

Contents

1	Basic
1.1	Run
1.2	Default
1.3	Black Magic
2	Data Structure
2.1	Disjoint Set
2.2	BIT RURQ
3	Graph
3.1	Dijkstra
3.2	SPFA(negative cycle)
4	Flow & Matching
4.1	KM
5	String
5.1	Manacher
6	DP
6.1	LIS
7	Math
7.1	Extended GCD

1 Basic

1.1 Run

```
1 #use -> sh run.sh {name}
2 g++ -O2 -std=c++14 -Wall -Wextra -Wshadow -o $1 $1.cpp
3 ./ $1 < t.in > t.out
```

1.2 Default

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using LL = long long;
4 #define IOS ios_base::sync_with_stdio(0); cin.tie(0);
5 #define pb push_back
6 #define eb emplace_back
7 const int INF = 1e9;
8 const int MOD = 1e9 + 7;
9 const double EPS = 1e-6;
10 const int MAXN = 0;
11
12 int main() {
13
14 }
```

1.3 Black Magic

```
1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 #include <ext/pb_ds/tree_policy.hpp>
4 #include <ext/pb_ds/priority_queue.hpp>
5 using namespace std;
6 using namespace __gnu_pbds;
7 using set_t =
8   tree<int, null_type, less<int>, rb_tree_tag,
9     tree_order_statistics_node_update>;
10 using map_t =
11   tree<int, int, less<int>, rb_tree_tag,
12     tree_order_statistics_node_update>;
13 using heap_t =
14   __gnu_pbds::priority_queue<int>;
15 using ht_t =
16   gp_hash_table<int, int>;
17 int main() {
18     //set-----
19     set_t st;
20     st.insert(5); st.insert(6);
```

```
21     st.insert(3); st.insert(1);
22
23     // the smallest is (0), biggest is (n-1), kth small
24     // is (k-1)
25     int num = *st.find_by_order(0);
26     cout << num << '\n'; // print 1
27
28     num = *st.find_by_order(st.size() - 1);
29     cout << num << '\n'; // print 6
30
31     // find the index
32     int index = st.order_of_key(6);
33     cout << index << '\n'; // print 3
34
35     // check if there exist x
36     int x = 5;
37     int check = st.erase(x);
38     if (check == 0) printf("st not contain 5\n");
39     else if (check == 1) printf("st contain 5\n");
40
41     //tree policy like set
42     st.insert(5); st.insert(5);
43     cout << st.size() << '\n'; // print 4
44
45     //map-----
46     map_t mp;
47     mp[1] = 2;
48     cout << mp[1] << '\n';
49     auto tmp = *mp.find_by_order(0); // pair
50     cout << tmp.first << " " << tmp.second << '\n';
51
52     //heap-----
53     heap_t h1, h2;
54     h1.push(1); h1.push(3);
55     h2.push(2); h2.push(4);
56     h1.join(h2);
57     cout << h1.size() << h2.size() << h1.top() << '\n';
58     // 404
59
60     //hash-table-----
61     ht_t ht;
62     ht[85] = 5;
63     ht[89975] = 234;
64     for (auto i : ht) {
65         cout << i.first << " " << i.second << '\n';
66     }
```

2 Data Structure

2.1 Disjoint Set

```
1 const int MAXN = 1000;
2 int boss[MAXN];
3 void init(int n) {
4     for (int i = 0; i < n; i++) {
5         boss[i] = -1;
6     }
7 }
8 int find(int x) {
9     if (boss[x] < 0) {
10         return x;
11     }
12     return boss[x] = find(boss[x]);
13 }
14 bool uni(int a, int b) {
15     a = find(a);
16     b = find(b);
17     if (a == b) {
18         return false;
19     }
20     if (boss[a] > boss[b]) {
21         swap(a, b);
22     }
```

```

23 boss[a] += boss[b];
24 boss[b] = a;
25 return true;
26 }

```

2.2 BIT RURQ

```

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

```

3 Graph

3.1 Dijkstra

```

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

```

```

32         continue;
33     }
34     for (auto &e : G[now.at]) {
35         if (dis[e.at] > now.cost + e.cost) {
36             dis[e.at] = now.cost + e.cost;
37             pq.push({ e.at, dis[e.at] });
38         }
39     }
40 }
41 }

```

3.2 SPFA(negative cycle)

```

1 const long long INF = 1e18;
2 const int MAXN = ;
3 struct Edge {
4     int at;
5     long long cost;
6 };
7
8 int n;
9 long long dis[MAXN];
10 vector<Edge> G[MAXN];
11
12 void init() {
13     for (int i = 0; i < n; i++) {
14         G[i].clear();
15         dis[i] = INF;
16     }
17 }
18 bool SPFA(int st) {
19     vector<int> cnt(n, 0);
20     vector<bool> inq(n, false);
21     queue<int> q;
22
23     q.push(st);
24     dis[st] = 0;
25     inq[st] = true;
26     while (!q.empty()) {
27         int now = q.front();
28         q.pop();
29         inq[now] = false;
30         for (auto &e : G[now]) {
31             if (dis[e.at] > dis[now] + e.cost) {
32                 dis[e.at] = dis[now] + e.cost;
33                 if (!inq[e.at]) {
34                     cnt[e.at]++;
35                     if (cnt[e.at] > n) {
36                         // negative cycle
37                         return false;
38                     }
39                     inq[e.at] = true;
40                     q.push(e.at);
41                 }
42             }
43         }
44     }
45     return true;
46 }

```

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] -
35             G[i][j]);
36         }
37       }
38     }
39   }
40   for (int i = 1; i <= n; i++) {
41     if (vx[i]) {
42       lx[i] -= delta;
43     }
44     if (vy[i]) {
45       ly[i] += delta;
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 }

```