41

#### Contents

```
1 17
1 Basic
1.1 Run .
2 Data Structure
3 Graph
3.1 Dijkstra
4 Flow & Matching
5 String
6
```

# Basic

#### 1.1 Run

```
1 #use -> sh run.sh {name}
 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 =
   tree<int, null_type, less<int>, rb_tree_tag,
      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 =
```

```
__gnu_pbds::priority_queue<int>;
   using ht_t =
15
     gp_hash_table<int, int>;
   int main() {
18
    //set----
19
     set_t st;
     st.insert(5); st.insert(6);
20
     st.insert(3); st.insert(1);
22
     // the smallest is (0), biggest is (n-1), kth small
     int num = *st.find_by_order(0);
24
25
     cout << num << '\n'; // print 1
26
27
     num = *st.find_by_order(st.size() - 1);
     cout << num << '\n'; // print 6</pre>
28
29
     // find the index
     int index = st.order_of_key(6);
31
     cout << index << '\n'; // print 3
32
     // check if there exists x
     int x = 5;
     int check = st.erase(x);
36
     if (check == 0) printf("st not contain 5\n");
37
     else if (check == 1) printf("st contain 5\n");
38
40
     //tree policy like set
     st.insert(5); st.insert(5);
42
     cout << st.size() << '\n'; // print 4</pre>
43
44
     //map-----
45
     map_t mp;
46
     mp[1] = 2;
     cout << mp[1] << '\n';
47
48
     auto tmp = *mp.find_by_order(0); // pair
     cout << tmp.first << " " << tmp.second << '\n';</pre>
49
50
     //heap-------
52
     heap_t h1, h2;
     h1.push(1); h1.push(3);
53
     h2.push(2); h2.push(4);
55
     h1. ioin(h2):
     cout << h1.size() << h2.size() << h1.top() << '\n';</pre>
56
57
     // 404
58
     //hash-table-----
59
     ht_t ht;
60
     ht[85] = 5;
     ht[89975] = 234;
63
     for (auto i : ht) {
       cout << i.first << " " << i.second << '\n';</pre>
64
65
66 }
```

# Data Structure

#### Disjoint Set

```
1 // 0-base
2 const int MAXN = 1000;
3
  int boss[MAXN];
  void init(int n) {
   for (int i = 0; i < n; i++) {
      boss[i] = -1;
    }
7
8
  }
9
  int find(int x) {
  if (boss[x] < 0) {
10
11
      return x;
    }
12
13
    return boss[x] = find(boss[x]);
14 }
15 bool uni(int a, int b) {
```

```
16
     a = find(a);
     b = find(b);
17
     if (a == b) {
18
19
       return false;
20
     if (boss[a] > boss[b]) {
21
22
       swap(a, b);
23
     boss[a] += boss[b];
24
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;
7 void add(vector<int> &tr, int id, int val) {
    for (; id <= n; id += lowbit(id)) {</pre>
      tr[id] += val;
    }
10
11 }
12 void range_add(int 1, int r, int val) {
   add(B1, 1, val);
13
    add(B1, r + 1, -val);
15
    add(B2, 1, val * (1 - 1));
16
    add(B2, r + 1, -val * r);
17 }
18 int sum(vector<int> &tr, int id) {
    int ret = 0;
19
    for (; id >= 1; id -= lowbit(id)) {
20
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 1, int r) {
29 return prefix_sum(r) - prefix_sum(1 - 1);
30 }
```

#### 2.3 zkw RMQ

```
1 // 0-base
2 const int INF = 1e9;
3 const int MAXN = ;
6 int a[MAXN], tr[MAXN << 1];</pre>
8 // !!! remember to call this function
9 void build() {
   for (int i = 0; i < n; i++) {
10
11
      tr[i + n] = a[i];
12
    for (int i = n - 1; i > 0; i--) {
13
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 1, int r) { // [1, r)
23
    int ret = -INF;
    for (1 += n, r += n; 1 < r; 1 >>= 1, r >>= 1) {
24
      if (1 & 1) {
25
```

```
26          ret = max(ret, tr[1++]);
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 = ;
  struct Edge {
    int to;
    LL cost;
 7
    bool operator < (const Edge &other) const {</pre>
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) {
    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();
       if (now.to == ed) {
29
30
         return:
31
       }
32
       if (now.cost > dis[now.to]) {
33
         continue;
34
35
       for (auto &e : G[now.to]) {
36
         if (dis[e.to] > now.cost + e.cost) {
           dis[e.to] = now.cost + e.cost;
37
38
           pq.push({ e.to, dis[e.to] });
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 to;
6   LL cost;
7  };
8  int n;
10  LL dis[MAXN];
11  vector<Edge> G[MAXN];
12
13  void init() {
14  for (int i = 0; i < n; i++) {</pre>
```

```
15
       G[i].clear();
       dis[i] = INF;
16
17
    }
18 }
19 bool SPFA(int st) {
    vector<int> cnt(n, 0);
20
    vector<bool> inq(n, false);
21
     queue<int> q;
22
23
24
     q.push(st);
25
     dis[st] = 0;
     inq[st] = true;
26
27
     while (!q.empty()) {
       int now = q.front();
28
29
       q.pop();
       inq[now] = false;
30
       for (auto &e : G[now]) {
31
32
         if (dis[e.to] > dis[now] + e.cost) {
           dis[e.to] = dis[now] + e.cost;
33
34
           if (!inq[e.to]) {
35
             cnt[e.to]++;
             if (cnt[e.to] > n) {
36
37
               // negative cycle
                return false;
38
39
40
             inq[e.to] = true;
41
             q.push(e.to);
42
43
44
    }
45
46
     return true;
```

# 3.3 Floyd Warshall

47 }

```
1 // 0-base
2 // G[i][i] < 0 -> negative cycle
3 const LL INF = 1e18;
4 const int MAXN = ;
6 int n;
7 LL G[MAXN][MAXN];
9 void init() {
10
    for (int i = 0; i < n; i++) {
       for (int j = 0; j < n; j++) {</pre>
11
12
         G[i][j] = INF;
13
       }
14
       G[i][i] = 0;
15
    }
16 }
17 void floyd() {
    for (int k = 0; k < n; k++) {
18
       for (int i = 0; i < n; i++) {</pre>
19
         for (int j = 0; j < n; j++) {
20
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;
```

```
for (auto &li : G) {
       for (int x : li) {
9
10
         ++indeg[x];
       }
11
     }
12
     // use priority queue for lexic. largest ans
13
     queue<int> q;
14
15
     for (int i = 0; i < n; i++) {
       if (!indeg[i]) {
16
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;
```

#### 3.5 Kosaraju SCC

```
1 // 0-base
2 int n;
  vector<vector<int>>> G, G2; // G2 = G rev
  vector<bool> vis;
  vector<int> s, color;
  int sccCnt;
7
  void dfs1(int u) {
    vis[u] = true;
8
    for (int v : G[u]) {
      if (!vis[v]) {
10
11
         dfs1(v);
12
       }
13
    }
14
    s.pb(u);
15 }
16 void dfs2(int u) {
17
    color[u] = sccCnt;
18
    for (int v : G2[u]) {
19
      if (!color[v]) {
20
        dfs2(v);
21
       }
22
    }
  }
23
24
  void Kosaraju() {
25
    sccCnt = 0;
26
    for (int i = 0; i < n; i++) {
      if (!vis[i]) {
27
28
         dfs1(i);
29
30
    }
31
     for (int i = n - 1; i >= 0; i--) {
      if (!color[s[i]]) {
32
         ++sccCnt;
33
34
         dfs2(s[i]);
35
36
    }
37 }
```

#### 3.6 Tree Diameter

```
1 // 0-base;
2 const int MAXN = ;
3
4 struct Edge {
   int to;
6 int cost;
```

52

in[root] = 0;

```
HJackH
7 };
8
9 int n, d = 0;
10 int d1[MAXN], d2[MAXN];
11 vector < Edge > G[MAXN];
12 // dfs(0, -1);
13 void dfs(int u, int from) {
     d1[u] = d2[u] = 0;
15
     for (auto e : G[u]) {
       if (e.to == from) {
16
17
         continue;
18
19
       dfs(e.to, u);
       int t = d1[e.to] + e.cost;
20
21
       if (t > d1[u]) {
         d2[u] = d1[u];
22
         d1[u] = t;
23
24
       } else if (t > d2[u]) {
25
         d2[u] = t;
26
27
28
     d = max(d, d1[u] + d2[u]);
29 }
  3.7 Directed MST
1 // 0-base
2 const LL INF = 1e18;
```

```
3 const int MAXN = ;
5 struct Edge {
    int from;
6
    int to;
8
    LL cost;
9 };
10
11 struct DMST {
12
    int n;
     int vis[MAXN], pre[MAXN], id[MAXN];
13
14
     LL in [MAXN]:
15
     vector < Edge > edges;
     void init(int _n) {
16
17
       n = _n;
18
       edges.clear();
19
20
     void add_edge(int from, int to, LL cost) {
       edges.push_back({from, to, cost});
21
22
     LL run(int root) {
23
24
       LL ret = 0;
       while (true) {
25
         for (int i = 0; i < n; i++) {</pre>
26
27
           in[i] = INF;
28
29
          // find in edge
30
         for (auto &e : edges) {
31
32
           if (e.cost < in[e.to] && e.from != e.to) {</pre>
33
              pre[e.to] = e.from;
34
              in[e.to] = e.cost;
           }
35
36
37
          // check in edge
38
39
          for (int i = 0; i < n; i++) {</pre>
           if (i == root) {
40
              continue;
           }
42
43
           if (in[i] == INF) {
              return -1;
44
45
         }
46
47
48
         int nodenum = 0;
          memset(id, -1, sizeof(id));
49
         memset(vis, -1, sizeof(vis));
50
```

```
53
          // find cycles
         for (int i = 0; i < n; i++) {</pre>
54
55
            ret += in[i];
56
            int v = i;
            while (vis[v] != i && id[v] == -1 && v !=
57
                root) {
              vis[v] = i;
58
59
              v = pre[v];
60
            if (id[v] == -1 && v != root) {
61
62
              for (int j = pre[v]; j != v; j = pre[j]) {
                id[j] = nodenum;
63
64
65
             id[v] = nodenum++;
66
           }
67
68
69
          // no cycle
         if (nodenum == 0) {
70
71
            break;
72
73
74
         for (int i = 0; i < n; i++) {</pre>
           if (id[i] == -1) {
75
              id[i] = nodenum++;
76
            }
77
78
79
80
          // grouping the vertices
81
         for (auto &e : edges) {
82
           int to = e.to;
83
            e.from = id[e.from];
            e.to = id[e.to];
84
85
            if (e.from != e.to) {
86
              e.cost -= in[to]; //!!!
87
            }
         }
88
89
90
         n = nodenum;
91
         root = id[root];
92
93
       return ret;
94
    }
95 };
```

# Flow & Matching

#### 4.1 KM

```
1 const int INF = 1e9;
  const int MAXN = ;
  struct KM { //1-base
 3
     int n, G[MAXN][MAXN];
     int lx[MAXN], ly[MAXN], my[MAXN];
     bool vx[MAXN], vy[MAXN];
 6
     void init(int _n) {
8
       n = _n;
       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;
       for (int j = 1; j <= n; j++) {</pre>
17
         if (lx[i] + ly[j] == G[i][j] && !vy[j]) {
18
19
           vy[j] = true;
20
           if (!my[j] || match(my[j])) {
21
             my[j] = i;
22
             return true;
```

```
24
                                                                  23
                                                                         G[v].eb(u, 0, G[u].size() - 1);
       }
                                                                         // undirected graph
25
                                                                  24
       return false;
                                                                  25
                                                                          // G[v].eb(u, c, G[u].size() - 1);
26
                                                                       }
27
     }
                                                                  26
28
     void update() {
                                                                  27
                                                                       bool bfs(int st, int ed) {
                                                                         fill(level, level + n + 1, -1);
29
       int delta = INF;
                                                                  28
       for (int i = 1; i <= n; i++) {</pre>
                                                                  29
                                                                         queue<int> q;
30
31
         if (vx[i]) {
                                                                  30
                                                                         q.push(st);
           for (int j = 1; j <= n; j++) {</pre>
                                                                         level[st] = 0;
32
                                                                  31
33
              if (!vy[j]) {
                                                                  32
                                                                          while (!q.empty()) {
34
                delta = min(delta, lx[i] + ly[j] -
                                                                  33
                                                                            int u = q.front();
                     G[i][j]);
                                                                            q.pop();
                                                                  34
35
              }
                                                                  35
                                                                            for (const auto &e : G[u]) {
                                                                              if (e.cap > 0 && level[e.to] == -1) {
           }
                                                                  36
36
37
         }
                                                                  37
                                                                                level[e.to] = level[u] + 1;
                                                                                q.push(e.to);
38
                                                                  38
       for (int i = 1; i <= n; i++) {</pre>
                                                                  39
39
                                                                           }
40
         if (vx[i]) {
                                                                  40
                                                                  41
41
           lx[i] -= delta;
42
                                                                  42
                                                                         return level[ed] != -1;
         if (vy[i]) {
                                                                  43
43
44
           ly[i] += delta;
                                                                  44
                                                                       LL dfs(int u, int ed, LL limit) {
         }
                                                                         if (u == ed) {
45
                                                                  45
       }
                                                                  46
                                                                            return limit;
46
47
                                                                  47
                                                                         LL ret = 0;
48
     int run() {
                                                                  48
       for (int i = 1; i <= n; i++) {</pre>
                                                                         for (int &i = now[u]; i < G[u].size(); i++) {</pre>
49
                                                                  49
                                                                            auto &e = G[u][i];
50
         lx[i] = ly[i] = my[i] = 0;
                                                                  50
         for (int j = 1; j <= n; j++) {</pre>
51
                                                                  51
                                                                            if (e.cap > 0 && level[e.to] == level[u] + 1) {
52
           lx[i] = max(lx[i], G[i][j]);
                                                                  52
                                                                              LL f = dfs(e.to, ed, min(limit, e.cap));
                                                                              ret += f;
53
                                                                  53
54
                                                                              limit -= f;
                                                                              e.cap -= f;
55
       for (int i = 1; i <= n; i++) {
                                                                  55
56
         while (true) {
                                                                  56
                                                                              G[e.to][e.rev].cap += f;
           for (int i = 1; i <= n; i++) {</pre>
57
                                                                  57
                                                                              if (!limit) {
58
              vx[i] = vy[i] = 0;
                                                                  58
                                                                                return ret:
59
           }
                                                                  59
                                                                              }
           if (match(i)) {
                                                                  60
                                                                           }
60
61
              break;
                                                                  61
                                                                          if (!ret) {
62
           } else {
                                                                  62
              update();
                                                                  63
                                                                            level[u] = -1;
63
           }
                                                                  64
64
         }
                                                                  65
65
                                                                         return ret;
                                                                  66
66
                                                                       LL flow(int st, int ed) {
67
       int ans = 0;
                                                                  67
       for (int i = 1; i <= n; i++) {</pre>
                                                                  68
                                                                         LL ret = 0;
68
69
         ans += lx[i] + ly[i];
                                                                  69
                                                                          while (bfs(st, ed)) {
70
                                                                  70
                                                                            fill(now, now + n + 1, 0);
71
       return ans;
                                                                  71
                                                                            ret += dfs(st, ed, INF);
     }
                                                                  72
72
73 };
                                                                  73
                                                                          return ret;
                                                                       }
                                                                  74
                                                                  75 };
```

# 4.2 Dinic

```
1 #define eb emplace_back
2 const LL INF = 1e18;
3 const int MAXN = ;
4 struct Edge {
    int to;
5
    LL cap;
6
7
    int rev:
    Edge(int v, LL c, int r) : to(v), cap(c), rev(r) {}
9 };
10 struct Dinic {
11
    int level[MAXN], now[MAXN];
12
13
     vector < Edge > G[MAXN];
     void init(int _n) {
14
       n = _n;
15
       for (int i = 0; i <= n; i++) {</pre>
16
17
         G[i].clear();
       }
18
19
     }
20
     void add_edge(int u, int v, LL c) {
21
       G[u].eb(v, c, G[v].size());
       // directed graph
22
```

# 5 String

#### 5.1 Manacher

```
1 int p[2 * MAXN];
2 int Manacher(const string &s) {
    string st = "@#";
3
    for (char c : s) {
5
       st += c;
       st += '#';
6
    }
7
8
    st += '$';
9
    int id = 0, mx = 0, ans = 0;
10
     for (int i = 1; i < st.length() - 1; i++) {</pre>
      p[i] = (mx > i ? min(p[2 * id - i], mx - i) : 1);
11
12
       for (; st[i - p[i]] == st[i + p[i]]; p[i]++);
       if (mx < i + p[i]) {</pre>
13
14
         mx = i + p[i];
         id = i;
15
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;
    for (int i = 0; i < a.size(); i++) {</pre>
3
       if (s.empty() || s.back() < a[i]) {</pre>
         s.push_back(a[i]);
       } else {
6
         *lower_bound(s.begin(), s.end(), a[i],
7
           [](int x, int y) {return x < y;}) = a[i];
8
9
    }
10
11
    return s.size();
12 }
```

#### 6.2 LCS

```
1 int LCS(string s1, string s2) {
    int n1 = s1.size(), n2 = s2.size();
3
     vector<vector<int>> dp(n1 + 1, vector<int>(n2 + 1,
         0));
     for (int i = 1; i <= n1; i++) {</pre>
5
       for (int j = 1; j <= n2; j++) {</pre>
         if (s1[i - 1] == s2[j - 1]) {
7
           dp[i][j] = dp[i - 1][j - 1] + 1;
         } else {
8
9
           dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
10
         }
11
      }
12
    }
13
     return dp[n1][n2];
14 }
```

# 7 Math

#### 7.1 Extended GCD

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