WEEK-6

Q-1

```
#include <bits/stdc++.h>
using namespace std;
void dfs(vector<int> arr[], int source, int V, bool *visited)
{
   visited[source] = true;
   for (int i = 0; i < V; i++)
   {
      if (arr[source][i] != 0 && !visited[i])
      {
         dfs(arr, i, V, visited);
      }
   }
}
bool checkPath(vector<int> arr[], int V, int source, int destination)
{
   bool visited[V];
   for (int i = 0; i < V; i++)
      visited[i] = false;
   dfs(arr, source, V, visited);
   return visited[destination];
}
int main()
{
   int n;
   cin >> n;
   vector<int> arr[n];
   int temp;
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         cin >> temp;
         arr[i].push_back(temp);
      }
   }
   int source, destination;
   cin >> source >> destination;
```

```
    if (checkPath(arr, n, source - 1, destination - 1))
    {
        cout << "Yes Path Exists.\n";
    }
    else
    {
        cout << "No Such Path Exists.\n";
    }
    return 0;
}
```

OUTPUT

```
5
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0
1 5
Yes Path Exists.
```

Q-2

```
#include <bits/stdc++.h>
using namespace std;
bool isBipartiteUtil(vector<int> G[], int src, int colorArr[], int V)
{
   colorArr[src] = 1;
   queue<int> q;
   q.push(src);
   while (!q.empty())
   {
      int u = q.front();
      q.pop();
      if (G[u][u] == 1)
         return false;
      for (int v = 0; v < V; ++v)
      {
         if (G[u][v] != 0 && colorArr[v] == -1)
         {
            colorArr[v] = 1 - colorArr[u];
            q.push(v);
         }
         else if (G[u][v] != 0 && colorArr[v] == colorArr[u])
            return false;
      }
   }
   return true;
}

bool isBipartite(vector<int> G[], int V)
{
   int colorArr[V];
   for (int i = 0; i < V; ++i)
      colorArr[i] = -1;
   for (int i = 0; i < V; i++)
      if (colorArr[i] == -1)
         if (isBipartiteUtil(G, i, colorArr, V) == false)
            return false;
   return true;
}
```

```
int main()
{
    int n;
    cin >> n;
    vector<int> G[n];
    int temp;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> temp;
            G[i].push_back(temp);
        }
    }
    if (isBipartite(G, n))
    {
        cout << "Yes Bipartite\n";
    }
    else
    {
        cout << "Not Bipartite\n";
    }
    return 0;
}
```

OUTPUT

```
5
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0
Not Bipartite
```

Q-3

```
#include <bits/stdc++.h>
using namespace std;
bool CheckCycle(int node, vector<int> adj[], int vis[], int dfsvis[])
{
   vis[node] = 1;
   dfsvis[node] = 1;
   for (auto it : adj[node])
   {
      if (!vis[it])
      {
         if (CheckCycle(it, adj, vis, dfsvis))
            return true;
      }
      else if (dfsvis[it])
         return true;
   }
   dfsvis[node] = 0;
   return false;
}
bool isCycle(vector<int> adj[], int N)
{

   int vis[N + 1], dfsVis[N + 1];
   memset(vis, 0, sizeof(vis));
   memset(dfsVis, 0, sizeof(dfsVis));
   for (int i = 1; i <= N; i++)
   {
      if (!vis[i])
      {
         if (CheckCycle(i, adj, vis, dfsVis))
            return true;
      }
   }
   return false;
}
int main()
{
   int n, m;
   cin >> n >> m;
```

```
    vector<int> adj[n + 1];

    for (int i = 1; i <= m; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
    }

    if (isCycle(adj, n))
        cout << "Cycle Exists" << endl;
    else
        cout << "No Cycle Exists" << endl;
    return 0;
}
```

OUTPUT

```
5
0 1 1 0 0
No Cycle Exists
```

WEEK-7

Q-1

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int minDisIndex(int *dis,bool *vis,int v){
   int i;
   int minDis=INT_MAX;
   int minIndex=-1;
   for(i=0;i<v;i++)
   {
      if(vis[i]==false && dis[i]<=minDis)
      {
         minDis=dis[i];
         minIndex=i;
      } }
   return minIndex;
}
void dijkstra(vector<vector<int>> mat,int v,int s){
   int dis[v];
   bool vis[v];
   int parent[v];
   int i,j;
   for(i=0;i<v;i++)
   {
      dis[i]=INT_MAX;
      vis[i]=false;
      parent[i]=-1;
   }
   dis[s]=0;
   parent[s]=s;
   for(i=0;i<v;i++)
   {
      int m=minDisIndex(dis,vis,v);
      vis[m]=true;
      for(j=0;j<v;j++){
         if(dis[m]!=INT_MAX && !vis[j] && mat[m][j])
         {
            if(dis[j]>dis[m]+mat[m][j]) {
```

```
                dis[j]=dis[m]+mat[m][j];
                parent[j]=m;
            } }  } }
    for(i=0;i<v;i++)   {
        if(i==s)     {
            cout<<i+1<<" : "<<dis[i]<<endl;
            continue;
        }
        cout<<i+1;
        j=i;
        while(parent[j]!=s)    {
            cout<<" "<<parent[j]+1;
            j=parent[j];
        }
        cout<<" "<<s+1<<" : "<<dis[i]<<endl;
    }
}
int main(){
    int i,j;
    int v;
    cin>>v;
    vector<vector<int>> mat(v,vector<int> (v));
    for(i=0;i<v;i++)
    for(j=0;j<v;j++)
    cin>>mat[i][j];
    int s;
    cin>>s;
    dijkstra(mat,v,s-1);
    return 0;}
```

OUTPUT

```
5
0 4 1 0 0
0 0 0 0 4
0 2 0 4 0
0 0 0 0 4
0 0 0 0 0
1
1 : 0
2 3 1 : 3
3 1 : 1
4 3 1 : 5
5 2 3 1 : 7
```

Q-2

```
#include <bits/stdc++.h>
using namespace std;
void calulate(vector<int> &pa, int i)
 {
   cout << i + 1 << " ";
   if (pa[i] >= 0)
      calulate(pa, pa[i]);
}
void find_path(int **graph, int m, int sour)
 {
   vector<int> dis(m, INT_MAX), pa(m, -1);
   dis[sour] = 0;
   for (int ki = 0; ki < m - 1; ki++)
      {
      for (int i = 0; i < m; i++)
            {
         for (int j = 0; j < m; j++)
            {
            if (graph[i][j] != 0)
                {
               if (dis[j] > dis[i] + graph[i][j])
                  {
                  dis[j] = dis[i] + graph[i][j];
                  pa[j] = i;
                       }
                   }
                }
            }
      }
   for (int i = 0; i < m; i++)
  {
      calulate(pa, i);
      cout << ": " << dis[i] << endl;
   }
}

int main()
{
   int m, source, ed;
```

```
    cin >> m;
    int **graph = (int **)malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++)
        graph[i] = (int *)malloc(m * sizeof(int));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            cin >> graph[i][j];
        }
    }
    cin >> source;
    find_path(graph, m, source - 1);
}
```

OUTPUT

```
5
0 4 1 0 0
0 0 0 0 4
0 2 0 4 0
0 0 0 0 4
0 0 0 0 0
1
1 : 0
2 3 1 : 3
3 1 : 1
4 3 1 : 5
5 2 3 1 : 7
```

Q-3

```
#include <bits/stdc++.h>
using namespace std;
int shortest_weigt(int **graph, int ver, int u, int v, int k)
{
   if (k <= 0)
      return INT_MAX;
   if (k == 0 && u == v)
      return 0;
   if (k == 1 && graph[u][v] != INT_MAX)
      return graph[u][v];
   int res = INT_MAX;
   for (int i = 0; i < ver; i++) {
      if (graph[u][i] != 0 && u != i && v != i) {
         int recu = shortest_weigt(graph, ver, i, v, k - 1);
         if (recu != INT_MAX)
            res = min(res, graph[u][i] + recu);
      }
   }
   return res;
}
int main()
{
   int ver, u, v, k, ans;
   cin >> ver;
   int **graph = (int **)malloc(ver * sizeof(int *));
   for (int i = 0; i < ver; i++)
      graph[i] = (int *)malloc(sizeof(int) * ver);
   for (int i = 0; i < ver; i++)
      for (int j = 0; j < ver; j++)
         cin >> graph[i][j];
   cin >> u >> v >> k;
   ans = shortest_weigt(graph, ver, u - 1, v - 1, k);
   cout << "Weight of shortest path from (" << u
       << "," << v << ") with " << k << " edges :" << ans;
}
```

OUTPUT

```
4
0 10 3 2
0 0 0 7
0 0 0 6
0 0 0 0
1 4
2
Weight of shortest path from (1,4) with 2 edges :9
```

WEEK-8

Q-1

```
#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
```

Q-2

```
#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
```

Q-3

```
#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
```

WEEK-9

Q-1

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

int main()
{
   int n, i, j, k, w;
   cin >> n;
   int graph[n][n];
   string temp;
   for (i = 0; i < n; i++)
  {
      for (j = 0; j < n; j++)
     {
         cin >> temp;
         if (temp != "INF")
        {
            graph[i][j] = stoi(temp);
         } else {
            graph[i][j] = 1e8;
         }
      }
   }
   for (k = 0; k < n; k++)
   {
      for (i = 0; i < n; i++)
       {
         for (j = 0; j < n; j++)
        {
            if (graph[i][k] + graph[k][j] < graph[i][j])
              {
               graph[i][j] = graph[i][k] + graph[k][j];
                }
          }
       }
    }
   cout << "The shortest path matrix: " << endl;
   for (i = 0; i < n; i++)
```

```
        {
      for (j = 0; j < n; j++)
            {
         if(graph[i][j] >= 1e8) cout << "INF";
         else cout << graph[i][j];
         cout << " ";
          }
      cout << endl;
    }
   return 0;
}
```

OUTPUT

```
5
0 10 5 5 INF
INF 0 5 5 5
INF INF 0 INF 10
INF INF INF 0 20
INF INF INF 5 0
The shortest path matrix:
0 10 5 5 15
INF 0 5 5 5
INF INF 0 15 10
INF INF INF 0 20
INF INF INF 5 0
```

Q-2

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   int n;
   cin >> n;
   vector<double> items(n);
   vector<double> val(n);
   vector<vector<double>> job;
   for (int i = 0; i < n; i++)
   {
      cin >> items[i];
   }
   for (int i = 0; i < n; i++)
  {
      cin >> val[i];
      job.push_back({val[i] / items[i], items[i], (double)(i + 1)});
   }
   double k;
   cin >> k;
   sort(job.rbegin(), job.rend());
   vector<pair<double, double>> ls;
   float profit = 0;
   for (int i = 0; i < n; i++)
     {
      if (job[i][1] >= k)
         {
         profit += k * job[i][0];
         ls.push_back(make_pair(k, job[i][2]));
         break;
         }
      else
         {
         profit += job[i][1] * job[i][0];
          }
      ls.push_back(make_pair(job[i][1], job[i][2]));
      k = k - job[i][1];
     }
   cout << "Maximum Value : " << profit << endl;
```

```
    cout << "Item - Weight" << endl;
    for (auto it : ls)
    cout << it.second << " - " << it.first << endl;
    return 0;
}
```

OUTPUT

```
6
6 10 3 5 1 3
6 2 1 8 3 5
16
Maximum Value : 22.3333
Item - Weight
5 - 1
6 - 3
4 - 5
1 - 6
3 - 1
```

Q-3

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   int n;
   cin >> n;
   vector<int> a(n);
   for (int i = 0; i < n; i++)
   {
      cin >> a[i];
   }
   priority_queue<int, vector<int>, greater<int>> minheap;
   for (int i = 0; i < n; i++) {
      minheap.push(a[i]);
   }
   int ans = 0;
   while (minheap.size() >1)
   {
      int e1 = minheap.top();
      minheap.pop();
      int e2 = minheap.top();
      minheap.pop();
      ans += e1 + e2;
      minheap.push(e1 + e2);
   }
   cout << ans;
   return 0;
}
```

OUTPUT

```
10
10 5 100 50 20 15 5 20 100 10
895
```

WEEK-10

Q-1

```
#include<bits/stdc++.h>
using namespace std;
int main() {
   int n;
   cin>>n;
   int i,s[n],f[n];
   for(i=0;i<n;i++)
   cin>>s[i];
   for(i=0;i<n;i++)
   cin>>f[i];
   vector<vector<int>> a;
   vector<int> act;
   for(i=0;i<n;i++)
   a.push_back({f[i],s[i],i+1});
   sort(a.begin(),a.end());
   int e=INT_MIN,c=0;
   for(i=0;i<n;i++)
   {
      if(a[i][1]>=e)
      {
         e=a[i][0];
         c++;
         act.push_back(a[i][2]);
      }
   }
   cout<<"No. of non-conflicting activities : "<<c<<endl;
   cout<<"List of selected activities : ";
   for(i=0;i<act.size();i++)
   cout<<act[i]<<",";
   return 0;
}
```

OUTPUT

```
10
1 3 0 5 3 5 8 8 2 12
4 5 6 7 9 9 11 12 14 16
No. of non-conflicting activities : 4
List of selected activities : 1,4,7,10,
```

Q-2

```
#include<bits/stdc++.h>
using namespace std;
int main(){
   int n;
   cin>>n;
   int i,t[n],f[n];
   for(i=0;i<n;i++)
   cin>>t[i];
   for(i=0;i<n;i++)
   cin>>f[i];
   vector<vector<int>> a;
   vector<int> act;
   for(i=0;i<n;i++)
   a.push_back({f[i],f[i]-t[i],i+1});
   sort(a.begin(),a.end());
   int e=INT_MIN,c=0;
   for(i=0;i<n;i++)
   {
      if(a[i][1]>=e)
      {
         e=a[i][0];
         c++;
         act.push_back(a[i][2]);
      }
   }
   sort(act.begin(),act.end());
   cout<<"Max number of tasks : "<<c<<endl;
   cout<<"Selected task Numbers : ";
   for(i=0;i<act.size();i++)
   cout<<act[i]<<",";
   return 0;}
```

OUTPUT

```
7
2 1 3 2 2 2 1
2 3 8 6 2 5 3
Max number of tasks : 4
Selected task Numbers : 1,2,3,6,
```

Q-3

```
#include<bits/stdc++.h>
using namespace std;
int main(){
   int n;
   cin>>n;
   int i,a[n],c,j;
   for(i=0;i<n;i++)
   cin>>a[i];
   bool f=0;
   sort(a,a+n);
   for(i=0;i<n;i++)
   {
      c=1;
      j=i+1;
      while(j<n && a[j++]==a[i])
         c++;
      if(c>n/2)
      {
         cout<<"yes\n";
         f=1;
         break;
      }
      i=j-1;
   }
   if(f==0)
   cout<<"no\n";
   if(n%2!=0)
   cout<<a[n/2];
   else
   cout<<((float)a[n/2]+a[n/2-1])/2;
   return 0;
}
```

OUTPUT

```
9
4 4 2 3 2 2 3 2 2
yes
2
```

WEEK-11

Q-1

```
#include<bits/stdc++.h>
using namespace std;
long matChainOrder(int *p,int n)
  {

    int m[n][n];
    int i,j,k,l,q;

  for(i=1;i<n;i++)
    m[i][i]=0;
    for(l=2;l<n;l++)
    {
        for(i=1;i<n-l+1;i++)
        {
            j=i+l-1;
            m[i][j]=INT_MAX;
            for(k=i;k<=j-1;k++)
            {
                q=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
                if(q<m[i][j])
                m[i][j]=q;
            }
        }
    }
 return m[1][n-1];
}

int main()
{

int n;
    cin>>n;
    int p[n+1];
    for(int i=0;i<n;i++)
   {
        cin>>p[i]>>p[i+1];
    }
```

```
  cout<<matChainOrder(p,n+1);

return 0;
}
```

OUTPUT

```
3
10 30
30 5
5 60
4500
```

Q-2

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int n,amt;
   cin>>n;
   int i,j,a[n];
   for(i=0;i<n;i++)
   cin>>a[i];
   cin>>amt;
   int ans[amt+1];
   for(i=1;i<=amt;i++)
   ans[i]=0;
   ans[0]=1;
   for(j=0;j<n;j++)
   {
      for(i=1;i<=amt;i++)
      {
         if(a[j]<=i)
         ans[i]+=(ans[i-a[j]]);
      }
   }
   cout<<ans[amt];
   return 0;
}
```

OUTPUT

```
4
2 5 6 3
10
5
```

Q-3

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

int main()
 {
   int n;
   cin>>n;
   int i,j,a[n];
   for(i=0;i<n;i++)
   cin>>a[i];
   int sum=0;
   for(i=0;i<n;i++)
   sum+=a[i];
   if(sum%2!=0)
   {
      cout<<"no";
      return 0;
   }

   sum=sum/2;
   bool s[n+1][sum+1];
   for(i=0;i<=n;i++)
   {
      for(j=0;j<=sum;j++)
      {
         if(j==0)
         s[i][j]=1;
         else if(i==0)
         s[i][j]=0;
         else
         {
            if(a[i-1]>j)
            s[i][j]=s[i-1][j];
            else
            s[i][j]=(s[i-1][j] || s[i-1][j-a[i-1]]);
         }
      }
   }
   if(s[n][sum])
```

```
    cout<<"yes";
    else
    cout<<"no";
    return 0;
}
```

OUTPUT

```
7
1 5 4 11 5 14 10
yes
```