Review

Both the questions are recursive.

1. Split a list into two parts; the length of the first part is given.

   ?- split([a,b,c,d,e,f,g,h,i,k],3,L1,L2).
   L1 = [a,b,c]
   L2 = [d,e,f,g,h,i,k]
2. Insert an element at a given position into a list.

   ?- insert_at(alfa,[a,b,c,d],2,L).
   L = [a,alfa,b,c,d]

(Both the problems are taken from p-99. The solutions are there too. Don't look for them)

Search

We'll implement a breadth-first search on the Romania example. Download the lab2_search.zip file and fill in the missing sections of code.

You will need to fill in 2 predicates.

1. *bfs(Fringe, Goal, SolutionPath)*
2. *bfs_successor(Path, Successors).*

The romania example has been translated to prolog in romania.prolog . We have defined a predicate succ/3 as follows:

succ(SourceNode, SuccNode, _)

SuccNode is a node reachable from *SourceNode*. (The third argument is is the distance between *SourceNode* and *SuccNode*. Ignore it for BFS. We'll use this later).

We have left comments in the code to explain what each predicate and argument are supposed to do.

Here's the pseudocode for Tree-Search taken from the 'Problem Solving & Search' slides

```
function Tree-Search( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

The difference between various search procedures lies in choosing the leaf node to expand. In our case, The strategy is BFS.

(If you're done early, Try to implement Uniform Cost Search. Or try out some p99)