New York University ICPC Team Notebook (2018-19)

Contents

1 General

1.1 General Template

```
#include <bits/stdc++.h>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <sstream>
#include <queue>
#include <deque>
#include <bitset>
#include <iterator>
#include <list>
#include <stack>
#include <unordered_map>
#include <unordered_set>
#include <map>
#include <array>
#include <set>
#include <complex>
#include <functional>
#include <numeric>
#include <utility>
#include <limits>
#include <time.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <climits>
#include <assert.h>
#include <chrono>

using namespace std;

typedef long long ll;
typedef long double ld;
typedef complex<ld> cd;

#define EPS 1e-14
#define PI 3.14159265358979323846264338327950
#define MOD 1000000007
#define INFLL (ll)1e18 //Long long infinity
#define INF (int)1e9 //Int infinity
#define MX 100005

typedef pair<int, int> pi;
typedef pair<ll,ll> pl;
typedef pair<ld,ld> pd;

typedef vector<int> vi;
typedef vector<ld> vd;
typedef vector<ll> vl;
typedef vector<pi> vpi;
typedef vector<pl> vpl;
```

```cpp
typedef vector<cd> vcd;
typedef vector<vi> vvi;
typedef vector<vl> vvl;
typedef vector<vd> vvd;
typedef vector<string> vstr;

#define us unordered_set
#define um unordered_map
#define bs bitset

typedef vector<um<int, int>> graph;

#define RANGE(i,a,b,d) for (int i=min((int)a,(int)b); i<max((int)a,(int)b); i+=d)
#define RRANGE(i,a,b,d) for (int i=max((int)a,(int)b); i>min((int)a,(int)b); i+=d)
#define FOR(i,a,b) RANGE(i,a,b,1)
#define RFOR(i,a,b) RRANGE(i,a,b,-1)
#define REP(i,s) FOR(i,0,s)
#define RREP(i,s) RFOR(i,s-1,-1)
#define FORIT(it,l) for (auto it = l.begin(); it != l.end(); it++)
#define EACH(x,v) for (auto &x : v)

#define sz(x) (int)(x).size()
#define len(x) (int)sizeof(x)/sizeof(*x)
#define mp make_pair
#define pb push_back
#define F first
#define S second
#define lb lower_bound
#define ub upper_bound
#define all(x) x.begin(), x.end()
#define contains(m,x) (m.find(x) != m.end()) // check if an element in a map
#define isIn(S, s) (S.find(s) != string::npos) // check if substring

#define pv(v) EACH(x, v) cout << x << " "; cout << endl; // print vector/array
#define pvv(vv) EACH(xx, vv){pv(xx);} // print 2-d vector/2-d array
#define pm(m) EACH(x, m) cout << x.F << ":" << x.S << " "; cout << endl; //print map/lookup table

namespace std{
  // Used for hashing pair
  template <> struct hash<pi> {
    inline size_t operator()(const pi &v) const {
      hash<int> int_hasher;
      return int_hasher(v.F) ^ int_hasher(v.S);
    }
  };
};

// This is min queue
template<class T> using pqg = priority_queue<T, vector<T>, greater<T>>;

template <typename T>
void print(T t){
  cout << t << endl;
}

//Python style printing
template<typename T, typename... Args>
void print(T t, Args... args){
  cout << t << " ";
  print(args...);
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);
  return 0;
}
```

1.2 Makefile

```makefile
# Reference from http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
# Also https://codeforces.com/blog/entry/15547
CC = g++
CFLAGS = -I. -Wall -Wextra -pedantic -std=c++14 -O2 -Wshadow -Wformat=2 -Wfloat-equal -Wconversion -
      Wcast-qual -Wcast-align -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC -D_FORTIFY_SOURCE=2 -
      fsanitize=address -fsanitize=undefined -fstack-protector
OBJ = main.o

%.o: %.cpp
	$(CC) -c -o $(OBJ) $< $(CFLAGS)

main: $(OBJ)
	$(CC) -o main $^ $(CFLAGS)

clean:
	rm *.o
	rm main

all: run

run: main
	./main < input.txt

.PHONY: all run
```

2 Data Structures

2.1 Binary Index Tree

```cpp
/**
 * Description: 1D range sum query with point update
 */

// Indexed start at 1
template < class T, int SZ > struct BIT {
  T bit[SZ + 1];

  BIT() { memset(bit, 0, sizeof bit);}

  void upd(int k, T val) { // add val to index k
    for (; k <= SZ; k += (k & -k)) bit[k] += val;
  }

  T query(int k) {
    T temp = 0;
    for (; k > 0; k -= (k & -k)) temp += bit[k];
    return temp;
  }
  T query(int l, int r) {// range query [l,r]
    return query(r) - query(l - 1);
  }
};
```

2.2 2D Binary Index Tree

```cpp
// Indexed start at 1
template<class T, int SZ> struct BIT2D {
    BIT<T,SZ> bit[SZ+1];
    void upd(int X, int Y, T val) {
        for (; X <= SZ; X += (X&-X)) bit[X].upd(Y,val);
    }
    T query(int X, int Y) {
        T ans = 0;
        for (; X > 0; X -= (X&-X)) ans += bit[X].query(Y);
        return ans;
    }
    // X1 <= X2 and Y1 <= Y2
    T query(int X1, int X2, int Y1, int Y2) {
        return query(X2,Y2)-query(X1-1,Y2)-query(X2,Y1-1)+query(X1-1,Y1-1);
    }
};
```

2.3 Union-Find

```cpp
/**
 * Description: Disjoint Set Union
 */

template < int SZ > struct DSU {
  int par[SZ], sz[SZ];
  DSU() {
    REP(i, SZ) par[i] = i, sz[i] = 1;
  }

  int get(int x) { // path compression
    if (par[x] != x) par[x] = get(par[x]);
    return par[x];
  }

  bool unite(int x, int y) { // union-by-rank
    x = get(x), y = get(y);
    if (x == y) return 0;
    if (sz[x] < sz[y]) swap(x, y);
    sz[x] += sz[y], par[y] = x;
```

```
    return 1;
  }
};
```

2.4 KD Tree

```
struct kd_tree{
  struct kd_node {
        int axis;
    ld val;
    array<ld,2> p;
        kd_node *left, *right;
  } *root;

  struct cmp_points {
        int axis;
        cmp_points(){}
        cmp_points(int x): axis(x) {}
        bool operator () (const array<ld,2>& a, const array<ld,2>& b) const {
      return a[axis]<b[axis];
        }
  };

  void build(vector<array<ld,2>>& points){
    build_tree(points, root, 0, sz(points)-1, 0);
  }

  void build_tree(vector<array<ld,2>>& points, kd_node*& node, int from, int to, int axis) {
        if (from>to) {
                node=NULL;
                return;
        }
    node=new kd_node();
        if (from==to) {
                node->p=points[from];
                node->left=NULL;
                node->right=NULL;
                return;
        }
        int mid=(from+to)/2;
        nth_element(points.begin()+from,points.begin()+mid,points.begin()+to+1,cmp_points(axis));
        node->val=points[mid][axis];
        node->axis=axis;
        build_tree(points,node->left,from,mid,axis^1);
        build_tree(points,node->right,mid+1,to,axis^1);
  }

  ld dist(array<ld,2>& p, array<ld,2>& q){
    cd pp = cd(p[0],p[1]);
    cd qq = cd(q[0],q[1]);
    return abs(pp-qq);
  }

  ld nn(array<ld,2>& q){
    ld ans = INFLL;
    nnei(root, q, ans);
    return ans;
  }

  void nnei(kd_node*& node, array<ld,2>& q, ld &ans) {
        if(node==NULL) return;
        if(node->left==NULL && node->right==NULL) {
                if(q!=node->p) ans=min(ans,dist(node->p,q));
                return;
        }

        if(q[node->axis]<=node->val) {
                nnei(node->left,q,ans);
                if(q[node->axis]+ans>=node->val) nnei(node->right,q,ans);
        }

        else {
                nnei(node->right,q,ans);
                if(q[node->axis]-ans<=node->val) nnei(node->left,q,ans);
        }
  }
};
```

2.5 Sorted Set

```
// C++ program to demonstrate the
// ordered set in GNU C++
#include <iostream>
using namespace std;

// Header files, namespaces,
// macros as defined above
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type,less<int>, rb_tree_tag,tree_order_statistics_node_update>

// Driver program to test above functions
int main(){
    // Ordered set declared with name o_set
    ordered_set o_set;

    // insert function to insert in
    // ordered set same as SET STL
    o_set.insert(5);
    o_set.insert(1);
    o_set.insert(2);

    // Finding the SECOND smallest element
    // in the set using * because
    //  find_by_order returns an iterator
    cout << *(o_set.find_by_order(1))
         << endl;

    // Finding the number of elements
    // strictly less than k=4
    cout << o_set.order_of_key(4)
         << endl;

    // Finding the count of elements less
    // than or equal to 4 i.e. striclty less
    // than 5 if integers are present
    cout << o_set.order_of_key(5)
         << endl;

    // Deleting 2 from the set if it exists
    if (o_set.find(2) != o_set.end())
        o_set.erase(o_set.find(2));

    // Now after deleting 2 from the set
    // Finding the SECOND smallest element in the set
    cout << *(o_set.find_by_order(1))
         << endl;

    // Finding the number of
    // elements strictly less than k=4
    cout << o_set.order_of_key(4)
         << endl;
    return 0;
}
```

2.6 Lazy Segment Tree

```
/**
 * Description: 1D range update, range query
 */

template < class T, int SZ > struct LazySegTree {
  T sum[2 * SZ], mn[2 * SZ], lazy[2 * SZ]; // set SZ to large enough power of 2

  //remember to change initial value appropriately to the operation
  // 0 for sum
  // INF for min, -INF for max
  LazySegTree() {
    memset(sum, 0, sizeof sum);
    memset(mn, 0, sizeof mn);
    memset(lazy, 0, sizeof lazy);
  }

  void push(int ind, int L, int R) {
    sum[ind] += (R - L + 1) * lazy[ind];
    mn[ind] += lazy[ind];
    if (L != R) lazy[2 * ind] += lazy[ind], lazy[2 * ind + 1] += lazy[ind];
    lazy[ind] = 0;
  }

  void pull(int ind) {
    sum[ind] = sum[2 * ind] + sum[2 * ind + 1];
    mn[ind] = min(mn[2 * ind], mn[2 * ind + 1]);
  }

  // If A[i]=val then set sum[i^SZ]=val and mn[i^SZ]=val
  // call this after you set the initial value of array
  void build() {
```

```
    RREP(i, SZ) pull(i);
  }

  T qsum(int lo, int hi, int ind = 1, int L = 0, int R = SZ - 1) {
    push(ind, L, R);
    if (lo > R || L > hi) return 0;
    if (lo <= L && R <= hi) return sum[ind];
    int M = (L + R) / 2;
    return qsum(lo, hi, 2 * ind, L, M) + qsum(lo, hi, 2 * ind + 1, M + 1, R);
  }

  T qmin(int lo, int hi, int ind = 1, int L = 0, int R = SZ - 1) {
    push(ind, L, R);
    if (lo > R || L > hi) return INF;
    if (lo <= L && R <= hi) return mn[ind];

    int M = (L + R) / 2;
    return min(qmin(lo, hi, 2 * ind, L, M), qmin(lo, hi, 2 * ind + 1, M + 1, R));
  }

  void upd(int lo, int hi, ll inc, int ind = 1, int L = 0, int R = SZ - 1) {
    push(ind, L, R);
    if (hi < L || R < lo) return;
    if (lo <= L && R <= hi) {
      lazy[ind] = inc;
      push(ind, L, R);
      return;
    }

    int M = (L + R) / 2;
    upd(lo, hi, inc, 2 * ind, L, M);
    upd(lo, hi, inc, 2 * ind + 1, M + 1, R);
    pull(ind);
  }
};
```

2.7 Link Cut Tree

```
using namespace splayTree;
template<int SZ> struct LCT {
    ps PS[SZ];
    LCT () { REP(i,SZ) PS[i] = new snode(i); }

    void dis(ps x, int d) {
        ps y = x->c[d];
        if (x) x->c[d] = NULL, recalc(x);
        if (y) y->p = NULL, y->pp = x;
    }

    void con(ps x, int d) { setLink(x->pp,x,d); x->pp = NULL; }

    void setPref(ps x) { splay(x->pp), dis(x->pp,1), con(x,1); splay(x); }

    ps access(ps x) { // x is brought to the root of auxiliary tree
        dis(splay(x),1);
        while (x->pp) setPref(x);
        return x;
    }

    //////// UPDATES

    ps makeRoot(ps v) { access(v)->flip = 1; return access(v); } // make new root

    void link(ps v, ps w) { // v is root of its tree, w is not in the same tree with v
        access(w)->pp = makeRoot(v);
        con(w,0);
    }

    void cut(ps x) { // cut link between x and its parent i.e x is not a root
        ps y = access(x)->c[0];
        dis(x,0); y->pp = NULL;
    }

    //////// QUERIES

    int getDepth(ps v) { access(v); return getsz(v->c[0]); }

    int getRoot(ps v) { return getExtreme(access(v),0)->val; }

    int lca(ps x, ps y) {
        ps root = getExtreme(access(y),0);

        dis(splay(x),1);
        auto z = getExtreme(x,0);
        if (z == root) return x->val;
        splay(x);

        while (x->pp) {
            auto z = getExtreme(splay(x->pp),0);
            if (z == root) return x->pp->val;
            setPref(x);
        }

        return -1;
    }
};
```

2.8 Map/Set

```
struct cmp {
  bool operator()(const int & l,
    const int & r) const {
    return l < r; // increasing order
  }
};

int main() {
  set < int, cmp > s;
  map < int, int, cmp > m;
  s.insert(3);
  s.insert(10);
  m[0] = 2;
  m[1] = 3;
  auto lm = m.rbegin(); // last iterator
  print(lm - > F, ":", lm - > S);
  auto fm = m.begin(); // first iterator
  print(fm - > F, ":", fm - > S);

  int smallest = * s.begin(); // smallest int
  int biggest = * s.rbegin(); // biggest int
  print(smallest, biggest);
  return 0;
}

/*
1 : 3
0 : 2
3 10
*/
```

2.9 Segment Tree

```
template < class T, int SZ > struct Seg {
  T seg[2 * SZ], MN = 0;

  Seg() {
    memset(seg, 0, sizeof seg);
  }

  T comb(T a, T b) {
    return a + b;
  } // easily change this to min or max

  void upd(int p, T value) { // set value at position p
    for (seg[p += SZ] = value; p > 1; p >>= 1)
      seg[p >> 1] = comb(seg[(p | 1) ^ 1], seg[p | 1]); // non-commutative operations
  }

  // If A[i]=val then set seg[i^SZ]=val
  // call this after you set the initial value of array
  void build() {
    RREP(i, SZ) seg[i] = comb(seg[2 * i], seg[2 * i + 1]);
  }

  T query(int l, int r) { // sum on interval [l, r]
    T res1 = MN, res2 = MN;
    r++;
    for (l += SZ, r += SZ; l < r; l >>= 1, r >>= 1) {
      if (l & 1) res1 = comb(res1, seg[l++]);
      if (r & 1) res2 = comb(seg[--r], res2);
    }
    return comb(res1, res2);
  }
};
```

2.10 Segment Tree Beats

```
template < int SZ > struct SegTreeBeats {
  int N;
  ll sum[2 * SZ];
  int mx[2][2 * SZ], maxCnt[2 * SZ];

  void pull(int ind) {
    mx[0][ind] = max(mx[0][2 * ind], mx[0][2 * ind + 1]);
    mx[1][ind] = max(mx[1][2 * ind], mx[1][2 * ind + 1]);
    maxCnt[ind] = 0;

    REP(i, 2) {
      if (mx[0][2 * ind ^ i] == mx[0][ind]) maxCnt[ind] += maxCnt[2 * ind ^ i];
      else mx[1][ind] = max(mx[1][ind], mx[0][2 * ind ^ i]);
    }

    sum[ind] = sum[2 * ind] + sum[2 * ind + 1];
  }

  void build(vi & a, int ind = 1, int L = 0, int R = -1) {
    if (R == -1) R += N;
    if (L == R) {
      mx[0][ind] = sum[ind] = a[L];
      maxCnt[ind] = 1;
      mx[1][ind] = -1;
      return;
    }

    int M = (L + R) / 2;
    build(a, 2 * ind, L, M);
    build(a, 2 * ind + 1, M + 1, R);
    pull(ind);
  }

  void push(int ind, int L, int R) {
    if (L == R) return;
    REP(i, 2)
    if (mx[0][2 * ind ^ i] > mx[0][ind]) {
      sum[2 * ind ^ i] -= (ll) maxCnt[2 * ind ^ i] * (mx[0][2 * ind ^ i] - mx[0][ind]);
      mx[0][2 * ind ^ i] = mx[0][ind];
    }
  }

  void upd(int x, int y, int t, int ind = 1, int L = 0, int R = -1) { // set a_i = min(a_i,t)
    if (R == -1) R += N;
    if (R < x || y < L || mx[0][ind] <= t) return;
    push(ind, L, R);
    if (x <= L && R <= y && mx[1][ind] < t) {
      sum[ind] -= (ll) maxCnt[ind] * (mx[0][ind] - t);
      mx[0][ind] = t;
      return;
    }
    if (L == R) return;
    int M = (L + R) / 2;
    upd(x, y, t, 2 * ind, L, M);
    upd(x, y, t, 2 * ind + 1, M + 1, R);
    pull(ind);
  }

  ll qsum(int x, int y, int ind = 1, int L = 0, int R = -1) {
    if (R == -1) R += N;
    if (R < x || y < L) return 0;
    push(ind, L, R);
    if (x <= L && R <= y) return sum[ind];

    int M = (L + R) / 2;
    return qsum(x, y, 2 * ind, L, M) + qsum(x, y, 2 * ind + 1, M + 1, R);
  }

  int qmax(int x, int y, int ind = 1, int L = 0, int R = -1) {
    if (R == -1) R += N;
    if (R < x || y < L) return -1;
    push(ind, L, R);
    if (x <= L && R <= y) return mx[0][ind];

    int M = (L + R) / 2;
    return max(qmax(x, y, 2 * ind, L, M), qmax(x, y, 2 * ind + 1, M + 1, R));
  }
};
```

2.11 Sparse Table

```
template<int LOG> struct SparseTable{
  int rmq [1<<LOG][LOG+1];
  void preprocess(int *a, int n) {
      FOR(i,1,n+1) rmq[i][0]=a[i];
    int i, j;
      for(j=1;(1<<j)<=n;j++) for(i=1;i+(1<<j)-1<=n;i++)
        rmq[i][j]=min(rmq[i][j-1],rmq[i+(1<<(j-1))][j-1]);
  }

  // All indexed at 1
  ll qMin(int l, int r) {
      int k=log2(r-l+1);
      return min(rmq[l][k],rmq[r-(1<<k)+1][k]);
  }
};
```

2.12 Splay Tree

```
namespace splayTree {
    typedef struct snode* ps;
    struct snode {
        int val; ps p, pp, c[2]; // essential
        int sz; // # nodes in subtree
        bool flip; // range flip

        snode (int _val) {
            val = _val; c[0] = c[1] = p = pp = NULL;
            sz = 1; flip = 0;
        }
    };

    int getsz(ps x) { return x?x->sz:0; }
    int getDir(ps x, ps y) { return x?(x->c[1] == y):-1; }

    void trav(ps x, vi& v) {
        if (!x) return;
        trav(x->c[0],v); v.pb(x->val); trav(x->c[1],v);
    }

    ps recalc(ps x) {
        x->sz = 1+getsz(x->c[0])+getsz(x->c[1]);
        return x;
    }

    void setLink(ps x, ps y, int d) { // x propagated
        if (x) x->c[d] = y, recalc(x);
        if (y) y->p = x;
    }

    void prop(ps x) {
        if (!x || !x->flip) return;
        swap(x->c[0],x->c[1]);
        if (x->c[0]) x->c[0]->flip ^= 1;
        if (x->c[1]) x->c[1]->flip ^= 1;
        x->flip = 0;
    }

    void pushDown(ps x) {
        if (!x) return;
        if (x->p) pushDown(x->p);
        prop(x);
    }

    void rot(ps x, int d) { // precondition: x & parents propagated
        snode *y = x->c[d], *z = x->p;
        prop(y);
        setLink(x, y->c[d^1], d);
        setLink(y, x, d^1);
        setLink(z, y, getDir(z, x));
        y->pp = x->pp; x->pp = NULL;
    }

    ps splay(ps x) {
        pushDown(x);
        while (x && x->p) {
            ps y = x->p, z = y->p;
            int dy = getDir(y, x), dz = getDir(z, y);
            if (!z) rot(y, dy);
            else if (dy == dz) rot(z, dz), rot(y, dy);
            else rot(y, dy), rot(z, dz);
        }
        return x;
    }

    ps getExtreme(ps x, int d) { // get leftmost or rightmost node
        prop(x);
        if (x->c[d]) return getExtreme(x->c[d],d);
        return splay(x);
    }
}
```

2.13 struct comparison

```
// example of struct, constructor and comparator.
struct point {
    int x, y;
    // overloading of < operator
    bool operator<(const point &rhs) const{
        return (x == rhs.x) ? (y < rhs.y) : (x < rhs.x);
    }
};

int main(){
    point a = {1, 2}; //aggregate initialization
    point b = {2, 1};
    point c = {-1, 0};
    vector<point> v = {a, b, c};
    sort(v.begin(), v.end());
    EACH(p, v) cout << "(" << p.x << ", " << p.y << ")" << endl;
    return 0;
}
```

2.14 Wavelet Tree

```
// all index parameters are indexed from 1
struct wavelet {
  int lo, hi;
  int * from, * to;
  wavelet * l = nullptr, * r = nullptr;
  vi b;
  //nos are in range [x,y]
  //array indices are [from, to)
  wavelet(int * from, int * to, int x, int y) {
    this->from = from;
    this->to = to;
    lo = x, hi = y;
    if (lo == hi or from >= to) return;
    int mid = (lo + hi) / 2;
    b.reserve(to - from + 1);
    b.pb(0);
    //b[i] = no of elements from first "i" elements that go to left node
    auto f = [mid](int x) {
      return x <= mid;
    };
    for (auto it = from; it != to; it++)
      b.pb(b.back() + f( * it));
    auto pivot = stable_partition(from, to, f);
    l = new wavelet(from, pivot, lo, mid);
    r = new wavelet(pivot, to, mid + 1, hi);
  }

      ~wavelet() {
    delete l;
    delete r;
  }

  //kth smallest element in [l, r]
  int kth(int l, int r, int k) {
    if (l > r || k <= 0 || k > r - l + 1) return 0;
    if (lo == hi) return lo;
    //how many nos are there in left node from [l, r]
    int inLeft = b[r] - b[l - 1];
    int lb = b[l - 1]; //amt of nos in first (l-1) nos that go in left
    int rb = b[r]; //amt of nos in first (r) nos that go in left
    if (k <= inLeft) return this->l->kth(lb + 1, rb, k);
    return this->r->kth(l - lb, r - rb, k - inLeft);
  }

  //count of nos in [l, r] Less than or equal to k
  int LTE(int l, int r, int k) {
    if (l > r || k < lo) return 0;
    if (hi <= k) return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k);
  }

  //count of nos in [l, r] equal to k
  int count(int l, int r, int k) {
    if (l > r || k < lo || k > hi) return 0;
    if (lo == hi) return r - l + 1;
    int lb = b[l - 1], rb = b[r], mid = (lo + hi) / 2;
    if (k <= mid) return this->l->count(lb + 1, rb, k);
    return this->r->count(l - lb, r - rb, k);
  }

  // perform swapping between ith and (i+1)th elements
  void Swap(int i) {
    if (lo >= hi || i <= 0 || i >= to - from) return;
    int f = from[i - 1];
    int s = from[i];
    int mid = (lo + hi) / 2;
    if (f <= mid) {
      if (s > mid) b[i]--, swap(from[i - 1], from[i]);
      else if (l != nullptr) l->Swap(b[i]);
    } else {
      if (s <= mid) b[i]++, swap(from[i - 1], from[i]);
      else if (r != nullptr) r->Swap(i - b[i]);
    }
  }
};
```

3 Dynamic Programming

3.1 Assembly Line Scheduling

```
#define st 4
int carAssembly(int a[][st], int t[][st], int *e, int *x)
{
    int T1[st], T2[st], i;
    T1[0] = e[0] + a[0][0];
    T2[0] = e[1] + a[1][0];
    for (i = 1; i < st; ++i)
    {
        T1[i] = min(T1[i-1] + a[0][i], T2[i-1] + t[1][i] + a[0][i]);
        T2[i] = min(T2[i-1] + a[1][i], T1[i-1] + t[0][i] + a[1][i]);
    }
    return min(T1[st-1] + x[0], T2[st-1] + x[1]);
}
int main()
{
    int a[][st] = {{4, 5, 3, 2},
                   {2, 10, 1, 4}};
    int t[][st] = {{0, 7, 4, 5},
                   {0, 9, 2, 8}};
    int e[] = {10, 12}, x[] = {18, 7};
    printf("%d", carAssembly(a, t, e, x));
    return 0;
}
```

3.2 Coin Change

```
int count( int S[], int m, int n )
{
    int i, j, x, y;
    int table[n + 1][m];
    for (i = 0; i < m; i++)
        table[0][i] = 1;
    FOR(i,1,n+1)
    {
        FOR(j,0,m)
        {
            x = (i-S[j] >= 0) ? table[i - S[j]][j] : 0;
            y = (j >= 1) ? table[i][j - 1] : 0;
            table[i][j] = x + y;
        }
    }
    return table[n][m - 1];
}
```

3.3 Longest Increasing Subsequence

```
int inde(std::vector<int>& v, int l, int r, int key)
{
    while (r - l > 1) {
        int m = l + (r - l) / 2;
        if (v[m] >= key)
            r = m;
        else
            l = m;
    }
    return r;
}
```

```
int lisub(std::vector<int>& v)
{
    if (v.size() == 0)
        return 0;
    vi tail (v.size(), 0);
    int length = 1;

    tail[0] = v[0];
    for (size_t i = 1; i < v.size(); i++) {
        if (v[i] < tail[0])
            tail[0] = v[i];
        else if (v[i] > tail[length - 1])
            tail[length++] = v[i];
        else
            tail[inde(tail, -1, length - 1, v[i])] = v[i];
    }
    return length;
}
```

3.4 Minimum Difference Partition

```
int findMin(int arr[], int n)
{
        int sum = 0;
        FOR(i,0,n)
                sum += arr[i];
        bool dp[n+1][sum+1];
        FOR(i,0,n)
                dp[i][0] = true;
        FOR(i,1,sum+1)
                dp[0][i] = false;
        FOR(i,1,n+1)
        {
                FOR(j,1,sum+1)
                {
                        dp[i][j] = dp[i-1][j];
                if (arr[i-1] <= j)
                                dp[i][j] |= dp[i-1][j-arr[i-1]];
                }
        }
        int diff = INT_MAX;
        RFOR(j,sum/2,-1)
        {
                if (dp[n][j] == true)
                {
                        diff = sum-2*j;
                        break;
                }
        }
        return diff;
}
```

3.5 Minimum Coin

```
int minCoins(int coins[], int m, int V)
{
    int table[V+1];
    table[0] = 0;
    FOR(i,1,V+1)
        table[i] = INT_MAX;
    FOR(i,1,V+1)
    {
        FOR(j,0,m)
          if (coins[j] <= i)
          {
              int sub_res = table[i-coins[j]];
              if (sub_res != INT_MAX && sub_res + 1 < table[i])
                  table[i] = sub_res + 1;
          }
    }
    return table[V];
}
```

3.6 Optimal Binary Search Tree

```
int sum[100][100];
int optimalSearchTree(int keys[], int freq[], int n)
{
    FOR(i,0,n)
        FOR(j,i,n){
            if(i==j)
                sum[i][j] = freq[i];
            else
                sum[i][j] = sum[i][j-1] + freq[j];
        }

        int cost[n][n];
        FOR(i,0,n)
                cost[i][i] = freq[i];
        FOR(L,2,n+1)
        {
                FOR(i,0,n-L+2)
                {
                        int j = i+L-1;
                        cost[i][j] = INT_MAX;
                        FOR(r,i,j+1)
                        {
                        int c = ((r > i)? cost[i][r-1]:0) +
                                        ((r < j)? cost[r+1][j]:0) +
                                        sum[i][j];
                        if (c < cost[i][j])
                                cost[i][j] = c;
                        }
                }
        }
        return cost[0][n-1];
}
```

3.7 Travelling Salesman Problem

```
const int MX = 15;

int N, dp[MX][1<<MX], dist[MX][MX];

int solve() {
    FOR(i,N) FOR(j,1<<N) dp[i][j] = MOD;

    dp[0][1] = 0;
    FOR(j,1<<N) FOR(i,N) if (j&(1<<i))
        FOR(k,N) if (!(j&(1<<k)))
            dp[k][j^(1<<k)] = min(dp[k][j^(1<<k)],
                                  dp[i][j]+dist[i][k]);

    int ans = MOD;
    FOR(j,1,N) ans = min(ans,dp[j][(1<<N)-1]+dist[j][0]);
    return ans;
}
```

3.8 Unbounded Knapsack

```
int uks(int w, int n, int val[], int wt[])
{
        int dp[w+1];
        memset(dp, 0, sizeof dp);
        int ans = 0;
        FOR(i,0,w+1)
        FOR(j,0,n)
                if (wt[j] <= i)
                        dp[i] = max(dp[i], dp[i-wt[j]]+val[j]);
        return dp[w];
}
```

4 Flow

4.1 Dinic

```
/**
* Description: Faster Flow, Bipartite Matching
*/

template<int SZ> struct Dinic {
    struct Edge {
        int v;
        ll flow, cap;
        int rev;
```

```
    };

    vector<Edge> adj[SZ];

    void addEdge(int u, int v, ll cap) {
        Edge a{v, 0, cap, sz(adj[v])};
        Edge b{u, 0, 0, sz(adj[u])};
        adj[u].pb(a), adj[v].pb(b);
    }

    int level[SZ], st[SZ];

    bool bfs(int s, int t) {
        REP(i,SZ) level[i] = -1, st[i] = 0;
        level[s] = 0;

        queue<int> q; q.push(s);
        while (sz(q)) {
            int u = q.front(); q.pop();
            for (auto e: adj[u])
                if (level[e.v] < 0 && e.flow < e.cap) {
                    level[e.v] = level[u] + 1;
                    q.push(e.v);
                }
        }

        return level[t] >= 0;
    }

    ll sendFlow(int s, int t, ll flow) {
        if (s == t) return flow;

        for (  ; st[s] < sz(adj[s]); st[s] ++) {
            Edge &e = adj[s][st[s]];

            if (level[e.v] != level[s]+1 || e.flow == e.cap) continue;
            ll temp_flow = sendFlow(e.v, t, min(flow, e.cap - e.flow));

            if (temp_flow > 0) {
                e.flow += temp_flow;
                adj[e.v][e.rev].flow -= temp_flow;
                return temp_flow;
            }
        }

        return 0;
    }

    ll maxFlow(int s, int t) {
        if (s == t) return -1;
        ll total = 0;
        while (bfs(s, t))  while (ll flow = sendFlow(s, t, INT_MAX)) total += flow;
        return total;
    }
};
```

4.2 Konig

```
/**
 * Turn a maximum matching in a bipartite graph
 * into the minimum vertex cover.
 * Let n1 be the number of vertices in U.
 * Let n2 be the number of vertices in V.
 */

vvi graph;
int match[n1+n2]; // match[i] = j if (i, j) is an edge in tne matching.
               //            -1 otherwise.
us z;
bool visited[n1+n2];

void dfs(int u, bool in_matching) {
    if(visited[u]) continue;
    visited[u] = true;
    z.insert(u);
    if(u < n1) {
        dfs(match[u], in_matching);
    }else{
        EACH(v, g[u]) {
            if(match[v] == -1) dfs(v, !in_matching);
        }
    }
}


vi min_vertex_cover() {
    vi mvc;
    REP(i, n1) {
        if(match[i] == -1 && !visited[i]) {
            dfs(i, false);
        }
        REP(i, n1) if(z.find(i) == z.end()) mvc.pb(i);
        FOR(i,n1, n1+n2) if(z.find(i) != z.end()) mvc.pb(i);
    }
    return mvc;

}
```

4.3 Min-Cost Max Flow

```
/**
* Description: Min Cost Max Flow
* Not min cost flow
*/

struct Edge {
  int v, flow, C, rev, cost;
};

template<int SZ> struct mcf {
  pi pre[SZ];
  int cost[SZ], num[SZ], SC, SNC;
  ll flo, ans, ccost;
  vector<Edge> adj[SZ];

  void addEdge(int u, int v, int C, int cost) {
      Edge a{v, 0, C, sz(adj[v]), cost};
      Edge b{u, 0, 0, sz(adj[u]), -cost};
      adj[u].pb(a), adj[v].pb(b);
  }

  void reweight() {
    REP(i,SZ) {
      for (auto& p: adj[i]) p.cost += cost[i]-cost[p.v];
    }
  }

  bool spfa() {
    REP(i,SZ) cost[i] = MOD, num[i] = 0;
    cost[SC] = 0, num[SC] = MOD;
    pqg<pi> todo;
    todo.push({0,SC});

    while (todo.size()) {
      pi x = todo.top(); todo.pop();
      if (x.F > cost[x.S]) continue;
      for (auto a: adj[x.S]) if (x.F+a.cost < cost[a.v] && a.flow < a.C) {
        pre[a.v] = {x.S,a.rev};
        cost[a.v] = x.F+a.cost;
        num[a.v] = min(a.C-a.flow,num[x.S]);
        todo.push({cost[a.v],a.v});
      }
    }

    ccost += cost[SNC];
    return num[SNC] > 0;
  }

  void backtrack() {
    flo += num[SNC], ans += (ll)num[SNC]*ccost;
    for (int x = SNC; x != SC; x = pre[x].F) {
      adj[x][pre[x].S].flow -= num[SNC];
      int t = adj[x][pre[x].S].rev;
      adj[pre[x].F][t].flow += num[SNC];
    }
  }

  pi mincostflow(int sc, int snc) {
    SC = sc, SNC = snc;
    flo = ans = ccost = 0;

    spfa();
    while (1) {
      reweight();
      if (!spfa()) return {flo,ans};
      backtrack();
    }
  }
};
```

4.4 Push Relabel

```
/**
 * Push Relabel Max Flow Template
 * Computes max flow but the amount of flow
 * across each edge ins't accurate.
 * (Excess isn't pushed back to source).
 */

struct Edge {
    int v;
    ll flow, C;
    int rev;
};

template <int SZ> struct PushRelabel {
    vector<Edge> adj[SZ];
    ll excess[SZ];
    int dist[SZ], count[SZ+1], b = 0;
    bool active[SZ];
    vi B[SZ];

    void addEdge(int u, int v, ll C) {
        Edge a{v, 0, C, sz(adj[v])};
        Edge b{u, 0, 0, sz(adj[u])};
        adj[u].pb(a), adj[v].pb(b);
    }

    void enqueue (int v) {
        if (!active[v] && excess[v] > 0 && dist[v] < SZ) {
            active[v] = 1;
            B[dist[v]].pb(v);
            b = max(b, dist[v]);
        }
    }

    void push (int v, Edge &e) {
        ll amt = min(excess[v], e.C-e.flow);
        if (dist[v] == dist[e.v]+1 && amt > 0) {
            e.flow += amt, adj[e.v][e.rev].flow -= amt;
            excess[e.v] += amt, excess[v] -= amt;
            enqueue(e.v);
        }
    }

    void gap (int k) {
        FOR(v,SZ) if (dist[v] >= k) {
            count[dist[v]] --;
            dist[v] = SZ;
            count[dist[v]] ++;
            enqueue(v);
        }
    }

    void relabel (int v) {
        count[dist[v]] --; dist[v] = SZ;
        for (auto e: adj[v]) if (e.C > e.flow) dist[v] = min(dist[v], dist[e.v] + 1);
        count[dist[v]] ++;
        enqueue(v);
    }

    void discharge(int v) {
        for (auto &e: adj[v]) {
            if (excess[v] > 0) push(v,e);
            else break;
        }
        if (excess[v] > 0) {
            if (count[dist[v]] == 1) gap(dist[v]);
            else relabel(v);
        }
    }

    ll maxFlow (int s, int t) {
        for (auto &e: adj[s]) excess[s] += e.C;

        count[0] = SZ;
        enqueue(s); active[t] = 1;

        while (b >= 0) {
            if (sz(B[b])) {
                int v = B[b].back(); B[b].pop_back();
                active[v] = 0; discharge(v);
            } else b--;
        }
        return excess[t];
    }
};
```

5 Geometry

5.1 Circle

```
typedef pair<cd,ld> circle;

// Finding x-th intersection of two circles a,b
// If they dont intersect then return (nan,nan)
cd intersect(circle a, circle b, int x = 0) {
    ld d = sqrt(norm(a.F-b.F));
    ld co = (a.S*a.S+d*d-b.S*b.S)/(2*a.S*d);
    ld theta = acos(co);

    cd tmp = (b.F-a.F)/d;
    if (x == 0) return a.F+tmp*a.S*polar((ld)1.0,theta);
    return a.F+tmp*a.S*polar((ld)1.0,-theta);
}

// Calculating arc-length assuming a,b on circle x
ld arc(circle x, cd a, cd b) {
    cd d = (a-x.F)/(b-x.F);
    return x.S*acos(d.real());
}

bool on (circle x, cd y) {
    return abs(norm(y-x.F) - x.S*x.S) == 0;
}
```

5.2 Closest Pairs

```
ld dist(pd a, pd b) {
    return sqrt(pow(a.F-b.F,2)+pow(a.S-b.S,2));
}

// Closest pairs using sweepline in O(nlogn)
pair<pd,pd> solve(vector<pd>& v) {
    pair<ld,pair<pd,pd>> bes; bes.F = 1e9;

    set<pd> S;
    int ind = 0;

    sort(all(v));
    REP(i,sz(v)) {
        if (i && v[i] == v[i-1]) return {v[i],v[i]};

        while (v[i].F-v[ind].F >= bes.F) {
            S.erase({v[ind].S,v[ind].F});
            ind ++;
        }

        for (auto it = S.ub({v[i].S-bes.F,INF});
            it != S.end() && it->F < v[i].S+bes.F;
            it = next(it)) {
            pd t = {it->S,it->F};
            bes = min(bes,{dist(t,v[i]),{t,v[i]}});
        }
        S.insert({v[i].S,v[i].F});
    }

    return bes.S;
}
```

5.3 Complex

```
namespace ComplexOp {
    template<class T> istream& operator>> (istream& is, complex<T>& p) {
        T value;
        is >> value; p.real(value);
        is >> value; p.imag(value);
        return is;
    }

    bool operator<(const cd& a, const cd& b) {
        if (a.real() != b.real()) return a.real() < b.real();
        return a.imag() < b.imag();
    }
    bool operator>(const cd& a, const cd& b) {
```

```
        if (a.real() != b.real()) return a.real() > b.real();
        return a.imag() > b.imag();
    }
    bool operator<=(const cd& a, const cd& b) { return a < b || a == b; }
    bool operator>=(const cd& a, const cd& b) { return a > b || a == b; }
    cd max(const cd& a, const cd& b) { return a>b?a:b; }
    cd min(const cd& a, const cd& b) { return a<b?a:b; }

    ld cross(cd a, cd b) { return (conj(a)*b).imag(); }
    ld area(cd a, cd b, cd c) { return cross(b-a,c-a); }
    ld dot(cd a, cd b) { return (conj(a)*b).real(); }

    cd reflect(cd p, cd a, cd b) { return a+conj((p-a)/(b-a))*(b-a); }
    cd proj(cd p, cd a, cd b) { return (p+reflect(p,a,b))/(ld)2; }

    cd line(cd a, cd b, cd c, cd d) {
        ld x = area(a,b,c), y = area(a,b,d);
        return (x*d-y*c)/(x-y);
    }

    vcd segment(cd A, cd B, cd C, cd D) { // kattis segmentintersection
        if (A > B) swap(A,B);
        if (C > D) swap(C,D);

        ld a1 = area(A,B,C), a2 = area(A,B,D);
        if (a1 > a2) swap(a1,a2);
        if (!(a1 <= 0 && a2 >= 0)) return {};

        if (a1 == 0 && a2 == 0) {
            if (area(A,C,D) != 0) return {};
            cd x1 = max(A,C), x2 = min(B,D);
            if (x1 > x2) return {};
            if (x1 == x2) return {x1};
            return {x1,x2};
        }

        cd z = line(A,B,C,D);
        if (A <= z && z <= B) return {z};
        return {};
    }
}
```

5.4 Convex Hull

```
ll cross(pi O, pi A, pi B) {
    return (ll)(A.F-O.F)*(B.S-O.S)-(ll)(A.S-O.S)*(B.F-O.F);
}

// Divide and conquer convexHull in O(nlogn)
vpi convex_hull(vpi& P) {
    sort(all(P)); P.erase(unique(all(P)),P.end());
    int n = sz(P);
    if (n == 1) return P;

    vpi bot = {P[0]};
    FOR(i,1,n) {
        while (sz(bot) > 1 && cross(bot[sz(bot)-2], bot.back(), P[i]) <= 0) bot.pop_back();
        bot.pb(P[i]);
    }
    bot.pop_back();

    vpi up = {P[n-1]};
    RREP(i,n-1) {
        while (sz(up) > 1 && cross(up[sz(up)-2], up.back(), P[i]) <= 0) up.pop_back();
        up.pb(P[i]);
    }
    up.pop_back();

    bot.insert(bot.end(),all(up));
    return bot;
}
```

5.5 3D Geometry

```
namespace Geo3d {
    vl operator-(vl a, vl b) {
        vl c(sz(a)); REP(i,sz(a)) c[i] = a[i]-b[i];
        return c;
    }

    bool ismult(vl b, vl c) {
        if ((ld)b[0]*c[1] != (ld)b[1]*c[0]) return 0;
        if ((ld)b[0]*c[2] != (ld)b[2]*c[0]) return 0;
        if ((ld)b[2]*c[1] != (ld)b[1]*c[2]) return 0;
        return 1;
    }

    bool collinear(vl a, vl b, vl c) {
        b = b-a, c = c-a;
        return ismult(b,c);
    }

    vl cross(vl a, vl b) {
        return {a[1]*b[2]-a[2]*b[1],
                a[2]*b[0]-a[0]*b[2],
                a[0]*b[1]-a[1]*b[0]};
    }

    bool coplanar(vl a, vl b, vl c, vl d) {
        b = b-a, c = c-a, d = d-a;
        return ismult(cross(b,c),cross(b,d));
    }
}
```

5.6 Polygon

```
// Signed area of simple polygon
ld area(vcd& v) {
    ld x = 0;
    REP(i,sz(v)) {
        int j = (i+1)%sz(v);
        x += (ld)v[i].real()*v[j].imag();
        x -= (ld)v[j].real()*v[i].imag();
    }
    return x/2;
}
```

6 Graph

6.1 Articulation Points

```
/**
* Finds the articulation points in an undirected graph
* O(N + M)
*/

int n; // number of nodes
vvi adj; // graph as adjacency list

vector<bool> visited;
vi tin, fup;
int timer;
set<int> articulation_points;

// note: a node v might be added to articulation_points multiple times.
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = fup[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            fup[v] = min(fup[v], tin[to]);
        } else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] >= tin[v] && p!=-1)
                articulation_points.insert(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
      articulation_points.insert(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    fup.assign(n, -1);
    REP(i, n)
        if (!visited[i])
            dfs (i);
}
```

6.2 Bellman Ford

```
/**
* Description: Shortest Path w/ negative edge weights
    * Can be useful with linear programming
    * Constraints of the form x_i-x_j<k
*/

template<int SZ> struct BellmanFord {
    bool bad[SZ]; // reachable from source and a negative cycle
    vector<pair<pi,int>> edge;
    ll dist[SZ];

    void addEdge(int A, int B, int C) {
      edge.pb({{A,B},C});
    }

    ll query(int x) {
        if (bad[x]) return -INF;
        return dist[x];
    }

    void gen(int s) {
        REP(i,SZ) dist[i] = INF, bad[i] = 0;
        dist[s] = 0;

        REP(i,SZ) for (auto a: edge)
            if (dist[a.F.F] < INF) dist[a.F.S] = min(dist[a.F.S], dist[a.F.F]+a.S);

        for (auto a: edge) if (dist[a.F.F] < INF)
            if (dist[a.F.S] > dist[a.F.F]+a.S) bad[a.F.S] = 1;

        REP(i,SZ) for (auto a: edge)
            if (bad[a.F.F]) bad[a.F.S] = 1;
    }
};
```

6.3 Connected Components

```
template<int SZ> struct CC{
  int comp[SZ]; //vertex->comp
  vvi comps; //comp->vextices
  vi edges[SZ];
  vi visited;
  int N; //set N
  void addEdge(int u, int v){
    edges[u].pb(v);
    edges[v].pb(u);
  }
  void dfs(int src){
    visited[src] = 1;
    comps[sz(comps)-1].pb(src);
    comp[src]=sz(comps)-1;
    EACH(nei, edges[src])
      if (visited[nei] == 0)
        dfs(nei);
  }
  void solve(){
    visited.assign(N, 0);
    REP(i,N) if (visited[i] == 0){
      comps.pb(vi());
      dfs(i);
    }
  }
};
```

6.4 Dijkstra

```
/**
* Description: shortest path!
* Works with negative edge weights (aka SPFA?)
*/
template<class T> T poll(pqg<T>& x) {
    T y = x.top(); x.pop();
    return y;
}

template<int SZ> struct Dijkstra {
    ll dist[SZ]; int pa[SZ];
    vpi adj[SZ];
    pqg<pl> q;

    void addEdge(int A, int B, int C) {
        adj[A].pb({B,C}), adj[B].pb({A,C}); // undirected
    }

    void gen(int st) {
        fill_n(dist,SZ,INF);
        fill_n(pa,SZ,-1);
        q = pqg<pl>(); q.push({dist[st] = 0,st});
        while (sz(q)) {
                auto x = poll(q);
                if (dist[x.S] < x.F) continue;
                for (auto y: adj[x.S]) if (x.F+y.S < dist[y.F]){
          q.push({dist[y.F] = x.F+y.S,y.F});
          pa[y.F] = x.S;
        }
        }
    }
};
```

6.5 Euler Tour/HLD preprocessing

```
/**
 * Reference: https://codeforces.com/blog/entry/53170
 * Create in[i], out[i], nxt[i], such that [in[i], out[i]) is the subtree
 * under node i, and nxt[i] is the node at the top of the heavy path that
 * i is on. [in[nxt[i]], in[i]] is the hld path from i to nxt[i].
 *
 * Use LCA with HLD.
 */

// pick a power of 2 two times greater than the number of nodes in the tree.
const int MAXN = 262144;

// rin[i] maps an index in the segment tree to a node in the tree.
int sz[MAXN], in[MAXN], rin[MAXN], nxt[MAXN], out[MAXN];

void dfs_sz(int v = 0)
{
    sz[v] = 1;
    for(auto &u: g[v])
    {
        dfs_sz(u);
        sz[v] += sz[u];
        if(sz[u] > sz[g[v][0]])
            swap(u, g[v][0]);
    }
}

int t;
void dfs_hld(int v = 0)
{
    in[v] = t++;
    rin[in[v]] = v;
    for(auto u: g[v])
    {
        nxt[u] = (u == g[v][0] ? nxt[v] : u);
        dfs_hld(u);
    }
    out[v] = t;
}
```

6.6 Floyd Warshall

```
/**
* Description: All-Pairs Shortest Path
*/

template<int SZ> struct FloydWarshall {
    int n; // vertices
    ll dist[SZ][SZ];
    bool bad[SZ][SZ];
    ll query(int x, int y) {
        if (bad[x][y]) return -INF;
        return dist[x][y];
    }

    void addEdge(int u, int v, int w){
      dist[u][v] = min(dist[u][v],(ll)w);
    }
```

```
    void solve() {
      REP(i,n) REP(j,n) dist[i][j] = INF, bad[i][j] = 0;
      REP(i,n) dist[i][i] = 0;
        REP(k,n) REP(i,n) REP(j,n) if (dist[i][k] != INF && dist[k][j] != INF)
            dist[i][j] = min(dist[i][j],dist[i][k]+dist[k][j]);

        REP(k,n) REP(i,n) REP(j,n) if (dist[i][k] != INF && dist[k][j] != INF)
            if (dist[i][j] > dist[i][k]+dist[k][j])
                bad[i][j] = 1;

        REP(k,n) REP(i,n) REP(j,n) {
            if (dist[i][k] < INF && bad[k][j]) bad[i][j] = 1;
            if (bad[i][k] && dist[k][j] < INF) bad[i][j] = 1;
        }
    }
};
```

6.7 Topological Sort

```
/**
* Description: sorts vertices such that if there exists an edge x->y, then x goes before y
*/

template<int SZ> struct Topo {
    int N, in[SZ]; //set N
    vi res, adj[SZ];
    void addEdge(int x, int y) {
        adj[x].pb(y), in[y]++;
    }

    void sort() {
        queue<int> todo;
        REP(i,N) if (in[i] == 0) todo.push(i);
        while (sz(todo)) {
            int x = todo.front(); todo.pop();
            res.pb(x);
            for (int i: adj[x]) {
                in[i]--;
                if (!in[i]) todo.push(i);
            }
        }
    }
};
```

6.8 Kosaraju

```
template<int SZ> struct scc {
    vi adj[SZ], radj[SZ], todo;
    int N, comp[SZ]; // vertex->comp
    vvi comps; // comp->vertices
    bitset<SZ> visit;
    void dfs(int v) {
        visit[v] = 1;
        for (int w: adj[v]) if (!visit[w]) dfs(w);
        todo.pb(v);
    }

    void dfs2(int v, int val) {
        comp[v] = val;
        comps[sz(comps)-1].pb(v);
        for (int w: radj[v]) if (comp[w] == -1) dfs2(w,val);
    }

    void addEdge(int a, int b) { adj[a].pb(b), radj[b].pb(a); }

    void genSCC() {
        REP(i,N) comp[i] = -1, visit[i] = 0;
        REP(i,N) if (!visit[i]) dfs(i);
        reverse(all(todo)); // toposort
        for (int i: todo) if (comp[i] == -1){
          comps.pb(vi());
          dfs2(i,sz(comps)-1);
        }
    }
};
```

6.9 Kruskal

```
/**
* Description: computes the minimum spanning tree in O(ElogE) time
*/

ll kruskal(vvi edge) { // edge[i] = {weight, u, v};
    DSU<MX> D;
    sort(all(edge));
    ll ans = 0;
    for (auto a: edge) if (D.unite(a[1],a[2])) ans += a[0]; // edge is in MST
    return ans;
}
```

6.10 Lowest Common Ancestor

```
template<int SZ> struct LCA {
    vi adj[SZ];
    SparseTable<pi,2*SZ> r;
    vpi tmp;
    int depth[SZ], pos[SZ];
    void addEdge(int u, int v) {
        adj[u].pb(v), adj[v].pb(u);
    }

    void dfs(int u, int prev){
        pos[u] = sz(tmp); depth[u] = depth[prev]+1;
        tmp.pb({depth[u],u});
        for (int v: adj[u]) if (v != prev) {
            dfs(v, u);
            tmp.pb({depth[u],u});
        }
    }

    // All nodes are indexed from 1
    void init(int R) {
        depth[0]=-1;
        dfs(R, 0);
        r.build(tmp);
    }

    int lca(int u, int v){
        u = pos[u], v = pos[v];
        if (u > v) swap(u,v);
        return r.query(u,v).S;
    }

    int dist(int u, int v) {
        return depth[u]+depth[v]-2*depth[lca(u,v)];
    }
};
```

6.11 Tarjan BCC

```
template<int SZ> struct BCC {
    int N; // set N
    vi adj[SZ];
    vector<vpi> biComps; // biconnected components
    vi cutPoints; // articulation points
    vpi bridges; // bridges

    void addEdge(int u, int v) { adj[u].pb(v), adj[v].pb(u); }

    int ti = 0, disc[SZ], low[SZ], comp[SZ], par[SZ];
    vpi st;
    void BCCutil(int u, bool root = 0) {
        disc[u] = low[u] = ti++;
        int child = 0;
        for (int v: adj[u])
          if (v != par[u]){
            if (disc[v] == -1) {
                child ++; par[v] = u;
                st.pb({u,v});
                BCCutil(v);
                low[u] = min(low[u],low[v]);
                if (low[v] > disc[u]) bridges.pb({u, v});
                if ((root && child > 1) || (!root && disc[u] <= low[v])) { // articulation point!
                    vpi tmp;
                    cutPoints.pb(u);
                    while (st.back() != mp(u,v)) tmp.pb(st.back()), st.pop_back();
                    tmp.pb(st.back()), st.pop_back();
                    biComps.pb(tmp);
                }
            } else if (disc[v] < disc[u]) {
                low[u] = min(low[u],disc[v]);
```

```
                st.pb({u,v});
            }
        }
    }

    void bcc() {
        REP(i,N) par[i] = disc[i] = low[i] = -1;
        REP(i,N) if (disc[i] == -1) {
            BCCutil(i,1);
            if (sz(st)) biComps.pb(st);
            st.clear();
        }
    }
};
```

6.12 Diameter of a Tree

```
struct TreeDiameter {
    int n, dist[MX], pre[MX];
    vi adj[MX];

    void addEdge(int a, int b) {
        adj[a].pb(b), adj[b].pb(a);
    }

    void dfs(int cur) {
        for (int i: adj[cur]) if (i != pre[cur]) {
            pre[i] = cur;
            dist[i] = dist[cur]+1;
            dfs(i);
        }
    }

    void genDist(int cur) {
        memset(dist,0,sizeof dist);
        pre[cur] = -1;
        dfs(cur);
    }

    int diameterLength() {
        genDist(1);
        int bes = 0; FOR(i,1,n+1) if (dist[i] > dist[bes]) bes = i;
        genDist(bes); FOR(i,1,n+1) if (dist[i] > dist[bes]) bes = i;
        return dist[bes];
    }

    vi genCenter() {
        int t = diameterLength();
        int bes = 0; FOR(i,1,n+1) if (dist[i] > dist[bes]) bes = i;
        REP(i,t/2) bes = pre[bes];
        if (t&1) return {bes,pre[bes]};
        return {bes};
    }
};
```

7 Math

7.1 Advanced Combinatorials

```
using namespace modulo;
using namespace factor;
// Extends combinatorial in O(log(N))
// The maximum prime power should be small
struct combin{
  ll mod; vpl facs;
  combin(ll _mod=MOD){
    mod = _mod;
    vpl factors = factorize(mod);
    REP(i, sz(factors)){
      ll psz = (ll)pow(factors[i].F, factors[i].S);
      facs.pb({psz, factors[i].F});
    }
  }

  void addRange(vl& v, ll start, ll end, int delta, ll P) {
        int N = sz(v);
        if (start > end) return;
        if (end == 0) return;
        REP(i,N) {
                ll x = start + i;
                if(x > end) break;
                ll inRange = (end - x) / N + 1;
                v[x%N] += delta * inRange;
        }
        addRange(v, (start+P-1)/P , end/P, delta, P);
  }

  ll primeNCR(ll n, ll r, ll prime) {
        if(r == 0) return 1;
        ll balance = getLegendre(n, prime) - getLegendre(n-r, prime) - getLegendre(r, prime);
        if (balance > 0) return 0;
        vl factorCount(prime);
        addRange(factorCount, 1, r, -1, prime);
        addRange(factorCount, n - r + 1, n, 1, prime);
        ll prod = 1;
        FOR(i,1,prime) {
                ll p = factorCount[i];
                if (p == 0) continue;
                ll base = i;
                if (p < 0) {
                        base = modExp(base, prime - 2, prime);
                        p = -p;
                }
                ll part = modExp(base, p, prime);
                prod = modMul(prod, part, prime);
        }
        return prod;
  }

  ll powPrimeNCR(ll n, ll r, ll modulo, ll prime) {
    ll P = modulo / prime * (prime - 1) - 1;
        if (r == 0) return 1;
        ll balance = getLegendre(n, prime) - getLegendre(n-r, prime) - getLegendre(r, prime);
        ll factor = modExp(prime, balance, modulo);
        if (factor == 0) return 0;
        vl factorCount(modulo);
        addRange(factorCount, 1, r, -1, prime);
        addRange(factorCount, n - r + 1, n, 1, prime);

        ll prod = factor;
        FOR(i,0,modulo) {
                ll p = factorCount[i];
                if(p == 0) continue;
                if(i % prime == 0)continue;

                ll base = i;
                if(p < 0) {
                        base = modExp(base, P, modulo);
                        p = -p;
                }
          ll part = modExp(base, p, modulo);
                prod = modMul(prod, part, modulo);
        }
        return prod;
  }

  ll nCr(ll n, ll r){
    r = min(n-r, r);
    pl res = {0, 1};
    REP(i, sz(facs)){
      pl a;
      if (facs[i].F == facs[i].S) a = {primeNCR(n, r, facs[i].S), facs[i].S};
      else a = {powPrimeNCR(n, r, facs[i].F, facs[i].S), facs[i].F};
      res = CRT(a, res);
    }
    return res.F;
  }
};
```

7.2 Combinatorials

```
using namespace modulo;
using namespace factor;
// Extends combinatorial upto SZ
// SZ <= 500000 is fine
// To boost to 1000000 use int instead of ll for fac and ifac
template<int SZ> struct combi{
  ll mod, fac[SZ+1], ifac[SZ+1];
  vpl factors;

  combi(ll _mod=MOD) {
    mod = _mod; factors = factorize(mod);
    fac[0] = ifac[0] = 1;
    FOR(i,1,SZ+1) {
      int I = i;
      REP(j,sz(factors))
        while (I % factors[j].F == 0)
          I /= factors[j].F;
```

```
      fac[i] = modMul(I,fac[i-1],mod), ifac[i] = modInv(fac[i],mod);
    }
  }

  ll nCr(ll n, ll r) {
    if (n < r || r < 0) return 0;
    ll tmp = modMul(modMul(fac[n],ifac[r],mod),ifac[n-r],mod);
    REP(i,sz(factors)) {
      ll prime = factors[i].F;
      ll t = getLegendre(n,prime)-getLegendre(n-r,prime)-getLegendre(r,prime);
      tmp = modMul(tmp,modExp(prime,t,mod),mod);
    }
    return tmp;
  }
};
```

7.3 Factor

```
namespace factor{
        vpl factorize(ll n) { // O(n^1/2)
                vpl pri;

                int t = 0;
                while (n%2==0) n/=2, t++;
                if (t > 0) pri.pb({2, t});

                for (ll i = 3; i*i <= n; i+=2)
                        if (n % i == 0) {
                                int t = 0;
                                while (n % i == 0) n /= i, t ++;
                                pri.pb({i,t});
                        }

                if (n > 1) pri.pb({n,1});
                return pri;
        }

        void getDivsUtil(vl& divs, vpl& facs, int index, ll prod){
                if (index == -1){
                        divs.pb(prod);
                        return;
                }
                ll fpow = 1;
                FOR(i, 0, facs[index].S + 1){
                        getDivsUtil(divs, facs,  index - 1, prod*fpow);
                        fpow *= facs[index].F;
                }
        }

        // Get all divisors in O(n^1/3)
        vl getDivs(vpl& facs){
                vl divs;
                getDivsUtil(divs, facs, sz(facs)-1, 1);
                return divs;
        }
}
```

7.4 FFT

```
int Debruijn[32] = {
  0, 1, 28, 2, 29, 14, 24, 3, 30, 22, 20, 15, 25, 17, 4, 8,
  31, 27, 13, 23, 21, 19, 16, 7, 26, 12, 18, 6, 11, 5, 10, 9
};
int lsb(int n){
  return 1 << Debruijn[((uint32_t)((n & -n) * 0x077CB531U)) >> 27];
}

vcd w;
void getRoots(int n){
  REP(i,n/2) w.pb(0);
  w[0] = cd(1, 0);
  for (int i = 1; i < n/2; i <<= 1){
    ld angle = 2*i*PI/n;
    w[i] = cd(cos(angle), sin(angle));
  }
  for (int i = 2; i < n/2; i ++){
    if (w[i] != cd(0, 0)) continue;
    int low = lsb(i);
    int rest = i - low;
    w[i] = w[low] * w[rest];
  }
}

void FFT(vcd& v, bool inv){
  int n = sz(v);
  int k = 0;
  // Bit reversal permutation
  FOR(i,1,n){
    int bit = n >> 1;
    for (; (k & bit) > 0; bit >>= 1)
      k ^= bit;
    k ^= bit;
    if (i < k){
      cd temp = v[i];
      v[i] = v[k];
      v[k] = temp;
    }
  }

  for (int len = 2; len <= n; len <<= 1){
    int skip = n/len;
    RANGE(j, 0, n, len){
      REP(k, len/2){
        int ind = k*skip;
        cd root = inv ? conj(w[ind]) : w[ind];
        cd c = root * v[j+k+len/2];
        v[j+k+len/2] = v[j+k] - c;
        v[j+k] += c;
      }
    }
  }

  if (inv) REP(j, n) v[j] /= n;
}

int hsb(int n){
  n |= n >> 1;
  n |= n >> 2;
  n |= n >> 4;
  n |= n >> 8;
  n |= n >> 16;
  n ++;
  n <<= 1;
  return n;
}

vl convolve(vi p1, vi p2){
  int n = hsb(max(sz(p1)-1, sz(p2)-1));
  getRoots(n);
  vcd cp1(n), cp2(n);
  REP(i, n){
    if (i < sz(p1)) cp1[i] = cd(p1[i], 0);
    else cp1[i] = cd(0, 0);
    if (i < sz(p2)) cp2[i] = cd(p2[i], 0);
    else cp2[i] = cd(0, 0);
  }
  FFT(cp1, false);
  FFT(cp2, false);
  vcd cp(n);
  REP(i, n) cp[i] = cp1[i]*cp2[i];
  FFT(cp, true);
  vl p;
  REP(i, n){
    p.pb((ll)nearbyint(cp[i].real()));
  }
  return p;
}
```

7.5 Matrix Inversion

```
namespace matrix_inv{
  // Do gaussian elimination ON mat and return determinant+rank
  pair<ld, int> gauss(vvd& mat){
    ld prod = 1.0;
    int nex = 0;
    int rows = sz(mat);
    int rank = rows;
    int cols = sz(mat[0]);
    REP(i, rows){
      int row = -1;
      FOR(j, nex, rows) if (abs(mat[j][i]) > EPS){ row = j; break; }
      if (row == -1) { rank --; prod = 0; continue; }
      if (row != nex) prod *= -1, mat[row].swap(mat[nex]);
      prod *= mat[nex][i];

      ld inv = 1.0/mat[nex][i];
      FOR(k, i, cols) mat[nex][k]*= inv;
      REP(k, rows) if (k!=nex){
        ld x = mat[k][i];
        FOR(j, i, cols) mat[k][j] -= x*mat[nex][j];
      }
```

```
      nex ++;
    }
    REP(i, rows) REP(j, cols) if (abs(mat[i][j]) <= EPS) mat[i][j] = 0;
    return {prod, rank};
  }

  vvd matInv(vvd& mat){
    int n = sz(mat);
    vvd eq; REP(i, n){eq.pb(vd()); REP(j, 2*n) eq[i].pb(0);};
    REP(i, n) REP(j, n) eq[i][j] = mat[i][j];
    REP(i, n) eq[i][i+n] = 1.0;
    if (gauss(eq).F == 0) return {};
    vvd inv; REP(i, n){inv.pb(vd()); REP(j, n) inv[i].pb(0);};
    REP(i, n) REP(j, n) inv[i][j] = eq[i][j+n];
    return inv;
  }
}
```

7.6 Matrix

```
namespace matrix{
  vvl matCopy(const vvl& A){
    vvl C; newVVl(C, sz(A), sz(A[0]), 0);
    REP(i, sz(A)) REP(j, sz(A[0])) C[i][j] = A[i][j];
    return C;
  }
  vvl matAdd(const vvl& A, const vvl& B, ll mod=MOD){
    vvl C; newVVl(C, sz(A), sz(A[0]), 0);
    REP(i, sz(A)) REP(j, sz(A[0])) C[i][j] = (A[i][j]+B[i][j])%mod;
    return C;
  }
  vvl constMul(const vvl& A, ll c, ll mod=MOD){
    vvl C; newVVl(C, sz(A), sz(A[0]), 0);
    c %= mod;
    REP(i, sz(A)) REP(j, sz(A[0])) C[i][j] = (A[i][j]*c)%mod;
    return C;
  }
  vvl matMul(const vvl& A, const vvl& B, ll mod=MOD){
    vvl C; newVVl(C, sz(A), sz(A[0]), 0);
    REP(i, sz(A)) REP(j, sz(A[0])) REP(k, sz(A[0])){
      C[i][j] += (A[i][k]*B[k][j]);
      C[i][j] %= mod;
    }
    return C;
  }
  vvl matEye(int n){
    vvl A; newVVl(A, n, n, 0);
    REP(i, n) A[i][i] = 1;
    return A;
  }
  vvl matExp(const vvl& A, ll p, ll mod=MOD){
    vvl res = matEye(sz(A));
    vvl base = matCopy(A);
    while (p){
      if (p&1) res = matMul(res, base, mod);
      base = matMul(base, base, mod);
      p /= 2;
    }
    return res;
  }
};
using namespace matrix;
```

7.7 Modular Matrix Inversion

```
using namespace modulo;
namespace mod_matrix_inv{
  // Do gaussian elimination ON mat and return determinant+rank
  pair<ll, int> gauss(vvl& mat, ll mod=MOD){
    ll prod = 1;
    int nex = 0;
    int rows = sz(mat);
    int rank = rows;
    int cols = sz(mat[0]);
    REP(i, rows){
      int row = -1;
      FOR(j, nex, rows) if (modPos(mat[j][i], mod)){ row = j; break; }
      if (row == -1) { rank --; prod = 0; continue; }
      if (row != nex) prod *= -1, mat[row].swap(mat[nex]);
      prod = modMul(prod, mat[nex][i], mod);
      ll inv = modInv(mat[nex][i], mod);
      FOR(k, i, cols) mat[nex][k]=modMul(mat[nex][k], inv, mod);
      REP(k, rows) if (k!=nex){
        ll x = mat[k][i];
        FOR(j, i, cols) mat[k][j] -= modMul(x, mat[nex][j], mod);
      }
      nex ++;
    }
    REP(i, rows) REP(j, cols) mat[i][j] = modPos(mat[i][j], mod);
    return {modPos(prod, mod), rank};
  }

  vvl matInv(vvl& mat, ll mod=MOD){
    int n = sz(mat);
    vvl eq (n, vl(2*n, 0));
    REP(i, n) REP(j, n) eq[i][j] = mat[i][j];
    REP(i, n) eq[i][i+n] = 1;
    if (gauss(eq, mod).F == 0) return {};
    vvl inv (n, vl(n, 0));
    REP(i, n) REP(j, n) inv[i][j] = eq[i][j+n];
    return inv;
  }
}
```

7.8 Modulo

```
namespace modulo{
  ll gcd(ll a, ll b) {
      if (a == 0) return b;
      return gcd(b%a, a);
  }

  ll lcm(ll a, ll b){
    return a/gcd(a,b)*b;
  }

  // return (x,y) such that ax+by=gcd(a,b)>0
  pl gcdExtended(ll a, ll b){
    if (a == 0) return {0, 1};
    pl temp = gcdExtended(b%a, a);
    ll x = temp.S-(b/a)*temp.F;
    ll y = temp.F;
    if (a*x+b*y > 0) return {x, y};
    else return {-x, -y};
    return {temp.S-(b/a)*temp.F, temp.F};
  }

  // get positive value of modulo
  ll modPos(ll a, ll mod=MOD){
    if (a >= 0) return a%mod;
    return (a%mod+mod)%mod;
  }

  // return x such that ax~1(mod)
  // make sure gcd(a,mod)=1
  ll modInv(ll a, ll mod) {
    return gcdExtended(a, mod).F;
  }

  ll modMul(ll a, ll b, ll mod=MOD){
    b=modPos(b, mod);
    a %= mod;
    if (b == 0) return 0;
    if ((1L<<63)/b>abs(a)) return (a*b)%mod;
    ll res = 0;
    while (b){
      if (b&1) res = (res+a)%mod;
      a = (2*a)%mod;
      b >>= 1;
    }
    return res;
  }

  ll modExp(ll b, ll p, ll mod=MOD){
    ll res = 1;
    ll base = b;
    while (p){
      if (p&1) res = modMul(res, base, mod);
      base = modMul(base, base, mod);
      p >>= 1;
    }
    return res;
  }

  //Highest power of p in n!
  ll getLegendre(ll n, ll p){
    ll res = 0; n /= p;
    while (n){
      res += n;
      n /= p;
    }
```

```
    return res;
  }

  // Solving x^a.F(a.S) and x^b.F(b.S)
  // Return {x,l} where x in modulo of l
  // Return {-1,-1} if no solution
  pl CRT(pl a, pl b) {
    ll g = gcd(a.S,b.S), l = a.S/g*b.S;
    if ((b.F-a.F) % g != 0) return {-1,-1};
    ll A = a.S/g, B = b.S/g;
    ll mul = (b.F-a.F)/g*modInv(A%B,B) % B;
    return {((modMul(mul,a.S,l)+a.F)%l+l)%l,l};
  }
}
```

7.9 Pollard Rho

```
using namespace modulo;
namespace pollard{
        sieve<1<<20> S = sieve<1<<20>(); // should take care of all primes up to n^(1/3)

        bool prime(ll p) { // miller-rabin
                if (p == 2) return true;
                if (p == 1 || p % 2 == 0) return false;
                ll s = p - 1;
                while (s % 2 == 0) s /= 2;
                REP(i,15) {
                        ll a = rand() % (p - 1) + 1, tmp = s;
                        ll mod = modExp(a, tmp, p);
                        while (tmp != p - 1 && mod != 1 && mod != p - 1) {
                                mod = modMul(mod, mod, p);
                                tmp *= 2;
                        }
                        if (mod != p - 1 && tmp % 2 == 0) return false;
                }
                return true;
        }

        ll f(ll a, ll n) { return (modMul(a, a, n) + 1) % n; }

        vpl factorize(ll n) { // can factor up to maximum ll
                vpl res;
                vl pr = S.primes;
                for (int i = 0; i < sz(pr) && pr[i]*pr[i] <= n; i++) if (n % pr[i] == 0) {
                     int co = 0;
                        while (n % pr[i] == 0) n /= pr[i], co ++;
                        res.pb({pr[i],co});
                }

                if (n > 1) { // d is now a product of at most 2 primes.
                        // pollard rho factorization
                        if (prime(n)) res.pb({n,1});
                        else while (1) {
                                ll x = 2, y = 2, d = 1;
        while (d == 1){
          x = f(x, n);
          y = f(f(y, n), n);
          d = gcd(abs(x-y), n);
        }
                                if (d != n) {
                                    n /= d; if (n > d) swap(n,d);
                                    if (n == d) res.pb({d,2});
                                    else res.pb({d,1}), res.pb({n,1});
                                        break;
                                }
                        }
                }
                return res;
        }
}
```

7.10 Sieve

```
template<int SZ> struct sieve {
        vl factors; // contains smallest prime factors
        vl primes; // contains primes
        vl phi; // euler totient
        vl mobius; // mobius function
        sieve(){
                REP(i, SZ+1){
                        factors.pb(i);
                        phi.pb(i-1);
                        mobius.pb(-1);
                }
                phi[1] = 1;
                mobius[1] = 1;
                FOR(i, 2, SZ+1){
                        if (factors[i] == i) primes.push_back(i);
                        for (int j = 0; j < primes.size() && i * primes[j] < SZ+1; j++) {
                                factors[i * primes[j]] = primes[j];
                                if (i % primes[j] == 0){
                                        phi[i * primes[j]] = phi[i] * primes[j];
                                        mobius[i * primes[j]] = 0;
                                        break;
                                } else{
                                        phi[i * primes[j]] = phi[i] * phi[primes[j]];
                                        mobius[i * primes[j]] = mobius[i] * mobius[primes[j]];
                                }
                        }
                }
        }
        // Factorize in O(logn) given smallest factors if n<sz(factors)
        // O(n^1/2) otherwise
        vpl factorize(ll n){
                vpl facs;
                if (n >= sz(factors)){
                        EACH(p, primes){
                                if (p * p > n) break;
                                int cnt = 0;
                                while (n%p == 0){
                                        cnt ++;
                                        n /= p;
                                }
                                if (cnt) facs.pb({p, cnt});
                        }
                }
                while (n > 1 && n < sz(factors)){
                        int p = factors[n];
                        int cnt = 0;
                        while (n%p == 0){
                                cnt ++;
                                n /= p;
                        }
                        if (cnt) facs.pb({p, cnt});
                }
                if (n > 1) facs.pb({n, 1});
                return facs;
        }
};
```

8 Misc

8.1 2-SAT

```
//kosaraju scc dependency
template<int SZ> struct twosat {
    scc<2*SZ> S;
    int N;

    // for each symbol i, 2*i is the positive sym, 2*i+1 is the negative sym
    // Example CNF: (0v~1)(~2v3)
    // equivalent of OR(0,3) and OR(5,6)
    void OR(int x, int y) { S.addEdge(x^1,y); S.addEdge(y^1,x); }

    int tmp[2*SZ];
    bitset<SZ> ans;

    bool solve() {
      S.N = 2*N; S.genSCC();
        for (int i = 0; i < 2*N; i += 2) if (S.comp[i] == S.comp[i^1]) return 0;
        REP(i, N){
        int posComp = S.comp[i<<1];
        int negComp = S.comp[(i<<1)^1];
        if (tmp[posComp] == 0) tmp[posComp] = 1, tmp[negComp] = -1;
      }
          REP(i,N) if (tmp[S.comp[i<<1]] == 1) ans[i] = 1;
        return 1;
    }
};
```

8.2 Bit Manipulations

```
// Lowest set bit
```

```
int Debruijn[32] = {
  0, 1, 28, 2, 29, 14, 24, 3, 30, 22, 20, 15, 25, 17, 4, 8,
  31, 27, 13, 23, 21, 19, 16, 7, 26, 12, 18, 6, 11, 5, 10, 9
};
int lsb(int n){
  return 1 << Debruijn[((uint32_t)((n & -n) * 0x077CB531U)) >> 27];
}

// Highest set bit
int hsb(int n){
  n |= n >> 1;
  n |= n >> 2;
  n |= n >> 4;
  n |= n >> 8;
  n |= n >> 16;
  n ++;
  n <<= 1;
  return n;
}
```

8.3 Index Compression

```
// compress values to the number of different values
// For example A=[1,10,100,1000,10000]
// Then compress(A)=[2,4,6,8,10]
// The odd values represent values between two consecutive values
// Enc=encrypt original value to compressed value
// Dec=decrypt compressed value to original value
struct compress{
  vi v;
  void getV(us<int>& set){
    v.resize(sz(set)+2); int ind = 0;
    EACH(s, set) v[ind]=s, ind ++;
    v[ind] = INF; v[ind+1] = -INF;
    sort(all(v));
  }
  compress(int* from, int* to){
    us<int> set;
    for(auto it = from; it != to; it++) set.insert(*it);
    getV(set);
  }
  compress(vi& a){
    us<int> set; EACH(i, a) set.insert(i);
    getV(set);
  }
  int enc(int x){
    auto low = lb(all(v), x);
    if (x == *low) return (low-v.begin())*2;
    else return (low-v.begin())*2-1;
  }
  int dec(int ind){ return v[ind/2];}
};
```

8.4 Longest Increasing Subsequence

```
// Return list of indice of longest strictly increasing sequecne in O(nlogn)
vi liss(vi& v){
  auto cmp = [&](int x, int y){
    return v[x] < v[y];
  };
  set<int,decltype(cmp)> S(cmp);
  vi pre(sz(v), -1);
  REP(i, sz(v)){
    auto it = S.insert(i).first;
    if (it != S.begin()) pre[i] = *prev(it);
    if (*it == i && next(it) != S.end()) S.erase(next(it));
  }
  vi ans;
  ans.pb(*S.rbegin());
  while (pre[ans.back()] != -1) ans.pb(pre[ans.back()]);
  reverse(all(ans));
  return ans;
}

// Return list of indice of longest non-decreasing sequence in O(nlogn)
vi lis(vi& v){
  auto cmp = [&](int x, int y){
    return v[x] < v[y];
  };
  multiset<int,decltype(cmp)> S(cmp);
  vi pre(sz(v), -1);
  REP(i, sz(v)){
    auto it = S.insert(i);
    if (it != S.begin()) pre[i] = *prev(it);
    if (*it == i && next(it) != S.end()) S.erase(next(it));
  }
  vi ans;
  ans.pb(*S.rbegin());
  while (pre[ans.back()] != -1) ans.pb(pre[ans.back()]);
  reverse(all(ans));
  return ans;
}
```

8.5 Mo's Algorithm

```
typedef array<int,3> query;
template<int SZ> struct moAlgo{
  int N, ans[SZ]; // setN
  vector<query> todo; //add query [L,R]
  bool compare(query& a1, query& a2){
    int sq = (int)floor(sqrt(N));
    if (a1[0]/sq != a2[0]/sq) return a1[0] < a2[0];
    return a1[1] < a2[1];
  }
  void updAns(){
    //Update your answer depends on whether you move your interval
  }
  void solve(){
    sort(all(todo), [this](query& a, query& b){return compare(a, b);});
    int l = 0, r = -1; um<string, int> cntMap;
    REP(i, sz(todo)){
      int ind = todo[i][2]; //query index
      int L = todo[i][0];
      int R = todo[i][1];
      while (r<R) updAns();
      while (r>R) updAns();
      while (l>L) updAns();
      while (l<L) updAns();
      ans[ind]=//Set answer//
    }
  }
};
```

8.6 Random

```
//PRNG
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
//Random shuffle a vector 'permutation'
shuffle(permutation.begin(), permutation.end(), rng);
// generate a random integer from start to end inclusive using our PRNG
uniform_int_distribution<int>(start, end)(rng)
```

8.7 Splitting Strings

```
// Split a string s using the delimeter c
vector<string> split(const string& s, char c) {
    vector<string> v;
    stringstream ss(s);
    string x;
    while (getline(ss, x, c))
        v.eb(x); //emplace_back
    return std::move(v);
}
```

9 String

9.1 Aho-Corasick

```
/**
 * Find the occurences of a set of strings in a text.
 * Verification: https://open.kattis.com/problems/stringmultimatching
*/

template<int SZ> struct AhoCorasick {
    int link[SZ], dict[SZ], sz = 1, num = 0;
```

```
    vpi ind[SZ];
    map<char,int> to[SZ];
    vi oc[SZ];
    queue<int> q;

    AhoCorasick() {
        memset(link,0,sizeof link);
        memset(dict,0,sizeof dict);
    }

    void add(string s) { // add a string from the set.
        int v = 0;
        EACH(c, s){
            if (!to[v].count(c)) to[v][c] = sz++;
            v = to[v][c];
        }
        dict[v] = v; ind[v].pb({++num,sz(s)});
    }

    void pushLinks() {  // call after adding all strings
        link[0] = -1; q.push(0);
        while (sz(q)) {
            int v = q.front(); q.pop();
            EACH(it, to[v]){
                char c = it.F; int u = it.S, j = link[v];
                while (j != -1 && !to[j].count(c)) j = link[j];
                if (j != -1) {
                    link[u] = to[j][c];
                    if (!dict[u]) dict[u] = dict[link[u]];
                }
                q.push(u);
            }
        }
    }

    void process(int pos, int cur) { // process matches
        cur = dict[cur];
        while (cur) {
            for (auto a: ind[cur]) oc[a.F].pb(pos-a.S+1);
            cur = dict[link[cur]];
        }
    }

    int nex(int pos, int cur, char c) {
        // get position after adding character
        // speed up with memoization
        while (cur != -1 && !to[cur].count(c)) cur = link[cur];
        if (cur == -1) cur = 0;
        else cur = to[cur][c];
        process(pos, cur);
        return cur;
    }
};

/**
 * Input:
 * int n
 * n strings of length at most 1e5 on each line.
 * 1 string of length at most 2e5 on a line.
 * repeat until eof.
 * Output:
 * n lines, each line has the occurances of the ith string
 * in the text. (Empty if no occurences).
 */

const int MAXLEN = 200005;
int n;
string str;

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);
  while(cin >> n) {
    cin.ignore();
    AhoCorasick<MAXLEN> aho;
    REP(i, n) {
      getline(cin, str);
      aho.add(str);
    }
    aho.pushLinks();
    getline(cin, str);
    int cur = 0;
    REP(i, str.length()) cur = aho.nex(i, cur, str[i]);
    FOR(i,1,n+1) {
      for (int j: aho.oc[i]) cout << j << " ";
      cout << "\n";
    }
  }
  return 0;
}
```

9.2 KMP

```
/**
 * String Mathcing. Find occurences of a pattern in a text.
 */

vi prefix_function(string s) {
    int n = (int)s.length();
    vi pi(n);
    FOR(i, 1, n) {
      int j = pi[i-1];
      while(j > 0 && s[i] != s[j])
        j = pi[j-1];
      if(s[i] == s[j])
        j++;
      pi[i] = j;
    }
    return pi;
}

bool kmp(string pattern, string text) {
  vi pi = prefix_function(pattern + "#" + text);
  REP(i, pi.size()){
    if(pi[i] == pattern.length()) {
        // an occurence of pattern exists
        // at i - (2*pattern.length()) in text
        return true;
    }
  }
  return false;
}
```

9.3 Manacher

```
/**
* Description: Calculates length of largest palindrome centered at each character of string
* O(n)
*/

vi manacher(string s) {
    string s1 = "$";
    for (char c: s) s1 += c, s1 += "#";
    s1[s1.length()-1] = '&';

    vi ans(s1.length()-1);
    int lo = 0, hi = 0;
    FOR(i,1,s1.length()-1) {
        if (i != 1) ans[i] = min(hi-i,ans[hi-i+lo]);
        while (s1[i-ans[i]-1] == s1[i+ans[i]+1]) ans[i] ++;
        if (i+ans[i] > hi) lo = i-ans[i], hi = i+ans[i];
    }

    ans.erase(ans.begin());
    REP(i,sz(ans)) if ((i&1) == (ans[i]&1)) ans[i] ++; // adjust lengths
    return ans;
}
```

9.4 Trie

```
/**
 * Trie, the string data structure
 */

const int MAXN = 100005; // number of nodes
const int MAXNODES = 200005; // total number of letters of strings added to trie

int trieSize = 0;
struct TrieNode {
    int next[26], numEndNodes;
    bool isEndNode;
    TrieNode() {
        memset(next, 0, sizeof(next));
        numEndNodes = 0;
        isEndNode = false;
    }
} nodes[MAXNODES];

void add(string str) {
    int k = 0;
    REP(i, str.length()) {
```

```
        char c = str[i];
        int t = c-'a';
        if(nodes[k].next[t] == 0) {
            nodes[k].next[t] = ++trieSize;
            nodes[trieSize] = TrieNode();
        }
        nodes[k].numEndNodes++;
        k = nodes[k].next[t];
    }
    nodes[k].isEndNode = true;
    nodes[k].numEndNodes++;
}

// Returns true if str is in the trie
bool search(string str) {
    int k = 0;
    REP(i, str.length()) {
        char c = str[i];
        int t = c-'a';
        if(nodes[k].next[t] == 0) {
            return false;
        }
        k = nodes[k].next[t];
    }
    return nodes[k].isEndNode;
}
```