Contents

Configuration

Template

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

#define FILL(X, V) memset((X), (V), sizeof(X))
#define SIZE(V) int((V).size())
#define FOR2(c,i,j) for(int(c)=(i),_MAX=(j);(c)<_MAX;(c)++)
#define FOR(cont,max) FOR2((cont), 0, (max))
#define LOG(x) (31 - __builtin_clz(x))
#define W(x) cerr << "\033[31m" << #x << "=" << x << "\033[0m" << "\n";
#define ii      pair<int, int>
#define ff      first
#define ss      second
#define oo      1e9
#define ep      1e-9
#define pb      push_back

typedef long long ll
typedef unsigned long long ul

int main() {
   ios::sync_with_stdio(false);
}
```

Vim

```
set number
set showmatch
set autoindent
set cindent
set shiftwidth=4
set smartindent
set smarttab
set softtabstop=4
set backspace=indent,eol,start
set visualbell
set hlsearch
set incsearch
set ruler
set undolevels=1000
syntax on
```

Ad-Hoc

Debrujin

```
string seq;

int pw(int b,int a){
   int ans = 1;
   while( a ){
      if(a&1) ans *= b;
      b *= b;
      a /= 2;
   }
   return ans;
}

void debruijn( int n, int k ){
   seq = "";
   char s[n];
   if( n == 1 ){
      for( int i = 0; i < k; i++ )
         seq += char('0'+i);
   } else {
      for( int i = 0; i < n-1; i++ )
         s[i] = k-1;

      int kn = pw(k,n-1);
      char nxt[kn]; memset(nxt,0,sizeof(nxt));
      kn *= k;
      for( int h = 0; h < kn; h++ ){
         int m = 0;
         for( int i = 0; i < n-1; i++ ){
            m *= k;
            m += s[(h+i)%(n-1)];
         }
         seq += char('0'+nxt[m]);
         s[h%(n-1)] = nxt[m];
         nxt[m]++;
      }
   }
}
```

Josephus Problem

```
int f(int n, int k){ // Quantidade de pessoas e o tamanho do salto
   return (n == 1) 1 : (f(n-1, k) + k - 1) % n + 1;
}
```

LIS

```
vector<int> lis(vector<int>& seq) {
   int smallest_end[seq.size()+1], prev[seq.size()];
   smallest_end[1] = 0;

   int sz = 1;
   for(int i = 1; i < seq.size(); ++i) {
      int lo = 0, hi = sz;
      while(lo < hi) {
         int mid = (lo + hi + 1)/2;
         if(seq[smallest_end[mid]] <= seq[i])
            lo = mid;
         else
            hi = mid - 1;
      }

      prev[i] = smallest_end[lo];
      if(lo == sz)
         smallest_end[++sz] = i;
      else if(seq[i] < seq[smallest_end[lo+1]])
         smallest_end[lo+1] = i;
   }

   vector<int> ret;
   for(int cur = smallest_end[sz]; sz > 0; cur = prev[cur], --sz)
      ret.push_back(seq[cur]);
   reverse(ret.begin(), ret.end());

   return ret;
}
```

Subsets

```
for (int i=0; i < (1<<n); ++i) {
   for(int i2 = i; i2 > 0; i2 = (i2-1) & i) {
   }
}
```

Josephus Problem

```
for (int i=0; i < (1<<n); ++i) {
   for(int i2 = i; i2 > 0; i2 = (i2-1) & i) {
   }
}
```

Fib in Compile Time

```
template<ul N>
struct fibonacci : integral_constant<ul, (fibonacci<N-1>{} + fibonacci<N-2>{})> {};
template<> struct fibonacci<1> : integral_constant<ul,1> {};
template<> struct fibonacci<0> : integral_constant<ul,0> {};
#define F0(x) fib[x]=fibonacci<x>{}
```

Tips

```
next_permutation(myints,myints+3)
prev_permutation(myints,myints+3)

bool is_power_of_2(int n) { return (n <= 0)? 0 : !(n & (n - 1)); }

scanf("%x"); // le como hexadecimal
scanf("%e"); // le como notacao cientifica
```

Geometry

Graph

Bron Kerbosch

```
ll adj[70];
vector<ll> cliques;

void bron_kerbosh(ll r, ll p, ll e){
   if(!p && !e){
      cliques.push_back(r);
      return;
   }

   for(int i = 0; i < 64; i++){
      if(!(p & (1LL << i))) continue;
      bron_kerbosh(r | 1LL << i, p & adj[i], e & adj[i]);
      p ^= 1LL << i;
      e |= 1LL << i;
   }
}
```

Dinic Min Cost

```
#define wt second.second
#define nv second.first
#define cp first.second
#define vv first.first
#define OO 1000000000000000000
#define oo 1000000000
#define ff first
#define ss second
```

```
#define mp make_pair
#define pb push_back

long long level[1000005];
int v[1000005];
int vis[1000005];
vector<pair<pair<int,int>, pair<int,int> > > adj[1000005];
int maxflow;
long long mincost;
int ed;

void add_edge(int v, int u, int cap, int pes){
   adj[v].pb(mp(mp(u, cap), mp(adj[u].size(), pes)));
   adj[u].pb(mp(mp(v, 0), mp(adj[v].size()-1, -pes)));
}

int dij(int s, int t){
   priority_queue<pair<long long, int> > q;
   level[s] = 0;
   q.push(mp(0,s));

   while(!q.empty()){
      long long p = -q.top().ff;
      int v = q.top().ss;
      q.pop();

      for(int i = 0; i < adj[v].size(); i++){
         if(level[adj[v][i].vv] > level[v] + adj[v][i].wt && adj[v][i].cp > 0){
            level[adj[v][i].vv] = level[v] + adj[v][i].wt;
            q.push(mp(-level[adj[v][i].vv],adj[v][i].vv));
         }
      }
   }

   return level[t] == OO;
}

pair<int,long long> dfs(int x, int t, int flow){
   if(vis[x]) return mp(0,0);
   if(x == t) return mp(flow,0);
   vis[x] = 1;

   for(; v[x] < adj[x].size(); v[x]++){
      pair<pair<int,int>, pair<int,int> > at = adj[x][v[x]];

      if(level[at.vv] >= level[x] + at.wt && at.cp > 0){
         pair<int, long long> currflow;
         currflow.ff = min(flow,at.cp);
         currflow = dfs(at.vv, t, currflow.ff);
         currflow.ss += at.wt;
         if(currflow.ff){
            adj[x][v[x]].cp -= currflow.ff;
            adj[at.vv][at.nv].cp += currflow.ff;

            vis[x] = 0;
            return currflow;
         }
      }
   }
   vis[x] = 0;
   return mp(0,0);
}

void dinic(int s,int t){
   while(1){
      for(int i = 0; i <= ed; i++) level[i] = OO;
      if(dij(s,t)) break;

      memset(v, 0, sizeof v);

      pair<int,long long> temp;
      temp = dfs(s,t,oo);
      if(!temp.ff) break;
      maxflow += temp.ff;
      mincost += temp.ss;
   }
}
```

Dinic

```
#define nv second
#define cp first.second
#define vv first.first
#define nv second.first
#define cp first.second
#define vv first.first
#define oo 1000000000
#define ff first
#define ss second
#define mp make_pair
#define pb push_back

int level[1000005];
int v[1000005];
int maxflow;
vector<pair<pair<int,int>, int> > adj[100005];

void add_edge(int v, int u, int cap){
   adj[v].pb(mp(mp(u,cap),adj[u].size()));
   adj[u].pb(mp(mp(v,0),adj[v].size()-1));
}

int bfs(int s,int t){
```

```
   queue<int> q;
   int v;
   int i;
   level[s] = 0;
   q.push(s);

   while(!q.empty()){
      v = q.front();
      q.pop();

      for(i = 0; i < adj[v].size(); i++){
         if(level[adj[v][i].vv] == -1 && adj[v][i].cp > 0){
            q.push(adj[v][i].vv);
            level[adj[v][i].vv] = level[v] + 1;
         }
      }
   }

   return level[t] == -1;
}

int dfs(int x, int t, int flow){
   if(x == t) return flow;

   for(; v[x] < adj[x].size(); v[x]++){
      pair<pair<int,int>, int> at = adj[x][v[x]];

      if(level[at.vv] == level[x] + 1 && at.cp > 0){
         int currflow = min(flow,at.cp);
         currflow = dfs(at.vv, t, currflow);

         if(currflow){
            at.cp -= currflow;
            adj[at.vv][at.nv].cp += currflow;

            adj[x][v[x]] = at;
            return currflow;
         }
      }
   }

   return 0;
}

void dinic(int s,int t){
   while(1){
      memset(level, -1, sizeof level);
      if(bfs(s,t)) break;

      memset(v, 0, sizeof v);
      while(int temp = dfs(s,t,oo)) maxflow += temp;
   }
}
```

Edmonds Karp

```
int bfsek(int s,int t){
   queue<int> q;
   int v;
   int i, vis[305];
   memset(vis,0,sizeof(vis));
   q.push(s);

   while(!q.empty()){
      v = q.front();
      q.pop();
      vis[v] = 1;

      for(i = 0; i < adj[v].size(); i++){
         // Se a capacidade for igual a 0 a aresta nao existe
         if(!vis[adj[v][i]] && cap[v][adj[v][i]] > 0){
            flow[adj[v][i]] = min(cap[v][adj[v][i]],flow[v]);
            q.push(adj[v][i]);
            p[adj[v][i]] = v;
            if(adj[v][i] == t) return flow[t];
         }
      }
   }
   return 0;
}

int mflow(int s,int t){
   int mi, vai, sai, j;
   int maxflow = 0;

   while(1){
      for(j = 0; j <= 2*(n+1); j++) flow[j] = oo;

      mi = bfsek(s,t);
      if(mi == 0) break;

      maxflow += mi;
      sai = p[t];
      vai = t;

      // Subtrai o fluxo que passou das arestas utilizadas e soma nas arestas inversas
      while(vai != s){
         cap[sai][vai] -= mi;
         cap[vai][sai] += mi;
         vai = sai;
         sai = p[sai];
      }
   }
```

```
   return maxflow;
}
```

MaxClique

```
for(int i = 0; i < n; i++){
   for(int j = 0; j < n; j++){
     int x;
     scanf("%d",&x);
     if(x || i == j)
       adj[i] |= 1LL << j;
   }
 }

 int resto = n - n/2;
 int C = n/2;
 for(int i = 1; i < (1 << resto); i++){
   int x = i;
   for(int j = 0; j < resto; j++)
     if(i & (1 << j))
       x &= adj[j + C] >> C;
   if(x == i){
     dp[i] = __builtin_popcount(i);
   }
 }

 for(int i = 1; i < (1 << resto); i++)
   for(int j = 0; j < resto; j++)
     if(i & (1 << j))
       dp[i] = max(dp[i], dp[i ^ (1 << j)]);

 int maxCliq = 0;
 for(int i = 0; i < (1 << C); i++){
   int x = i, y = (1 << resto) - 1;
   for (int j = 0; j < C; j++)
     if (i & (1 << j))
       x &= adj[j] & ((1 << C) - 1), y &= adj[j] >> C;
   if (x != i) continue;
   maxCliq = max(maxCliq, __builtin_popcount(i) + dp[y]);
 }
```

Bip Match

```
/* Maximum Bipartite Matching (Minimum Vertex Cover) on unweighted graph */
#define MAXN 111

int N, M; // N - # of vertexes on X, M - # of vertexes on Y
vector< int > gr[MAXN]; // gr[u] -- edges from u in X to v in Y
bool seen[MAXN];
int m[MAXN], m1[MAXN]; // with whom it's matched

int dfs(int u) {
   if (u < 0) return 1;
   if (seen[u]) return 0;
   seen[u] = true;
   for (size_t i = 0, sz = gr[u].size(); i < sz; ++i) {
      if (dfs(m1[ gr[u][i] ])) {
         m[u] = gr[u][i];
         m1[ gr[u][i] ] = u;
         return 1;
      }
   }
   return 0;
}

int dfsExp(int u) {
   for (int i = 0; i < N; ++i) seen[i] = false;
   return dfs(u);
}


int bipMatch() {
   for (int i = 0; i < N; ++i) m[i] = -1;
   for (int i = 0; i < M; ++i) m1[i] = -1;
   int aug, ans = 0;
   do {
      aug = 0;
      bool first = true;
      for (int i = 0; i < N; ++i) if (m[i] < 0) {
         if (first) aug += dfsExp(i);
         else aug += dfs(i);
         first = false;
      }
      ans += aug;
   } while (aug);
   return ans;
}

/* needed for minium vertex cover.. */
int vx[MAXN], vy[MAXN];
void buildVC( int u ){
   seen[u] = true;
   vx[u] = 0;
   for (size_t w = 0, sz = gr[u].size(); w < sz; ++w)
      if (gr[u][w] != m[u] && vy[ gr[u][w] ] == 0) {
         vy[ gr[u][w] ] = 1;
         if (!seen[ m1[ gr[u][w] ] ]) buildVC(m1[ gr[u][w] ]);
      }
}

// T ~ Unmatched L + reachable using alternating paths
```

```
// ANS .. (L \ T) U ( R intersect T )
for (int i = 0; i < N; ++i) {
   seen[i] = false;
   if (m[i] == -1) vx[i] = 0;  // T -- unmatched L
   else vx[i] = 1;     // L \ T -- for now..
}

for (int i = 0; i < M; ++i) vy[i] = 0; // R .. ~T -- for now..
for (int i = 0; i < N; ++i) if (vx[i] == 0 && !seen[i]) buildVC(i);
```

Djikstra

```
vector<int> dist; // answer -> dist[destiny]
vector<vector<ii>> g; // u w v

void djikstra(int src = 1) {
   dist.resize(g.size(), oo);
   priority_queue<ii, vector<ii>, greater<ii>> pq;
   pq.pb({dist[src] = 0, src});
   while(not pq.empty()) {
      auto c = pq.top(); pq.pop();
      for(auto e : g[c.ss])
         if(dist[e.ss] > c.ff + e.ff)
            pq.pb({dist[e.ss] = c.ff + e.ff, e.ss});
   }
}
```

Floyd Warshall

```
/*
   init: p[i][j] = i;
   if (i,k)+(k,j) < (i,j)
    p[i][j] = p[k][j]
*/


void show( int from, int to ){
   if( from != to ){
      show( from, p[from][to] );
      cout << "␣";
   }
   cout << to;
}
```

Graphs Paths

```
// 1. Crie uma matriz de adjacencia com o numero e caminhos de um ponto a outro. Isso inicialmente para todos os elementos do grafo
// 2. A matriz resultante e elevada (por exponenciacao rapida) a K, snedo K o numero de arestas etc...
```

Kruskal

```
vector<iii> out;
void kruskal(){
   for(int i = 0; i < n_vertices; i++) make_set(i);
   sort(graph.bg(), graph.nd()); // ii(peso, ii(u, v))
   for(int i = 0; i < graph.size(); i++)
      if(find(graph[i].ss.ff) != find(graph[i].second.second))
         joint(graph[i].second.first, graph[i].second.second);
      else
         out.pb(graph[i]);
}
```

LCA

```
#include <bits/stdc++.h>

#define INF 0x3F3F3F3F
#define LINF 0x3F3F3F3FFFFFFFFFLL

#define FILL(X, V) memset( X, V, sizeof(X) )
#define TI(X) __typeof((X).begin())
#define ALL(V) V.begin(), V.end()
#define SIZE(V) int((V).size())

#define FOR(i, a, b) for(int i = a; i <= b; ++i)
#define RFOR(i, b, a) for(int i = b; i >= a; --i)
#define REP(i, N) for(int i = 0; i < N; ++i)
#define RREP(i, N) for(int i = N-1; i >= 0; --i)
#define FORIT(i, a) for( TI(a) i = a.begin(); i != a.end(); i++ )

#define PB push_back
#define MP make_pair

template<typename T> T inline SQR( const T &a ){ return a*a; }
template<typename T> T inline ABS( const T &a ){ return a < 0 ? -a : a; }
template<typename T> T inline MIN( const T& a, const T& b){ if( a < b ) return a; retur
template<typename T> T inline MAX( const T& a, const T& b){ if( a > b ) return a; retur

const double EPS = 1e-9;
inline int SGN( double a ){ return ((a > EPS) ? (1) : ((a < -EPS) ? (-1) : (0))); }
inline int CMP( double a, double b ){ return SGN(a - b); }

typedef long long int64;
typedef unsigned long long uint64;

using namespace std;

inline int next_int() {
   int n; char c; while((c = getchar_unlocked()) < '-');
```

```
   if( c == EOF ) exit(0);
   while (!('0' <= c && c <= '9')) {
      if( c == '-' ) neg = -1;
      c = getchar_unlocked();
      if( c == EOF ) exit(0);
   }
   while ('0' <= c && c <= '9') {
      n = n * 10 + c - '0';
      c = getchar_unlocked();
   }
   return neg*n;
}

int nxt_cmd(){
   char c = getchar_unlocked();
   while( c < 'A' || c > 'Z' ) c = getchar_unlocked();
   if( c == 'K' ){
      getchar_unlocked(); getchar_unlocked();
      return 1;
   }
   c = getchar_unlocked();
   if( c == 'O' ){
      getchar_unlocked(); getchar_unlocked();
      return -1;
   }
   getchar_unlocked(); getchar_unlocked();
   return 0;
}

struct edge_t{
   int v, c;
   edge_t( int vv = 0, int cc = 0 ) : v(vv), c(cc) {}
};

#define MAXN 10001
int N, parent[MAXN], L[MAXN], dis[MAXN];
int dp[15][MAXN];

vector< edge_t > gr[MAXN];

int lca( int u, int v ){
   if( L[u] < L[v] ){
      u ^= v; v ^= u; u ^= v;
   }
   int lg;
   for( lg = 1; (1<<lg) <= L[u]; lg++ );
   lg--;

   for( int i = lg; i >= 0; i-- )
      if( L[u] - (1<<i) >= L[v] )
         u = dp[i][u];
   if( u == v ) return u;
   for( int i = lg; i >= 0; i-- )
      if( dp[i][u] != -1 && dp[i][u] != dp[i][v] )
         u = dp[i][u], v = dp[i][v];
   return parent[u];
}

int kth( int k, int u ){
   while( k > 0 ){
      int lg = 0;
      while( (1<<lg) <= k ) lg++;
      lg--;
      u = dp[lg][u];
      k -= (1<<lg);
   }
   return u;
}

int main( int argc, char* argv[] ){

   int t, u, v, c, k, wut;

   t = next_int();
   while( t-- ){
      N = next_int();
      FOR( i, 1, N ){ parent[i] = -1; gr[i].clear(); }
      REP( i, N-1 ){
         u = next_int();
         v = next_int();
         c = next_int();
         gr[u].PB( edge_t(v, c) );
         gr[v].PB( edge_t(u, c) );
      }

      L[1] = 0; dis[1] = 0; parent[1] = 1;
      queue< int > q;
      q.push( 1 );
      while( !q.empty() ){
         u = q.front(); q.pop();
         REP( i, SIZE(gr[u]) ){
            v = gr[u][i].v;
            c = gr[u][i].c;
            if( parent[v] == -1 ){
               parent[v] = u;
               dis[v] = dis[u]+c;
               L[v] = L[u] + 1;
               q.push(v);
            }
         }
      }
      parent[1] = -1;

      for( int lg = 0; (1<<lg) < N; lg++ )
```

```
         FOR( i, 1, N ) dp[lg][i] = -1;

      FOR( i, 1, N ) dp[0][i] = parent[i];

      for( int lg = 1; (1<<lg) < N; lg++ )
         FOR( i, 1, N ) if( dp[lg-1][i] != -1 )
            dp[lg][i] = dp[lg-1][dp[lg-1][i]];

   qry:
   wut = nxt_cmd();
      if( wut != -1 ){
         cin >> u >> v;
         int x = lca(u,v);
         if( wut == 0 )
            cout << dis[u]+dis[v]-2*dis[x] << "\n";
         else {
            k = next_int();
            k--;

            if( L[u]-L[x] >= k ){
               printf("%d\n", kth( k, u ) );

            } else {
               k -= (L[u]-L[x]);
               k = L[v]-L[x]-k;
               printf("%d\n", kth( k, v ) );
            }
         }

         goto qry;
      }
      puts("");
   }
   return 0;
}
```

Link Cut

```
class splay {
public:
   splay *sons[2], *up, *path_up;
   splay() : up(NULL), path_up(NULL) {
      sons[0] = sons[1] = NULL;
   }

   bool is_r(splay* n) {
      return n == sons[1];
   }
};

void rotate(splay* t, bool to_l) {
   splay* n = t->sons[to_l]; swap(t->path_up, n->path_up);
   t->sons[to_l] = n->sons[!to_l]; if(t->sons[to_l]) t->sons[to_l]->up = t;
   n->up = t->up; if(n->up) n->up->sons[n->up->is_r(t)] = n;
   n->sons[!to_l] = t; t->up = n;
}

void do_splay(splay* n) {
   for(splay* p; (p = n->up) != NULL; )
      if(p->up == NULL)
         rotate(p, p->is_r(n));
      else {
         bool dirp = p->is_r(n), dirg = p->up->is_r(p);
         if(dirp == dirg)
            rotate(p->up, dirg), rotate(p, dirp);
         else
            rotate(p, dirp), rotate(n->up, dirg);
      }
}

struct link_cut {
   splay* vtxs;
   link_cut(int numv) { vtxs = new splay[numv]; }
   ~link_cut() { delete[] vtxs; }

   void access(splay* ov) {
      for(splay *w = ov, *v = ov; w != NULL; v = w, w = w->path_up) {
         do_splay(w);
         if(w->sons[1]) w->sons[1]->path_up = w, w->sons[1]->up = NULL;
         if(w != v) w->sons[1] = v, v->up = w, v->path_up = NULL;
         else w->sons[1] = NULL;
      }
      do_splay(ov);
   }

   splay* find(int v) {
      splay* s = &vtxs[v];
      access(s); while(s->sons[0]) s = s->sons[0]; do_splay(s);
      return s;
   }

   void link(int parent, int son) {
      access(&vtxs[son]); access(&vtxs[parent]);
      assert(vtxs[son].sons[0] == NULL);
      vtxs[son].sons[0] = &vtxs[parent];
      vtxs[parent].up = &vtxs[son];
   }

   void cut(int v) {
      access(&vtxs[v]);
      if(vtxs[v].sons[0]) vtxs[v].sons[0]->up = NULL;
      vtxs[v].sons[0] = NULL;
   }
```

```
   int lca(int v, int w) {
      access(&vtxs[v]); access(&vtxs[w]); do_splay(&vtxs[v]);
      if(vtxs[v].path_up == NULL) return v;
      return vtxs[v].path_up - vtxs;
   }
};
```

LIS

```
#include "template.hpp"

int lis(vector<int> &v){
   int n = v.size();
   vector<int> st(n+1,oo);
   vector<int> mx;

   st[0] = -oo;
   int last=0;

   FOR(i,n){
      if(v[i] > st[last]){
         st[++last] = v[i];

      }
      else{
         *lower_bound(st.begin(),st.end(),v[i]) = v[i];
      }
   }
   return last;
}

int main(){
   vector<int> v = {1,5,0,2,5,5,2,3,4};
   cout << "lis␣size␣=␣" << lis(v) << endl;
}
```

Seg Tree

```
class SegTree{
   vector<int> st, st2, id;
   vector<int> lazy;
   int
   void prop(int p, int L, int R){
      if(lazy[p]){
         st[p] += lazy[p];
         lazy[2*p] += lazy[p];
         lazy[2*p+1] += lazy[p];
         lazy[p] = 0;
      }
   }

   void upd(int p, int L, int R, int i, int j, int v){
      prop(p, L, R);

      if(j < L || i > R) return;
      if(i <= L && R <= j){
         lazy[p] = v;
         prop(p, L, R);
         return;
      }

      int mid = (L+R)/2;

      upd(2*p, L, mid, i, j, v);
      upd(2*p+1, mid+1, R, i, j, v);

      st[p] = max(st[2*p],st[2*p+1]);
   }

   int qry(int p, int L, int R, int i, int j){
      prop(p, L, R);

      if(j < L || i > R) return 0;
      if(i <= L && R <= j) return st[p];

      int mid = (L+R)/2;

      return max(qry(2*p, L, mid, i, j), qry(2*p+1, mid+1, R, i, j));
   }

public:

   SegTree(int sz){
      n = sz;
      st.assign(6*(n + 1), 0);
      lazy.assign(6*(n + 1), 0);
   }

   int qry(int i, int j){
      return qry(1, 1, n, i, j);
   }

   void upd(int i, int j, int v){
      upd(1, 1, n, i, j, v);
   }

};
```

Tarjan

```
/* Complexity: O(E + V)
 Tarjan's algorithm for finding strongly connected
components.
 *d[i] = Discovery time of node i. (Initialize to -1)
 *low[i] = Lowest discovery time reachable from node
 i. (Doesn't need to be initialized)
 *scc[i] = Strongly connected component of node i. (Doesn't
 need to be initialized)
 *s = Stack used by the algorithm (Initialize to an empty
 stack)
 *stacked[i] = True if i was pushed into s. (Initialize to
 false)
 *ticks = Clock used for discovery times (Initialize to 0)
 *current_scc = ID of the current_scc being discovered
 (Initialize to 0)
*/
vector<int> g[MAXN];
int d[MAXN], low[MAXN], scc[MAXN];
bool stacked[MAXN];
stack<int> s;
int ticks, current_scc;
void tarjan(int u){
  d[u] = low[u] = ticks++;
  s.push(u);
  stacked[u] = true;
  const vector<int> &out = g[u];
  for (int k=0, m=out.size(); k<m; ++k){
    const int &v = out[k];
    if (d[v] == -1){
      tarjan(v);
      low[u] = min(low[u], low[v]);
    }else if (stacked[v]){
      low[u] = min(low[u], low[v]);
    }
  }
  if (d[u] == low[u]){
    int v;
    do{
      v = s.top();
      s.pop();
      stacked[v] = false;
      scc[v] = current_scc;
    }while (u != v);
    current_scc++;
  }
}
```

Union Find

```
struct UFind{
   int cont;
   vector<int> pai;
   uFind(int n) :cont{n}, pai(n) {
      FOR(i, n) pai[i] = i;
   }
   int find(int i) {
      return pai[i] = (pai[i]==i)? i : find(pai[i]);
   }
   void merge(int i,int j){
      int a = find(i),b = find(j);
      if(a != b){
         cont--;
         pai[a]=b;
      }
   }
};

// Alternative Union-Find
// int parent[MAXVERTICES];
// void make_set(int x){ parent[x] = x; }
// int find(int x){ return (parent[x] == x)? x : parent[x] = find(parent[x]); }
// int joint(int x, int y){ return parent[find(x)] = find(y); }
```

Math

Phi

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

void calc_sieve()
{
   phi[1] = 1;
   for (int i = 2; i <= N; ++i)
   {
      if (lp[i] == 0)
      {
         lp[i] = i;
         phi[i] = i - 1;
         pr.push_back(i);
      }
      else
      {
         //Calculating phi
         if (lp[i] == lp[i / lp[i]])
            phi[i] = phi[i / lp[i]] * lp[i];
         else
            phi[i] = phi[i / lp[i]] * (lp[i] - 1);
```

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

Binomial

```
long binomial_coefficient(n,m){ /* Calculo de Arranjo Rapido */
   int n,m;
   int i,j;
   long bc[MAXN][MAXN];
   for(i=0; i<=n; i++) bc[i][0] = 1;
   for(j=0; j<=n; j++) bc[j][j] = 1;
   for(i=1; i<=n; i++)
      for(j=1; j<i; j++)
         bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
   return( bc[n][m] );
}
```

Bit

```
y = (x & (1 << i)) // Get the i-th bit
x |= (1 << i) // Set the i-th bit
x &= ~(1 << i) // Clear the i-th bit
```

Fast Pot

```
#include "template.hpp"

ll expRap(ll a,ll b,ll mod = oo){
   ll ans=1;
   while(b){
      if(b%2)ans = (ans*a) % mod;
      b /= 2;
      a = (a*a) % mod;
   }
   return ans;
}

int main(){
   cout << expRap(15,15) << endl;
}
```

Fast Fib

```
typedef struct { int v[2][2]; } matriz;
matriz I, FIB;

matriz MM(matriz x, matriz y){ // MATRIZ MULTIPLICATION
   matriz k;
   for(int i = 0; i < 2; i++)
      for(int j = 0; j < 2; j++)
         k.v[i][j] = (x.v[i][0] * y.v[0][j] + x.v[i][1] * y.v[1][j]);
   return k;
}

matriz fastPot(matriz x, int exp){ // FASTPOT MATRIZ EDITION
   if(exp <= 0) return I;
   if(exp%2) return MM(x, fastPot(MM(x, x), (exp-1)/2));
   return fastPot(MM(x, x), exp/2);
}

void startFastPot(int N) { // resposta esta em FIB.v[1][0]
   I.v[0][1] = I.v[1][0] = 0; I.v[0][0] = I.v[1][1] = 1; // matriz identidade
   FIB.v[0][0] = FIB.v[0][1] = FIB.v[1][0] = 1; FIB.v[1][1] = 0;
   FIB = fastPot(INI, N); // N eh o n-ezimo numero de fibonacci
}
```

FFT

```
typedef complex<long double> Complex;
long double PI = 2 * acos(0.0L);
// Decimation-in-time radix-2 FFT.
//
// Computes in-place the following transform:
// y[i] = A(w^(dir*i)),
// where
// w = exp(2pi/N) is N-th complex principal root of unity,
// A(x) = a[0] + a[1] x + ... + a[n-1] x^{n-1}m
// dir \in {-1, 1} is FFTs direction (+1=forward, -1=inverse).
//
// Notes:
// * N must be a power of 2,
// * scaling by 1/N after inverse FFT is callers resposibility.
void FFT(Complex *a, int N, int dir) {
   int lgN;
   for (lgN = 1; (1 << lgN) < N; lgN++);
   assert((1 << lgN) == N);

   for (int i = 0; i < N; ++i) {
      int j = 0;
      for (int k = 0; k < lgN; ++k)
         j |= ((i>>k)&1) << (lgN-1-k);
      if (i < j) swap(a[i], a[j]);
   }
   for (int s = 1; s <= lgN; ++s) {
      int h = 1 << (s - 1);
      Complex t, w, w_m = exp(Complex(0, dir*PI/h));
```

```
      for (int k = 0; k < N; k += h+h) {
         w = 1;
         for (int j = 0; j < h; ++j) {
            t = w * a[k+j+h];
            a[k+j+h] = a[k+j] - t;
            a[k+j] += t;
            w *= w_m;
         }
      }
   }
}
```

GCD

```
int gcd(int a, int b){ return (a%b)? gcd(b, a%b) : b; }
```

LCM

```
int lcm(int a, int b){
   int g = gcd(a, b);
   return g ? (a / g * b) : 0;
}
```

Sieve

```
#include "template.hpp"

///////////////
// CRIVO DE ERASTHOTENES
///////////////

void sieveErathostenes(vector<int>& out, int n){
   vector<bool> v(n+1,false);
   out.push_back(2);
   int i;
   for(i = 3; i*i <= n; i += 2){
      if(!v[i]){
         out.push_back(i);
         for(int j = i*i; j <= n; j *= i)v[j] = true;
      }
   }
   for(;i <= n;i += 2)if(!v[i]) out.push_back(i);
}
```

Sums

```
sumOfLinears = n * (n + 1) * 0.5 // somatorio 1 + ... + n
sumOfSquares = (n * (n + 1) * (2*n + 1)) / 6 // somatoria 1 + ... + n^2
sumOfCubes = sumLinear * sumLinear // somatoria 1 + ... + n^3
```

String

Aho Corasick

```
void trieza(string s,int id){
   int et = 1,n = s.size();
   for(int i = 0; i < n; i++){
      if(trie[et][s[i] - 'a'] == 0)
         trie[et][s[i] - 'a'] = at++;
      et = trie[et][s[i] - 'a'];
   }

   final[et] = id;
}

void aho(){
   queue<pair<int,int>> q;
   q.push(mp(1,-1));
   while(!q.empty()){
      int v = q.front().ff;
      int l = q.front().ss;
      q.pop();
      for(int i = 0; i < 26; i++){
         if(trie[v][i]){
            pai[trie[v][i]] = v;
            q.push(mp(trie[v][i],i));
         }
      }

      if(erro[v] != -1){
         int a = erro[pai[v]];
         while(!erro[v]){
            if(a == -1)
               erro[v] = 1;
            else if(trie[a][l])
               erro[v] = trie[a][l];
            a = erro[a];
         }
      }
      if(acerto[v] != -1)
         acerto[v] = final[erro[v]] ? erro[v] : acerto[erro[v]];
   }
}

int ton(int estado, int c){
   while(!trie[estado][c - 'a']){
      estado = erro[estado];
      if(estado == -1){
         estado = 1;
         break;
```

```
      }
   }

   if(trie[estado][c - 'a'])
      estado = trie[estado][c - 'a'];

   return estado;
}

void corasick(string t,int id){
   int n = t.size(),et = 1,ac;
   for(int i = 0; i < n; i++){

      et = ton(et,t[i]);

      ac = et;
      while(ac != -1)
         ac = acerto[ac];
   }
}
```

Hash

```
#define MAXN 10000
#define BASE 33ULL
#define VALUE(c) ((c)-'a')

typedef unsigned long long hash;

hash h[MAXN], pw[MAXN];

hash calc_hash(int beg, int end) {
   return h[end] - h[beg]*pw[end-beg];
}

void init() {
   pw[0] = 1ULL;
   for (int i=1; i<MAXN; ++i) {
      pw[i] = pw[i-1]*BASE;
   }
   h[0] = 0ULL;
   for (int j=0; s[j]!='\0'; ++j) {
      h[j+1] = h[j]*BASE + VALUE(s[j]);
   }
}
```

Prefix Function

```
void prefixfunction(string S) {
   int N = SIZE(S);
   p[0] = p[1] = 0;
   FOR(i, 2, N) {
      int j = p[i-1];
      while (S[i-1] != S[j]) {
         if (j == 0) { j = -1; break; }
         j = p[j];
      }
      p[i] = ++j;
   }
}
```

Suffix Array

```
/* O( N log N ) SA build + O( N ) LCP build, #include <cstring> :P */
#define MAXN 100000
string S;
int N, SA[MAXN], LCP[MAXN], rank[MAXN], bucket[CHAR_MAX-CHAR_MIN+1];
char bh[MAXN+1];

void buildSA( bool needLCP = false ){
   int a, c, d, e, f, h, i, j, x;
   int *cnt = LCP;
   memset( bucket, -1, sizeof(bucket) );
   for( i = 0; i < N; i++ ){
      j = S[i] - CHAR_MIN;
      rank[i] = bucket[j];
      bucket[j] = i;
   }
   for( a = c = 0; a <= CHAR_MAX-CHAR_MIN; a++ ){
      for( i = bucket[a]; i != -1; i=j ){
         j = rank[i]; rank[i] = c;
         bh[c++] = (i==bucket[a]);
      }
   }
   bh[N] = 1;
   for( i = 0; i < N; i++ )
      SA[ rank[i] ] = i;
   x = 0;
   for( h = 1; h < N; h *= 2 ){
      for( i = 0; i < N; i++ ){
         if( bh[i] & 1 ){
            x = i;
            cnt[x] = 0;
         }
         rank[ SA[i] ] = x;
      }
      d = N-h; e = rank[d];
      rank[d] = e + cnt[e]++;
      bh[rank[d]] |= 2;
```

```
      i = 0;
      while( i < N ){
         for( j = i; (j == i || !(bh[j] & 1)) && j < N; j++ ){
            d = SA[j]-h;
            if( d >= 0 ){
               e = rank[d]; rank[d] = e + cnt[e]++; bh[rank[d]] |= 2;
            }
         }
         for( j = i; (j == i || !(bh[j] & 1)) && j < N; j++ ){
            d = SA[j]-h;
            if( d >= 0 && (bh[rank[d]] & 2)){
               for( e = rank[d]+1; bh[e] == 2; e++);
               for( f = rank[d]+1; f < e; f++ ) bh[f] &= 1;
            }
         }
         i = j;
      }
      for( i = 0; i < N; i++ ){
         SA[rank[i]] = i;
         if( bh[i] == 2 ) bh[i] = 3;
      }
   }

   if( needLCP ){
      LCP[0] = 0;
      for( i = 0, h = 0; i < N; i++ ){
         e = rank[i];
         if( e > 0 ){
            j = SA[e-1];
            while( ((i+h) < N) && ((j+h) < N) && (S[i+h] == S[j+h]) ) h++;
            LCP[e] = h;
            if( h > 0 ) h--;
         }
      }
   }
}
```

Suffix Automata

```
#define MAXN 250000

struct state_t {
   int len, link;
   map< char, int > next;

   bool clone;
   int first_pos;
   vector<int> inv_link;

   int cnt, nxt;
};

int sz, last;
state_t state[2*MAXN];

void automata_init() {
   sz = last = 0;
   state[0].len = 0;
   state[0].link = -1;
   ++sz;
}

void automata_extend(char c) {
   int cur = sz++;
   state[cur].len = state[last].len+1;
   state[cur].first_pos = state[last].len;
   state[cur].cnt = 1;
   int p = last;

   for (; p != -1 && !state[p].next.count(c); p = state[p].link) {
      state[p].next[c] = cur;
   }

   if (p == -1) {
      state[cur].link = 0;
   } else {
      int q = state[p].next[c];
      if (state[p].len+1 == state[q].len) {
         state[cur].link = q;
      } else {
         int clone = sz++;
         state[clone].len = state[p].len+1;
         state[clone].next = state[q].next;
         state[clone].link = state[q].link;
         state[clone].first_pos = state[q].first_pos;
         state[clone].clone = true;
         for (; p != -1 && state[p].next[c]==q; p=state[p].link) {
            state[p].next[c] = clone;
         }
         state[q].link = state[cur].link = clone;
      }
   }
   last = cur;
}

for (int v = 1; v < sz; ++v)
   state[ state[v].link ].inv_link.push_back(v);

int first[n+1];
memset(first,-1,sizeof(first));
for (int v = 0; v < sz; ++v) {
   state[v].nxt = first[state[v].len];
   first[state[v].len] = v;
```

```
}

for (int i = n; i >= 0; --i) {
   for (int u = first[i]; u != -1; u = state[u].nxt) {
      if (state[u].link != -1)
         state[ state[u].link ].cnt += state[u].cnt;
   }
}
```

Z Function

```
void zfunction(string S) {
   int N = SIZE(S), a = 0, b = 0;
   REP(i,N) z[i] = N;

   FOR(i, 1, N-1) {
      int k = (i<b) ? min(b-i,z[i-a]) : 0;
      while (i+k < N && s[i+k]==s[k]) ++k;
      z[i] = k;
      if (i+k > b) { a = i; b = i+k; }
   }
}
```