Games

(Main file name: games)

You have decided to come to a block party in Brooklyn, where there are n game booths G_1, G_2, ..., $G_n$from one end of the street to the other. You want to play all of the games in the order from G_1 to $G_n$without skipping any. The games are not free, and each game costs 1 token to play once. You can play a game as many times as you can, as long as you have enough tokens. Your pocket can hold at most T tokens, and the pocket is full in the beginning. After you finish one game G_i and before you start G_{i+1}, you will be refilled with K tokens. However, since the capacity of your pocket is T, any extra tokens will be wasted.

Some games are more fun than others. For each game Gi, you have an integer value Vi that indicates how fun this game is to you. For a game that you don't enjoy, the value can be negative. Note that you have to play each game at least once, even if the game is not fun to you. The total fun you gain from a game is the number of times you play the game times the fun value. You want to manage your tokens so that you gain the most fun from the games.

Input Formats

An input file for LP systems contains a fact of the form num(N), which specifies the number of games N ($4 \leq N \leq 10$), a fact of the form cap(T), which means that the capacity of your pocket is T ($3 \leq T \leq 10$), a fact of the form refill(K), which gives the number to tokens you receive after each game booth ($0 < K \leq T$); N facts of the form fun(i,Vi), which indicates that the fun value of game Gi is Vi ($1 \leq i \leq N$, $-10 \leq Vi \leq 10$).

An input file for Minizinc specifies the following constants: num, the number of games N; cap, the capacity of your pocket; refill, the number of tokens you receive after each game booth; fun, an array of fun values of the N games.

Output format

The output should contain exactly one fact of the form total_fun(V), where V is the maximum fun you can gain from playing these games. For ASP systems, the output may consist of multiple answer sets, and only the final one is treated as a solution.

Samples

LP Input	Minizinc Input	Output

num(4).
cap(5).
refill(2).
fun(1,4).
fun(2,1).
fun(3,2).
fun(4,3).
	num = 4;
	cap = 5;
	refill = 2;
	fun = [4,1,2,3];
		total_fun(35).

num(4).
cap(5).
refill(2).
fun(1,4).
fun(2,-1).
fun(3,-2).
fun(4,3).
	num = 4;
	cap = 5;
	refill = 2;
	fun = [4,-1,-2,3];
		total_fun(29).

num(5).
cap(3).
refill(2).
fun(1,4).
fun(2,1).
fun(3,-2).
fun(4,3).
fun(5,4).
	num = 5;
	cap = 3;
	refill = 2;
	fun = [4,1,-2,3,4];
		total_fun(30).

Fixing N-Queens

(Main file name: fqueens)

The N-queens problem is probably one of the most famous problems in computer science. Given an NxN chess board and N queens, the goal of the N-queens problem is to place the N-queens on the squares of the board such that no two queens attack each other, meaning that no two queens are on the same row, the same column or the same diagonal.

You are given a configuration of N-queens on the board, which may not constitute a correct solution to the N-queens problem. Your goal is to transform the configuration into a correct solution by rearranging the queens in the fewest valid moves. In chess, the queen can move horizontally, vertically, or diagonally, and no other piece can occur between the starting square and the ending square of the move.

Input Formats

An input file for LP systems contains a fact of the form board_size(N), which specifies the number of queens N ($4 \leq N \leq 6$), and N facts of the form pos(R,C), which indicates that there is a queen at row R and column C ($1 \leq R , C \leq N$).

An input file for Minizinc specifies the size of the board, board_size, and two arrays, named row and col, respectively, that give the initial positions of the queens.

Output format

The output should contain exactly one fact of the form moves(K), where K is the minimum number of moves needed to rearrange the queens in the input configuration to build a correct solution to the N-queens problem. For ASP systems, the output may consist of multiple answer sets, and only the final one is treated as a solution.

Samples

LP Input	Minizinc Input	Output

board_size(4).
pos(1,2).
pos(2,4).
pos(3,1).
pos(4,3).
	board_size = 4;
	row = [1,2,3,4];
	col = [2,4,1,3];
		moves(0).

board_size(4).
pos(1,2).
pos(1,3).
pos(2,1).
pos(4,3).
	board_size = 4;
	row = [1,1,2,4];
	col = [2,3,1,3];
		moves(2).

board_size(5).
pos(1,1).
pos(2,5).
pos(3,3).
pos(5,2).
pos(5,5).
	board_size = 5;
	row = [1,2,3,5,5];
	col = [1,5,3,2,5];
		moves(3).

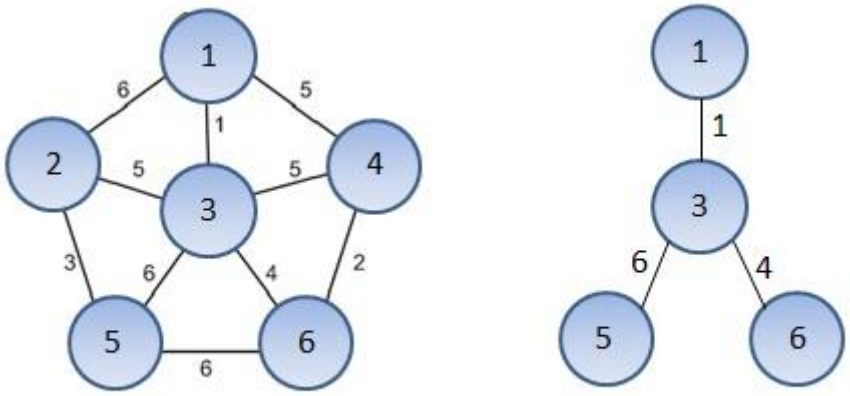
Logistics

(Main file name: logistics)

John is a truck driver. He needs to deliver a truckload of packages to different destination cities. In each city other truck drivers can help transport part of the packages with their trucks. So nobody is required to solve the traveling salesman problem. Also, no driver is required to return to his starting city. Nevertheless, John has to pay for the distances the trucks travel, and he wants to find routes for himself and the helpers such that the total travel distance is minimized.

The problem can be formulated as a graph problem as follows: Given an undirected weighted graph, a starting vertex, and a set of destination vertices, find a subgraph of the graph that has the minimum cost and covers the starting vertex and all of the destination vertices.

For example, for graph (a) shown below, assume the starting vertex is 1, and the set of destination vertices is {5,6}, then graph (b) shows a minimum covering tree.



Input Formats

An input file for LP systems contains the following facts:

- One fact of the form graph_size(N), which specifies the number of vertices N ($4 \leq N \leq 20$).
- One fact of the form start(V), which specifies the starting vertex ($1 \leq V \leq N$).
- A relation that consists of facts of the form dest(V), which specifies a destination vertex V ($1 \leq V \leq N$).
- A relation that consists of facts of the form edge(V1,V2,C), which indicates an edge between V1 and V2 with travel cost C ($1 \leq V1, V2 \leq N, 1 \leq C \leq 100$).

An input file for Minizinc specifies the size of the graph (graph_size), the starting vertex (start), the number of destinations (n_dests), an array of destination vertices (dest), the number of edges (n_edges), and three arrays that define the edge relation (from, to, and cost).

Output format

The output should contain exactly one fact of the form min_cost(K), where K is the travel cost of a minimum tree of the given graph that covers the starting vertex and the destination vertices. For ASP systems, the output may consist of multiple answer sets, and only the final one is treated as a solution.

Samples

LP Input	Minizinc Input	Output

graph_size(6).
start(1).
dest(6).
edge(1,2,4).
edge(1,3,2).
	graph_size = 6;
	start = 1;
	n_dests = 1;
	dest = [6];
	n_edges = 7;
		min_cost(20).

edge(2,3,5).
edge(2,4,10).
edge(3,5,3).
edge(4,5,4).
edge(4,6,11).
	from = [1,1,2,2,3,4,4];
	to = [2,3,3,4,5,5,6];
	cost = [4,2,5,10,3,4,11];	

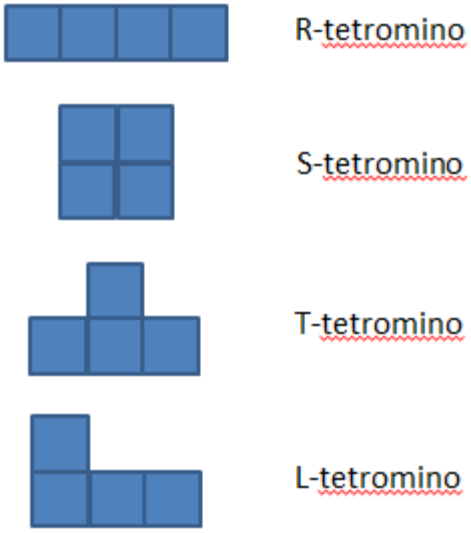
graph_size(6).
start(1).
dest(5).
dest(6).
edge(1,2,4).
edge(1,3,2).
edge(2,3,5).
edge(2,4,10).
edge(3,5,3).
edge(4,5,4).
edge(4,6,11).
	graph_size = 6;
	start = 1;
	n_dests = 2;
	dest = [5,6];
	n_edges = 7;
	from = [1,1,2,2,3,4,4];
	to = [2,3,3,4,5,5,6];
	cost = [4,2,5,10,3,4,11];
		min_cost(20).

graph_size(6).
start(1).
dest(5).
dest(6).
edge(1,2,6).
edge(1,3,1).
edge(1,4,5).
edge(2,3,5).
edge(2,5,3).
edge(3,4,5).
edge(3,5,6).
edge(3,6,4).
edge(4,6,2).
	graph_size = 6;
	start = 1;
	n_dests = 2;
	dest = [5,6];
	n_edges = 9;
	from = [1,1,1,2,2,3,3,3,4];
	to = [2,3,4,3,5,4,5,6,6];
	cost = [6,1,5,5,3,5,6,4,2];
		min_cost(11).

Packing

(Main file name: packing)

Packing is one of many problems that are tackled with constraint/logic programming. Given a 4*4 grid board and 4 tetrominoes of the following four types, is it possible to pack the board with the tetrominoes such that no tetrominoes overlap each other and the board is completely covered? Note that tetrominoes can be flipped and/or rotated before being put on the board.



Input Formats

An input file for LP systems contains the following four facts: r(R), there are R number of R-tetrominoes; s(S), there are S number of S-tetrominoes; t(T), there are T number of T-tetrominoes; and l(L), there are L number of L-tetrominoes, where $0 \leq R, S, T, L \leq 4$ and $R+S+T+L = 4$.

An input file for Minizinc specifies the following constants: r, the number of R-tetrominoes; s, the number of S-tetrominoes; t, the number of T-tetrominoes; l, the number of L-tetrominoes.

Output format

The output should be 'yes.' if the given number of tetrominoes can be packed into the 4*4 board, and 'no.' otherwise.

Samples

LP Input	Minizinc Input	Output

r(2).
s(2).
l(0).
t(0).
	r = 2;
	s = 2;
	l = 0;
	t = 0;
		yes.

r(1).
s(0).
l(1).
t(2).
	r = 1;
	s = 0;
	l = 1;
	t = 2;
		yes.

r(0).
s(0).
l(2).
t(2).
	r = 0;
	s = 0;
	l = 2;
	t = 2;
		no.

Pizza

(Main file name: pizza)

The problem arises in the University College Cork student dorms. There is a large order of pizzas for a party, and many of the students have vouchers for acquiring discounts in purchasing pizzas. A voucher is a pair of numbers e.g. (2,4), which means if you pay for 2 pizzas then you can obtain for free up to 4 pizzas as long as they each cost no more than the cheapest of the 2 pizzas you paid for. Similarly a voucher (3,2) means that if you pay for 3 pizzas you can get up to 2 pizzas for free as long as they each cost no more than the cheapest of the 3 pizzas you paid for. The aim is to obtain all the ordered pizzas for the least possible cost. Note that not all vouchers need to be used, and a voucher does not need to be totally used.

Input Formats

An input file for LP systems contains the following facts:

- One fact of the form n_pizzas(N), which specifies the number of pizzas to obtain.
- For each I in 1..N, there is a fact of the form pizza(I,C), which gives the price C of pizza I.
- One fact of the form n_vouchers(M), which gives the number of vouchers M.
- For each I in 1..M, there is a fact of the form voucher(I,B,F), which indicates that with voucher I one can get F free pizzas when buying B pizzas.

An input file for MiniZinc defines the following constants:

- n = number of pizzas to obtain;
- price = list of n pizza prices;
- m = number of vouchers;
- Two lists of size m named buy and free, which specify the vouchers. For i in 1..m, one can get free[i] free pizzas when buying buy[i] pizzas.

Output format

The output should contain exactly one fact of the form cost(K), where K is the minimum cost required to obtain the pizzas.

Samples

LP Input	Minizinc Input	Output

n_pizzas(4).
pizza(1,10).
pizza(2,5).
pizza(3,20).
pizza(4,15).
n_vouchers(2).
voucher(1,1,1).
voucher(2,2,1).
	n = 4;
	price = [10,5,20,15];
	m = 2;
	buy = [1,2];
	free = [1,1];
		cost(35).

n_pizzas(4).
pizza(1,10).
pizza(2,15).
pizza(3,20).
pizza(4,15).
n_vouchers(7).
voucher(1,1,1).
voucher(2,2,1).
voucher(3,2,2).
voucher(4,8,9).
voucher(5,3,1).
voucher(6,1,0).
voucher(7,4,1).
	n = 4;
	price = [10,15,20,15];
	m = 7;
	buy = [1,2,2,8,3,1,4];
	free = [1,1,2,9,1,0,1];
		cost(35).

n_pizzas(10).
pizza(1,70).
pizza(2,10).
pizza(3,60).
pizza(4,60).
pizza(5,30).
pizza(6,100).
pizza(7,60).
	n = 10;
	price = [70,10,60,60,30,100,60,40,60,20];
	m = 4;
	buy = [1,2,1,1];
	free = [1,1,1,0];
		cost(340).

pizza(8,40). pizza(9,60). pizza(10,20). n_vouchers(4). voucher(1,1,1). voucher(2,2,1). voucher(3,1,1). voucher(4,1,0).		