TCR

1 Setup

1.1 header.h

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

#define ll long long
#define ull unsigned ll
#define ld long double
#define pl pair<ll, ll>
#define pi pair<int, int>
#define vl vector<ll>
#define vi vector<int>
#define vvi vector<vi>
#define vvl vector<vl>
#define vpl vector<pl>
#define vpi vector<pi>
#define vld vector<ld>
#define in(el, cont) (cont.find(el) != cont.end())

constexpr int INF  = 2000000010;
constexpr ll LLINF = 9000000000000000010LL;

template <typename T, template <typename ELEM, typename ALLOC = std::allocator<ELEM> > class Container>
std::ostream& operator<<(std::ostream& o, const Container<T>& container) {
  typename Container<T>::const_iterator beg = container.begin();
  if (beg != container.end()) {
    o << *beg++;
    while (beg != container.end()) {
      o << " " << *beg++;
    }
  }
  return o;
}

// int main() {
//   ios::sync_with_stdio(false);  // do not use cout + printf
//   cin.tie(NULL);
//   cout << fixed << setprecision(12);
//   return 0;
// }
```

1.2 Bash for c++ compile with header.h

```
#!/bin/bash
if [ $# -ne 1 ];then echo "Usage: $0 <input_file>"; exit 1;fi
f="$1";d=code/;o=a.out
[ -f $d/$f ] || { echo "Input file not found: $f"; exit 1; }
g++ -I$d $d/$f -o $o && echo "Compilation successful. Executable '$o' created." || echo "Compilation failed."
```

1.3 Bash for run tests c++

```
g++ $1/$1.cpp -o $1/$1.out
for file in $1/*.in; do diff <($1/$1.out < "$file") "${file%.in}.ans"; done
```

1.4 Bash for run tests python

```
for file in $1/*.in; do diff <(python3 $1/$1.py < "$file") "${file%.in}.ans"; done
```

1.4.1 Auxiliary helper stuff

```cpp
#include "header.h"

int main() {
    // Read in a line including white space
    string line;
    getline(cin, line);
    // When doing the above read numbers as follows:
    int n;
    getline(cin, line);
    stringstream ss(line);
    ss >> n;

    // Count the number of 1s in binary represnatation of a number
    ull number;
    __builtin_popcountll(number);
}
```

2 Python

2.1 Graphs

2.1.1 BFS

```python
from collections import deque
def bfs(g, roots, n):
    q = deque(roots)
    explored = set(roots)
    distances = [float("inf")]*n
    distances[0][0] = 0

    while len(q) != 0:
        node = q.popleft()
        if node in explored: continue
        explored.add(node)
        for neigh in g[node]:
            if neigh not in explored:
                q.append(neigh)
                distances[neigh] = distances[node] + 1
    return distances
```

2.1.2 Dijkstra

```python
from heapq import *
def dijkstra(n, root, g):  # g = {node: (cost, neigh)}
  dist = [float("inf")]*n
  dist[root] = 0
  prev = [-1]*n

  pq = [(0, root)]
  heapify(pq)
  visited = set([])

  while len(pq) != 0:
    _, node = heappop(pq)

    if node in visited: continue
    visited.add(node)

    # In case of disconnected graphs
    if node not in g:
      continue

    for cost, neigh in g[node]:
      alt = dist[node] + cost
      if alt < dist[neigh]:
        dist[neigh] = alt
        prev[neigh] = node
        heappush(pq, (alt, neigh))
  return dist
```

2.2 Dynamic Programming

2.3 Trees

2.4 Number Theory

2.4.1 nCk % prime

```python
# Note: p must be prime and k < p
def fermat_binom(n, k, p):
    if k > n:
        return 0
    # calculate numerator
    num = 1
    for i in range(n-k+1, n+1):
        num *= i % p
    num %= p
    # calculate denominator
    denom = 1
    for i in range(1,k+1):
        denom *= i % p
    denom %= p
    # numerator * denominator^(p-2) (mod p)
    return (num * pow(denom, p-2, p)) % p
```

2.5 Strings

2.5.1 LCS

```python
def longestCommonSubsequence(text1, text2):  # O(m*n) time, O(m) space
    n = len(text1)
    m = len(text2)

    # Initializing two lists of size m
    prev = [0] * (m + 1)
    cur = [0] * (m + 1)

    for idx1 in range(1, n + 1):
        for idx2 in range(1, m + 1):
```

```python
            # If characters are matching
            if text1[idx1 - 1] == text2[idx2 - 1]:
                cur[idx2] = 1 + prev[idx2 - 1]
            else:
                # If characters are not matching
                cur[idx2] = max(cur[idx2 - 1], prev[idx2])

        prev = cur.copy()

    return cur[m]
```

2.6 Geometry

2.7 Combinatorics

2.8 Other Algorithms

2.8.1 Rotate matrix

```python
def rotate_matrix(m):
    return [[m[j][i] for j in range(len(m))] for i in range(len(m[0])
        -1,-1,-1)]
```

2.9 Other Data Structures

2.9.1 Segment Tree

```python
N = 100000  # limit for array size
tree = [0] * (2 * N)  # Max size of tree

def build(arr, n):  # function to build the tree
    # insert leaf nodes in tree
    for i in range(n):
        tree[n + i] = arr[i]

    # build the tree by calculating parents
    for i in range(n - 1, 0, -1):
        tree[i] = tree[i << 1] + tree[i << 1 | 1]

def updateTreeNode(p, value, n):  # function to update a tree node
    # set value at position p
    tree[p + n] = value
    p = p + n

    i = p  # move upward and update parents
    while i > 1:
        tree[i >> 1] = tree[i] + tree[i ^ 1]
        i >>= 1

def query(l, r, n):  # function to get sum on interval [l, r)
    res = 0
    # loop to find the sum in the range
    l += n
    r += n
    while l < r:
        if l & 1:
            res += tree[l]
            l += 1
        if r & 1:
            r -= 1
            res += tree[r]
        l >>= 1
        r >>= 1
    return res
```

2.10 Other Mathematics

3 C++

3.1 Graphs

3.1.1 BFS

```cpp
#include "header.h"
#define graph unordered_map<ll, unordered_set<ll>>
vi bfs(int n, graph& g, vi& roots) {
    vi parents(n+1, -1); // nodes are 1..n
    unordered_set<int> visited;
    queue<int> q;
    for (auto x: roots) {
        q.emplace(x);
        visited.insert(x);
    }
    while (not q.empty()) {
        int node = q.front();
        q.pop();

        for (auto neigh: g[node]) {
            if (not in(neigh, visited)) {
                parents[neigh] = node;
                q.emplace(neigh);
                visited.insert(neigh);
            }
        }
    }
    return parents;
}
vi reconstruct_path(vi parents, int start, int goal) {
    vi path;
    int curr = goal;
    while (curr != start) {
        path.push_back(curr);
        if (parents[curr] == -1) return vi(); // No path, empty vi
        curr = parents[curr];
    }
    path.push_back(start);
    reverse(path.begin(), path.end());
    return path;
}
```

3.1.2 DFS Cycle detection / removal

```cpp
#include "header.h"
void removeCyc(ll node, unordered_map<ll, vector<pair<ll, ll>>>& neighs,
    vector<bool>& visited,
vector<bool>& recStack, vector<ll>& ans) {
    if (!visited[node]) {
        visited[node] = true;
        recStack[node] = true;
        auto it = neighs.find(node);
        if (it != neighs.end()) {
            for (auto util: it->second) {
                ll nnode = util.first;
                if (recStack[nnode]) {
                    ans.push_back(util.second);
                } else if (!visited[nnode]) {
                    removeCyc(nnode, neighs, visited, recStack, ans);
                }
            }
        }
    }
    recStack[node] = false;
}
```

3.1.3 Dijkstra

```cpp
#include "header.h"
vector<int> dijkstra(int n, int root, map<int, vector<pair<int, int>>>& g) {
  unordered_set<int> visited;
  vector<int> dist(n, INF);
    priority_queue<pair<int, int>> pq;
    dist[root] = 0;
    pq.push({0, root});
    while (!pq.empty()) {
        int node = pq.top().second;
        int d = -pq.top().first;
        pq.pop();

        if (in(node, visited)) continue;
        visited.insert(node);

        for (auto e : g[node]) {
            int neigh = e.first;
            int cost = e.second;
            if (dist[neigh] > dist[node] + cost) {
                dist[neigh] = dist[node] + cost;
                pq.push({-dist[neigh], neigh});
            }
        }
    }
    return dist;
}
```

3.1.4 Floyd-Warshall

```cpp
#include "header.h"
// g[i][j] = infty if not path from i to j
// if g[i][i] < 0, i is contained in a negative cycle
void warshall(vvl g) {
    for (int i=0; i<g.size(); ++i) {
        for (int j=0; j<g.size(); ++j) {
            for (int k=0; k<g.size(); ++k) {
                if (g[i][k] < LLINF and g[k][j] < LLINF and g[i][j] > g[i][k
                    ] + g[k][j]) {
                    g[i][j] = g[i][k] + g[k][j];
}}}}}
```

3.1.5 Kruskal Minimum spanning tree of undirected weighted graph

```cpp
#include "header.h"
#include "disjoint_set.h"
// O(E log E)
pair<set<pair<ll, ll>>, ll>  kruskal(vector<tuple<ll, ll, ll>>& edges, ll n)
     {
    set<pair<ll, ll>> ans;
    ll cost = 0;

    sort(edges.begin(), edges.end());
    DisjointSet<ll> fs(n);

    ll dist, i, j;
    for (auto edge: edges) {
        dist = get<0>(edge);
        i = get<1>(edge);
        j = get<2>(edge);

        if (fs.find_set(i) != fs.find_set(j)) {
            fs.union_sets(i, j);
            ans.insert({i, j});
            cost += dist;
        }
    }
    return pair<set<pair<ll, ll>>, ll> {ans, cost};
}
```

3.1.6 Hungarian algorithm

```cpp
#include "header.h"

template <class T> bool ckmin(T &a, const T &b) { return b < a ? a = b, 1 :
    0; }
/**
 * Given J jobs and W workers (J <= W), computes the minimum cost to assign
     each
 * prefix of jobs to distinct workers.
 * @tparam T a type large enough to represent integers on the order of J *
 * max(|C|)
 * @param C a matrix of dimensions JxW such that C[j][w] = cost to assign j-
     th
 * job to w-th worker (possibly negative)
 *
 * @return a vector of length J, with the j-th entry equaling the minimum
     cost
```

```
 * to assign the first (j+1) jobs to distinct workers
 */
template <class T> vector<T> hungarian(const vector<vector<T>> &C) {
    const int J = (int)size(C), W = (int)size(C[0]);
    assert(J <= W);
    // job[w] = job assigned to w-th worker, or -1 if no job assigned
    // note: a W-th worker was added for convenience
    vector<int> job(W + 1, -1);
    vector<T> ys(J), yt(W + 1);  // potentials
    // -yt[W] will equal the sum of all deltas
    vector<T> answers;
    const T inf = numeric_limits<T>::max();
    for (int j_cur = 0; j_cur < J; ++j_cur) {  // assign j_cur-th job
        int w_cur = W;
        job[w_cur] = j_cur;
        // min reduced cost over edges from Z to worker w
        vector<T> min_to(W + 1, inf);
        vector<int> prv(W + 1, -1);  // previous worker on alternating path
        vector<bool> in_Z(W + 1);    // whether worker is in Z
        while (job[w_cur] != -1) {   // runs at most j_cur + 1 times
            in_Z[w_cur] = true;
            const int j = job[w_cur];
            T delta = inf;
            int w_next;
            for (int w = 0; w < W; ++w) {
                if (!in_Z[w]) {
                    if (ckmin(min_to[w], C[j][w] - ys[j] - yt[w]))
                        prv[w] = w_cur;
                    if (ckmin(delta, min_to[w])) w_next = w;
                }
            }
            // delta will always be non-negative,
            // except possibly during the first time this loop runs
            // if any entries of C[j_cur] are negative
            for (int w = 0; w <= W; ++w) {
                if (in_Z[w]) ys[job[w]] += delta, yt[w] -= delta;
                else min_to[w] -= delta;
            }
            w_cur = w_next;
        }
        // update assignments along alternating path
        for (int w; w_cur != W; w_cur = w) job[w_cur] = job[w = prv[w_cur]];
        answers.push_back(-yt[W]);
    }
    return answers;
}
```

3.2 Dynamic Programming

3.3 Trees

3.4 Number Theory

3.4.1 Modular exponentiation Or use pow() in python

```
#include "header.h"
ll mod_pow(ll base, ll exp, ll mod) {
  if (mod == 1) return 0;
    if (exp == 0) return 1;
    if (exp == 1) return base;

  ll res = 1;
  base %= mod;
  while (exp) {
    if (exp % 2 == 1) res = (res * base) % mod;
    exp >>= 1;
    base = (base * base) % mod;
  }

  return res % mod;
}
```

3.4.2 GCD Or use math.gcd() in python

```
#include "header.h"
ll gcd(ll a, ll b) {
  if (a == 0) {
    return b;
  }
  return gcd(b % a, a);
}
```

3.4.3 Sieve of Eratosthenes

```
#include "header.h"

vector<ll> primes;
void getprimes(ll n) {
    vector<bool> p(n, true);
    p[0] = false;
    p[1] = false;
    for(ll i = 0; i < n; i++) {
        if(p[i]) {
            primes.push_back(i);
            for(ll j = i*2; j < n; j+=i) {
                p[j] = false;
            }
        }
    }
}
```

3.4.4 Fibonacci % prime

```
#include "header.h"
const ll MOD = 1000000007;
unordered_map<ll, ll> Fib;
ll fib(ll n) {
    if (n < 2) return 1;
    if (Fib.find(n) != Fib.end()) return Fib[n];
    Fib[n] = (fib((n + 1) / 2) * fib(n / 2) + fib((n - 1) / 2) * fib((n - 2)
         / 2)) % MOD;
    return Fib[n];
}
```

3.4.5 nCk % prime

```cpp
#include "header.h"
ll binom(ll n, ll k) {
    ll ans = 1;
    for(ll i = 1; i <= min(k,n-k); ++i) ans = ans*(n+1-i)/i;
    return ans;
}
ll mod_nCk(ll n, ll k, ll p ){
    ll ans = 1;
    while(n){
        ll np = n%p, kp = k%p;
        if(kp > np) return 0;
        ans *= binom(np,kp);
        n /= p; k /= p;
    }
    return ans;
}
```

3.5 Strings

3.5.1 Aho-Corasick algorithm

Also can be used as Knuth-Morris-Pratt algorithm

```cpp
#include "header.h"

map<char, int> cti;
int cti_size;
template <int ALPHABET_SIZE, int (*mp)(char)>
struct AC_FSM {
  struct Node {
    int child[ALPHABET_SIZE], failure = 0, match_par = -1;
    vi match;
    Node() { for (int i = 0; i < ALPHABET_SIZE; ++i) child[i] = -1; }
  };
  vector<Node> a;
  vector<string> &words;
  AC_FSM(vector<string> &words) : words(words) {
    a.push_back(Node());
    construct_automaton();
  }
  void construct_automaton() {
    for (int w = 0, n = 0; w < words.size(); ++w, n = 0) {
      for (int i = 0; i < words[w].size(); ++i) {
        if (a[n].child[mp(words[w][i])] == -1) {
          a[n].child[mp(words[w][i])] = a.size();
          a.push_back(Node());
        }
        n = a[n].child[mp(words[w][i])];
      }
      a[n].match.push_back(w);
    }
    queue<int> q;
    for (int k = 0; k < ALPHABET_SIZE; ++k) {
      if (a[0].child[k] == -1) a[0].child[k] = 0;
      else if (a[0].child[k] > 0) {
        a[a[0].child[k]].failure = 0;
        q.push(a[0].child[k]);
      }
    }
    while (!q.empty()) {
      int r = q.front(); q.pop();
      for (int k = 0, arck; k < ALPHABET_SIZE; ++k) {
        if ((arck = a[r].child[k]) != -1) {
          q.push(arck);
          int v = a[r].failure;
          while (a[v].child[k] == -1) v = a[v].failure;
          a[arck].failure = a[v].child[k];
          a[arck].match_par = a[v].child[k];
          while (a[arck].match_par != -1
              && a[a[arck].match_par].match.empty())
            a[arck].match_par = a[a[arck].match_par].match_par;
        }
      }
    }
  }
  void aho_corasick(string &sentence, vvi &matches){
    matches.assign(words.size(), vi());
    int state = 0, ss = 0;
    for (int i = 0; i < sentence.length(); ++i, ss = state) {
      while (a[ss].child[mp(sentence[i])] == -1)
        ss = a[ss].failure;
      state = a[state].child[mp(sentence[i])]
          = a[ss].child[mp(sentence[i])];
      for (ss = state; ss != -1; ss = a[ss].match_par)
        for (int w : a[ss].match)
          matches[w].push_back(i + 1 - words[w].length());
    }
  }
};
int char_to_int(char c) {
  return cti[c];
}
```

3.6 Geometry

3.6.1 essentials.cpp

```cpp
#include "../header.h"
using C = ld; // could be long long or long double
constexpr C EPS = 1e-10;  // change to 0 for C=ll
struct P {    // may also be used as a 2D vector
  C x, y;
  P(C x = 0, C y = 0) : x(x), y(y) {}
  P operator+ (const P &p) const { return {x + p.x, y + p.y}; }
  P operator- (const P &p) const { return {x - p.x, y - p.y}; }
  P operator* (C c) const { return {x * c, y * c}; }
  P operator/ (C c) const { return {x / c, y / c}; }
  C operator* (const P &p) const { return x*p.x + y*p.y; }
  C operator^ (const P &p) const { return x*p.y - p.x*y; }
  P perp() const { return P{y, -x}; }
  C lensq() const { return x*x + y*y; }
  ld len() const { return sqrt((ld)lensq()); }
  static ld dist(const P &p1, const P &p2) {
    return (p1-p2).len(); }
  bool operator==(const P &r) const {
    return ((*this)-r).lensq() <= EPS*EPS; }
```

```
};
C det(P p1, P p2) { return p1^p2; }
C det(P p1, P p2, P o) { return det(p1-o, p2-o); }
C det(const vector<P> &ps) {
  C sum = 0; P prev = ps.back();
  for(auto &p : ps) sum += det(p, prev), prev = p;
  return sum;
}
// Careful with division by two and C=ll
C area(P p1, P p2, P p3) { return abs(det(p1, p2, p3))/C(2); }
C area(const vector<P> &poly) { return abs(det(poly))/C(2); }
int sign(C c){ return (c > C(0)) - (c < C(0)); }
int ccw(P p1, P p2, P o) { return sign(det(p1, p2, o)); }

// Only well defined for C = ld.
P unit(const P &p) { return p / p.len(); }
P rotate(P p, ld a) { return P{p.x*cos(a)-p.y*sin(a), p.x*sin(a)+p.y*cos(a)}; }
```

3.6.2 Convex Hull

```
#include "header.h"
#include "essentials.cpp"
struct ConvexHull {  // O(n lg n) monotone chain.
  size_t n;
  vector<size_t> h, c;  // Indices of the hull are in `h`, ccw.
  const vector<P> &p;
  ConvexHull(const vector<P> &_p) : n(_p.size()), c(n), p(_p) {
    std::iota(c.begin(), c.end(), 0);
    std::sort(c.begin(), c.end(), [this](size_t l, size_t r) -> bool {
        return p[l].x != p[r].x ? p[l].x < p[r].x : p[l].y < p[r].y; });
    c.erase(std::unique(c.begin(), c.end(), [this](size_t l, size_t r) {
        return p[l] == p[r]; }), c.end());
    for (size_t s = 1, r = 0; r < 2; ++r, s = h.size()) {
      for (size_t i : c) {
        while (h.size() > s && ccw(p[h.end()[-2]], p[h.end()[-1]], p[i]) <= 0)
          h.pop_back();
        h.push_back(i);
      }
      reverse(c.begin(), c.end());
    }
    if (h.size() > 1) h.pop_back();
  }
  size_t size() const { return h.size(); }
  template <class T, void U(const P &, const P &, const P &, T &)>
  void rotating_calipers(T &ans) {
    if (size() <= 2)
      U(p[h[0]], p[h.back()], p[h.back()], ans);
    else
      for (size_t i = 0, j = 1, s = size(); i < 2 * s; ++i) {
        while (det(p[h[(i + 1) % s]] - p[h[i % s]], p[h[(j + 1) % s]] - p[h[j]]) >= 0)
          j = (j + 1) % s;
        U(p[h[i % s]], p[h[(i + 1) % s]], p[h[j]], ans);
      }
  }
};
// Example: furthest pair of points. Now set ans = 0LL and call
// ConvexHull(pts).rotating_calipers<ll, update>(ans);
void update(const P &p1, const P &p2, const P &o, ll &ans) {
  ans = max(ans, (ll)max((p1 - o).lensq(), (p2 - o).lensq()));
}
```

3.7 Combinatorics

3.8 Other Algorithms

3.9 Other Data Structures

3.9.1 Disjoint set (i.e. union-find)

```
template <typename T>
class DisjointSet {
    typedef T * iterator;
    T *parent, n, *rank;
    public:
        // O(n), assumes nodes are [0, n)
        DisjointSet(T n) {
            this->parent = new T[n];
            this->n = n;
            this->rank = new T[n];

            for (T i = 0; i < n; i++) {
                parent[i] = i;
                rank[i] = 0;
            }
        }

        // O(log n)
        T find_set(T x) {
            if (x == parent[x]) return x;
            return parent[x] = find_set(parent[x]);
        }

        // O(log n)
        void union_sets(T x, T y) {
            x = this->find_set(x);
            y = this->find_set(y);

            if (x == y) return;

            if (rank[x] < rank[y]) {
                T z = x;
                x = y;
                y = z;
            }

            parent[y] = x;
            if (rank[x] == rank[y]) rank[x]++;
        }
};
```

3.9.2 Fenwick tree (i.e. BIT) eff. update + prefix sum calc.

```
#include "header.h"
#define maxn 200010
int t,n,m,tree[maxn],p[maxn];

void update(int k, int z) {
    while (k <= maxn) {
        tree[k] += z;
        k += k & (-k);
        // cout << "k: " << k << endl;
    }
}

int sum(int k) {
    int ans = 0;
    while(k) {
        ans += tree[k];
        k -= k & (-k);
    }
    return ans;
}
```

3.10 Other Mathematics