltona, czyli drogi zamkniętej, w której każdy wierzchołek grafu został odwiedzony jeden raz(oprócz początkowego). Można rozróżnić również dwa rodzaje wyżej wymienionego problemu czyli wariant symetryczny oraz asymetryczny. W przypadku pierwszego z nich odległość pomiędzy wierzchołkami np. A i B jest taka sama zarówno podczas przejścia z A do B jak i z B do A, natomiast w drugim wariancie waga drogi może być różna. Ze względu na swój charakter metoda przeglądu zupełnego rozwiązuje zarówno problem symetryczny oraz asymetryczny, ponieważ w przypadku tego problemu rozwiązanie polega na sprawdzeniu sumy wag krawędzi dla każdej z permutacji wierzchołków.

[spis]

2. Opis metody

Metodą wykorzystaną przy projektowaniu algorytmu rozwiązującego zadany problem jest przegląd zupełny, który polega na znalezieniu wszystkich dopuszczalnych rozwiązań problemu oraz wybraniu potrzebnego nam ekstremum, czyli w tej sytuacji będzie to minimum. Efektem działania jest uzyskanie optymalnego i jednocześnie najlepszego możliwego rozwiązania problemu

Metoda przeglądu zupełnego jest jedną z prostszych w implementacji, jednak sprawdzenie wszystkich możliwych kombinacji skutkuje długim lub wręcz nieakceptowalnym czasem rozwiązywania. Widoczne jest to szczególnie w problemach o dużej złożoności takich jak np. rozwiązywany w aktualnym zadaniu problem komiwojażera, którego złożoność wynosi O(n!). Przeszukiwanie wyczerpujące posiada również swoje zalety, czyli pewność znalezienia rozwiązania (o ile istnieje w danej instancji problemu) oraz to że jest ono najlepsze z możliwych.

[spis]

3. Opis algorytmu

Zmienne wykorzystywane w algorytmie:

size - ilość wierzchołków potrzebna do warunków kończących pętle

matrix - tablica dwuwymiarowa reprezętująca graf pełny

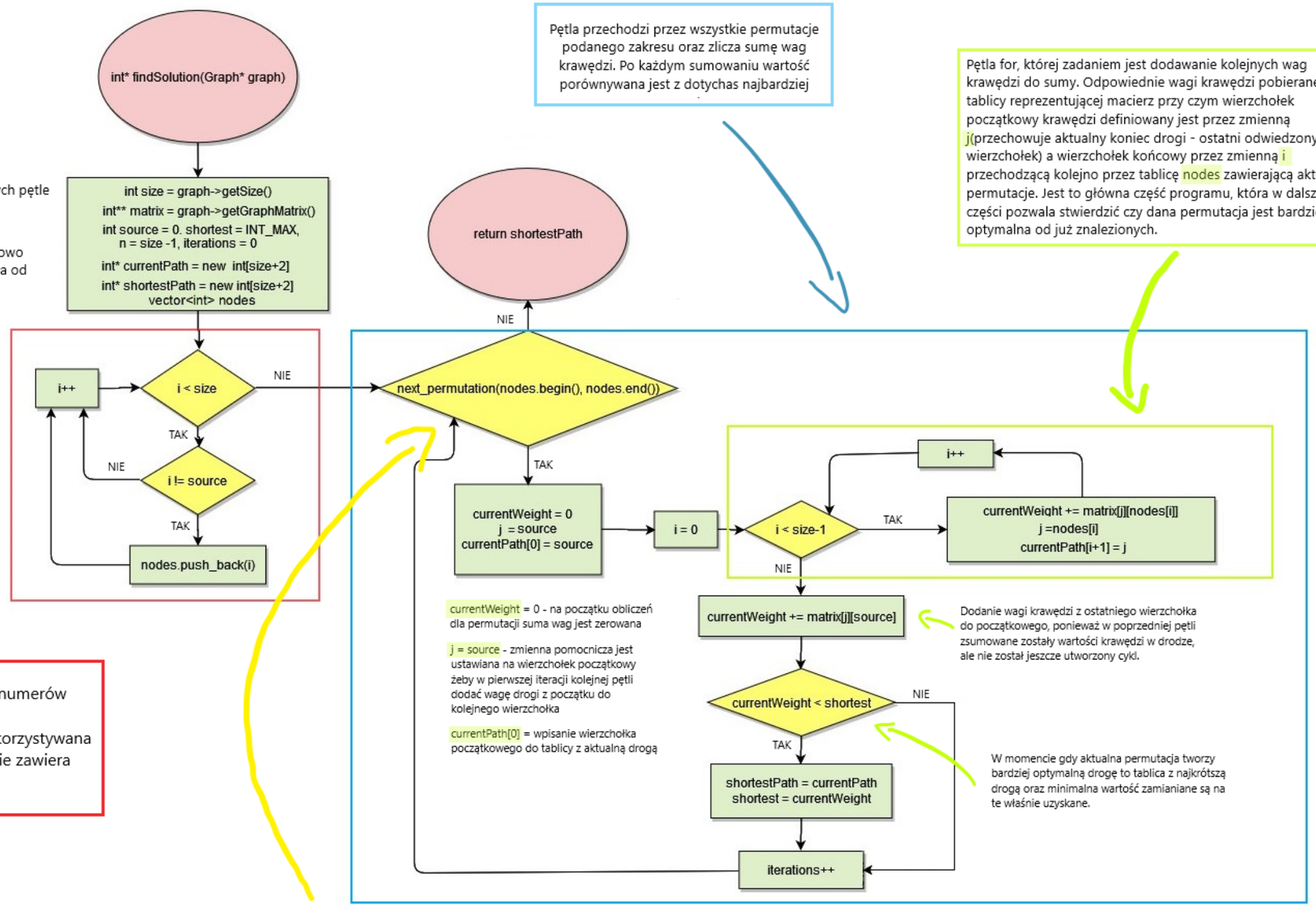
source - wierzchołek początkowy

shortest - aktualnie najkrótsza ścieżka w grafie jest początkowo ustawiona na INT_MAX, aby pierwsza obliczona wartość była od niej mniejsza

currentPath - podczas algorytmu zapisywana jest do niej aktualna ścieżka

shortestPath - jeżeli aktualna ścieżka jest krótsza od najkrótszej znalezionej dotychczas to przyjmuje jej wartość

nodes - tablica dynamiczna przechowująca numery wierzchołków pomijając początkowy - służy do wyznaczania permutacji



Pętla for, której zadaniem jest dodanie numerów wierzchołków do tablicy dynamicznej.
W dalszej częsci programu jest ona wykorzystywana przy obliczaniu kolejnych permutacji. Nie zawiera wierzchołka początkowego

next_permutation - zwraca true jeżeli istnieje leksykograficznie większa permutacja oraz modyfikuje podany zakres a w przeciwnym przypadku zwraca false

Rysunek 1: Schemat blokowy wraz z komentarzami

[spis]

3.1 Opis słowny algorytmu

Algorytm rozpoczyna swoje działanie od utworzenia tablicy dynamicznej(*vector*) i uzupełnienia jej numerami wszystkich wierzchołków oprócz początkowego.

W dalszej części programu jest ona wykorzystywana jako zakres podawany do funkcji *next_permutation*, która wyznacza leksykograficznie większą permutację dla zadanego zakresu. Wymieniona funkcja zwraca również wartość bool, dlatego wykorzystana została od razu jako warunek kolejnej pętli. W momencie gdy nie istnieje większa permutacja, czyli wróciliśmy do ułożenia początkowego to program opuszcza pętle do..while.

Natomiast wewnątrz tej pętli dokonywane jest sumowanie kolejnych krawędzi pomiędzy wierzchołkami w kolejności wyznaczonej przez obecną permutację. W momencie gdy po zsumowaniu aktualnego cyklu wartość jest mniejsza od aktualnego minimum(na początku jest to INT_MAX) to minimum przyjmuje wartość tej sumy. Ponadto podczas zliczania wag zapisywana jest również kolejność wierzchołków do tablicy.

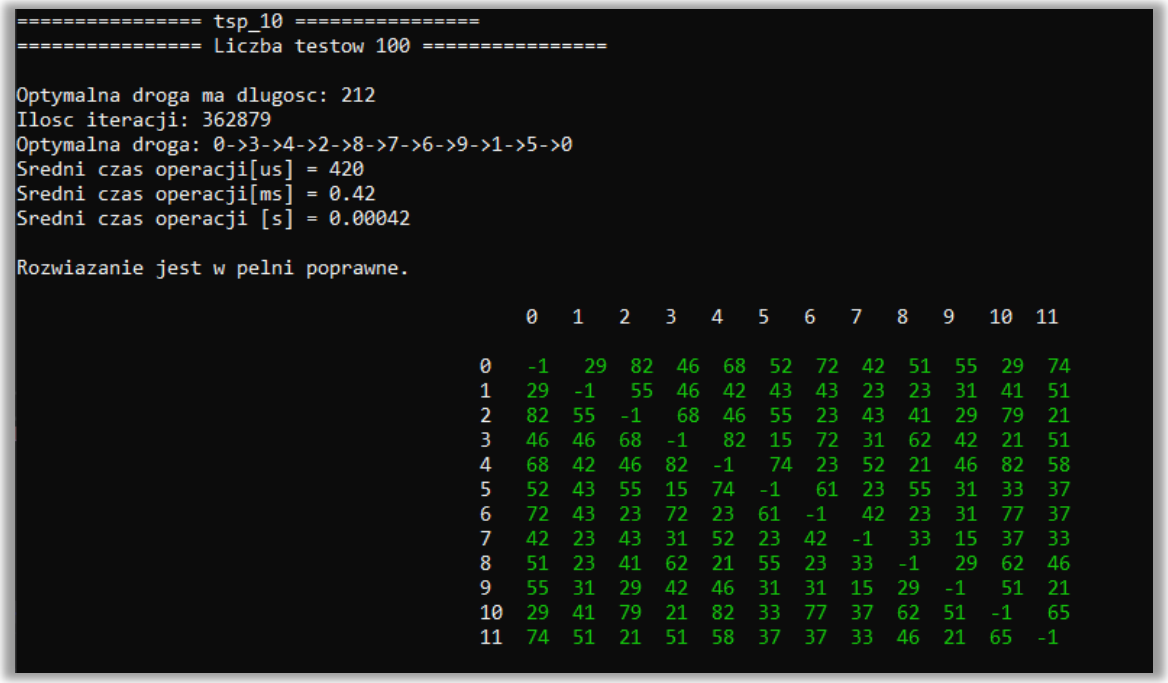
Efektem działania algorytmu jest najlepsze rozwiązanie problemu oraz kolejność wierzchołków w cyklu.

[spis]

4. Dane testowe

Do przetestowania poprawności działania algorytmu oraz badań wybrane zostały następujące instancje grafów:

- tsp_6_1.txt
- tsp_6_2.txt
- tsp_10.txt
- tsp_12.txt
- tsp_13.txt
- tsp_14.txt
- tsp_15.txt; [źródło]



Rysunek 2: Przykładowy wynik testu poprawności

Przed przystąpieniem do fazy testów czasowych zostały wykonane testy poprawności wczytywania oraz wyników dla każdego z plików testowych, przykładowy wynik działania programu (rysunek 2) prezentuje uzyskane wyniki oraz wyświetloną macierz testową dla pliku tsp_10.txt. Program na podstawie wczytanych danych z pliku inicjalizującego sam sprawdza uzyskane wyniki(rozwiązanie, ilość iteracji oraz cykl). W przypadku plików 14 i 15 może pojawiać się komunikat o niepoprawnych wynikach ze względu na brak informacji o ilości iteracji w zadanych plikach, jednak ścieżka oraz długość drogi jest poprawna.

[spis]

5. Procedura badawcza

Badanie polegało na zmierzeniu czasu wykonania algorytmu dla każdej z instancji. W przypadku miejszych instancji pomiary wykonywane były wielokrotnie i wyciągana była ich średnia, wraz z zwiększaniem ilości wierzchołków ilość testów była zmniejszana ze względu na czas potrzebny do realizacji algorytmu, który w przypadku tsp_15 zajmował już około 20min.

Ze względy na wykorzystanie metody przeglądu zupełnego do rozwiązania problemu przeprowadzenie testów polegało na włączeniu programu wykorzystując odpowiedni plik konfiguracyjny.

```
8 0 0 output ->
1000 132 119 tsp_6_1 0->1->2->3->4->5->0
1000 80 119 tsp_6_2 0->5->1->2->3->4->0
500 212 362879 tsp_10 0->3->4->2->8->7->6->9->1->5->0
10 264 39916799 tsp_12 0->1->8->4->6->2->11->9->7->5->3->10->0
5 269 479001599 tsp_13 0->10->3->5->7->9->11->2->6->4->8->1->12->0
3 282 1 tsp_14 0->10->3->5->7->9->13->11->2->6->4->8->1->12->0
1 291 1 tsp_15 0->12->1->14->8->4->6->2->11->13->9->7->5->3->10->0

```

Rysunek 3: przykładowy plik inicjalizacyjny

Informacje zawarte w pliku reprezentują nazwy plików testowych, ilość testów oraz oczekiwane rezultaty. Pierwsza linia zawiera dodatkowo nazwę pliku wyjściowego csv oraz ilość elementów w pliku (dane o wynikach zostały uzupełnione 0 oraz pustą drogą aby zachować strukturę). Dokładna kolejność informacji to:

[ilość testów; wynik; ilość iteracji; nazwa pliku; cykl wynikowy]

Program testowany był w wersji 64 bit release dla wszystkich plików podanych w informacjach o danych testowych.

```
Nazwa pliku,czas[ms],czas[s],ilosc testow, droga
tsp_6_1,1.732e-06,0.00173,100, 0->1->2->3->4->5->0
tsp_6_2,2.48e-06,0.00248,100, 0->5->1->2->3->4->0
tsp_10,0.00435,4.35,50, 0->3->4->2->8->7->6->9->1->5->0
tsp_12,0.599,599,20, 0->1->8->4->6->2->11->9->7->5->3->10->0

```

Rysunek 4: Przykładowy plik wynikowy .csv

[spis]

5.1 Specyfikacja sprzętowa

System Operacyjny	Windows 10 Home 64 bit
Procesor	Intel i5-8250U 1,6GHz – 3,4GHz
Pamięć RAM	16 GB
Płyta główna	ASUS X510UNR
IDE	Visual Studio

Tabela 1: Specyfikacja sprzętowa komputera na którym wykonano pomiary

5.2 Pomiar czasu

Do pomiaru czasu operacji została użyta funkcja *QueryPerformanceCounter*, która umożliwia dokładniejszy pomiar rzędu mikrosekund. Przed wykonaniem algorytmu wartość zegara wpisywana była do zmiennej a następnie odejmowana od stanu zegara po wykonaniu. Jako że wyżej wymieniona funkcja zwraca jedynie stan zegara wynik należało podzielić jeszcze przez jego częstotliwość. Wszystkie wyniki cząstkowe były sumowane, aby po wykonaniu wyliczyć średnią.

```
long long int frequency, start = 0, elapsed, sum, size = 0;
QueryPerformanceFrequency((LARGE_INTEGER*)&frequency);
```

Rysunek 5: Inicjalizacja zmiennych używanych do pomiaru czasu

```
for (int i = 0; i < initValues[k].first[0]; i++)
{
    start = read_QPC();
    result = BruteForce::findSolution(graph);
    elapsed = read_QPC() - start;
    sum += elapsed;
}
```

Rysunek 6: Pętla pomiaru czasu – initValues[k].first[0] to ilość testów

[spis]

```cpp
long long int Test::read_QPC()
{
    LARGE_INTEGER count;
    QueryPerformanceCounter(&count);
    return((long long int)count.QuadPart);
}
```

Rysunek 7: Pobranie wartości zegara

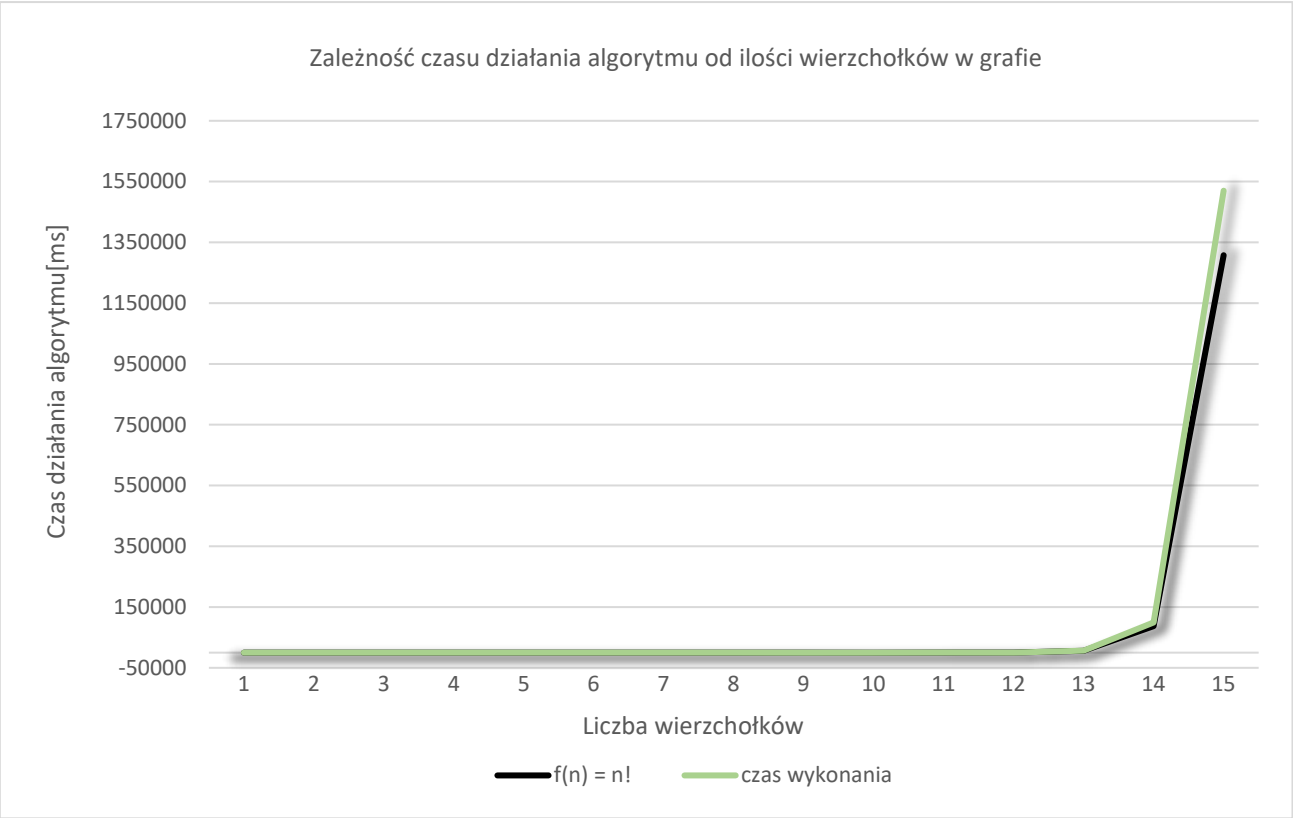
```cpp
    cout << path << endl;
    cout << "Sredni czas operacji[s] = " << setprecision(3) << float(sum / float(test))/f << endl;
    cout << "Sredni czas operacji[ms] = " << setprecision(3) << float(sum * 1000.0)/ float(test)/f << endl;
    cout << "Sredni czas operacji [us] = " << setprecision(3) << float(sum * 1000000.0) / float(test)/f << endl << endl;
```

Rysunek 8: Wyświetlenie wyników – f to częstotliwość

[spis]

6. Wyniki

Wynik testów zostały umieszczone w pliku output.csv, który powstał w wyniku działania programu. Zebrane wyniki przedstawione zostały w postaci wykresu zależności ilości wierzchołków w instancji do czasu wykonania algorytmu(wykres 1).



Wykres 1: Wykres zależności czasu działania algorytmu od ilości wierzchołków

[spis]

7. Analiza wyników i wnioski

Wyniki przedstawione na wykresie(wykres 1) przedstawiają czas(*zielony*) potrzebny na rozwiązanie problemu komiwojażera w zależności od ilości wierzchołków w zadanym grafie oraz przeskalowany wykres *n!* (*czarny*). Z wykresu wynika że czasy uzyskane podczas testów rosną w sposób bardzo zbliżony do zakładanego n!. Oprócz wykresu wskazuje na to zależność pomiędzy poszczególnymi wynikami tzn. wynik dla instancji o ilości wierzchołków 14 jest w przybliżeniu równy wynikowi dla instancji z 13 wierzchołkami pomnożonej przez 14. Identyczna zależność występuje dla wszystkich następujących po sobie wynikach, co wskazuje na złożoność O(n!).

W przypadku wyniku dla pliku testowego *tsp_15.txt* wartość zaczyna odbiegać od wykresu n! (wykres 1) – zaczyna być większa niż można by zakładać. Najprawdopodobniej jest to spowodowane faktem, że złożoność n! byłaby idealnym rozwiązaniem dla metody przeglądu zupełnego, a zastosowany algorytm wykonuje również dodatkowe operacje mające na celu zebranie danych. Nie są one zbyt czasochłonne, ale biorąc pod uwagę ilość permutacji dla instancji 15 mogą mieć już znaczący wpływ na ostateczny wynik czasowy algorytmu.

[spis]