```
//Max Subarray Sum


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

#define int long long

int CrossSubSum(vector<int>&arr,int left,int mid,int right){
    int sum=0;
    int leftSum=INT_MIN;

    for(int i=mid;i>=left;i--){
        sum+=arr[i];
        leftSum=max(sum,leftSum);
    }

    int rightSum=INT_MIN;
    sum=0;


    for(int i=mid+1;i<=right;i++){
        sum+=arr[i];
        rightSum=max(sum,rightSum);
    }

    return leftSum+rightSum;

}

int maxSubSum(vector<int>&arr,int left,int right){
    if(left==right){
        return arr[left];
    }
    int mid=(left+right)/2;

    int leftSum=maxSubSum(arr,left,mid);
    int rightSum=maxSubSum(arr,mid+1,right);
    int crossSum=CrossSubSum(arr,left,mid,right);

    return max({leftSum,rightSum,crossSum});

}

signed main(){
    int n;
    cin>>n;
    vector<int>arr(n);

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

    cout<<maxSubSum(arr,0,n-1);

    return 0;
}
```

```
//longest nice substring
string longest_nice(string s){
    int len=s.length();
    if(len<2){
        return "";
    }

    set<char>st;
    for(int i=0;i<len;i++){
        st.insert(s[i]);
    }

    for(int i=0;i<s.length();i++){
            char ch=s[i];
        if(st.count(tolower(ch)) &&
st.count(toupper(ch)) )
            continue;

        string s1=longest_nice(s.substr(0,i));
        string s2=longest_nice(s.substr(i+1));

        if(s1.length()>=s2.length())
            return s1;
        else return s2;
    }
    return s;
}
```

```
//edmond
using LL = long long;
vector<vector<LL>>adj;
vector<vector<LL>>capacity;

int n,m;
const LL INF=1e18;

LL bfs(vector<LL>&parent,LL s,LL t){
    fill(parent.begin(),parent.end(),-
1);
    parent[s]=-2;

    queue<pair<LL,LL>>q;
    q.push({s,INF});

    while(!q.empty()){
        LL u=q.front().first;
        LL flow=q.front().second;
        q.pop();

        for(LL v:adj[u]){
            if(parent[v]==-1 &&
capacity[u][v]>0){
                parent[v]=u;
                LL
new_flow=min(flow,capacity[u][v]);
                if(v==t){
                    return new_flow;
                }
                q.push({v,new_flow});
            }
        }
    }
    return 0;
}

LL edmond_flow(LL s,LL t){
    LL max_flow=0;
    vector<LL>parent(n);

    while(true){
        LL flow=bfs(parent,s,t);
        if(flow==0)
            break;
        LL curr=t;

        while(curr!=s){
```

```
            LL prev=parent[curr];
            capacity[prev][curr]-
=flow;

capacity[curr][prev]+=flow;
            curr=prev;
        }
        max_flow+=flow;
    }
    return max_flow;
}

void solve(int tc)
{
        cin>>n>>m;

        adj.assign(n,{});

capacity.assign(n,vector<LL>(n,0));

        LL a,b,c;
        for(int i=0;i<m;i++){
            cin>>a>>b>>c;
            a--;
            b--;
            adj[a].push_back(b);
            adj[b].push_back(a);
            capacity[a][b]+=c;
        }
        LL s,t;
        s=0;
        t=n-1;

        cout<<edmond_flow(s,t)<<"\n";

}


//dijkstra


using LL = long long;

const int N=1e5+9;
const long long INF=1e18;
vector<pair<long long,long
long>>adj[N];
vector<long long>dist(N,INF);
vector<long long>vis(N,0);



void dijkstra(int source){
     set<pair<long long,long long>>st;
      st.insert({0,source});

      dist[source]=0;

      while(st.size()>0){
         auto node=*st.begin();
         long long v=node.second;
         st.erase(st.begin());

         if(vis[v]){
             continue;
         }
         vis[v]=1;

         for(auto& u:adj[v]){
             long long wt=u.second;
             int child=u.first;


if((wt+dist[v])<dist[child]){

st.erase({dist[child],child});

dist[child]=dist[v]+wt;

st.insert({dist[child],child});
             }
         }

      }
}

void solve(int tc)
{
   int n,m;
   cin>>n>>m;

   int x,y,wt;
   for(int i=0;i<m;i++){
    cin>>x>>y>>wt;
    adj[x].push_back({y,wt});
   }

   dijkstra(1);

   for(int i=1;i<=n;i++){
    cout<<dist[i]<<" ";
   }

}


//bellmen ford
```

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

const int INF = 1e9;
struct Edge {
    int u, v, weight;
};

void bellmanFord(int V, int E, int
src, vector<Edge> &edges) {
    vector<int> dist(V, INF);
    dist[src] = 0;

    for (int i = 0; i < V - 1; i++) {
        for (auto edge : edges) {
            if (dist[edge.u] != INF &&
dist[edge.u] + edge.weight <
dist[edge.v]) {
                dist[edge.v] =
dist[edge.u] + edge.weight;
            }
        }
    }

    for (auto edge : edges) {
        if (dist[edge.u] != INF &&
dist[edge.u] + edge.weight <
dist[edge.v]) {
            cout << "Graph contains a
negative weight cycle.\n";
            return;
        }
    }

    cout << "Vertex\tDistance from
Source\n";
    for (int i = 0; i < V; i++) {
        if (dist[i] == INF)
            cout << i << "\t" <<
"INF\n";
        else
            cout << i << "\t" <<
dist[i] << "\n";
    }
}

int main() {
    int V, E;
    cout << "Enter number of vertices
and edges: ";
    cin >> V >> E;

    vector<Edge> edges(E);
    cout << "Enter each edge in
format: from to weight\n";
    for (int i = 0; i < E; i++) {
        cin >> edges[i].u >>
edges[i].v >> edges[i].weight;
    }
    int src;
    cout << "Enter source vertex: ";
    cin >> src;
    bellmanFord(V, E, src, edges);
    return 0;
}

//floyd warshall

const int INF = 1e18; const int N =
501;
int main() {
int n, m, q;
cin >> n >> m >> q;

vector<vector<long long>> dist(n+1,
```

```
vector<long long>(n+1, INF));

for (int i = 1; i <= n; ++i)
    dist[i][i] = 0;

for (int i = 0; i < m; ++i) {
    int a, b;
    long long c;
    cin >> a >> b >> c;
    dist[a][b] = min(dist[a][b], c);
    dist[b][a] = min(dist[b][a], c); //
undirected
}



for (int k = 1; k <= n; ++k)
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j)
            dist[i][j] = min(dist[i][j],
dist[i][k] + dist[k][j]);

while (q--) {
    int a, b;
    cin >> a >> b;
    if (dist[a][b] == INF) cout << -1 <<
'\n';
    else cout << dist[a][b] << '\n';
}

return 0;

}



//random



long long inside=0;
    long long n=100000000;
     mt19937 mt(time(nullptr));

    for(int i = 0; i < n; i++){
        double x = (double)mt() /
mt.max();
        double y = (double)mt() /
mt.max();



        if((x * x) + (y * y) <= 1){
            inside++;
        }
    }
```

```
cout<<(4*inside)/(1.0*n);
```