euclid

```
// ax+by=gcd
int gcd(int a,int b,int&x,int&y){
  x=1;y=0;
  int x1=0,y1=1,a1=a,b1=b;
  while(b1){
    int q=a1/b1;
    tie(x,x1)=make_tuple(x1,x-q*x1);
    tie(y,y1)=make_tuple(y1,y-q*y1);
    tie(a1,b1)=make_tuple(b1,a1-q*b1);
  }
  return a1;
}
```

maxflow

```
struct MaxFlow{
  int n;
  vector<vector<int>>adj;
  vector<vector<ll>>cp;

  MaxFlow(int n):n(n){
    adj=vector<vector<int>>(n);
    cp=vector(n,vector(n,0LL));
  }

  // connect a and b with capacity c
  void add(int a,int b,int c){
    if(cp[a][b]==0){
      adj[a].push_back(b);
      adj[b].push_back(a);
    }
    cp[a][b]+=c;
  }

  ll bfs(int s,int t,vector<int>&p){
    fill(p.begin(),p.end(),-1);
    p[s]=-2;
    queue<pair<int,int>>q;
    q.push({s,1e9});
    while(!q.empty()){
      int c=q.front().first;
      ll f=q.front().second;
      q.pop();
      for(int x:adj[c]){
        if(p[x]==-1&&cp[c][x]){
          p[x]=c;
          ll r=min(f,cp[c][x]);
          if(x==t)
            return r;
          q.push({x,r});
        }
      }
    }

    return 0;
  }

  // call only once
  ll maxflow(int s,int t){
    ll f=0,r;
    vector<int>pr(n);
    while((r=bfs(s,t,pr))){
      f+=r;
      int cu=t;
      while(cu!=s){
        int p=pr[cu];
        cp[p][cu]-=r;
        cp[cu][p]+=r;
        cu=p;
      }
    }
    return f;
  }
};
```

modulo_int

```
const int MOD=1e9+7,PC=1e7;
ll M_inv[PC+1];
struct M{
  ll val=0;
  static void precomp(){M_inv[1]=1;for(ll i=2;i<=PC;++i){
    M_inv[i]=(MOD-MOD/i)*M_inv[MOD%i]%MOD;if(M_inv[i]<0)M_inv[i]+=MOD;}}
  static ll calc_inv(ll i){if(i<=PC)return M_inv[i];
    return(MOD-MOD/i)*calc_inv(MOD%i)%MOD;}
  M(ll x){val=(x%MOD+MOD)%MOD;}
  M()=default;
  template<class T>bool operator==(T b){return val==M(b).val;}
  template<class T>bool operator!=(T b){return val!=M(b).val;}
  template<class T>M operator+(T bt){M b(bt);M r(val);r.val+=b.val;if(r.val>=MOD)
    r.val-=MOD;return r;}
  template<class T>M operator-(T bt){M b(bt);M r(val);r.val+=MOD-b.val;if(r.val>=MOD)
    r.val-=MOD;return r;}
  template<class T>M operator*(T bt){M b(bt);M r(val);r.val*=b.val;r.val%=MOD;return r;}
  template<class T>M&operator+=(T bt){M b(bt);*this=*this+b;return*this;}
  template<class T>M&operator-=(T bt){M b(bt);*this=*this-b;return*this;}
  template<class T>M&operator*=(T bt){M b(bt);*this=*this*b;return*this;}
```

```
  M pow(ll e){M r(1);M a(val);while(e){if(e%2)r=r*a;a=a*a;e/=2;}return r;}
  template<class T>M operator/(T bt){ M b(bt); return M(val*calc_inv(b.val)); }
  template<class T>M&operator/=(T bt){M b(bt);*this*=calc_inv(b.val);return*this;}
};
namespace std{template<>struct hash<M>{
  inline size_t operator()(const M&x)const{return x.val;}
};}
template<class T>bool operator<(T at,T bt){M b(bt);M a(at);return a.val<b.val;}
ostream&operator<<(ostream&s,M m){s<<m.val;return s;}
istream&operator>>(istream&s,M&m){s>>m.val;return s;}
```

segtree

```
template<typename T>
struct SegTree{
  int n;
  vector<T>t;
  T (*m)(T,T);

  // n = size, d = default value, m = merge function
  SegTree(int n,T d,T(*m)(T,T)):n(n),m(m){
    t=vector<T>(2*n,d);
    for(int i=n-1;i>0;i--)
      t[i]=m(t[2 * i],t[2 * i + 1]);
  }

  // point update
  void upd(int i,T x){
    i+=n;
    t[i]=x;
    while(i>1){
      i/=2;
      t[i]=m(t[2*i],t[2*i+1]);
    }
  }

  // range query
  T get(int l,int r){
    l+=n;
    r+=n;
    T rs=t[l];
    l++;
    while(l<r){
      if(l%2){
        rs=m(rs,t[l]);
        l++;
      }
      if(r%2){
        r--;
        rs=m(rs,t[r]);
      }
      l/=2;
      r/=2;
    }
    return rs;
  }
};
```

suffix_array

```
vector<int> sort_cyclic_shifts(string const& s) {
  int n = s.size();
  const int alphabet = 256;
  vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
  for (int i = 0; i < n; i++)
    cnt[s[i]]++;
  for (int i = 1; i < alphabet; i++)
    cnt[i] += cnt[i-1];
  for (int i = 0; i < n; i++)
    p[--cnt[s[i]]] = i;
  c[p[0]] = 0;
  int classes = 1;
  for (int i = 1; i < n; i++) {
    if (s[p[i]] != s[p[i-1]])
      classes++;
    c[p[i]] = classes - 1;
  }
  vector<int> pn(n), cn(n);
  for (int h = 0; (1 << h) < n; ++h) {
    for (int i = 0; i < n; i++) {
      pn[i] = p[i] - (1 << h);
      if (pn[i] < 0)
        pn[i] += n;
    }
    fill(cnt.begin(), cnt.begin() + classes, 0);
    for (int i = 0; i < n; i++)
      cnt[c[pn[i]]]++;
    for (int i = 1; i < classes; i++)
      cnt[i] += cnt[i-1];
    for (int i = n-1; i >= 0; i--)
      p[--cnt[c[pn[i]]]] = pn[i];
    cn[p[0]] = 0;
    classes = 1;
    for (int i = 1; i < n; i++) {
      pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
      pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
      if (cur != prev)
        ++classes;
      cn[p[i]] = classes - 1;
    }
```

```
    c.swap(cn);
  }
  return p;
}

vector<int> suffix_array_construction(string s) {
  s += "$";
  vector<int> sorted_shifts = sort_cyclic_shifts(s);
  sorted_shifts.erase(sorted_shifts.begin());
  return sorted_shifts;
}

int compare(int i, int j, int l, int k) {
  pair<int, int> a = {c[k][i], c[k][(i+l-(1 << k))%n]};
  pair<int, int> b = {c[k][j], c[k][(j+l-(1 << k))%n]};
  return a == b ? 0 : a < b ? -1 : 1;
}

int lcp(int i, int j) {
  int ans = 0;
  for (int k = log_n; k >= 0; k--) {
    if (c[k][i % n] == c[k][j % n]) {
      ans += 1 << k;
      i += 1 << k;
      j += 1 << k;
    }
  }
  return ans;
}

vector<int> lcp_construction(string const& s, vector<int> const& p) {
  int n = s.size();
  vector<int> rank(n, 0);
  for (int i = 0; i < n; i++)
    rank[p[i]] = i;

  int k = 0;
  vector<int> lcp(n-1, 0);
  for (int i = 0; i < n; i++) {
    if (rank[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = p[rank[i] + 1];
    while (i + k < n && j + k < n && s[i+k] == s[j+k])
      k++;
    lcp[rank[i]] = k;
    if (k)
      k--;
  }
  return lcp;
}
```

z_algorithm

```
//  z[i] pove, da se z[i] znakovl zacensi z i-tim ujemajo s prvimi z[i] znaki stringa s.
vector<int> z_function(string s) {
  int n = s.size();
  vector<int> z(n);
  int l = 0, r = 0;
  for(int i = 1; i < n; i++) {
    if(i < r) {
      z[i] = min(r - i, z[i - l]);
    }
    while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
      z[i]++;
    }
    if(i + z[i] > r) {
      l = i;
      r = i + z[i];
    }
  }
  return z;
}
```

tarjan

```
int N, M, timer, tin[maxN], low[maxN];
bool vis[maxN];
vector<int> G[maxN];
vector<pii> bridges;

void dfs(int u = 1, int p = 0){
    vis[u] = true;
    tin[u] = low[u] = ++timer;
    for(int v : G[u]){
        if(v != p){
            if(vis[v])  low[u] = min(low[u], tin[v]);
            else {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] > tin[u])
                    bridges.push_back({u, v});
            }
        }
    }
}
```

tree_hash

```
const ll MOD=100000004987;

ll primes[]={
    1000000007,
    1000000207,
    1000000409,
    1000000531,
    1000000829,
    1000001021,
    1000001269,
    1000001447,
};

int n;
vector<int>nodes[100000];
int sub_size[100000];

void init_tree(int node,int par){
    sub_size[node]=1;
    for(int ne:nodes[node]){
        if(ne!=par){
            init_tree(ne,node);
            sub_size[node]+=sub_size[ne];
        }
    }
}

ll hash_tree(int node,int par,int depth){
    ll prime=primes[(depth+sub_size[node])%8];

    if(nodes[node].size()==1&&node!=par){
        return prime;
    }

    vector<ll>hashes;
    for(int ne:nodes[node]){
        if(ne==par)
            continue;
        hashes.push_back(hash_tree(ne,node,depth+1));
    }
    sort(hashes.begin(),hashes.end());
    ll res=0;
    ll cp=prime;
    for(ll i:hashes){
        res+=cp*i;
        res%=MOD;
        cp*=prime;
        cp%=MOD;
    }

    /*if(par==0||node==0)
        cout<<res<<"\n";*/

    return res;
}

pair<int,int> get_centroid(int node,int par){
    int biggest=-1;
    for(int ne:nodes[node]){
        if(ne==par)
            continue;
        if(biggest==-1||sub_size[ne]>sub_size[biggest])
            biggest=ne;
    }

    if(sub_size[biggest]<=sub_size[0]/2)
        return {node,biggest};
    return get_centroid(biggest,node);
}

// USAGE
cin>>n;
for(int i=0;i<n;i++)
  nodes[i].clear();
for(int i=0;i<n-1;i++){
  int a,b;
  cin>>a>>b;
  a--;b--;
  nodes[a].push_back(b);
  nodes[b].push_back(a);
}
init_tree(0,0);
pair<int,int> root=get_centroid(0,0);
init_tree(root.first,root.first);
ll hash11=hash_tree(root.first,root.first,0);
init_tree(root.second,root.second);
ll hash12=hash_tree(root.second,root.second,0);

for(int i=0;i<n;i++)
  nodes[i].clear();
for(int i=0;i<n-1;i++){
  int a,b;
  cin>>a>>b;
  a--;b--;
  nodes[a].push_back(b);
  nodes[b].push_back(a);
}
init_tree(0,0);
root=get_centroid(0,0);
init_tree(root.first,root.first);
ll hash21=hash_tree(root.first,root.first,0);
init_tree(root.second,root.second);
ll hash22=hash_tree(root.second,root.second,0);

cout<<((hash11==hash21||hash12==hash22||hash11==hash22||hash12==hash21)?"YES\n":"NO\n");
```

hash_string

```
const int HMOD=1e9+7;
const int HX=31;

class FastString{
    string str;
    vector<ll>hashes;
    vector<ll>pows;
    int len;
public:
    FastString(string str):str(str){
        len=(int)str.size();
        hashes.resize(len+1);
        pows.resize(len+1);
        pows[0]=1;
        for(int i=1;i<=len;i++)
            pows[i]=(pows[i-1]*HX)%HMOD;

        hashes[0]=0;
        for(int i=1;i<=len;i++)
            hashes[i]=(hashes[i-1]+pows[i-1]*(str[i-1]-'a'+1))%HMOD;
    }

    ll get(int l,int r){
        ll hsh=hashes[r]+HMOD-hashes[l];
        hsh%=HMOD;
        hsh*=pows[len-l];
        hsh%=HMOD;
        return hsh;
    }
};
```

convex_hull

```
struct pt {
    double x, y;
    bool operator == (pt const& t) const {
        return x == t.x && y == t.y;
    }
};

int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<pt>& a, bool include_collinear = false) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    if (include_collinear == false && st.size() == 2 && st[0] == st[1])
        st.pop_back();

    a = st;
}
```

setup

```
/*

setxkbmap us -option caps:escape

~/.vimrc

set number
set relativenumber
syntax enable
set autoindent
set tabstop=2
set shiftwidth=2
```

```
set wrap
set scrolloff=20
filetype plugin indent on
autocmd FileType cpp setlocal cindent

Makefile

CXX = g++
CXXFLAGS = -std=c++20 -O2 -Wall -DLOC

%: %.cpp
  $(CXX) $(CXXFLAGS) $< -o $@

 */
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
#define fwd(i,a,b) for(int i=(a); i<(b);i++)
#define rep(i,n) fwd(i,0,n)
#define all(X) (X).begin(), (X).end()
#define sz(X)((int)(X).size())
#define st first
#define nd second
#define pb push_back
typedef vector<int> vi;
typedef pair<int,int> pii;

#ifdef LOC
template<typename T1,typename T2>
auto& operator<<(ostream& out, pair<T1, T2> a) {return out<<"("<<a.st<<", "<<a.nd<<")";}
template<typename T>
auto& operator<<(ostream& out, vector<T>& a) {
    out<<"{";
    int cnt=0;
    for(auto b:a){
      cnt++;
      out<<b;
      if(cnt!=a.size())
        out<<", ";
    }
    return out<<"}";
}

#define debug(x) cerr<<"[" #x "]: "<<x<<"\n";
#else
#define debug(x) ;
#endif

void solve(){
}

int main(){
  cin.tie(0);cout.tie(0);ios::sync_with_stdio(false);
  int t=1;cin>>t;
  while(t--)solve();
  return 0;
}
```

min_cost_flow

```
struct Edge
{
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
```

```
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```