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;
```

```
20      }
21      if (boss[a] > boss[b]) {
22        swap(a, b);
23      }
24      boss[a] += boss[b];
25      boss[b] = a;
26      return true;
27  }
```

2.2 BIT RARSQ

```
 1  // 1-base
 2  #define lowbit(k) (k & -k)
 3
 4  int n;
 5  vector<int> B1, B2;
 6
 7  void add(vector<int> &tr, int id, int val) {
 8    for (; id <= n; id += lowbit(id)) {
 9      tr[id] += val;
10    }
11  }
12  void range_add(int l, int r, int val) {
13    add(B1, l, val);
14    add(B1, r + 1, -val);
15    add(B2, l, val * (l - 1));
16    add(B2, r + 1, -val * r);
17  }
18  int sum(vector<int> &tr, int id) {
19    int ret = 0;
20    for (; id >= 1; id -= lowbit(id)) {
21      ret += tr[id];
22    }
23    return ret;
24  }
25  int prefix_sum(int id) {
26    return sum(B1, id) * id - sum(B2, id);
27  }
28  int range_sum(int l, int r) {
29    return prefix_sum(r) - prefix_sum(l - 1);
30  }
```

2.3 zkw RMQ

```
 1  // 0-base
 2  const int INF = 1e9;
 3  const int MAXN = ;
 4
 5  int n;
 6  int a[MAXN], tr[MAXN << 1];
 7
 8  // !!! remember to call this function
 9  void build() {
10    for (int i = 0; i < n; i++) {
11      tr[i + n] = a[i];
12    }
13    for (int i = n - 1; i > 0; i--) {
14      tr[i] = max(tr[i << 1], tr[i << 1 | 1]);
15    }
16  }
17  void update(int id, int val) {
18    for (tr[id += n] = val; id > 1; id >>= 1) {
19      tr[id >> 1] = max(tr[id], tr[id ^ 1]);
20    }
21  }
22  int query(int l, int r) { // [l, r)
23    int ret = -INF;
24    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
25      if (l & 1) {
26        ret = max(ret, tr[l++]);
27      }
28      if (r & 1) {
29        ret = max(ret, tr[--r]);
```

```
30      }
31    }
32    return ret;
33  }
```

3 Graph

3.1 Dijkstra

```
 1  // 0-base
 2  const LL INF = 1e18;
 3  const int MAXN = ;
 4  struct Edge {
 5    int at;
 6    LL cost;
 7    bool operator < (const Edge &other) const {
 8      return cost > other.cost;
 9    }
10  };
11
12  int n;
13  LL dis[MAXN];
14  vector<Edge> G[MAXN];
15
16  void init() {
17    for (int i = 0; i < n; i++) {
18      G[i].clear();
19      dis[i] = INF;
20    }
21  }
22  void Dijkstra(int st, int ed = -1) {
23    priority_queue<Edge> pq;
24    pq.push({ st, 0 });
25    dis[st] = 0;
26    while (!pq.empty()) {
27      auto now = pq.top();
28      pq.pop();
29      if (now.at == ed) {
30        return;
31      }
32      if (now.cost > dis[now.at]) {
33        continue;
34      }
35      for (auto &e : G[now.at]) {
36        if (dis[e.at] > now.cost + e.cost) {
37          dis[e.at] = now.cost + e.cost;
38          pq.push({ e.at, dis[e.at] });
39        }
40      }
41    }
42  }
```

3.2 SPFA(negative cycle)

```
 1  // 0-base
 2  const LL INF = 1e18;
 3  const int MAXN = ;
 4  struct Edge {
 5    int at;
 6    LL cost;
 7  };
 8
 9  int n;
10  LL dis[MAXN];
11  vector<Edge> G[MAXN];
12
13  void init() {
14    for (int i = 0; i < n; i++) {
15      G[i].clear();
16      dis[i] = INF;
17    }
18  }
```

```
19 bool SPFA(int st) {
20   vector<int> cnt(n, 0);
21   vector<bool> inq(n, false);
22   queue<int> q;
23
24   q.push(st);
25   dis[st] = 0;
26   inq[st] = true;
27   while (!q.empty()) {
28     int now = q.front();
29     q.pop();
30     inq[now] = false;
31     for (auto &e : G[now]) {
32       if (dis[e.at] > dis[now] + e.cost) {
33         dis[e.at] = dis[now] + e.cost;
34         if (!inq[e.at]) {
35           cnt[e.at]++;
36           if (cnt[e.at] > n) {
37             // negative cycle
38             return false;
39           }
40           inq[e.at] = true;
41           q.push(e.at);
42         }
43       }
44     }
45   }
46   return true;
47 }
```

3.3 Floyd Warshall

```
1  // 0-base
2  // G[i][i] < 0 -> negative cycle
3  const LL INF = 1e18;
4  const int MAXN = ;
5
6  int n;
7  LL G[MAXN][MAXN];
8
9  void init() {
10   for (int i = 0; i < n; i++) {
11     for (int j = 0; j < n; j++) {
12       G[i][j] = INF;
13     }
14     G[i][i] = 0;
15   }
16 }
17 void floyd() {
18   for (int k = 0; k < n; k++) {
19     for (int i = 0; i < n; i++) {
20       for (int j = 0; j < n; j++) {
21         if (G[i][k] != INF && G[k][j] != INF) {
22           G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
23         }
24       }
25     }
26   }
27 }
```

3.4 Topological Sort

```
1  // 0-base
2  // if ret.size < n -> cycle
3  int n;
4  vector<vector<int>> G;
5
6  vector<int> topoSort() {
7    vector<int> indeg(n), ret;
8    for (auto &li : G) {
9      for (int x : li) {
10       ++indeg[x];
11     }
```

```
12   }
13   // use priority queue for lexic. largest ans
14   queue<int> q;
15   for (int i = 0; i < n; i++) {
16     if (!indeg[i]) {
17       q.push(i);
18     }
19   }
20   while (!q.empty()) {
21     int u = q.front();
22     q.pop();
23     ret.pb(u);
24     for (int v : G[u]) {
25       if (--indeg[v] == 0) {
26         q.push(v);
27       }
28     }
29   }
30   return ret;
31 }
```

4 Flow & Matching

4.1 KM

```
1  const int INF = 1e9;
2  const int MAXN = ;
3  struct KM { //1-base
4    int n, G[MAXN][MAXN];
5    int lx[MAXN], ly[MAXN], my[MAXN];
6    bool vx[MAXN], vy[MAXN];
7    void init(int _n) {
8      n = _n;
9      for (int i = 1; i <= n; i++) {
10       for (int j = 1; j <= n; j++) {
11         G[i][j] = 0;
12       }
13     }
14   }
15   bool match(int i) {
16     vx[i] = true;
17     for (int j = 1; j <= n; j++) {
18       if (lx[i] + ly[j] == G[i][j] && !vy[j]) {
19         vy[j] = true;
20         if (!my[j] || match(my[j])) {
21           my[j] = i;
22           return true;
23         }
24       }
25     }
26     return false;
27   }
28   void update() {
29     int delta = INF;
30     for (int i = 1; i <= n; i++) {
31       if (vx[i]) {
32         for (int j = 1; j <= n; j++) {
33           if (!vy[j]) {
34             delta = min(delta, lx[i] + ly[j] -
                     G[i][j]);
35           }
36         }
37       }
38     }
39     for (int i = 1; i <= n; i++) {
40       if (vx[i]) {
41         lx[i] -= delta;
42       }
43       if (vy[i]) {
44         ly[i] += delta;
45       }
46     }
47   }
48   int run() {
```

```
49      for (int i = 1; i <= n; i++) {
50        lx[i] = ly[i] = my[i] = 0;
51        for (int j = 1; j <= n; j++) {
52          lx[i] = max(lx[i], G[i][j]);
53        }
54      }
55      for (int i = 1; i <= n; i++) {
56        while (true) {
57          for (int i = 1; i <= n; i++) {
58            vx[i] = vy[i] = 0;
59          }
60          if (match(i)) {
61            break;
62          } else {
63            update();
64          }
65        }
66      }
67      int ans = 0;
68      for (int i = 1; i <= n; i++) {
69        ans += lx[i] + ly[i];
70      }
71      return ans;
72    }
73 };
```

5 String

5.1 Manacher

```
 1 int p[2 * MAXN];
 2 int Manacher(const string &s) {
 3   string st = "@#";
 4   for (char c : s) {
 5     st += c;
 6     st += '#';
 7   }
 8   st += '$';
 9   int id = 0, mx = 0, ans = 0;
10   for (int i = 1; i < st.length() - 1; i++) {
11     p[i] = (mx > i ? min(p[2 * id - i], mx - i) : 1);
12     for (; st[i - p[i]] == st[i + p[i]]; p[i]++);
13     if (mx < i + p[i]) {
14       mx = i + p[i];
15       id = i;
16     }
17     ans = max(ans, p[i] - 1);
18   }
19   return ans;
20 }
```

6 DP

6.1 LIS

```
 1 int LIS(vector<int> &a) {
 2   vector<int> s;
 3   for (int i = 0; i < a.size(); i++) {
 4     if (s.empty() || s.back() < a[i]) {
 5       s.push_back(a[i]);
 6     } else {
 7       *lower_bound(s.begin(), s.end(), a[i],
 8         [](int x, int y) {return x < y;}) = a[i];
 9     }
10   }
11   return s.size();
12 }
```

7 Math

7.1 Extended GCD

```
 1 // ax + by = c
 2 int extgcd(int a, int b, int c, int &x, int &y) {
 3   if (b == 0) {
 4     x = c / a;
 5     y = 0;
 6     return a;
 7   }
 8   int d = extgcd(b, a % b, c, x, y);
 9   int tmp = x;
10   x = y;
11   y = tmp - (a / b) * y;
12   return d;
13 }
```