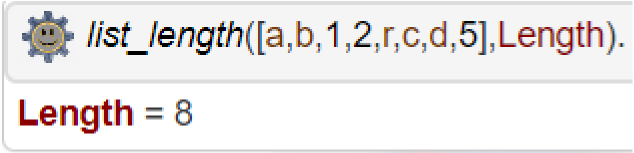
ARTIFICIAL INTELLIGENCE
ASSIGNMENT 6

U20CS005
BANSI MARAKANA

Q1.Write a PROLOG program based on list:-
A) To find the length of a list.

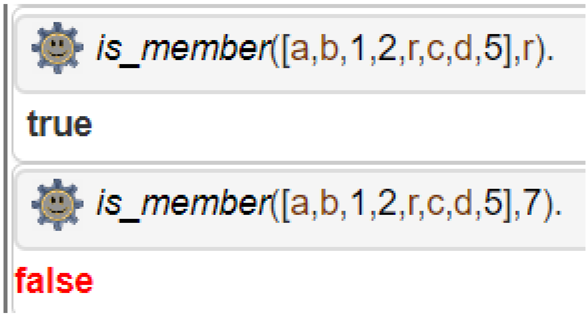
list_length([], 0).
list_length([_|TAIL], Length) :- list_length(TAIL, N1), Length is N1 + 1.



```
list_length([a,b,1,2,r,c,d,5],Length).
Length = 8
```

B) To find whether a given element is a member of a list.

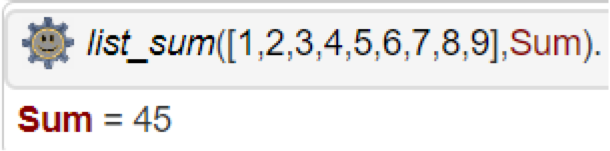
is_member([Element | _], Element):- !.
is_member([_ | Tail], Element):- is_member(Tail, Element).



```
is_member([a,b,1,2,r,c,d,5],r).
true
is_member([a,b,1,2,r,c,d,5],7).
false
```

C) To add the member of a given list (sum of elements of List).

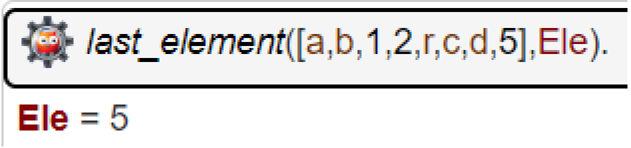
list_sum([],0).
list_sum([Head|Tail], Result) :- list_sum(Tail ,Temp), Result is Head + Temp.



```
list_sum([1,2,3,4,5,6,7,8,9],Sum).
Sum = 45
```

D) To find the last element of a list.

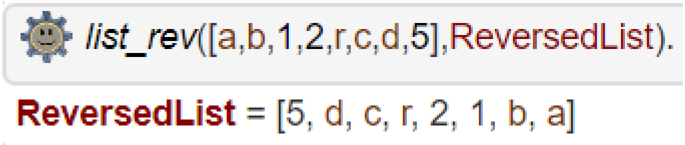
last_element([X],X).
last_element([_|L],X) :- last_element(L,X).



```
last_element([a,b,1,2,r,c,d,5],Ele).
Ele = 5
```

E) To reverse a list.

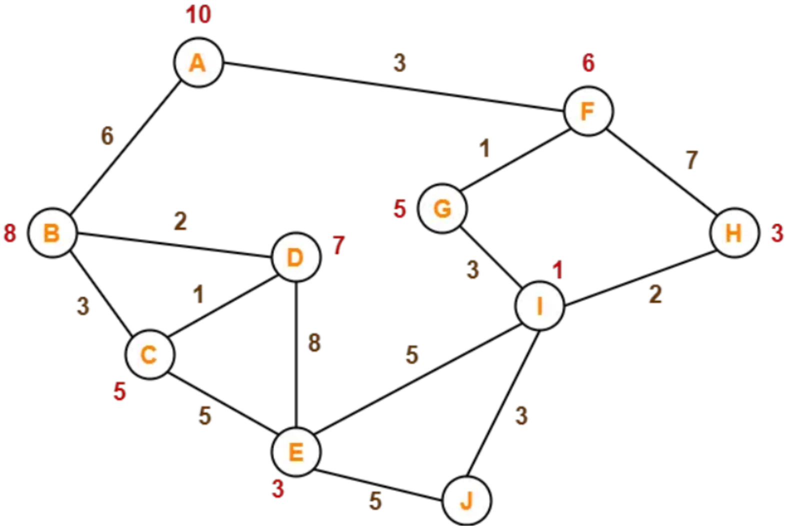
```
list_concat([],L,L).
list_concat([X1|L1],L2,[X1|L3]) :- list_concat(L1,L2,L3).
list_rev([],[]).
list_rev([Head|Tail],Rev) :- list_rev(Tail, RevTail),list_concat(RevTail, [Head],Rev).
```

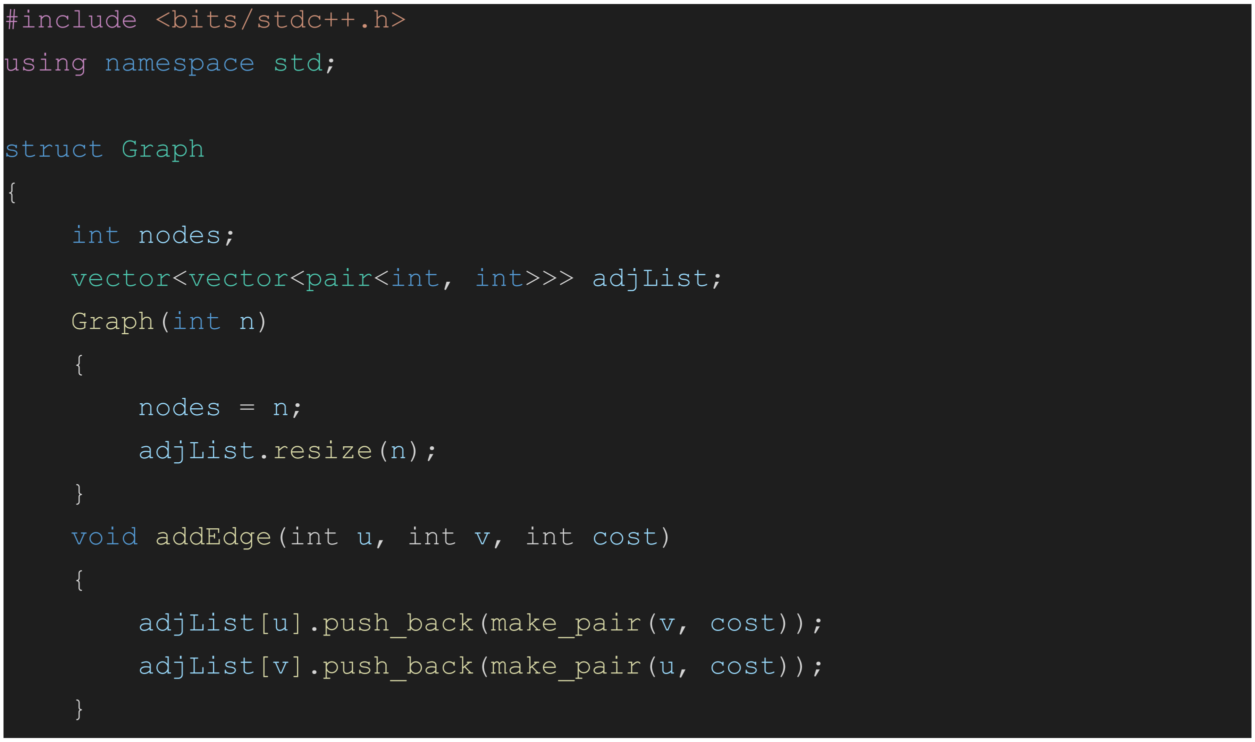


```
list_rev([a,b,1,2,r,c,d,5],ReversedList).

ReversedList = [5, d, c, r, 2, 1, b, a]
```

Q2. Implement A* algorithm. You can implement the algorithm in any language. Try to solve following problem as shown figure 1 below -Find shortest path from A to J





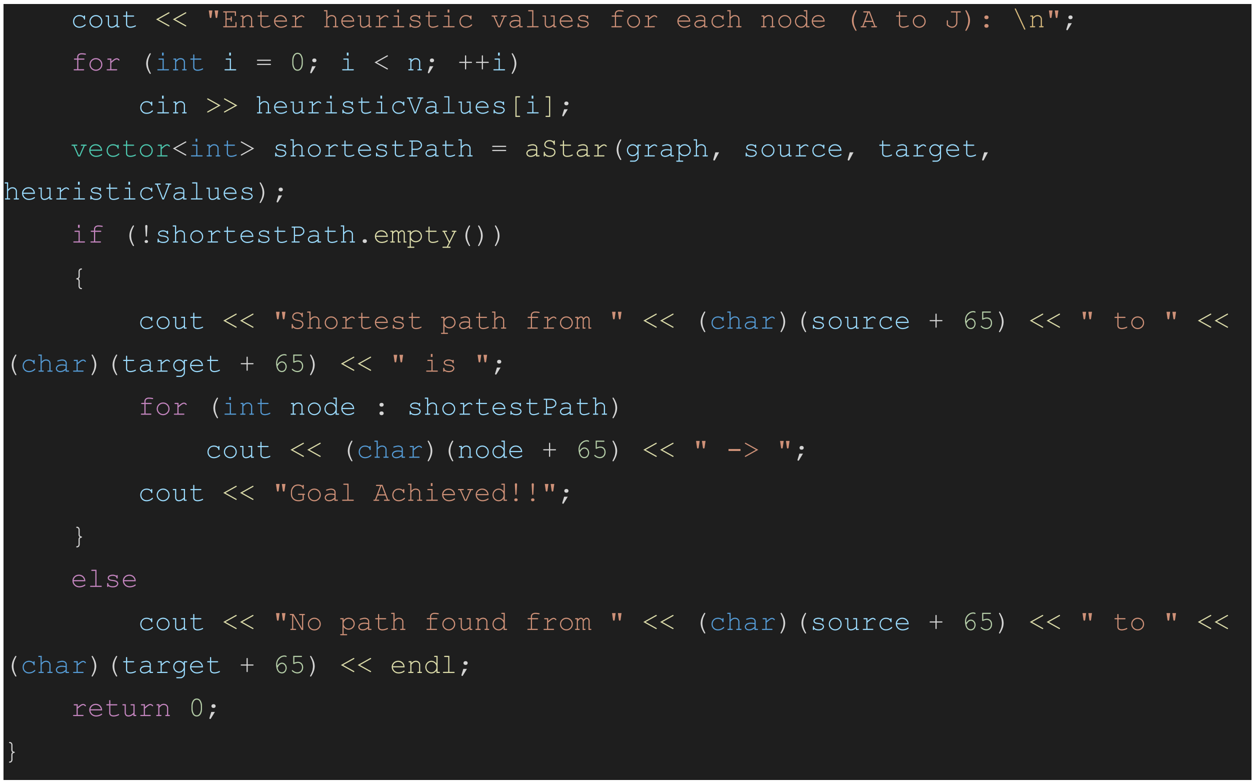
```
#include <bits/stdc++.h>
using namespace std;

struct Graph
{
    int nodes;
    vector<vector<pair<int, int>>> adjList;
    Graph(int n)
    {
        nodes = n;
        adjList.resize(n);
    }
    void addEdge(int u, int v, int cost)
    {
        adjList[u].push_back(make_pair(v, cost));
        adjList[v].push_back(make_pair(u, cost));
    }
```

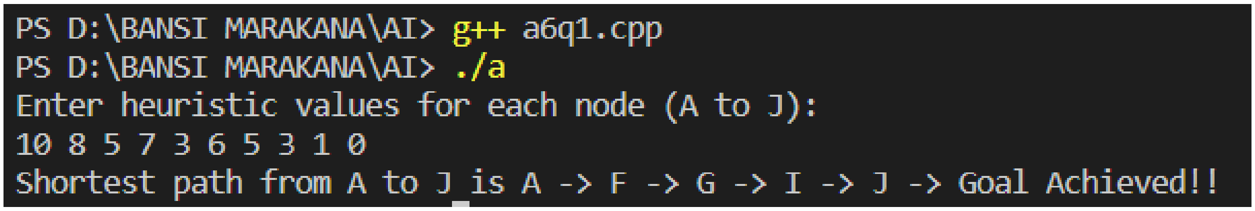
```
};

vector<int> aStar(const Graph &graph, int source, int target, const
vector<int> &heuristic)
{
    int n = graph.nodes;
    vector<int> g(n, numeric_limits<int>::max());
// g(n) to store cost from source to node n
    vector<int> f(n, numeric_limits<int>::max());
// f(n) to store estimated total cost from source to target passing
through node n
    vector<int> parent(n, -1);
// parent(n) to store parent node of node n in the shortest path
    vector<bool> visited(n, false);
// visited(n) to mark visited nodes
    priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<pair<int, int>>> pq; // min heap to store nodes with lowest f
value
    g[source] = 0;
    f[source] = heuristic[source];
    pq.push(make_pair(f[source], source));
    while (!pq.empty())
    {
        int u = pq.top().second;
        pq.pop();
        if (u == target)
        {
            // Target node reached, return the shortest path
            vector<int> shortestPath;
            while (u != -1)
            {
                shortestPath.push_back(u);
                u = parent[u];
            }
            reverse(shortestPath.begin(), shortestPath.end());
            return shortestPath;
        }
        if (visited[u]) // Skip already visited nodes
            continue;
        visited[u] = true;
        for (const auto &edge : graph.adjList[u])
        {
            int v = edge.first;
            int cost = edge.second;
            int tentative_g = g[u] + cost;
            if (tentative_g < g[v])
            {
                // Update g value and f value
                g[v] = tentative_g;
                f[v] = g[v] + heuristic[v];
                parent[v] = u;
                pq.push(make_pair(f[v], v));
            }
        }
    }
    // No path found from source to target
    return vector<int>();
}

int main()
{
    int n = 10;
    Graph graph(n);
    graph.addEdge(0, 1, 6);
    graph.addEdge(0, 5, 3);
    graph.addEdge(1, 3, 2);
    graph.addEdge(1, 2, 3);
    graph.addEdge(2, 3, 1);
    graph.addEdge(2, 4, 5);
    graph.addEdge(3, 4, 8);
    graph.addEdge(4, 9, 5);
    graph.addEdge(4, 8, 5);
    graph.addEdge(8, 9, 3);
    graph.addEdge(8, 6, 3);
    graph.addEdge(8, 7, 2);
    graph.addEdge(5, 7, 7);
    graph.addEdge(5, 6, 1);
    int source = 0;
    int target = 9;
    vector<int> heuristicValues(n);
```



```
    cout << "Enter heuristic values for each node (A to J): \n";
    for (int i = 0; i < n; ++i)
        cin >> heuristicValues[i];
    vector<int> shortestPath = aStar(graph, source, target,
heuristicValues);
    if (!shortestPath.empty())
    {
        cout << "Shortest path from " << (char)(source + 65) << " to " <<
(char)(target + 65) << " is ";
        for (int node : shortestPath)
            cout << (char)(node + 65) << " -> ";
        cout << "Goal Achieved!!";
    }
    else
        cout << "No path found from " << (char)(source + 65) << " to " <<
(char)(target + 65) << endl;
    return 0;
}
```



```
PS D:\BANSI MARAKANA\AI> g++ a6q1.cpp
PS D:\BANSI MARAKANA\AI> ./a
Enter heuristic values for each node (A to J):
10 8 5 7 3 6 5 3 1 0
Shortest path from A to J is A -> F -> G -> I -> J -> Goal Achieved!!
```