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

Projektowanie efektywnych algorytmów

Labolatorium

P2||Cmax - szeregowanie zadań na 2 identycznych procesorach

Prowadzący:
mgr. inż. Karolina Mokrzysz

1. **Opis problemu**

   Problem podziału zadań na dwa jednakowe procesory z kryterium Cmax, jest trywailnym przypadkiem problemu plecakowego.
   P2||Cmax polega na rozdzieleniu zadań na dwa procesory, tak aby całkowity czas wykonywania zadań był jak najkrótszy. W celu rozwiązania tego problemu wykorzystuje się programowanie dynamiczne, inicjaluzjąc dwu wymiarową tablicę o rozmiarze ZxP, gdzie Z - to ilość zadań; P - podłoga z łącznego czasu wszystkich zadań podzielonego na dwa.

2. **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.

```
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 2. 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, gdy pozostały czas to 0 lub
mniej
            if pos-tasks[line_id-1].time <= 0:
                return results

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

Kod 3. 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)-set(p1))
```

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

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.

3. **Przykład działania**

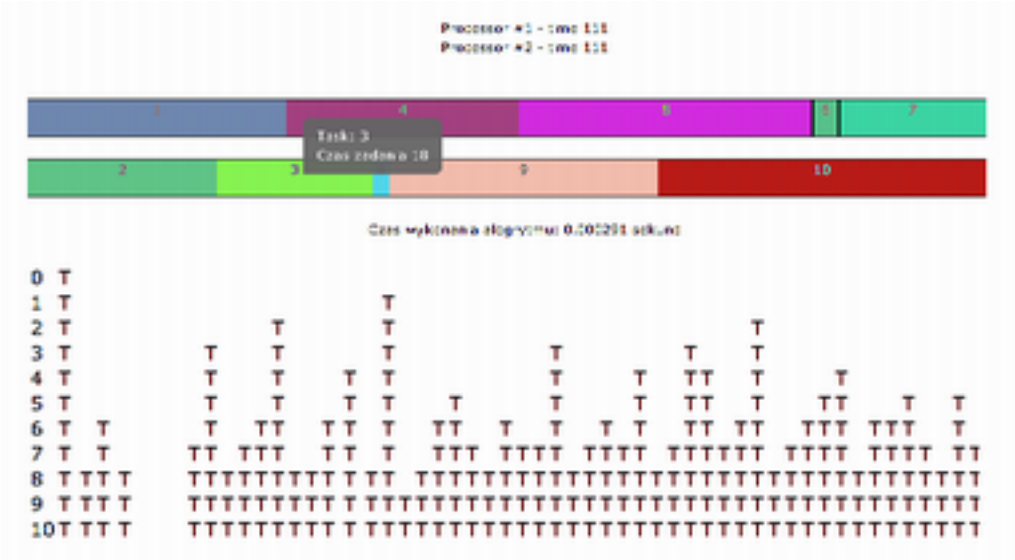
Pierwszym krokiem jest wygenerowanie danych wejściowych poleceniem "./gen.py [ilość zadań] [max_przedziału_czasowego]"

./gen.py 10 40

Następnym krokiem jest uruchomienie algorytmu, którego rezultatem jest graficzne przedstawienie rozwiązania.

./app.py

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.

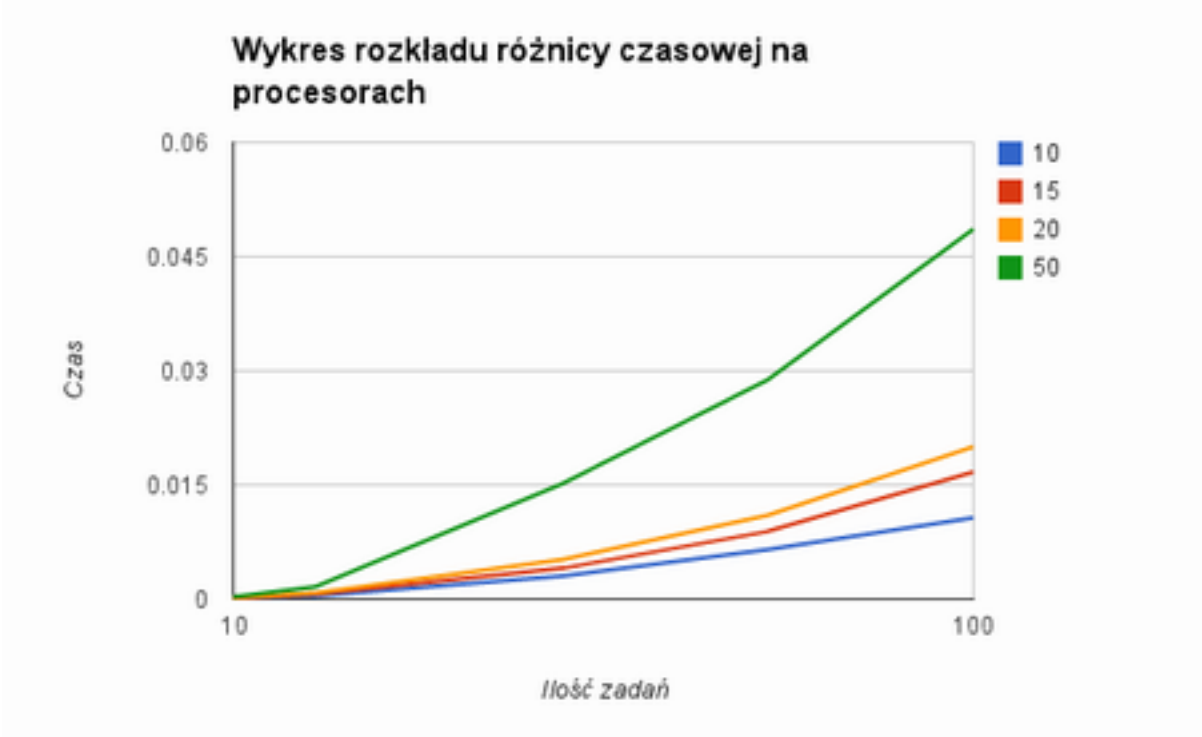
4. **Wyniki**

   a. **Seria #1**
      i. **Ilość zadań:** 10
      ii. **Przedział czasowy:** 1-10
      iii. **Dane:** 7, 2, 6, 6, 1, 6, 6, 9, 5, 9
      iv. **Wynik:**
      Procesor 1: 27, Procesor 2: 28

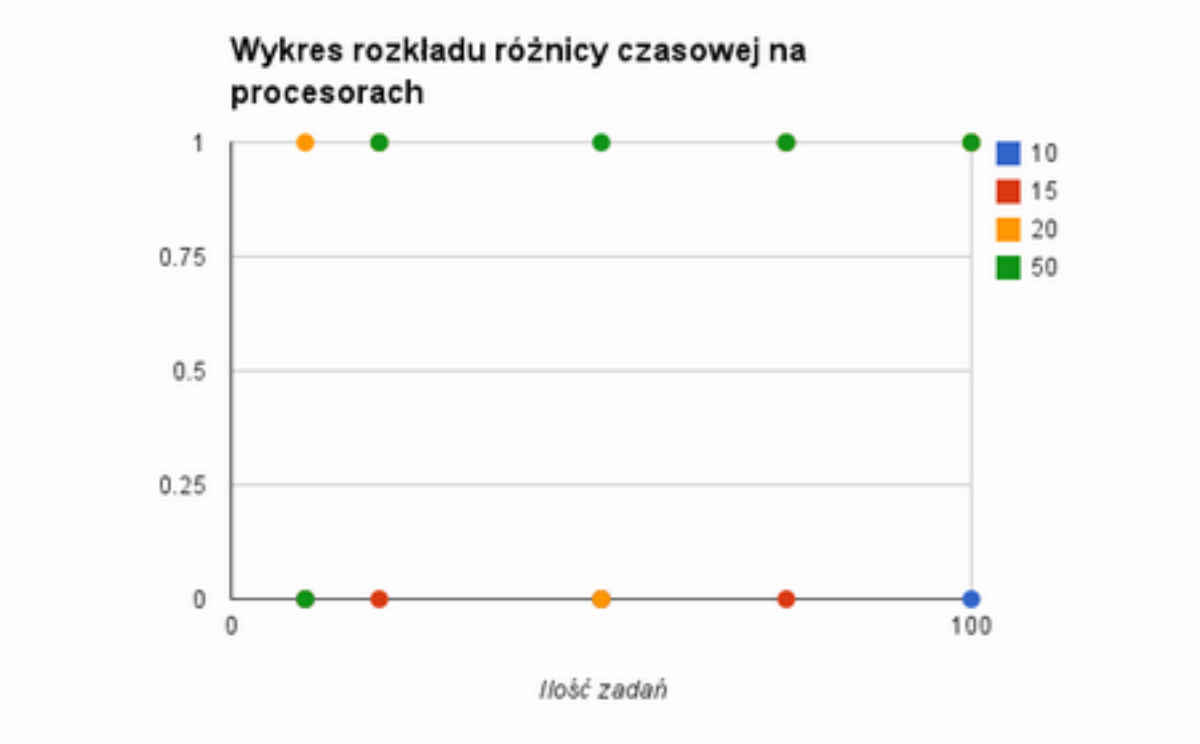
   b. **Seria #2**
      i. **Ilość zadań:** 10
      ii. **Przedział czasowy:** 1-100
      iii. **Dane:** 86, 58, 79, 20, 73, 37, 87, 72, 57, 10
      iv. **Wynik:**
      Procesor 1: 289, Procesor 2: 290

Ilość zadań	Przedział (do)	Proc 1	Proc 2	delta	Czas
10	10	30	30	0	0.000253
10	15	35	35	0	0.00012
10	20	59	60	1	0.000198
10	50	133	133	0	0.000292
20	10	53	54	1	0.000419
20	15	90	90	0	0.000653
20	20	117	118	1	0.000807
20	50	261	262	1	0.001627
50	10	156	156	0	0.003021
50	15	207	207	0	0.004069
50	20	290	290	0	0.005193
50	50	743	744	1	0.01515
75	10	205	206	1	0.006511
75	15	288	288	0	0.008918
75	20	350	351	1	0.011017
75	50	1001	1002	1	0.02882
100	10	269	269	0	0.01069
100	15	414	415	1	0.016702
100	20	526	527	1	0.020026
100	50	1280	1281	1	0.048613

Tab 1. Wyniki przeprowadzone dla zadań [10,20,50,75, 100] oraz przedziałów czasowych [10,15,20,50].



Wykres 1. Ilustracja czasowego przebiegu zadań z tabeli.



Wykres 2. Ilustracja rozkładu różnicy czasu pracy procesorów dla danych z tabeli.

5. **Wnioski**

    a. Algorytm działa poprawnie, jego poprawność zstała przetestowana na zbiorach z poprawnymi rozwiązaniami
    b. Dla zadanych instancji czasy są bardzo małe, alokacje oraz procesy związane z algorytmem mogą mieć narzut na wynik
    c. Testy zostały przeprowadzone dla zadań [10,20,50,75, 100] oraz przedziałów czasowych [10,15,20,50]
    d. Złożoność algorytmu jest bliska n!, gdzie n to ilość tasków