AI - Assignment-2

Harjeet Singh Yadav
Roll no. 2020561

q1)

a) **Yes,** the new algorithm will be optimal even if we don't use the explored set because the heuristic function is admissible, which means it will never overestimate the cost of reaching the goal node. If the heuristic function is admissible, the values of the f(n) along any path are non-decreasing.

The proof follows directly from the definition of consistency. Suppose n' is a successor of n; then g(n) = g(n) + c(n, a, n) for some action a, and we have
f (n') = g(n') + h(n‘) + g(n) + c(n, a, n') + h(n') ≥ g(n) + h(n) = f (n)
And h(n) ≤ c(n, a, n) + h(n‘)
f(n ‘) >= f(n)

Even if there is a loop, the heuristic cost function will be more when we return to the parent node so that that path won't be selected. And it'll not get stuck in the loop.
The next step is to prove that whenever A ∗ selects a node n for expansion, the optimal path to that node has been found. Were this not the case, there would have to be another frontier node n on the optimal path from the start node to n, by the graph separation property of because f is nondecreasing along any path, n would have lower
f-cost than n and would have been selected first. Hence, the solution will be optimal.

b) **Yes,** the algorithm will be complete. This means it will find the solution if there exists one.
If the heuristic is admissible, it will reach the goad node; we have proved that in the a) part. If the branching factor is not infinite, it will find the solution.

c) The new algorithm will **not be faster** than the one where we use the explored set because while exploring, we will add more nodes into the

priority queue. If there is a cycle or loop in the graph, then parent nodes will be added to the queue more than once and will increase the time complexity of sorting the queue. If the branching factor is infinite, the increase in complexity will be significant. On the contrary, if the branching factor is finite, the increase in complexity will be moderate or very less.

q2) best first search:-

```
void bestFirstSearch(int start, int goal){

    priority_queue<pii, vector<pii>, greater<pii>> pq;

    vector<bool> visited(graph.size(), false);
    vector<int> parent(graph.size(), -1);
    vector<int> cost(graph.size(), INT_MAX);

    cost[start] = 0;
    pq.push({0, start});

    while(!pq.empty()){

        int u = pq.top().second;
        pq.pop();

        if(u == goal){
            cout << "Goal found \nwith cost : " << cost[u] << endl;
            cout << "Path: ";

            while(u != -1){
                cout << u << " ";
                u = parent[u];
                cout<<"<-";
            }

            return;
        }

        if(visited[u]) continue;
        visited[u] = true;

        for(auto v : graph[u]){

            if(!visited[v.first]){

                if(cost[v.first] > cost[u] + v.second){
                    cost[v.first] = cost[u] + v.second;
                    parent[v.first] = u;
                }

                pq.push({v.second, v.first});
            }
        }
    }

    cout << "Goal not found" << endl;
}
```

A Star:-

```
void aStarSearch(int start, int goal){

    priority_queue<pii, vector<pii>, greater<pii>> pq;

    vector<bool> visited(graph.size(), false);
    vector<int> parent(graph.size(), -1);
    vector<int> cost(graph.size(), INT_MAX);

    cost[start] = 0;
    pq.push({ heuristic[start], start});

    while(!pq.empty()){

        int u = pq.top().second;
        pq.pop();

        if(visited[u]) continue;
        visited[u] = true;

        for(auto v : graph[u]){
            int child = v.first;
            int path_cost = v.second;

            if(!visited[child]){

                if(cost[child] > cost[u] + path_cost){
                    cost[child] = cost[u] + path_cost;
                    parent[child] = u;
                }

                pq.push({ cost[child] + heuristic[child], child});
            }
        }
    }

    if (cost[goal] == INT_MAX)
    {
        cout << "Goal not found" << endl;
        return;

    }else{
        cout << "Goal found \nwith cost : " << cost[goal] << endl;
        cout << "Path: ";

        while(goal != -1){
            cout << goal << " ";
            goal = parent[goal];
            cout<<"<-";
```

Dijkstra:-

```cpp
void dijkstra(int start, int goal){
    priority_queue<pii, vector<pii>, greater<pii>> pq;

    vector<bool> visited(graph.size(), false);
    vector<int> parent(graph.size(), -1);
    vector<int> cost(graph.size(), INT_MAX);

    cost[start] = 0;
    pq.push({0, start});

    while (!pq.empty())
    {
        int u = pq.top().second;
        pq.pop();

        for( auto v : graph[u]){

            int child = v.first;
            int path_cost = v.second;

            if(!visited[child]){

                if(cost[child] > cost[u] + path_cost){
                    cost[child] = cost[u] + path_cost;
                    parent[child] = u;
                }

                pq.push({cost[child], child});
            }
        }

        visited[u] = true;
    }

    if (cost[goal] == INT_MAX)
    {
        cout << "Goal not found" << endl;
        return;

    }else{
        cout << "Goal found \nwith cost : " << cost[goal] << endl;
        cout << "Path: ";

        while(goal != -1){
            cout << goal << " ";
            goal = parent[goal];
            cout<<"<-";
        }
```

Ans by all three methods:-

```
[Running] cd "/home/harjeet/Downloads/AI/A2/" && g++ bestfs.cpp -o bestfs &&
Goal found
with cost : 14
Path: 7 <-9 <-8 <-4 <-1 <-0 <-
[Done] exited with code=0 in 0.759 seconds

[Running] cd "/home/harjeet/Downloads/AI/A2/" && g++ dijkstra.cpp -o dijkstra
Goal found
with cost : 14
Path: 7 <-9 <-8 <-4 <-1 <-0 <-
[Done] exited with code=0 in 0.805 seconds

[Running] cd "/home/harjeet/Downloads/AI/A2/" && g++ astar.cpp -o astar && "/
Goal found
with cost : 14
Path: 7 <-9 <-8 <-4 <-1 <-0 <-
[Done] exited with code=0 in 0.834 seconds
```

q3)

For the **admissible heuristic**, I have used a **straight line distance (Euclidian distance)** between the cities. This is because the Euclidean distance heuristic is admissible. After all, it satisfies the conditions of being an underestimate of the true cost to reach the goal. This is because the actual road distance between two cities can never be shorter than the straight-line distance between them (the shortest path is a straight line).

For the **inadmissible heuristic**, I have used an intermediate node or city between the current node and the goal node. So, t**he h(n) value will be h(n to intermediate) + h(intermediate to goal).** This means the heuristic calculates the estimated distance of reaching the goal via the intermediate node. However, this may not always be correct as there could be another shorter road to reaching the goal that does not pass through the intermediate node. Hence, the heuristic is inadmissible. We can prove this with the **triangle inequality.**