Programowanie w Logice
Globalne ograniczenia kombinatoryczne
(Lista 9)

Przemysław Kobylański

Wstęp

brak

Zadania

Zadanie 1 (5 pkt)

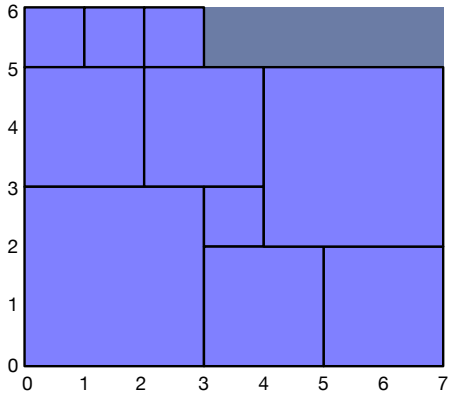
Predykat `tasks(Tasks)` definiuje listę zadań. Każde zadanie opisane jest listą trójelementową, której pierwszym elementem jest czas trwania zadania, drugim elementem jest liczba jednostek pierwszego zasobu potrzebnego do jego wykonania a trzecim jest liczba jednostek drugiego zasobu potrzebnego do jego wykonania:

```
tasks([
    %D  R1 R2
    [2, 1, 3],
    [3, 2, 1],
    [4, 2, 2],
    [3, 3, 2],
    [3, 1, 1],
    [3, 4, 2],
    [5, 2, 1]]).
```

Predykat `resources(Res1, Res2)` definiuje liczbę dostępnych w każdej chwili jednostek pierwszego i drugiego zasobu:

```
%         R1 R2
resources(5, 5).
```

Napisz predykat `schedule(Horizon, Starts, MakeSpan)`, który dla danego horyzontu (koniec dostępnego zakresu czasu), oddaje listę chwil rozpoczęcia zadań `Starts`, tż. wykonując zadania w tych chwilach nie przekroczy się zadanych limitów zasobów oraz termin zakończenia najpóźniej kończącego się zadania `MakeSpan` jest jak najwcześniejszy.



Rysunek 1: Rozcięcie prostokąta na dziesięć kwadratów.

Przykład

Przykładowe zapytanie:

```
?- schedule(20, S, MS).
S = [4, 8, 0, 8, 0, 5, 0],
MS = 11 .
```

Zadanie 2 (3 pkt)

Napisz predykat `kwadraty(Rozmiary, Szerokość, Wysokość, Współrzędne)`, który dla listy n liczb całkowitych dodatnich będących długościami boków n kwadratów (lista `Rozmiary`), zadanej szerokości i wysokości prostokąta, rozstrzyga czy da się te kwadraty wykroić z prostokąta danego rozmiaru. Jeśli odpowiedź jest twierdząca, to oddaje ostatnim argumentem współrzędne rogów kwadratów względem lewego dolnego rogu prostokąta:

$$[x_1, y_1, x_2, y_2, \ldots, x_n, y_n]$$

Przykład

Przykładowe wywołanie:

```
?- kwadraty([1,1,1,1,2,2,2,2,3,3], 7, 6, X).
X = [3,2,0,5,1,5,2,5,3,0,5,0,0,3,2,3,0,0,4,2]
```

Na rysunku 1 przedstawiono powyższe rozwiązanie.

Zadanie 3* (2 pkt)

Na rysunku 2 przedstawiono trzy zestawy danych dla gry `Tetravex`, o której była mowa na 4. wykładzie o poszukiwaniu rozwiązań.

```
tiles(t6a, 6, 6,
    [[0,6,2,4],[2,9,4,7],[2,3,2,0],[6,6,3,0],[1,8,2,0],[9,1,6,1],
     [5,0,0,6],[2,4,8,7],[3,6,4,2],[9,4,2,1],[4,3,9,1],[5,8,8,4],
     [8,0,4,9],[1,9,6,8],[3,9,8,1],[4,0,9,3],[2,7,3,3],[4,6,5,5],
     [4,1,9,0],[3,0,1,3],[4,5,4,3],[6,6,9,8],[1,7,5,7],[3,3,3,6],
     [5,3,0,9],[8,6,2,8],[7,4,6,9],[8,9,5,6],[2,1,5,6],[9,1,6,4],
     [9,4,4,9],[3,9,9,1],[6,3,8,6],[8,1,1,3],[9,8,5,9],[1,2,4,4]]).
tiles(t6b, 6, 6,
    [[1,6,5,7],[9,7,5,8],[2,2,5,4],[5,6,4,4],[6,1,2,3],[0,4,0,7],
     [4,3,8,9],[1,9,1,2],[6,3,4,1],[5,0,2,1],[2,0,1,7],[9,5,5,5],
     [9,6,8,8],[2,8,0,0],[2,4,8,6],[8,7,5,1],[3,3,9,6],[4,6,4,6],
     [2,1,7,9],[5,7,2,7],[7,7,0,4],[6,9,3,9],[4,5,2,6],[8,1,5,8],
     [0,1,9,4],[6,7,7,0],[3,9,5,9],[9,6,1,7],[5,9,3,3],[7,5,6,8],
     [1,8,6,6],[0,4,6,9],[5,1,2,0],[4,7,6,7],[5,9,3,4],[5,3,5,8]]).
tiles(t6c, 6, 6,
    [[9,6,3,9],[9,7,2,3],[7,5,9,8],[7,7,2,4],[6,6,4,3],[0,9,0,5],
     [9,6,2,7],[0,8,2,6],[0,4,8,1],[3,4,8,8],[7,3,5,7],[4,2,4,4],
     [1,6,1,5],[4,1,1,4],[3,0,0,0],[0,7,9,2],[0,1,9,4],[2,3,0,6],
     [1,6,3,1],[9,9,0,0],[2,8,3,7],[4,4,6,1],[8,7,4,0],[8,4,1,3],
     [2,0,4,3],[5,2,8,3],[1,6,0,2],[5,4,3,9],[3,7,6,2],[0,5,3,6],
     [1,5,9,0],[2,2,7,6],[6,3,7,2],[9,1,5,5],[4,2,4,2],[3,0,7,7]]).
```

Rysunek 2: Trzy zestawy danych rozmiaru 6×6

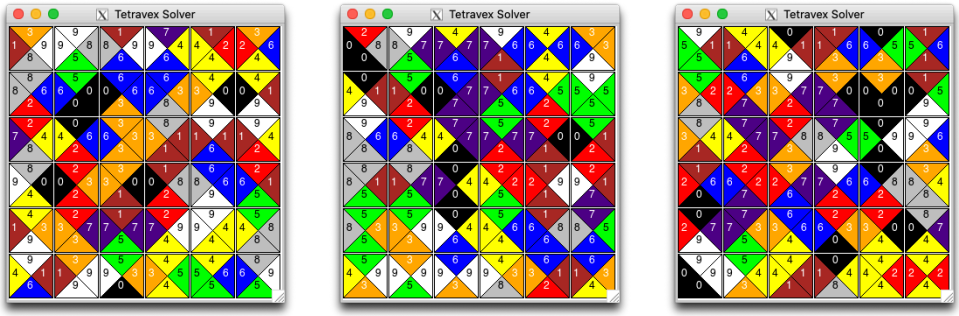
Napisz program w SWI-Prologu, który wykorzystując moduł `clpfd` znajduje rozwiązanie dla podanych zestawów danych.

Przykład

Na rysunku 3 przedstawiono rozwiązania dla zestawów danych z rysunku 2.

Wskazówka

Pomyśl o zastosowaniu ograniczenia `tuples_in/2`.



a) t6a b) t6b c) t6c

Rysunek 3: Rozwiązania trzech zestawów danych