Contents

1 KMP

- If the prefix of length i of a string is written as A^k, the largest k is $\frac{i}{i-lookup[i]}$ iff $(i - lookup[i]) \mid i$

```cpp
// KMP for string searching
#include <bits/stdc++.h>
using namespace std;

const int LIM = 10005;
int lookup[LIM];                // lookup is one indexed wrt the string target
// lookup[i] stores the largest j < i st target[1..j] is a
// suffix of target[1...i]
string target;
void compute_table() {
  lookup[0] = -1;
  lookup[1] = 0;
  int pref = 0;
  for(int i = 2; i <= target.size(); i++) {
    while(pref != -1 && target[pref] != target[i - 1]) {
      pref = lookup[pref];
    }
    pref++;
    lookup[i] = pref;
  }
}

string str;
int main() {
  int pref;                         // pref simply stores the largest prefix length metched till str[i]
  while(cin >> str >> target) {
    compute_table();
    pref = 0;
    for(int i = 0; i < str.size(); i++) {
      while(pref != -1 && str[i] != target[pref]) {
        pref = lookup[pref];
      }
      pref++;
      if(pref == target.size()) {
        printf("match found starting at index %d\n", i + 1 - (int)target.size());
        pref = lookup[pref];
      }
    }
  }
  return 0;
}
```

2 Max Bipartite Matching

- Konig's theorem - In a Bipartite graph, minimum vertex cover = maximum matching.
- There is a perfect matching in a bipartite graph $G = (L, R, E)$ for L iff $\forall l \subseteq L, |N(l)| \geq |l|$.

```cpp
//   OUTPUT: mr[i] = assignment for row node i, -1 if unassigned
//           mc[j] = assignment for column node j, -1 if unassigned
vector<int> adjlist[N];
int mr[N], mc[N], seen[N], nl, nr;;

bool FindMatch(int i) {
  for (int j = 0; j < adjlist[i].size(); j++) {
    if (!seen[adjlist[i][j]]) {
      seen[adjlist[i][j]] = true;
      if (mc[adjlist[i][j]] < 0 || FindMatch(mc[adjlist[i][j]])) {
        mr[i] = adjlist[i][j];
        mc[adjlist[i][j]] = i;
        return true;
      }
    }
  }
  return false;
}

int BipartiteMatching() {
  memset(mr, -1, sizeof mr);
  memset(mc, -1, sizeof mc);
  int ct = 0;
  for (int i = 0; i < nl; i++) {
    memset(seen, 0, sizeof seen);
    if (FindMatch(i)) ct++;
  }
  return ct;
}
```

3 Maximum Flow Dinic

- Maximum Weighted independent set in a bipartite graph with weights on nodes - for each node l_i, add edge of weight w_i with s. Same for r_j and sink t. For each edge (l_i, r_j) in the graph, add (l_i, r_j, ∞) in the graph. Max wt indep set

= (Total weight - min cut).

- Circulation with demand - every $v \in V$ has a demand $d[v] = f_{in}(v) - f_{out}(v)$. $d[v] < 0$ for producer and $d[v] > 0$ for consumer with $\sum_{v \in V} d[v] = 0$. Every edge has a capacity. For finding whether a circulation with the given demand exists, add edge $(s, v, -d[v]) \forall$ producers P, add $(v, t, d[v]) \forall$ consumers C, flow must be $\sum_{v \in P} d[v] = D$.

- Minimum path cover - a path cover (set of vertex-disjoint paths (any len ≥ 0) such that every vertex is included in a path) with minimum cardinality. for min path cover, let $V = \{1, 2 \cdots n\}$, construct $G' = (V', E')$ s.t $V' = \{x_0, x_1 \cdots x_n\} \cup \{y_0, y_1 \cdots y_n\}$ $E' = \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_j) : (i, j) \in E\}$. TODO write the cond

- dependencies resolving, maximizing the difference between resources consumed and resources generated.

```cpp
#define INF 2000000000
struct flow_graph{
    int MAX_V,E,s,t,head,tail;
    int *cap,*to,*next,*last,*dist,*q,*now;
    flow_graph(){}
    flow_graph(int V, int MAX_E){
        MAX_V = V; E = 0;
        cap = new int[2*MAX_E], to = new int[2*MAX_E], next = new int[2*MAX_E];
        last = new int[MAX_V], q = new int[MAX_V], dist = new int[MAX_V], now = new int[MAX_V];
        fill(last,last+MAX_V,-1);
    }
    ~flow_graph() {
      delete[] cap; delete[] to; delete[] next; delete[] last;
      delete[] dist; delete[] q; delete[] now;
    }
    void clear(){
        fill(last,last+MAX_V,-1);
        E = 0;
    }
    void add_edge(int u, int v, int uv, int vu = 0){
        to[E] = v, cap[E] = uv, next[E] = last[u]; last[u] = E++;
        to[E] = u, cap[E] = vu, next[E] = last[v]; last[v] = E++;
    }
  bool bfs(){
    fill(dist,dist+MAX_V,-1);
    head = tail = 0;
    q[tail] = t; ++tail;
    dist[t] = 0;
    while(head<tail){
      int v = q[head]; ++head;
      for(int e = last[v];e!=-1;e = next[e]){
        if(cap[e^1]>0 && dist[to[e]]==-1){
          q[tail] = to[e]; ++tail;
          dist[to[e]] = dist[v]+1;
        }
      }
    }
    return dist[s]!=-1;
  }
  int dfs(int v, int f){
    if(v==t) return f;
    for(int &e = now[v];e!=-1;e = next[e]){
      if(cap[e]>0 && dist[to[e]]==dist[v]-1){
        int ret = dfs(to[e],min(f,cap[e]));
        if(ret>0){
          cap[e] -= ret;
          cap[e^1] += ret;
          return ret;
        }
      }
    }
    return 0;
  }
  long long max_flow(int source, int sink){
    s = source; t = sink;
    long long f = 0;
    int x;
    while(bfs()){
      for(int i = 0;i<MAX_V;++i) now[i] = last[i];
      while(true){
        x = dfs(s,INF);
        if(x==0) break;
        f += x;
      }
    }
    return f;
  }
}G;
int main(){
  int V,E,u,v,c;
  scanf("%d %d",&V,&E);
  G = flow_graph(V,E);
  for(int i = 0;i<E;++i){
    scanf("%d %d %d",&u,&v,&c);
    G.add_edge(u-1,v-1,c,c);
```

```
  }
  printf("%lld\n",G.max_flow(0,V-1));
  return 0;
}
```

4 Min Cost Max Flow

- For finding minimum cost with a fixed flow K, add edge $(t, t^{'})$ with capacity k and cost 0. Find flow from s to $t^{'}$.

```
// Implementation of min cost max flow algorithm using adjacency
// matrix (Edmonds and Karp 1972).  This implementation keeps track of
// forward and reverse edges separately (so you can set cap[i][j] !=
// cap[j][i]).  For a regular max flow, set all edge costs to 0.
//
// Running time, O(|V|^2) cost per augmentation
//     max flow:           O(|V|^3) augmentations
//     min cost max flow:  O(|V|^4 * MAX_EDGE_COST) augmentations
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - (maximum flow value, minimum cost value)
//     - To obtain the actual flow, look at positive values only.

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = numeric_limits<L>::max() / 4;

struct MinCostMaxFlow {
  int N;
  VVL cap, flow, cost;
  VI found;
  VL dist, pi, width;
  VPII dad;

  MinCostMaxFlow(int N) :
    N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
    found(N), dist(N), pi(N), width(N), dad(N) {}

  void AddEdge(int from, int to, L cap, L cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
  }

  void Relax(int s, int k, L cap, L cost, int dir) {
    L val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k]) {
      dist[k] = val;
      dad[k] = make_pair(s, dir);
      width[k] = min(cap, width[s]);
    }
  }

  L Dijkstra(int s, int t) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
      int best = -1;
      found[s] = true;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
        Relax(s, k, flow[k][s], -cost[k][s], -1);
        if (best == -1 || dist[k] < dist[best]) best = k;
      }
      s = best;
    }

    for (int k = 0; k < N; k++)
```

```
      pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
  }

  pair<L, L> GetMaxFlow(int s, int t) {
    L totflow = 0, totcost = 0;
    while (L amt = Dijkstra(s, t)) {
      totflow += amt;
      for (int x = t; x != s; x = dad[x].first) {
        if (dad[x].second == 1) {
          flow[dad[x].first][x] += amt;
          totcost += amt * cost[dad[x].first][x];
        } else {
          flow[x][dad[x].first] -= amt;
          totcost -= amt * cost[x][dad[x].first];
        }
      }
    }
    return make_pair(totflow, totcost);
  }
};
```

5 Min cost Bipartite Matching

```
//   cost[i][j] = cost for pairing left node i with right node j
//   Lmate[i] = index of right node that left node i pairs with
//   Rmate[j] = index of left node that right node j pairs with
// The values in cost[i][j] may be positive or negative.  To perform
// maximization, simply negate the cost[][] matrix.
typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());
  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
  for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
  }
  // construct primal solution satisfying complementary slackness
  Lmate = VI(n, -1);
  Rmate = VI(n, -1);
  int mated = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (Rmate[j] != -1) continue;
      if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
  Lmate[i] = j;
  Rmate[j] = i;
  mated++;
  break;
      }
    }
  }
  VD dist(n);
  VI dad(n);
  VI seen(n);
  // repeat until primal solution is feasible
  while (mated < n) {
    // find an unmatched left node
    int s = 0;
    while (Lmate[s] != -1) s++;

    // initialize Dijkstra
    fill(dad.begin(), dad.end(), -1);
    fill(seen.begin(), seen.end(), 0);
    for (int k = 0; k < n; k++)
      dist[k] = cost[s][k] - u[s] - v[k];
    int j = 0;
    while (true) {
      // find closest
      j = -1;
      for (int k = 0; k < n; k++) {
  if (seen[k]) continue;
  if (j == -1 || dist[k] < dist[j]) j = k;
      }
      seen[j] = 1;
      // termination condition
      if (Rmate[j] == -1) break;
      // relax neighbors
      const int i = Rmate[j];
      for (int k = 0; k < n; k++) {
```

```
  if (seen[k]) continue;
  const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
  if (dist[k] > new_dist) {
    dist[k] = new_dist;
    dad[k] = j;
  }
      }
    }
    // update dual variables
    for (int k = 0; k < n; k++) {
      if (k == j || !seen[k]) continue;
      const int i = Rmate[k];
      v[k] += dist[k] - dist[j];
      u[i] -= dist[k] - dist[j];
    }
    u[s] += dist[j];
    // augment along path
    while (dad[j] >= 0) {
      const int d = dad[j];
      Rmate[j] = Rmate[d];
      Lmate[Rmate[j]] = j;
      j = d;
    }
    Rmate[j] = s;
    Lmate[s] = j;

    mated++;
  }
  double value = 0;
  for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];
  return value;
}
```

6 Min Cut

```
// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//
// OUTPUT:
//     - (min cut value, nodes in half of min cut)

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {
  int N = weights.size();
  VI used(N), cut, best_cut;
  int best_weight = -1;

  for (int phase = N-1; phase >= 0; phase--) {
    VI w = weights[0];
    VI added = used;
    int prev, last = 0;
    for (int i = 0; i < phase; i++) {
      prev = last;
      last = -1;
      for (int j = 1; j < N; j++)
  if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
      if (i == phase-1) {
  for (int j = 0; j < N; j++) weights[prev][j] += weights[last][j];
  for (int j = 0; j < N; j++) weights[j][prev] = weights[prev][j];
  used[last] = true;
  cut.push_back(last);
  if (best_weight == -1 || w[last] < best_weight) {
    best_cut = cut;
    best_weight = w[last];
  }
      } else {
  for (int j = 0; j < N; j++)
    w[j] += weights[last][j];
  added[last] = true;
      }
    }
  }
```

```
  return make_pair(best_weight, best_cut);
}
```

7 Segment Tree

- Check if using only `set` can solve it.
- `IDENTITY` is such that `merge(n, IDENTITY)` = `n`.

```
const int N = 100003;
const int LGN = 17;

int h, n, left_most, right_most;

struct node {

  node() {}
  node(int val) {}
  void merge(node& l, node& r) {}
  void split(node& l, node& r) {}
}tree[1<<(LGN + 1)];

void update_single(node& n, int d) {}

const node IDENTITY;

node query(int root, int left_leaf, int right_leaf, const int u, const int v) {
  if(u >= v) return IDENTITY;
  if(u <= left_leaf && right_leaf <= v)
    return tree[root];
  int mid = (left_leaf + right_leaf) >> 1,
      lc = root<<1, rc = lc | 1;
  tree[root].split(tree[lc], tree[rc]);
  node ret, l, r;
  if(u < mid) l = query(lc, left_leaf, mid, u, v);
  if(v > mid) r = query(rc, mid, right_leaf, u, v);
  ret.merge(l, r);
  return ret;
}

void splitdown(int idx) {
  if(idx > 1) splitdown(idx>>1);
  tree[idx].split(tree[idx<<1], tree[(idx<<1)|1]);
}

void update(int idx, node new_node) {
  idx |= (1<<h);
  splitdown(idx>>1);
  tree[idx] = new_node;
  idx >>= 1;
  while(idx > 0) {
    tree[idx].merge(tree[idx<<1], tree[(idx<<1)|1]);
    idx >>= 1;
  }
}

void range_update(int root, int left_leaf, int right_leaf, const int u, const int v, int d) {
  if(u >= v) return;
  if(u <= left_leaf && right_leaf <= v)
    return update_single(tree[root], d);
  int mid = (left_leaf + right_leaf) >> 1,
      lc = root<<1, rc = lc | 1;
  if(u < mid) range_update(lc, left_leaf, mid, u, v, d);
  if(v > mid) range_update(rc, mid, right_leaf, u, v, d);
  tree[root].merge(tree[lc], tree[rc]);
}

// searches for the last place i, such that mrege[0...i] compares less than k
// requires < operator defined
int binary_search(node k) {
  int root = 1;
  node nd;
  while(root < left_most) {
    int lc = root<<1, rc = lc|1;
    tree[root].split(tree[lc], tree[rc]);
    node m;
    m.merge(nd, tree[lc]);
    if(m < k) {
      nd = m;
      root = rc;
    } else {
      root = lc;
    }
  }
  node ret;
  ret.merge(nd, tree[root]);
  if(m<k)return root - left_most;
  else return root - 1 - left_most;
```

```
}

void init(int size, int A[]) {
  n = size;
  h = ceil(log2(n));
  left_most = 1<<h, right_most = left_most<<1;
  for(int i = 0; i < n; i++) tree[i|(1<<h)] = (node){A[i]};
  for(int i = left_most - 1; i > 0; i--) tree[i].merge(tree[i<<1], tree[(i<<1)|1]);
}
```

8 BIT

```
/*
   In this BIT, $a[1...n]$ is the frequency array
   BIT[idx] stores for $a[idx - 2^r + 1] ... a[idx]$
   r is the lowest bit of idx that is 1;
   $2^r$ = idx & -idx
*/
//returns a[idx] + a[idx-1] + ... + a[1].
int read(int idx){
  int sum = 0;
  while (idx > 0){
    sum += BIT[idx];
    idx -= (idx & -idx);
  }
  return sum;
}

void update(int idx ,int val){
  while (idx <= n){
    BIT[idx] += val;
    idx += (idx & -idx);
  }
}
```

9 $\binom{n}{r}$

- Recurrence for derangements : $d[0] = 1, d[1] = 0, d[i] = (i-1) * (d[i-1] + d[i-2])$

```
/*
   $\binom{n}{r} = \binom{n}{r-1} * \frac{n-r+1}{r}$
   $\binom{n}{r} = \binom{n-1}{r-1} * \frac{n}{k}$
   $\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$
*/
LL binom[N][N];
LL nc[N];
void fillncr() {
  binom[1][1] = 1;
  for(int n = 0; n < N; n++) binom[n][0] = 1;
  for(int n = 1; n < N; n++)
    for(int r = 1; r <= n; r++)
      binom[n][r] = binom[n-1][r-1] + binom[n-1][r];
}

void fillnc(int n) {
  int r;
  nc[0] = 1; nc[1] = n;
  for(r = 2; r<=n; r++) nc[r] = (nc[r-1] * (n-r+1)) / r;
}
```

10 Tricks

- For updating, query in a subtree of a tree, build a segtree using dfs-order (in dfs-order, do not increment time on finish and update using only start time).
- For bfs with multi nodes, push them all at once in the beginning of the dfs as source.
- Use square root decomposition :P, on arrays and / or queries. In case of queries, run update every $\sqrt{n}$ queries and for the queries inside the chunk, keep a `vector new`, check using other method on `new`.
- For maintaining min / max under bitwise operations (`xor`,..),store the numbers as fixed length binary strings in a trie. Maintain a variable `pending` that keeps the number with which the entire chunk has to be operated with. For a query, find the number bit by bit, starting from the most significant bit.
- If there multiple updates of the form *increase / decrease [l, r] by* δ on array `A[]` and only one query at the end, keep a separate array `d[]`. Process *increase [l, r] by* $\delta \rightarrow$ *d[l] +=* δ*, d[r + 1] -=* δ.
  At the end, for i in $[0, n)$ `A[i] = A[i - 1] + d[i]`.

11 Trie

-

```
struct trie {
  trie *child[10];
  bool isLeaf;
  trie() {
    memset(child, 0, sizeof child);
    isLeaf = false;
  }
  bool insert(const string& s) {
    int p;
    trie* tr = this;
    for(int i = 0; i < s.size(); i++) {
      p = s[i] - '0';
      if(tr->child[p] == NULL) tr->child[p] = new trie();
      tr = tr->child[p];
    }
    tr->isLeaf = true;
  }
};
```

12 Permanent of a Matrix

- Permanent of the adjacency matrix of a bipartite graph is the number of perfect matchings
- Number of restriced permutations is the permanent of the restriction matrix ($a[i][j] = 1$ iff $a[i]$ can go to pos j).

```
static int MOD = 100000007;
public static long permanent(int[][] A) {
  int n = A.length;
  long[] dp = new long[1<<n];
  dp[0] = 1;
  for(int i = 0;i < 1<<n;i++){
    for(int j = 0;j < n;j++){
      if((i&(1<<j))==0){
        dp[i|(1<<j)] += dp[i]*A[Integer.bitCount(i)][j];
        dp[i|(1<<j)] %= MOD;
      }
    }
  }
  return dp[(1<<n)-1];
}
```

13 Disjoint Set Union

- If there is union as well as breaking of sets, use square root decomposition.

```
int par[N], rank[N];
int find_set(int x) {
  if(x == par[x]) return x;
  else return (par[x] = find_set(par[x]));
}
void merge_set(int x, int y)
  int px = find_set(x), py = find_set(y);
  if(rank[px] > rank[py]) par[py] = px;
  else par[px] = py;
  if((px != py) && (rank[px] == rank[py])) rank[py]++;
  }
void init() {
  for(int i = 0; i < N: i++) par[i] = i;
  memset(rank, 0, sizeof rank);
}
```

14 Range Minima Query

```
1  // RM[i, j] = min(A[k]), k ∈ [i, i + 2^j)
2  // for storing index as well, just change RM from int to pair<int, int> and line 7 to RM[i][0] = make_pair(A[i], 0)
3
4  int RM[N][LGN];
5  void make_rmq(const int n, int A[]) {
6    for(int i = 0; i < n; i++)
7      RM[i][0] = A[i];
8    for(int j = 1; (1<<j) <= n; j++)
9      for(int i = 0; i + (1<<j) <= n; i++)
10       RM[i][j] = min(RM[i][j-1], RM[i + (1 << (j-1))][j-1]);
11 }
12 int minq(int i, int j) {                // minima in interval [i, j]
13   int k = log2(j + 1 - i);
14   return min(RM[i][k], RM[j + 1 - (1<<k)][k]);
15 }
```

15 Lowest Common Ancestor

- Any path from $u \to v$ in a tree is broken into a path from $u \to lca(u, v) \to v$.
- To check whether 2 paths a and b intersect -

```
bool rootp_intersect(int u, int up, int v, int vp) {
  return (level[lca(u,v)] >= max(level[up], level[vp]));
}

bool intersect(route& a, route& b) {
  return ( rootp_intersect(a.x, a.lca, b.x, b.lca)
    ||   rootp_intersect(a.x, a.lca, b.y, b.lca)
    ||   rootp_intersect(a.y, a.lca, b.y, b.lca)
    ||   rootp_intersect(a.y, a.lca, b.x, b.lca) );
}
```

```
int parent[N], level[N];
int par[N][LGN];                // par[i][j] is 2^j th ancestor of i
int n;
void make_p() {
  memset(par, -1, sizeof(par));
  for(int i=1; i<=N; i++)
    par[i][0] = parent[i];
  for(int j=1; 1<<j < N; j++)
    for(int i=1; i<=N; i++)
      if(par[i][j-1] != -1)
        par[i][j] = par[par[i][j-1]][j-1];
}
int lca(int u, int v) {
  int log;
  if(level[u] < level[v]) swap(u,v);
  if(level[u] == 0) return u;
  log = log2(level[u]);
  for(int i = log; i >= 0; i--)
    if(level[u] - (1<<i) >= level[v])
      u = par[u][i];
  if(u == v) return u;
  for(int i = log; i >= 0; i--)
    if(par[u][i] != -1 && par[u][i] != par[v][i])
      u = par[u][i], v = par[v][i];
  return parent[u];
}
```

16 Convex Hull

```
// Compute the 2D convex hull of a set of points using the monotone chain
// algorithm.  Eliminate redundant points from the hull if REMOVE_REDUNDANT is
// #defined.
// Running time: O(n log n)
//   INPUT:   a vector of input points, unordered.
//   OUTPUT:  a vector of points in the convex hull, counterclockwise, starting
//            with bottommost/leftmost point
#define REMOVE_REDUNDANT
typedef double T;
const T EPS = 1e-7;
struct PT {
  T x, y;
  PT() {}
  PT(T x, T y) : x(x), y(y) {}
  bool operator<(const PT &rhs) const { return make_pair(y,x) < make_pair(rhs.y,rhs.x); }
  bool operator==(const PT &rhs) const { return make_pair(y,x) == make_pair(rhs.y,rhs.x); }
};
T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) + cross(c,a); }
#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c) {
  return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif
void ConvexHull(vector<PT> &pts) {
  sort(pts.begin(), pts.end());
  pts.erase(unique(pts.begin(), pts.end()), pts.end());
  vector<PT> up, dn;
  for (int i = 0; i < pts.size(); i++) {
    while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
    while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
    up.push_back(pts[i]);
    dn.push_back(pts[i]);
  }
  pts = dn;
  for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);
#ifdef REMOVE_REDUNDANT
  if (pts.size() <= 2) return;
  dn.clear();
  dn.push_back(pts[0]);
```

```
  dn.push_back(pts[1]);
  for (int i = 2; i < pts.size(); i++) {
    if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
    dn.push_back(pts[i]);
  }
  if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
    dn[0] = dn.back();
    dn.pop_back();
  }
  pts = dn;
#endif
}
```

17 Dijkstra

- For dijkstra from multi nodes, push them all in the priority queue in the beginning with distance 0.
- BELLMAN FORD - *initialize(s); for i in* $[1, |V|)$ *:* $\forall(u, v) \in E$ *: Relax*(u, v, w)
- For any edge (u, v) after BELLMAN-FORD, if $d[v] > d[u] + w(u, v)$, then the graph has a -ve wt cycle.
- Floyd warshall

```
for(int k = 0; k < n; k++)
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
      adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j])
```

```
vector<int> adjlist[N];
int dist[N];

void dijkstra(int s) {
  bool vis[N];
  memset(vis, false, sizeof vis);
  fill(dist, dist + N,INF);
  priority_queue< ii, vector<ii>, greater<ii> > q;
  dist[s] = 0;
  q.push( ii(0, s) );
  while(!q.empty()) {
    ii front = q.top(); q.pop();
    int u = front.second, d = front.first;
    if(vis[u]) continue;
    vis[u] = true;
    for(ii wv : Adjlist[u]) {
      if(dist[wv.second] > dist[u] + wv.first) {  // relax operation
        dist[wv.second] = dist[u] + wv.first;
        q.push(ii(dist[wv.second], wv.second));
      }
    }
  }
  return;
}
```

18 Catalan Number

- $C_n = \frac{1}{n+1}\binom{2n}{n} = \prod_{k=2}^{n} \frac{n+k}{k}$
- First few Catalan numbers for n = 0, 1, 2, 3 $\cdots$ are 1, 1, 2, 5, 4, 14, 42, 132 $\cdots$
- Recurrence $C_0 = 1$ and $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} = \frac{2(2n+1)}{n+2} C_n$
- C_n is the number of Dyck words of length 2n. A Dyck word is a string consisting of n X's and n Y's such that no initial segment of the string has more Y's than X's.
- C_n counts the number of expressions containing n pairs of parentheses which are correctly matched.
- C_n is the number of different ways n + 1 factors can be completely parenthesized.
- C_n is the number of full binary trees with n + 1 leaves (n internal nodes).
- C_n is the number of different ways a convex polygon with n + 2 sides can be cut into triangles by connecting vertices with straight lines.
- C_n is the number of permutations of 1, ..., n that avoid the pattern 123 (or any of the other patterns of length 3). For n = 4, they are 1432, 2143, 2413, 2431, 3142, 3214, 3241, 3412, 3421, 4132, 4213, 4231, 4312 and 4321.
- C_n is the number of ways to tile a stairstep shape of height n with n rectangles.

19 Longest Palindromic Substring

- longest palindromic substring or longest symmetric factor problem is the problem of finding a maximum-length contiguous substring of a given string that is also a palindrome.

```
// Manachar's algorithm
// Returns half of length of largest panlindrome centered at every position in the string
vector<int> manacher(string s) {
  vector<int> ans(s.size(),0);
  int maxi = 0;
  for(int i=1;i<s.size();i++) {
    int k = 0;
    if(maxi+ans[maxi]>=i)
    k = min(ans[maxi]+maxii,ans[2*maxii]);
    for(;s[i+k]==s[ik] && ik>=0 && i+k<s.size();k++);
    ans[i] = k1;
    if(i+ans[i]>maxi+ans[maxi])
    maxi = i;
  }
  return ans;
}
```

20 Graph Coloring

- Color the nodes of a graph using m colors such that no adjacent vertices have the same color

```
#define V 4
void printSolution(int color[]);
bool isSafe (int v, bool graph[V][V], int color[], int c) {
  for (int i = 0; i < V; i++)
    if (graph[v][i] && c == color[i])
      return false;
  return true;
}
/* A recursive utility function to solve m coloring problem */
bool graphColoringUtil(bool graph[V][V], int m, int color[], int v) {
  /* base case: If all vertices are assigned a color then
     return true */
  if (v == V)
    return true;
  /* Consider this vertex v and try different colors */
  for (int c = 1; c <= m; c++) {
    /* Check if assignment of color c to v is fine*/
    if (isSafe(v, graph, color, c))
    {
      color[v] = c;
      /* recur to assign colors to rest of the vertices */
      if (graphColoringUtil (graph, m, color, v+1) == true)
        return true;
      /* If assigning color c doesn't lead to a solution
         then remove it */
      color[v] = 0;
    }
  }
  /* If no color can be assigned to this vertex then return false */
  return false;
}
bool graphColoring(bool graph[V][V], int m)
{
  // Initialize all color values as 0. int *color = new int[V];
  for (int i = 0; i < V; i++)
    color[i] = 0;
  // Call graphColoringUtil() for vertex 0
  if (graphColoringUtil(graph, m, color, 0) == false) {
    printf("Solution does not exist");
    return false;
  }
  // Print the solution
  printSolution(color);
  return true;
}
// driver program to test above function
int main()
{
  bool graph[V][V] = {{0, 1, 1, 1},{1, 0, 1, 0}};
  int m = 3; // Number of colors
  graphColoring (graph, m);
  return 0;
}
```

21 Euler Tour

First let's see the conditions for an undirected graph:

- An undirected graph has an eulerian circuit if and only if it is connected and each vertex has an even degree (degree is the number of edges that are adjacent to that vertex).

- An undirected graph has an eulerian path if and only if it is connected and all vertices except 2 have even degree. One of those 2 vertices that have an odd degree must be the start vertex, and the other one must be the end vertex.

For a directed graph we have:

- A directed graph has an eulerian circuit if and only if it is connected and each vertex has the same in-degree as out-degree.
- A directed graph has an eulerian path if and only if it is connected and each vertex except 2 have the same in-degree as out-degree, and one of those 2 vertices has out-degree with one greater than in-degree (this is the start vertex), and the other vertex has in-degree with one greater than out-degree (this is the end vertex).

Algorithm for undirected graphs:

1. Start with an empty stack and an empty circuit (eulerian path).
   - If all vertices have even degree - choose any of them.
   - If there are exactly 2 vertices having an odd degree choose one of them.
   - Otherwise no euler circuit or path exists.
2. If current vertex has no neighbors - add it to circuit, remove the last vertex from the stack and set it as the current one. Otherwise (in case it has neighbors) - add the vertex to the stack, take any of its neighbors, remove the edge between selected neighbor and that vertex, and set that neighbor as the current vertex.
3. Repeat step 2 until the current vertex has no more neighbors and the stack is empty. Note that obtained circuit will be in reverse order -from end vertex to start vertex.

Algorithm for directed graphs:

1. Start with an empty stack and an empty circuit (eulerian path).
   - If all vertices have same out-degrees as in-degrees choose any of them.
   - If all but 2 vertices have same out-degree as in-degree, and one of those 2 vertices has out-degree with one greater than its in-degree, and the other has in-degree with one greater than its out-degree - then choose the vertex that has its out-degree with one greater than its in-degree.

22 Pick's theorem

- Area A of a polygon constructed on a grid of equal-distanced points with i lattice points in the interior located in the polygon and the number b lattice points on the boundary placed on the polygon's perimeter is $A = i + \frac{b}{2} - 1$.
- It can be used to calculate the number of integral points inside the polygon, for example in a quadrilateral -

```
// points are (xa, ya), (xb, yb), (xc, yc), (xd, yd)
// ans = i + b, the number of points in the interior as well as the boundary of the quadrilateral
cin >> xa >> ya >> xb >> yb;
cin >> xc >> yc >> xd >> yd;
int b = 0;
b = gcd(abs(xa-xb), abs(ya-yb)) + gcd(abs(xb-xc),abs(yb-yc)) + gcd(abs(xc-xd),abs(yc-yd))  + gcd(abs(xd-xa),abs(yd-ya));
int A2 = 0;
x1 = xc - xa;
y1 = yc - ya;
x2 = xd - xb;
y2 = yd - yb;
A2 = abs(x1*y2 - x2*y1);
ans = (A2 + 2 + b)/2;
```

23 Topological Sort

1. dfs() on graph G and then reverse sort in order of finish time.

24 Strongly connected components

1. dfs() on graph G and then sort in reverse order of finish times.
2. do dfs() on G^r (reversed edges) but in the main loop start dfs from each vertex in reverse order of finish time. Label each component as a separate scc.
3. Articulation points and Bridges -

```
// algo for articulation points abr bridges
// d[] is the discovered time
// low[v] is the lowest d[] among all the vertices reachable (taking the edges in the same direction as dfs) from subtree rooted at v,
// vertex 0 is the root, d[] is -1 initially

const int N = 100003;
vector<int> adjlist[N];
int d[N], tm = 0, low[N], par[N], rnk[N];
int n, m;

void dfs(int u, int parent) {
```

```
  d[u] = tm++;
  low[u] = d[u];
  for(int v : adjlist[u]) {
    if(v == parent) continue;
    if(d[v] == -1) {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if(low[v] == d[v]) {
        // (u, v) is a bridge
      }
      if ((low[v] >= d[u]) || (!u && adjlist[u].size() > 1)) {
        // u is an articulation point
      }
    } else {
      // back edge
      low[u] = min(low[u], d[v]);
    }
  }
  tm++;
}
```

25 Euler's Totient fn

- $\varphi(n)$ is the number of integers k in the range $1 \leq k \leq n$ for which the $gcd(n, k) = 1$
- Euler's theorem : $a^{\varphi(n)} \equiv 1(modn)$ where $gcd(a, n) = 1$.
- $\varphi(n) = n \prod_{p|n}(1 - \frac{1}{p})$
- $\sum_{d|n} \varphi(d) = n$
- $a|b \Rightarrow \varphi(a)|\varphi(b)$
- $n|\varphi(a^n - 1)$ (a, n > 1)
- $\varphi(mn) = \varphi(m)\varphi(n)\frac{d}{\varphi(d)}$ where d = gcd(m, n)
- $\varphi(2m) = 2\varphi(m)$ if m is even, $\varphi(m)$ if m is odd.
- $\varphi(n^m) = n^{m-1}\varphi(n)$
- $\varphi(lcm(m, n))\varphi(gcd(m, n)) = \varphi(m)\varphi(n)$
- $\sum_{1 \leq k \leq n, (k,n)=1} k = \frac{1}{2}n\varphi(n)$
- $\sum_{1 \leq k \leq n, (k,n)=1} gcd(k - 1, n) = \varphi(n)d(n)$, d(n) is the number of divisors of n

```
int etf[N];
void etf_fill( int *etf) {
  etf[1] = 1;
  for(int i=2; i<=N; i++) {
    etf[i] = i;
  }
  for(int i=2; i<=N; i++) {
    if(etf[i] == i) {         // i is prime
      etf[i] = i-1;         // correcting its value
      for(int j=2; j*i <= N; j++) { //multiply (i-1) / i to all its multiples
        etf[j*i] = ( etf[j*i] / i) * (i - 1);
      }
    }
  }
  for(int i=2; i<=N; i++) {
    if(etf[i] == i) {         //prime
      etf[i] = i-1;         //correcting for primes > sqrt(N)
    }
  }
}
/*Directly computing, taken from topcoder*/
int fi(int n)  {
  int result = n;
  int i;
  for( i=2;i*i <= n;i++) {
    if (n % i == 0) result -= result / i;
    while (n % i == 0) n /= i;
  }
  if (n > 1) result -= result / n;
  return result;
}
```

26 Pollard Rho

```
/* pollard rho integer factorisation */
LL pollard_rho(LL n) {
	LL y;
	LL x=rand()%n;
	y = x;
	LL d=1;
	while(d==1 || d == n) {
		x=(x*x+1)%n;
		y=((y*y+1) * (y*y+1) + 1)%n;
		d = gcd(n,(LL)abs(x - y));
	}
	return d;
}

void all_factor(LL n)
{
	LL d;
	while(n>1) {
		d = pollard_rho(n);
		printf("%lld\n",d);
		n = n/d;
	}
	return;
}
```

27 Wilson Theorem, Lucas Theorem, Kummer's Theorem, Fermat's theorem on sum of squares

1. p is prime iff $(p-1)! \equiv -1(modp)$
2. $\binom{m}{n} = \prod_{i=0}^{k} \binom{m_i}{n_i} (modp)$ where
   $m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0$
   $n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$
3. Given $n \geq m \geq 0$ and a prime p, the maximum integer k such that p^k divides $\binom{n}{m}$ is equal to the number of carries when m is added to $n-m$ in base p.
4. An odd an odd prime p is expressible as $p = x^2 + y^2$ with x and y integers, if and only if $p \equiv 1(mod4)$

28 Game Theory

- Staircase nim : Stairs plies are (x1, x2, x3, ...). A move of staircase nim consists of moving any positive number of coins from any step, j, to the next lower step, j 1. Coins reaching the ground (step 0) are removed from play. Last to move wins.
  A move from an even stair to the stair below it is not useful as one can replicate the move by moving the same pile to the pile on the next stair. Thus, the odd positions make a nim P position (winning for Previous) or N position (winning for Next)
- Circular nim $CN(n,k)$: n stacks of tokens arranged in a circle. Select k consecutive stacks and remove atleast one token from at least one of the stacks.
  $L(CN(n,1)) = L(xor = 0)$, $L(CN(n,n-1)) = \{a, a, \cdots, a | a \geq 0\}$.
  $L(CN(4,2)) = \{(a,b,a,b) | a,b \geq 0\}$, $L(CN(5,2)) = \{(a^*, b, c, d, b) | a^* + b = c + d, a^* = max(p)\}$
  $L(CN(5,3)) = \{(0,b,c,d,b) | b = c + d\}$, $L(CN(6,3)) = \{(a,b,c,d,e,f) | a + b = d + e, b + c = e + f\}$

29 Polynomial

```
#define ctype int
typedef vector<ctype> poly;
poly d(const poly& f) {
	poly df;
	for(int i=1; i < f.size(); i++) {
		df.push_back(f[i] * i);
	}
	//df.push_back(0);
	return df;
}
poly operator+ (const poly &f, const poly &g) {
	poly h;
	int i=0;
	for(; i < min(f.size(), g.size()); i++) h.push_back(f[i] + g[i]);
	for(; i < f.size(); i++) h.push_back(f[i]);
	for(; i < g.size(); i++) h.push_back(g[i]);
	return h;
}
poly operator- (const poly &f, const poly &g) {
	poly h;
	int i=0;
	for(; i < min(f.size(), g.size()); i++) h.push_back(f[i] - g[i]);
	for(; i < f.size(); i++) h.push_back(f[i]);
```

```
        for(; i < g.size(); i++) h.push_back(-g[i]);
        return h;
}
poly operator* (const poly &f, const poly &g) {
        poly h(f.size() + g.size() - 1, 0);
        for(int i=0; i < f.size(); i++) {
                for(int j=0; j < g.size(); j++) {
                        h[i+j] += f[i] * g[j];
                }
        }
        return h;
}
ostream &operator<< (ostream &os, const poly &f) {                // for direct output
        if(!f.size()) os << "0";
        else {
                os << f[0];
                for(int i=1; i < f.size(); i++) {
                        os << " + " << f[i] << "x^" << i;
                }
        }
}
```

30 Miller Rabin

```
bool Miller(ll p,int iteration) {
    if (p < 2) return false;
    if (p != 2 && p % 2==0) return false;
    ll s = p - 1;
    while (s % 2 == 0) s /= 2;
    for (int i = 0; i < iteration; i++) {
        ll a = rand() % (p - 1) + 1, temp = s;
        ll mod = modpow(a, temp, p);
        while (temp != p - 1 && mod != 1 && mod != p - 1)
        {
            mod = mulmod(mod, mod, p);
            temp *= 2;
        }
        if (mod != p - 1 && temp % 2 == 0) return false;
    }
    return true;
}
```

31 Longest Increasing Subsequence

```
int M[LIM];          // M[i] contains the index of the smallest element of A[] at which a non-decreasing sequence of
// length i ends, A[M[1]], A[M[2]], ... form a nondecreasing subsequence as if there is a subsequence of length i ending at
// A[M[i]] then there exists a subsequence of length i-1 ending at P[M[i]]
int P[LIM];                  // P[i] contains the index of the parent of A[i] in the lis
int N;
int lis(int A[]) {
  M[0] = -1;
  memset(P, -1, sizeof P);
  M[1] = 0;
  int maxL = 1;                  // maxL is the current length of the longest subsequence found, L is the length of subsequence
  // found ending at A[i]
  int low, high, mid;
  for(int i = 1; i < N; i++) {
    // for A[i], find the largest j such that A[M[j]] <= A[i]
    low = 0; high = maxL;
    while(high - low > 1) {
      mid = (low + high) >> 1;
      if(A[M[mid]] <= A[i]) low = mid;
      else high = mid;
    }
    if(A[M[high]] <= A[i]) mid = high;
    else mid = low;
    P[i] = M[mid];
    if(mid == maxL) M[++maxL] = i;
    else if(A[i] < A[M[mid + 1]]) M[mid+1] = i;
  }
  return maxL;
}
void printlis(int idx, const int A[]) {
  if(P[idx] >= 0) printlis(P[idx], A);
  cout << A[idx] << " ";
}
int main() {
  int A[LIM];
  cin >> N;
  for(int i = 0; i < N; i++) cin >> A[i];
  int ans = lis(A);
  printlis(M[ans], A);
  return 0;
}
```

32 Suffix Array and LCP array

- Suffix array is the ranking of the suffixes of a string in lexicographical order.

```
int LCP(int, int);        // return the longest commom prefix of substrings starting from x and y, also prints the LCP
void make_array(char* );
int LCPlen(int, int);        // return the length of the longest common prefix
char str[N];                // Find the suffix array of string str
int Sarray[N];        // The suffix array (lexographically ascending order)
int LCParray[N];          // LCP array
int P[LGN][N];         // P[k][i] stores the position of (substring of str[] of length 2^k starting at i ) in
// the sorted array of all substrings of str of length 2^k
int len, stp;
struct entry {
  int nr[2];          // in case of a substring of len 2^k, will store the 'position' of 1st half and 2nd half
  int p;
} L[N];                 // will store the temporary positions
bool comp(entry a, entry b){
  return a.nr[0] == b.nr[0] ? (a.nr[1] < b.nr[1]) : (a.nr[0] < b.nr[0]);
}
void make_array(char* str) {
  int cnt;
  len = strlen(str);
  for(int i=0; i<len; i++)        // Initialising P for single characters
    P[0][i] = str[i] - 'a';                 // in case of small letters str[i] - 'a', 'A' if caps included

  for(cnt = 1, stp=1; cnt>>1 < len; stp++, cnt <<= 1) {        // stp gives level of matrix P,
    // cnt gives the substring size to take
    for(int i=0; i<len; i++) {                                    // i is the starting point of the substring
      L[i].nr[0] = P[stp-1][i];
      L[i].nr[1] = i + cnt < len ? P[stp-1][i+cnt] : -1;          // taking into account the length when calculating the end
      // point of string. Also, -1, like '£' will come first in sorting
      L[i].p = i;                 // pointer to the substring
    }
    sort(L, L + len, comp);                 // ordering the new 2^(k+1) substrings lexographically
    // can be done in linear time using countsort
    for(int i=0; i<len; i++) {
      // equal substrings must be given the same position
      P[stp][L[i].p] = ( (i>0) && (L[i].nr[0] == L[i-1].nr[0]) && (L[i].nr[1] == L[i-1].nr[1]) ) ? P[stp][L[i-1].p] : i;
    }

  }
}
int LCPlen(int x, int y) {
  /*Given 2 suffixes str_x^k (length 2^k) and str_y^k, using the matrix P, we descend from highest k to 0 till we get str_x^k = str_y^k. Thus a pr
   * We increase both x and y by 2^k to see of the prefix extends */
  int k=0, ret=0;
  if(x == y) return len - x;
  for(k= stp-1; k>=0 && x < len && y < len; k--)
    if(P[k][x] == P[k][y])                          // prefix of length 2^k found
      x += 1 << k, y += 1<< k, ret += 1 << k;           // increment x to get the more mathches
  return ret;
}
int LCP(int x, int y) {
  /*Given 2 suffixes str_x^k (length 2^k) and str_y^k, using the matrix P, we descend from highest k to 0 till we get str_x^k = str_y^k. Thus a pr
   * We increase both x and y by 2^k to see of the prefix extends */
  int k=0, ret=0, i;
  if(x == y) {
    i=x;
    for(; i<len; i++)
      putchar(str[i]);
    putchar('\n');
    return len - x;
  }
  for(k= stp-1; k>=0 && x < len && y < len; k--)
    if(P[k][x] == P[k][y])        {               // prefix of length 2^k found
      i=x;
      x += 1 << k, y += 1<< k, ret += 1 << k;           // increment x to get the more mathches
      for(; i<x; i++)
        putchar(str[i]);
    }
  printf("\n");
  return ret;
}
void printSarray() {
  for(int i=0; i<len; i++)        // will print the rank of substr[i]
  {
    //printf("%d %s\n", P[stp-1][i], str + i);
    Sarray[ P[stp-1][i] ] = i;
  }
  for(int i=0; i<len; i++)        // will print the suffix array
  {
    //printf("%d %s\n", Sarray[i], str + Sarray[i]);
  }
}
void printLCParray() {
  for(int i=1; i<len; i++) {
    LCParray[i] = LCPlen(Sarray[i-1], Sarray[i]);
    cout << LCParray[i] << " ";
  }
```

```
  cout << "\n";
}
```

33 Ternary Search

```
int ternary_search(int l, int r) {
        int ans;
        int m1, m2;
        while(r - l >= 3) {                 // (r - l >= EPS) here EPS = 3, for less values, it may go into an infinite loop
                m1 = l + (r-l) / 3;
                m2 = r - (r-l) / 3;
                if(f(m1) < f(m2)) l = m1;         // in case of mimimum change to if(f(m1) > f(m2))
                else r = m2;
        }
        ans = f(l++);
        while(l <= r) ans = max(ans, f(l)),l++;         // if minimum ,change to min(ans, f(l))
        return ans;
}
```

34 Vimrc

```
inoremap {      {}<Left>
inoremap {<CR>  {<CR>}<Esc>O
inoremap {{     {
inoremap {}     {}
inoremap        (   ()<Left>
inoremap <expr> )   strpart(getline('.'), col('.')-1, 1) == ")" ? "\<Right>" : ")"
inoremap        "   ""<Left>
inoremap        '   ''<Left>
inoremap        [   []<Left>
inoremap <expr> ]   strpart(getline('.'), col('.')-1, 1) == "]" ? "\<Right>" : "]"
set shiftwidth=2
set expandtab
set softtabstop=2
set shiftround
set nu
set scrolloff=999
set hlsearch
```

35 CRT

1. To find, $a \equiv a_i(modn_i)$ modulo $n = \prod_{i=1}^{k} n_i$ n_i's are coprime
2. find $m_i = \frac{n}{n_i}$
3. $c_i = m_i(m_i^{-1} modn_i)$
4. $a = \sum a_i c_i (modn)$

36 FFT

```
struct cpx {
  cpx(){}
  cpx(double aa):a(aa){}
  cpx(double aa, double bb):a(aa),b(bb){}
  double a;
  double b;
  double modsq(void) const {
    return a * a + b * b;
  }
  cpx bar(void) const {
    return cpx(a, -b);
  }
};
cpx operator +(cpx a, cpx b) {
  return cpx(a.a + b.a, a.b + b.b); }
cpx operator *(cpx a, cpx b) {
  return cpx(a.a * b.a - a.b * b.b, a.a * b.b + a.b * b.a); }
cpx operator /(cpx a, cpx b) {
  cpx r = a * b.bar();
  return cpx(r.a / b.modsq(), r.b / b.modsq());
}
cpx EXP(double theta) {  return cpx(cos(theta),sin(theta)); }
const double two_pi = 4 * acos(0);
// in:     input array
// out:    output array
// step:   {SET TO 1} (used internally)
// size:   length of the input/output {MUST BE A POWER OF 2}
// dir:    either plus or minus one (direction of the FFT)
// RESULT: out[k] = \sum_{j=0}^{size - 1} in[j] * exp(dir * 2pi * i * j * k / size)
void FFT(cpx *in, cpx *out, int step, int size, int dir) {
  if(size < 1) return;
```

```
  if(size == 1) {
    out[0] = in[0];
    return;
  }
  FFT(in, out, step * 2, size / 2, dir);
  FFT(in + step, out + size / 2, step * 2, size / 2, dir);
  for(int i = 0 ; i < size / 2 ; i++) {
    cpx even = out[i];
    cpx odd = out[i + size / 2];
    out[i] = even + EXP(dir * two_pi * i / size) * odd;
    out[i + size / 2] = even + EXP(dir * two_pi * (i + size / 2) / size) * odd;
  }
}
// Usage:
// f[0...N-1] and g[0..N-1] are numbers
// Want to compute the convolution h, defined by
// h[n] = sum of f[k]g[n-k] (k = 0, ..., N-1).
// Here, the index is cyclic; f[-1] = f[N-1], f[-2] = f[N-2], etc.
// Let F[0...N-1] be FFT(f), and similarly, define G and H.
// The convolution theorem says H[n] = F[n]G[n] (element-wise product).
// To compute h[] in O(N log N) time, do the following:
//   1. Compute F and G (pass dir = 1 as the argument).
//   2. Get H by element-wise multiplying F and G.
//   3. Get h by taking the inverse FFT (use dir = -1 as the argument)
//      and *dividing by N*. DO NOT FORGET THIS SCALING FACTOR.

int main(void)
{
  printf("If rows come in identical pairs, then everything works.\n");
  cpx a[8] = {0, 1, cpx(1,3), cpx(0,5), 1, 0, 2, 0};
  cpx b[8] = {1, cpx(0,-2), cpx(0,1), 3, -1, -3, 1, -2};
  cpx A[8];
  cpx B[8];
  FFT(a, A, 1, 8, 1);
  FFT(b, B, 1, 8, 1);
  for(int i = 0 ; i < 8 ; i++) printf("%7.2lf%7.2lf", A[i].a, A[i].b); printf("\n");
  for(int i = 0 ; i < 8 ; i++) {
    cpx Ai(0,0);
    for(int j = 0 ; j < 8 ; j++) { Ai = Ai + a[j] * EXP(j * i * two_pi / 8); }
    printf("%7.2lf%7.2lf", Ai.a, Ai.b);
  }
  printf("\n");
  cpx AB[8];
  for(int i = 0 ; i < 8 ; i++)
    AB[i] = A[i] * B[i];
  cpx aconvb[8];
  FFT(AB, aconvb, 1, 8, -1);
  for(int i = 0 ; i < 8 ; i++)
    aconvb[i] = aconvb[i] / 8;
  for(int i = 0 ; i < 8 ; i++){  printf("%7.2lf%7.2lf", aconvb[i].a, aconvb[i].b); }
  printf("\n");
  for(int i = 0 ; i < 8 ; i++) {
    cpx aconvbi(0,0);
    for(int j = 0 ; j < 8 ; j++) {  aconvbi = aconvbi + a[j] * b[(8 + i - j) % 8]; }
    printf("%7.2lf%7.2lf", aconvbi.a, aconvbi.b);
  }
  printf("\n");
  return 0;
}
```

37 Gauss

```
class MixingColors:
 def minColors(self, colors):
    colors = list(colors)
    n = len(colors)

    j = 0 #number of top rows to ignore
    for i in range(0, 31):
        # find a row that has 1 in the i-th bit:
        k = j
        while (k < n) and ( (colors[k] & (1<<i)) == 0 ):
            k += 1
        # if at least one of those rows exists:
        if k != n:
            # swap row k and j
            (colors[j], colors[k]) = (colors[k], colors[j])
            # zero this column in the remaining rows (using xor)
            for k in range(j + 1, n):
                if (colors[k] & (1<<i)) != 0:
                    colors[k] = colors[k] ^ colors[j]
            # ignore one more row
            j += 1
    return sum(c > 0 for c in colors ) # sum of non-zero rows
```

Theoretical Computer Science Cheat Sheet

Definitions

$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)$ $\forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0$ $\forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim\limits_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\left[{n \atop k}\right]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\left\{{n \atop k}\right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\left\langle{n \atop k}\right\rangle$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ with k ascents.
$\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n} \frac{1}{i}, \quad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \quad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\displaystyle \binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\displaystyle \sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\displaystyle \binom{n}{k} = \binom{n}{n-k}$,

4. $\displaystyle \binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\displaystyle \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\displaystyle \binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\displaystyle \sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\displaystyle \sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\displaystyle \sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\displaystyle \binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\displaystyle \left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

12. $\displaystyle \left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\displaystyle \left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

14. $\displaystyle \left[{n \atop 1}\right] = (n-1)!$, **15.** $\displaystyle \left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\displaystyle \left[{n \atop n}\right] = 1$, **17.** $\displaystyle \left[{n \atop k}\right] \ge \left\{{n \atop k}\right\}$,

18. $\displaystyle \left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\displaystyle \left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\displaystyle \sum_{k=0}^{n}\left[{n \atop k}\right] = n!$, **21.** $\displaystyle C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\displaystyle \left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$, **23.** $\displaystyle \left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-1-k}\right\rangle$, **24.** $\displaystyle \left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$,

25. $\displaystyle \left\langle{0 \atop k}\right\rangle = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\displaystyle \left\langle{n \atop 1}\right\rangle = 2^n - n - 1$, **27.** $\displaystyle \left\langle{n \atop 2}\right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $\displaystyle x^n = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{x+k}{n}$, **29.** $\displaystyle \left\langle{n \atop m}\right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $\displaystyle m!\left\{{n \atop m}\right\} = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{k}{n-m}$,

31. $\displaystyle \left\langle{n \atop m}\right\rangle = \sum_{k=0}^{n}\left\{{n \atop k}\right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\displaystyle \left\langle\!\left\langle{n \atop 0}\right\rangle\!\right\rangle = 1$, **33.** $\displaystyle \left\langle\!\left\langle{n \atop n}\right\rangle\!\right\rangle = 0 \quad \text{for } n \ne 0$,

34. $\displaystyle \left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle{n-1 \atop k}\right\rangle\!\right\rangle + (2n-1-k)\left\langle\!\left\langle{n-1 \atop k-1}\right\rangle\!\right\rangle$, **35.** $\displaystyle \sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\displaystyle \left\{{x \atop x-n}\right\} = \sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\displaystyle \left\{{n+1 \atop m+1}\right\} = \sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k}$,

Theoretical Computer Science Cheat Sheet

Identities Cont.

38. $\left[{n+1 \atop m+1}\right] = \sum_k \left[{n \atop k}\right]\binom{k}{m} = \sum_{k=0}^{n}\left[{k \atop m}\right] n^{\underline{n-k}} = n!\sum_{k=0}^{n}\frac{1}{k!}\left[{k \atop m}\right]$,

39. $\left[{x \atop x-n}\right] = \sum_{k=0}^{n}\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle\binom{x+k}{2n}$,

40. $\left\{{n \atop m}\right\} = \sum_k \binom{n}{k}\left\{{k+1 \atop m+1}\right\}(-1)^{n-k}$,

41. $\left[{n \atop m}\right] = \sum_k \left[{n+1 \atop k+1}\right]\binom{k}{m}(-1)^{m-k}$,

42. $\left\{{m+n+1 \atop m}\right\} = \sum_{k=0}^{m} k\left\{{n+k \atop k}\right\}$,

43. $\left[{m+n+1 \atop m}\right] = \sum_{k=0}^{m} k(n+k)\left[{n+k \atop k}\right]$,

44. $\binom{n}{m} = \sum_k \left\{{n+1 \atop k+1}\right\}\left[{k \atop m}\right](-1)^{m-k}$,

45. $(n-m)!\binom{n}{m} = \sum_k \left[{n+1 \atop k+1}\right]\left\{{k \atop m}\right\}(-1)^{m-k}$, for $n \ge m$,

46. $\left\{{n \atop n-m}\right\} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left[{m+k \atop k}\right]$,

47. $\left[{n \atop n-m}\right] = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left\{{m+k \atop k}\right\}$,

48. $\left\{{n \atop \ell+m}\right\}\binom{\ell+m}{\ell} = \sum_k \left\{{k \atop \ell}\right\}\left\{{n-k \atop m}\right\}\binom{n}{k}$,

49. $\left[{n \atop \ell+m}\right]\binom{\ell+m}{\ell} = \sum_k \left[{k \atop \ell}\right]\left[{n-k \atop m}\right]\binom{n}{k}$.

Trees

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:
$$\sum_{i=1}^{n} 2^{-d_i} \le 1,$$
and equality holds only if every internal node has 2 sons.

Recurrences

Master method:
$$T(n) = aT(n/b) + f(n), \quad a \ge 1, b > 1$$
If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then
$$T(n) = \Theta(n^{\log_b a}).$$
If $f(n) = \Theta(n^{\log_b a})$ then
$$T(n) = \Theta(n^{\log_b a}\log_2 n).$$
If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \le cf(n)$ for large n, then
$$T(n) = \Theta(f(n)).$$
Substitution (example): Consider the following recurrence
$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$
Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have
$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$
Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get
$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$
Substituting we find
$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$
which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.
Summing factors (example): Consider the following recurrence
$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$
Rewrite so that all terms involving T are on the left side
$$T(n) - 3T(n/2) = n.$$
Now expand the recurrence, and choose a factor which makes the left side "telescope"
$$\begin{aligned}
1\big(T(n) - 3T(n/2) &= n\big)\\
3\big(T(n/2) - 3T(n/4) &= n/2\big)\\
\vdots \quad \vdots \quad &\vdots\\
3^{\log_2 n - 1}\big(T(2) - 3T(1) &= 2\big)
\end{aligned}$$
Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get
$$\sum_{i=0}^{m-1}\frac{n}{2^i}3^i = n\sum_{i=0}^{m-1}\left(\tfrac{3}{2}\right)^i.$$
Let $c = \frac{3}{2}$. Then we have
$$\begin{aligned}
n\sum_{i=0}^{m-1} c^i &= n\left(\frac{c^m - 1}{c - 1}\right)\\
&= 2n(c^{\log_2 n} - 1)\\
&= 2n(c^{(k-1)\log_c n} - 1)\\
&= 2n^k - 2n,
\end{aligned}$$
and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider
$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$
Note that
$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$
Subtracting we find
$$\begin{aligned}
T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j\\
&= T_i.
\end{aligned}$$
And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i.
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i.

Example:
$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$
Multiply and sum:
$$\sum_{i\ge 0} g_{i+1}x^i = \sum_{i\ge 0} 2g_i x^i + \sum_{i \ge 0} x^i.$$
We choose $G(x) = \sum_{i \ge 0} x^i g_i$. Rewrite in terms of $G(x)$:
$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i\ge 0} x^i.$$
Simplify:
$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$
Solve for $G(x)$:
$$G(x) = \frac{x}{(1-x)(1-2x)}.$$
Expand this using partial fractions:
$$\begin{aligned}
G(x) &= x\left(\frac{2}{1-2x} - \frac{1}{1-x}\right)\\
&= x\left(2\sum_{i\ge 0} 2^i x^i - \sum_{i \ge 0} x^i\right)\\
&= \sum_{i \ge 0}(2^{i+1} - 1)x^{i+1}.
\end{aligned}$$
So $g_i = 2^i - 1$.

Dependencies: -

```
long long power(long long n, long long k, long long m = LLONG_MAX) {
    long long ret = 1;
    while (k) {
        if (k & 1) ret = (ret * n) % m;
        n = (n * n) % m;
        k >>= 1;
    }
    return ret;
}
```

Great Common Divisor

a와 b의 최대공약수를 구한다.
Dependencies: -

```
long long gcd(long long a, long long b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
```

Extended GCD

ac + bd = gcd(a, b)가 되는 (c, d)를 찾는다.
Dependencies: -

```
pair<long long, long long> extended_gcd(long long a, long long b) {
    if (b == 0) return make_pair(1, 0);
    pair<long long, long long> t = extended_gcd(b, a % b);
    return make_pair(t.second, t.first - t.second * (a / b));
}
```

Modular Inverse

ax = gcd(a, m) (mod m)가 되는 x를 찾는다.
Dependencies: extended_gcd(a, b)

```
long long modinverse(long long a, long long m) {
    return (extended_gcd(a, m).first % m + m) % m;
}
```

Chinese Remainder Theorem

x = a (mod n)가 되는 x를 찾는다.
Dependencies: gcd(a, b), modinverse(a, m)

```
long long chinese_remainder(long long *a, long long *n, int size) {
    if (size == 1) return *a;
    long long tmp = modinverse(n[0], n[1]);
    long long tmp2 = (tmp * (a[1] - a[0]) % n[1] + n[1]) % n[1];
    long long ora = a[1];
    long long tgcd = gcd(n[0], n[1]);
    a[1] = a[0] + n[0] / tgcd * tmp2;
    n[1] *= n[0] / tgcd;
    long long ret = chinese_remainder(a + 1, n + 1, size - 1);
    n[1] /= n[0] / tgcd;
    a[1] = ora;
    return ret;
}
```

Binomial Calculation

nCm의 값을 구한다.
Dependencies: -
파스칼의 삼각형을 이용하거나, 미리 계산된 값을 가져오도록 이 함수를 수정하면
lucas_theorem, catalan_number 함수의 성능을 향상시킬 수 있다.

```
long long binomial(int n, int m) {
    if (n > m || n < 0) return 0;
    long long ans = 1, ans2 = 1;
    for (int i = 0 ; i < m ; i++) {
        ans *= n - i;
        ans2 *= i + 1;
    }
    return ans / ans2;
}
```

Lucas Theorem

nCm mod p의 값을 구한다.
Dependencies: binomial(n, m)
n, m은 문자열로 주어지는 정수이다. p는 소수여야 한다.

```
int lucas_theorem(const char *n, const char *m, int p) {
    vector<int> np, mp;
    int i;
    for (i = 0 ; n[i] ; i++) {
        if (n[i] == '0' && np.empty()) continue;
        np.push_back(n[i] - '0');
    }
    for (i = 0 ; m[i] ; i++) {
        if (m[i] == '0' && mp.empty()) continue;
        mp.push_back(m[i] - '0');
```

```
    }

    int ret = 1;
    int ni = 0, mi = 0;
    while (ni < np.size() || mi < mp.size()) {
        int nmod = 0, mmod = 0;
        for (i = ni ; i < np.size() ; i++) {
            if (i + 1 < np.size())
                np[i + 1] += (np[i] % p) * 10;
            else
                nmod = np[i] % p;
            np[i] /= p;
        }
        for (i = mi ; i < mp.size() ; i++) {
            if (i + 1 < mp.size())
                mp[i + 1] += (mp[i] % p) * 10;
            else
                mmod = mp[i] % p;
            mp[i] /= p;
        }
        while (ni < np.size() && np[ni] == 0) ni++;
        while (mi < mp.size() && mp[mi] == 0) mi++;
        ret = (ret * binomial(nmod, mmod)) % p;
    }
    return ret;
}
```

Catalan Number

Dependencies: binomial(n, m)

```
long long catalan_number(int n) {
    return binomial(n * 2, n) / (n + 1);
}
```

Euler's Totient Function

phi(n), n 이하의 양수 중 n과 서로 소인 것의 개수를 구한다.
Dependencies: -

```
// phi(n) = (p_1 - 1) * p_1 ^ (k_1 - 1) * (p_2 - 1) * p_2 ^ (k_2-1)
long long euler_totient2(long long n, long long ps) {
    for (long long i = ps ; i * i <= n ; i++) {
        if (n % i == 0) {
            long long p = 1;
            while (n % i == 0) {
                n /= i;
                p *= i;
            }
            return (p - p / i) * euler_totient2(n, i + 1);
        }
        if (i > 2) i++;
    }
    return n - 1;
}
long long euler_totient(long long n) {
    return euler_totient2(n, 2);
}
```

Matrix Inverse

Dependencies: -

```
inline bool eq(double a, double b) {
    static const double eps = 1e-9;
    return fabs(a - b) < eps;
}

// returns empty vector if fails
vector<vector<double> > mat_inverse(vector<vector<double> > matrix, int n) {
    int i, j, k;
    vector<vector<double> > ret;
    ret.resize(n);
    for (i = 0 ; i < n ; i++) {
        ret[i].resize(n);
        for (j = 0 ; j < n ; j++)
            ret[i][j] = 0;
        ret[i][i] = 1;
    }
    for (i = 0 ; i < n ; i++) {
        if (eq(matrix[i][i],0)) {
            for (j = i + 1 ; j < n ; j++) {
                if (!eq(matrix[j][i], 0)) {
                    for (k = 0 ; k < n ; k++) {
                        matrix[i][k] += matrix[j][k];
                        ret[i][k] += ret[j][k];
                    }
                    break;
                }
            }
            if (j == n) {
                ret.clear();
                return ret;
```

```
            }
        }
        double tmp = matrix[i][i];
        for (k = 0 ; k < n ; k++) {
            matrix[i][k] /= tmp;
            ret[i][k] /= tmp;
        }
        for (j = 0 ; j < n ; j++) {
            if (j == i) continue;
            tmp = matrix[j][i];
            for (k = 0 ; k < n ; k++) {
                matrix[j][k] -= matrix[i][k] * tmp;
                ret[j][k] -= ret[i][k] * tmp;
            }
        }
    }
    return ret;
}
```

Modular Matrix Inverse

Dependencies: modinverse(a, m)

```
// returns empty vector if fails
vector<vector<long long> > mat_inverse(vector<vector<long long> > matrix, int n,
long long mod) {
    int i, j, k;
    vector<vector<long long> > ret;
    ret.resize(n);
    for (i = 0 ; i < n ; i++) {
        ret[i].resize(n);
        for (j = 0 ; j < n ; j++)
            ret[i][j] = 0;
        ret[i][i] = 1 % mod;
    }
    for (i = 0 ; i < n ; i++) {
        if (matrix[i][i] == 0) {
            for (j = i + 1 ; j < n ; j++) {
                if (matrix[j][i] != 0) {
                    for (k = 0 ; k < n ; k++) {
                        matrix[i][k] = (matrix[i][k] + matrix[j][k]) % mod;
                        ret[i][k] = (ret[i][k] + ret[j][k]) % mod;
                    }
                    break;
                }
            }
            if (j == n) {
                ret.clear();
                return ret;
            }
        }
        long long tmp = modinverse(matrix[i][i], mod);
        for (k = 0 ; k < n ; k++) {
            matrix[i][k] = (matrix[i][k] * tmp) % mod;
            ret[i][k] = (ret[i][k] * tmp) % mod;
        }
        for (j = 0 ; j < n ; j++) {
            if (j == i) continue;
            tmp = matrix[j][i];
            for (k = 0 ; k < n ; k++) {
                matrix[j][k] -= matrix[i][k] * tmp;
                matrix[j][k] = (matrix[j][k] % mod + mod) % mod;
                ret[j][k] -= ret[i][k] * tmp;
                ret[j][k] = (ret[j][k] % mod + mod) % mod;
            }
        }
    }
    return ret;
}
```

Matrix Determinants

Dependencies: -

```
double mat_det(vector<vector<double> > matrix, int n) {
    int i, j, k;
    double ret = 1;
    for (i = 0 ; i < n ; i++) {
        if (eq(matrix[i][i], 0)) {
            for (j = i + 1 ; j < n ; j++) {
                if (!eq(matrix[j][i], 0)) {
                    for (k = 0 ; k < n ; k++)
                        matrix[i][k] += matrix[j][k];
                    break;
                }
            }
            if (j == n)
                return 0;
        }
        double tmp = matrix[i][i];
        for (k = 0 ; k < n ; k++)
            matrix[i][k] /= tmp;
```

```
        ret *= tmp;
        for (j = 0 ; j < n ; j++) {
            if (j == i) continue;
            tmp = matrix[j][i];
            for (k = 0 ; k < n ; k++)
                matrix[j][k] -= matrix[i][k] * tmp;
        }
    }
    return ret;
}
```

Kirchhoff’s Theorem

주어진 그래프에서 가능한 신장트리의 경우의 수를 구한다.
Dependencies: mat_det(matrix, n)

```
long long count_spantree(vector<int> graph[], int size) {
    int i, j;
    vector<vector<double> > matrix(size - 1);
    for (i = 0 ; i < size - 1 ; i++) {
        matrix[i].resize(size - 1);
        for (j = 0 ; j < size - 1 ; j++)
            matrix[i][j] = 0;
        for (j = 0 ; j < graph[i].size() ; j++) {
            if (graph[i][j] < size - 1) {
                matrix[i][graph[i][j]]--;
                matrix[i][i]++;
            }
        }
    }
    return (long long)(mat_det(matrix, size - 1) + 0.5);
}
```

Gaussian Elimination

gaussian::run(size_eq, size_var, A, B, C);
A는 1차원 배열의 꼴로 주어지는 2차원 행렬이다. 배열 C의 값을 채워 넣는 루틴은 별도로 구현하라. val_t로 double을 사용할 경우 abs 함수의 구현을 적절히 수정하라.

```
#include <algorithm>
using namespace std;

long long gcd(long long a, long long b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

struct rational {
    long long p, q;

    void red() {
        if (q < 0) {
            p *= -1;
            q *= -1;
        }
        long long t = gcd((p >= 0 ? p : -p), q);
        p /= t;
        q /= t;
    }

    rational() {}
    rational(long long p_): p(p_), q(1) {}
    rational(long long p_, long long q_): p(p_), q(q_) { red(); }

    bool operator==(const rational& rhs) const {
        return p == rhs.p && q == rhs.q;
    }
    bool operator!=(const rational& rhs) const {
        return p != rhs.p || q != rhs.q;
    }
    bool operator<(const rational& rhs) const {
        return p * rhs.q < rhs.p * q;
    }
    const rational operator+(const rational& rhs) const {
        return rational(p * rhs.q + q * rhs.p, q * rhs.q);
    }
    const rational operator-(const rational& rhs) const {
        return rational(p * rhs.q - q * rhs.p, q * rhs.q);
    }
    const rational operator*(const rational& rhs) const {
        return rational(p * rhs.p, q * rhs.q);
    }
    const rational operator/(const rational& rhs) const {
        return rational(p * rhs.q, q * rhs.p);
    }
};

namespace gaussian
{
```