# Team Notebook

PichonesDeIngenieros

October 14, 2024

## Contents

# 1 Data Structures

## 1.1 Fenwick Tree

```cpp
template <typename T>
struct BIT {
   vector<T> ft;
   int N;
   BIT(int n): ft(n + 1), N(n) {}

   BIT(const vector<T>& a): ft(sz(a) + 1), N(sz(a)) {
      forn(i, sz(a)) {
         upd(i + 1, a[i]);
      }
   }

   T qry(int i) {
      T ans = 0;
      for (; i; i -= i & -i) ans += ft[i];
      return ans;
   }

   T qry(int l, int r) {
      return qry(r) - qry(l - 1);
   }

   void upd(int i, T v) {
      for (; i < sz(ft); i += i & -i) ft[i] += v;
   }
   //This is equivalent to calculating
   //lower_bound on prefix sums array
   int bit_search(int v) { // O(log(N))
      int sum = 0;
      int pos = 0;
      int LOGN = int(log2(N));
      for (int i = LOGN; i >= 0; i--) {
         if (pos + (1 << i) < N && sum + ft[pos + (1 << i)] <
               v) {
            sum += ft[pos + (1 << i)];
            pos += (1 << i);
         }
      }
      return pos + 1;
   }
};
```

## 1.2 Fenwick$_{r}ange_{u}pd$

```cpp
// [1, n]
//
template <typename T>
struct ft_range {
  vector<T> ft1, ft2;

  ft_range(int n) {
    ft1.assign(n + 1, 0);
    ft2.assign(n + 1, 0);
  }

  ft_range(vector<T> &a) : ft1(sz(a) + 1), ft2(sz(a) + 1) {
    forab(i, 1, sz(a) + 1)
    update(i, i, a[i]);
  }

  T query(vector<T> & ft, int i) {
    T sum = 0;
    for(; i; i -= (i & -i)) sum += ft[i];
    return sum;
  }
  void update(vector<T> & ft, int i, int v) {
    for(; i < sz(ft); i += (i & -i))
      ft[i] += v;
  }
  void update(int i, int j, T v) {
    update(ft1, i, v);
    update(ft1, j + 1, -v);
    update(ft2, i, v * (i - 1));
    update(ft2, j + 1, -v * j);
  }
  T query(int i) {
    return query(ft1, i) * i - query(ft2, i);
  }
  T query(int i, int j) {
    return query(j) - query(i - 1);
  }
};

int main() {
  int n;
  cin >> n;
  vector<int> v(n + 1);
  forab(i, 1, n + 1) cin >> v[i];
  ft_range<int> bit(v);
  int q;
  cin >> q;
  int l, r, val;
  while(q--) {
    char op;
    cin >> op;
    if(op == 'q') {
      cin >> l >> r;
      cout << bit.query(l, r) << ln;
    } else {
      cin >> l >> r >> val;
      bit.update(l, r, val);
    }
  }
  return 0;
}
```

## 1.3 Heavy Light Decomposition

```cpp
const int N = 2e5 + 5;
const int D = 19;
const int S = (1 << D);

int n, q;
ll v[N]; //value of each node
ll st[S]; //segtree
vi g[N]; //graph
int tam[N];//subtree size
int p[N], dep[N];
int id[N];//id of node in the segtree
int tp[N];


ll oper(ll a, ll b) {
  return max(a, b);
}

void update(int idx, ll val) {
  st[idx += n] = val;
  for (idx /= 2; idx; idx /= 2){
    st[idx] = oper(st[2 * idx], st[2 * idx + 1]);
  }
}

ll query(int lo, int hi) {
  ll ra = 0, rb = 0;
  for (lo += n, hi += n + 1; lo < hi; lo /= 2, hi /= 2) {
    if (lo & 1) ra = oper(ra, st[lo++]);
    if (hi & 1) rb = oper(rb, st[--hi]);
  }
```

```
  return oper(ra, rb);
}

int dfs_sz(int cur, int par) {
  tam[cur] = 1;
  p[cur] = par;
  for (int chi : g[cur]) {
    if (chi == par) continue;
    dep[chi] = dep[cur] + 1;
    p[chi] = cur;
    tam[cur] += dfs_sz(chi, cur);
  }
  return tam[cur];
}

int ct = 1;

void dfs_hld(int cur, int par, int top) {
  id[cur] = ct++;
  tp[cur] = top;
  update(id[cur], v[cur]);
  int h_chi = -1, h_sz = -1;
  for (int chi : g[cur]) {
    if (chi == par) continue;
    if (tam[chi] > h_sz) {
      h_sz = tam[chi];
      h_chi = chi;
    }
  }
  if (h_chi == -1) return;
  dfs_hld(h_chi, cur, top);
  for (int chi : g[cur]) {
    if (chi == par || chi == h_chi) continue;
    dfs_hld(chi, cur, chi);
  }
}

ll path(int x, int y) {
  ll ret = 0;
  while (tp[x] != tp[y]) {
    if (dep[tp[x]] < dep[tp[y]]) swap(x, y);
    ret = oper(ret, query(id[tp[x]], id[x]));
    x = p[tp[x]];
  }
  if (dep[x] > dep[y]) swap(x, y);
  ret = oper(ret, query(id[x], id[y]));
  return ret;
}

int main() {
  cin >> n >> q;
  forab(i, 1,n + 1) cin >> v[i];
  forn(i, n - 1) {
    int a, b;
    cin >> a >> b;
    g[a].pb(b);
    g[b].pb(a);
  }
  dfs_sz(1, 1);
  dfs_hld(1, 1, 1);
  int op;
  while (q--) {
    cin >> op;
    if(op == 1){
      int s, x;
      cin >> s >> x;
      v[s] = x;
      update(id[s], x);
    }else{
      int a, b;
      cin >> a >> b;
      ll res = path(a, b);
      cout << res << " ";
    }
  }
  cout << ln;
  return 0;
}
```

## 1.4 Mo's

```
int S, n, q;

struct query {
  int l, r, idx;
  query(int l, int r, int idx): l(l), r(r), idx(idx) {}

  bool operator < (const query &x) const {
    if (l / S != x.l / S) return l / S < x.l / S;
    return (l / S & 1) ? r < x.r: r > x.r;
  }
};

const int MAXN = 1e6 + 1;
ll sum = 0;
vector<query> qu;
vector<ll> ans;
vi feq(MAXN, 0);
vll a;

void add(int idx) {
  ll val = a[idx];
  sum = sum - (val * feq[val] * feq[val]);
  feq[val]++;
  sum = sum + (val * feq[val] * feq[val]);
}

/*
Very common pattern:
Subtract power of element from total
Update count
Add back power of element to total
*/
void del(int idx) {
  ll val = a[idx];
  sum = sum - (val * feq[val] * feq[val]);
  feq[val]--;
  sum = sum + (val * feq[val] * feq[val]);
}

ll get_ans() {
  return sum;
}

void mo_s() {
  S = sqrt(n);
  sort(all(qu));
  ans.resize(q);
  int l = 0, r = -1;
  for (query &it: qu) {
    while (r < it.r) add(++r);
    while (l > it.l) add(--l);
    while (r > it.r) del(r--);
    while (l < it.l) del(l++);
    ans[it.idx] = get_ans();
  }
}

int main() {
  cin >> n >> q;
  a = vll(n);
  forn(i, n) cin >> a[i];

  int l, r;
  forn(i, q) {
    cin >> l >> r;
    l--;
    r--;
```

```
    qu.pb({l, r, i});
  }
  mo_s();
  forn(i, q) cout << ans[i] << ln;
  return 0;
}
```

## 1.5 Prefix sum

```
//arr[0] es ignorado
vi arr = {0, 1, 6, 4, 2, 5, 3 };
int n = arr.size();
vll pf(n + 1, 0);
forab (i, 1, n + 1) {
    pf[i] = pf[i - 1] + arr[i];
}
//query en rango desde a hasta b
// a va desde 1 hasta n
// b va desde 1 hasta n
//[a, b] inclusive
int a = 2;
int b = 6;
ll query = pf[b] - pf[a - 1];
cout << query << ln;
```

## 1.6 SegmentTree 2D

```
template<typename T>
struct STree {
   int n, m;
   T neutro = T(0);
   vector<vector<T>> st;

   STree(vector<vector<T>> &a) {
      n = sz(a);
      m = sz(a[0]);
      st = vector<vector<T>>(2 * n, vector<T>(2 * m, neutro))
           ;
      build(a);
   }

   inline T oper(T a, T b) {
      return a + b;
   }

   void build(vector<vector<T>> &a) {
      forn (i, n) forn (j, m)
      st[i + n][j + m] = a[i][j];

      forn (i, n) rform (j, m - 1, 1, 1)
      st[i + n][j] = oper(st[i + n][j << 1], st[i + n][j << 1
           | 1]);

      rform (i, n - 1, 1, 1) forn (j, 2 * m)
      st[i][j] = oper(st[i << 1][j], st[i << 1 | 1][j]);
   }
   //esquina izq arriba y esquina der abajo [row, col]
   T qry(int x1, int y1, int x2, int y2) { // [x1, y1] (x2,
      y2)
      T ans = neutro;
      for(int i0 = x1 + n, i1 = x2 + n; i0 < i1; i0 >>= 1, i1
           >>= 1) {
         int t[4], q = 0;
         if (i0 & 1) t[q++] = i0++;
         if (i1 & 1) t[q++] = --i1;
         forn (k, q)
         for (int j0 = y1 + m, j1 = y2 + m; j0 < j1; j0 >>=
              1, j1 >>= 1) {
            if(j0 & 1) ans = oper(ans,st[t[k]][j0++]);
            if(j1 & 1) ans = oper(ans,st[t[k]][--j1]);
         }
      }
      return ans;
   }

   void upd(int l, int r, T val) {
      st[l + n][r + m] = val;
      for (int j = r + m; j > 1; j >>= 1)
         st[l + n][j >> 1] = oper(st[l + n][j], st[l + n][j ^
              1]);

      for (int i = l + n; i > 1; i >>= 1)
         for (int j = r + m; j; j >>= 1)
            st[i >> 1][j] = oper(st[i][j], st[i ^ 1][j]);
   }
};
```

## 1.7 SegmentTree Iterativo

```
template<typename T>
struct STree {
  vector<T> st;
  int n;
  T neutro = T(1e9);
  T oper(T a, T b) { return min(a, b); }

  STree(vector<T> &a) {
    n = sz(a);
    st.resize(n * 2);
    forn (i, n) st[n + i] = a[i];
    rform (i, n - 1, 1, 1) st[i] = oper(st[i << 1], st[i << 1
         | 1]);
  }

  void upd(int p, T val) {
    for (st[p += n] = val; p > 1; p >>= 1) st[p >> 1] = oper(
         st[p], st[p ^ 1]);
  }

  T query(int l, int r) { //[l, r)
    T v = neutro;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) v = oper(v, st[l++]);
      if (r & 1) v = oper(v, st[--r]);
    }
    return v;
  }
};
```

## 1.8 SegmentTree Lazy

```
template<typename T>
struct STree {
  int n; vector<T> st, lazy;
  T neutro = T(0);

  STree(int m) {
    n = m;
    st.resize(n * 4);
    lazy.resize(n * 4);
  }

  STree(vector<T> &a) {
    n = sz(a);
    st.resize(n * 4);
    lazy.resize(n * 4);
    build(1, 0, n - 1, a);
  }

  T oper(T a, T b) { return a + b; }

  void build(int v, int tl, int tr, vector<T> &a) {
    if(tl == tr) {
```

```
    st[v] = a[tl];
    return;
  }
  int tm = (tl + tr) / 2;
  build(v * 2, tl, tm, a);
  build(v * 2 + 1, tm + 1, tr, a);
  st[v] = oper(st[v * 2], st[v * 2 + 1]);
}

void push(int v, int tl, int tr) {
  if (!lazy[v]) return;
  st[v] += (tr - tl + 1) * lazy[v];
  if (tl != tr) {
    lazy[v * 2] += lazy[v];
    lazy[v * 2 + 1] += lazy[v];
  }
  lazy[v] = 0;
}

void upd(int v, int tl, int tr, int l, int r, T val) {
  push(v, tl, tr);
  if(tr < l || tl > r) return;
  if (tl >= l && tr <= r) {
    lazy[v] = val;
    push(v, tl, tr);
    return;
  }
  int tm = (tl + tr) / 2;
  upd(v * 2, tl, tm, l, r, val);
  upd(v * 2 + 1, tm + 1, tr, l, r, val);
  st[v] = oper(st[v * 2], st[v * 2 + 1]);
}

T query(int v, int tl, int tr, int l, int r) {
  push(v, tl, tr);
  if(tl > r || tr < l) return neutro;
  if (l <= tl && tr <= r) return st[v];
  int tm = (tl + tr) / 2;
  return oper(query(v * 2, tl, tm, l, r), query(v * 2 + 1,
      tm + 1, tr, l, r));
}

void upd(int l, int r, T val) { upd(1, 0, n - 1, l, r, val
    ); }
T query(int l, int r) { return query(1, 0, n - 1, l, r); }
};
```

## 1.9 SegmentTree Regular Bracket

```
struct node {
  int start, end, maxLen;
  /*{start, end}*/
};
struct STregularBracket {
  vector<node> seg;
  int size;

  STregularBracket(string& ch){
    size = ch.length();
    seg.resize(4 * size);
    build(1, 1, size - 1, ch);
  }

  void build(int idx, int s, int e, string& ch){
    if (s == e) {
      if (ch[s] == '(') {
        seg[idx] = { 1, 0 };
      } else {
        seg[idx] = { 0, 1 };
      }
      return;
    }
    build(idx << 1, s, (s + e) / 2, ch);
    build(idx << 1 | 1, (s + e) / 2 + 1, e, ch);
    seg[idx] = { seg[idx << 1 | 1].start, seg[idx << 1].end
        };
    int dif = seg[idx << 1].start - seg[idx << 1 | 1].end;
    int mini = min(seg[idx << 1].start, seg[idx << 1 | 1].end
        );
    seg[idx].maxLen += mini * 2 + seg[idx << 1 | 1].maxLen +
        seg[idx << 1].maxLen;
    if (dif > 0) {
      seg[idx].start += dif;
    } else {
      seg[idx].end -= dif;
    }
  }

  node query(int idx, int s, int e, int l, int r){
    if (l > e || s > r) {
      return { 0, 0 };
    }
    if (s >= l && e <= r) {
      return seg[idx];
    }
    node p1 = query(idx << 1, s, (s + e) / 2, l, r);
    node p2 = query(idx << 1 | 1, (s + e) / 2 + 1, e, l, r);
    node ans = { p2.start, p1.end };
    int dif = p1.start - p2.end;
    ans.maxLen += p1.maxLen + p2.maxLen;
    ans.maxLen += min(p1.start, p2.end) * 2;
    if (dif > 0) {
      ans.start += dif;
    } else {
      ans.end -= dif;
    }
    return ans;
  }
  // [1, n]
  node query(int l, int r){
    return query(1, 1, size - 1, l, r);
  }
};
// https://codeforces.com/contest/380/problem/C

int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);

  string w;
  cin >> w;
  w = "0" + w; // se agrega un comodin al inicio
  STregularBracket seg(w);
  int q, l, r;
  cin >> q;
  while (q--) {
    cin >> l >> r;
    cout << (seg.query(l, r).maxLen) << ln;
  }
  return 0;
}
```

## 1.10 SegmetTree

```
template <typename T>
struct STree {
  int n;
  vector<T> st;
  T neutro = T(0);

  STree(vector<T>& a){
    n = sz(a);
    st.resize(n * 4);
    build(1, 0, n - 1, a);
  }

  T oper(T a, T b) { return max(a, b); }
```

```
void build(int v, int tl, int tr, vector<T>& a){
  if (tl == tr) {
    st[v] = a[tl];
    return;
  }
  int tm = (tr + tl) / 2;
  build(v * 2, tl, tm, a);
  build(v * 2 + 1, tm + 1, tr, a);
  st[v] = oper(st[v * 2], st[v * 2 + 1]);
}

T query(int v, int tl, int tr, int l, int r){
  if (tl > r || tr < l)
    return neutro;
  if (l <= tl && tr <= r)
    return st[v];
  int tm = (tl + tr) / 2;
  return oper(query(v * 2, tl, tm, l, r), query(v * 2 + 1,
       tm + 1, tr, l, r));
}

void upd(int v, int tl, int tr, int pos, T val){
  if (tl == tr) {
    st[v] = val;
    return;
  }
  int tm = (tr + tl) / 2;
  if (pos <= tm)
    upd(v * 2, tl, tm, pos, val);
  else
    upd(v * 2 + 1, tm + 1, tr, pos, val);
  st[v] = oper(st[v * 2], st[v * 2 + 1]);
}

int countQuery(int v, int tl, int tr, int l, int r, T x){
  if (tl > r || tr < l)
    return 0;
  if (l <= tl && tr <= r) {
    if (st[v] <= x) {
      /*
      Para mayores st[v] <= x query max(a,b)
      Para mayores o equ st[v] < x query max(a,b)
      Para menores st[v] >= x query min(a,b)
      Para menores o equ st[v] > x query min(a,b)
      */
      return 0;
    }
    if (tl == tr)
       return 1;
    }
    int tm = (tl + tr) / 2;
    return countQuery(v * 2, tl, tm, l, r, x) + countQuery(v
         * 2 + 1, tm + 1, tr, l, r, x);
  }
  int countQuery(int a, int b, T x){
    return countQuery(1, 0, n - 1, a, b, x);
  }
  void upd(int pos, T val) { upd(1, 0, n - 1, pos, val); }
  T query(int l, int r) { return query(1, 0, n - 1, l, r); }
};
```

## 1.11 Sparse Table

```
struct STable {
  int n, K;
  vector<vi> st;

  STable(const vi &a) {
    n = sz(a);
    K = int(log2(n)) + 1;
    st.assign(n + 1, vi(K));
    forn (i, n) st[i][0] = a[i];
    forn (j, K - 1)
      for (int i = 0; i + (1 << (j + 1)) <= n; ++i)
        st[i][j + 1] = oper(st[i][j], st[i + (1 << j)][j]);
  }

  int oper(int a, int b) { return __gcd(a, b); }

  int query(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    return oper(st[l][k], st[r - (1 << k) + 1][k]);
  }
};
```

## 1.12 Trie XOR

```
struct Node {
   Node* childs[2];
};

struct Trie {
   Node* root;

   Trie() {
      root = new Node();
   }

   void insert(int x) {
      Node* cur = root;
      int i = 32;
      while (i--) {
         int bit = (x >> i) & 1;
         if (cur->childs[bit] == NULL) {
            cur->childs[bit] = new Node();
         }
         cur = cur->childs[bit];
      }
   }

   // max xor entre un elemento y query
   int maxXorQuery(int query) {
      Node* cur = root;
      int ans = 0;
      int i = 32;
      while (i--) {
         int bit = (query >> i) & 1;
         if (cur->childs[1 - bit] != NULL) {
            ans = ans | (1 << i);
            cur = cur->childs[1 - bit];
         } else {
            cur = cur->childs[bit];
         }
      }
      return ans;
   }

   // maximo xor entre dos elementos en un arreglo
   int maxXor(vi &arr) {
      int n = sz(arr);
      int max_val = 0;
      insert(arr[0]);
      for (int i = 1; i < n; i++) {
         max_val = max(maxXorQuery(arr[i]), max_val);
         insert(arr[i]);
      }
      return max_val;
   }
};
```

## 1.13 Union Find

```
struct UnionFind {
  vi parent;
  vi comp_sz;
```

```
  vi maxi;
  vi mini;

  UnionFind(int n) {
    n++;
    parent = vi(n);
    comp_sz = vi(n);
    maxi = vi(n);
    mini = vi(n);
    forn(i, n) {
      parent[i] = i;
      comp_sz[i] = 1;
      maxi[i] = i;
      mini[i] = i;
    }
  }
  // Path compression optimization
  int find(int a) {
    return parent[a] = (parent[a] == a) ? a : find(parent[a])
        ;
  }
  // Union by size
  void unite(int x, int y) {
    int a = find(x);
    int b = find(y);
    if (a == b)return;
    if(comp_sz[a] < comp_sz[b]) swap(a, b);
    parent[b] = a;
    mini[a] = min(mini[a], mini[b]);
    maxi[a] = max(maxi[a], maxi[b]);
    comp_sz[a] += comp_sz[b];
  }

  bool isSame(int a, int b) {
    return find(a) == find(b);
  }
};
```

# 2 Dynamic Programming

## 2.1 Counting paths in a DAG

```
int main() {
  cin >> n >> m;
  int a, b;
  vi grado(n + 1, 0);
  forn(i, m){
    cin >> a >> b;
    g[a].pb(b);
    grado[b]++;
  }
  vi topo = topo_sort(grado);
  int source = 1;
  int dest = n;
  //cuidado con overflow
  int dp[n] = { 0 };
  dp[dest] = 1;
  // traverse in reverse order
  rforn (i, sz(topo)) { // O(n + m)
    forn (j, sz(g[topo[i]])) {
      dp[topo[i]] += dp[g[topo[i]][j]];
    }
  }
  //number of paths from source
  //to dest
  cout << dp[source] << ln;
  return 0;
}
```

## 2.2 Counting tilings

```
//counting tilings
const int MOD = 1e9 + 7;
int dp[1001][1<<10];
int n, m;

void fill_colum(int column, int idx, int cur_mask, int
     next_mask) {
   if (idx == n) { // he llenado toda la columna
      dp[column + 1][next_mask] = (dp[column + 1][next_mask]
          + dp[column][cur_mask]) % MOD;
      return;
   }
   if ((cur_mask) & (1 << idx)) { // si el tile actual esta
       full
      fill_colum(column, idx + 1, cur_mask, next_mask);
   } else {
      // horizontal
      fill_colum(column, idx + 1, cur_mask, next_mask | (1 <<
           idx));
      // vertical
      if (idx + 1 < n && (!(cur_mask & (1 << (idx + 1))))) {
         fill_colum(column, idx + 2, cur_mask, next_mask);
      }
   }
}

int main() {
   ios_base::sync_with_stdio(0);
   cin.tie(0);
   cout.tie(0);
   cin >> n >> m;
   dp[0][0] = 1;
   forn(column, m) { // todas las coumnas
      forn(mask, (1 << n)) { // todas las posibles mask
         if (dp[column][mask] > 0) {
            // hay forma de llenar la i-esima columna
            fill_colum(column, 0, mask, 0);
         }
      }
   }
   cout << dp[m][0] << ln;

   return 0;
}
```

## 2.3 Generating sums

```
const int MAX_N = 101;
const int MAX_SUM = 1e5 + 1;
bool dp[MAX_N + 1][MAX_SUM + 1];

int main() {
   //dada una cantidad de monedas, decir todas
   //sumas que se pueden formar con ellas
   int n;
   cin >> n;
   vi coins(n);
   for(auto &i: coins) cin >> i;
   dp[0][0] = true;
   forab (i, 1, n + 1) {
      forab (curSum, 0, MAX_SUM + 1) {
         dp[i][curSum] = dp[i - 1][curSum];
         int prevSum = curSum - coins[i - 1];
         if (prevSum >= 0 && dp[i - 1][prevSum]) {
            dp[i][curSum] = true;
         }
      }
   }
   vi ans;
   forab (sum, 1, MAX_SUM + 1) {
      if (dp[n][sum])ans.pb(sum);
   }
```

```
  cout << sz(ans) << ln;
  for (int sum : ans)
    cout << sum << " ";
  cout << ln;
  return 0;
}
```

## 2.4 LIS logn

```
//longest increasing subsecuence
//O(n log n)
const ll oo = 1e18;
int main() {
  int n;
  cin >> n;
  vll a(n);
  for(auto &i: a) cin >> i;
  vll dp(n + 1, oo);
  dp[0] = -oo;
  forab(i, 0, n) {
    int l = upper_bound(all(dp), a[i]) - dp.begin();
    if(dp[l - 1] < a[i] && a[i] < dp[l]) {
      dp[l] = a[i];
    }
  }
  ll ans = 0;
  forab(l, 0, n + 1) {
    if(dp[l] < oo) ans = l;
  }
  cout << ans << ln;
  return 0;
}
```

## 2.5 Traveling Salesman Problem

```
//shortest route
//it is a closed cycle where it
//ends at the same point it starts

// it visits each node exactly once
const int oo = 1e9;
int n, m;
const int N = 16;
vector<vi> g(N, vi(N, oo)); // directed graph
int dp[1 << N][N];

int go(int mask, int u) {
  if(mask == ((1 << n) - 1)) {
    return g[u][0] != oo ? g[u][0] : oo;
  }
  if(dp[mask][u] != - 1) return dp[mask][u];

  int ans = oo;
  forn(v, n) {
    if((mask & (1 << v)) == 0) {
      int aux = go(mask | (1 << v), v) + g[u][v];
      ans = min(ans, aux);
    }
  }
  return dp[mask][u] = ans;
}

int main() {
  cin >> n >> m;
  int a, b, w;
  forn(i, m) {
    cin >> a >> b >> w;
    g[a][b] = w;
  }
  memset(dp, -1, sizeof dp);
  int ans = go(1, 0); // mask, node
  cout << (ans == oo ? -1: ans) << ln;
  return 0;
}
```

## 2.6 digit dp

```
//digit dp
//[0, a]
const int MAXDIGT = 20;
short num[MAXDIGT];
pair<ll, ll> dp[MAXDIGT][2];
int n; // number of digits
int k;

//What if we want [a, b]
//ans(b) - ans(a - 1)

pair<ll, ll> go(int idx, bool is_eq){
  if(dp[idx][is_eq].first != -1) return dp[idx][is_eq];
  if(idx == n) return {1, 0};

  ll cont = 0, sum = 0;
  for(int i = 0; i <= (is_eq ? num[idx] : 9); i++){
    pair<ll, ll> cur = go(idx + 1, is_eq && i == num[idx]);
    cont += cur.first;
    sum += cur.second + cur.first * i;
  }
  return dp[idx][is_eq] = {cont, sum};
}

ll solve(ll x){
  string aux = to_string(x);
  n = sz(aux);
  forn(i, n) num[i] = short(aux[i] - '0');
  memset(dp, -1, sizeof dp);
  return go(0, 1).second;
}

int main() {
  while(true){
    ll a, b; cin >> a >> b;
    if(a < 0 && b < 0) break;
    ll aa = solve(a - 1);
    ll bb = solve(b);
    cout << bb - aa << ln;
  }
  return 0;
}
```

## 2.7 digit permutation

```
//we have two integers N and M,
//we need to find how many numbers
// obtained by rearranging
//digits of N are divisible by M.

int getdigit(char c) {
  return int(c - '0');
}

string s;
ll m;
ll dp[101][(1 << 18) + 1];
ll base = 10;

// go(0, 0);
ll go(ll rem, int mask) {
  if(rem == 0LL && mask == ((1 << sz(s)) - 1)) {
    return 1;
  }
  if(dp[rem][mask] != -1) return dp[rem][mask];
  ll ans = 0;
```

```
  forn(i, sz(s)) {
    if(!(mask & (1 << i))) {
      int digit = getdigit(s[i]);
      //leading zeros
      if(mask == 0 && digit == 0) continue;
      ans += go(((rem * base) + digit) % m, mask | (1 << i));
    }
  }
  return dp[rem][mask] = ans;
}

int main() {
  cin >> s >> m;
  vi cnt(10, 0);
  ll repetidos = 1;
  for(char c: s) {
    repetidos *= ++cnt[c - '0'];
  }
  vector<vll> dp(101, vll(1 << 18, 0));
  dp[0][0] = 1LL;
  forn(mask, (1 << sz(s))) {
    forn(i, sz(s)) {
      if(!(mask & (1 << i))) {
        int digit = getdigit(s[i]);
        if(mask == 0 && digit == 0) continue;
        forn(rem, m) {
          dp[((1LL * rem * base) + digit) % m][mask | (1 << i
              )] += dp[rem][mask];
        }
      }
    }
  }
  ll ans = dp[0][(1 << sz(s))- 1];
  cout << (ans / repetidos) << ln;
  return 0;
}
```

# 3 Flows

## 3.1 Dinic

```
// Min Vertex Cover: vertices de L con level[v]==-1 y
    vertices de R con level[v]>0
// Max Independent Set: vertices NO tomados por el Min
    Vertex Cover
typedef pair<int, int> ii;

struct FlowEdge {
  int v, u;
  ll cap, flow = 0;
  FlowEdge(int _v, int _u, ll _cap) : v(_v), u(_u), cap(_cap
      ) {}
};

struct Dinic { // O(V^2 * E)
  const ll flow_inf = 1e18;
  vector<FlowEdge> edges;
  vector<vector<int>> g;
  int n, m = 0;
  int s, t;
  vector<int> level, ptr;
  queue<int> q;

  Dinic(int _n, int _s, int _t) : n(_n), s(_s), t(_t) {
    g.resize(n);
    level.resize(n);
    ptr.resize(n);
  }

  void add_edge(int v, int u, ll cap) {
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    g[v].push_back(m);
    g[u].push_back(m + 1);
    m += 2;
  }

  bool bfs() {
    while (sz(q)) {
      int v = q.front();
      q.pop();
      for (int id : g[v]) {
        if (edges[id].cap - edges[id].flow < 1) continue;
        if (level[edges[id].u] != -1) continue;
        level[edges[id].u] = level[v] + 1;
        q.push(edges[id].u);
      }
    }
    return level[t] != -1;
  }

  ll dfs(int v, ll pushed) {
    if (pushed == 0) return 0;
    if (v == t) return pushed;
    for (int &cid = ptr[v]; cid < sz(g[v]); ++cid) {
      int id = g[v][cid];
      int u = edges[id].u;
      if (level[v] + 1 != level[u] || edges[id].cap - edges[
          id].flow < 1) continue;
      ll tr = dfs(u, min(pushed, edges[id].cap - edges[id].
          flow));
      if (tr == 0) continue;
      edges[id].flow += tr;
      edges[id ^ 1].flow -= tr;
      return tr;
    }
    return 0;
  }

  ll flow() {
    ll f = 0;
    while (true) {
      fill(all(level), -1);
      level[s] = 0;
      q.push(s);
      if (!bfs()) break;
      fill(all(ptr), 0);
      while (ll pushed = dfs(s, flow_inf)) {
        f += pushed;
      }
    }
    return f;
  }

  vector<ii> min_cut() {
    //llamar flow();
    vector<ii> cut;
    for (auto &e : edges)
      if (level[e.v] != -1 && level[e.u] == -1 && e.cap > 0)
        cut.pb({e.v, e.u});
    return cut;
  }
};
//Dinic dd(n + 2, s, t);

//int nodos = n + 5;
//Dinic dd(nodos, nodos - 2, nodos - 1);
```

## 3.2 Hopcroft Karp

```
using ii = pair<int, int>;

struct mbm { // O(E * sqrt(V))
  int nl, nr, flow = 0;
  vector<vector<int>> g;
```

```
vector<int> dist, mfl, mfr;

mbm(int nl, int nr):
  nl(nl), nr(nr), g(nl), mfl(nl, -1),
  mfr(nr, -1), dist(nl) {}

void add(int u, int v) { g[u].push_back(v); }

void bfs() {
  queue<int> q;
  forn (u, nl)
    if (!~mfl[u]) q.push(u), dist[u] = 0;
    else dist[u] = -1;
  while (sz(q)) {
    int u = q.front();
    q.pop();
    for (auto &v : g[u])
      if (~mfr[v] && !~dist[mfr[v]]) {
        dist[mfr[v]] = dist[u] + 1;
        q.push(mfr[v]);
      }
  }
}

bool dfs(int u) {
  for (auto &v : g[u])
    if (!~mfr[v]) {
      mfl[u] = v, mfr[v] = u;
      return true;
    }
  for (auto &v : g[u])
    if (dist[mfr[v]] == dist[u] + 1 && dfs(mfr[v])) {
      mfl[u] = v, mfr[v] = u;
      return true;
    }
  return false;
}

int get_matching() {
  while (true) {
    bfs();
    int agt = 0;
    forn (u, nl)
      if (!~mfl[u]) agt += dfs(u);
    if (!agt) break;
    flow += agt;
  }
  return flow;
}
```

```
  pair<vector<int>, vector<int>> MVC() {
    vector<int> L, R;
    forn (u, nl)
      if (!~dist[u]) L.push_back(u);
      else if (~mfl[u]) R.push_back(mfl[u]);
    return {L, R};
  }

  vector<ii> get_edges() {
    vector<ii> ans;
    forn (u, nl)
      if (mfl[u] != -1)
        ans.pb({u, mfl[u]});
    return ans;
  }
};
```

## 3.3 $\text{Min}_c ost_m ax_f low$

```
#define INF 0x3f3f3f

template<typename T> struct mcmf {
  struct edge {
    int to, rev, flow, cap; // para, id da reversa, fluxo,
        capacidade
    bool res; // se eh reversa
    T cost; // custo da unidade de fluxo
    edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(
        false) {}
    edge(int to_, int rev_, int flow_, int cap_, T cost_,
        bool res_)
      : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_)
          , cost(cost_) {}
  };

  vector<vector<edge>> g;
  vi par_idx, par;
  T inf;
  vector<T> dist;

  mcmf(int n) : g(n), par_idx(n), par(n), inf(numeric_limits
      <T>::max()/3) {}

  void add(int u, int v, int w, T cost) { // de u pra v com
      cap w e custo cost
    edge a = edge(v, sz(g[v]), 0, w, cost, false);
    edge b = edge(u, sz(g[u]), 0, 0, -cost, true);

    g[u].pb(a);
    g[v].pb(b);
  }

  vector<T> spfa(int s) { // nao precisa se nao tiver custo
      negativo
    deque<int> q;
    vb is_inside(sz(g), 0);
    dist = vector<T>(sz(g), inf);

    dist[s] = 0;
    q.pb(s);
    is_inside[s] = true;

    while (!q.empty()) {
      int v = q.front();
      q.pop_front();
      is_inside[v] = false;

      for (int i = 0; i < sz(g[v]); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;

          if (is_inside[to]) continue;
          if (!q.empty() and dist[to] > dist[q.front()]) q.
              push_back(to);
          else q.push_front(to);
          is_inside[to] = true;
        }
      }
    }
    return dist;
  }
  bool dijkstra(int s, int t, vector<T>& pot) {
    priority_queue<pair<T, int>, vector<pair<T, int>>,
        greater<>> q;
    dist = vector<T>(sz(g), inf);
    dist[s] = 0;
    q.emplace(0, s);
    while (sz(q)) {
      auto [d, v] = q.top();
      q.pop();
      if (dist[v] < d) continue;
      for (int i = 0; i < sz(g[v]); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        cost += pot[v] - pot[to];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;
          q.emplace(dist[to], to);
          par_idx[to] = i, par[to] = v;
```

```
      }
    }
  }
  return dist[t] < inf;
}

pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
  vector<T> pot(sz(g), 0);
  pot = spfa(s); // mudar algoritmo de caminho minimo aqui

  int f = 0;
  T ret = 0;
  while (f < flow and dijkstra(s, t, pot)) {
    for (int i = 0; i < sz(g); i++)
      if (dist[i] < inf) pot[i] += dist[i];

    int mn_flow = flow - f, u = t;
    while (u != s) {
      mn_flow = min(mn_flow,
                    g[par[u]][par_idx[u]].cap - g[par[u]][
                         par_idx[u]].flow);
      u = par[u];
    }

    ret += pot[t] * mn_flow;

    u = t;
    while (u != s) {
      g[par[u]][par_idx[u]].flow += mn_flow;
      g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
      u = par[u];
    }

    f += mn_flow;
  }

  return make_pair(f, ret);
}

// Opcional: retorna as arestas originais por onde passa
     flow = cap
vector<pair<int,int>> recover() {
  vector<pair<int,int>> used;
  for (int i = 0; i < sz(g); i++) for (edge e : g[i])
      if(e.flow == e.cap && !e.res) used.push_back({i, e.to
           });
  return used;
}
};
```

```
//nodos = (n + m) + 10;
// s = n - 1;
// t = n - 2;
```

# 4 Geometry

## 4.1 line

```


//cordenada (x, y) de cada punto
vi X;
vi Y;
//valida si los 3 puntos estan en la misma linea
bool sameLine(int p1, int p2, int p3) {
  ll v1 = X[p1] * Y[p2] + X[p2] * Y[p3] + X[p3] * Y[p1];
  ll v2 = X[p2] * Y[p1] + X[p3] * Y[p2] + X[p1] * Y[p3];
  return v1 == v2;
}
```

# 5 Graph

## 5.1 2 edge connected comp

```

typedef pair<int, int> pii;

//Given a undirected graph
//you can remove exactly
//one edge from the graph.
//Your task is to minimize
//the number of pairs of vertices (u, v)
//between which there exists a path in this graph

int n, m;
const int MAXN = 1e5 + 5;

int timer, tagTree;
vector<pii> g[MAXN];
//2-edge-connected component tree
//(Bridge Tree)
vi tree[MAXN];
vb vis, isBridge;
vi tin, low;
vi id; // u pertenece a la comp id[u]
vector<array<int, 3>> edges;

//optional
vi cntNodos; //nodos de la componente i
vi tam; // subtree size

//Tarjan
void dfs1(int u, int p){
  tin[u] = low[u] = ++timer;
  vis[u] = true;
  for(auto &[to, id]: g[u]) {
    if(to == p) continue;
    if(vis[to]) {
      low[u] = min(low[u], tin[to]);
    } else {
      dfs1(to, u);
      low[u] = min(low[u], low[to]);
      if(low[to] > tin[u]) {
        isBridge[id] = true;
      }
    }
  }
}

//assing id
void dfs2(int u){
  vis[u] = 1;
  id[u] = tagTree;
  for(auto &[to, id]: g[u]){
    //skip bridges
    if(isBridge[id]) continue;
    if(!vis[to]) dfs2(to);
  }
}

//build edge tree
void build(){
  timer = 0;
  tagTree = 0;
  dfs1(1, 0);
  fill(all(vis), 0);
  forab(i, 1, n + 1){
    if(!vis[i]){
      tagTree++;
      dfs2(i);
    }
  }
  int bridges = 0;
  forab(i, 1, m + 1){
    if(isBridge[i]){
```

```
      auto [u, v, idx] = edges[i];
      tree[id[u]].pb(id[v]);
      tree[id[v]].pb(id[u]);
    }
  }
}

//do something in bridge tree
// subtree size of each comp
void dfs3(int u, int p){
  tam[u] = cntNodos[u];
  for(int v: tree[u]){
    if(v == p) continue;
    dfs3(v, u);
    tam[u] += tam[v];
  }
}

int main() {
  int t; cin >> t;
  while(t--){
    cin >> n >> m;
    vis = vb(n + 1, 0);
    isBridge = vb(m + 1, 0);

    tin = vi(n + 1);
    low = vi(n + 1);
    id = vi(n + 1);
    cntNodos = vi(n + 1, 0);
    tam = vi(n + 1, 0);

    edges = vector<array<int, 3>>(m + 1);
    int a, b;
    forab(id, 1, m + 1){
      cin >> a >> b;
      g[a].pb({b, id});
      g[b].pb({a, id});
      edges[id] = {a, b, id};
    }

    build();

    forab(u, 1, n + 1){
      cntNodos[id[u]]++;
    }
    dfs3(1, 0); //subtree size
    ll mx_remove = 0;
    forab(v, 1, tagTree + 1){
      mx_remove = max(mx_remove, (ll) (n - tam[v]) * tam[v]);
    }
    ll cant = 1LL * n * (n - 1LL) / 2;
    cout << (cant - mx_remove) << ln;
    if(t){
      forab(i, 1, n + 1) {
        g[i].clear();
        tree[i].clear();
      }
    }
  }
  return 0;
}

//minimun number of edges
//such that there are no
//bridges in the new graph
//ans = ceil(leaf / 2)

//if we can remove only once
//edge, we can reduce the number
//of bridges to
//(bridges - tree diameter)
```

## 5.2 Belman Ford

```
int n, m;
const int MAXN = 2505;
const int MAXM = 5005;
const ll oo = 1e18 + 5;

array<ll, 3> edges[MAXM];
int par[MAXN];
ll dis[MAXN];

int main() {
  cin >> n >> m;
  forn(i, m){
    int a, b, w;
    cin >> a >> b >> w;
    edges[i] = {a, b, -w};
  }

  //negative cycle
  vb change(n + 1, 0);
  forab(i, 1, n + 1) dis[i] = oo;
  dis[1] = 0;
  int x;
  forn(i, n){
    x = -1;
    for(auto [a, b, w]: edges){
      if(dis[a] < oo){
        if(dis[b] > dis[a] + w){
          if(i == n - 1) change[b] = 1;
          dis[b] = max(-oo, dis[a] + w);
          par[b] = a;
          x = b;
        }
      }
    }
  }
  bool cycle = (x != -1);
  bool valid_cycle = 0;

  if(cycle){
    /*
    int y = x;
    forn(i, n) y = par[y];
    vi path;
    for(int cur = y;; cur = par[cur]){
      path.pb(cur);
      if(cur == y && sz(path) > 1) break;
    }
    */
    dfs1(1);//normal graph
    dfs2(n);//reverse graph

    forab(i, 1, n + 1){
      if(change[i]){
        if(vis1[i] && vis1[n]){
          if(vis2[i] && vis2[1]){
            valid_cycle = 1;
            break;
          }
        }
      }
    }
  }

  if(valid_cycle){
    puts("-1");
  }else{
    printf("%lld\n", -dis[n]);
  }
  return 0;
}

  //shortest path
  forab(i, 1, n + 1) dis[i] = oo;
  int source = 1;
```

```
  int dest = n;
  par[source] = -1;
  d[source] = 0;
  forn (i, n - 1){
    for (auto [a, b, w] : edges){
      if (dis[a] < oo){
        dis[b] = min(dis[b], dis[a] + w);
        par[b] = a;
      }
    }
  }
  if (dis[dest] == oo){
    //no path
  }else{
    vi path;
    for (int cur = dest; cur != -1; cur = par[cur])
      path.pb(cur);
    reverse(all(path));
  }
```

## 5.3 Bipartite check

```
const int N = 2e5+5;
vi g[N];
vi color;
bool bipartite = 1;

void dfs(int u, int c) {
   if(~color[u]) {
      if(color[u] ^ c) {
         bipartite = 0;
      }
      return;
   }
   color[u] = c;
   for(int next: g[u]) {
      dfs(next, c^1);
   }
}

int main() {
   int n, m;
   cin >> n >> m;
   color = vi(n + 1, -1);
   int a, b;
   forn(i, m) {
      cin >> a >> b;
      g[a].pb(b);
      g[b].pb(a);
   }
   for(int i = 1; i < n + 1 && bipartite; i++) {
      if(color[i] < 0) dfs(i, 1);
   }
   if(bipartite) {
      //color of each node
      forab(i, 1, n + 1) {
         cout << color[i] + 1 <<" ";
      }
   } else {
      cout << "No bipartite" << ln;
   }
   return 0;
}

/*validar distancias
entre dos nodos A y B
si es par o impar
if(!bipartite){
 dist pares e impares;
}else{
 if(color[a] == color[b]){
  dist par
 }
 if(color[a] != color[b]){
  dist impar
 }
}*/
```

## 5.4 Bridges and articulation Points

```
typedef pair<int, int> ii;

// Tarjan's Algorithm
// Finding bridges in a graph in O(N + M)
// Undirected graph.

const int MAXN = (int) 1e5 + 5; // cant nodos
vi g[MAXN];
vi tin, low;
int timer = 0;
vector<ii> bridges;
vb is_articulation, vis;
int n, m;

void dfs(int v, int p = -1){
  vis[v] = true;
  tin[v] = low[v] = timer++;
  int children = 0;
  for (int u : g[v]) {
    if (u == p) continue;
    if (vis[u]) {
      low[v] = min(low[v], tin[u]);
    } else {
      dfs(u, v);
      low[v] = min(low[v], low[u]);
      if (low[u] >= tin[v] && p != -1)
        is_articulation[v] = true;
      ++children;
      if (low[u] > tin[v])
        bridges.pb({min(u, v), max(u, v)});
    }
  }
  if (p == -1 && children > 1)
    is_articulation[v] = true;
}

int main(){
  cin >> n >> m;
  tin = vi(n + 1, -1);
  low = vi(n + 1, -1);
  vis = vb(n + 1, 0);
  is_articulation = vb(n + 1, 0);

  int a, b;
  forn(i, m) {
    cin >> a >> b;
    g[a].pb(b); g[b].pb(a);
  }
  // find bridges and articulation points
  forab (i, 1, n + 1) {
    if (!vis[i]) dfs(i);
  }
  // print articulation points
  forab (i, 1, n + 1) {
    if (is_articulation[i]) {
      cout << i << ln;
    }
  }
  // print bridges
  for (auto [u, v] : bridges) {
    cout << u << " " << v;
  }
  return 0;
}
```

## 5.5 Detect cycle

```cpp
//directed graph
int n, m;
const int MAXN = 1e5 + 5;
vi g[MAXN];
vi par(MAXN, -1);
vi color(MAXN, 0);
int inicio = -1, fin = -1;
vi cycle;

bool dfs(int u){
  color[u] = 1;
  for(int v: g[u]){
    if(color[v] == 0){
      par[v] = u;
      if(dfs(v)) return 1;
    }else if(color[v] == 1){
      if(inicio != -1 && fin != -1) return 1;
      inicio = u;
      fin = v;
      vi aux;
      int cur = u;
      aux.pb(cur);
      while(cur != v){
        cur = par[cur];
        aux.pb(cur);
      }
      aux.pb(u);
      if(sz(aux) >= 2){
        reverse(all(aux));
        cycle = aux;
      }
      return 1;
    }
  }
  color[u] = 2;
  return 0;
}

int main() {
  cin >> n >> m;
  int a, b;
  forn(i, m){
    cin >> a >> b;
    g[a].pb(b);
  }
  forab(i, 1, n + 1){
    if(color[i] == 0 && dfs(i))break;
    if(sz(cycle) > 0) break;
  }
  if(sz(cycle) == 0){
    puts("IMPOSSIBLE");
  }else{
    printf("%d\n", sz(cycle));
    for(int i : cycle){
      printf("%d ", i);
    }
    puts("");
  }
  return 0;
}

//undirected graph
const int MAXN = 1e5 + 5;
vi g[MAXN];
vb vis;
vi parent;
int n, m;
int cycle_start = -1;
int cycle_end = -1;

bool dfs(int node, int par) {
  vis[node] = 1;
  parent[node] = par;
  for(int next: g[node]) {
    if(next == par) continue;
    if(vis[next]) {
      cycle_start = next;
      cycle_end = node;
      return 1;
    }
    if(!vis[next] && dfs(next, node)) return 1;
  }
  return false;
}
int main() {
  cin >> n >> m;
  vis = vb(n + 1, 0);
  parent = vi(n + 1);
  forn(i, m) {
    int a, b;
    cin >> a >> b;
    g[a].pb(b);
    g[b].pb(a);
  }
  forab(i, 1, n + 1) {
    if(!vis[i] && dfs(i, -1)) break;
  }
  if(cycle_start == -1) //no hay ciclo
    else //hay ciclo
      //se puede recorrer el ciclo
      //usando el array de padres
      return 0;
}
```

## 5.6 Dijkstra

```cpp
struct edge{
  int u;
  ll w;
  bool operator < (const edge &x) const {
    return x.w < w;
  }
};

int n, m;
const int MAXN = 2e5 + 5;
const ll oo = (1LL << 62);
vector<edge> g[MAXN];
vi par(MAXN);

vll dijkstra(int s) {
  priority_queue<edge> pq;
  vll dis(n + 1, oo);
  pq.push(edge{s, 0});
  dis[s] = 0;
  par[s] = -1;
  while (sz(pq)) {
    auto [u, cur_dis] = pq.top();
    pq.pop();
    ll min_dis = dis[u];
    if(cur_dis != min_dis) continue;
    for (auto [v, w] : g[u]) {
      if (dis[u] + w < dis[v]) {
        dis[v] = dis[u] + w;
        par[v] = u;
        pq.push(edge{v, dis[v]});
      }
    }
  }
  return dis;
}

int main() {
  cin >> n >> m;
  forn(i, m){
    int a, b, w;
    cin >> a >> b >> w;
```

```
    g[a].pb({b, w});
    g[b].pb({a, w});
  }
  vll dis = dijkstra(1);
  if(dis[n] != oo){
    vi path;
    for (int i = n; i != -1; i = par[i]) {
      path.pb(i);
    }
    rforn (i, sz(path)) {
      cout << path[i] << " ";
    }
    cout << ln;
  } else {
    cout << "-1" << ln;
  }
  return 0;
}

//validar si una arista
//esta en el camino mas corto
//desde (a, b)
vll da = dijkstra(a);
ll minDis = da[b];
vll db = dijkstra(b);
for (edge e : aristas) {
  //edge is on shortest path
  if(disSource[e.u] + e.w + disDest[e.v] == minDis) {
    continue;
  }
  //edge is on shortest path
  if(disSource[e.v] + e.w + disDest[e.u] == minDis) {
    continue;
  }
}
```

## 5.7 Flattenig Tree

```
//Warning
//STree recursive [l, r] from 0
//STree iterative [l, r + 1) from 0
template<typename T>
struct STree {
  int n;
  vector<T> st, lazy;
  T neutro = T(0ll);

  STree(int m) {
    n = m;
    st.resize(n * 4);
    lazy.resize(n * 4);
  }

  STree(vector<T> &a) {
    n = sz(a);
    st.resize(n * 4);
    lazy.resize(n * 4);
    build(1, 0, n - 1, a);
  }

  T oper(T a, T b) {
    return a | b;
  }

  void build(int v, int tl, int tr, vector<T> &a) {
    if(tl == tr) {
      st[v] = 1ll << a[tl];
      return;
    }
    int tm = (tl + tr) / 2;
    build(v * 2, tl, tm, a);
    build(v * 2 + 1, tm + 1, tr, a);
    st[v] = oper(st[v * 2], st[v * 2 + 1]);
  }

  void push(int v, int tl, int tr) {
    if (!lazy[v]) return;
    st[v] = 1ll << lazy[v];
    if (tl != tr) {
      lazy[v * 2] = lazy[v * 2 + 1] = lazy[v];
    }
    lazy[v] = 0;
  }

  void upd(int v, int tl, int tr, int l, int r, T val) {
    push(v, tl, tr);
    if(tr < l || tl > r) return;
    if (tl >= l && tr <= r) {
      lazy[v] = val;
      push(v, tl, tr);
      return;
    }
    int tm = (tl + tr) / 2;
    upd(v * 2, tl, tm, l, r, val);
    upd(v * 2 + 1, tm + 1, tr, l, r, val);
    st[v] = oper(st[v * 2], st[v * 2 + 1]);
  }

  T query(int v, int tl, int tr, int l, int r) {
    push(v, tl, tr);
    if(tl > r || tr < l) return neutro;
    if (l <= tl && tr <= r) return st[v];
    int tm = (tl + tr) / 2;
    return oper(query(v * 2, tl, tm, l, r), query(v * 2 + 1,
        tm + 1, tr, l, r));
  }

  void upd(int l, int r, T val) {
    upd(1, 0, n - 1, l, r, val);
  }
  T query(int l, int r) {
    return query(1, 0, n - 1, l, r);
  }
};

const int MAXN = 4e5 +5;
vi g[MAXN];
int tin[MAXN];
int tout[MAXN];
vll aplanado;
int timer = 0;

int n, q;
vi color;
void dfs(int node, int par) {
  tin[node] = timer;
  //se usa el color como el aplanado
  //se puede cambiar segun sea
  aplanado[timer] = color[node];
  timer++;
  for(int next: g[node]) {
    if(next != par) {
      dfs(next, node);
    }
  }
  tout[node] = timer - 1;
}

//https://codeforces.com/contest/620/problem/E
int main() {
  cin >> n >> q;
  color = vi(n + 1);
  aplanado = vll(n + 1);
  forab(i, 1, n + 1) {
    cin >> color[i];
  }
  int a, b;
  forn(i, n - 1) {
    cin >> a >> b;
```

```
    g[a].pb(b);
    g[b].pb(a);
  }
  dfs(1, 0);
  int oper, v, c;
  STree<ll> st(aplanado);
  while(q--) {
    cin >> oper;
    if(oper == 1) {
      cin >> v >> c;
      // l = tin[v]
      // r = tout[v]
      // (l, r) = subArbol del nodo v
      //cambiamos el color del subarbol de v
      st.upd(tin[v], tout[v], c);
    } else {
      cin >> v;
      ll ans = st.query(tin[v], tout[v]);
      //cantidad de colores diferentes en
      //el subarbol de v
      cout << __builtin_popcountll(ans) << ln;
    }
  }
  return 0;
}
```

## 5.8 Floyd Warshall

```
const ll oo = 1e18 / 10;
int main() {
  int n, m, q;
  cin >> n >> m >> q;
  vector<vll> g(n, vll(n));
  forn(i, n) {
    forab(j, i + 1, n) {
      g[i][j] = g[j][i] = oo;
    }
  }
  int a, b;
  ll w;
  forn(i, m) {
    cin >> a >> b >> w;
    a--;
    b--;
    g[a][b] = g[b][a] = min(g[a][b], w);
  }
  //Floyd Warshall O(n^3)
  forn(k, n) {
    forn (i, n) {
      forn (j, n) {
        g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
      }
    }
  }
  while(q--) {
    cin >> a >> b;
    a--;
    b--;
    cout << (g[a][b] == oo ? -1: g[a][b]) <<ln;
  }
  return 0;
}
```

## 5.9 Kosaraju

```
const int MAXN = 1e5 + 5;
int n, m;
vi g[MAXN];
vi rg[MAXN];
vb vis(MAXN);
vi id(MAXN);
int tagSCC;
vector<vi> SCC;
vi curComp;

void dfs1(int u, stack<int> &st){
  vis[u] = 1;
  for(int v: g[u]){
    if(!vis[v]) dfs1(v, st);
  }
  st.push(u);
}
void dfs2(int u){
  vis[u] = 1;
  id[u] = tagSCC;
  curComp.pb(u);
  //u esta en la SCC id[u]
  for(int v: rg[u]){
    if(!vis[v])dfs2(v);
  }
}

int main(){
  cin >> n >> m;
  int a, b;
  forn(i, m){
    cin >> a >> b;
    g[a].pb(b);
    rg[b].pb(a);
  }
  stack<int> st;
  forab(i, 1, n + 1){
    if(!vis[i])dfs1(i, st);
  }
  vis = vb(n + 1, 0);
  while(!st.empty()){
    int u = st.top(); st.pop();
    if(vis[u]) continue;
    tagSCC++;
    curComp.clear();
    dfs2(u);
    SCC.pb(curComp);
  }
  //build DAG
  const int MAXDAG = tagSCC + 1;
  vi dag[MAXDAG];
  forab(u, 1, n + 1){
    for(int v: g[u]){
      if(id[u] == id[v]) continue;
      dag[id[u]].pb(id[v]);
    }
  }
  //print SCC
  int cp = 1;
  for(vi i: SCC){
    printf("SCC #%d: ", cp++);
    for(int u: i){
      printf("%d ", u);
    }
    printf("\n");
  }
  return 0;
}
```

## 5.10 LCA binary lifting

```
int n, m;
const int MAXN = 3e5 + 5;
const int LOG = 21; //calcular

vi g[MAXN];
int tin[MAXN];
int tout[MAXN];
int deep[MAXN];
```

```
int timer = 1;
int up[MAXN][LOG];
int max_up[MAXN][LOG];

map<pair<int, int>, int> weights;

void dfs(int u, int p) {
  tin[u] = timer++;
  deep[u] = deep[p] + 1;

  // initialize binary lifting arrays
  up[u][0] = p;
  max_up[u][0] = weights[ {p, u}];

  for (int v : g[u]) {
    if (v != p) {
      dfs(v, u);
    }
  }
  tout[u] = timer - 1;
}

bool is_ancestor(int x, int y) {
  return tin[x] <= tin[y] && tout[y] <= tout[x];
}

int lca(int x, int y) {
  if (is_ancestor(x, y)) return x;

  for (int i = LOG - 1; i >= 0; i--) {
    if (!is_ancestor(up[x][i], y)) {
      x = up[x][i];
    }
  }
  return up[x][0];
}

void build(int root) {
  dfs(root, root);
  forab(k, 1, LOG) {
    forab(u, 1, n + 1) {
      up[u][k] = up[up[u][k - 1]][k - 1];
      // take max of weights from left and right
      max_up[u][k] = max(max_up[u][k - 1], max_up[up[u][k -
          1]][k - 1]);
    }
  }
}
//get max weight edge from (u, lcaU)
int get_max_up(int u, int v) {
  if(deep[u] < deep[v]) swap(u, v);
  int df = abs(deep[v] - deep[u]);
  int res = 0;
  forn(i, LOG) {
    if(df & (1 << i)) {
      //up the lowest node
      res = max(res, max_up[u][i]);
      u = up[u][i];
    }
  }
  return res;
}

bool cmp(array<int, 4> &a, array<int, 4> &b) {
  return a[2] < b[2];
}

int main() {
  cin >> n >> m;
  vector<array<int, 4>> edges(m);
  forn(i, m) {
    int a, b, w;
    cin >> a >> b >> w;
    edges[i] = {a, b, w, i};
    weights[ {a, b}] = w;
    weights[ {b, a}] = w;
  }

  //Kruskal
  sort(all(edges), cmp);
  UnionFind uf(n + 1);
  ll mst = 0;
  for(auto [a, b, w, id]: edges) {
    if(!uf.isSame(a, b)) {
      uf.unite(a, b);
      mst += w;
      g[a].pb(b);
      g[b].pb(a);
    }
  }

  build(1);
  vll ans(m);

  for(auto [a, b, w, id]: edges) {
    int LCA = lca(a, b);
    int mx_a = get_max_up(a, LCA);
    int mx_b = get_max_up(b, LCA);

    ll res = mst - max(mx_a, mx_b) + w;
    ans[id] = res;
  }

  forn(i, m) cout << ans[i] << ln;
  return 0;
}
```

## 5.11 LCA

```
// Lowest Common Ancestor
// Se usa el Euler tour para encontrar un nodo que
//  est  entre u y v que tiene la altura mas baja
struct LCA {
  vi height, euler, first, st;//SegmentTree
  vector<vi> table; // Sparse Table
  vector<bool> vis;
  int n;
  bool SegTree;

  // LCA with segment tree -> option = 1
  // LCA with Sparse Table -> option = 2
  LCA(vector<vi>& g, int root, int option) {
    n = g.size();
    height.resize(n);
    first.resize(n);
    euler.reserve(n * 2);
    vis.assign(n, false);
    dfs(g, root, 0);
    if (option == 1) {
      SegTree = true;
      int m = euler.size();
      st.resize(m * 4);
      build(1, 0, m - 1);
    } else if(option == 2) {
      SegTree = false;
      build();
    }
  }

  void dfs(vector<vi>& g, int node, int h) {
    vis[node] = true;
    height[node] = h;
    first[node] = euler.size();
    euler.pb(node);
    for (auto to : g[node]) {
      if (!vis[to]) {
        dfs(g, to, h + 1);
        euler.pb(node);
      }
```

```
  }
}

void build(int node, int b, int e) {
  if (b == e) {
    st[node] = euler[b];
  } else {
    int mid = (b + e) / 2;
    build(node << 1, b, mid);
    build(node << 1 | 1, mid + 1, e);
    int l = st[node << 1], r = st[node << 1 | 1];
    st[node] = (height[l] < height[r]) ? l : r;
  }
}

int query(int node, int b, int e, int L, int R) {
  if (b > R || e < L)
    return -1;
  if (b >= L && e <= R)
    return st[node];
  int mid = (b + e) >> 1;

  int left = query(node << 1, b, mid, L, R);
  int right = query(node << 1 | 1, mid + 1, e, L, R);
  if (left == -1)
    return right;
  if (right == -1)
    return left;
  return height[left] < height[right] ? left : right;
}

int lca(int u, int v) {
  int left = first[u], right = first[v];
  if (left > right)
    swap(left, right);
  if (SegTree) {
    return query(1, 0, euler.size() - 1, left, right);
  } else {
    return query(left, right);
  }
}

void build() {
  int N = euler.size();
  int K = int(log2(N)) + 1;
  table.assign(N + 1, vi(K));
  forn(i, N) {
    table[i][0] = euler[i];
  }
  forn(j, K - 1) {
    for (int i = 0; i + (1 << (j + 1)) <= N; ++i) {
      int a = table[i][j];
      int b = table[i + (1 << j)][j];
      table[i][j + 1] = height[a] < height[b] ? a : b;
    }
  }
}

int query(int l, int r) {
  int k = 31 - __builtin_clz(r - l + 1);
  int a = table[l][k];
  int b = table[r - (1 << k) + 1][k];
  return height[a] < height[b] ? a : b;
}
};

int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  int n, q;
  cin >> n >> q;
  vector<vi> g(n + 1, vi());
  int a, b;
  for (int i = 0; i < n - 1; i++) {
    cin >> a >> b;
    g[a].pb(b);
    g[b].pb(a);
  }
  LCA lca(g, 1, 2);
  while (q--) {
    cin >> a >> b;
    int LCA = lca.lca(a, b);
    cout << lca.height[a] + lca.height[b] - (2 * lca.height[
        LCA]) << ln;
  }

  return 0;
}
```

## 5.12 Tarjan

```
int n, m;
const int MAXN = 1e5 + 5;
vi g[MAXN];

struct Tarjan {
  vi low, num, comp;
  stack<int> st;
  int n, scc, cont;
  const int oo = int(1e9);

  Tarjan(int n) {
    this->n = n;
    low.resize(n);
    num.assign(n, -1);
    comp.resize(n);
    scc = cont = 0;
  }

  void dfs(int v) {
    low[v] = num[v] = cont++;
    st.push(v);
    for (int &u : g[v]) {
      if (num[u] == -1) dfs(u);
      low[v] = min(low[v], low[u]);
    }
    if (low[v] == num[v]) {
      int u;
      do {
        u = st.top(); st.pop();
        low[u] = oo;
        comp[u] = scc;
      } while (u != v);
      scc++;
    }
  };

  vector<vi> getComp(){
    forab (i, 1, n)
      if (num[i] == -1) dfs(i);
    vector<vi> cc(scc);
    forab(i, 1, n){
       cc[comp[i]].pb(i);
    }
    return cc;
  }
};

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    int a, b;
    forn(i, m){
      cin >> a >> b;
      g[a].pb(b);
```

```
    }
    Tarjan t(n + 1);
    vector<vi> components = t.getComp();
    for(vi cc: components){
      for(int u: cc){
          cout << u << " ";
      }
      cout << ln;
    }
    return 0;
}
```

## 5.13 Tree Diameter

```
const int MAXN = 3e5 + 5;
vi g[MAXN];
vb vis(MAXN, 0);
vi dis(MAXN, 0);
int n, m;
int nodoLejano, maxDis;
void bfs(int src) {
  vis = vb(n + 1, 0);
  dis = vi(n + 1);
  queue<int> q;
  vis[src] = 1;
  nodoLejano = src;
  q.push(src);
  while(!q.empty()) {
    int u = q.front();
    q.pop();
    if(dis[u] > maxDis) {
      nodoLejano = u;
      maxDis = dis[u];
    }
    for(int v: g[u]) {
      if(!vis[v]) {
        vis[v] = 1;
        dis[v] = dis[u] + dis[v] + 1;
        q.push(v);
      }
    }
  }
}
int main() {
  cin >> n;
  int a, b;
  forn(i, n - 1) {
    cin >> a >> b;
    g[a].pb(b);
    g[b].pb(a);
  }
  maxDis = 0;
  bfs(1);
  bfs(nodoLejano);
  cout << maxDis << ln;
  return 0;
}
```

## 5.14 Tree distances

```
const int MAXN = 2e5 + 5;
int n;
ll down[MAXN], up[MAXN];
vector<pair<int, ll>> tree[MAXN];

void dfs_down(int u, int parent) {
  down[u] = 0;
  for (auto [v, w] : tree[u]) {
    if (v != parent) {
      dfs_down(v, u);
      down[u] = max(down[u], down[v] + w);
    }
  }
}

void dfs_up(int u, int parent) {
  int max1 = -1, max2 = -1; // Las dos mayores distancias
       hacia abajo desde u
  // Encuentra las dos mayores distancias hacia abajo
  for (auto [v, w] : tree[u]) {
    if (v != parent) {
      int total_dist = down[v] + w;
      if (total_dist > max1) {
        max2 = max1;
        max1 = total_dist;
      } else if (total_dist > max2) {
        max2 = total_dist;
      }
    }
  }

  //up[v] para cada hijo v
  for (auto [v, w] : tree[u]) {
    if (v != parent) {
      if (down[v] + w == max1) {
        up[v] = max(up[u] + w, max2 + w);
      } else {
        up[v] = max(up[u] + w, max1 + w);
      }
      dfs_up(v, u);
    }
  }
}

int main() {
  //build tree

  dfs_down(1, 1);
  up[1] = 0;
  dfs_up(1, 1);

  forab(i, 1, n + 1){
    ll best = max(up[i], down[i]);
    if(i == n) printf("%lld\n", best);
    else printf("%lld ", best);
  }
  return 0;
}
```

## 5.15 Two Sat

```
//si A y  A  estan en la misma SCC,
//entonces no hay solucion.
struct sat2 {
  int n;
  vector<vector<vector<int>>> g;
  vector<bool> vis, val;
  vector<int> comp;
  stack<int> st;

  sat2(int n) : n(n), g(2, vector<vector<int>>(2*n)), vis(2*
       n), val(2*n), comp(2*n) { }

  int neg(int x) {
    return 2*n-x-1;
  }
  void make_true(int u) {
    add_edge(neg(u), u);
  }
  void make_false(int u) {
    make_true(neg(u));
  }
  void add_or(int u, int v) {
    implication(neg(u), v);
  }
```

```
  void diff(int u, int v) {
    eq(u, neg(v));
  }
  void eq(int u, int v) {
    implication(u, v);
    implication(v, u);
  }
  void implication(int u, int v) {
    add_edge(u, v);
    add_edge(neg(v), neg(u));
  }

  void add_edge(int u, int v) {
    g[0][u].pb(v);
    g[1][v].pb(u);
  }

  void dfs(int id, int u, int t = 0) {
    vis[u] = true;
    for (auto &v : g[id][u])
      if (!vis[v]) dfs(id, v, t);
    if (id) comp[u] = t;
    else st.push(u);
  }

  void kosaraju() {
    for (int u = 0; u < n; u++) {
      if (!vis[u]) dfs(0, u);
      if (!vis[neg(u)]) dfs(0, neg(u));
    }
    vis.assign(2*n, false);
    int t = 0;
    while (!st.empty()) {
      int u = st.top();
      st.pop();
      if (!vis[u]) dfs(1, u, t++);
    }
  }

  bool check() {
    kosaraju();
    for (int i = 0; i < n; i++) {
      if (comp[i] == comp[neg(i)]) return false;
      val[i] = comp[i] > comp[neg(i)];
    }
    return true;
  }
};

int main() {
  int t, n, m;
  cin >> t;
  int caso = 1;
  while(t--) {
    cin >> n >> m;
    sat2 ts(n + 1);
    forn(i, m) {
      int a, b;
      cin >> a >> b;
      ts.implication(a, ts.neg(b));
      ts.implication(ts.neg(a), b);
    }
    bool ok = ts.check();
    if(ok)puts("si hay solucion");
    else puts("no hay solucion");
  }
  return 0;
}
```

## 5.16 bfs grid

```
//find shortest path from A to B
int n, m;
const int MAXN = 1002;
char grid[MAXN][MAXN];

//der, abajo, izq, arriba
vi dx = {0, 1, 0, -1};
vi dy = {1, 0, -1, 0};

bool valid(int i, int j) {
  if(i < 0 || i >= n || j < 0 || j >= m) {
    return 0;
  }
  if(grid[i][j] == '#') return 0;
  return 1;
}
string path;
bool bfs(int i, int j) {
  queue<array<int, 3>> q;
  vector<vb> vis(n, vb(m, 0));
  char br[n][m];
  q.push({i, j, 0});
  vis[i][j] = 1;

  while(sz(q)) {
    auto &[x, y, dis] = q.front(); q.pop();
    if(grid[x][y] == 'B') { //found
      while(true){ //build path
        path.pb(br[x][y]);
        if(path.back() == 'R') y--;
        if(path.back() == 'D') x--;
        if(path.back() == 'L') y++;
        if(path.back() == 'U') x++;
        if(x == i && y == j) break;
        if(!valid(x, y)) break;
      }
      return 1;
    }

    forn(k, 4) { // dir
      int nx = x + dx[k];
      int ny = y + dy[k];
      if(valid(nx, ny) && !vis[nx][ny]) {
        if(k == 0) br[nx][ny] = 'R';
        if(k == 1) br[nx][ny] = 'D';
        if(k == 2) br[nx][ny] = 'L';
        if(k == 3) br[nx][ny] = 'U';
        vis[nx][ny] = 1;
        q.push({nx, ny, dis + 1});
      }
    }
  }
  return 0;
}
```

## 5.17 functional graph

```
const int MAXN = 2e5 + 5;
vi rg[MAXN]; // reverse graph
vll valor(MAXN);
vb vis(MAXN, 0);
vi p(MAXN); //parent
vi in_cycle(MAXN, 0);

void mark(int u){
  if(vis[u]) return;
  vis[u] = 1;
  for(int v: rg[u]){
    mark(v);
  }
}

int cycle(int u){
  int x = p[u];
  int y = p[p[u]];
```

```
  while(x != y){
    x = p[x];
    y = p[p[y]];
  }
  //traverse the cycle
  ll min_cycle = valor[x];
  x = p[x];
  while(x != y){
    in_cycle[x] = 1;
    min_cycle = min(min_cycle, valor[x]);
    x = p[x];
  }
  mark(x);
  return min_cycle;
}

int main() {
  int n; cin >> n;
  forab(i, 1, n + 1){
    cin >> valor[i];
  }

  forab(i, 1, n + 1){
    cin >> p[i];
    rg[p[i]].pb(i);
  }

  ll ans = 0;
  forab(i, 1, n + 1){
    if(!vis[i]){
      ans += cycle(i);
    }
  }
  cout << ans << ln;
  return 0;
}
```

## 5.18 topo sort

```
vi topo_sort(vi &grado) {
  vi topo;
  queue<int> q;
  forab(i, 1, n + 1)
    if(grado[i] == 0) q.push(i);
  while(sz(q)){
    int u = q.front();
    q.pop();
    topo.pb(u);
    for(int v: g[u]){
      if(--grado[v] == 0){
        q.push(v);
      }
    }
  }
  if(sz(topo) == n) return topo;
  else return vi();
}
```

# 6 Math

## 6.1 Binary

```
//numeros desde [0, n] en binario
vi pre(int n) {
  vi bin;
  bin.pb(0);
  forab(i, 1, n + 1) {
    int j = 32;
    int x = i;
    string s;
    bool turn = 0;
    while (j--) {
      int bit = (x >> j) & 1;
      if (bit)
        turn = 1;
      if (turn) {
        s += (bit ? "1" : "0");
      }
    }
    int X = stoi(s);
    bin.pb(X);
  }
  return bin;
}

string binString(ll x) {
  int j = 32;//bits
  string s;
  bool turn = 0;
  while (j--) {
    int bit = (x >> j) & 1;
    if (bit) turn = 1;
    if (turn) s += (bit ? "1" : "0");
  }
  if(sz(s) == 0) s+="0";
  return s;
}
```

## 6.2 Divisors of a number

```
//todos los divisores n en O(sqrt(n))
vll divisors(ll n){
  vll div;
  for (ll i = 1; i * i <= n; i++) {
    if (n % i == 0) {
      div.pb(i);
      if ((n/i) != i)
        div.pb(n / i);
    }
  }
  return div;
}
```

## 6.3 ExtendedEuclid

```
ll euclid(ll a, ll b, ll &x, ll &y) {
}
//dado un N decir si se puede formar
//por la multiplicacion de numeros
//en su representacion binaria
//N = bin * bin
//121 = 11 * 11
int main() {
  int t;
  cin >> t;
  vi bin = pre(32);
  int MAX = 1e5 + 10;
  vb sol(MAX, 0);
  sol[1] = 1;
  forab(i, 1, MAX) {
    if(sol[i]) {
      for(int x : bin) {
        if((ll) i * x < MAX) sol[i*x] = 1;
      }
    }
  }
  while(t--) {
    ll n;
    cin >> n;
    cout << (sol[n] ? "YES" :"NO" ) <<ln;
  }
  return 0;
}
```

```
  if(b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  ll xi, yi;
  ll g = euclid(b, a % b, xi, yi);
  x = yi;
  y = xi - yi * (a / b);
  return g;
}

int main() {
  //hallar Ax + By + C = 0
  ll a, b, c;
  cin >> a >> b >> c;
  ll x, y;
  ll g = euclid(a, b, x, y);
  if(c % g) cout << "-1" << ln;
  else cout << (-(c / g) * x) << " " << (-(c / g) * y) << ln
       ;
  return 0;
}
```

## 6.4 Factorials

```
// define MOD
ll MOD = 1e9 + 7;

//use binpow() method here

const int MAXN = 1e6 + 1;
ll fact[MAXN];
ll ifact[MAXN];

void cal_factorials(int n){
  fact[0] = fact[1] = 1;
  for (int i = 2; i <= n; i++) {
    fact[i] = (fact[i - 1] * i) % MOD;
  }

  //fact_inv O(n)
  ifact[n] = binpow(fact[n], MOD - 2);
  for (int i = n - 1; i >= 0; i--) {
    ifact[i] = ((i + 1) * ifact[i + 1]) % MOD;
  }
}

ll binomial_coefficient(int n, int k) {
  return fact[n] * ifact[k] % MOD * ifact[n - k] % MOD;
}

int main() {
  int t; cin >> t;
  int caso = 1;
  cal_factorials(MAXN);
  while(t--){
    int n, k;
    cin >> n >> k;
    ll ans = binomial_coefficient(n, k);
    printf("Case %d: %lld\n", caso++, ans);
  }
  return 0;
}
```

## 6.5 Factorization sieve

```
const int MAX = 10e6;
int primediv[MAX]; // 10^6
vll primes;

void sieve(){
  forn(i, MAX) primediv[i] = i;
  int root = sqrt(MAX) + 1;
  forab(i, 2, MAX){
    if (primediv[i] != i) continue;
    primes.pb(i);
    if (i > root) continue;
    form(j, i * i, MAX, i) primediv[j] = i;
  }
}

map<ll, int> factorize(ll n){ // n <= 10^12
  map<ll, int> factors;
  for (int i = 0; i < sz(primes) && n >= MAX; ++i) {
    while (n % primes[i] == 0) {
      factors[primes[i]]++;
      n /= primes[i];
    }
  }
  if (n >= MAX) {
    factors[n]++;
    return factors;
  }
  while (n > 1) {
    factors[primediv[n]]++;
    n /= primediv[n];
  }
  return factors;
}
```

## 6.6 FibLog

```
typedef pair<ll, ll> pll;
//first = n-esimo termino
//second = siguiente termino
pll fib_log(ll n, ll mod){
  if (n == 0) return {0, 1};
  auto [a, b] = fib_log(n >> 1, mod);
  ll c = a * (2*b - a + mod) % mod;
  ll d = ((a*a % mod) + (b*b % mod)) % mod;
  if (n & 1) return {d, (c + d) % mod};
  else return {c, d};
}
```

## 6.7 Linear Sieve

```
//O(n)
const int MAXN = 10000000;
int lp[MAXN + 1]; //minimo factor primo
vi pr;

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

## 6.8 Math utils

```
ll MOD = 1e9 + 7;
```

```
ll inv_mod(ll a){
  //binpow con MOD igual
  return binpow(a, MOD - 2);
}

ll suma(ll a, ll b){
  return ((a % MOD) + (b % MOD)) % MOD;
}

ll resta(ll a, ll b){
  return ((a % MOD) - (b % MOD) + MOD) % MOD;
}

ll multi(ll a, ll b){
  return ((a % MOD) * (b % MOD)) % MOD;
}

ll divi(ll a, ll b){
  return multi(a, inv_mod(b));
}

//GCD
ll gcd(ll a, ll b) {
  return b == 0 ? (a < 0 ? -a : a) : gcd(b, a % b);
}
//LCM
ll lcm(ll a, ll b) {
  ll lcm = (a / gcd(a, b)) * b;
  return lcm > 0 ? lcm : -lcm;
}
//lcm(A, B, C, ....) =
//p1 max(a1, b1, c1, ...) *
//p2 max(a2, b2, c2, ...) *
//p3 max(a3, b3, c3, ...) * ... ...* pk max(ak, bk, ck, ...)
//pi = prime, ai = exponent

//cuantas veces B puede restar a A
x = floor(A/B + 0.0)

//distancia de X a Y dando saltos de tamao D
int dis(int x, int y, int d) {
  return (abs(x - y) + (d - 1)) / d;
}
```

## 6.9 Miller Rabin

```
bool probably_prime(ll n, ll a, ll d, int s){
    ll x = binpow(a, d, n);
    if(x == 1 || x+1 == n) return true;
    forn(r, s){
        x = mulmod(x,x,n);
        if(x == 1) return false;
        if(x+1 == n) return true;
    }
    return false;
}

bool miller_rabin(ll n){//check (n is prime)?
    if(n < 2) return false;
    const int a[] = {2,3,5,7,11,13,17,19,23};
    int s = -1;
    ll d = n-1;
    while(!d&1) d >>= 1, s++;

    forn(i, 9){
        if(n == a[i]) return true;
        if(!probably_prime(n, a[i], d, s))
            return false;
    }
    return true;
}
```

## 6.10 Modular Inverse

```
int inv[MAXN];

void modular_inverse_range(int m){
    inv[0] = 0; inv[1] = 1;
    forab(i, 2, MAXN)
        inv[i] = (-(m/i)*inv[m%i] + m) % m;
}

int modular_inverse_binpow(int a, int m){
    return binpow(a, phi(m)-1, m);
}

int modular_inverse_extEuclid(int a, int m){
    int x, y;
    int g = extEuclid(a, m, x, y);
    if(g != 1)
        return -1;
    x = (x % m + m) % m;
    return x;
}

vi inversos(vi a, int m){
    vi inv;
    int v = 1;
    forn(i, sz(a)){
        inv.pb(v);
        v = (v * a[i]) % m;
    }

    int x, y;
    extEuclid(v, m, x, y);
    x = (x % m + m) % m;

    rforn(i, sz(a)){
        inv[i] = inv[i] * x;
        x = (x * a[i]) % m;
    }
    return inv;
}
```

## 6.11 Ocurrencia de multiplos

```
//hay una rana i que salta de a[i] en a[i]
//decir por cual hoja pasaron mas ranas
void ocurrencia_multiplos(vi a) { //O(nlogn)
  int n = sz(a);
  vi occ(n + 1);
  unordered_map<int, int> mp;
  forn(i, n) {
    if (a[i] <= n) {
      mp[a[i]]++;
    }
  }
  forab(i, 1, n + 1) {
    if (mp[i] == 0) continue;
    form(j, i, n + 1, i) {
      occ[j]+=mp[i];
    }
  }
  /* multiplo mas repetido y menos repetido */
  ll MAX = 0;
  ll MIN = 1e18;
  for (ll i : occ) {
    MAX = max(MAX, i);
    MIN = min(MIN, i);
  }
}
```

## 6.12 Phi Euler

```cpp
typedef unsigned long long ull;

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

  if(n > 1) result -= result / n;
  return result;
}

//number of integers
//which are coprime
//gcd(a, b) == 1

vi phi_1_to_n(int n) {
  vi phi;
  forab(i, 0, n + 1) phi.pb(i);

  for(int i = 2; i <= n; ++i) {
    if(phi[i] != i) continue;

    for(int j = i; j <= n; j += i)
      phi[j] -= phi[j] / i;
  }
  return phi;
}

vi phi_1_to_n2(int n) {
  vi phi;
  forab(i, 0, n + 1) phi.pb(i - 1);
  phi[1] = 1;

  for(int i = 2; i <= n; ++i) {
    for(int j = i * 2; j <= n; j += i)
      phi[j] -= phi[i];
  }
  return phi;
}


int main() {
  int t; cin >> t;
  int caso = 1;
  int MAXN = 5e6;
  vi phi = phi_1_to_n(MAXN);

  //sumatoria en rango de ph[i] * phi[i]
  vector<ull> pf(MAXN + 1, 0);
  forab(i, 2, MAXN + 1){
    pf[i] = pf[i - 1] + (ull(phi[i]) * ull(phi[i]));
  }

  while(t--){
    int a, b;
    cin >> a >> b;
    ull ans = pf[b] - pf[a - 1];
    printf("Case %d: %llu\n", caso++, ans);
  }
  return 0;
}
```

## 6.13 Segmented Sieve

```cpp
vector<int> prime; // sqrt(MAX R)

vector<ll> segmented_criba(ll l, ll r) {
  l = max(l, 2ll);
  vector<bool> vis(r - l + 1);
  for (int &pp : prime) {
    if ((ll) pp * pp > r) break;
    ll mn = (l + pp - 1) / pp;
    if (mn == 1ll) mn++;
    mn *= pp;
    for (ll i = mn; i <= r; i += pp) {
      vis[i - l] = true;
    }
  }
  vector<ll> ans;
  forn (i, sz(vis)) if (!vis[i]) ans.push_back(l + i);
  return ans;
}
```

## 6.14 Sieve

```cpp
const int MAX = int(1e6);
bitset<MAX + 5> bs;
vi prime;

void sieve() {
  bs.set();
  bs[0] = bs[1] = 0;
  form (i, 2, MAX + 1, 1) {
    if (bs[i]) {
      prime.pb(i);
      for (ll j = (ll) i * i; j <= MAX; j += i) {
        bs[j] = 0;
      }
    }
  }
}
```

## 6.15 Ternary Search

```cpp
double f(double x) {
  return x;
}

double ternary_search(double l, double r) {
  double eps = 1e-9;
  while (r - l > eps) {
    double m1 = l + (r - l) / 3;
    double m2 = r - (r - l) / 3;
    double f1 = f(m1);
    double f2 = f(m2);
    if (f1 < f2) l = m1;
    else r = m2;
  }
  return f(l);
}
```

## 6.16 binpow

```cpp
//a^b % MOD
ll binpow(ll a, ll b, ll m) {
  a %= m;
  ll res = 1;
  while (b > 0) {
    if (b & 1)
      res = (res * a) % m;
    a = (a * a) % m;
    b >>= 1;
  }
  return res;
}

ll binpow(ll a, ll b) {
  ll res = 1;
  while (b > 0) {
```

```
    if (b & 1)
      res = res * a;
    a = a * a;
    b >>= 1;
  }
  return res;
}
```

## 6.17 oper with string

```
//multiply big numbers
string multyply(string a, int b){
  int carry = 0;
  string ans = "";
  forn(i, sz(a)){
    carry = ((a[i] - '0') * b + carry);
    ans += carry % 10 + '0';
    carry /= 10;
  }
  while(carry != 0){
    ans += carry % 10 + '0';
    carry /= 10;
  }
  return ans;
}

string res = "1";
for(auto [p, ex] : fac){
  forn(i, ex){
    res = multyply(res, p);
  }
}
reverse(all(res));
```

# 7 Strings

## 7.1 AhoCorasick

```
//https://github.com/JJuanJr/Notebook-ICPC/blob/main/String/
    AhoCorasick.cpp
Complejidad: build(|patrones| * M)
Complejidad: query(|text|)


const int M = 26;

struct node{
 vector<int> child;
 int p = -1;
 char c = 0;
 int suffixLink = -1, endLink = -1;
 int id = -1;
 //int cnt = 0; Para contar patrones repetidos
 node(int p = -1, char c = 0) : p(p), c(c){
  child.resize(M, -1);
 }
};

struct AhoCorasick{
 vector<node> t;
 vector<int> lenghts;
 int wordCount = 0;

 AhoCorasick(){
  t.emplace_back();
 }

 void add(const string & s){
  int u = 0;
  for(char c : s){
   if(t[u].child[c-'a'] == -1){
    t[u].child[c-'a'] = t.size();
    t.emplace_back(u, c);
   }
   u = t[u].child[c-'a'];
  }
  t[u].id = wordCount++;
  lenghts.push_back(s.size());
 }

 void link(int u){
  if(u == 0){
   t[u].suffixLink = 0;
   t[u].endLink = 0;
   return;
  }
  if(t[u].p == 0){
   t[u].suffixLink = 0;
   if(t[u].id != -1) t[u].endLink = u;
   else t[u].endLink = t[t[u].suffixLink].endLink;
   return;
  }
  int v = t[t[u].p].suffixLink;
  char c = t[u].c;
  while(true){
   if(t[v].child[c-'a'] != -1){
    t[u].suffixLink = t[v].child[c-'a'];
    break;
   }
   if(v == 0){
    t[u].suffixLink = 0;
    break;
   }
   v = t[v].suffixLink;
  }
  if(t[u].id != -1) t[u].endLink = u;
  else t[u].endLink = t[t[u].suffixLink].endLink;
 }

 void build(){
  queue<int> Q;
  Q.push(0);
  while(!Q.empty()){
   int u = Q.front(); Q.pop();
   link(u);
   for(int v = 0; v < M; ++v)
    if(t[u].child[v] != -1)
     Q.push(t[u].child[v]);
  }
 }

 int match(const string & text){
  int u = 0;
  int ans = 0;
  for(int j = 0; j < text.size(); ++j){
   int i = text[j] - 'a';
   while(true){
    if(t[u].child[i] != -1){
     u = t[u].child[i];
     break;
    }
    if(u == 0) break;
    u = t[u].suffixLink;
   }
   int v = u;
   while(true){
    v = t[v].endLink;
    if(v == 0) break;
    ++ans;
    int idx = j + 1 - lenghts[t[v].id];
    // cout << "Found word #" << t[v].id << " at position "
         << idx << "\n";
    v = t[v].suffixLink;
   }
  }
  return ans;
```

```
 }
};
```

## 7.2 KMP

```
vector<int> prefix_function(string &s) {
    int n = s.size();
    vector<int> pf(n);
    pf[0] = 0;
    for (int i = 1, j = 0; i < n; i++) {
        while (j && s[i] != s[j]) j = pf[j-1];
        if (s[i] == s[j]) j++;
        pf[i] = j;
    }
    return pf;
}

int kmp(string &s, string &p) {
    int n = s.size(), m = p.size(), cnt = 0;
    vector<int> pf = prefix_function(p);
    for(int i = 0, j = 0; i < n; i++) {
        while(j && s[i] != p[j]) j = pf[j-1];
        if(s[i] == p[j]) j++;
        if(j == m) {
            cnt++;
            j = pf[j-1];
        }
    }
    return cnt;
}
```

## 7.3 String Hash

```
struct hash_str {
  ll c, mod;
  vll h, p;
  hash_str(const string &s, ll c, ll mod) : c(c), mod(mod),
      h(sz(s) + 1), p(sz(s) + 1) {
    p[0] = 1;
    h[0] = 0;
    forn (i, sz(s)) {
      h[i + 1] = (c * h[i] + s[i]) % mod;
      p[i + 1] = (c * p[i]) % mod;
    }
  }
  // Returns hash of interval s[a ... b] (where 0 <= a <= b
      < sz(s))
  ll get(int a, int b) {
    return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod) %
        mod;
  }
};

bool is_pal(int l, int r, hash_str &h1, hash_str &h2) {
  int l2 = sz(h1.h) - 2 - r;
  int r2 = sz(h1.h) - 2 - l;
  return (h1.get(l, r) == h2.get(l2, r2));
}

//how many substrings of a given string are palindromes.
ll count_palindromes(string &s, hash_str &h1, hash_str &h2)
    {
  ll ans = 0;

  forn(i, sz(s)) {
    // Palindromes odd length
    int l = 0, r = min(i + 1, sz(s) - i);
    while (l < r) {
      ll mid = (l + r + 1) / 2;
      if (is_pal(i - mid + 1, i + mid - 1, h1, h2)) {
        l = mid;
      } else {
        r = mid - 1;
      }
    }
    ans += l;

    //Palindromes even length
    l = 0, r = min(i + 1, sz(s) - i - 1);
    while (l < r) {
      ll mid = (l + r + 1) / 2;
      if (is_pal(i - mid + 1, i + mid, h1, h2)) {
        l = mid;
      } else {
        r = mid - 1;
      }
    }
    ans += l;
  }

  return ans;
}

int main() {
  ll mod = 1e9 + 7;
  string s; cin >> s;

  //normal hash
  hash_str h1(s, 131, mod);

  //reverse hash
  string rev_s = s;
  reverse(all(rev_s));
  hash_str h2(rev_s, 131, mod);

  ll ans = count_palindromes(s, h1, h2);
  cout << ans << ln;
  return 0;
}
```

## 7.4 Suffix Array

```
typedef long long ll;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<ll> vll;

struct SuffixArray {
  int K = 256; // alphabet size
  int n;
  vi sa, lcp, c, cnt;

  vi build(string& str) {
    string s = str;
    s += "$";
    n = sz(s);
    sa = vi(n);
    c = vi(2 * n);
    cnt = vi((max(n, K)));
    forab (k, 0, n) {
      cnt[(int)(s[k])]++;
    }
    forab (k, 1, K) {
      cnt[k] += cnt[k - 1];
    }
    forab (k, 0, n) {
      sa[--cnt[(int)(s[k])]] = k;
    }
    c[sa[0]] = 0;
    int classes = 1;
    forab (k, 1, n) {
      if (s[sa[k]] != s[sa[k - 1]]) {
        classes++;
      }
```

```
    c[sa[k]] = classes - 1;
  }
  vi pn(n);
  vi cn(2 * n);

  for (int h = 0; (1 << h) < n; h++) {
    forab (k, 0, n) {
      pn[k] = sa[k] - (1 << h);
      if (pn[k] < 0) {
        pn[k] += n;
      }
    }
    fill(all(cnt), 0);
    forab (k, 0, n) {
      cnt[c[pn[k]]]++;
    }
    forab (k, 1, classes) {
      cnt[k] += cnt[k - 1];
    }
    for (int k = n - 1; k >= 0; k--) {
      sa[--cnt[c[pn[k]]]] = pn[k];
    }
    cn[sa[0]] = 0;
    classes = 1;
    forab (k, 1, n) {
      int cur1 = c[sa[k]];
      int cur2 = c[sa[k] + (1 << h)];
      int prev1 = c[sa[k - 1]];
      int prev2 = c[sa[k - 1] + (1 << h)];
      if (cur1 != prev1 || cur2 != prev2) {
        classes++;
      }
      cn[sa[k]] = classes - 1;
    }
    swap(c, cn);// swap c and cn
  }
  return sa;
}

/*
 * Longest Common Preffix
 * Use Kasai algorithm to build LCP array
 * LCP is an array in which every index
 * tracks how many characters two sorted
 * adjacent suffixes have in common.
 */
void kasai(string& str) {
  string s = str;
  s += "$";
  int n = sz(s);
  lcp = vi(n);
  vi inv(n);
  forab (i, 0, n)
  inv[sa[i]] = i;
  for (int i = 0, k = 0; i < n; i++) {
    if (inv[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = sa[inv[i] + 1];
    while ((i + k < n) && (j + k < n) && s[i + k] == s[j +
        k])
      k++;
    lcp[inv[i]] = k;
    if (k > 0)
      k--;
  }
}

/*
 * n = s.length();
 * unique subStrings of s = (n*(n + 1)/2) - ( lcp[i]);
 */
ll uniqueSubStrings(string& str) {
  string s = str;
  build(s);
  kasai(s);
  int n = s.length();
  ll ans = n - sa[0];
  forab (i, 1, lcp.size()) {
    ans += (n - sa[i]) - lcp[i - 1];
  }
  return ans;
}

/*
 * To find the LCS of two Strings
 * Let's combine two strings into one through the symbol "
     sharp" s1#s2
 * identify the suffixes which start in positions wich are
      inside the s1
 * identify the suffixes which start in positions wich are
      inside the s2
 * then let's build SA and LCP of combined string
 *
 * then we need to find two suffixes,
 * one should be inside s1 and other should be inside s2
 *
 * such that the length of their common prefix is a big as
      possible
 */
//longest common substring
string lcs(string& s1, string& s2) {
  string combined = s1;

  int leftS1 = 0;
  combined += ("#");
  int rightS1 = combined.length() - 1;

  int leftS2 = combined.length();
  combined += (s2);
  int rightS2 = combined.length();

  build(combined);
  kasai(combined);
  int MAX = 0;
  int start = -1;

  forab (i, 0, sa.size() - 1) {
    // if sa[i] inside s1 && sa[i + 1] inside s2
    if (sa[i] >= leftS1 && sa[i] < rightS1 && sa[i + 1] >=
        leftS2 && sa[i + 1] < rightS2) {
      if (lcp[i] > MAX) {
        MAX = lcp[i];
        start = i;
      }
      // else sa[i] inside s2 && sa[i + 1] inside s1
    } else if (sa[i + 1] >= leftS1 && sa[i + 1] < rightS1
        && sa[i] >= leftS2 && sa[i] < rightS2) {
      if (lcp[i] > MAX) {
        MAX = lcp[i];
        start = i;
      }
    }
  }
  if (start == -1) {
    return "";
  } else {
    for (int i = sa[start]; i < sa[start] + MAX; i++) {
      cout << combined[i];
    }
    cout << ln;
    // string lcs = combined.substr(sa[start], sa[start] +
        MAX);
    return "";
  }
}

int match(string& s, string& w) {
  int n = sz(s);
```

```
    int l = 1;
    int r = n;
    int lower = -1;
    while (l <= r) {
      int mid = l + (r - l) / 2;
      if (compare(s, sa[mid], w) >= 0) {
        lower = mid;
        r = mid - 1;
      } else {
        l = mid + 1;
      }
    }
    l = 1;
    r = n;
    int upper = -1;
    while (l <= r) {
      int mid = l + (r - l) / 2;
      if (compare(s, sa[mid], w) <= 0) {
        upper = mid;
        l = mid + 1;
      } else {
        r = mid - 1;
      }
    }
    upper++;
    if (lower == -1 || compare(s, sa[lower], w) != 0) {
      return -1; // no aparece
    } else {
      //cantidad de veces que aparece
      return upper - lower + 1;
    }
  }

  int compare(string& s, int x, string& w) {
    int n = sz(s);
    int qn = sz(w);

    int i = 0;
    while (i + x < n && i < qn && s[i + x] == w[i])
      i++;
    if (i >= qn) {
      return 0;
    } else if (i + x >= n) {
      return -1;
    }
    return (int)s[i + x] - (int)w[i];
  }
};

int main() {
  string s, t;
  cin >> s >> t;
  SuffixArray sf;
  sf.lcs(s, t);
  return 0;
}
```

## 7.5 Trie

```
int maxlen;
int cnt;
struct Node {
  int cnt;
  Node* child[26];
};

struct Trie {
  Node* root;
  Trie() {
    root = new Node();
  }

  void insert(const string &s) {
    Node* cur = root;
    int len = 0;
    forn(i, sz(s)) {
      ++len;
      int c = s[i] - 'a';
      if (cur->child[c] == NULL) {
        cur->child[c] = new Node();
      }
      cur->child[c]->cnt++;
      cur = cur->child[c];
    }
  }

  void dfs(Node* u, int dep){
    if(u->cnt >=3 && dep > maxlen){
      maxlen = dep;
      cnt = u->cnt;
    }
    forn(v, 26)
      if(u->child[v]!=NULL)
        dfs(u->child[v], dep + 1);
  }

  pair<int, int> query(const string &s) {
    Node* cur = root;
    forn (i, sz(s)) {
      int c = s[i] - 'a';
      if (cur->child[c] == NULL) {
        return {i, cur->cnt};
      }
      cur = cur->child[c];
    }
    return {sz(s), cur->cnt};
  }

};
```

## 7.6 Z function

```
Dada una string s, devuelve un vector Z
donde Z[i] representa el prefijo
de mayor longitud de s, que tambien
es prefijo del sufijo de s que inicia
en i. 01234567
Ejemplo:
aabzaaba "aab" es un prefijo de s
y "aaba" es un sufijo de s, Z[4] = 3.

Otra definicion: Dado un string s retorna un vector z
donde z[i] es igual al mayor
numero de caracteres desde s[i]
que coinciden con los caracteres desde s[0]

Complejidad: O(|n|)
vector<int> z_function (string &s) {
    int n = s.size();
    vector<int> z(n);
    for (int i = 1, x = 0, y = 0; i < n; i++) {
        z[i] = max(0, min(z[i-x], y-i+1));
        while (i+z[i] < n && s[z[i]] == s[i+z[i]]) {
            x = i, y = i+z[i], z[i]++;
        }
    }
    return z;
}
```

## 7.7 manacher

```
Devuelve un vector p donde,
para cada i, p[i] es igual al largo del palindromo mas
largo con centro en i.
```

```
Tener en cuenta que el string debe tener el siguiente
     formato:
%#s[0]#s[1]#...#s[n-1]#$ (s es el string original y n es el
     largo del string)

Complejidad: O(|n|)
vector<int> manacher(string s) {
    int n = s.size();
    vector<int> p(n, 0);
    int c = 0, r = 0;
    for (int i = 1; i < n-1; i++) {
        int j = c - (i-c) ;
        if (r > i) p[i] = min(r-i , p[j]);
        while (s[i+1+p[i]] == s[i-1-p[i]])
            p[i]++;
        if (i+p[i] > r) {
            c = i;
            r = i+p[i];
        }
    }
    return p;
}

Recibe el string original.

vector<int> impar(string s){
    int n = sz(s);
    vector<int> d1 (n);
    int l=0, r=-1;
    for (int i=0; i<n; ++i) {
        int k = i>r ? 1 : min(d1[l+r-i], r-i+1);
        while (i+k < n && i-k >= 0 && s[i+k] == s[i-k])
            ++k;
        d1[i] = k;
        if (i+k-1 > r)
            l = i-k+1, r = i+k-1;
    }
    return d1;
}

vector<int> par(string s){
    int n = sz(s);
    vector<int> d2 (n);
    l=0, r=-1;
    for (int i=0; i<n; ++i) {
        int k = i>r ? 0 : min (d2[l+r-i+1], r-i+1);
        while (i+k < n && i-k-1 >= 0 && s[i+k] == s[i-k-1])
            ++k;
        d2[i] = k;
        if (i+k-1 > r)
            l = i-k, r = i+k-1;
    }
    return d2;
}
```

## 7.8 minimum expression

```
Dado un string s devuelve el indice donde comienza
la rotacin lexicograficamente menor de s.

Complejidad: O(|n|)
//Factorizacion de lyndon
int minimum_expression(string s) {
    s = s+s; // si no se concatena devuelve el indice del
         sufijo menor
    int len = s.size(), i = 0, j = 1, k = 0;
    while (i+k < len && j+k < len) {
        if (s[i+k] == s[j+k]) k++;
        else if (s[i+k] > s[j+k]) i = i+k+1, k = 0; // <-
             here
        // cambiar por < para maximum
        else j = j+k+1, k = 0;
        if (i == j) j++;
    }
    return min(i, j);
}
```

## 7.9 palindrome subarrays

```
// how many subarrays s[l,r]
// exists such that you
//can reorder its characters
// to get a palindrome.
const int N = 1e6 + 5;
const int MASK = (1<<26);
int F[MASK];

int main() {
  int n;
  cin >> n;
  string s;
  cin >> s;
  ll ans = 0;
  int cur_mask = 0;
  F[0]++;
  forn(r, n) {
    cur_mask ^= 1 <<(s[r] - 'a');
    ans+=F[cur_mask];//caso par
    forn(y, 26) {
      ans+=F[cur_mask ^ (1 << y)];//caso impar
    }
    F[cur_mask]++;
  }
  cout << ans << ln;
  return 0;
}
```

## 7.10 palindrome substr range

```
  //numero de substrings
  //que son palindromos
  //en el rango [l, r]
  string s; cin >> s;
  int n = sz(s);
  const int MAXN = 5000 + 10;
  vector<vb> isPal(MAXN, vb(MAXN));
  vector<vi> dp(MAXN, vi(MAXN));
  forn(i, n){
    isPal[i][i] = 1;
    dp[i][i] = 1;
    isPal[i + 1][i] = 1;
  }
  forab(len, 2, n + 1){
    forn(i, n - len + 1){
      isPal[i][i + len - 1] = isPal[i + 1][i + len - 2] & s[i
          ] == s[i + len - 1];
      dp[i][i + len - 1] = dp[i][i + len - 2] +
      dp[i + 1][i + len - 1] - dp[i + 1][i + len - 2] +
       isPal[i][i + len - 1];
    }
  }
  int q; cin >> q;
  while(q--){
    int l, r; cin >> l >> r;
    l--; r--;
    cout << dp[l][r] << ln;
  }
```

## 7.11 psa

```
vector<vi> prefix(string &s) {
  vector<vi> psa(26, vi(sz(s) + 1));
```

```
  forab(i, 1, sz(s) + 1) {
    forn(j, 26) {
      psa[j][i] = psa[j][i - 1];
    }
    char c = s[i - 1];
    psa[c - 'a'][i]++;
  }
  return psa;
}
//freq de char c en el rango [l, r]
//int query = psa[c -'a'][r] - psa[c-'a'][l];
```

## 7.12 string utils

```
//String completa a miniscula
transform(all(in), in.begin(), ::tolower);


//A es subsecuencia de B?
bool is_subsecuence(vi &a, vi &b){
  int at = 0;
  for(auto i: b){
    if(i == a[at]) at++;
    if(at == sz(a)) return 1;
  }
  return 0;
}
```

# 8 utils

## 8.1 Bits

```
Operaciones a nivel de bits.

n & 1  -> Verifica si n es impar o no
n & (1<<k) -> Verifica si el k-esimo bit esta encendido o no
n | (1<<k) -> Enciende el k-esimo bit
n & ~(1<<k) -> Apaga el k-esimo bit
n ^ (1<<k) -> Invierte el k-esimo bit
~n     -> Invierte todos los bits
n & -n  -> Devuelve el bit encendido mas a la derecha
~n & (n+1) -> Devuelve el bit apagado mas a la derecha
n | (n+1) -> Enciende el bit apagado mas a la derecha
n & (n-1) -> Apaga el bit encendido mas a la derecha

__builtin_popcountll(x) -> Cuantos bits tiene encendidos
```

## 8.2 helps

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

bool cmp(pii &a, pii &b) {
  if(a.second == b.second) return a.first < b.first;
  return a.second > b.second;
}

//for set
set<pii, decltype(&cmp)> q(&cmp);


//funciones dentro del main
function<ll(int, int, int)> suma = [&](int a, int b, int c)
    {
  return ll(a + b + c);
};
cout << suma(11, 11, 1) << ln;

//input / output
string x;
getline(cin, x);//lee linea completa

//imprime long double con 6 decimales
printf("%.6LF\n", value);//long double
```

## 8.3 main

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

#define ln '\n'
#define all(x) x.begin(), x.end()
#define forn(i, n) for(int i = 0; i < n; i++)
#define forab(i, a, b) for (int i = a; i < b; i++)
#define pb push_back
#define sz(x) int(x.size())
#define rforn(i, n) for (int i = n-1; i >= 0; --i)
#define form(i, n, m, x) for (int i = n; i < m; i += x)
#define rform(i, n, m, x) for (int i = n; i >= m; i -= x)

#ifdef LOCAL
#include "debug.h"
#else
#define dbg(...)
#endif
```

```
typedef long long ll;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<ll> vll;

int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  #ifdef LOCAL
    freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
  #endif

  return 0;
}
```

## 8.4 template

```
//g++ -std=c++17 -Wall -Wextra -O2 -DLOCAL main.cpp -o main
    && ./main < in.txt
#include <bits/stdc++.h>
using namespace std;

void _print() {
  cerr << "]" << endl;
}
template<typename T, typename... V>
void _print(T t, V... v) {
  cerr << t;
  if (sizeof...(v)) cerr << ", ";
  _print(v...);
}

#ifdef LOCAL
#define dbg(x...) cerr << "[" << #x << "]: ["; _print(x)
#else
#define dbg(x...)
#define endl '\n'
#endif

#define pb push_back
#define sz(x) int(x.size())
#define all(x) x.begin(), x.end()
#define forn(i, n) for (int i = 0; i < n; ++i)
#define rforn(i, n) for (int i = n-1; i >= 0; --i)
#define forab(i, a, b) for (int i = a; i < b; ++i)
#define form(i, n, m, x) for (int i = n; i < m; i += x)
#define rform(i, n, m, x) for (int i = n; i >= m; i -= x)
```

```
typedef long long ll;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<ll> vll;
typedef pair<int, int> ii;
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  #ifdef LOCAL
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
  #endif LOCAL

  //codes here
  return 0;
}
```