Piotr Giedziun 184731 Termin: Pn/TP - 15:15

Projektowanie efektywnych algorytmów
Labolatorium

Problem szeregowania zadań na dwóch procesorach z wykorzystaniem schematu aproksymacyjnego i programowania dynamicznego jako algorytmu optymalnego

Prowadzący:
mgr. inż. Karolina Mokrzysz

1. **Opis problemu**

   Celem ćwiczenia była implementacja algorytmów schematu aproksymacyjnych PTAS i FPTAS.

2. **PTAS - Wielomianowy schemat aproksymacyjny**

   Algorytm aproksymacyjny, który pozwala na uzyskanie dowolnie dobrego rozwiązania przybliżonego danego problemu optymalizacyjnego, i którego złożoność czasowa jest wielomianowa dla każdej żądanej dokładności.

$$TIME(A_\epsilon) = O\left(p(N(I)) \cdot e\left(\frac{1}{\epsilon}\right)\right),$$

   Rys. 1 - czas działania algorytmu

   Parametry:
   n - ilość zadań rozwiązywanych przez DP (programowanie dynamiczne) dane wzorem $\frac{1-2\epsilon}{\epsilon}$

3. **FPTAS - W pełni wielomianowy schemat aproksymacyjny**

   Algorytm aproksymacyjny, który pozwala na uzyskanie dowolnie dobrego rozwiązania przybliżonego danego problemu optymalizacyjnego, i którego złożoność czasowa jest wielomianowa względem rozmiaru instancji rozwiązywanego problemu i rośnie wielomianowo w miarę wzrostu żądanej dokładności.

$$TIME(A_\epsilon) = O\left(p\left(N(I), \frac{1}{\epsilon}\right)\right),$$

   Rys. 2 - czas działania algorytmu

   Parametry:
   n - ilość zadań rozwiązywanych przez DP (programowanie dynamiczne) dane wzorem $\frac{1-2\epsilon}{\epsilon}$

   k - współczynnik dzielenia długości zadania - dane wzorem $k = \frac{\epsilon S}{2\pi}$ WZ
   S - suma długości wszyskich wykonywanych zadań

4. **Implementacja**

Program został zaimplementowany w języku Python (wersja 2.7.x).

W celu prawidłowego działania należy przypisać prawa do wykonywania wszyskim pliką o rozszerzeniu .py (przykładowo poprzez "chmod +x *.py"). Jest to niezbędne, z punktu widzenia aplikacji odpowiedzialnej za zarządzanie podprogramami.

Składa się z 3 programów:

- **app.py** - algorytm rozwiązujący P2||Cmax dla danych wejściowych
- **gen.py** - algorytm generujący dane wejściowe dla zadanego zakresu czasu oraz ilości zadań
- **test.py** - worker, mający na celu zarządzanie app.py oraz gen.py, steruje ich wejściami w celu wygenerowania oraz obrobienia outputu. Wynikiem działania programu są dane, które niezbędne były do sporządzenia tego sprawozdania.

Program abstachuje zadanie, w celu przejżystości kodu oraz możliwości operacji na danych jako zbiorze.

```
class Task:
    def __init__(self, id, time):
        self.id = id+1
        self.time = time
```

Kod 1. Klasa Task.

Tworzenie tabeli, niezbędnej do uzyskania wyniku odbywa się poprzez rekurencję. Jako, iż python nie wymaga alokowania pamięci struktura tabeli tworzona jest dynamicznie, linia po linii. Pozwala to na oszczędności zwiazane z przepisywaniem wartośći na niższe wiersze tabeli, w przypadku tego algorytmu kolejna linia powstaje z poprzedniej, następnie wykonywany jest na nej algorytm przypisania zadania.

Następnie zadania sortowane są majaląco wg. kryterium czasu działania.

```
# sortuj po czasie
tasks = sorted(tasks, key=lambda x: x.time, reverse=True)
```

Kod 2. Algorytm sortujący zadania

Algorytm DP (programowania dynamicznego) przeprowadzany jest tylko dla n pierwszych zadań. Parametr n wyliczany jest wzorem podanym w Kod. 3.

```
# ile zadań
n = int((1-2*e)/e))
```

Kod 3. Algorytm wyliczający ilość zadań obliczanych przez DP

```
def rec(tasks, lines):
    # populacja nowej linii, poprzez ostatnią
    new_line = list(lines[-1])

    # algorytm przypisujący czasy dla obecnej linii

    # warunek końca rekurencji
    if len(lines) > len(tasks):
        return lines

    return rec(tasks, lines)
```

Kod 4. Fragment kodu ilustrujący tworzenie oraz uzupełnianie tabeli.

Po stworzeniu oraz uzupełnieniu tabeli kolejnym krokiem jest backtracking. Został on rozwiazany również przez rekurencję,

```
# output - rezultat wypracowany przez pierwszą funkcję
# tasks - lista wszystkich zadań
# pos - pozycja (default: podłoga z łączny czas/2)
# result - lista zadań wykonana na jednym z procesorów
def back(output, tasks, pos, results):
    # szuka pierwszego wystąpnia "T"
    for line_id in range(0, len(output)):
        if output[line_id][pos] ==  "T":
            # dodaje zadanie do wyniku
            results.append(tasks[line_id-1])
            # kończy działanie
            if pos-tasks[line_id-1].time <= 0:
                return results

    return back(output, tasks, pos-tasks[line_id-1].time, results)
```

Kod 5. Fragment obliczający zadania wykonane na jednym z procesorów.

Efektem tego jest zbiór (truple) zadań (obiektów) które zostaną wykonane na pierwszym procesorze. Dzięki temu, iż zadania są obiektami możemy zidentyfikować jakie obiekty pozostały (wykonane zostaną na procesorze drugim).

Dzięki temu, iż python jest językiem matemtycznym w prosty sposób obliczamy dopełnienie zbioru.

```
# zadania wykonane na procesorze 2
# p1 - zadania wykonane na 1-wszym procesorze
# tasks - wszystkie zadania
p2 = list(set(tasks[n:])-set(p1))
```

Kod 4. Fragment ilustrujący obliczenie zadań dla 2-giego procesora.

Następnie obliczamy czas działania zadań dla procesora 1 i 2. Pozostałe zadania szeregowane są algorytmem LSA (list scheduling algorithm). Zasada jego działania jest trywialna - kolejne zadanie przypisuje procesorowi, który pracuje najkrócej.

Następnie w zależności od atrybutu, z którym został uruchomiony program wykonywana jest prezencja wyniku.

- -output_html **(default)**

  Wynikiem działania programu jest wizualne przedstawienie zadań podzielonych na dwa procesory, tabeli niezbędnej do wygenerowania wyniku oraz danych checujących procesory (ilość czasu na poszczególnym procesorze itp.)

- -output_json

  Wynikiem jest tekst, który jest sformatowany dla test.py, w celu prowadzenia testów.

5. **Przykład działania**

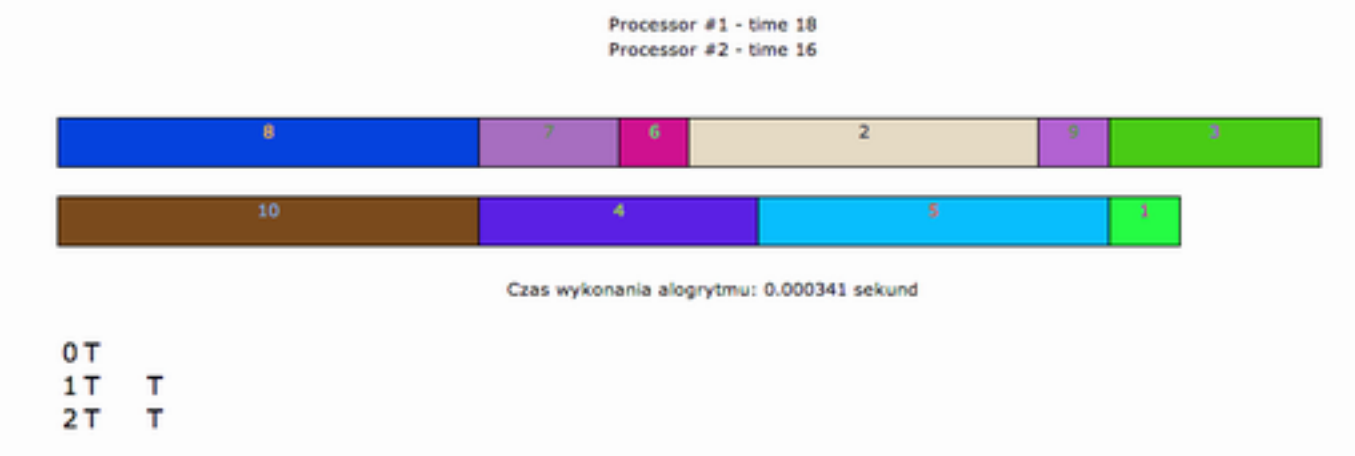
   Pierwszym krokiem jest wygenerowanie danych wejściowych poleceniem "./gen.py [ilość zadań] [max_przedziału_czasowego]"

   ./gen.py 10 40

   Następnym krokiem jest uruchomienie algorytmu, którego rezultatem jest graficzne przedstawienie rozwiązania. "./gen.py -e [maksymalny błąd] [fptas]" - parametr decydujacy (fptas/ptas)

   ./app.py -e 0.2

   Program metodą programowania dynamicznego oblicza wyniki, następnie prezentuje go graficznie w przeglądarce. (Program automatycznie przekazuje rezultat do domyślnej przeglądarki polceniem "open")

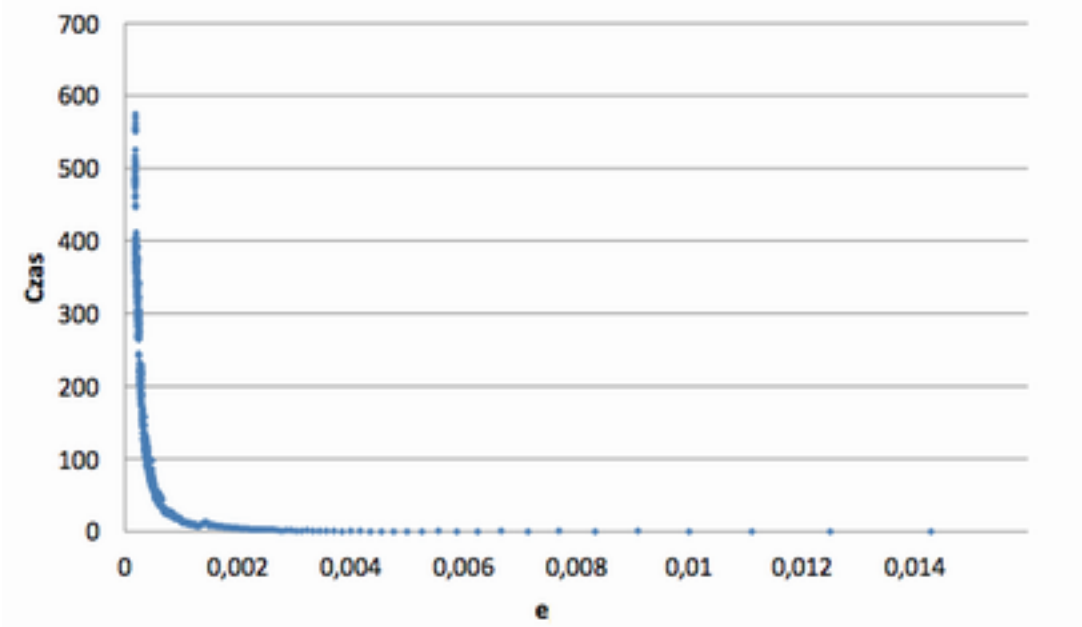


Rys 1. Działanie programu dla instancji 10 elementowej (czasy 1-10 sek)

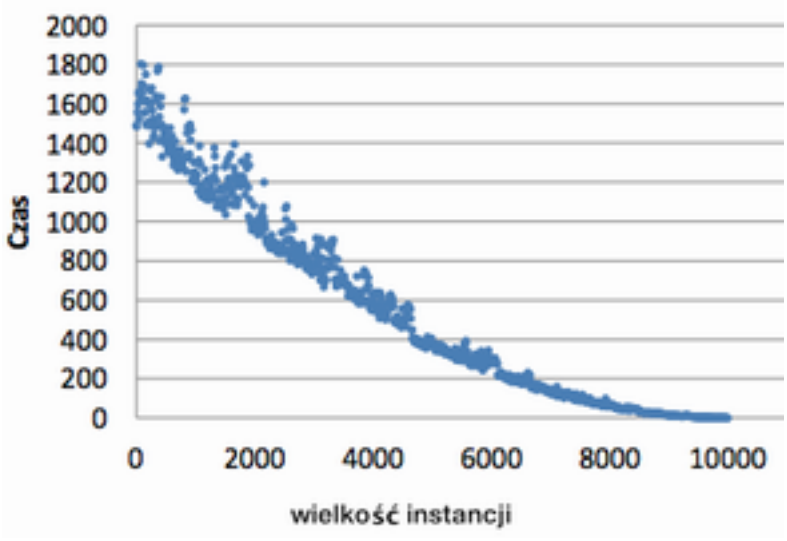
6. **Wyniki**

   a. **Seria #1**
      i. **Ilość zadań:** 10
      ii. **Przedział czasowy:** 1-10
      iii. **Błąd:** e = 0.02
      iv. **Dane:** 1, 5, 3, 4, 5, 1, 2, 6, 1, 6
      v. **Wynik:**
      Procesor 1: 18, Procesor 2: 16

b. **Seria #2**
   i. **Ilość zadań:** 10
   ii. **Przedział czasowy:** 1-100
   iii. **Dane:** 45, 27, 95, 91, 63, 37, 39, 73, 56, 13
   iv. **Wynik:**
   Procesor 1: 284, Procesor 2: 255



Wykres 1 - rozkład czasu od błędu



Wykres 2 - czas od wielkości instancji

7. **Wnioski**

    a. Algorytm działa poprawnie, jego poprawność została przetestowana na zbiorach z poprawnymi rozwiązaniami
    b. Wielomianowy schemat aproksymacyjny pozwala na znaczne przyspieszenie działania algorytmu przy nie wielkim wpływie na błąd według podziału optymalnego