Contents

1 Basic

1.1 Run

```
#use -> sh run.sh {name}
g++ -O2 -std=c++14 -Wall -Wextra -Wshadow -o $1 $1.cpp
./$1 < t.in > t.out
```

1.2 Default

```cpp
#include <bits/stdc++.h>
using namespace std;
using LL = long long;
#define IOS ios_base::sync_with_stdio(0); cin.tie(0);
#define pb push_back
#define eb emplace_back
const int INF = 1e9;
const int MOD = 1e9 + 7;
const double EPS = 1e-6;
const int MAXN = 0;

int main() {

}
```

1.3 Black Magic

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace std;
using namespace __gnu_pbds;
using set_t =
  tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>;
using map_t =
  tree<int, int, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>;
using heap_t =
  __gnu_pbds::priority_queue<int>;
using ht_t =
  gp_hash_table<int, int>;
int main() {
  //set-----------------------------------
  set_t st;
  st.insert(5); st.insert(6);
  st.insert(3); st.insert(1);

  // the smallest is (0), biggest is (n-1), kth small is (k-1)
  int num = *st.find_by_order(0);
  cout << num << '\n'; // print 1

  num = *st.find_by_order(st.size() - 1);
  cout << num << '\n'; // print 6

  // find the index
  int index = st.order_of_key(6);
  cout << index << '\n'; // print 3

  // check if there exists x
  int x = 5;
  int check = st.erase(x);
  if (check == 0) printf("st not contain 5\n");
  else if (check == 1) printf("st contain 5\n");

  //tree policy like set
  st.insert(5); st.insert(5);
  cout << st.size() << '\n'; // print 4

  //map-----------------------------------
  map_t mp;
  mp[1] = 2;
  cout << mp[1] << '\n';
  auto tmp = *mp.find_by_order(0); // pair
  cout << tmp.first << " " << tmp.second << '\n';

  //heap----------------------------------
  heap_t h1, h2;
  h1.push(1); h1.push(3);
  h2.push(2); h2.push(4);
  h1.join(h2);
  cout << h1.size() << h2.size() << h1.top() << '\n';
  // 404

  //hash-table----------------------------
  ht_t ht;
  ht[85] = 5;
  ht[89975] = 234;
  for (auto i : ht) {
    cout << i.first << " " << i.second << '\n';
  }
}
```

2 Data Structure

2.1 Disjoint Set

```cpp
// 0-base
const int MAXN = 1000;
int boss[MAXN];
void init(int n) {
  for (int i = 0; i < n; i++) {
    boss[i] = -1;
  }
}
int find(int x) {
  if (boss[x] < 0) {
    return x;
  }
  return boss[x] = find(boss[x]);
}
bool uni(int a, int b) {
  a = find(a);
  b = find(b);
  if (a == b) {
    return false;
  }
  if (boss[a] > boss[b]) {
    swap(a, b);
  }
  boss[a] += boss[b];
  boss[b] = a;
  return true;
}
```

2.2 BIT RARSQ

```
// 1-base
#define lowbit(k) (k & -k)

int n;
vector<int> B1, B2;

void add(vector<int> &tr, int id, int val) {
  for (; id <= n; id += lowbit(id)) {
    tr[id] += val;
  }
}
void range_add(int l, int r, int val) {
  add(B1, l, val);
  add(B1, r + 1, -val);
  add(B2, l, val * (l - 1));
  add(B2, r + 1, -val * r);
}
int sum(vector<int> &tr, int id) {
  int ret = 0;
  for (; id >= 1; id -= lowbit(id)) {
    ret += tr[id];
  }
  return ret;
}
int prefix_sum(int id) {
  return sum(B1, id) * id - sum(B2, id);
}
int range_sum(int l, int r) {
  return prefix_sum(r) - prefix_sum(l - 1);
}
```

2.3 zkw RMQ

```
// 0-base
const int INF = 1e9;
const int MAXN = ;

int n;
int a[MAXN], tr[MAXN << 1];

// !!! remember to call this function
void build() {
  for (int i = 0; i < n; i++) {
    tr[i + n] = a[i];
  }
  for (int i = n - 1; i > 0; i--) {
    tr[i] = max(tr[i << 1], tr[i << 1 | 1]);
  }
}
void update(int id, int val) {
  for (tr[id += n] = val; id > 1; id >>= 1) {
    tr[id >> 1] = max(tr[id], tr[id ^ 1]);
  }
}
int query(int l, int r) { // [l, r)
  int ret = -INF;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l & 1) {
      ret = max(ret, tr[l++]);
    }
    if (r & 1) {
      ret = max(ret, tr[--r]);
    }
  }
  return ret;
}
```

3 Graph

3.1 Dijkstra

```
// 0-base
const LL INF = 1e18;
const int MAXN = ;
struct Edge {
  int to;
  LL cost;
  bool operator < (const Edge &other) const {
    return cost > other.cost;
  }
};

int n;
LL dis[MAXN];
vector<Edge> G[MAXN];

void init() {
  for (int i = 0; i < n; i++) {
    G[i].clear();
    dis[i] = INF;
  }
}
void Dijkstra(int st, int ed = -1) {
  priority_queue<Edge> pq;
  pq.push({st, 0});
  dis[st] = 0;
  while (!pq.empty()) {
    auto now = pq.top();
    pq.pop();
    if (now.to == ed) {
      return;
    }
    if (now.cost > dis[now.to]) {
      continue;
    }
    for (auto &e : G[now.to]) {
      if (dis[e.to] > now.cost + e.cost) {
        dis[e.to] = now.cost + e.cost;
        pq.push({ e.to, dis[e.to] });
      }
    }
  }
}
```

3.2 SPFA(negative cycle)

```
// 0-base
const LL INF = 1e18;
const int MAXN = ;
struct Edge {
  int to;
  LL cost;
};

int n;
LL dis[MAXN];
vector<Edge> G[MAXN];

void init() {
  for (int i = 0; i < n; i++) {
    G[i].clear();
    dis[i] = INF;
  }
}
bool SPFA(int st) {
  vector<int> cnt(n, 0);
  vector<bool> inq(n, false);
  queue<int> q;

  q.push(st);
  dis[st] = 0;
  inq[st] = true;
  while (!q.empty()) {
    int now = q.front();
    q.pop();
    inq[now] = false;
    for (auto &e : G[now]) {
      if (dis[e.to] > dis[now] + e.cost) {
        dis[e.to] = dis[now] + e.cost;
        if (!inq[e.to]) {
          cnt[e.to]++;
          if (cnt[e.to] > n) {
            // negative cycle
            return false;
          }
          inq[e.to] = true;
          q.push(e.to);
        }
      }
    }
  }
  return true;
}
```

3.3 Floyd Warshall

```
// 0-base
// G[i][i] < 0 -> negative cycle
const LL INF = 1e18;
const int MAXN = ;

int n;
LL G[MAXN][MAXN];

void init() {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      G[i][j] = INF;
    }
    G[i][i] = 0;
  }
}
void floyd() {
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        if (G[i][k] != INF && G[k][j] != INF) {
          G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
        }
      }
    }
  }
}
```

3.4 Topological Sort

```
// 0-base
// if ret.size < n -> cycle
int n;
vector<vector<int>> G;

vector<int> topoSort() {
  vector<int> indeg(n), ret;
  for (auto &li : G) {
    for (int x : li) {
      ++indeg[x];
    }
  }
  // use priority queue for lexic. largest ans
  queue<int> q;
  for (int i = 0; i < n; i++) {
    if (!indeg[i]) {
      q.push(i);
    }
  }
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    ret.pb(u);
    for (int v : G[u]) {
      if (--indeg[v] == 0) {
        q.push(v);
      }
    }
  }
  return ret;
}
```

3.5 Kosaraju SCC

```
// 0-base
int n;
vector<vector<int>> G, G2; // G2 = G rev
vector<bool> vis;
vector<int> s, color;
int sccCnt;
void dfs1(int u) {
  vis[u] = true;
  for (int v : G[u]) {
    if (!vis[v]) {
      dfs1(v);
    }
  }
  s.pb(u);
}
void dfs2(int u) {
  color[u] = sccCnt;
  for (int v : G2[u]) {
    if (!color[v]) {
      dfs2(v);
    }
  }
}
void Kosaraju() {
  sccCnt = 0;
  for (int i = 0; i < n; i++) {
    if (!vis[i]) {
      dfs1(i);
    }
  }
  for (int i = n - 1; i >= 0; i--) {
    if (!color[s[i]]) {
      ++sccCnt;
      dfs2(s[i]);
    }
  }
}
```

3.6 Tree Diameter

```
// 0-base;
const int MAXN = ;

struct Edge {
  int to;
  int cost;
};

int n, d = 0;
int d1[MAXN], d2[MAXN];
vector<Edge> G[MAXN];
// dfs(0, -1);
void dfs(int u, int from) {
  d1[u] = d2[u] = 0;
  for (auto e : G[u]) {
    if (e.to == from) {
      continue;
    }
    dfs(e.to, u);
    int t = d1[e.to] + e.cost;
    if (t > d1[u]) {
      d2[u] = d1[u];
      d1[u] = t;
    } else if (t > d2[u]) {
      d2[u] = t;
    }
  }
  d = max(d, d1[u] + d2[u]);
}
```

3.7 Directed MST

```
// 0-base
const LL INF = 1e18;
const int MAXN = ;

struct Edge {
  int from;
  int to;
  LL cost;
};

struct DMST {
  int n;
  int vis[MAXN], pre[MAXN], id[MAXN];
  LL in[MAXN];
  vector<Edge> edges;
  void init(int _n) {
    n = _n;
    edges.clear();
  }
  void add_edge(int from, int to, LL cost) {
    edges.push_back({from, to, cost});
  }
  LL run(int root) {
    LL ret = 0;
    while (true) {
      for (int i = 0; i < n; i++) {
        in[i] = INF;
      }

      // find in edge
      for (auto &e : edges) {
        if (e.cost < in[e.to] && e.from != e.to) {
          pre[e.to] = e.from;
          in[e.to] = e.cost;
        }
      }

      // check in edge
      for (int i = 0; i < n; i++) {
        if (i == root) {
          continue;
        }
        if (in[i] == INF) {
          return -1;
        }
      }

      int nodenum = 0;
      memset(id, -1, sizeof(id));
      memset(vis, -1, sizeof(vis));
      in[root] = 0;

      // find cycles
      for (int i = 0; i < n; i++) {
        ret += in[i];
        int v = i;
        while (vis[v] != i && id[v] == -1 && v != root) {
          vis[v] = i;
          v = pre[v];
        }
        if (id[v] == -1 && v != root) {
          for (int j = pre[v]; j != v; j = pre[j]) {
            id[j] = nodenum;
          }
          id[v] = nodenum++;
        }
      }

      // no cycle
      if (nodenum == 0) {
        break;
      }

      for (int i = 0; i < n; i++) {
        if (id[i] == -1) {
          id[i] = nodenum++;
        }
      }

      // grouping the vertices
      for (auto &e : edges) {
        int to = e.to;
        e.from = id[e.from];
        e.to = id[e.to];
        if (e.from != e.to) {
          e.cost -= in[to]; //!!!
        }
      }

      n = nodenum;
      root = id[root];
    }
    return ret;
  }
};
```

4 Flow & Matching

4.1 KM

```
const int INF = 1e9;
const int MAXN = ;
struct KM { //1-base
  int n, G[MAXN][MAXN];
  int lx[MAXN], ly[MAXN], my[MAXN];
  bool vx[MAXN], vy[MAXN];
  void init(int _n) {
    n = _n;
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
        G[i][j] = 0;
      }
    }
  }
  bool match(int i) {
    vx[i] = true;
    for (int j = 1; j <= n; j++) {
      if (lx[i] + ly[j] == G[i][j] && !vy[j]) {
        vy[j] = true;
        if (!my[j] || match(my[j])) {
          my[j] = i;
          return true;
        }
      }
    }
    return false;
  }
  void update() {
    int delta = INF;
    for (int i = 1; i <= n; i++) {
      if (vx[i]) {
        for (int j = 1; j <= n; j++) {
          if (!vy[j]) {
            delta = min(delta, lx[i] + ly[j] - G[i][j]);
          }
        }
      }
    }
    for (int i = 1; i <= n; i++) {
      if (vx[i]) {
        lx[i] -= delta;
      }
      if (vy[i]) {
        ly[i] += delta;
      }
    }
  }
  int run() {
    for (int i = 1; i <= n; i++) {
      lx[i] = ly[i] = my[i] = 0;
      for (int j = 1; j <= n; j++) {
        lx[i] = max(lx[i], G[i][j]);
      }
    }
    for (int i = 1; i <= n; i++) {
      while (true) {
        for (int i = 1; i <= n; i++) {
          vx[i] = vy[i] = 0;
        }
        if (match(i)) {
          break;
        } else {
          update();
        }
      }
    }
    int ans = 0;
    for (int i = 1; i <= n; i++) {
      ans += lx[i] + ly[i];
    }
    return ans;
  }
};
```

4.2 Dinic

```
#define eb emplace_back
const LL INF = 1e18;
const int MAXN = ;
struct Edge {
  int to;
  LL cap;
  int rev;
  Edge(int v, LL c, int r) : to(v), cap(c), rev(r) {}
};
struct Dinic {
  int n;
  int level[MAXN], now[MAXN];
  vector<Edge> G[MAXN];
  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++) {
      G[i].clear();
    }
  }
  void add_edge(int u, int v, LL c) {
    G[u].eb(v, c, G[v].size());
    // directed graph
    G[v].eb(u, 0, G[u].size() - 1);
    // undirected graph
    // G[v].eb(u, c, G[u].size() - 1);
  }
  bool bfs(int st, int ed) {
    fill(level, level + n + 1, -1);
    queue<int> q;
    q.push(st);
    level[st] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (const auto &e : G[u]) {
        if (e.cap > 0 && level[e.to] == -1) {
          level[e.to] = level[u] + 1;
          q.push(e.to);
        }
      }
    }
    return level[ed] != -1;
  }
  LL dfs(int u, int ed, LL limit) {
    if (u == ed) {
      return limit;
    }
    LL ret = 0;
    for (int &i = now[u]; i < G[u].size(); i++) {
      auto &e = G[u][i];
      if (e.cap > 0 && level[e.to] == level[u] + 1) {
        LL f = dfs(e.to, ed, min(limit, e.cap));
        ret += f;
        limit -= f;
        e.cap -= f;
        G[e.to][e.rev].cap += f;
        if (!limit) {
          return ret;
        }
      }
    }
    if (!ret) {
      level[u] = -1;
    }
    return ret;
  }
  LL flow(int st, int ed) {
    LL ret = 0;
    while (bfs(st, ed)) {
      fill(now, now + n + 1, 0);
      ret += dfs(st, ed, INF);
    }
    return ret;
  }
};
```

5 String

5.1 Manacher

```
int p[2 * MAXN];
int Manacher(const string &s) {
  string st = "@#";
  for (char c : s) {
    st += c;
    st += '#';
  }
  st += '$';
  int id = 0, mx = 0, ans = 0;
  for (int i = 1; i < st.length() - 1; i++) {
    p[i] = (mx > i ? min(p[2 * id - i], mx - i) : 1);
    for (; st[i - p[i]] == st[i + p[i]]; p[i]++);
    if (mx < i + p[i]) {
      mx = i + p[i];
      id = i;
    }
    ans = max(ans, p[i] - 1);
  }
  return ans;
}
```

6 DP

6.1 LIS

```
int LIS(vector<int> &a) {
  vector<int> s;
  for (int i = 0; i < a.size(); i++) {
    if (s.empty() || s.back() < a[i]) {
      s.push_back(a[i]);
    } else {
      *lower_bound(s.begin(), s.end(), a[i],
        [](int x, int y) {return x < y;}) = a[i];
    }
  }
  return s.size();
}
```

6.2 LCS

```
int LCS(string s1, string s2) {
  int n1 = s1.size(), n2 = s2.size();
  vector<vector<int>> dp(n1 + 1, vector<int>(n2 + 1,
      0));
  for (int i = 1; i <= n1; i++) {
    for (int j = 1; j <= n2; j++) {
      if (s1[i - 1] == s2[j - 1]) {
        dp[i][j] = dp[i - 1][j - 1] + 1;
      } else {
        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
      }
    }
  }
  return dp[n1][n2];
}
```

7 Math

7.1 Extended GCD

```
// ax + by = c
int extgcd(int a, int b, int c, int &x, int &y) {
  if (b == 0) {
    x = c / a;
    y = 0;
    return a;
  }
  int d = extgcd(b, a % b, c, x, y);
  int tmp = x;
  x = y;
  y = tmp - (a / b) * y;
  return d;
}
```