Stringfunctions:
Stringstream:
clear() — to clear the stream
str() — to get and set string object whose content is present in stream.
operator << — add a string to the stringstream object.
operator >> — read something from the stringstream object.

Using stringstream we can convert string into int,float or double data type/*int x = 0;
    ss >> x;*/

stoi(both for strings and character array) and atoi(only for character array) functions can also be used to convert a string to integer.Similarly atof ..

Using stringstreams we can split a string wrt a delimiter
```
/*stringstream check1(line);
 string intermediate;
 while(getline(check1, intermediate, ' '))
  {
    vector.push_back(intermediate);
  }
*/
```

Stringclass functions:
c_str returns null terminated char array version of string /* const char* charstr = str.c_str(); */

find returns index where pattern is found.If pattern is not there it returns predefined constant npos whose value is -1/*if (str6.find(str4) != string::npos)
            cout << "str4 found in str6 at " << str6.find(str4)<< " position" << endl;
          else
            cout << "str4 not found in str6" << endl;
                    */
compare(string_to_compare):- It is used to compare two strings. It returns the difference of second string and first string in integer

find("string"):Searches the string for the first occurrence of the substring specified in arguments. It returns the position of the first occurrence of substring.

find_first_of("string"):Searches the string for the first character that matches any of the characters specified in its arguments. It returns the position of the first character that matches.
find_last_of("string"):Searches the string for the last character that matches any of the characters specified in its arguments. It returns the position of the last character that matches.
rfind("string"):Searches the string for the last occurrence of the substring specified in arguments. It returns the position of the last occurrence of substring

substr:returns the substring of a string./* str.substr (3,5) starting from index 3 of length 5*/

KMP algorithm: //0 based indexing
```
int* pi;
int* computepi(string p){
   int m=p.length();
   pi=new int[m];
   pi[0]=-1;
   int k=-1;
   for(int q=1;q<m;q++){
      while(k>-1&&p[q]!=p[k+1])
      k=pi[k];
      if(p[q]==p[k+1])
      k=k+1;
      pi[q]=k;
   }
   return pi;
}

void kmp(string t,string p){
   int n=t.length();
   int m=p.length();
   pi=computepi(p);
   int k=-1;
   for(int i=0;i<n;i++){
      while(k>-1&&t[i]!=p[k+1])
      k=pi[k];
      if(t[i]==p[k+1])
      k=k+1;
      if(k==m-1){
         cout<<i-m+1<<" ";
```

```
            k=pi[k];
        }
    }
    cout<<"\n";
}
```

next_permutation and prev_permutation //before calling these array should be sorted

```
do {
    cout << myints[0] << ' ' << myints[1] << ' ' <<
myints[2] << '\n';
 } while (next_permutation(myints,myints+3) );
```

lower_bound and upper_bound //before calling these array should be sorted

```
low=lower_bound (v.begin(), v.end(), 20);
up=upper_bound (v.begin(), v.end(), 20); // they
return iterator not index


int modularExponentiation(int x,int n,int M)
{
    int result=1;
    while(n>0)
    {
        if(n% 2 ==1)
            result=(result * x)%M;
        x=(x*x)%M;
        n=n/2;
    }
    return result;
}

int d, x, y;  //d is gcd and Ax+By=d
void extendedEuclid(int A, int B) {
    if(B == 0) {
        d = A;
        x = 1;
        y = 0;
    }
    else {
        extendedEuclid(B, A%B);
        int temp = x;
        x = y;
        y = temp - (A/B)*y;
    }
}

int modInverse(int A,int M)
{
    return modularExponentiation(A,M-2,M);
}
```

Prime factorisation:

```
#define MAXN  100001
int spf[MAXN];
void sieve()
{
    spf[1] = 1;
    for (int i=2; i<MAXN; i++)
    spf[i] = i;
    for (int i=4; i<MAXN; i+=2)
    spf[i] = 2;
    for (int i=3; i*i<MAXN; i++)
    {
    if (spf[i] == i)
    {
            for (int j=i*i; j<MAXN; j+=i)
                        if (spf[j]==j)
                        spf[j] = i;
            }
 }
}
vector<int> getFactorization(int x)
{
            vector<int> ret;
            while (x != 1)
            {
                        ret.push_back(spf[x]);
                        x = x / spf[x];
            }
     return ret;
}
```

bfs:

```
int main() {
            int n,m,i,u,v;
            cin>>n>>m;
            vector<int> adj[n+1];
            for(i=0;i<m;i++){
                cin>>u>>v;
                adj[u].push_back(v);
                adj[v].push_back(u);
            }
            vector<int> visited(n+1,false);
            visited[1]=true;
```

```
        queue<int> q;
        q.push(1);
        while(!q.empty()){
            int x=q.front();
            q.pop();
            for(i=0;i<adj[x].size();i++){
                if(!visited[adj[x][i]]){
                    visited[adj[x][i]]=true;
                    q.push(adj[x][i]);
                }
            }
        }
        return 0;
}
```

iterative dfs:

```
void dfs(int v,vector<int> &visited,vector<vector
<int> > adj){
    stack<int> s;
    s.push(v);
    while(!s.empty()){
        int x=s.top();
        s.pop();
        if(!visited[x]){
            cout<<x<<" ";
            visited[x]=true;
        }
        for(int i=0;i<adj[x].size();i++)
        if(!visited[adj[x][i]])
        s.push(adj[x][i]);
    }
}

int main() {
        int n,m,i,u,v;
        cin>>n>>m;
        vector<vector <int> > adj(n+1);
        for(i=0;i<m;i++){
            cin>>u>>v;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        vector<int> visited(n+1,false);
        for(i=1;i<=n;i++)
        if(!visited[i])
        dfs(i,visited,adj);
        return 0;
}
```

Kruskals MST:

```
struct edge{
    int u,v,w;
};
bool comp(edge e1,edge e2){
    return (e1.w<e2.w);
}
int find(int x,int pa[]){
    if(pa[x]==x) return x;
    return pa[x]=find(pa[x],pa);
}
int main() {
        int n,m,i,u,v,w;
        cin>>n>>m;
        vector<edge> edges(m);
        for(i=0;i<m;i++){
            cin>>edges[i].u>>edges[i].v>>edges[i].w;
        }
        sort(edges.begin(),edges.end(),comp);
        int pa[n+1],rank[n+1];
        for(i=1;i<=n;i++)
        pa[i]=i,rank[i]=0;
        int j=0,ans=0;
        i=0;
        while(j<n-1){
            struct edge e=edges[i++];
            int px=find(e.u,pa);
            int py=find(e.v,pa);
            if(px!=py){
                ans=ans+e.w;
                j++;
                 if(rank[px]<rank[py]) pa[px]=py;
          else if(rank[px]>rank[py]) pa[py]=px;
          else pa[py]=px,rank[px]++;
            }
        }
        cout<<ans<<"\n";
        return 0;
}
```

Prims MST: //vertices are 1,2,3,...

```
typedef pair<int,int> pii;
struct heapnode{
    int v;
    int w;
};
```

```
int hsize;
void swap(heapnode heap[],int x,int y, int index[]){
   index[heap[x].v]=y;
   index[heap[y].v]=x;
   struct heapnode t=heap[x];
   heap[x]=heap[y];
   heap[y]=t;
}
void minheapify(heapnode heap[],int i,int index[] ){
   int sml=i;
   int l=2*i+1;
   int r=2*i+2;
   if(l<hsize&&heap[sml].w>heap[l].w)
   sml=l;
   if(r<hsize&&heap[sml].w>heap[r].w)
   sml=r;
   if(sml!=i){
      swap(heap,sml,i,index);
      minheapify(heap,sml,index);
   }
}
void deletemin(heapnode heap[],int index[]){
   if (hsize == 1)
   {
      hsize--;
      return;
   }
   swap(heap,0,hsize-1,index);
   hsize--;
   minheapify(heap,0,index);
}
void decreasekey(heapnode heap[],int i,int w,int
index[]){
   heap[i].w=w;
   while(i!=0&&heap[(i-1)/2].w>heap[i].w){
      swap(heap,(i-1)/2,i,index);
      i=(i-1)/2;
   }
}
int main() {
        int n,m,i,u,v,w;
        cin>>n>>m;
        vector<vector <pii> > adj(n);
        for(i=0;i<m;i++){
           cin>>u>>v>>w;
           u--;v--;
           adj[u].push_back(make_pair(v,w));
           adj[v].push_back(make_pair(u,w));
        }
        vector<int> visited(n,false);
        heapnode heap[n];
        for(i=0;i<n;i++){
     heap[i].v=i;
     heap[i].w=INT_MAX;
   }
   hsize=n;
   heap[0].w=0;
   int index[n];
   for(i=0;i<n;i++)
   index[i]=i;
   int ans=0;
   for(int j=0;j<n;j++){
      u=heap[0].v;
      visited[u]=true;
      ans=ans+heap[0].w;
      deletemin(heap,index);
      for(i=0;i<adj[u].size();i++){

if(!visited[adj[u][i].first]&&heap[index[adj[u][i].first]].
w>adj[u][i].second){


decreasekey(heap,index[adj[u][i].first],adj[u][i].sec
ond,index);
        }

      }
   }
   cout<<ans<<"\n";
        return 0;
}
```

Dijkstra's shortest path algorithm: //Single source shortest path algorithm
Works for only positive edge weights same as prims except

```
if(!visited[adj[u][i].first]&&heap[index[adj[u][i].first]].
w>adj[u][i].second+dist[u]){


decreasekey(heap,index[adj[u][i].first],adj[u][i].sec
ond+dist[u],index);
 }
```

dist array contains shortest path from source for every other vertex.

Bellman ford algoritm: //Single source shortest path algorithm
Works for negative edge weights and returns false if the graph contains negative weight cycles

```
INITIALIZE  SINGLE SOURCE (G,S)
for i=1 to |G.V|-1
        for each edge (u,v)
           if(v.d>u.d+w(u,v))
                v.d=u.d+w(u,v)
for each edge (u,v)
        if(v.d>u.d+w(u,v))
                return FALSE;
return TRUE;
```

FLOYD WARSHALL ALGORITHM: //all pairs shortest path problem

```
int dist[n][n];
//initilize dist array with initial weights
for(k=0;k<n;k++){
     for(i=0;i<n;i++){
        for(j=0;j<n;j++){
                if(dist[i][k]+dist[k][j]<dist[i][j])
                     dist[i][j]=dist[i][k]+dist[k][j];
        }
    }
}
```

Priority Queue: //min pq

```
struct node{
   int e;
   int p;
};
class compare{
   public:
   bool operator() (node a,node b){
      return a.p>b.p;
   }
};
int main() {
   priority_queue<node,  vector<node>, compare>
pq;
   int n;
   cin>>n;
   node x;
   for(int i=0;i<n;i++){
      cin>>x.e>>x.p;
      pq.push(x);
   }
   while(!pq.empty()){
      x=pq.top();pq.pop();
      cout<<x.e<<" "<<x.p<<"\n";
   }
       return 0;
}
```

PQ Operations:
empty,size,push,pop,top

Queue:
empty,size,push,pop,front,back

Stack:
empty,size,push,pop,top

Vector:
empty,size,push_back,pop_back,begin,end,rbegin ,rend,insert,erase,clear,at,front,back

Set:

```
typedef set<int> si;
typedef vector<int> vi;
#define tr(container,it) \
  for
(typeof(container.begin())it=container.begin();it!=c
ontainer.end();it++)
int main() {
   si s;
   vi v=vi(2,3);
   s.insert(1);
   s.insert(v.begin(),v.end());
   s.insert(9);
   s.insert(6);
   s.insert(8);
   s.erase(2);
   si::iterator it=s.find(8);
   s.erase(it,s.end());
   s.insert(11);
   s.insert(8);
   it=s.lower_bound(8);
   cout<<*it<<"\n";
   it=s.upper_bound(8);
   cout<<*it<<"\n";
```

```
    tr(s,it)
    cout<<*it<<" ";
    cout<<"\n";
        return 0;
}

Multiset:
typedef multiset<int> msi;
int main() {
    msi s;
    vi v=vi(2,3);
    s.insert(1);
    s.insert(v.begin(),v.end());
    v=vi(3,4);
    s.insert(v.begin(),v.end());
    s.insert(9);
    s.insert(6);
    s.insert(8);
    //s.erase(3); multiple elements are erased
    msi::iterator it;
    /*it=s.find(4);//returns an iterator to 1st four
    s.erase(it,s.end()); */
    cout<<s.count(3)<<" "<<s.count(4)<<"\n";
    pair<msi::iterator,msi::iterator> pmsit;
    pmsit=s.equal_range(4);
    s.erase(pmsit.first,pmsit.second);
    tr(s,it)
    cout<<*it<<" ";
    cout<<"\n";
        return 0;
}

Map:
typedef multimap<int,int> mmii;
typedef map<int,int> mii;
typedef pair<int,int> pii;
int main() {
        mii m;
        m[1]=7;
        m.insert(pii(1,4));
        m.insert(pii(5,7));
        m.insert(make_pair(3,7));
        m.insert(mii::value_type(2,8));
        tr(m,it){
            cout<<(*it).first<<" "<<(*it).second<<"\n";
        }
        //cout<<m.max_size()<<"\n";
        //m.size()
        //m.empty()
        mii::iterator it;
        it=m.find(7);
        if(it!=m.end())
        cout<<(*it).first<<" "<<(*it).second<<"\n";
        else
        cout<<"Not found\n";
        //m1.swap(m2) where m1 and m2 are maps
of same type
        mii m1=m;
        cout<<m1.size()<<"\n";
        m1.clear();
        cout<<m1.size()<<"\n";
        m.erase(4);
        cout<<m.size()<<"\n";
        tr(m,it){
            cout<<(*it).first<<" "<<(*it).second<<"\n";
        }
        //erase(iterator) erase(iterator first,iterator
last)
        return 0;
}


Multimap:
int main() {
        mmii m;
        m.insert(pii(1,5));
        m.insert(mp(3,7));
        m.insert(mmii::value_type(2,8));
        m.insert(pii(1,2));
        m.insert(pii(3,5));
        m.insert(pii(1,11));
        cout<<m.count(1)<<" "<<m.count(2)<<"
"<<m.count(3)<<"\n";
        pair<mmii::iterator,mmii::iterator> pitit;
        pitit=m.equal_range(3);
        for(mmii::iterator
it=pitit.first;it!=pitit.second;it++)
        cout<<(*it).first<<" "<<(*it).second<<"\n";
        tr(m,it)
        cout<<(*it).first<<" "<<(*it).second<<"\n";
        return 0;
}

////////----priyatam
```

```
//// union find ..written
ll getpar(ll *par,ll u){
    if(par[u]==-1)
    return u;
    else return par[u]=getpar(par,par[u]);
}
void make_union(ll u,ll v,ll *par){
  ll x=getpar(par,u);
  ll y=getpar(par,v);
  par[x]=y;
  return;
}

//////////////////

///fenwick algo gym
///1 indexing
int fen[MAX_N];

//update is adding excess val
void update(int p,int val){
        for(int i = p;i <= n;i += i & -i)
                fen[i] += val;
}
int sum(int p){
        int ans = 0;
        for(int i = p;i;i -= i & -i)
                ans += fen[i];
        return ans;
}

//////////////////dont forget to refer geeks aswell



/////////seg tree
/////////1 indexing;pishty tree;

void dfs(long long int x,struct node *tr[],long long
cc){
    struct node *temp;
    temp=tr[x];
    ind++;
    fst[x]=ind;
    walk[ind]=x;
    mgc[ind]=cc;
    while(temp!=NULL){
        if(temp->ver!=par[x]){
            par[temp->ver]=x;
            dfs(temp->ver,tr,temp->mn);

        }
        temp=temp->next;
    }

    ind++;
    snd[x]=ind;
    walk[ind]=x;
    mgc[ind]=cc;
}

 ////////////////-------dfs pishty specefic
        long long bound[1000005][2],seg[1000005];

void buildnew(long long id,long long l,long long r){
            bound[id][0]=l;
            bound[id][1]=r;
            seg[id]=0;
            if(l==r)//////////seg[id]=arr[l];
            return;
           long long  mid=(l+r)/2;
            buildnew(2*id,l,mid);
            buildnew(2*id+1,mid+1,r);
            ///////seg[id]=seg[2*id]+seg[2*id+1];or rmq
respectively;
        }

void update(long long id,long long val,long long
tmpin){

///seg[id]+=(val-arr[tmpin]);
        if(bound[id][0]>tmpin||bound[id][1]<tmpin)
            return;
        seg[id]=seg[id]^val;


            if(bound[id][0]==bound[id][1])
            return;
            update(2*id,val,tmpin);
            update(2*id+1,val,tmpin);


        }

long long ans;
long long x,y,l,r,k;
        //////////////problem specefic;
```

```
long long calculate(long long id,long long l,long long r){
   if(bound[id][0]>r||bound[id][1]<l)
   return 0;
    if(bound[id][0]>=l&&bound[id][1]<=r){

         return seg[id];
          }

         if(bound[id][0]==bound[id][1])
          return 0;
   return calculate(2*id,l,r)^calculate(2*id+1,l,r);
         }



//////////////////////


//////////-------algo gym

void build(int id = 1,int l = 1,int r = n){
         if(r ==l){         //         l + 1 == r
         s[id] = a[l];
                  return ;
         }
         int mid = (l+r)/2;
         build(id * 2, l, mid);
         build(id * 2 + 1, mid+1, r);
         s[id] = s[id * 2] + s[id * 2 + 1];
}

//So, before reading the queries, we should call build() .

//Modify function :

void modify(int p,int x,int id = 1,int l = 1,int r = n){
         s[id] += x - a[p];
         if(r ==l){         //         l = r - 1 = p
                  a[p] = x;
                  return ;
         }
         int mid = (l + r)/2;
         if(p < mid)
                  modify(p, x, id * 2, l, mid);
         else
                  modify(p, x, id * 2 + 1, mid+1, r);
}

//(We should call modify(p,   x))

//Ask for sum function :
//////////l,r correspond to id

int sum(int x,int y,int id = 1,int l = 0,int r = n){
         if(x >= r or l >= y)      return 0;
         if(x <= l && r <= y)    return s[id];
         int mid = (l+r)/2;
         return sum(x, y, id * 2, l, mid) +
                sum(x, y, id * 2 + 1, mid+1, r);
}

//////////////////

///----------lazypropogation

void upd(int id,int l,int r,int x){//      increase all members in this interval by x
         lazy[id] += x;
         s[id] += (r - l) * x;
}


void shift(int id,int l,int r){//pass update information to the children
         int mid = (l+r)/2;
         upd(id * 2, l, mid, lazy[id]);
         upd(id * 2 + 1, mid+1, r, lazy[id]);
         lazy[id] = 0;// passing is done
}



void increase(int x,int y,int v,int id = 1,int l = 0,int r = n){
         if(x >r or l > y)          return ;
         if(x <= l && r <= y){
                  upd(id, l, r, v);
                  return ;
         }
         shift(id, l, r);
         int mid = (l+r)/2;
         increase(x, y, v, id * 2, l, mid);
```

```
        increase(x, y, v, id*2+1, mid+1, r);
        s[id] = s[id * 2] + s[id * 2 + 1];
}


int sum(int x,int y,int id = 1,int l = 0,int r = n){
        if(x >= r or l >= y)     return 0;
        if(x <= l && r <= y)    return s[id];
        shift(id, l, r);
        int mid = (l+r)/2;
        return sum(x, y, id * 2, l, mid) +
              sum(x, y, id * 2 + 1, mid+1, r);
}

////////////

////////-------**---dfs**

ll visited[1000000];
void dfs(ll hd){
  ll sz=grph[hd].size();
  ll i;
  visited[hd]=1;

  mn=min(mn,cost[hd]);
  //cout<<mn;

  for(i=0;i<sz;i++){
    if(!visited[grph[hd][i]]){
      dfs(grph[hd][i]);
    }
  }

}

////////////////


////////////////
//////////----------**-trie**

////as used for sub string xor..nov

void make_trie(ll hgt,struct node *hd){
    if(hgt==0){
        return;
    }
   //cout<<hgt<<" ";
    struct node *tmp;
    tmp=new(node);
    tmp->val=tmp->cnt=0;
    hd->rgt=tmp;
    tmp=new(node);
    tmp->val=tmp->cnt=0;
    hd->lft=tmp;

    make_trie(hgt-1,hd->lft);
    make_trie(hgt-1,hd->rgt);

}

void insertintotr(ll vl,struct node *hd){
    ll ht=20;////////////////

    while(ht!=0){
        (hd->cnt)++;
        if(vl&(1<<(ht-1))){
            hd=hd->lft;
        }else{
            hd=hd->rgt;
        }
        ht--;
    }

    hd->val=vl;
    (hd->cnt)++;
}
/////////--end 1 tr


////////---------sparse table for rmq..order nlogn.

void process2(int M[MAXN][LOGMAXN], int
A[MAXN], int N)
  {
      int i, j;

  //initialize M for the intervals with length 1
      for (i = 0; i < N; i++)
          M[i][0] = i;
  //compute values from smaller to bigger intervals
```

```
    for (j = 1; 1 << j <= N; j++)
        for (i = 0; i + (1 << j) - 1 < N; i++)
            if (A[M[i][j - 1]] < A[M[i + (1 << (j - 1))][j -
1]])
                M[i][j] = M[i][j - 1];
            else
                M[i][j] = M[i + (1 << (j - 1))][j - 1];
 }
```

```
//////////
 //////////---------
```
-combinatorics

```
ll pw(ll a, ll b,ll MOD)
   {
        if(b==0)
                return 1;
        ll temp = pw(a,b/2,MOD);
        temp = (temp*temp)%MOD;
        if(b%2 == 0)
                return temp;
        else
                return (temp*a)%MOD;
   }
ll inv_mod(ll a,ll MOD)
   {
        return pw(a,MOD-2,MOD);
   }

ll NCR(ll n, ll r,ll MOD)
   {
        ll ans = fact[n];
        ans = (ans*inv_mod(fact[r],MOD))%MOD;
        ans = (ans*inv_mod(fact[n-r],MOD))%MOD;
        return ans;
   }
```

```
long long Lucas(long long  n,long long  m,long
long  p)
{
   if (n==0 && m==0) return 1;

   int ni = n % p;
   int mi = m % p;
   if (mi>ni) return 0;
   if(m==0) return 1;
   return (Lucas(n/p, m/p, p)*NCR(ni, mi, p))%p;
}
```

```
////////////
///////-------
```
-seive of erosth.

```
int isprime[10000000],lo[10000000];
long int lim=1000000;
void primegen();

void primegen()

{

        int i,j;
        for(i=0;i<10000000;++i)
        {
        isprime[i]=1;
        lo[i]=i;
        }
        for(i=2;i<=lim;++i)
        {
        if(isprime[i])
        {
        for(j=i;j<lim;j+=i)
                        if(isprime[j])
                        {
                                isprime[j]=0;
                lo[j]=i;
                        }
                }
        }

}
#define ll long long
 ll query[1000000];
```

```
 void getqryfill(ll *pr,ll *cnt,ll index,ll fact){
 if(index<0){
   query[fact]++;
   return;
 }
 ll times=cnt[index];
 ll fct=1;
 while(times>=0){
   getqryfill(pr,cnt,index-1,fct*fact);
   fct=fct*pr[index];
   times--;
 }
```

```
}


void getcal(ll num){
  ll prime[20],count[20],ind=0;
  ll least=lo[num];
ll prev=prime[0]=least;
  count[0]=0;
  while(least>1){
    if(least==prev){
      count[ind]++;
    }
    else
    ind++,prime[ind]=least,count[ind]=1,prev=least;
    num=num/least;
    least=lo[num];
  }

  getqryfill(prime,count,ind,1);
}
//////////////////////------
```
-end combi.

```
////////----------
```
-algo for lca sq. root decomposition

```
///////nr is sq. root of height of tree which is to be pre calculated.

 ///////T[i] is the father of node i in the tree, nr is [sqrt(H)] and L[i] is the level of the node i

void dfs(int node, int T[MAXN], int N, int P[MAXN], int L[MAXN], int nr)  {
      int k;
      if (L[node] < nr)
          P[node] = 1;
      else
          if(!(L[node] % nr))
              P[node] = T[node];
          else
              P[node] = P[T[node]];

     for each son k of node
         dfs(k, T, N, P, L, nr);
  }



  int LCA(int T[MAXN], int P[MAXN], int L[MAXN], int x, int y)
   {
     while (P[x] != P[y])
         if (L[x] > L[y])
            x = P[x];
         else
             y = P[y];

   //now they are in the same section, so we trivially compute the LCA
      while (x != y)
         if (L[x] > L[y])
            x = T[x];
         else
            y = T[y];
      return x;
   }

   ////////////////////


   //////////////---------
```
Another easy solution in <O(N logN, O(logN)>

```
   //process_3 using dp in logn

   void process3(int N, int T[MAXN], int P[MAXN][LOGMAXN])
   {
      int i, j;

   //we initialize every element in P with -1
      for (i = 0; i < N; i++)
         for (j = 0; 1 << j < N; j++)
             P[i][j] = -1;

   //the first ancestor of every node i is T[i]
      for (i = 0; i < N; i++)
         P[i][0] = T[i];

   //bottom up dynamic programing
      for (j = 1; 1 << j < N; j++)
        for (i = 0; i < N; i++)
            if (P[i][j - 1] != -1)
               P[i][j] = P[P[i][j - 1]][j - 1];
   }

   /////query for logn lca
```

```
  int query(int N, int P[MAXN][LOGMAXN], int
T[MAXN],
  int L[MAXN], int p, int q)
  {
     int tmp, log, i;

  //if p is situated on a higher level than q then we
swap them
     if (L[p] < L[q])
        tmp = p, p = q, q = tmp;

  //we compute the value of [log(L[p)]
     for (log = 1; 1 << log <= L[p]; log++);
     log--;

  //we find the ancestor of node p situated on the
same level
  //with q using the values in P
     for (i = log; i >= 0; i--)
        if (L[p] - (1 << i) >= L[q])
           p = P[p][i];

     if (p == q)///////----if p and q are equal it may
return parent of p(or q) so to overcome that..
        return p;

  //we compute LCA(p, q) using the values in P
     for (i = log; i >= 0; i--)
        if (P[p][i] != -1 && P[p][i] != P[q][i])
           p = P[p][i], q = P[q][i];

     return T[p];
  }

  /////------in the above algo it is nothing but in last
for loop it executes untill pand q reach to just child
that is if lca is at dist x..it will reach upto x-1;try to
understand that.

  ////////////


  //////////////
  //////////////-----HEAVY LIGHT DECOMPOSITION

  #include <cstdio>
#include <vector>
using namespace std;

#define root 0
#define N 10100
#define LN 14

vector <int> adj[N], costs[N], indexx[N];
int baseArray[N], ptr;
int chainNo, chainInd[N], chainHead[N],
posInBase[N];
int depth[N], pa[LN][N], otherEnd[N], subsize[N];
int st[N*6], qt[N*6];

/*
 * make_tree:
 * Used to construct the segment tree. It uses the
baseArray for construction
 */
void make_tree(int cur, int s, int e) {
        if(s == e-1) {
                st[cur] = baseArray[s];
                return;
        }
        int c1 = (cur<<1), c2 = c1 | 1, m = (s+e)>>1;
        make_tree(c1, s, m);
        make_tree(c2, m, e);
        st[cur] = st[c1] > st[c2] ? st[c1] : st[c2];
}

/*
 * update_tree:
 * Point update. Update a single element of the
segment tree.
 */
void update_tree(int cur, int s, int e, int x, int val) {
        if(s > x || e <= x) return;
        if(s == x && s == e-1) {
                st[cur] = val;
                return;
        }
        int c1 = (cur<<1), c2 = c1 | 1, m = (s+e)>>1;
        update_tree(c1, s, m, x, val);
        update_tree(c2, m, e, x, val);
        st[cur] = st[c1] > st[c2] ? st[c1] : st[c2];
}

/*
 * query_tree:
 * Given S and E, it will return the maximum value
```

```
in the range [S,E)
 */
void query_tree(int cur, int s, int e, int S, int E) {
	if(s >= E || e <= S) {
		qt[cur] = -1;
		return;
	}
	if(s >= S && e <= E) {
		qt[cur] = st[cur];
		return;
	}
	int c1 = (cur<<1), c2 = c1 | 1, m = (s+e)>>1;
	query_tree(c1, s, m, S, E);
	query_tree(c2, m, e, S, E);
	qt[cur] = qt[c1] > qt[c2] ? qt[c1] : qt[c2];
}

/*
 * query_up:
 * It takes two nodes u and v, condition is that v is an ancestor of u
 * We query the chain in which u is present till chain head, then move to next chain up
 * We do that way till u and v are in the same chain, we query for that part of chain and break
 */

int query_up(int u, int v) {
	if(u == v) return 0; // Trivial
	int uchain, vchain = chainInd[v], ans = -1;
	// uchain and vchain are chain numbers of u and v
	while(1) {
		uchain = chainInd[u];
		if(uchain == vchain) {
			// Both u and v are in the same chain, so we need to query from u to v, update answer and break.
			// We break because we came from u up till v, we are done
			if(u==v) break;
			query_tree(1, 0, ptr, posInBase[v]+1, posInBase[u]+1);
			// Above is call to segment tree query function
			if(qt[1] > ans) ans = qt[1]; // Update answer
			break;
		}
		query_tree(1, 0, ptr, posInBase[chainHead[uchain]], posInBase[u]+1);
		// Above is call to segment tree query function. We do from chainHead of u till u. That is the whole chain from
		// start till head. We then update the answer
		if(qt[1] > ans) ans = qt[1];
		u = chainHead[uchain]; // move u to u's chainHead
		u = pa[0][u]; //Then move to its parent, that means we changed chains
	}
	return ans;
}

/*
 * LCA:
 * Takes two nodes u, v and returns Lowest Common Ancestor of u, v
 */
int LCA(int u, int v) {
	if(depth[u] < depth[v]) swap(u,v);
	int diff = depth[u] - depth[v];
	for(int i=0; i<LN; i++) if( (diff>>i)&1 ) u = pa[i][u];
	if(u == v) return u;
	for(int i=LN-1; i>=0; i--) if(pa[i][u] != pa[i][v]) {
		u = pa[i][u];
		v = pa[i][v];
	}
	return pa[0][u];
}

void query(int u, int v) {
	/*
	 * We have a query from u to v, we break it into two queries, u to LCA(u,v) and LCA(u,v) to v
	 */
	int lca = LCA(u, v);
	int ans = query_up(u, lca); // One part of path
	int temp = query_up(v, lca); // another part of path
	if(temp > ans) ans = temp; // take the maximum of both paths
```

```
	printf("%d\n", ans);
}

/*
 * change:
 * We just need to find its position in segment tree and update it
 */
void change(int i, int val) {
	int u = otherEnd[i];
	update_tree(1, 0, ptr, posInBase[u], val);
}

/*
 * Actual HL-Decomposition part
 * Initially all entries of chainHead[] are set to -1.
 * So when ever a new chain is started, chain head is correctly assigned.
 * As we add a new node to chain, we will note its position in the baseArray.
 * In the first for loop we find the child node which has maximum sub-tree size.
 * The following if condition is failed for leaf nodes.
 * When the if condition passes, we expand the chain to special child.
 * In the second for loop we recursively call the function on all normal nodes.
 * chainNo++ ensures that we are creating a new chain for each normal child.
 */
/////HLD
void HLD(int curNode, int cost, int prev) {
	if(chainHead[chainNo] == -1) {
		chainHead[chainNo] = curNode; // Assign chain head
	}
	chainInd[curNode] = chainNo;
	posInBase[curNode] = ptr; // Position of this node in baseArray which we will use in Segtree
	baseArray[ptr++] = cost;

	int sc = -1, ncost;
	// Loop to find special child
	for(int i=0; i<adj[curNode].size(); i++) if(adj[curNode][i] != prev) {
		if(sc == -1 || subsize[sc] < subsize[adj[curNode][i]]) {
			sc = adj[curNode][i];
			ncost = costs[curNode][i];
		}
	}

	if(sc != -1) {
		// Expand the chain
		HLD(sc, ncost, curNode);
	}

	for(int i=0; i<adj[curNode].size(); i++) if(adj[curNode][i] != prev) {
		if(sc != adj[curNode][i]) {
			// New chains at each normal node
			chainNo++;
			HLD(adj[curNode][i], costs[curNode][i], curNode);
		}
	}
}

/*
 * dfs used to set parent of a node, depth of a node, subtree size of a node
 */
void dfs(int cur, int prev, int _depth=0) {
	pa[0][cur] = prev;
	depth[cur] = _depth;
	subsize[cur] = 1;
	for(int i=0; i<adj[cur].size(); i++)
		if(adj[cur][i] != prev) {
			otherEnd[indexx[cur][i]] = adj[cur][i];
			dfs(adj[cur][i], cur, _depth+1);
			subsize[cur] += subsize[adj[cur][i]];
		}
}


 int main() {
	int t;
	scanf("%d ", &t);
	while(t--) {
		ptr = 0;
		int n;
		scanf("%d", &n);
		// Cleaning step, new test case
```

```
		for(int i=0; i<n; i++) {
			adj[i].clear();
			costs[i].clear();
			indexx[i].clear();
			chainHead[i] = -1;
			for(int j=0; j<LN; j++) pa[j][i] =
-1;
		}
		for(int i=1; i<n; i++) {
			int u, v, c;
			scanf("%d %d %d", &u, &v,
&c);
			u--; v--;
			adj[u].push_back(v);
			costs[u].push_back(c);
			indexx[u].push_back(i-1);
			adj[v].push_back(u);
			costs[v].push_back(c);
			indexx[v].push_back(i-1);
		}

		chainNo = 0;
		dfs(root, -1); // We set up subsize,
depth and parent for each node
		HLD(root, -1, -1); // We decomposed
the tree and created baseArray
		make_tree(1, 0, ptr); // We use
baseArray and construct the needed segment tree

		// Below Dynamic programming code
is for LCA.
		for(int i=1; i<LN; i++)
			for(int j=0; j<n; j++)
				if(pa[i-1][j] != -1)
					pa[i][j] =
pa[i-1][pa[i-1][j]];

		while(1) {
			char s[100];
			scanf("%s", s);
			if(s[0]=='D') {
				break;
			}
			int a, b;
			scanf("%d %d", &a, &b);
			if(s[0]=='Q') {
				query(a-1, b-1);
			} else {
				change(a-1, b);
			}
		}
	}
}
```

//////////////END HEAVY LIGHT DEC.

////////-----sos dp brute force

```
for(int mask = 0;mask < (1<<N); ++mask){
	for(int i = 0;i < (1<<N); ++i){
		if((mask&i) == i){
			F[mask] += A[i];
		}
	}
}

//order=(4^n);
```

//----------Suboptimal Solution

```
// iterate over all the masks
for (int mask = 0; mask < (1<<n); mask++){
	F[mask] = A[0];
  // iterate over all the subsets of the mask
  for(int i = mask; i > 0; i = (i-1) & mask){
	F[mask] += A[i];
  }
}

//order--3^n;
```

///-----sos dp solution

```
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
	dp[mask][-1] = A[mask];	//handle base
case separately (leaf states)
	for(int i = 0;i < N; ++i){
		if(mask & (1<<i))
			dp[mask][i] = dp[mask][i-1] +
dp[mask^(1<<i)][i-1];
		else
			dp[mask][i] = dp[mask][i-1];
	}
	F[mask] = dp[mask][N-1];
```

```
}

//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
        F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask <
(1<<N); ++mask){
        if(mask & (1<<i))
                F[mask] += F[mask^(1<<i)];
}
//--------order n*(2^n);

//////prbv old doc
GRAPHS
*No Eulerian Path &lt;=&gt; No of Odd Degree
Vetrices
&gt;=2
*Eulerian Circuit No odd degree vertex
*Maximal Independent Set = Total_Nodes -
Minimal_Vertex_Cover
*Directed Graph is Eulerian(having E.Cycle) iff
in_degree=out_degree(of all vertices) and all
vertices with non 0 degree belong to a single SCC
*Maximum Bipartite Matching = Minimal Vertex
Cover (Konigs Theorem)
MAXIMAL BIPARTITE MATCHING
#define MAXN 100010 //Maximum Nodes
#define MAXM 150000 //Maximum Edges
int n1, n2, edges, last[MAXN];
int previous[MAXM], head[MAXM];
int matching[MAXN], dist[MAXN], Q[MAXN];
bool used[MAXN], vis[MAXN];
void init(int a, int b) {
n1 = a;
n2 = b;
edges = 0;
fill(last, last + n1, -1);
}

//u-&gt;Node of left side, v-&gt;Right Side
void addEdge(int u, int v) {
head[edges] = v;
previous[edges] = last[u];
last[u] = edges++;
}
void bfs() {
fill(dist, dist + n1, -1);
int sizeQ = 0;
for (int u = 0; u &lt; n1; ++u) {
if (!used[u]) {
Q[sizeQ++] = u;
dist[u] = 0;
}
}
for (int i = 0; i &lt; sizeQ; i++) {
int u1 = Q[i];
for (int e = last[u1]; e &gt;= 0; e =
previous[e]) {
int u2 = matching[head[e]];
if (u2 &gt;= 0 &amp;&amp; dist[u2] &lt; 0) {
dist[u2] = dist[u1] + 1;
Q[sizeQ++] = u2;
}
}
}
}
bool dfs(int u1) {
vis[u1] = true;
for (int e = last[u1]; e &gt;= 0; e = previous[e]) {
int v = head[e];
int u2 = matching[v];
if (u2 &lt; 0 || !vis[u2] &amp;&amp; dist[u2] ==
dist[u1] + 1 &amp;&amp; dfs(u2)) {
matching[v] = u1;
used[u1] = true;
return true;
}
}
return false;
}
int maxMatching() {
fill(used, used + n1, false);
fill(matching, matching + n2, -1);
for (int res = 0;;) {
bfs();
fill(vis, vis + n1, false);

int f = 0;
for (int u = 0; u &lt; n1; ++u)
if (!used[u] &amp;&amp; dfs(u))
++f;
if (!f)
return res;
res += f;
}
}//FIRST init() then addEdges then, maxMatching
```

BRIDGES

```
void getBridges(int u){
low[u]=dt[u]=tim++;
visited[u]=1;
for(auto i=G[u].begin();i!=G[u].end();i++){
        int v=*i;
        if(!visited[v]){
        P[v]=u;
        getBridges(v);
        low[u]=min(low[v],low[u]);
        if(low[v]&amp;gt;dt[u]){
        if(u!=v)
        B.pb(pii(min(u,v),max(u,v))); //BRIDGE
}
}
        else{
        if(v!=P[u])
        low[u]=min(low[u],dt[v]);
}
}
}
```

Articulation Point:

(1) u is root of DFS tree and has two or more children. (2) If u is not root and low value of one of its child is more than discovery value of u.

```
if(parent[u] != NIL &amp;amp;&amp;amp; low[v] &amp;gt;= dt[u])
ap[u] = true;
```

DINIC'S MAXFLOW

```
const long long maxnodes = 5005;
long long capacity[maxnodes][maxnodes];
long long nodes = maxnodes, src, dest;
long long dist[maxnodes], q[maxnodes],
work[maxnodes];
struct Edge {
long long to, rev;
long long f, cap;
};

vector&amp;lt;Edge&amp;gt; g[maxnodes];
// Adds bidirectional edge
// if unidirectional is needed, make cap=0 in Edge
b
void addEdge(long long s, long long t, long long
cap){
Edge a = {t, g[t].size(), 0, cap};
Edge b = {s, g[s].size(), 0, cap};
g[s].push_back(a);
g[t].push_back(b);
}
bool dinic_bfs() {
fill(dist, dist + nodes, -1);
dist[src] = 0;
long long qt = 0;
q[qt++] = src;
for (long long qh = 0; qh &amp;lt; qt; qh++) {
long long u = q[qh];
for (long long j = 0; j &amp;lt; (long long) g[u].size();
j++) {
Edge &amp;amp;e = g[u][j];
long long v = e.to;
if (dist[v] &amp;lt; 0 &amp;amp;&amp;amp; e.f &amp;lt; e.cap) {
dist[v] = dist[u] + 1;
q[qt++] = v;
}
}
}
return dist[dest] &amp;gt;= 0;
}
long long dinic_dfs(long long u, long long f) {
if (u == dest)
return f;
for (long long &amp;amp;i = work[u]; i &amp;lt; (long long)
g[u].size(); i++) {
Edge &amp;amp;e = g[u][i];
if (e.cap &amp;lt;= e.f) continue;
long long v = e.to;
if (dist[v] == dist[u] + 1) {
long long df = dinic_dfs(v, min(f, e.cap - e.f));
if (df &amp;gt; 0) {
e.f += df;
g[v][e.rev].f -= df;
return df;
}
}
}

return 0;
}
long long maxFlow(long long _src, long
long_dest) {
src = _src;
dest = _dest;
long long result = 0;
while (dinic_bfs()) {
fill(work, work + nodes, 0);
while (long long delta = dinic_dfs(src,
```

```
INT_MAX))
result += delta;
}
return result;
}
int main()
{
cin&gt;&gt;nodes;
while(nodes!=0)
{
long long s,t,m;
//cin&gt;&gt;s&gt;&gt;t&gt;&gt;m;
cin&gt;&gt;m;
s=0;t=nodes-1;
//s-- ;t-- ;//if nodes are from 0 to n-1
for(long long i=0;i&lt;nodes;i++)
for(long long j=0;j&lt;nodes;j++)
capacity[i][j]=0;

while(m-- )
{
long long u,v,c;
cin&gt;&gt;u&gt;&gt;v&gt;&gt;c;
u-- ;v-- ; //if nodes are from 0 to n-1
capacity[u][v]+=c;
}
for(long long i=0;i&lt;nodes;i++) g[i].clear();
for (long long i = 0; i &lt; nodes; i++)
for (long long j = 0; j &lt; nodes; j++)
if (capacity[i][j] != 0)
addEdge(i, j, capacity[i][j]);
cout&lt;&lt; maxFlow(s, t) &lt;&lt; endl;
//cin&gt;&gt;nodes;
nodes=0;

}
}
```

8 1 8 15 1 2 10 1 3 5 1 4 15 2 5 9 2 6 15	7 3 6 3 6 8 5 6 5 6 7 15 5 8 10 6 8 10 7 8 10

2 3 4 3 4 4 4 7 16	

==>30 undirectional, 28 directional

STRONGLY CONNECTED COMPONENT

```
#define MAX 100001
#define pb push_back
using namespace std;
vector&lt;int&gt; V[MAX];
vector&lt;int&gt; InV[MAX];
int visited[MAX]={0};
stack&lt;int&gt; S;
void dfs(int u){
        visited[u]=1;
        for(unsigned i=0;i&lt;V[u].size();i++){
        int v=V[u][i];
        if(!visited[v]){
        dfs(v);
        }
        }
        S.push(u);
        }
        int P[MAX];
        int Rank[MAX]={0};
        int find(int i){
        if(P[i]==i)
        return i;
        else{
        P[i]=find(P[i]);
        return P[i];
        }
}
void Union(int a,int b){
        int u=find(a),v=find(b);
        if(u!=v){
        if(Rank[u]&lt;Rank[v]){

        P[u]=P[v];
        }
        else{
        P[v]=P[u];
        if(Rank[u]==Rank[v])
        Rank[u]++;
```

```
        }
        }
}
void dfs2(int u,int &amp;m){
        visited[u]=0;
        m=min(m,u);
for(unsigned i=0;i&lt;InV[u].size();i++){
        if(visited[InV[u][i]]){
        Union(u,InV[u][i]);
        dfs2(InV[u][i],m);
        m=min(m,InV[u][i]);
        }
        }
}
int main(){
        int n,m;
        cin&gt;&gt;n&gt;&gt;m;
        for(int i=0;i&lt;m;i++){
        int a,b;
        cin&gt;&gt;a&gt;&gt;b;
        V[a].pb(b);
        InV[b].pb(a);
        }
        for(int i=0;i&lt;n;i++){
        P[i]=i;
        if(!visited[i])
        dfs(i);

        }
        int M[MAX]={0};
        while(!S.empty()){
        int v=S.top();
        S.pop();
        int m;
        if(visited[v]){
        m=v;
        dfs2(v,m);
        M[find(v)]=m;
        }
        }
        //Prints the minimum valued vertex in the
        SCC of
        i
        for(int i=0;i&lt;n;i++)

        cout&lt;&lt;M[find(i)]&lt;&lt;&#39;\n&#39;;

}
```

DIJKSTRA's Shortest Path

```
typedef pair<int, int> pii;
typedef vector<vector<pii> > Graph;
//prio has the distance
void dijkstra(Graph &g, int s, vector<int> &prio,
vector<int> &pred) {
   int n = g.size();
   prio.assign(n, INT_MAX);
   prio[s] = 0;
   pred.assign(n, -1);
   priority_queue<pii, vector<pii> , greater<pii> >
q;
   q.push(make_pair(0, s));

   while (!q.empty()) {
      int d = q.top().first;
      int u = q.top().second;
      q.pop();
      if (d != prio[u])
         continue;
      for (int i = 0; i < (int) g[u].size(); i++) {
         int v = g[u][i].first;
         int nprio = prio[u] + g[u][i].second;
         if (prio[v] > nprio) {
            prio[v] = nprio;
            pred[v] = u;
            q.push(make_pair(nprio, v));
         }
      }
   }
}
//~ Graph g(n+1);
//~ addEdge(a,b,cost) =>
g[a].push_back(pii(b,cost))
```

EULER TOTIENT

```
int phi(int n) {
   int result = n;
   for(int i = 2; i * i <= n; ++i)
      if(n % i == 0) {
         while(n % i == 0)
            n /= i;
         result -= result / i;
      }
   if(n > 1)
```

```
        result -= result / n;
    return result;
}
```

long long factMOD(int n, int MOD)

```
{
    long long res = 1;
    while (n > 0)
    {
        for (int i=2, m=n%MOD; i<=m; i++)
            res = (res * i) % MOD;
        if ((n/=MOD)%2 > 0)
            res = MOD - res;
    }
    return res;
}
```

// --------- **pollard rho brent factorization** ---------

```
def brent(N):
    if N%2==0:
        return 2
    y,c,m = random.randint(1, N-1),random.randint(1,
N-1),random.randint(1, N-1)
    g,r,q = 1,1,1
    while g==1:
        x = y
        for i in range(r):
            y = ((y*y)%N+c)%N
        k = 0
        while (k<r and g==1):
            ys = y
            for i in range(min(m,r-k)):
                y = ((y*y)%N+c)%N
                q = q*(abs(x-y))%N
            g = gcd(q,N)
            k = k + m
        r = r*2
    if g==N:
        while True:
            ys = ((ys*ys)%N+c)%N
            g = gcd(abs(x-ys),N)
            if g>1:
                break
    return g
```

// ----------- **miller rabin primality test** --------

```
def modulo(a,b,c):
    x = 1
    y = a
    while b>0:
        if b%2==1:
            x = (x*y)%c
        y = (y*y)%c
        b = b/2
    return x%c
```

def millerRabin(N,iteration):

```
    if N<2:
        return False
    if N!=2 and N%2==0:
        return False

    d=N-1
    while d%2==0:
        d = d/2

    for i in range(iteration):
        a = random.randint(1, N-1)
        temp = d
        x = modulo(a,temp,N)
        while (temp!=N-1 and x!=1 and x!=N-1):
            x = (x*x)%N
            temp = temp*2

        if (x!=N-1 and temp%2==0):
            return False

    return True
```

// --------- **Z algo for string matching in linear time**

```
bool zAlgorithm(string pattern, string target)
{
    string s = pattern + '$' + target ;
    int n = s.length();
    vector<int> z(n,0);
```

```
   int goal = pattern.length();
   int r = 0, l = 0, i;
   for (int k = 1; k<n; k++)
   {
      if (k>r)
      {
         for (i = k; i<n && s[i]==s[i-k]; i++);
         if (i>k)
         {
            z[k] = i - k;
            l = k;
            r = i - 1;
         }
      }
      else
      {
         int kt = k - l, b = r - k + 1;
         if (z[kt]>b)
         {
            for (i = r + 1; i<n && s[i]==s[i-k]; i++);
            z[k] = i - k;
            l = k;
            r = i - 1;
         }
      }
      if (z[k]==goal)
         return true;
   }
   return false;
}
```

// **Sieve of Eratosthenes with linear time work**

```
const int N = 10000000;
int lp[N+1];
vector<int> pr;

for (int i=2; i<=N; ++i) {
        if (lp[i] == 0) {
                lp[i] = i;
                pr.push_back (i);
        }
        for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] &&
i*pr[j]<=N; ++j)
                lp[i * pr[j]] = pr[j];
}
```

MATRICES

```
typedef vector<int> vi;
typedef vector<vi> vvi;
const int mod = 100000007;
vvi matrixUnit(int n) {
   vvi res(n, vi(n));
   for (int i = 0; i < n; i++)
      res[i][i] = 1;
   return res;
}
```

MATRIX MULTIPLICATION

```
vvi matrixMul(const vvi &a, const vvi &b) {
   int n = a.size();
   int m = a[0].size();
   int k = b[0].size();
   vvi res(n, vi(k));
   for (int i = 0; i < n; i++)
 for (int j = 0; j < k; j++)
  for (int p = 0; p < m; p++)
   res[i][j]=(res[i][j]+(long long) a[i][p] * b[p][j]) % mod;
   return res;
}
vvi matrixPow(const vvi &a, int p) {
   if (p == 0)
      return matrixUnit(a.size());
   if (p & 1)
      return matrixMul(a, matrixPow(a, p - 1));
   return matrixPow(matrixMul(a, a), p / 2);
}
```

KMP

```
* Terminologies Used
* T   ==> Input string(the long one)
* P   ==> Pattern to be matched
* P^k ==> P[1....k]
* =]  ==> Suffix (A =] B means A is a suffix of B. eg: cd =]
abcd)
* pi[q] = max{ k : k<q && P^k =] P^q }
*/

vector <char> T;
vector <char> P;
```

```
vector <long long> pi;
void compute_pi(long long m)
{
        //P[1.....m] is the Pattern to be matched
        //pi[1....m] will be the new array
        pi.clear();
        pi.push_back(0); //pi[0]=0;
        pi.push_back(0); //pi[1]=0;
        long long k=0;
        for(long long q=2;q<=m;q++)
        {
                while((k>0) && (P[k+1]!=P[q]))
                    k=pi[k];
                if(P[k+1]==P[q]) k++;
                pi.push_back(k);
        }
}
void match(long long m)  // Add a newline at the end
of T to determine termination
{
        //T[0....(n-1)] is the input string
        //P[1....m] is the pattern to be matched
        //Compute_pi() should be called for P
        int q=0;
        char x;
        int i=0;
        x=T[i];
        while(x!='\n')
        {
           i++;
            while((q>0) && (P[q+1]!=x))
                q=pi[q];
                if(P[q+1]==x) q++;
                if(q==m) {
cout /*<<"Match at "*/<<i-m<<endl;q=pi[q];}
                x=T[i];
                //cout<<x;
        }
}
int main()
{
        long long m;
        while(cin>>m)
        {
                P.clear();
                char x;
                scanf("%c",&x);
                P.push_back('a'); //Pushing junk
                                        at P[0]
                for(long long i=0;i<m;i++)
                {
                        scanf("%c",&x);
                        //cout<<x;
                        P.push_back(x);
                }
                scanf("%c",&x);
                scanf("%c",&x);
                T.clear();
                while(x!='\n')
                {
                        T.push_back(x);
                        scanf("%c",&x);
                }
                T.push_back(x);
                compute_pi(m);
                match(m);
        }
}
```

KRUSKAL's MST

```
#define pii pair<int,int>
vector<pii> E;
vector<pii> Edges;
vector<int> P;
vector<int> R;
int find(int i){
 if(P[i]==i)
  return i;
 else
  return (P[i]=find(P[i]));
}
int main(){
 cout.sync_with_stdio(0);
 cin.tie(0);
        int n,m;
        cin>>n>>m;
        for(int i=0;i<m;i++){
                int a,b,c;
  cin>>a>>b>>c;
```

```
  E.push_back(pii(c,i));
  Edges.push_back(pii(a,b));
 }
 sort(E.begin(),E.end());
 R.assign(n+1,0);
 for(int i=0;i<=n;i++)
  P.push_back(i);
 int e=0;
 long long cost=0;
 for(int i=0;e<n-1;i++){
  int a=E[i].second;
  int u=Edges[a].first;
  int v=Edges[a].second;
  if(find(u)!=find(v)){
   e++;
   u=P[u],v=P[v];
   cost+=E[i].first;
   if(R[u]<R[v])
    P[u]=P[v];
   else{
    P[v]=P[u];
    if(R[v]==R[u])
     R[u]++;
   }
  }
 }
 cout<<cost<<'\n';
}
```

FLOYD WARSHALL

```
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
        mat[i][j]=INT_MAX;
while(m--)
{
        int a,b,x;
        cin>>a>>b>>x;
        mat[a][b]=x;
}
for(i=1;i<=n;i++)
    mat[i][i]=0;

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

//mat[a][b] => distance between A & B
```

Longest Increasing Subsequence

```
int main()
{
  int n, num;
  vector<int> v;
  cin >> n;
  while (n--){
  cin >> num;
  if (v.size() == 0 || num > v.back())
        v.push_back(num);
        else *lower_bound(v.begin(), v.end(), num) =
num;
        //for(unsigned int i=0;i<v.size();i++)
        //cout<<v[i]<<' ';cout<<endl;
        }
        cout << v.size();
}
```

KD TREES

```
 typedef pair<int, int> pii;
 typedef vector<pii> vpii;

 const int maxn = 100000;
 int tx[maxn];
 int ty[maxn];
 bool divX[maxn];

 bool cmpX(const pii &a, const pii &b) {
   return a.first < b.first;
 }

 bool cmpY(const pii &a, const pii &b) {
   return a.second < b.second;
 }

 void buildTree(int left, int right, pii
 points[]) {
   if (left >= right)
     return;
   int mid = (left + right) >> 1;
```

```
   //sort(points + left, points + right + 1,
divX ? cmpX : cmpY);
   int minx = INT_MAX;
   int maxx = INT_MIN;
   int miny = INT_MAX;
   int maxy = INT_MIN;
   for (int i = left; i < right; i++) {
      checkmin(minx, points[i].first);
      checkmax(maxx, points[i].first);
      checkmin(miny, points[i].second);
      checkmax(maxy, points[i].second);
   }
   divX[mid] = (maxx - minx) >= (maxy -
miny);
   nth_element(points + left, points +
mid, points + right, divX[mid] ? cmpX :
cmpY);

   tx[mid] = points[mid].first;
   ty[mid] = points[mid].second;

   if (left + 1 == right)
      return;
   buildTree(left, mid, points);
   buildTree(mid + 1, right, points);
}

long long closestDist;
int closestNode;

void findNearestNeighbour(int left, int
right, int x, int y) {
   if (left >= right)
      return;
   int mid = (left + right) >> 1;
   int dx = x - tx[mid];
   int dy = y - ty[mid];
   long long d = dx * (long long) dx + dy
* (long long) dy;
   if (closestDist > d && d) {
      closestDist = d;
      closestNode = mid;
   }
   if (left + 1 == right)
      return;

   int delta = divX[mid] ? dx : dy;
   long long delta2 = delta * (long long)
delta;
   int l1 = left;
   int r1 = mid;
   int l2 = mid + 1;
   int r2 = right;
   if (delta > 0)
      swap(l1, l2), swap(r1, r2);

   findNearestNeighbour(l1, r1, x, y);
   if (delta2 < closestDist)
      findNearestNeighbour(l2, r2, x, y);
}

int findNearestNeighbour(int n, int x,
int y) {
   closestDist = LLONG_MAX;
   findNearestNeighbour(0, n, x, y);
   return closestNode;
}

int main() {
   vpii p;
   p.push_back(make_pair(0, 2));
   p.push_back(make_pair(0, 3));
   p.push_back(make_pair(-1, 0));

   p.resize(unique(p.begin(), p.end()) -
p.begin());

   int n = p.size();
   buildTree(1, 0, n - 1, &(vpii(p)[0]));
   int res = findNearestNeighbour(n, 0,
0);

   cout << p[res].first << " " <<
p[res].second << endl;
```

```
    return 0;
}
```

PRIMS

```
typedef pair<int, int> pii;
typedef vector<vector<pii> > Graph;

long long prim(Graph &g, vector<int> &
pred) {
    int n = g.size();
    pred.assign(n, -1);
    vector<bool> vis(n);
    vector<int> prio(n, INT_MAX);
    prio[0] = 0;
    priority_queue<pii, vector<pii> , great
er<pii> > q;
    q.push(make_pair(0, 0));
    long long res = 0;

    while (!q.empty()) {
        int d = q.top().first;
        int u = q.top().second;
        q.pop();
        if (vis[u])
            continue;
        vis[u] = true;
        res += d;
        for (int i = 0; i < (int) g[u].size(); i++)
 {
            int v = g[u][i].first;
            if (vis[v])
                continue;
            int nprio = g[u][i].second;
            if (prio[v] > nprio) {
                prio[v] = nprio;
                pred[v] = u;
                q.push(make_pair(nprio, v));
            }
        }
    }
    return res;
}
```

CONVEX HULL

```
typedef pair<double, double> point;
bool cw(const point &a, const point &b,
 const point &c) {
    return (b.first - a.first) * (c.second - a.
second) - (b.second - a.second) * (c.first
 - a.first) < 0;
}
vector<point> convexHull(vector<point
> p) {
    int n = p.size();
    if (n <= 1)
        return p;
    int k = 0;
    sort(p.begin(), p.end());
    vector<point> q(n * 2);
    for (int i = 0; i < n; q[k++] = p[i++])
        for (; k >= 2 && !cw(q[k - 2], q[k - 1]
, p[i]); --k)
            ;
    for (int i = n - 2, t = k; i >= 0; q[k++] = p
[i--])
        for (; k > t && !cw(q[k - 2], q[k - 1],
p[i]); --k)
            ;
    q.resize(k - 1 - (q[0] == q[1]));
    return q;
}

int main() {
    vector<point> points(4);
    points[0] = point(0, 0);
    points[1] = point(3, 0);
    points[2] = point(0, 3);
    points[3] = point(1, 1);
    vector<point> hull = convexHull(point
s);
    cout << (3 == hull.size()) << endl;
}
```