Consider the Prolog knowledge base in `basegraph.pl`. The knowledge base consists of two parts.

The first part encodes a *weighted undirected graph* using `edge/3` facts, where `edge(v,w,c)` indicates that nodes `v` and `w` are connected through an edge whose associated cost is the positive, real number `c`. In particular, `basegraph.pl` encodes the following graph:

![Base graph](image)

**Figure 1**: Base graph.

This first part can be changed so as to encode a different graph.

The second part of the knowledge base is instead fixed, and encodes two useful additional predicates:

- A predicate `connection(N1,N2,C)` that is true whenever `N1` and `N2` are two different nodes from the graph, such that there is an edge between them with cost `C`. Notice that this predicate is symmetric, i.e., `connection(N1,N2,C)` is true if and only if `connection(N2,N1,C)` is true.

- A predicate `graph_nodes(L)` that is true when `L` is a list containing the set of all nodes in the graph.
By exploiting these two predicates, you have to:

1. Write a predicate \texttt{path(S,T,P,L)} that is true if \( P \) is a path connecting \( S \) and \( P \) with total length \( L \). For example, by asking all solutions of the query \( ?- \texttt{path(s,t,P,L)} \), we should obtain all paths connecting \( s \) with \( t \) (with their respective lengths).

2. Write a predicate \texttt{spath(S,T,P,L)} that is true if \( P \) is (one of) the shortest path(s) connecting \( S \) and \( P \), with total length \( L \). For example, by querying the knowledge base with \texttt{spath(s,t,P,L)}, we should get the following:

\[
?- \texttt{spath(s,t,P,L)}.
\]
\[
P = [s, \text{n3, n5, t}],
\]
\[
L = 8.7.
\]

3. Write a predicate \texttt{connected} that is true if the graph encoded in the knowledge base is \textit{connected}, that is, guarantees the existence of a path between every pair of vertices.

4. Write a predicate \texttt{prim(SP)} that checks whether the graph encoded in the knowledge base is connected and, if so, binds \( SP \) to a minimum spanning tree of the graph, computed using \textit{Prim’s algorithm}. For simplicity, it is enough for the predicate to match with only one of the minimum spanning trees if more than one exists.

For the definition of minimum spanning tree, refer to the corresponding Wikipedia page:
\texttt{https://en.wikipedia.org/wiki/Minimum_spanning_tree}

For the implementation of Prim’s algorithm, refer to the pseudocode in the description section of the corresponding Wikipedia page:
\texttt{https://en.wikipedia.org/wiki/Prim%27s_algorithm}

For example, we should get the following:

\[
?- \texttt{prim(ST)}, \texttt{writeln(ST)}.
\]
\[
\text{[s,n3,edge(n3,s),n1,edge(n1,n3),n2,edge(n2,n3),n5,edge(n5,n3),n4,edge(n4,n5),t,edge(t,n5)]}
\]
\[
\text{ST} = [s, n3, \text{edge(n3, s), n1, edge(n1, n3), n2, edge(n2, n3), n5, edge(..., ...)...]}
\]