VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K·S·I·SIVANI Roll No. -052 Page No. 54

PRELAB QUESTIONS-5

1) How Greedy approach different from Divide & conquer?

Divide and Conquer	Greedy Algorithm.
→ Finds the sol^n but does not aim to find optimal sol^n.	→ Tries to find an optimal sol^n from set of feasible sol^n.
→ divides the problem into small sub problems and each problem is solved independently.	→ It is solved to get feasible sol^n; then we shall determine optimal sol^n.
→ Recursive in nature, slower and inefficient.	→ Iterative in nature; and faster.

2) Write the control abstraction for Greedy approach:

```
Algorithm Greedy (a, n)
{ solution := 0
  for i := 1 to n do
```

{ x:=select(a);
if feasible (solution, x) then
solution:=union(solution,x);}
return solution;}

3) Differentiate between feasible solⁿ & optimal solⁿ:

→ feasible solⁿ satisfies all constraints

→ optimal solⁿ which is a feasible solⁿ which optimizes objective fn.

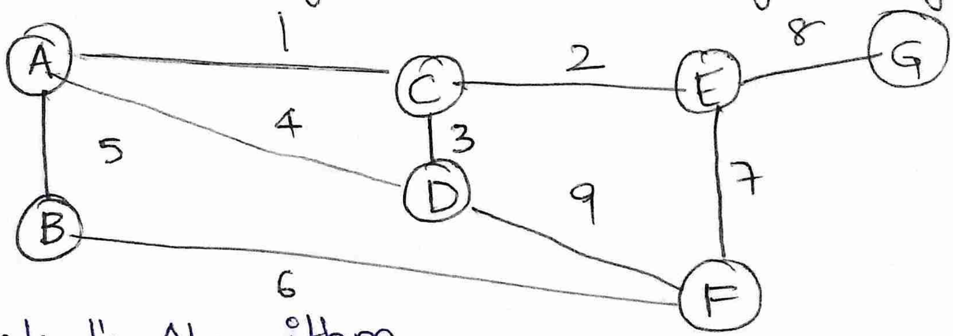
4) What is the time complexity of Prim's algorithm?

→ Time complexity of Prim's algorithm is $O((E+V)\log V)$ where $E \rightarrow$ no. of edges, $V \rightarrow$ no. of vertices.

5) Write the significance of union and find operation in Kruskal's algorithm.

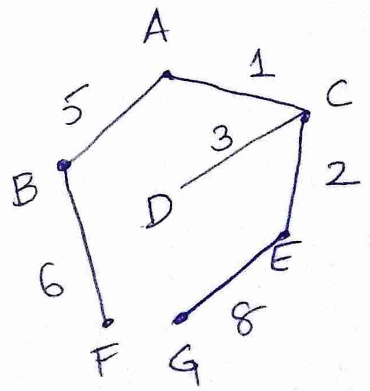
→ It merges 2 different subsets into a single subset and the representative of one set becomes representative of another. The disjoint-set also supports one other important operation called MakeSet, which creates a set containing only a given element in it.

6) Find minimum spanning tree by using Prim's and Kruskal's algorithms for the given graph.

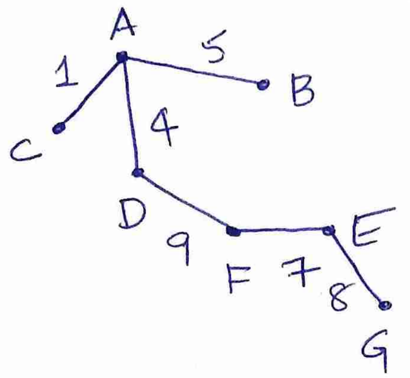


* Kruskal's Algorithm

<A,C> = 1
<C,E> = 2
<C,D> = 3
<A,D> = 4
<A,B> = 5
<B,F> = 6
<F,E> = 7
<E,G> = 8
<D,F> = 9



$\boxed{\text{Cost} = 25}$



$\boxed{\text{Cost} = 34}$

PRELAB PROGRAMS – 5:

1) Implement Kruskal's algorithm using C:

```
#include<stdio.h>
#include<stdlib.h>
int comparator(const void *p1,const void *p2)
{   const int (*x)[3] =p1;
    const int (*y)[3] =p2;

    return (*x)[2] - (*y)[2]; }

void makeSet(int parent[], int rank[], int n)
{  for(int i=0 ; i<n; i++)
      { parent[i] =i;
        rank[i]=0; }}
```

```
int findParent(int parent[], int component)
{ if(parent[component]==component)
      return component;
   return parent[component]=findParent(parent,
                                parent[component]);
}
void unionSet(int u, int v, int parent[],
                  int rank[], int n)
{ u= findParent(parent,u);
  v= findParent(parent,v);
  if(rank[u]<rank[v]){ parent[u]=v;}
  else if(rank[u]>rank[v]){ parent[v]=u;}
  else { parent[v]=u; rank[u]++;}}
void Kruskal(int n, int edge[n][3])
{ qsort(edge,n,sizeof(edge[0]),comparator);
  int parent[n],rank[n];
  makeSet(parent,rank,n);
```

```
int minCost=0;
print("Following are the edges in the spanning
        tree:\n");
for(int i=0;i<n;i++)
{ int v1 = findParent(parent, edge[i][0]);
  int v2 = findParent (parent, edge[i][1]);
  int wt = edge[i][2];
  if(v1!=v2)
  { unionSet(v1,v2, parent,rank,n);
    minCost+=wt;
    printf("%d--%d==%d\n", edge[i][0],
                    edge[i][1],wt);}}
printf("Minimum Cost Spanning Tree:%d\n",
                    minCost);}
int main()
{ int edge[5][3] = {{0,1,10},{0,2,6},{0,3,5},
                    {1,3,15},{2,3,4}};

Kruskal(5,edge);
return 0;}
```

Output:

Following are the edges in the spanning tree

2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10

Minimum Cost Spanning Tree : 19.

2) Implement Prim's algorithm in C.

```
#include <limits.h>
#include <stdbool.h>
#include<stdio.h> # define V 5
int minKey (int key[], bool, mstSet[])
{ int min = INT_MAX, min_index;
  for(int v=0; v<V; v++)
   { if (mtSet[v] == false && key[v] < min)
        { min = key[v], min_index = v; } }
  return min_index; }
int printMST (int parent [], int graph[v][v])
 { printf("Edge \t Weight \n");
   for(int i=1; i<V; i++)
    { printf("%d - %d \t %d \n", parent[i], i,
                              graph[i][parent[i]]); }
void primMST (int graph[V][V])
 { int parent[V]; int key[v]; bool mstSet[V];
```

```
for(int i=0; i<V; i++)
  { key[i]= INT_MAX, mstSet[i]=false; }
 key[0]=0; parent[0]=-1;
for (int count=0; count<V-1; count++)
 { int u=minKey(key, mstSet);
   mstSet[u]=true;
   for (int v=0; v<V; v++)
   { if(graph[u][v] && mstSet[v]==false &&
                 graph[u][v] <key[v])
      { parent[v]=u; key[v]=graph[u][v]; }}
   printMST(parent, graph); }
int main()
{ int graph[V][V]={{0,2,0,6,0},{2,0,3,8,5},
                   {0,3,0,0,7},{6,8,0,0,9},
                   {0,5,7,9,0}};
  primMST(graph); return 0; }
```

Output:

Edge	Weight
0 – 1	2
1 – 2	3
0 – 3	6
1 – 4	5.