

2 Splitting a Train (*split.pl*)

One large train arrives at the shunting yard, and it needs to be split into two smaller trains. Trains consist of wagons, and each (type of) wagon has an identifier (something ground in Prolog). The three trains - the large one and the two smaller ones - are specified as a sequence of such identifiers (in Prolog, these will be lists of the identifiers). We are lucky: the large train has all the wagons to make the smaller trains, and in the correct order. So it is only a matter of deciding for each wagon to which track it goes, and we are done. This involves changing the track setting and decoupling (for the large train) and coupling (for the smaller trains), and since this takes a lot of time, we want to minimize that. Just one example: suppose there are five types of wagons (*i*, *c*, *l*, *f* and *p*), the large train is represented by `[i,i,c,l,c,f,p,p]` and the two smaller trains are `[i,c,l,p]` and `[i,c,f,p]` respectively, then it is easy to split the larger one into the two smaller ones by a split represented as `[1,2,1,1,2,2,1,2]` which means that the first wagon *i* goes to the first smaller train, the second wagon *i* goes to the second smaller train, the third wagon *c* goes to the first smaller train etc. But another split is `[1,2,1,1,2,2,2,1]` which has one less track change (+ the (de)coupling) so it is a better split.

Write a `split/4` predicate whose queries look like

```
?- split(Large,Small1,Small2,How).
```

where `Large` represents a large train, `Small1` and `Small2` represent the smaller trains, and `How` is free. The predicate must unify `How` with a *best* split.

3 Panoz (*panoz.pl*)

Manhattan has several outlets of the famous Belgian bakery Panoz. During an experiment with a new kind of chocolate pastry, the old bakery has exploded. A new bakery will be built, somewhere in Manhattan, and this occasion is used to optimize the distribution of all the goodies. Clearly Panoz wants the location in Manhattan which minimizes the maximal (Manhattan) distance from the bakery to any outlet.

Write a predicate `panoz/2` whose queries look like `?- panoz(Outlets,Sol)`. where `Sol` is the location of the new bakery. The outlet coordinates `Outlets` are given as a list of tuples, e.g. `[(0,1),(0,2),(4,0),(4,3)]`. Here is a typical query and its answer:

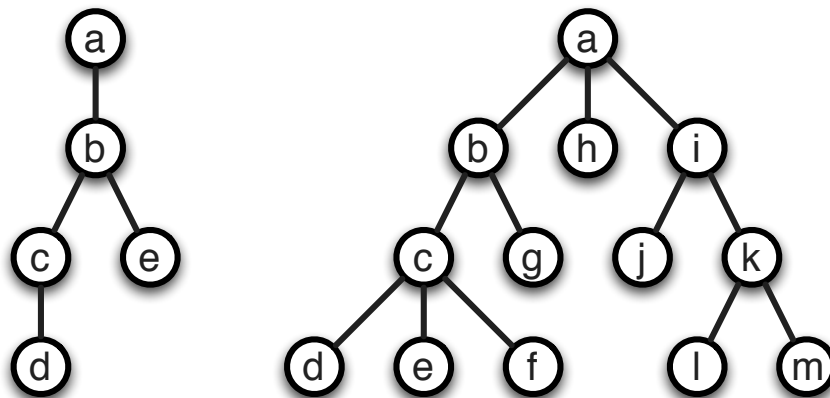
```
?- panoz([(0,1),(0,2),(4,0),(4,3)],Sol).  
Sol = (2,2)
```

Of course, `Sol = (2,1)` would also have been a good solution.

4 Three on the Circumference (*circ.pl*)

Given an undirected tree (specified as a list of `edge/2` terms), produce a set of three nodes which determine the *circumference* of the tree. The circumference of a tree is the maximal sum of the distances between three different nodes A,B,C of the tree, that is the sum of the distance from A to B, B to C and C to A - along the edges of the tree of course, where each edge has length 1.

Below are two trees with corresponding queries and answers.



```
?- circ([edge(a,b), edge(b,c), edge(b,e), edge(c,d)], Three, Len).
Len = 8
Three = [a,d,e]
```

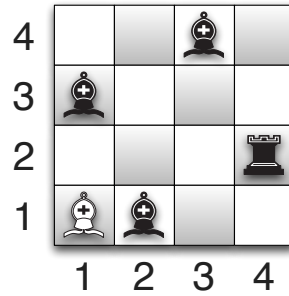
```
?- circ([edge(a,b), edge(a,h), edge(a,i), edge(b,c), edge(b,g),
        edge(i,k), edge(c,d), edge(c,e), edge(c,f), edge(k,l), edge(k,m)],
        Three, Len).
Len = 14
Three = [d,e,l]
```

These are of course not the only valid solutions.

5 Queens, Bishops and Rooks (*bishop.pl*)

You know the N -queens problem: put N queens on an $N \times N$ board so that no queen attacks any other queen. It is clear that when you have a solution to the N -queens problem, and you replace all queens by rooks, you have a solution to the N -rooks problem. And equally clear is that if you replace all queens in an N -queens solution by bishops, you get a solution to the N -bishops problem. Even better: if in an N -queens solution you replace every queen by either a rook or a bishop, you get a solution to the N -rooks/bishops problem. Indeed, if you replace I queens by bishops and J queens by rooks, you have an (I, J) -solution (of course, $I + J = N$). In some (I, J) -solutions, it is possible to add extra bishops, so that no

piece attacks any other piece. Obviously, this is never possible in a $(0, N)$ -solution. But in a $(3, 1)$ -solution, it is:



There is even room for one more bishop!

For a given N , what is the maximal J so that an $(N - J, J)$ -solution can be extended with at least one extra bishop? Write a predicate `bishop/2` which has as input the N and which unifies the second argument with this maximal J . E.g.,

```
?- bishop(4,N).
N = 2.
```