Contents

1 Basic

1.1 PyMath

```python
import math

math.ceil(x) #上高斯
math.floor(x) #下高斯
math.factorial(x) #階乘
math.fabs(x) #絕對值
math.fsum(arr) #求和
math.gcd(x, y)
math.exp(x) # e^x
math.log(x, base)
math.log2(x)
math.log10(x)
math.sqrt(x)
math.pow(x, y, mod)
math.sin(x) # cos, tan, asin, acos, atan, atan2, sinh ...
math.hypot(x, y) #歐幾里德範數
math.degrees(x) #x從弧度轉角度
math.radians(x) #x從角度轉弧度
math.gamma(x) #x的gamma函數
math.pi #const
math.e #const
math.inf
```

2 Math

2.1 formula

1. Catalan Number

$$C_n = \frac{1}{n}\binom{2n}{n}, C_{n+1} = \frac{2(2n+1)}{n+2}C_n$$

$$C = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, \ldots$$

2. Euler's Formula

對於 V 個點，E 條邊，F 個面，C 個連通分量

$$V + F = E + 2$$

$$V + F = E + C + 1$$

3. Pick's Theorem

點座標均是整數或是正方形格子點的簡單多邊形，其面積 A 和內部點數量 i，邊上格點數量 b 的關係為

$$A = i + \frac{b}{2} - 1$$

2.2 extended gcd

給定 a，b，c，求 $ax + by = c$ 的解

```cpp
    ll extgcd(ll a, ll b, llc, ll &x, ll &y){
        if(b == 0){
            x = c/a ;
            y = 0 ;
            return a ;
        }
        ll d = extgcd(b, a%b, c, x, y), tmp = x ;
        x = y ;
        y = tmp - (a/b)*y ;
        return d ;
    }
```

3 Tree

3.1 SegmentTree

```cpp
#define lc (id << 1)
#define rc ((id << 1) | 1)

struct LazyTag{
    // type 0 : increase val
    // type 1 : set to val
    // type 1 can overwrite type 0
    int type ;
    ll val ;
} ;

struct Node{
    LazyTag tag ;
    ll sum ;
    int sz ;
}seg[Maxn << 2] ;

class SegmentTree{
private:
    void pull(int id){
        seg[id].sum = seg[lc].sum + seg[rc].sum ;
    }

    void AddTag(int id, LazyTag &tag){
        if(tag.type == 0){
            seg[id].sum += tag.val * seg[id].sz ;
            seg[id].tag.val += tag.val ;
        }
        else{
            seg[id].sum = tag.val * seg[id].sz ;
            seg[id].tag = {1, tag.val} ;
        }
    }

    void push(int id){
        AddTag(lc, seg[id].tag) ;
        AddTag(rc, seg[id].tag) ;
        seg[id].tag = {0, 0} ;
    }

public:
    void build(int L=1, int R=n, int id=1){
        seg[id].sum = 0 ;
        seg[id].tag = {0, 0} ;
        seg[id].sz = 1 ;

        if(L == R){
            seg[id].sum = arr[L] ;
            return ;
        }

        int M = (L + R) >> 1 ;
        build(L, M, lc) ;
        build(M+1, R, rc) ;

        pull(id) ;
        seg[id].sz = seg[lc].sz + seg[rc].sz ;
    }

    void modify(int l, int r, LazyTag &tag, int L=1, int R=n, int id=1){
        if(l <= L && R <= r){
            AddTag(id, tag) ;
            return ;
        }

        push(id) ;
        int M = (L + R) >> 1 ;
        if(r <= M) modify(l, r, tag, L, M, lc) ;
        else if(l > M) modify(l, r, tag, M+1, R, rc) ;
        else{
            modify(l, r, tag, L, M, lc) ;
            modify(l, r, tag, M+1, R, rc) ;
        }
        pull(id) ;
    }

    ll query(int l, int r, int L=1, int R=n, int id=1){
        if(l <= L && R <= r) return seg[id].sum ;

        push(id) ;
        int M = (L + R) >> 1 ;
        if(r <= M) return query(l, r, L, M, lc) ;
        else if(l > M) return query(l, r, M+1, R, rc) ;
        else return query(l, r, L, M, lc) + query(l, r, M+1, R, rc) ;
    }
}tree ;
```

3.2 HLD

```cpp
/* HLD */
int fa[Maxn], top[Maxn], son[Maxn], sz[Maxn], dep[Maxn] = {0}, dfn[Maxn], rnk[Maxn], dfscnt = 0 ;

void dfs1(int u, int from){
  fa[u] = from ;
  dep[u] = dep[from] + 1 ;
  sz[u] = 1 ;

  for ( auto v : g[u] ) if(v != from){
    dfs1(v, u) ;
    sz[u] += sz[v] ;
    if(son[u] == -1 || sz[v] > sz[son[u]]) son[u] = v ;
  }
}

void dfs2(int u, int t){
  top[u] = t ;
  dfn[u] = ++dfscnt ;
  rnk[dfscnt] = u ;

  if(son[u] == -1) return ;

  dfs2(son[u], t) ;

  for ( auto v : g[u] ) if(v != fa[u] && v != son[u]){
    dfs2(v, v) ;
  }
}

/* Segment Tree */
#define lc (id << 1)
#define rc ((id << 1) | 1)

struct ColorSeg{
  int left, right, tot ;

  ColorSeg operator+(const ColorSeg &o) const {
    if(tot == 0) return o ;
    if(o.tot == 0) return *this ;

    ColorSeg tmp ;
    tmp.left = left ;
    tmp.right = o.right ;
    tmp.tot = tot + o.tot - (right == o.left) ;

    return tmp ;
  }
} ;

struct Node{
  ColorSeg color ;
  int tag ;
}seg[Maxn << 2] ;

class SegmentTree{
private:
  void pull(int id){
    // normal pull
  }

  void AddTag(int id, int tag){
    // normal AddTag
  }

  void push(int id){
    // normal push
  }

  void modify(int l, int r, int tag, int L=1, int R=n, int id=1){
    // normal modify
  }

  ColorSeg query(int l, int r, int L=1, int R=n, int id=1){
    // normal query
  }
public:
  void build(int L=1, int R=n, int id=1){
    // normal build
  }

  // update val from u to v (simple path)
  void update(int u, int v, int val){
    while(top[u] != top[v]){
      if(dep[top[u]] < dep[top[v]]) swap(u, v) ;
      modify(dfn[top[u]], dfn[u], val) ;
      u = fa[top[u]] ;
    }

    if(dep[u] < dep[v]) swap(u, v) ;
    modify(dfn[v], dfn[u], val) ;
  }

  // get sum from u to v (simple path)
  int get(int u, int v){
    pair<int, ColorSeg> U, V ;
    ColorSeg M ;
    U = {u, {0, 0, 0}} ;
    V = {v, {0, 0, 0}} ;

    while(top[U.first] != top[V.first]){
      if(dep[top[U.first]] < dep[top[V.first]]) swap(U, V) ;
      U.second = query(dfn[top[U.first]], dfn[U.first]) + U.second ;
      U.first = fa[top[U.first]] ;
    }

    if(dep[U.first] < dep[V.first]) swap(U, V) ;

    M = query(dfn[V.first], dfn[U.first]) ;

    return (U.second.tot + V.second.tot + M.tot) - (U.second.left == M.right) - (V.second.left == M.left) ;
  }
}tree ;

void init(){
  memset(son, -1, sizeof(son)) ;
}
```

3.3 PST

```cpp
// Find range k-th largest number
struct Node{
  int sum, left, right ;
}seg[Maxn + 20 * Maxn] ;

class PersistentSegmentTree{
private:
  int n ;
  int cnt ;
  vector<int> version ;

  int build(int L, int R){
    int cur_cnt = cnt++ ;
    if(L == R){
      seg[cur_cnt] = {0, 0, 0} ;
      return cur_cnt ;
    }

    int M = (L + R) >> 1 ;
    int lc = build(L, M) ;
    int rc = build(M+1, R) ;

    seg[cur_cnt] = {0, lc, rc} ;
    return cur_cnt ;
  }
public:
  PersistentSegmentTree(int _n){
    n = _n ;
    cnt = 0 ;

    int root = build(1, n) ;
    version.push_back(root) ;
  }

  void update(int ver, int idx){
    auto upd = [&](auto &&self, const int cur, int L, int R){
      int cur_cnt = cnt++ ;

      if(L == R){
        seg[cur_cnt] = {seg[cur].sum + 1, 0, 0} ;
        return cur_cnt ;
      }

      int M = (L + R) >> 1 ;
      int lc = seg[cur].left ;
      int rc = seg[cur].right ;

      if(idx <= M) lc = self(self, seg[cur].left, L, M) ;
      else rc = self(self, seg[cur].right, M+1, R) ;

      seg[cur_cnt] = {seg[lc].sum + seg[rc].sum, lc, rc} ;

      return cur_cnt ;
    };

    int root = upd(upd, version[ver], 1, n) ;
    version.push_back(root) ;
  }

  int query(int verL, int verR, int k){
    auto qry = [&](auto &&self, const int cur_old, const int cur_new, int L, int R){
      if(L == R) return L ;

      int old_l = seg[cur_old].left, old_r = seg[cur_old].right ;
      int new_l = seg[cur_new].left, new_r = seg[cur_new].right ;

      int dl = seg[new_l].sum - seg[old_l].sum ;
      int dr = seg[new_r].sum - seg[old_r].sum ;

      int M = (L + R) >> 1 ;

      if(dl >= k) return self(self, old_l, new_l, L, M) ;
      k -= dl ;
      return self(self, old_r, new_r, M+1, R) ;
    };

    int idx = qry(qry, version[verL-1], version[verR], 1, n) ;
    return idx ;
  }
};
```

4 Graph

4.1 cut vertex AND bridges

```cpp
int dfn[Maxn] = {-1}, low[Maxn] = {-1}, dfscnt ;

void dfs(int u, int fa){
  dfn[u] = low[u] = ++dfscnt ;
  int child = 0 ;

  for ( auto v : g[u] ) if(v != fa){
    if(dfn[v] == -1){
      child++ ;
      dfs(v, u) ;
      low[u] = min(low[u], low[v]) ;

      if(low[v] >= dfn[u]){
        // this edge is a bridge
      }

      if(u != fa && low[v] >= dfn[u]){
        // this node v is a articulation point
      }
    }
    else low[u] = min(low[u], dfn[v]) ;
  }

  if(u == fa && child > 1){
    // this node u is a articulation point
  }
}
```

4.2 SCC - Tarjan

```cpp
vector<int> scc[Maxn] ;
int dfn[Maxn], low[Maxn], sccId[Maxn], dfscnt = 0, cnt_scc = 0 ;
stack<int> st ;
bitset<Maxn> inSt, vis ;

void dfs(int u, int from){
  dfn[u] = low[u] = ++dfscnt ;
  st.push(u) ;
  inSt[u] = 1 ;

  for ( auto v : g[u] ){
    if(!inSt[v] && dfn[v] != -1) continue ;
    if(dfn[v] == -1) dfs(v, u) ;
    low[u] = min(low[u], low[v]) ;
  }

  if(dfn[u] == low[u]){
    cnt_scc++ ;
    int x ;

    do{
      x = st.top() ;
      st.pop() ;

      inSt[x] = 0 ;
      sccId[x] = cnt_scc ;
      scc[cnt_scc].push_back(x) ;
    }
    while(x != u) ;
  }
}

// SCC to DAG (after dfs)
vector<int> dag[Maxn] ;

void scc_to_dag(){
  vector<int> dag[Maxn] ;
  for ( int u=1 ; u<=n ; u++ ){
    for ( auto v : g[u] ){
      if(sccId[u] != sccId[v]){
        dag[sccId[u]].push_back(sccId[v]) ;
      }
    }
  }
}

void init(){
  memset(dfn, -1, sizeof(dfn)) ;
  memset(low, -1, sizeof(low)) ;
}

int main(){
    init() ;
    input() ;
    for ( int i=1 ; i<=n ; i++ ) if(dfn[i] == -1){
    dfs(i, i) ;
  }
}
```

4.3 BCC - Tarjan

```cpp
struct Edge{
  int v, next ;
}e[Maxm << 1] ;
int head[Maxm], tot = 1 ;

void add(int u, int v){
  e[++tot] = {v, head[u]} ;
  head[u] = tot ;
  e[++tot] = {u, head[v]} ;
  head[v] = tot ;
}

bitset<Maxm << 1> bz ;
vector<vector<int> > bcc ;
int dfn_cnt = 0, dfn[Maxn], low[Maxn], vis_bcc[Maxn], bcc_cnt = 0 ;

void dfs1(int u, int from){
  dfn[u] = low[u] = ++dfn_cnt ;

  for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
    int v = e[i].v ;

    if(dfn[v] == -1){
      dfs1(v, i) ;
      if(dfn[u] < low[v]) bz[i] = bz[i^1] = 1 ;
      low[u] = min(low[u], low[v]) ;
    }
    else if(i != (from ^ 1)) low[u] = min(low[u], dfn[v]) ;
  }
}

void dfs2(int u, int id){
  vis_bcc[u] = id ;
  bcc[id].push_back(u) ;

  for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
    int v = e[i].v ;

    if(vis_bcc[v] != -1 || bz[i]) continue ;
    dfs2(v, id) ;
  }
}

void init(){
  memset(dfn, -1, sizeof(dfn)) ;
  memset(head, -1, sizeof(head)) ;
  memset(vis_bcc, -1, sizeof(vis_bcc)) ;
}

int main(){
    init() ;
    input() ;
    for ( int i=1 ; i<=n ; i++ ) if(dfn[i] == -1){
    dfs1(i, 0) ;
  }

  for ( int i=1 ; i<=n ; i++ ) if(vis_bcc[i] == -1){
    bcc.push_back(vector<int>()) ;
    dfs2(i, bcc_cnt++) ;
  }
}
```

4.4 Convex

```cpp
struct Coordinate{
  long long x, y ;

  friend bool operator<(const Coordinate&a, const Coordinate& b){
    if(a.x == b.x) return a.y < b.y ;
    return a.x < b.x ;
  }

  friend bool operator==(const Coordinate& a, const Coordinate& b){
    return a.x == b.x && a.y == b.y ;
  }
} ;

vector<Coordinate> nodes ;

long long cross(const Coordinate& o, const Coordinate& a, const Coordinate& b){
  return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x) ;
}

void input(){
  nodes.clear() ;

  int n, x, y ;
  char c ;
  cin >> n ;

  for ( int i=0 ; i<n ; i++ ){
    cin >> x >> y >> c ;
    if(c == 'Y') nodes.push_back({x, y}) ;
  }
}

```

```
void monotone(){
  sort(nodes.begin(), nodes.end()) ;

  int n = unique(nodes.begin(), nodes.end())
       - nodes.begin() ;

  vector<Coordinate> ch(n+1) ;

  int m = 0 ;

  for ( int i=0 ; i<n ; i++ ){
    while(m > 1 && cross(ch[m-2], ch[m-1],
         nodes[i]) < 0) m-- ;
    ch[m++] = nodes[i] ;
  }
  for  ( int i=n-2, t=m ; i>=0 ; i-- ){
    while(m > t && cross(ch[m-2], ch[m-1],
         nodes[i]) < 0) m-- ;
    ch[m++] = nodes[i] ;
  }

  if(n > 1) m-- ;
  cout << m << endl ;

  for ( int i=0 ; i<m ; i++ ) cout <<
       ch[i].x << " " << ch[i].y << endl ;
}
```

5 String

5.1 KMP

```
int Next[N] ;
void kmp(string &str){
    Next[0] = -1 ;
    if(str.size() <= 1) return ;
    Next[1] = 0 ;

    int cur = 2, check = 0 ;

    while(cur < str.size()){
        if(str[cur - 1 ] == str[check])
            Next[cur++] = ++check ;
        else if(check > 0) check =
            Next[check] ;
        else Next[cur++] = 0 ;
    }
}

int main(){
    ios::sync_with_stdio(false) ;
    cin.tie(nullptr) ;
    cout.tie(nullptr) ;

    string s1, s2 ;
    while(cin >> s1){
        s2 = s1 ;
        reverse(s2.begin(), s2.end()) ;
        kmp(s2) ;

        int x=0, y=0 ;
        while(x < s1.size() && y < s2.size()){
            if(s1[x] == s2[y]){
                x++ ;
                y++ ;
            }
            else if(y > 0) y = Next[y] ;
            else x++ ;
        }

        cout << s1 << s2.substr(y) << endl ;
    }

    return 0 ;
}
```

5.2 Trie

```
class TrieNode{
public:
    set<int> end ;
    TrieNode *next[26] ;

    TrieNode(){
        for ( int i=0 ; i<26 ; i++ ) next[i]
            = nullptr ;
    }
};

class Trie{
private:
    int cnt ;
    TrieNode *root ;
public:
    Trie() : cnt(0) {
        root = new TrieNode() ;
    }

    void insert(string &str, int n){
        TrieNode* node = root ;
        for ( auto s : str ){
            int path = s - 'a' ;

            if(node->next[path] == nullptr)
                node->next[path] = new
                TrieNode() ;
            node = node->next[path] ;
        }
        node->end.insert(n) ;
    }

    void search(string &str){
        TrieNode* node = root ;
        for ( auto s : str ){
            int path = s - 'a' ;
            if(node->next[path] == nullptr)
                return ;
            node = node->next[path] ;
        }

        int flg = 0 ;
        for ( auto n : node->end ){
            if(flg) cout << " " ;
            else flg = 1 ;

            cout << n ;
        }
    }

    void clear(TrieNode* node) {
        if (!node) return ;
        for (int i = 0; i < 26; i++) {
            if (node->next[i]) {
                clear(node->next[i]) ;
            }
        }
        delete node ;
    }

    ~Trie(){
        clear(root) ;
    }
};
```

5.3 ACAM

```
class ACAutomation{
private:
  vector<int> fail, end, order ;
  vector<vector<int>> tree ;

  int base, alpha ;

  int new_node(){
    tree.emplace_back(alpha, 0) ;
    fail.push_back(0) ;

    return tree.size() - 1 ;
  }
public:
  ACAutomation(int _base='a', int _alpha=26)
    : base(_base), alpha(_alpha) {
      clear() ;
    }

  void clear(){
    fail.assign(1, 0) ;
    order.clear() ;
    end.clear() ;
    tree.assign(1, vector<int>(alpha, 0)) ;
  }

  void add_pattern(const string &pattern){
    int u = 0 ;
    for ( auto ch : pattern ){
      int v = ch - base ;

      if(tree[u][v] == 0) tree[u][v] =
          new_node() ;
      u = tree[u][v] ;
    }

    end.push_back(u) ;
  }

  void build(){
    queue<int> q ;
    order.clear() ;
    order.push_back(0) ;

    for ( int i=0 ; i<alpha ; i++ )
        if(tree[0][i] > 0){
      q.push(tree[0][i]) ;
    }

    while(!q.empty()){
      int u = q.front() ; q.pop() ;
      order.push_back(u) ;

      for ( int i=0 ; i<alpha ; i++ ){
        if(tree[u][i] == 0) tree[u][i] =
            tree[fail[u]][i] ;
        else{
          fail[tree[u][i]] = tree[fail[u]][i]
              ;
          q.push(tree[u][i]) ;
        }
      }
    }
  }

  vector<int> count_per_pattern(const string
      &text) const {
    int u = 0 ;
    vector<int> vis(tree.size(), 0) ;

    for ( char ch : text ){
      u = tree[u][ch - base] ;
      vis[u]++ ;
    }

    for ( int i=order.size()-1 ; i>=1 ; i--
        ){
      int x = order[i] ;
      vis[fail[x]] += vis[x] ;
    }

    vector<int> ans(end.size(), 0) ;
    for ( int id=0 ; id<end.size() ; id++ ){
      ans[id] = vis[end[id]] ;
```

```cpp
    }

    return ans ;
  }
};
```

6 Algorithm

6.1 LCA

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

using namespace std ;

const int Maxn = 500005 ;

vector<int> e[Maxn] ;
int depth[Maxn] ;
int up[Maxn][40] ;
int MaxLog ;

void dfs(int u, int from, int d){
  up[u][0] = from ;
  depth[u] = d ;

  for ( int i=1 ; i<=MaxLog ; i++ ){
    up[u][i] = up[up[u][i - 1]][i - 1] ;
  }

  for ( auto v : e[u] ){
    if(v == from) continue ;
    dfs(v, u, d + 1) ;
  }
}

int lca(int u, int v){
  if(depth[u] < depth[v]) swap(u, v) ;

  for ( int i=MaxLog ; i>=0 ; i-- ) if(depth[u] - (1 << i) >= depth[v]){
    u = up[u][i] ;
  }

  if(u == v) return u ;

  for ( int i=MaxLog ; i>=0 ; i-- ) if(up[u][i] != up[v][i]){
    u = up[u][i] ;
    v = up[v][i] ;
  }

  return up[u][0] ;
}

int main(){
  int n, q, root ;
  scanf("%d%d%d", &n, &q, &root) ;
  MaxLog = __lg(n) ;

  for ( int i=0 ; i<n-1 ; i++ ){
    int u, v ;
    scanf("%d%d", &u, &v) ;
    e[u].push_back(v) ;
    e[v].push_back(u) ;
  }

  dfs(root, root, 0) ;

  while(q--){
    int u, v ;
    scanf("%d%d", &u, &v) ;
    printf("%d\n", lca(u, v)) ;
  }

  return 0 ;
}
```

6.2 MST

```cpp
struct Edge{
  int u, v, w ;
    // 這是最大生成樹, 最小生成樹要改成 w < o.w
  bool operator>(const Edge &o) const {return w > o.w ;} ;
} ;

int par[N] ;
int sz[N] ;
int sum ;

vector<Edge> edge ;

void init(){
  edge.clear() ;
  for ( int i=0 ; i<N ; i++ ){
    par[i] = i ;
    sz[i] = 1 ;
  }
  sum = 0 ;
}

int find(int x){
  if(x == par[x]) return x ;
  return par[x] = find(par[x]) ;
}

int merge(int x, int y){
  x = find(x) ;
  y = find(y) ;

  if(x == y) return 0 ;
  if(sz[x] > sz[y]) swap(x, y) ;
  par[x] = y ;
  sz[y] += sz[x] ;

  return 1 ;
}

void MST(){
  int cnt = 0 ;
  for ( int i=0 ; i<edge.size() && cnt < n-1 ; i++ ){
    auto [u, v, w] = edge[i] ;
    if(merge(u, v)){
      cnt++ ;
      sum -= w ;
    }
  }
}

int main(){
    for ( int i=0 ; i<m ; i++ ){
        scanf("%d%d%d", &u, &v, &w) ;
        edge.push_back({u, v, w}) ;
        sum += w ;
    }

    sort(edge.begin(), edge.end(), greater<Edge>()) ;
    MST() ;
}
```

6.3 SG

```cpp
long long SG(long long k){

  if(k % 2 == 0){
    return k / 2;
  }
  else{
    return SG(k / 2);
  }

}

int main(){
  int cas, n;

  scanf("%d", &cas);
  while(cas--){
    scanf("%d", &n);

    long long s, v = 0;

    for(int i = 0; i < n; i++){
      scanf("%lld", &s);
      v ^= SG(s); //XOR
    }

    if(v) printf("YES\n");
    else printf("NO\n");
  }
}

int SG[30] ;
int vis[Maxn], stone[Maxn] ;

void build(){
  SG[0] = 0 ;
  memset(vis, 0, sizeof(vis)) ;

  for ( int i=1 ; i<30 ; i++ ){
    int cur = 0 ;
    for ( int j=0 ; j<i ; j++ ) for ( int k=0 ; k<=j ; k++ ){
      vis[SG[j] ^ SG[k]] = i ;
    }
    while(vis[cur] == i) cur++ ;
    SG[i] = cur ;
  }
}

int main(){
  build() ;

  int T = 0 ;
  while(~scanf("%d", &n) && n){
    int ans = 0 ;

    for ( int i=1 ; i<=n ; i++ ) scanf("%d", &stone[i]) ;

    for ( int i=1 ; i<=n ; i++ ) if(stone[i] & 1){
      ans ^= SG[n-i] ;
    }
  }
}
```

6.4 Max Flow

```cpp
struct Edge{
  int v, cap, next ;
};

class MaxFlow{
private:
  int N, S, T ;
  vector<Edge> e ;
  vector<int> head, cur, dep ;

  bool bfs(){
    queue<int> q ;
    for ( int i=0 ; i<=N ; i++ ){
      cur[i] = head[i] ;
      dep[i] = -1 ;
    }

    q.push(S) ;
    dep[S] = 0 ;

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

      for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
        int v = e[i].v ;
        if(dep[v] == -1 && e[i].cap > 0){
          dep[v] = dep[u] + 1 ;
          if(v == T) return 1 ;
          q.push(v) ;
        }
      }
    }

    return 0 ;
  }

  int dfs(int u, int flow){
    if(u == T) return flow ;
    int d, rest = 0 ;

    for ( int &i=cur[u] ; i!=-1 ; i=e[i].next ){
      int v = e[i].v ;
      if(dep[v] == dep[u] + 1 && e[i].cap > 0){
        d = dfs(v, min(flow - rest, e[i].cap)) ;

        if(d > 0){
          e[i].cap -= d ;
          e[i^1].cap += d ;
          rest += d ;

          if(rest == flow) break ;
        }
      }
    }

    if(rest != flow) dep[u] = -1 ;
    return rest ;
  }
public:
  MaxFlow(int n, int s, int t){
    N = n ; S = s ; T = t ;
    e.reserve(n*n) ;
    head.assign(n+1, -1) ;
    cur.resize(n+1) ;
    dep.resize(n+1) ;
  }

  void AddEdge(int u, int v, int cap){
    e.push_back({v, cap, head[u]}) ;
    head[u] = e.size() - 1 ;
    e.push_back({u, 0, head[v]}) ;
    head[v] = e.size() - 1 ;
  }

  int run(){
    int ans = 0 ;
    while(bfs()){
      ans += dfs(S, 0x3f3f3f3f) ;
    }
    return ans ;
  }
};
```

6.5 min cut max flow

```cpp
struct Edge{
  int v, cap, cost , next ;
};

using pii = pair<int, int> ;
class MCMF{
private:
  int N, s, t, tot ;
  vector<Edge> e ;
  vector<int> head ;
public:
  MCMF(int n, int _s, int _t){
    N = n ;
    s = _s ;
    t = _t ;
    e.resize(n*n + 5) ;
    head.assign(n+5, -1) ;
    tot = -1 ;
  }

  void AddEdge(int u, int v, int cap, int cost){
    e[++tot] = {v, cap, cost, head[u]} ;
    head[u] = tot ;
    e[++tot] = {u, 0, -cost, head[v]} ;
    head[v] = tot ;
  }

  int run(){
    vector<int> dis(N+1), pot(N+1, 0), preE(N+1) ;
    int flow = 0, cost = 0 ;

    auto dijkestra = [&](){
      fill(dis.begin(), dis.end(), INF) ;
      priority_queue<pii, vector<pii>, greater<pii>> pq ;
      dis[s] = 0 ;
      pq.push({0, s}) ;

      while(!pq.empty()){
        auto [d, u] = pq.top() ; pq.pop() ;
        if(d > dis[u]) continue ;
        for ( int i=head[u] ; i!=-1 ; i=e[i].next ){
          int v = e[i].v, cap = e[i].cap, w = e[i].cost ;
          if(cap && dis[v] > d + w + pot[u] - pot[v]){
            dis[v] = d + w + pot[u] - pot[v] ;
            preE[v] = i ;
            pq.push({dis[v], v}) ;
          }
        }
      }

      return dis[t] != INF ;
    };

    while(dijkestra()){
      for ( int v=1 ; v<=N ; v++ ) if(dis[v] < INF){
        pot[v] += dis[v] ;
      }

      int aug = INT_MAX ;
      for ( int v=t ; v!=s ; v=e[preE[v]^1].v ){
        aug = min(aug, e[preE[v]].cap) ;
      }

      for ( int v=t ; v!=s ; v=e[preE[v]^1].v ){
        e[preE[v]].cap -= aug ;
        e[preE[v]^1].cap += aug ;
        cost += aug * e[preE[v]].cost ;
      }
    }

    return cost ;
  }
};
```

7 DP

7.1 輪廓線 DP

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

using namespace std ;
using ll = long long ;

ll dp[2][(1 << 10) + 5] ;
int n, m ;
int cur ;

void update(int s1, int s2){
  if(s2 & (1 << m)){
    dp[cur][s2 ^ (1 << m)] += dp[cur ^ 1][s1] ;
  }
}

int main(){
  while(~scanf("%d%d", &n, &m)){
    if(m > n) swap(n, m) ;
    memset(dp, 0, sizeof(dp)) ;
    cur = 0 ;
    dp[cur][(1 << m) - 1] = 1 ;
    for ( int i=0 ; i<n ; i++ ) for ( int j=0 ; j<m ; j++ ){
      cur ^= 1 ;
      memset(dp[cur], 0, sizeof(dp[cur])) ;

      for ( int k=0 ; k<(1 << m) ; k++ ){
        update(k, k << 1) ; // not put
        if(i && !(k & (1 << (m - 1))))
          update(k, (k << 1) | (1 << m) | 1) ; // put up
        if(j && !(k & 1)) update(k, (k << 1) | 3) ; // put left
      }
    }
    printf("%lld\n", dp[cur][(1 << m) - 1]) ;
  }
  return 0 ;
}
```

7.2 數位 DP

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

using namespace std ;

int K ;
int dp[20][105][105][2] ;
vector<int> dig ;

int solve(int pos, int sum, int dsum, bool lim){
    if(pos == -1){
        if(sum == 0 && dsum == 0) return 1 ;
        return 0 ;
    }

    int &d = dp[pos][sum][dsum][lim] ;
    if(d != -1) return d ;

    int up = lim ? dig[pos] : 9 ;
    int res = 0 ;
    for ( int i=0 ; i<=up ; i++ ){
        res += solve(pos-1, (sum * 10 + i) % K, (dsum + i) % K, lim && i==up) ;
    }

    return d = res ;
}
```

```cpp

int count(int n){
    memset(dp, -1, sizeof(dp)) ;
    dig.clear() ;

    while(n > 0){
        dig.push_back(n % 10) ;
        n /= 10 ;
    }

    return solve(dig.size() - 1, 0, 0, 1) ;
}

int main(){
    int T ;
    scanf("%d", &T) ;

    int a, b ;
    while(T--){
        scanf("%d%d%d", &a, &b, &K) ;
        if(K > 90) printf("0\n") ;
        else printf("%d\n", count(b) - count(a-1)) ;
    }

    return 0 ;
}
```

7.3 樹 DP

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

#define N 505
#define INF 0x3f3f3f3f

using namespace std ;

struct Edge{
  int v, w ;
} ;

vector<Edge> edge[N] ;
int n ;
int cnt[N] ;
int dp[N][N][2] ;

void init(){
  for ( int i=0 ; i<N ; i++ ) edge[i].clear() ;
  memset(cnt, 0, sizeof(cnt)) ;
  memset(dp, INF, sizeof(dp)) ;
}

void DFS(int u){
  cnt[u] = 1 ;
  for ( auto [v, w] : edge[u] ){
    DFS(v) ;
    cnt[u] += cnt[v] ;
  }

  dp[u][1][0] = dp[u][1][1] = 0 ;

  for ( auto [v, w] : edge[u] ){
    for ( int i=cnt[u] ; i>1 ; i-- ) for ( int j=1 ; j<i && j<=cnt[v] ; j++ ){
      dp[u][i][1] = min(dp[u][i][1], dp[u][i-j][1] + dp[v][j][1] + 2 * w) ;
      dp[u][i][0] = min(dp[u][i][0], dp[u][i-j][1] + dp[v][j][0] + w) ;
      dp[u][i][0] = min(dp[u][i][0], dp[u][i-j][0] + dp[v][j][1] + 2 * w) ;
    }
  }
}

int main(){
  int t = 0 ;

  while(~scanf("%d", &n) && n){
    init() ;
    for ( int i=0 ; i<n-1 ; i++ ){
      int u, v, w ;
      scanf("%d%d%d", &v, &u, &w) ;
      edge[u].push_back({v, w}) ;
    }

    DFS(0) ;
    printf("Case %d:\n", ++t) ;

    int q, e ;
    scanf("%d", &q) ;

    while(q--){
      scanf("%d", &e) ;

      for ( int i=n ; i>=1 ; i-- ) if(dp[0][i][0] <= e){
        printf("%d\n", i) ;
        break ;
      }
    }
  }

  return 0 ;
}
```