# Contents

# 1 Basic

## 1.1 Default

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

using ll = long long;
using ull = unsigned long long;
using ld = long double;
using uint = unsigned int;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
using vi = vector<int>;
using vl = vector<ll>;
using vvi = vector<vector<int>>;
using vvl = vector<vector<ll>>;
#define pb push_back
#define F first
#define S second
#define mid ((LB+RB)/2)
#define mkp make_pair
#define iter(x) x.begin(),x.end()
#define aiter(a,n) a,a+n
#define REP(n) for (int ___=n > 0 ? n : 0;___--;)
#define REP0(i,n) for (int i=0,___=n;i<___;++i)
#define REP1(i,n) for (int i=1,___=n;i<=___;++i)
#define MEM(e,val) memset (e,val,sizeof(e))
const double EPS = 1e-8;
const int INF = 0x3F3F3F3F;
const ll LINF = 4611686018427387903;
const int MOD = 1e9+7;
const int maxn = 1e5 + 25;

signed main() { ios::sync_with_stdio(0); cin.tie(0);

}
```

## 1.2 vimrc

```vim
set nu rnu is ls=2 hls ts=4 sw=4 et sts=4 ai bs=2 et sc
     acd mouse=a encoding=utf-8
syn on
filetype plugin indent on
colo desert
nnoremap <C-a> ggVG
vnoremap <C-c> "+y
inoremap <C-v> <ESC>"+pa
nnoremap <C-s> :w<CR>
inoremap <C-s> <ESC>:w<CR>a
inoremap {<CR> {<CR>}<Esc>O
nnoremap <F8> :w <bar> !g++ -std=c++17 % -o %:r -O2<CR>
nnoremap <F9> :w <bar> !g++ -std=c++17 % -o %:r -Wall -
    Wextra -Wconversion -Wshadow -Wfatal-errors -
    fsanitize=undefined,address -g -Dmichan <CR>
nnoremap <F10> :!./%:r <CR>
```

## 1.3 Pragma

```cpp
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC optimize("no-math-errno,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target("popcnt,abm,mmx,avx,tune=native")
```

# 2 Data Structure

## 2.1 Black Magic

```cpp
template<typename T>
using pbds_tree = tree<T, null_type, less<T>,
   rb_tree_tag, tree_order_statistics_node_update>;
// find_by_order: like array accessing, order_of_key
```

## 2.2 Linear Basis

```cpp
template<int BITS>
struct linear_basis {
  array<uint64_t, BITS> basis;
  linear_basis() { basis.fill(0); }
  void add(uint64_t x) {
    for(int i = BITS - 1; i >= 0; i--) if((x >> i) & 1)
     {
      if(basis[i] == 0) {
        basis[i] = x;
        continue;
      }
      x ^= basis[i];
    }
  }
  bool valid(uint64_t x) {
    for(int i = BITS - 1; i >= 0; i--)
      if((x >> i) & 1) x ^= basis[i];
    return x == 0;
  }
  // max xor sum: xor sum of all basis
  // min xor sum: zero(if possible) or min_element
}; // not tested
```

# 3 Graph

## 3.1 Dinic

```cpp
template<typename T>
struct dinic{
  const T IN_INF = (is_same_v<T, int>) ? INF : LINF;
  struct E{
    int v; T c; int r;
    E(int v, T c, int r):
      v(v), c(c), r(r){}
  };
  vector<E> adj[maxn];
  pair<int, int> is[maxn]; // counts of edges
  void add_edge(int u, int v, T c, int i){
    is[i] = {u, adj[u].size()};
    adj[u].pb(E(v, c, (int) adj[v].size()));
    adj[v].pb(E(u, 0, (int) adj[u].size() - 1));
  }
  int n, s, t;
  void init(int nn, int ss, int tt){
    n = nn, s = ss, t = tt;
    for(int i = 0; i <= n; ++i)
      adj[i].clear();
  }
  int le[maxn], it[maxn];
  int bfs(){
    fill(le, le + maxn, -1); le[s] = 0;
    queue<int> q; q.push(s);
    while(!q.empty()){
      int u = q.front(); q.pop();
      for(auto [v, c, r]: adj[u]){
        if(c > 0 && le[v] == -1)
          le[v] = le[u] + 1, q.push(v);
      }
    }
    return ~le[t];
  }
  int dfs(int u, int f){
    if(u == t) return f;
    for(int &i = it[u]; i < (int) adj[u].size(); ++i){
      auto &[v, c, r] = adj[u][i];
      if(c > 0 && le[v] == le[u] + 1){
        int d = dfs(v, min(c, f));
        if(d > 0){
          c -= d;
          adj[v][r].c += d;
          return d;
        }
      }
    }
    return 0;
  }
  T flow(){
    T ans = 0, d;
    while(bfs()){
      fill(it, it + maxn, 0);
      while((d = dfs(s, IN_INF)) > 0) ans += d;
    }
    return ans;
  }
  T rest(int i) {
    return adj[is[i].first][is[i].second].c;
  }
};
```

## 3.2 Min Cost Max Flow

```
struct cost_flow {
  static const int MXN = 1005;
  static const int64_t INF = 102938475610293847LL;
  struct Edge {
    int v, r;
    int64_t f, c;
    Edge(int a,int b,int _c,int d):v(a),r(b),f(_c),c(d)
    { }
  };
  int n, s, t, prv[MXN], prvL[MXN], inq[MXN];
  int64_t dis[MXN], fl, cost;
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t) {
    n = _n; s = _s; t = _t;
    for (int i = 0; i < n; i++) E[i].clear();
    fl = cost = 0;
  }
  void add_edge(int u, int v, int64_t f, int64_t c) {
    E[u].push_back(Edge(v, E[v].size() , f, c));
    E[v].push_back(Edge(u, E[u].size()-1, 0, -c));
  }
  pair<int64_t, int64_t> flow() {
    while (true) {
      for (int i = 0; i < n; i++) {
        dis[i] = INF;
        inq[i] = 0;
      }
      dis[s] = 0;
      queue<int> que;
      que.push(s);
      while (!que.empty()) {
        int u = que.front(); que.pop();
        inq[u] = 0;
        for (int i = 0; i < E[u].size(); i++) {
          int v = E[u][i].v;
          int64_t w = E[u][i].c;
          if (E[u][i].f > 0 && dis[v] > dis[u] + w) {
            prv[v] = u; prvL[v] = i;
            dis[v] = dis[u] + w;
            if (!inq[v]) {
              inq[v] = 1;
              que.push(v);
            }
          }
        }
      }
      if (dis[t] == INF) break;
      int64_t tf = INF;
      for (int v = t, u, l; v != s; v = u) {
        u = prv[v]; l = prvL[v];
        tf = min(tf, E[u][l].f);
      }
      for (int v = t, u, l; v != s; v = u) {
        u = prv[v]; l = prvL[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
      }
      cost += tf * dis[t];
      fl += tf;
    }
    return {fl, cost};
  }
};
```

## 3.3 Bridge CC

```
namespace bridge_cc {
  vector<int> tim, low;
  stack<int, vector<int>> st;
  int t, bcc_id;
  void dfs(int u, int p, const vector<vector<pair<int,
    int>>> &edge, vector<int> &pa) {
    tim[u] = low[u] = t++;
    st.push(u);
    for(const auto &[v, id] : edge[u]) {
      if(id == p)
        continue;
      if(tim[v])
        low[u] = min(low[u], tim[v]);
      else {
        dfs(v, id, edge, pa);
        if(low[v] > tim[u]) {
          int x;
          do {
            pa[x = st.top()] = bcc_id;
            st.pop();
          } while(x != v);
          bcc_id++;
        }
        else
          low[u] = min(low[u], low[v]);
      }
    }
  }
  vector<int> solve(const vector<vector<pair<int, int
    >>> &edge) { // (to, id)
    int n = edge.size();
    tim.resize(n);
    low.resize(n);
    t = bcc_id = 1;
    vector<int> pa(n);

    for(int i = 0; i < n; i++) {
      if(!tim[i]) {
        dfs(i, -1, edge, pa);
        while(!st.empty()) {
          pa[st.top()] = bcc_id;
          st.pop();
        }
        bcc_id++;
      }
    }
    return pa;
  } // return bcc id(start from 1)
};
```

# 4 Geometry

## 4.1 Basic

```
using pt = pair<ll, ll>;
using ptf = pair<ld, ld>;
pt operator+(pt a, pt b)
{ return pt {a.F + b.F, a.S + b.S}; }
pt operator-(pt a, pt b)
{ return pt {a.F - b.F, a.S - b.S}; }
ptf to_ptf(pt p) { return ptf {p.F, p.S}; }
int sign(ll x) { return (x > 0) - (x < 0); }
ll dot(pt a, pt b) { return a.F * b.F + a.S * b.S; }
ll cross(pt a, pt b) { return a.F * b.S - a.S * b.F; }
ld abs2(ptf a) { return dot(a, a); }
ld abs(ptf a) { return sqrtl(dot(a, a)); }
int ori(pt a, pt b, pt c)
{ return sign(cross(b - a, c - a)); }
bool operator(pt a, pt b)
{ return a.F != b.F ? a.F < b.F : a.S < b.S; }
```

## 4.2 2D Convex Hull

```
// returns a convex hull in counterclockwise order
// for a non-strict one, change cross >= to >
vector<pt> convex_hull(vector<pt> p) {
 sort(iter(p));
 if (p[0] == p.back()) return {p[0]};
 int n = p.size(), t = 0;
 vector<pt> h(n + 1);
 for (int _ = 2, s = 0; _--; s = --t, reverse(iter(p)))
  for (pt i : p) {
   while (t > s + 1 && cross(i, h[t-1], h[t-2])>=0)
```

```
    t--;
   h[t++] = i;
  }
 return h.resize(t), h;
} // not tested, but trust ckiseki!
```