Sprawozdanie - projekt 1.

Implementacja i analiza efektywności algorytmu podziału i ograniczeń   
dla problemu komiwojażera.

Autor: Michał Bańka, 235051  
Prowadzący: mgr inż. Antoni Sterna  
Termin zajęć: Czwartek, 7:30

1. Opis problemu

Problem komiwojażera (ang. TSP – Travelling Salesman Problem) jest problemem, który polega na znalezieniu najkrótszego cyklu Hamiltona w grafie. Oznacza to, że ścieżka powinna zawierać wszystkie wierzchołki dokładnie jeden raz i przez każdą krawędź przechodzić również najwyżej jeden raz. Dodatkowo ostatni wierzchołek powinien być połączony z pierwszym, tak aby ścieżka stała się cyklem.

Dla 4 wierzchołków mamy 3\*2\*1 możliwości wyboru ścieżek, dla 5 wierzchołków 4\*3\*2\*1 ścieżek itd. Tak więc dla *n* wierzchołków ilość ścieżek do sprawdzenia czy są najkrótszymi cyklami Hamiltona wyniesie (n-1)\*(n-2)\*…\*2\*1 = (n-1)! . Najprostsza metoda algorytmu rozwiązującego problem będzie miała złożoność (n-1)! .

Sprowadza to problem komiwojażera do zbioru problemów NP-trudnych, a właściwie do jego podzbioru problemów NP.-zupełnych. Oznacza to, że problem jest możliwy do sprawdzenia poprawności w czasie wielomianowym, ale nie jest możliwe znalezienie jego rozwiązania w czasie wielomianowym. Złożoność problemu komiwojażera jest wykładnicza, gdyż:  
n!>2n już od n=4.

1. Opis algorytmów
2. Przegląd zupełny

Ideą algorytmu przeglądu zupełnego jest jak sama nazwa mówi sprawdzenie każdego możliwego rozwiązania, czyli w tym wypadku sprawdzenie każdego cyklu Hamiltona w grafie i porównanie go z najmniejszym znalezionym dotychczas, a następnie zapamiętanie tego najkrótszego. Algorytm w napisanym programie wykonuje się rekurencyjnie przeglądając po kolei nieużyte dotychczas wierzchołki. Początkowo w grafie umieszczamy pierwszy wierzchołek (jako, że jest to cykl, nie ma znaczenia od którego zaczniemy). Sprawdzamy długość cyklu tylko w momencie kiedy ilość wierzchołków w cyklu jest równy ilości wierzchołków w grafie

C:\Users\MichalPC1\Downloads\graf.pngPrzykład dla grafu 1. :

Graf 1

Inicjalizujemy: Cykl: 0, Min = MAX\_INT = 2³¹-1

Przeglądamy wszystkie pozostałe wierzchołki które nie są jeszcze w Cyklu, po kolei:

Next = 0 - już jest w cyklu

Next = 1 - dodajemy do cyklu

Cykl: 0 -> 1

Next = 0 – pomijamy

Next = 1 – pomijamy

Next = 2 – dodajemy

Cykl: 0 -> 1 -> 2

Next = 0, 1, 2 – pomijamy

Next = 3 – dodajemy

Cykl: 0 -> 1 -> 2 -> 3

Cykl ma rozmiar równy ilości wierzchołków w grafie, więc obliczamy długość.

Długość(Cykl) = 11

Długość mniejsza niż Min, więc pod Min podstawiamy długość cyklu.

Min = 11

Brak wierzchołków do dodania, więc usuwamy ostatni (3) i cofamy o jedną

iterację.

Brak wierzchołków do dodanie – usuwamy ostatni (2).

Next = 3 – dodajemy

Cykl: 0 -> 1 -> 3

Next = 0, 1 – pomijamy

Next = 2 – dodajemy

Cykl: 0 -> 1 -> 3 -> 2

Liczymy długość.

Długość(Cykl) = 8

Min = 8

Brak wierzchołków do dodania – usuwamy ostatni (2).

Next = 3 – pomijamy

Usuwamy ostatni (1).

Usuwamy ostatni.

Next = 2 – dodajemy

Cykl: 0 -> 2

Next = 0 – pomijamy

Next = 1 – dodajemy

Cykl: 0 -> 2 -> 1

Next = 0, 1, 2 – pomijamy

Next = 3 – dodajemy

Cykl: 0 -> 2 -> 1 -> 3

Długość(Cykl) = 9

Nie jest mniejszy niż znaleziony minimalny cykl.

Usuwamy ostatni (3).

Usuwamy ostatni (1).

Next = 2 – pomijamy

Next = 3 – dodajemy

Cykl: 0 -> 2 -> 3

Next = 0 – pomijamy

Next = 1 – dodajemy

Cykl: 0 -> 2 -> 3 -> 1

Długość(Cykl) = 8

Nie mniejsze niż Min. Usuwamy ostatni (1).

Next = 2, 3 – pomijamy

Usuwamy ostatni (3).

Usuwamy ostatni(2).

Next = 3 – dodajemy

Cykl: 0 ->3

Next: 0, 3 – pomijamy

Next: 1 – dodajemy

Cykl: 0 -> 3 -> 1

Next = 0,1,3 – pomijamy

Next = 2 – dodajemy

Cykl: 0 -> 3 -> 1 -> 2

Długość(Cykl) = 9

Usuwamy ostatni (2).

Usuwamy ostatni(1).

Next: 2 – dodajemy

Cykl: 0 -> 3 -> 2

Next = 0,2,3 – pomijamy

Next = 1 – dodajemy

Cykl: 0-> 3 -> 2 -> 1

Długość(Cykl) = 11

Usuwamy ostatni(1).

Usuwamy ostatni(2).

Usuwamy ostatni(3).

Usuwamy ostatni(0).

Koniec algorytmu – najkrótszy cykl to 0 -> 1 -> 3 -> 2 o długości 8.

1. Algorytm podziału i ograniczeń

Algorytm ten opiera się na budowie drzewa rozwiązań. Każdy węzeł w drzewie rozwiązań zawiera w swoim poddrzewie rozwiązania, które można utworzyć z tego węzła. Jest to część podziału grafu. Druga część to część ograniczająca. Ograniczanie drzewa odbywa się przez odcinanie kolejnych gałęzi (poddrzew), które z obliczeń wiemy, że nie będą posiadać rozwiązania lepszego niż uzyskane dotychczas.

Dla zaoszczędzenia czasu poświęconego na obliczenia i pamięci, obie części algorytmu będą wykonywały się jednocześnie, tzn. węzły będą dodawane po kolei od góry drzewa i na bieżąco obliczane będą najlepsze rozwiązania poddrzewa („dolna granica”). Jeśli dolna granica będzie większa niż najlepsze uzyskane dotąd rozwiązanie („górna granica”), to nie jest tworzone dalej poddrzewo i dalej sprawdzamy podrozwiązania na tym samym poziomie drzewa lub przesuwamy się wyżej jeśli brak spełniających warunek podrozwiązań.

Strategią wybierania następnego do sprawdzenia poddrzewa z aktualnego wierzchołka jest rozwijanie tego, który ma najmniejszą dolną granicą (jest najbardziej obiecujący).

Sposób obliczania dolnej granicy znajduje się w punkcie 3.

Przykład dla grafu 1. :

C:\Users\MichalPC1\Downloads\Untitled Diagram.pngInicjując drzewo górnej granicy przypisujemy dowolną dużą wartość, jednak aby mieć pewność  
w tworzonym algorytmie górną granicą będzie suma wszystkich krawędzi.

Rysunek 1.1

Rysunek .2

C:\Users\MichalPC1\Downloads\Untitled Diagram (3).png

1. Opis implementacji algorytmów i struktur
2. przechowywanie grafu

Graf przechowywany jest w stworzonej specjalnie dla niego klasie Matrix. Klasa ta została stworzona do przechowywania macierzy kwadratowych i wykonywania prostych operacji na niej. Dodatkowo na macierzy można zastosować opisywane w punkcie 2. algorytmy.

Struktura grafu zapisana jest w postaci macierzy sąsiedztwa, gdzie jako pierwsza współrzędna macierzy podaje się wierzchołek, z którego krawędź wychodzi, a jako druga wierzchołek końcowy krawędzi.

Wszystkie struktury alokują pamięć dynamicznie, co zostało uwzględnione podczas mierzenia czasu.

1. Implementacja Algorytmu Podziału i Ograniczeń

Mimo, że algorytm ten opiera się na konstrukcji drzewa, to program jedynie symuluje jego budowę przechodząc rekurencyjnie przez drzewo dodając kolejno wierzchołki. Zarówno pamięć jak i zasoby obliczeniowe nie są zużywane na fizyczną budowę drzewa.

Funkcja ta używa struktur danych takich jak tablica reprezentowana przez bibliotekę STL, a konkretnie *vector*. Podczas używania *vectora* brana była pod uwagę realokacja pamięci wewnątrz kontenera. W miejscach programu, które posiadały wiele operacji *push\_back i pop\_back,*  czyli takie, które zmieniają rozmiar struktury, stosowana była funkcja *resize,* pozwalająca na tylko jednorazową zmianę rozmiaru.

Kolejną strukturą była: *pair*. Jej funkcjonalnością jest utrzymywanie pomiędzy dwoma tablicami stałych relacji pomiędzy indeksami. Typem każdej z tablic może być dowolna klasa.

Do podstawowych operacji na tablicach (*vectorach*) została użyta klasa Algorithm i jej funkcje. Pozwoliła ona na np.: posortowanie tablicy ze złożonością *nlog(n)*, sprawdzenie czy istnieje bądź nie istnieje w tablicy element spełniający warunek*.*

Funkcja obliczająca dolną granicę polega na wybieraniu najkrótszych krawędzi dla kolejnych wierzchołków nie będących w dotychczasowej ścieżce lub będącemu ostatnim wierzchołkiem tej ścieżki. Funkcja inicjalizuje zmienną całkowitą *lb = 0.* Następnie sprawdza po kolei wszystkie wierzchołki grafu i zwiększa wartość *lb*.

- Jeśli sprawdzany wierzchołek jest w znanej ścieżce i nie jest ostatni w ścieżce to do *lb* dodaje się jego odległość do następnego w ścieżce.

- Jeśli sprawdzany wierzchołek jest w znanej ścieżce i jest ostatni to do *lb* dodaje się najkrótszą z krawędzi nie prowadzącą do wierzchołka ze znanej ścieżki.

-Jeśli sprawdzany wierzchołek nie jest w znanej ścieżce to do *lb* dodaje się najkrótszą z krawędzi nie prowadzącą do wierzchołka ze znanej ścieżki oprócz pierwszego z niej i nie do samego siebie.

W ten sposób funkcja zwraca najkrótszą ścieżkę, która mogłaby wystąpić w którymś z podrozwiązań (lecz nie musi).

1. przegląd zupełny

Przegląd zupełny również używa tablicy pod postacią *vectora* i na tej samej zasadzie brana jest pod uwagę realokacja pamięci. Algorytm ten również jest napisany w wersji rekurencyjnej.

1. Plan eksperymentu

Plan eksperymentu zakłada wykonanie obu algorytmów po 100 razy dla rozmiarów macierzy *n × n,* gdzie *n* wynosić ma odpowiednio 8, 9, 10, 11, 12. Elementy w macierzy są zmieniane na losowe z zakresu [1-50], pomiędzy każdym wywołaniem funkcji wykonujących algorytmy. Rozmiar grafu został ograniczony do 12 z powodu zbyt długich czasów wykonywania się algorytmu przeglądu zupełnego – dla rozmiaru 12 jedno wykonanie to ok. 40 sekund.

Pomiary czasu mierzą jedynie czas wykonania algorytmów, bez zmian elementów, wypisywania danych do konsoli i innych czynności wymaganych do obsługi programu.

Pomiar czasu powinien odbywać się przez funkcję *QueryPerformanceCounter,* zapewniający stosunkowo wysoką jakość pomiarów rzędu mikrosekund.

Co więcej do wykresu przedstawiającego działanie metody podziału i ograniczeń dla większych grafów zostanie użyta inna funkcja, która będzie polegała na tej samej zasadzie generowania danych, ale badane rozmiary to od 10 do 17 wierzchołków.

1. Wyniki eksperymentów

Wykres 1.1 – Czasy obu metod, skala liniowa

|  |  |  |
| --- | --- | --- |
| Rozmiar | Metoda podziału i ograniczeń  Czas [ms.] | Przegląd zupełny  Czas [ms.] |
| 8 | 0,177 | 3,986 |
| 9 | 0,500 | 33,040 |
| 10 | 1,108 | 316,74 |
| 11 | 3,161 | 3421,56 |
| 12 | 8,255 | 40146,8 |

Wykres 1.2 – Czasy obu metod, skala logarytmiczna

Wykres – czas metody podziału i ograniczeń

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rozmiar | Metoda podziału i ograniczeń  Czas [ms.] |  | Rozmiar | Metoda podziału i ograniczeń  Czas[ms.] |
| 10 | 3,46 |  | **14** | 338,98 |
| 11 | 10,84 |  | **15** | 812,73 |
| 12 | 34,72 |  | **16** | 2291,26 |
| 13 | 95,50 |  | **17** | 10093,90 |

1. Wnioski

Zależnie od tego jaki jest nasz cel i zasoby obliczeniowe i czasowe, obie metody mogą być efektywne. Metoda przeglądu zupełnego podczas jednorazowego obliczania najkrótszego cyklu może mieć zastosowanie dla rozmiaru grafu maksymalnie ok. 12. Co ciekawe jeśli rozmiar grafu nie jest stały to czas obliczeń również jest stały z pewnymi odchyleniami zależnymi od urządzenia obliczającego itp. Wiąże się to ze stałą ilością operacji potrzebną do wykonania algorytmu.

W przypadku metody podziału i ograniczeń może wystąpić sytuacja kiedy czas obliczeń jest równy czasowi metody przeglądu zupełnego. Dzieje się tak tylko kiedy każda następna przeglądana gałąź tworzy krótszy cykl niż poprzednia, w związku z tym należy przejrzeć wszystkie z nich. Prawdopodobieństwo wystąpienia takiego zdarzenia wynosi 1/(n-1)! .

Jest to bardzo małe prawdopodobieństwo, malejące wraz ze wzrostem wielkości grafu, co pozwala przypuszczać, że ta metoda jest prawie zawsze skuteczniejsza od przeglądu zupełnego.

W niektórych przypadkach czas obliczeń metodą podziału i ograniczeń może być bardzo niski, podczas gdy metodą podziału zupełnego nie doczekamy się wyniku nawet w ciągu stuleci.

Czas trwania obliczeń można szacować z rekurencyjnego wzoru, w którym czas t1 jest zależny od procesora: tn = tn-1 \* 3.