AIT Semantic and Declarative Technologies Course

Homework P5

For each problem, write a Prolog predicate that corresponds to the provided head comment.

You are free to make use of the predicates defined in the slides and in earlier exercise solutions. Do not use library predicates that are not discussed in the slides.

You can define helper predicates. Try to provide the most accurate head comment (specification, in other words) for the helper predicates. Remember that a head comment is an English language sentence which describes the logical relationship between the arguments of the predicate.

1. Chopping a list

```
% chop(+N, +List, -LofLists): LofLists is a list whose elements are
% nonempty lists, such that the concatenation of these results in List.
% All elements of LofLists, except for the last, have length N, the
% length of the last should be between 1 and N (inclusive).

| ?- chop(2, [1,a,b,2,c], LL).            LL = [[1,a],[b,2],[c]] ? ; no
| ?- chop(3, [1,a,b,2,c], LL).            LL = [[1,a,b],[2,c]] ? ; no
| ?- chop(3, [1,a,b,2,c,5], LL).          LL = [[1,a,b],[2,c,5]] ? ; no
| ?- chop(3, [1,a], LL).                  LL = [[1,a]] ? ; no
| ?- chop(3, [], LL).                     LL = [] ? ; no
```

   Hint: use `split/4` from Homework P4.

2. Enumerating sublists

```
% list_sub(+Whole, ?Part, ?Before, ?Length, ?After): Part is a
sublist of Whole such that there are Before number of elements in
Whole before Part, After number of elements in Whole after Part
and the length of Part is Length.

| ?- list_sub([a,b], Part, Before, Length, After).
Part = [], After = 2, Before = 0, Length = 0 ? ;
Part = [a], After = 1, Before = 0, Length = 1 ? ;
Part = [a,b], After = 0, Before = 0, Length = 2 ? ;
Part = [], After = 1, Before = 1, Length = 0 ? ;
Part = [b], After = 0, Before = 1, Length = 1 ? ;
Part = [], After = 0, Before = 2, Length = 0 ? ; no
```

   Note that this predicate is available in SICStus `library(lists)` as `sublist/5`. Obviously, you should not use this library predicate.
   Hint: make a declarative, non-recursive solution using the predicates `append/4` (from Class Practice P3) and `length/2` (BIP).
   Warning: The suggested solution results in **very-very** simple code :-).

3. Plateaus in a list
   Consider a proper list L of arbitrary ground terms (i.e. terms containing no variables). A sublist P of L is called a plateau, if its length is at least two, all its elements are identical, and it is maximal, i.e. it can not be extended to a longer list of identical elements.

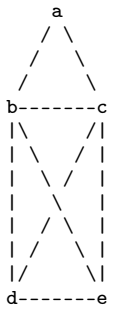
```
% plateau(+L, ?I, ?Len): There is a plateau of length Len starting at the
% I-th position of list L.

| ?- plateau([a,a,b,2,2,2,a+1,a+1,s(1,2)], I, Len).
I = 1, Len = 2 ? ;
I = 4, Len = 3 ? ;
I = 7, Len = 2 ? ; no
```

   You don't have to strive for ultimate efficiency, a declarative solution based on `append/4` and `length/2` is acceptable.

4. Drawing graphs with a single line

In the problem below the data structure *graph* is defined as follows:
A *graph* is a list of Prolog structures of the form `X-Y`, where both `X` and `Y` are atoms.
The Prolog list `[a1-b1,a2-b2,...,an-bn]` describes the graph (in mathematical sense) whose set of nodes is $\{a_1, .., a_n, b_1, ..., b_n\}$ and there is an undirected edge between a_i and b_i, for $i = 1, \ldots, n$.
Thus the Prolog terms `[a-b,a-c]`, `[a-c,b-a]`, `[b-a,a-c]`, `[c-a,a-b]`, etc. all describe the same graph.
The Prolog term `[a-b,a-c,b-c,b-d,b-e,c-d,c-e,d-e]` is one of the many terms describing the graph below:



You may have encountered the task of drawing this graph with a single line.
We define the notion of a *line* as a Prolog graph `[a1-b1,a2-b2,...,an-bn]` where $b_1 = a_2, b_2 = a_3, \ldots, b_{n-1} = a_n$.
We say that a graph `G` can be drawn by a line `L` iff `G` and `L` describe the same mathematical graph and furthermore `L` is a line.
Write a Prolog predicate `draw/2` which has the following head comment:

```
% draw(+G, ?L): Graph G can be drawn by line L.
```

The query `?- draw(G, L).` should thus return all lines `L` which "draw" graph `G`, i.e. describe the same graph as `G`.

```
| ?- draw([a-b,a-c], L).
L = [b-a,a-c] ? ;
L = [c-a,a-b] ? ;
no
| ?- draw([a-b,a-c,b-c,b-d,b-e,c-d,c-e,d-e], L), L = [d-e|_].
L = [d-e,e-b,b-a,a-c,c-b,b-d,d-c,c-e] ? ;
L = [d-e,e-b,b-a,a-c,c-d,d-b,b-c,c-e] ? ;
L = [d-e,e-b,b-c,c-a,a-b,b-d,d-c,c-e] ? ;
L = [d-e,e-b,b-c,c-d,d-b,b-a,a-c,c-e] ? ;
L = [d-e,e-b,b-d,d-c,c-a,a-b,b-c,c-e] ? ;
L = [d-e,e-b,b-d,d-c,c-b,b-a,a-c,c-e] ? ;
L = [d-e,e-c,c-a,a-b,b-c,c-d,d-b,b-e] ? ;
L = [d-e,e-c,c-a,a-b,b-d,d-c,c-b,b-e] ? ;
L = [d-e,e-c,c-b,b-a,a-c,c-d,d-b,b-e] ? ;
L = [d-e,e-c,c-b,b-d,d-c,c-a,a-b,b-e] ? ;
L = [d-e,e-c,c-d,d-b,b-a,a-c,c-b,b-e] ? ;
L = [d-e,e-c,c-d,d-b,b-c,c-a,a-b,b-e] ? ;
no
```

You don't have to strive for ultimate efficiency, but your program should be able to find all solutions in the last example in a couple of seconds.
Hints: Write the following helper predicate:

```
% draw(+G, +P, ?L): Graph G can be drawn by line L which starts at point P
% (in other words: L is of the form [P-_|_])
```

Use the predicate `select/3` from `library(lists)`, also defined in the slides.
Having completed predicate `draw/3` check if it works when called with a variable as the second argument (`P`). If this is the case (which is most probable), you should be able to define `draw/2` in terms of `draw/3` in an extremely simple way.
To load a library, e.g. `lists`, and to import the predicate `select/3`, include the following line at the beginning of your program:

```
:- use_module(library(lists), /*The list of imported predicate functors: */[select/3]).
```