WYDZIAŁ ELEKTRONIKI
POLITECHNIKA WROCŁAWSKA

PROJEKTOWANIE EFEKTYWNYCH ALGORYTMÓW

Jednoprocesorowy problem szeregowania zadań przy kryterium minimalizacji ważonej sumy opóźnień zadań

Metoda programowania dynamicznego

Autor:
Łukasz Broll
225972

Prowadzący:
dr inż. Mariusz Makuchowski

Termin zajęć:
czwartek 11^{15}-13^{00}

WROCŁAW, 2017r.

Spis treści

1. Informacje wstępne

Należy zaimplementować algorytm metaheurystyczny dla jednego z wybranych problemów oraz wykonać testy polegające na pomiarze czasu działania algorytmu w zależności od wielkości instancji oraz jakości dostarczanych rozwiązań. Należy porównać rozwiązanie dostarczone przez algorytm z najlepszymi znanymi rozwiązaniami dla przykładów testowych.

1.1 Teoretyczny opis problemu szeregowania zadań

Problem ten może być opisany następująco: Danych jest n zadań (o numerach od 1 do n), które mają być wykonane bez przerwań przez pojedynczy procesor mogący wykonywać co najwyżej jedno zadanie jednocześnie. Każde zadanie j jest dostępne do wykonania w chwili zero, do wykonania wymaga pj > 0 jednostek czasu oraz ma określoną wagę (priorytet) wj > 0 i oczekiwany termin zakończenia wykonania dj > 0. Zadanie j jest spóźnione, jeżeli zakończy się wykonywać po swoim terminie dj , a miarą tego opóźnienia jest wielkość Tj = max(0, Cj , −dj), gdzie Cj jest terminem zakończenia wykonywania zadania j. Zadanie polega na znalezieniu takiej kolejności 1 wykonywania zadań (permutacji), aby zminimalizować kryterium TWT.

Źródło: http://www.zio.iiar.pwr.wroc.pl/pea/pea_projekt_1718.pdf

1.2 Teoretyczny opis zadania programowania dynamicznego

W algorytmie opartym na programowaniu dynamicznym rozwiązuje się każdy podproblem tylko raz, po czym zapamiętuje się wynik w odpowiedniej tabeli, unikając w ten sposób wielokrotnych obliczeń dla tego samego podproblemu. Programowanie dynamiczne jest zwykle stosowane do problemów optymalizacyjnych. W tego typu zagadnieniach możliwych jest wiele różnych rozwiązań. Z każdym rozwiązaniem jest związana pewna liczba (koszt). Rozwiązanie optymalne to takie, które ma optymalny (minimalny lub maksymalny) koszt. Może być wiele rozwiązań optymalnych, wszystkie o tym samym optymalnym koszcie.

Proces projektowania algorytmu opartego na programowaniu dynamicznym można podzielić na cztery etapy:

1) Scharakteryzowanie struktury optymalnego rozwiązania.

2) Rekurencyjne zdefiniowanie kosztu optymalnego rozwiązania.

3) Obliczenie optymalnego kosztu metodą wstępującą, czyli rozpoczynając od najmniejszych podproblemów, rozwiązywać coraz większe, wykorzystując zapamiętane rozwiązania mniejszych.

4) Konstruowanie optymalnego rozwiązania na podstawie wyników wcześniejszych obliczeń.

Źródło: http://www.cs.put.poznan.pl/arybarczyk/TeoriaAiSD3.pdf

2. Implementacja

Aplikacja konsolowa zaprojektowana do rozwiązania problemu została wykonana w języku C++ w środowisku Microsoft Visual Studio 2015. Projekt oparty jest o 2 klasy oraz bazową metodę main():

2.1 Klasa `tasks`

Podstawowa struktura do zarządzania pojedynczym zadaniem. Przechowuje zmienne prywatne `_czas`, `_waga` oraz `_termin` każdego zadania oraz ich gettery. Umożliwia dodawanie kolejnych zadań oraz ich drukowanie. Zadania przechowywane są w tablicy `tasksT`.

2.2 Klasa `dynamic`

Klasa dziedzicząca po klasie `tasks`, odpowiedzialna za główne funkcjonalności aplikacji. Dodatkowo system pomiaru czasu został zaimplementowany wewnątrz klasy. Znajdują się w niej wszystkie niezbędne tablice, metody oraz zmienne, które szczegółowo zostały opisane w nagłówku klasy (`dynamic.h`).

2.3 Metoda `main()`

Zadaniem metody jest jedynie komunikacja z użytkownikiem. Znajduję się w niej nieskończona pętla menu funkcji dostępnych dla użytkownika oraz krótka informacja o programie.

3. Metoda programowania dynamicznego

Główna metoda całego projektu została zaprogramowana zgodnie z głównymi zasadami programowania dynamicznego przedstawionymi w punkcie 2. Cały algorytm działa w jednej pętli, która zawiera w sobie dwie mniejsze. Jako, że liczba permutacji wynosi dokładnie 2 do potęgi liczby zadań, główna pętla wykonuje się dokładnie tyle razy.

Pierwsza z wewnętrznych pętli została wykorzystana do permutacji zadań w systemie binarnym. Przy każdym obiegu pętli pierwsza wewnętrzna pętla konwertuje zapis dziesiętny iteracji do tablicy binarnej, w której każda komórka odpowiada za kolejne zadania (1 – zadanie się wykonuje, 0 – zadanie jest pomijane). Jedocześnie obliczany jest maksymalny czas potrzebny na wykonanie wszystkich zadań. Takie rozwiązanie permutacji, w przeciwieństwie do bitowej permutacji na zmiennych, nie ma ograniczenia w ilości zadań (maks. 64 bity), jest wydajne pamięciowo oraz nie generuje dużych opóźnień czasowych.

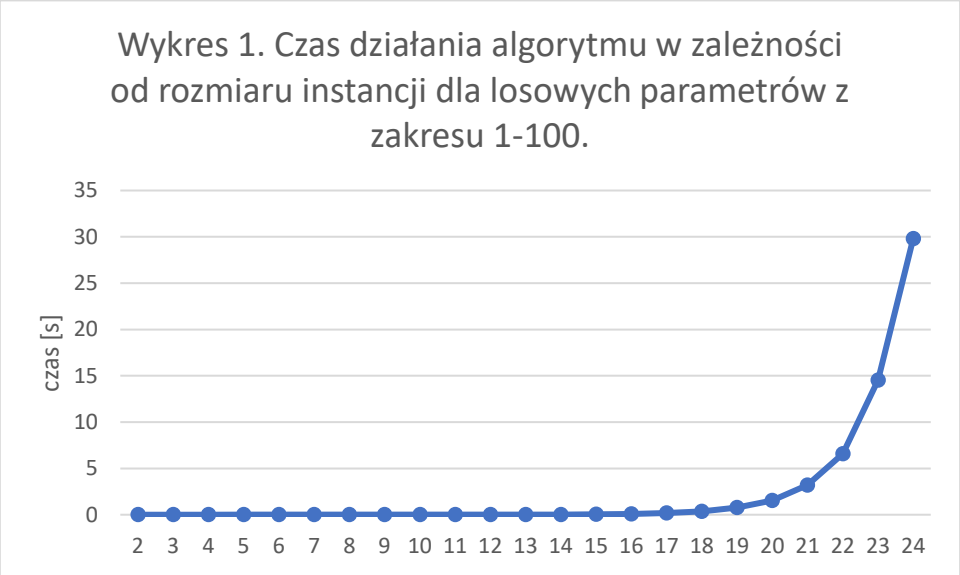
Druga z wewnętrznych pętli, to właściwe programowanie dynamiczne. Najpierw obliczany jest indeks tablicy wyników bez zadania, które będzie właśnie rozpatrywane. Indeks jest potrzebny do znalezienia wyniku w tablicy już rozpatrzonych przypadków. Jeśli w tablicy permutacji znajduje się 1 w danym indeksie, obliczany jest koszt wykonania zadania (iloczyn czasu i wagi). Następnie, jeśli koszt zadania i suma kosztów reszty zadań jest mniejsza niż maksymalny termin, to obliczony koszt opóźnienia wynosi 0. W przeciwnym wypadku, jeśli aktualne zadanie mieści się w terminie, ale koszt opóźnienia istnieje, to znaczy, że opóźniona jest suma innych zadań. Natomiast jeśli koszt opóźnienia wynika również z opóźnienia liczonego zadania, to znaczy, że należy zsumować koszt innych zadań z kosztem zadania wykonywanego w tej iteracji. To wszystkie warunki, które należało sprawdzić przed zapisaniem wyniku. W kolejnych pętlach należy rozpatrzyć koszty innych zadań oraz wybrać ten zestaw, który będzie generował najmniejszy koszt i zapisać jego wartość oraz numer zadania. Dzięki temu, w kolejnych permutacjach baza wyników zawsze będzie zawierała już policzone zestawy, co jest znacznie szybsze niż liczenie ich od nowa. Zapis numeru zadania z optymalnego rozwiązania umożliwi zapisanie kolejności w jakiej mają być wykonane zadania, a nie tylko wyniku z najmniejszym kosztem ich wykonania.

4. Test wydajności

Przeprowadzone zostały testy wydajności aplikacji w zależności od rozmiaru instancji (ilości zadań) oraz ich parametrów. Niestety testy na danych z plików, zaproponowanych w instrukcji projektu nie zostały zrealizowane. Najmniejsza instancja w plikach wynosiła 40 zadań, co daje 2^40 pośrednich wyników, które należy przechować w pamięci. Niestety przyjęty sposób zapisu pośrednich wyników w tablicy dynamicznej uniemożliwia utworzenie tak dużej struktury w pamięci RAM na domowym sprzęcie.

Przed przystąpieniem do pomiarów w aplikacji wyłączone zostały wszystkie komendy drukujące napisy na ekran. Program został uruchomiony w wersji Release x64 przy minimalnych obciążeniu systemu. Specyfikacja sprzętu, na którym przeprowadzone testy jest następująca:

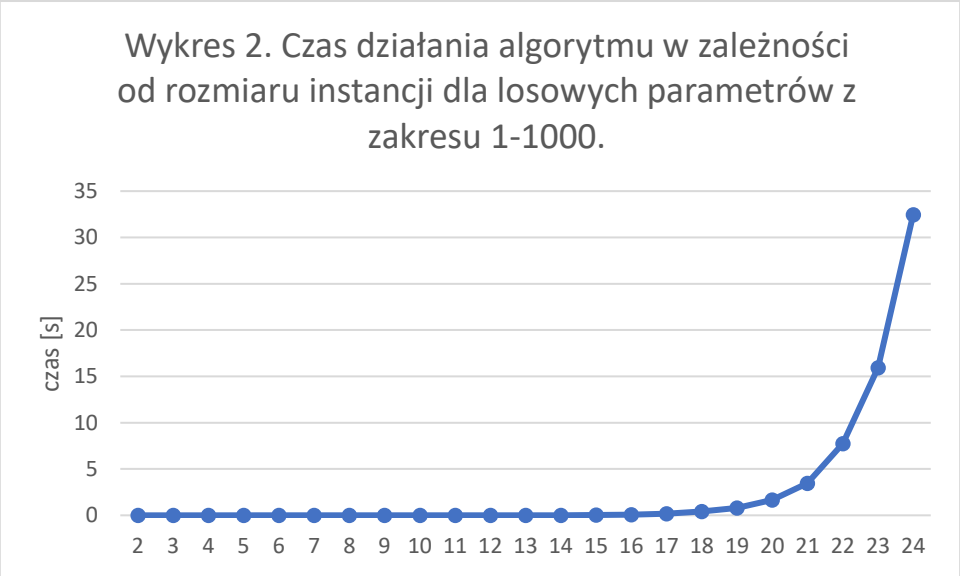
- Procesor: Intel® Core™ i5-2410M CPU @ 2.30Ghz
- RAM: 8.00 GB
- System: Windows 10 x64
- Dysk: SSD GOODRAM IRIDIUM PRO 240GB 560MB/S SATA III



Liczba zadań	Średnia z Czas [ms]	Średnia z Czas [s]
2	0.00308	0.00000308
3	0.00616	0.00000616
4	0.00866	0.00000866
5	0.01715	0.00001715
6	0.03866	0.00003866
7	0.08274	0.00008274
8	0.18570	0.00018570
9	0.40563	0.00040563
10	0.82816	0.00082816
11	1.74049	0.00174049
12	3.70433	0.00370433

Liczba zadań	Średnia z Czas [ms]	Średnia z Czas [s]
13	8.32251	0.00832251
14	17.01156	0.01701156
15	35.83244	0.03583244
16	78.46561	0.07846561
17	172.61858	0.17261858
18	353.37468	0.35337468
19	780.49170	0.78049170
20	1544.57780	1.54457780
21	3206.38020	3.20638020
22	6584.42080	6.58442080
23	14536.17000	14.53617000
24	29790.41400	29.79041400

Tabela 1. Tabela rozwiązań dla parametrów zadań z przedziału 1-100



Liczba zadań	Średnia z Czas [ms]	Średnia z Czas [s]
2	0.00225	0.00000225
3	0.00390	0.00000390
4	0.04477	0.00004477
5	0.01978	0.00001978
6	0.04141	0.00004141
7	0.09665	0.00009665
8	0.21081	0.00021081
9	0.42278	0.00042278
10	0.89465	0.00089465
11	1.85263	0.00185263
12	4.00923	0.00400923
13	9.01702	0.00901702
14	18.06483	0.01806483
15	38.52561	0.03852561
16	85.60279	0.08560279
17	189.04574	0.18904574
18	408.63388	0.40863388
19	786.60164	0.78660164
20	1646.29160	1.64629160
21	3476.57040	3.47657040
22	7755.10620	7.75510620
23	15940.63800	15.94063800
24	32462.22500	32.46222500

Tabela 2. Tabela rozwiązań dla parametrów zadań z przedziału 1-1000

5. Wnioski

Przygotowanie projektu pozwoliło na zapoznanie się z tematyką algorytmiki metaheurystycznej. Niestety, wykorzystanie tablicy dynamicznej w pamięci RAM do przechowywania pośrednich wyników znacząco ograniczyło możliwości aplikacji do obliczania instancji powyżej 30. zadań. Mimo to, algorytm programowania dynamicznego jest w stanie poradzić sobie z problemem NP-trudnym, co niewątpliwie jest jego dużą zaletą. Oprócz tego analizując otrzymane rezultaty dają się zauważyć pewne prawidłowości:

- Czas wykonywania algorytmu zwiększa się razem z liczbą instancji. Do 10 zadań algorytm wykonuje się niemal błyskawicznie. Razem ze wzrostem liczby instancji, wykładniczo wzrasta czas działania algorytmu.
- Losowość parametrów zadania ma mały wpływ na czas działania algorytmu. Różnice w losowaniu parametrów z puli 1-1000 oraz z puli 1-100 są znikome i wynikają prawdopodobnie z konieczności przetworzenia większej liczby przez sprzęt.

W związku z tym, że programowanie dynamiczne było pierwszym zadaniem całego projektu, nie ma możliwości porównania wyników i czasu działa z pozostałymi algorytmami. Jednoznacznie można stwierdzić jedynie, że jest to algorytm bardzo szybki, w porównaniu do algorytmów badanych w poprzednim semestrze.