Politechnika
Wrocławska

Komputerowo Zintegrowane Wytwarzanie- WiTi

Autor: Jan Ploska

Kierunek: Informatyczne Systemy Automatyki

Semestr: VI

Prowadzący: dr. hab. inż. Mariusz Makuchowski

Opis problemu: Podobnie jak dla problemu rpq, jest n zadań, wykonywanych na pojedynczej maszynie, przy czym jednocześnie może być wykonywane co najwyżej jedno zadanie. Zadanie posiadaja parametry

- P – czas pracy na maszynie
- W – waga związana z opóźnieniem
- D – czas po którym wykonanie zadania wiąze się z naliczaniem kary

Zadanie i $\in$ N musi być wykonywane nieprzerwanie przez p czasu. Problem WiTi należy do problemów których złożoność wykracza poza zlożonośc wielomianową.

Implementacja rozwiązania: Zainicjowałem 3 tablice zawierające maski, uszeregowania, posiadające namniejsze kary, najmniejsze kary. Wszystkie tablice posiadają rozmiar 2^n-1.

Przygotowanie mask oraz inicjalizacja pozostałych tablic.

```
def prepareMasks(zadania):
    masks = []
    n = len(zadania)
    bn = int(math.pow(2, n) - 1)

    for i in range(1, bn + 1):
        b = f"{i:0{n}b}"
        b_mask = str(b[::-1])
        masks.append(b_mask)
        fees.append(0)
        schedules.append([])

    return masks
```

Następnie dla każdej maski przygotowuje pod zestaw zadań wyznaczony przez ową maske:

```
for m in masks:
    str_m = str(m)
    sub_zadania = []
    j = 0
    for b in str_m:
        if b == '1':
            sub_zadania.append(zadania[j])
        j += 1
    masks_zadania.append(sub_zadania)
```

Poprzez tabice z tak przygotowanymi podzestawami zadań iteruje oraz obliczam kare dla podzestawu zadań z tylko jednym zadaniem:

```
for i in range(len(masks_zadania)):
    s = masks_zadania[i]
    if len(s) == 1:
        task = s[0]
        fees[i] = max(task.p - task.d, 0) * task.w
        schedules[i] = [task.id]
```

W przypadku gdy podzestaw zadań zawiera ich więcej każde możliwe zadanie umieszczam na końcu uszeregowania. Człon poprzedzający to zadanie został już wcześniej policzony, wieć odnajduję tą kolejność, odczytuje jej kare i obliczam kare dla ostatniego zadania. Wykunuej to dla każdego zadania i zapisuje uszeregowanie oraz kare dla przypadku z najmniejszą karą

```
    else:
        min_fee = math.inf
        best_schedule = []
        for k in range(len(s)):
            temp_s = s[:k] + s[k+1:]
            last_element = s[k]

            czlon_fee, czlon_schedule = find_fee_and_schedule(temp_s)
            czlon_time = sum(task.p for task in temp_s)

            last_element_fee = max(czlon_time+last_element.p-last_element.d,0)*last_element.w
            total_fee = czlon_fee + last_element_fee

            if total_fee < min_fee:
                min_fee = total_fee
                best_schedule = czlon_schedule + [last_element.id]

        fees[i] = min_fee
        schedules[i] = best_schedule

return schedules[-1], fees[-1]
```

Przykładowe Wyniki:

- Data10: Minimalna kara: 766 Najlepsze uszeregowanie: [6, 9, 2, 5, 10, 7, 4, 8, 3, 1]
- Data11: Minimalna kara: 799 Najlepsze uszeregowanie: [6, 9, 2, 11, 5, 7, 10, 4, 8, 3, 1]
- Data12: Minimalna kara: 742 Najlepsze uszeregowanie: [6, 9, 2, 11, 5, 12, 10, 7, 4, 8, 3, 1]