Week 8

Q1) Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm)

Input Format:

The first line of input takes number of vertices in the graph.

Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:

Output will be minimum spanning weight

```
#include <bits/stdc++.h>
#define ll long long
#define INF INT_MAX
using namespace std;
int prims(int **arr, int n)
{
   vector<bool> visited(n, false);
   vector<int> weight(n, INF);
   priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
min_heap;
   int src = 0;
   weight[src] = 0;
   min_heap.push(make_pair(weight[src], src));
   while (!min_heap.empty())
   {
      int u = min_heap.top().second;
      min_heap.pop();
      if (!visited[u])
      {
         visited[u] = true;
         for (int v = 0; v < n; ++v)
         {
            if (!visited[v] && arr[u][v] != 0 && arr[u][v] < weight[v])
            {
               weight[v] = arr[u][v];
               min_heap.push(make_pair(weight[v], v));
            }
```

```
            }
        }
    }

    int sum = 0;
    for (auto i : weight)
        sum += i;
    return sum;
}
int main()
{
    int n;
    cin >> n;
    int **arr;
    arr = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; ++i)
        arr[i] = (int *)malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> arr[i][j];
    cout << "Minimum spanning weight : " << prims(arr, n);
    return 0;
}
```

OUTPUT

```
7
0 0 7 5 0 0 0
0 0 8 5 0 0 0
7 8 0 9 7 0 0
5 0 9 0 15 6 0
0 5 7 15 0 8 9
0 0 0 6 8 0 11
0 0 0 0 9 11 0
Minimum spanning weight : 39
```

Q2) Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Kruskal algorithm)

Input Format:

The first line of input takes number of vertices in the graph.

Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:

Output will be minimum spanning weight

```
#include <bits/stdc++.h>
#define NIL -1
using namespace std;
int findParent(vector<int> parent, int u)
{
   if (parent[u] < 0)
      return u;
   return findParent(parent, parent[u]);
}
bool UnionByWeight(vector<int> &parent, int u, int v)
{
   int pu = findParent(parent, u);
   int pv = findParent(parent, v);
   if (pu != pv)
   {
      if (parent[pu] <= parent[pv])
      {
         parent[pu] += parent[pv];
         parent[pv] = pu;
      }
      else
      {
         parent[pv] += parent[pu];
         parent[pu] = pv;
      }
      return true;
   }
   return false;
}
```

```
int kruskals(int **graph, int n)
{
   vector<pair<int, pair<int, int>>> G;
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
         if (graph[i][j] != 0)
            G.push_back(make_pair(graph[i][j], make_pair(i, j)));
   sort(G.begin(), G.end());
   vector<int> parent(n, NIL);
   int s = 0;
   for (auto i : G)
   {
      int u = i.second.first;
      int v = i.second.second;
      int w = i.first;
      if (UnionByWeight(parent, u, v))
         s += w;
   }
   return s;
}
int main()
{
   int n;
   cin >> n;
   int **graph;
   graph = (int **)malloc(n * sizeof(int *));
   for (int i = 0; i < n; ++i)
      graph[i] = (int *)malloc(n * sizeof(int));
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
         cin >> graph[i][j];
   cout << "Minimum spanning weight : " << kruskals(graph, n) << endl;
   return 0;
}
```

OUTPUT

```
7
0 0 7 5 0 0 0
0 0 8 5 0 0 0
7 8 0 9 7 0 0
5 0 9 0 15 6 0
0 5 7 15 0 8 9
0 0 0 6 8 0 11
0 0 0 0 9 11 0
Minimum spanning weight : 39
```

Q3) Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.

Input Format:

The first line of input takes number of vertices in the graph.

Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:

Out will be maximum spanning weight.

```
#include <bits/stdc++.h>
#define NIL -1

using namespace std;

int findParent(vector<int> parent, int u)
{
   if (parent[u] < 0)
      return u;
   return findParent(parent, parent[u]);
}

bool UnionByWeight(vector<int> &parent, int u, int v)
{
   int pu = findParent(parent, u);
   int pv = findParent(parent, v);
   if (pu != pv)
   {
      if (parent[pu] <= parent[pv])
      {
         parent[pu] += parent[pv];
         parent[pv] = pu;
      }
      else
      {
         parent[pv] += parent[pu];
         parent[pu] = pv;
      }
      return true;
   }
   return false;
}

int kruskals(int **graph, int n)
```

```
{
   vector<pair<int, pair<int, int>>> G;
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
         if (graph[i][j] != 0)
            G.push_back(make_pair(graph[i][j], make_pair(i, j)));
   sort(G.begin(), G.end(), greater<pair<int, pair<int, int>>>());
   vector<int> parent(n, NIL);
   int s = 0;
   for (auto i : G)
   {
      int u = i.second.first;
      int v = i.second.second;
      int w = i.first;
      if (UnionByWeight(parent, u, v))
         s += w;
   }
   return s;
}

int main()
{
   int n;
   cin >> n;
   int **graph;
   graph = (int **)malloc(n * sizeof(int *));
   for (int i = 0; i < n; ++i)
      graph[i] = (int *)malloc(n * sizeof(int));
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
         cin >> graph[i][j];
   cout << "Minimum spanning weight : " << kruskals(graph, n) << endl;
   return 0;
}
```

OUTPUT

```
7
0 0 7 5 0 0 0
0 0 8 5 0 0 0
7 8 0 9 7 0 0
5 0 9 0 15 6 0
0 5 7 15 0 8 9
0 0 0 6 8 0 11
0 0 0 0 9 11 0
Minimum spanning weight : 59
```