Experiment No. 2: Greedy method strategy

Date:

Aim: Write a C program to implement the following program using Greedy method strategy

a. Prims algorithm.
b. Kruskals algorithm.
c. Single source shortest path algorithm.

THEORY:

- The greedy algorithm is a problem-solving approach that chooses the best available option at each step of the solution process.
- It's based on the principle of making locally optimal choices with the hope of finding a global optimum solution.
- The algorithm starts with an empty solution set and iteratively builds it up by selecting the best available option at each step.
- The selection of the best option is based on a certain criterion, which can be the highest or lowest value, the maximum or minimum weight, or any other relevant metric.
- The algorithm does not revisit the choices made in previous steps, which may result in a suboptimal solution.
- The greedy algorithm is useful for solving optimization problems, such as the Knapsack problem, minimum spanning tree, and shortest path problem.
- It's often fast and efficient, but may not always guarantee the optimal solution, particularly in complex problems.
- The correctness of the greedy algorithm depends on the problem structure and the choice of the criterion for selecting the best option.
- Sometimes, a greedy approach can be combined with other algorithms to improve the quality of the solution
- The advantage to using a greedy algorithm is that solutions to smaller instances of the problem can be straightforward and easy to understand.
- No memory is used The disadvantage is that it is entirely possible that the most optimal short-term solutions may lead to the worst possible long-term outcome

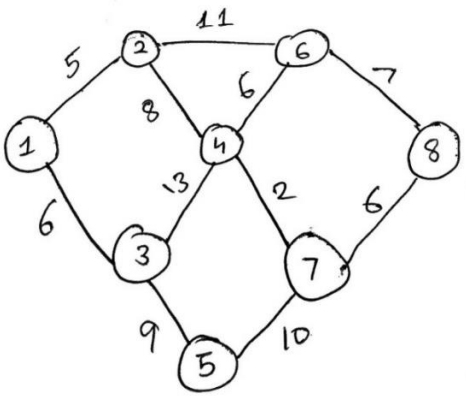
a. Prim's algorithm

Date:

Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

Problem Statement:

Write a c program to implement a minimum cost spanning tree using prims algorithm on following graph:



Algorithm

```
1   Algorithm Prim(E, cost, n, t)
2   // E is the set of edges in G. cost[1 : n, 1 : n] is the cost
3   // adjacency matrix of an n vertex graph such that cost[i, j] is
4   // either a positive real number or ∞ if no edge (i, j) exists.
5   // A minimum spanning tree is computed and stored as a set of
6   // edges in the array t[1 : n − 1, 1 : 2]. (t[i, 1], t[i, 2]) is an edge in
7   // the minimum-cost spanning tree. The final cost is returned.
8   {
9       Let (k, l) be an edge of minimum cost in E;
10      mincost := cost[k, l];
11      t[1, 1] := k; t[1, 2] := l;
12      for i := 1 to n do  // Initialize near.
13          if (cost[i, l] < cost[i, k]) then near[i] := l;
14          else near[i] := k;
15      near[k] := near[l] := 0;
16      for i := 2 to n − 1 do
17      { // Find n − 2 additional edges for t.
18          Let j be an index such that near[j] ≠ 0 and
19          cost[j, near[j]] is minimum;
20          t[i, 1] := j; t[i, 2] := near[j];
21          mincost := mincost + cost[j, near[j]];
22          near[j] := 0;
23          for k := 1 to n do // Update near[ ].
24              if ((near[k] ≠ 0) and (cost[k, near[k]] > cost[k, j]))
25                  then near[k] := j;
26      }
27      return mincost;
28  }
```

Time Complexity & Space Complexity

Time Complexity is O(n2). Assume that we are given V vertices and E edges in a graph for which we need to find an MST.

Time Complexity:

$$= V + V(\log(V)) + 2E + E(\log(v))$$

$$= V(\log(V)) + E(\log(v))$$

$$= O(V(\log(V)) + E(\log(v)))$$

So, time complexity is O((V+E)Log(V)). Best and Worst Cases for Prim's Algorithm Best case time complexity of Prim's is when the given graph is a tree itself and each node has a minimum number of adjacent nodes. Worst case time complexity would be when it's a graph with V2 edges.

Space Complexity of Prim's
We need an array to know if a node is in MST or not. Space O(V).We need an array to maintain Min-Heap. Space O(E). Total space complexity is of order O(V+E).

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define MAX 100

int cost[MAX][MAX];
int t[MAX][MAX];
int near[MAX];
int prim(int n, int e);
int main(){
   int n, e, s, d, cost1, mincost;
   printf("Enter the number of vertex : ");
   scanf("%d", &n);
   for (int i = 1; i <= n; i++){
      for (int j = 1; j <= n; j++){
         cost[i][j] = cost[j][i] = INT_MAX;
      } }
   printf("Enter the number of edges : ");
```

```
    scanf("%d", &e);
    for (int i = 0; i < e; i++){
        printf("Source : ");
        scanf("%d", &s);
        printf("Destination : ");
        scanf("%d", &d);
        printf("Cost : ");
        scanf("%d", &cost1);
        // save both at 1,2 and 2,1 (undirected graph)
        cost[s][d] = cost[d][s] = cost1;
    }
    mincost = prim(n, e);
    printf("Total MinCost = %d ", mincost);
}
int prim(int n, int e){
    int k, l, mincost;
    k = 1;
    l = 1;
    mincost = INT_MAX;

    //FIND MINIMUM COST EDGE
    for (int i = 1; i <= n; i++){
        for (int j = i + 1; j <= n; j++){
            if (cost[i][j] < mincost) {
                k = i;
                l = j;
                mincost = cost[i][j];
            }}}
    mincost = cost[k][l];
    t[1][1] = k;
    t[1][2] = l;
    for (int i = 1; i <= n; i++){
        if (cost[i][l] < cost[i][k]) {
            near[i] = l;
```

```
        }
        else{
            near[i] = k;
        }
}
    near[k] = near[l] = 0;
    int i = l, j = k, min = INT_MAX;
    for (int i = 2; i <= n - 1; i++){
        min = INT_MAX;
        for (int k = 1; k <= n; k++){
            if (near[k] != 0 && cost[k][near[k]] < min) {
                j = k;
                min = cost[k][near[k]];
            }
}
        t[i][1] = j;
        t[i][2] = near[j];
        printf("Step %d (cost) : %d\n",i-1,mincost);
        mincost = mincost + cost[j][near[j]];
        near[j] = 0;
        for (int k = 1; k <= n; k++){
            if (near[k] != 0 && cost[k][near[k]] >= cost[k][j]) {
                near[k] = j;
            }
}
}
    return mincost;
}
```

Output

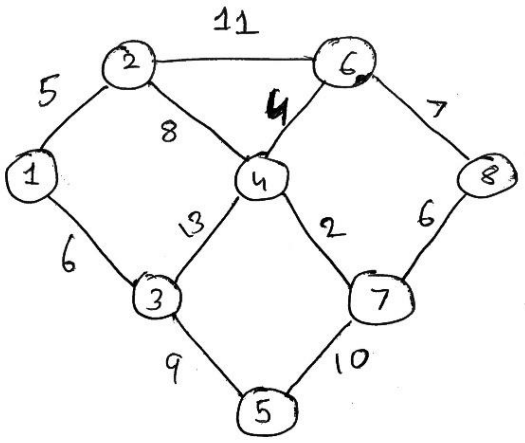
```
Enter the number of vertex : 8
Enter the number of edges : 11
Source : 1
Destination : 2
Cost : 5
Source : 1
Destination : 3
Cost : 6
Source : 3
Destination : 5
Cost : 9
Source : 3
Destination : 4
Cost : 13
Source : 4
Destination : 7
Cost : 2
Source : 5
Destination : 7
Cost : 10
Source : 4
Destination : 6
Cost : 6
Source : 7
Destination : 8
Cost : 6
Source : 6
Destination : 8
Cost : 7
Source : 2
Destination : 6
Cost : 11
Source : 2
Destination : 4
Cost : 8
Step 1 (cost) : 2
Step 2 (cost) : 8
Step 3 (cost) : 14
Step 4 (cost) : 22
Step 5 (cost) : 27
Step 6 (cost) : 33
Total MinCost = 42
```

b. Kruskals algorithm

Date :

Problem Statement:

Write a c program to implement a minimum cost spanning tree using kruskals algorithm on following graph:



Algorithm

```
1   Algorithm Kruskal(E, cost, n, t)
2   // E is the set of edges in G. G has n vertices. cost[u, v] is the
3   // cost of edge (u, v). t is the set of edges in the minimum-cost
4   // spanning tree. The final cost is returned.
5   {
6       Construct a heap out of the edge costs using Heapify;
7       for i := 1 to n do parent[i] := -1;
8       // Each vertex is in a different set.
9       i := 0; mincost := 0.0;
10      while ((i < n - 1) and (heap not empty)) do
11      {
12          Delete a minimum cost edge (u, v) from the heap
13          and reheapify using Adjust;
14          j := Find(u); k := Find(v);
15          if (j ≠ k) then
16          {
17              i := i + 1;
18              t[i, 1] := u; t[i, 2] := v;
19              mincost := mincost + cost[u, v];
20              Union(j, k);
21          }
22      }
23      if (i ≠ n - 1) then write ("No spanning tree");
24      else return mincost;
25  }
```

Time Complexity and Space Complexity

The time complexity of Kruskal's minimum spanning tree algorithm is O(E*logV), where E is the number of edges in the graph and V is the number of vertices.

Reason: Since we're sorting the edges, which takes O(E*logE) time. Then we check for each edge whether to add it or not by using the union-find algorithm, which takes at most O(logV) time for every edge E, Hence total O(ElogV). Thus the overall complexity will be O(ElogE + ElogV), which is approximately O(ElogV).

Space Complexity : The space complexity of Kruskal's minimum spanning tree algorithm is O(|E| + |V|), where E is the number of edges in the graph and V is the number of vertices. Reason: Since the disjoint set structure takes O(|V|) space to store the parents of vertices and O(|E|), we are additionally storing the edges of the graph.

Code

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <limits.h>
int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[9][9], parent[9];
int find(int i){
while (parent[i]){
i = parent[i];
}
return i;
}
int uni(int i, int j){
if (i != j){
parent[j] = i;
return 1;
}
return 0;
}
int main(){
int c, origin, dest;
printf("\nEnter the no. of vertices:");
scanf("%d", &n);
for (i = 1; i <= n; i++){
for (j = 1; j <= n; j++){
cost[i][j] = INT_MAX;
}}
int max_edges=(n*(n-1))/2;
for (i=1; i<=max_edges; i++){
printf("Enter origin and destination(-1 -1): ");
scanf("%d %d", &origin, &dest);
if (origin==-1 && dest==-1)
break;
printf("Enter cost: ");
scanf("%d", &c);
```

```
cost[origin][dest]=c;
cost[dest][origin]=c;
}
printf("\n");
while (ne < n){
min = INT_MAX;
for (i = 1; i <= n; i++){
for (j = 1; j <= n; j++){
if (cost[i][j] < min){
min = cost[i][j];
a = u = i;
b = v = j;
}}}
u = find(u);
v = find(v);
if (uni(u, v)){
mincost += min;
ne++;
printf("Step cost : %d\n",mincost);
}
cost[a][b] = cost[b][a] = INT_MAX;
}
printf("\n=> Minimum cost = %d\n",mincost);
return 0;
}
```

Output:

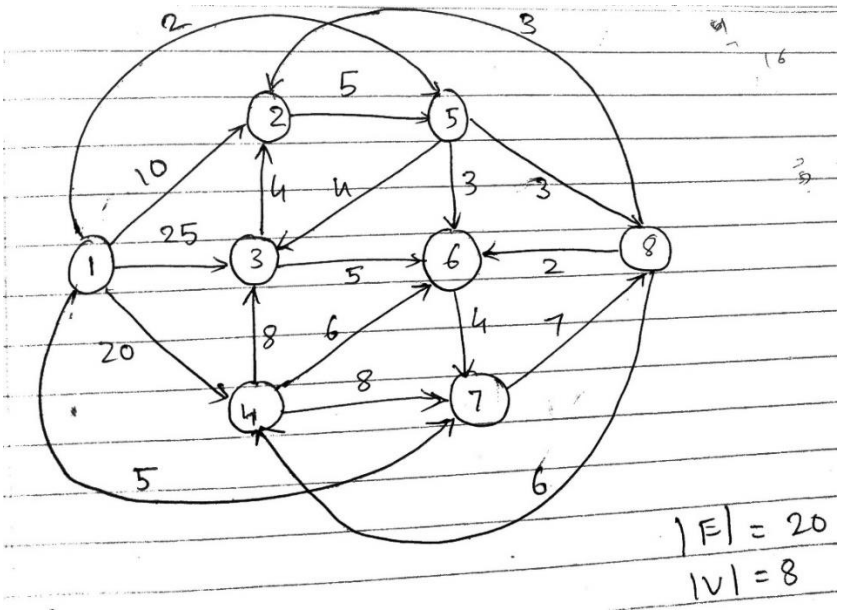
```
Enter the no. of vertices:8
Enter origin and destination(-1 -1): 1 2
Enter cost: 5
Enter origin and destination(-1 -1): 1 3
Enter cost: 6
Enter origin and destination(-1 -1): 2 4
Enter cost: 8
Enter cost: 4
Enter origin and destination(-1 -1): 6 8
Enter cost: 7
Enter origin and destination(-1 -1): 4 7
Enter cost: 2
Enter origin and destination(-1 -1): 7 8
Enter cost: 6
Enter origin and destination(-1 -1): 3 5
Enter cost: 9
Enter origin and destination(-1 -1): 5 7
Enter cost: 10
Enter origin and destination(-1 -1): -1 -1

Step cost : 2
Step cost : 6
Step cost : 11
Step cost : 17
Step cost : 23
Step cost : 31
Step cost : 40
=> Minimum cost = 40
```

c. Single source shortest path algorithm

Date:

Problem Statement:

Write a c program to find shortest path on following graph:



Algorithm

```
1   Algorithm ShortestPaths(v, cost, dist, n)
2   // dist[j], 1 ≤ j ≤ n, is set to the length of the shortest
3   // path from vertex v to vertex j in a digraph G with n
4   // vertices. dist[v] is set to zero. G is represented by its
5   // cost adjacency matrix cost[1 : n, 1 : n].
6   {
7       for i := 1 to n do
8       { // Initialize S.
9           S[i] := false; dist[i] := cost[v, i];
10      }
11      S[v] := true; dist[v] := 0.0; // Put v in S.
12      for num := 2 to n − 1 do
13      {
14          // Determine n − 1 paths from v.
15          Choose u from among those vertices not
16          in S such that dist[u] is minimum;
17          S[u] := true; // Put u in S.
18          for (each w adjacent to u with S[w] = false) do
19              // Update distances.
20              if (dist[w] > dist[u] + cost[u, w])) then
21                  dist[w] := dist[u] + cost[u, w];
22      }
23  }
```

Time Complexity and Space Complexity

Running time:

Depends on implementation of data structures for dist.

- Build a structure with n elements A
- at most m = □E □ times decrease the value of an item mB
- 'n' times select the smallest value nC
- For array A = O (n); B = O (1); C = O (n) which gives O (n2) total.
- For heap A = O (n); B = O (log n); C = O (log n) which gives O (n + m logn) total.

The time complexity of the Shortest-Paths algorithm using Dijkstra's algorithm is O(n^2) if implemented using an adjacency matrix and O((m+n) log n) if implemented using a priority queue and an adjacency list, where m is the number of edges in the graph. The space complexity is O(n) for storing the dist array and the S set.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>
#define MAX 100
int cost[MAX][MAX];
int dist[MAX];
bool s[MAX];
int minDistance(int dist[], bool sptSet[],int n)
{
// Initialize min value
int min = INT_MAX, min_index;
for (int v = 1; v <= n; v++)
if (sptSet[v] == false && dist[v] <= min)
        min = dist[v], min_index = v;
return min_index;
}
void shortestpath(int v,int e, int n){
   int i,num,u,min,w;
   for(i = 1; i <= n; i++){
      s[i] = false;
      dist[i] = cost[v][i];
      // dist[i] = INT_MAX;
   }
   s[v] = true;
   dist[v] = 0;
   for(num = 2; num <= n; num++){
      // min = INT_MAX;
       u = minDistance(dist, s,n);
      s[u] = true;
      for(w = 1; w<= n; w++){
```

```
        if((!s[w] && cost[u][w] && dist[u] != INT_MAX && dist[u]+cost[u][w] < dist[w])){
            dist[w] = dist[u] + cost[u][w];
        } } }
   printf("Vertex \t Distance from Source\n");
   for (i = 1; i <= n; i++) {
     printf("%d \t %d\n", i, dist[i]);
   } }
int main(){
   int n,cost1,i,j,e,s,d,v;
   v = 1;
   printf("Enter the number of vertex : ");
   scanf("%d", &n);
   for (int i = 1; i <= n; i++)
   {
      for (int j = 1; j <= n; j++)
      {
         cost[i][j] = cost[j][i] = 99999;
      } }
   printf("Enter the number of edges : ");
   scanf("%d", &e);
   for (int i = 0; i < e; i++)
   {
      printf("Source : ");
      scanf("%d", &s);
      printf("Destination : ");
      scanf("%d", &d);
      printf("Cost : ");
      scanf("%d", &cost1);
      cost[s][d] = cost1;
   }
   shortestpath(v,e,n);
   return 0;
}
```

Output

```
Enter the number of vertex : 8
Enter the number of edges : 19
Source : 1
Destination : 2
Cost : 10
Source : 1
Destination : 3
Cost : 25
Source : 1
Destination : 4
Cost : 20
Source : 1
Destination : 5
Cost : 2
Source : 1
Destination : 7
Cost : 5
Source : 2
Destination : 5
Cost : 5
```

```
Source : 3
Destination : 2
Cost : 4
Source : 3
Destination : 6
Cost : 5
Source : 4
Destination : 3
Cost : 8
Source : 4
Destination : 6
Cost : 6
Source : 4
Destination : 7
Cost : 8
Source : 5
Destination : 8
Cost : 3
Source : 5
Destination : 3
Cost : 4
Source : 5
Destination : 6
Cost : 3
Source : 6
Destination : 7
Cost : 4
Source : 7
Destination : 8
Cost : 7
Source : 8
Destination : 2
Cost : 3
Source : 8
Destination : 4
Cost : 6
Source : 8
Destination : 6
Cost : 2
Vertex   Distance from Source
1        0
2        8
3        6
4        11
5        2
6        5
7        5
8        5
```

CONCLUSION:

Greedy Algorithm was studied.The programs for (a) Minimum cost spanning tree using Prim's Algorithm, (b) Minimum cost spanning tree using Kruskal's Algorithm, and (c) Single source shortest path were studied and implemented successfully.