Projektowanie efektywnych algorytmów
Budynek C-3, sala 127L wtorek, 18:55
Prowadzący: Mgr inż. Antoni Sterna

Sprawozdanie

Projekt 1

Mateusz Hyzicki, nr indeksu: 225947

Problem:

Problem komiwojażera-zagadnienie optymalizacyjne, polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Dany jest graf o n wierzchołkach i wagi wszystkich krawędzi między nimi, celem jest przejście przez wszystkie wierzchołki grafu wraz z powrotem do wierzchołka początkowego, nie przechodząc przez żaden wierzchołek(prócz początkowego) więcej niż raz przez krawędzie o najmniejszej sumarycznej wadze.
Problem przedstawiany jest często z punktu widzenia sprzedawcy(komiwojażera), który chce odwiedzić tylko raz każde z miast i wrócić do miasta początkowego w możliwie jak najkrótszym czasie(najkrótsza droga).

Wykorzystane metody rozwiązania problemu:

Przegląd zupełny(tzw. Brute Force)-polega na prostym przejrzeniu wszystkich możliwych kombinacji grafu, policzeniu dla nich sum wag i wybraniu tej najmniejszej.
Złożoność $O(n!)$; pamięciowa $O(n^2)$

Metoda programowania dynamicznego-została opracowana przez Richarda Bellmana w połowie dwudziestego wieku. Polega na podziale problemu na podproblemy, których rozwiązania są zapamiętywane i używane, gdy wystąpią ponownie. Metoda ta ma zastosowanie do rozwiązywania tzw. problemów bez pamięci, spełniających tzw. własność Markowa w tym i problemu komiwojażera.
Złożoność $O(n!)$; pamięciowa $O(n!)$

Oba algorytmy zostały zaimplementowane za pomocą funkcji rekursywnej, która dochodzi do końcowego wierzchołka grafu i powracając wstecz podlicza wagę przebytej drogi. Wykorzystują tablicę reprezentującą odległości, które trzeba przebyć od jednego miasta do drugiego.
W programowaniu dynamicznym w odróżnieniu od przeglądu zupełnego dodatkowo zapisywane są konkretne odgałęzienia grafu(identyfikowane przez posortowaną listę jeszcze nie odwiedzonych miast oraz numer miasta, w którym się aktualnie znajdujemy; zawierające

najlepszą drogę od danego wierzchołka do końca grafu). Przed każdym kolejnym rekurencyjnym wywołaniem funkcji sprawdzane jest czy już nie wystąpiła taka sytuacja i w razie pomyślnego sprawdzenia korzysta się z gotowych wyników. Jest to bardzo korzystne, np. dla grafu o 10 wierzchołkach(miastach) algorytm 6858 razy korzysta z wcześniej obliczonych wyników. Przykład wykorzystania:
0 -> 1 -> 2 -> 3 -> … -> 0
0 -> 2 -> 1 -> 3 -> … -> 0
Powyższe trasy różnią się tylko na początkowym kawałkiem zaś optymalna droga przejścia od wierzchołka 3go do końca jest taka sama.

Przykład wykonania:

Dany jest graf o 5 wierzchołkach.
Sprawdzamy odnogi od wierzchołka 0(1,2,3,4)
Wybieramy wierzchołek nr 1 i sprawdzamy idące od niego odnogi(2,3,4)
Analogicznie wierzchołek nr 2(3,4)
Analogicznie wierzchołek nr 3(4)
Wierzchołek nr 4 ma pustą listę nieodwiedzonych miast, tworzymy ścieżkę uwzględniając powrót 4 -> 0 i zapisujemy ją w pamięci jako: miasto 3, lista nieodwiedzonych miast=[4]
Cofamy się do wierzchołka nr 3, tworzymy ścieżkę 3 -> 4 -> 0
cofamy się do miasta nr 2(3,4), wybieramy miasto nr 4 i jako jedyne nieodwiedzone nr 3. Tworzymy ścieżkę 3 -> 0
Cofamy się do wierzchołka nr 4, tworzymy ścieżkę 4 -> 3 -> 0 i porównujemy ją z ścieżką 3 -> 4 -> 0. Tą najkrótszą zapisujemy w pamięci. Dalej analogicznie:
4 -> 3 -> 0
2 -> 3 -> 4 -> 0
4 -> 0
2 -> 4 -> 0
2 -> 0
4 -> 2 -> 0
3 -> 2 -> 4 -> 0
3 -> 0
2 -> 3 -> 0
2 -> 0
3 -> 2 -> 0
4 -> 2 -> 3 -> 0
1 -> 2 -> 3 -> 4 -> 0
Tu jesteśmy przy wierzchołku 3, nieodwiedzony jest wierzchołek 4. Zapamiętaliśmy już co w takiej sytuacji jest optymalnym rozwiązaniem i je wykorzystujemy.

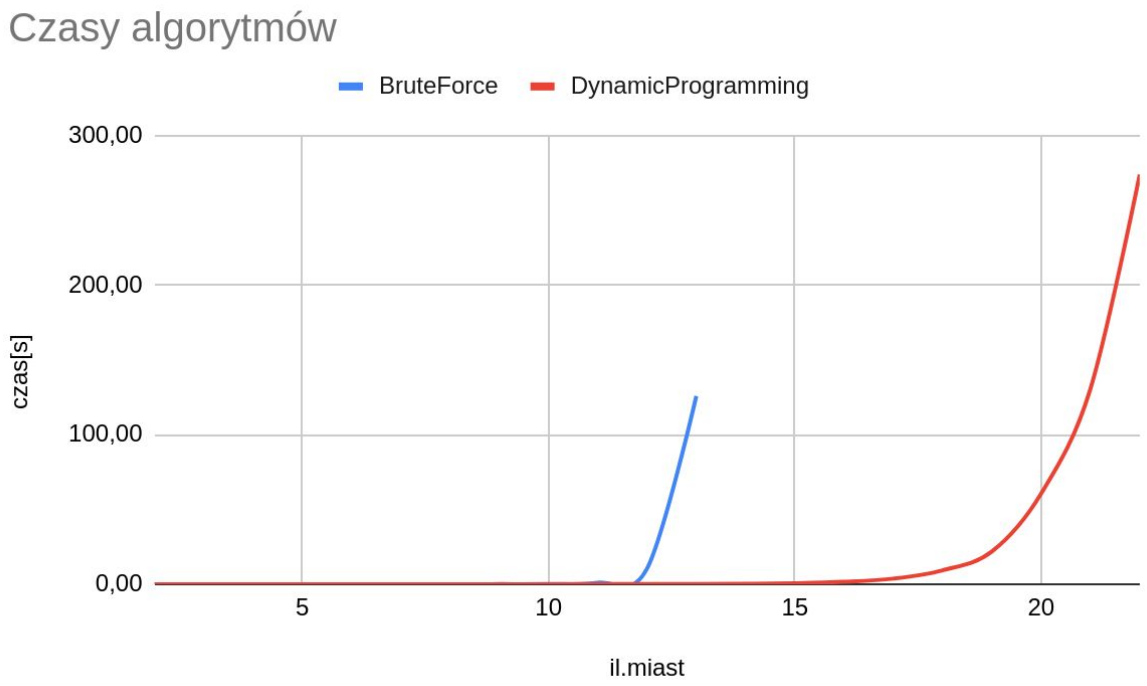
Wyniki:

Dla Przeglądu zupełnego przy 15 miastach program wykonywał się powyżej parudziesięciu minut. Było to sygnałem do zaprzestania testowania tego algorytmu.

Podczas testowania programowania dynamicznego przy 23 miastach komputer przestał odpowiadać, uległ tzw. zawieszeniu i konieczne było zresetowanie go przez odłączenie jego zasilania w celu kontynuacji pracy. Efektem tego jest następująca tabela

il.miast	BruteForce	DynamicProgramming
2	5,012	6,799
3	5,938	8,640
4	7,685	12,948
5	14,858	30,899
6	38,930	80,721
7	189,054	219,780
8	1262,430	636,520
9	10369,797	1762,445
10	91116,343	4898,039
11	937262,039	12828,085
12	10420416,470	34291,777
13	125788292,400	86716,926
14	1701258038,925	269159,411
15		630830,132
16		1564664,233
17		3709071,141
18		9254358,068
19		21849331,954
20		60587376,570
21		130638192,728
22		273842482,243

Na wykresie w celu uzyskania większej przejrzystości usunięty został rekord 14 miast dla przeglądu zupełnego, ponieważ jego wartość czasu zbyt odstawała od reszty(ponad 6x większa niż najwyższa zarejestrowana wartość dla programowania dynamicznego)



Do generowania grafów z miastami użyty został skrypt w języku python3

```python
#!/usr/bin/python3
import os
import sys
from random import randint

filename='testDataX.txt'
projectDir='/home/kuba/Qt/build-PEA-Desktop_Qt_5_7_0_GCC_64bit-Profile'

filePath='{}/{}'.format(projectDir,filename)
cities=int(sys.argv[1])
maxWeight=10

os.system('echo "{}" > {}'.format(cities,filePath))

for lineNr in range(0,cities):
  line=''
  for i in range(0,cities):
    if i==lineNr:
      weight=0
    else:
      weight=randint(1,maxWeight)
    line='{}{} '.format(line,weight)
  os.system('echo "{}" >> {}'.format(line,filePath))
```

Wnioski:

Algorytm programowania dynamicznego wielokrotnie redukuje czas wykonywania się programu, jednakże powoduje też duże obciążenie pamięci, widoczne przy testach(urządzenie coraz ciężej reagowało aż w końcu zawiesiło się kompletnie)