# Lista 2

Szymon Lewandowski

## Zadanie 1

Celem zadania jest znalezienie takiego sposobu rozcinania desek, aby zaspokoić potrzeby na deski o danych szerokościach oraz wytworzyć minimalną liczbę odpadów (desek ponad zapotrzebowania oraz resztek, z których nie da się wyciąć danych desek).

Przed przygotowaniem modelu następuje obliczenie wszystkich możliwych sposobów pokrojenia desek.

Dane:

- wektor wymaganych szerokości desek

- wektor zapotrzebowań na poszczególne szerokości desek

- standardowa szerokość deski (w calach)

Dodatkowo po wstępnych obliczeniach mamy także wszystkie sposoby pokrojenia deski na wymagane szerokości, włącznie z odpadami pozostałymi po każdym sposobie dzielenia.

Przykładowo, podział [i, j] oznacza ilość desek szerokości takiej jak j-ta wymagana szerokość, utworzonych przy podziale i

Zmienne:

- x[] >= 0 – wektor wielkości takiej, jak wektor możliwych podziałów, mówiący, ile desek musi zostać podzielonych danym podziałem

Ograniczenia:

- suma ilości desek dla każdej wymaganej szerokości powinna być większa lub równa zapotrzebowaniu na daną szerokość

Funkcja celu:

Suma wszystkich odpadów i desek ponad zapotrzebowanie -> min

Wyniki:

Dla standardowej szerokości deski = 22, wymaganych szerokości = [3, 5, 7] oraz zapotrzebowaniach = [80, 120, 110], wynikiem są podziały:

9 x ([5, 0, 1] i 0 odpadów)

37 x ([1, 1, 2] i 0 odpadów)

28 x ([0, 3, 1] i 0 odpadów)

Nadmiarowe deski tworzą 18 cali odpadów (mniej niż jedna standardowa deska)

## Zadanie 2

Celem tego zadania jest znalezienie kolejności n zadań, tak aby czas ukończenia każdego zadania był możliwie najmniejszy, przy czym zadania mają wagi (ważniejsze zadanie powinno być zrobione przed mniej ważnym, jeśli to możliwe).

Dane:

- czas potrzebny na wykonanie zadań

- wagi zadań

- czasy gotowości zadań do rozpoczęcia obliczeń

Dodatkowo jest stała bigNum, duża wartość

Zmienne:

- startTimes[i] – czasy rozpoczęcia zadania i

- endTimes[i] – czasy zakończenia zadania i

- after[i, j] – jeśli TRUE, zadanie j nastąpiło później niż zadanie i

Ograniczenia:

- startTimes[i] >= readyTimes[i] – czas rozpoczęcia musi być większy niż minimum dla danego zadania

- endTimes[i] == startTimes[i] + times[i] – czas zakończenia to czas rozpoczęcia + czas działania

- startTimes[i]-endTimes[j]+bigNum*after[i, j] >= 0 – jeśli czas rozpoczęcia i jest po czasie zakończenia j, to after[i, j] = FALSE

- startTimes[j]-endTimes[i]+bigNum*(1-after[i, j]) >= 0 – przeciwnie do poprzedniego

Funkcja celu:

Suma iloczynów zakończenia zadania oraz wagi zadania -> min

Wyniki:

Dla czasów = [1, 3, 2, 5], wag [1, 7, 3, 2] oraz czasów gotowości [0, 2, 1, 1], wynikiem są czasy zakończenia (1 -> 1, 2 -> 6, 3 -> 3, 4 -> 11) oraz wartość funkcji celu = 74

## Zadanie 3

To zadanie jest podobne do poprzedniego, ale tym razem mamy więcej niż jeden procesor, nie ma minimalnych czasów rozpoczęcia zadań oraz niektóre zadania muszą poprzedzać niektóre inne zadania. Szukamy minimalnego czasu skończenia się ostatniego zadania.

Dane:

- liczba zadań n

- liczba maszyn m

- czas potrzebny na wykonanie zadań

- relacja poprzedzania

Dodatkowo, podobnie jak wcześniej, bigNum.

Zmienne:

- ms – czas zakończenia ostatniego zadania

- startTimes[i] >= 0 – podobnie jak wcześniej

- endTimes[i] – podobnie jak wcześniej

- before[i, j] – podobnie jak wcześniej

- machines[i, j] – jeśli TRUE, to zadanie i zostaje wykonane na maszynie j

- sameMachine[i, j] – jeśli TRUE, to zadania i oraz j są wykonywane na tej samej maszynie

Ograniczenia:

- jeśli zadanie i ma być przed zadaniem j, to before[i, j] = 1, before[j, i] = 0 oraz startTimes[j] >= endTimes[i]

- endTimes[i] == startTimes[i] + Times[i] – oczywiste

- sum(machines[i, 1:m]) == 1 – zadanie i zostanie wykonane na dokładnie jednej maszynie

- endTimes[i]-ms <= 0 – funkcja celu jest równa największemu z czasów zakończenia

- machines[j, k]+machines[i,k] <= sameMachine[i, j]+1 – jeśli i oraz j zostały wykonane na maszynie k, to zostały wykonane na tej samej maszynie

- startTimes[i]-endTimes[j]+bigNum*(before[i, j])+(1-precedences[i, j])*bigNum*(1-sameMachine[i, j]) >= 0 – warunek podobny jak w poprzednim zadaniu, z tym że nienachodzenie na siebie zadań jest sprawdzane tylko jeśli zadania działają na tej samej maszynie

- startTimes[j]-endTimes[i]+bigNum*(1-before[i, j])+(1-precedences[i, j])*bigNum*(1-sameMachine[i, j]) >= 0 – jak wyżej

Funkcja celu:

ms -> min

Wyniki:

Dla 3 maszyn, 9 zadań, przy czasach wykonania = [1, 2, 1, 2, 1, 1, 3, 6, 2, 1] oraz relacji poprzedzania = [[1, 4], [2, 4], [2, 5], [3, 4], [3, 5], [4, 6], [4,7], [5, 7], [5, 8], [6, 9], [7, 9]]

Całkowity czas wykonania = 9

|2|2|5| |6| | |9|9|

| | | |8|8|8|8|8|8|

|3|1|4|4|7|7|7| | |

## Zadania 4

Zadanie podobne do poprzedniego, z tym, że teraz zamiast procesorów każde zadanie wykorzystuje surowce oraz ilość i typy surowców są dowolne.

Dane:

- liczba zadań n

- liczba surowców m

- czas potrzebny na wykonanie każdego zadania

- surowce zużywane przez każde zadanie

- maksymalne dozwolę zużycie każdego surowca w jednostce czasu

Zmienne:

- ms – czas zakończenia ostatniego zadania

- endTimes – jak w poprzednim zadaniu

- startTimes – jak w poprzednim zadaniu

- workingAtStart[i, j] – jeśli TRUE, to zadanie j działa w momencie startu zadania i

- startAfterI[i, j] – jeśli TRUE, to startTimes[j] >= startTimes[i]

- startAfterIEnd[i, j] – jeśli TRUE, to startTimes[j] >= endTimes[i]

- endAfterI[i, j] – jeśli TRUE, to endTimes[j] >= endTimes[i]

- endAfterIStart[i, j] – jeśli TRUE, to endTimes[j] >= startTimes[i]

-costs[i, j, k] – suma kosztu dla surowca k wszystkich zadań od 1 do j, które działają w momencie startu zadania i

Tablice startAfterI, startAfterIEnd, endAfterI oraz endAfterIStart będą dalej nazywane tablicami kolejności.

Ograniczenia:

- zostają ustawione stałe ograniczenia dla każdego zadania (czas minimalny czas startu, zależność między startem i końcem, wartości w tablicy kolejności między zadaniem i samym sobą)

- jeśli zadanie I ma być przed zadaniem j, zostawiają ustawione odpowiednie zależności między startTimes oraz endTimes, a także zostają ustawione odpowiednie wartości w tablicach kolejności

- podobnie jak wcześniej, wykorzystujemy jakąś bardzo dużą wartość, aby móc w tablicach kolejności ustawić odpowiednie wartości typu boolean (na przykład „startTimes[i]-startTimes[j]+bigNum*(startAfterI[i, j]) >= 1” oraz „startTimes[j]-startTimes[i]+bigNum*(1-startAfterI[i, j]) >= 0” zapewniają nam, że startAfterI[i, j] jest ustawiane na TRUE, jeśli startTimes[j] >= startTimes[i])

- dla każdej trójki (i, j, k), gdzie i,j są ze zbioru zadań i k jest ze zbioru typów surowców, jest nakładane ograniczenie na koszty:

  - Jeśli j to pierwsze zadanie i zadanie działa w momencie startu zadania i, to koszt wynosi zapotrzebowanie zadania j na surowiec k
  - W przeciwnym wypadku koszt wynosi costs[i, j-1, k] (czyli suma kosztów poprzednich zadań w momencie startu i) + koszt zadania j (jeśli ono działa)

Funkcja celu:

ms -> min

Wyniki:

Dla przykładowego problemu wyniki to:

Koniec ostatniego zadania: 237.0

Czasy rozpoczęcia:

1: 0.0

2: 96.0

3: 50.0

4: 50.0

5: 143.0

6: 118.0

7: 143.0

8: 175.0

Czasy zakończenia:

1: 50.0

2: 143.0

3: 105.0

4: 96.0

5: 175.0

6: 175.0

7: 158.0

8: 237.0