```
Contents
                                            17 int main() {
                                               //set---
                                            18
                                            19
                                                set_t st;
                                                st.insert(5); st.insert(6);
                                            20
 1 Basic
                                            21
                                               st.insert(3); st.insert(1);
   1
   1 22
   // the smallest is (0), biggest is (n-1), kth small
 2 Data Structure
                                               int num = *st.find_by_order(0);
                                            24
   cout << num << '\n'; // print 1</pre>
   2 25
                                            26
                                                num = *st.find_by_order(st.size() - 1);
                                            27
 3 Graph
                                            28
                                                cout << num << '\n'; // print 6
   29
   30
                                                // find the index
                                               int index = st.order_of_key(6);
                                            31
                                                cout << index << '\n'; // print 3</pre>
                                            32
 4 Flow & Matching
                                            33
   // check if there exists x
                                            34
   35
                                               int x = 5;
                                                int check = st.erase(x);
                                            36
   if (check == 0) printf("st not contain 5\n");
                                                else if (check == 1) printf("st contain 5\n");
                                            38
                                            39
   //tree policy like set
                                            40
                                           5
   Math
                                                st.insert(5); st.insert(5);
                                            41
   cout << st.size() << '\n'; // print 4</pre>
                                            42
                                            43
                                            44
                                                //map----
     Basic
                                            45
                                               map_t mp;
                                               mp[1] = 2;
                                            46
                                            47
                                                cout << mp[1] << '\n';
 1.1 Run
                                                auto tmp = *mp.find_by_order(0); // pair
                                            48
                                            49
                                                cout << tmp.first << " " << tmp.second << ' \ n';
                                            50
1 #use -> sh run.sh {name}
                                                //heap-----
                                            51
2 g++ -02 -std=c++14 -Wall -Wextra -Wshadow -o $1 $1.cpp
                                            52
                                               heap_t h1, h2;
3 ./$1 < t.in > t.out
                                               h1.push(1); h1.push(3);
                                            53
                                                h2.push(2); h2.push(4);
                                            55
                                               h1.join(h2);
 1.2 Default
                                                cout << h1.size() << h2.size() << h1.top() << '\n';</pre>
                                            57
                                            58
1 #include <bits/stdc++.h>
                                            59
                                                //hash-table------
2 using namespace std;
                                            60
                                               ht_t ht;
3 using LL = long long;
                                               ht[85] = 5;
                                            61
4 #define IOS ios_base::sync_with_stdio(0); cin.tie(0);
                                            62
                                               ht[89975] = 234;
5 #define pb push_back
                                               for (auto i : ht) {
                                            63
6 #define eb emplace_back
                                                 cout << i.first << " " << i.second << '\n';</pre>
                                            64
7 const int INF = 1e9;
                                               }
                                            65
8 const int MOD = 1e9 + 7;
                                            66 }
9 const double EPS = 1e-6;
10 const int MAXN = 0;
```

### 1.3 Black Magic

11

13 14 }

12 int main() {

```
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 =
   tree<int, int, less<int>, rb_tree_tag,
11
      tree_order_statistics_node_update>;
12
13 using heap_t =
   __gnu_pbds::priority_queue<int>;
14
15 using ht_t =
gp_hash_table<int, int>;
```

### 2 Data Structure

## 2.1 Disjoint Set

```
1 // 0-base
  const int MAXN = 1000;
3 int boss[MAXN];
4 void init(int n) {
    for (int i = 0; i < n; i++) {
6
      boss[i] = -1;
7
    }
8 }
9 int find(int x) {
10 if (boss[x] < 0) {
11
      return x:
12
    return boss[x] = find(boss[x]);
13
14 }
15 bool uni(int a, int b) {
16
    a = find(a);
17
    b = find(b);
   if (a == b) {
```

```
19     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;
7 void add(vector<int> &tr, int id, int val) {
8
    for (; id <= n; id += lowbit(id)) {</pre>
9
       tr[id] += val;
    }
10
11 }
12 void range_add(int 1, int r, int val) {
13 add(B1, 1, val);
    add(B1, r + 1, -val);
add(B2, l, val * (l - 1));
14
15
16
    add(B2, r + 1, -val * r);
17 }
18 int sum(vector<int> &tr, int id) {
19
    int ret = 0;
    for (; id >= 1; id -= lowbit(id)) {
20
      ret += tr[id];
21
22
   }
23
    return ret;
24 }
25 int prefix_sum(int id) {
    return sum(B1, id) * id - sum(B2, id);
26
27 }
28 int range_sum(int 1, 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 = ;
5 int n;
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
      tr[i + n] = a[i];
11
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) {
    for (tr[id += n] = val; id > 1; id >>= 1) {
18
19
       tr[id >> 1] = max(tr[id], tr[id ^ 1]);
    }
20
21 }
22 int query(int 1, int r) { // [1, r)
    int ret = -INF;
23
     for (1 += n, r += n; 1 < r; 1 >>= 1, r >>= 1) {
25
      if (1 & 1) {
26
         ret = max(ret, tr[1++]);
27
      if (r & 1) {
28
```

```
29         ret = max(ret, tr[--r]);
30         }
31         }
32         return ret;
33     }
```

# 3 Graph

# 3.1 Dijkstra

```
1 // 0-base
  const LL INF = 1e18;
  const int MAXN = ;
 4 struct Edge {
   int at;
    LL cost;
6
    bool operator < (const Edge &other) const {</pre>
      return cost > other.cost;
8
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++) {
      G[i].clear();
18
       dis[i] = INF;
19
20
    }
21 }
22
  void Dijkstra(int st, int ed = -1) {
    priority_queue < Edge > pq;
23
     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
       if (now.cost > dis[now.at]) {
32
33
         continue:
34
       }
35
       for (auto &e : G[now.at]) {
         if (dis[e.at] > now.cost + e.cost) {
36
           dis[e.at] = now.cost + e.cost;
37
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 {
    int at;
    LL cost;
7 };
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) {
     vector<int> cnt(n, 0);
    vector<bool> inq(n, false);
21
22
     queue < int > q;
23
    q.push(st);
24
25
     dis[st] = 0;
     inq[st] = true;
26
     while (!q.empty()) {
27
28
       int now = q.front();
       q.pop();
29
30
       inq[now] = false;
31
       for (auto &e : G[now]) {
32
         if (dis[e.at] > dis[now] + e.cost) {
           dis[e.at] = dis[now] + e.cost;
33
           if (!inq[e.at]) {
34
35
              cnt[e.at]++;
              if (cnt[e.at] > n) {
36
                // negative cycle
37
                return false;
38
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 = ;
6 int n;
7 LL G[MAXN][MAXN];
9 void init() {
    for (int i = 0; i < n; i++) {</pre>
10
       for (int j = 0; j < n; j++) {
11
12
         G[i][j] = INF;
13
14
       G[i][i] = 0;
15
    }
16 }
17
  void floyd() {
18
     for (int k = 0; k < n; k++) {
       for (int i = 0; i < n; i++) {</pre>
19
20
         for (int j = 0; j < n; j++) {
           if (G[i][k] != INF && G[k][j] != INF) {
21
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
     // use priority queue for lexic. largest ans
13
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);
       for (int v : G[u]) {
24
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;
  const int MAXN = ;
  struct KM { //1-base
    int n, G[MAXN][MAXN];
     int lx[MAXN], ly[MAXN], my[MAXN];
6
     bool vx[MAXN], vy[MAXN];
     void init(int _n) {
7
8
       n = _n;
       for (int i = 1; i <= n; i++) {</pre>
9
         for (int j = 1; j \le n; j++) {
10
           G[i][j] = 0;
11
12
13
       }
14
    }
15
     bool match(int i) {
16
       vx[i] = true;
17
       for (int j = 1; j \le n; j++) {
18
         if (lx[i] + ly[j] == G[i][j] && !vy[j]) {
           vy[j] = true;
19
20
           if (!my[j] || match(my[j])) {
21
             my[j] = i;
22
              return true;
           }
23
         }
24
25
       }
26
       return false;
27
28
     void update() {
       int delta = INF;
29
       for (int i = 1; i <= n; i++) {
30
         if (vx[i]) {
31
32
           for (int j = 1; j <= n; j++) {</pre>
33
             if (!vy[j]) {
34
                delta = min(delta, lx[i] + ly[j] -
                    G[i][j]);
             }
35
36
           }
         }
37
38
       for (int i = 1; i <= n; i++) {
39
40
         if (vx[i]) {
41
           lx[i] -= delta;
42
43
         if (vy[i]) {
44
           ly[i] += delta;
45
46
       }
47
    }
```

```
48
     int run() {
                                                                   48
                                                                          LL ret = 0;
       for (int i = 1; i <= n; i++) {</pre>
                                                                          for (int &i = now[u]; i < G[u].size(); i++) {</pre>
                                                                   49
49
50
          lx[i] = ly[i] = my[i] = 0;
                                                                   50
                                                                            auto &e = G[u][i];
          for (int j = 1; j <= n; j++) {</pre>
                                                                   51
                                                                            if (e.cap > 0 && level[e.to] == level[u] + 1) {
51
52
            lx[i] = max(lx[i], G[i][j]);
                                                                   52
                                                                               LL f = dfs(e.to, ed, min(limit, e.cap));
53
                                                                   53
                                                                               ret += f;
                                                                               limit -= f;
54
                                                                   54
55
       for (int i = 1; i <= n; i++) {
                                                                   55
                                                                               e.cap -= f;
                                                                               G[e.to][e.rev].cap += f;
          while (true) {
                                                                   56
56
            for (int i = 1; i <= n; i++) {</pre>
                                                                               if (!limit) {
57
                                                                   57
58
              vx[i] = vy[i] = 0;
                                                                   58
                                                                                 return ret;
                                                                               }
                                                                   59
59
            if (match(i)) {
                                                                   60
                                                                            }
60
                                                                   61
              break:
61
62
            } else {
                                                                   62
                                                                          if (!ret) {
63
              update();
                                                                   63
                                                                            level[u] = -1;
64
                                                                   64
65
         }
                                                                   65
                                                                          return ret;
       }
                                                                   66
66
67
       int ans = 0;
                                                                   67
                                                                        LL flow(int st, int ed) {
       for (int i = 1; i <= n; i++) {</pre>
                                                                          LL ret = 0;
68
                                                                   68
69
         ans += lx[i] + ly[i];
                                                                   69
                                                                          while (bfs(st, ed)) {
70
                                                                   70
                                                                            fill(now, now + n + 1, 0);
71
                                                                   71
                                                                            ret += dfs(st, ed, INF);
       return ans;
72
                                                                   72
73 };
                                                                   73
                                                                          return ret;
                                                                   74
                                                                        }
```

### 4.2 Dinic

47

```
1 #define eb emplace_back
2 const LL INF = 1e18;
3 const int MAXN = ;
 4 struct Edge {
5
    int to;
    LL cap;
6
    int rev;
7
    Edge(int v, LL c, int r) : to(v), cap(c), rev(r) {}
8
10 struct Dinic {
11
    int n:
12
     int level[MAXN], now[MAXN];
     vector<Edge> G[MAXN];
13
     void init(int _n) {
14
15
       n = _n;
       for (int i = 0; i <= n; i++) {</pre>
16
17
         G[i].clear();
18
19
     void add_edge(int u, int v, LL c) {
20
21
       G[u].eb(v, c, G[v].size());
22
       // directed graph
       G[v].eb(u, 0, G[u].size() - 1);
23
       // undirected graph
24
       // G[v].eb(u, c, G[u].size() - 1);
25
26
     bool bfs(int st, int ed) {
27
       fill(level, level + n + 1, -1);
28
       queue<int> q;
29
       q.push(st);
30
31
       level[st] = 0;
32
       while (!q.empty()) {
33
         int u = q.front();
34
         q.pop();
         for (const auto &e : G[u]) {
35
36
           if (e.cap > 0 && level[e.to] == -1) {
             level[e.to] = level[u] + 1;
37
38
             q.push(e.to);
           }
39
40
         }
41
       }
       return level[ed] != -1;
42
43
44
     LL dfs(int u, int ed, LL limit) {
45
       if (u == ed) {
         return limit;
46
```

# 5 String

75 };

### 5.1 Manacher

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