1.3

Black Magic

Contents

11

13

14 }

12 **int** main() {

```
1 #include <bits/stdc++.h>
                                        #include <ext/pb_ds/assoc_container.hpp>
 1 Basic
                                      1
                                        #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,
 2 Data Structure
  2.1 Disjoint Set
                                       9
                                           tree_order_statistics_node_update>;
   2 10
                                        using map_t =
   tree<int, int, less<int>, rb_tree_tag,
                                           tree_order_statistics_node_update>;
 3 Graph
                                     2 13
                                        using heap t =
   __gnu_pbds::priority_queue<int>;
   3 15
                                        using ht_t =
   3 16
                                          gp_hash_table<int, int>;
   17
                                        int main() {
  3 18
                                          //set----
   4 19
                                          set t st:
   st.insert(5); st.insert(6);
                                          st.insert(3); st.insert(1);
 4 Flow & Matching
   // the smallest is (0), biggest is (n-1), kth small
   is (k-1)
   5 24
                                          int num = *st.find_by_order(0);
                                          cout << num << '\n'; // print 1
   6 25
                                       26
 5 String
                                          num = *st.find_by_order(st.size() - 1);
                                       27
   28
                                          cout << num << '\n'; // print 6
                                       29
                                     7
                                       30
                                          // find the index
   int index = st.order_of_key(6);
                                       31
                                          cout << index << '\n'; // print 3</pre>
                                       32
                                       33
 7 Math
                                          // check if there exists x
   35
                                          int x = 5;
                                       36
                                          int check = st.erase(x);
                                      7
                                          if (check == 0) printf("st not contain 5\n");
                                       37
   else if (check == 1) printf("st contain 5\n");
                                       38
   39
                                       40
                                          //tree policy like set
  st.insert(5); st.insert(5);
                                       41
                                          cout << st.size() << '\n'; // print 4
                                       42
                                          //map-----
                                       44
    Basic
                                       45
                                          map_t mp;
                                       46
                                          mp[1] = 2;
                                       47
                                          cout << mp[1] << '\n';
     Run
                                       48
                                          auto tmp = *mp.find_by_order(0); // pair
                                          cout << tmp.first << " " << tmp.second << ' \setminus n';
                                       49
                                       50
1 #use -> sh run.sh {name}
                                          //heap------
2 g++ -02 -std=c++14 -Wall -Wextra -Wshadow -o $1 $1.cpp
                                          heap_t h1, h2;
                                       52
3 | ./$1 < t.in > t.out
                                       53
                                          h1.push(1); h1.push(3);
                                          h2.push(2); h2.push(4);
                                       54
                                       55
                                          h1.join(h2);
                                          cout << h1.size() << h2.size() << h1.top() << '\n';</pre>
                                       56
 1.2 Default
                                       57
                                       58
                                          //hash-table-----
                                       59
                                          ht_t ht;
1 #include <bits/stdc++.h>
                                       60
                                          ht[85] = 5:
 using namespace std;
                                       61
3 using LL = long long;
                                       62
                                          ht[89975] = 234;
4 #define IOS ios_base::sync_with_stdio(0); cin.tie(0);
                                          for (auto i : ht) {
                                           cout << i.first << " " << i.second << '\n';</pre>
5 #define pb push_back
                                       64
                                       65
6 #define eb emplace_back
                                       66 }
 const int INF = 1e9;
8 \mid const \mid int \mid MOD = 1e9 + 7;
9 const double EPS = 1e-6;
10 const int MAXN = 0;
                                        1.4 Binary Search
```

```
1 lower_bound(a, a + n, k);
                            //最左邊 ≥ k 的位置
2 upper_bound(a, a + n, k);
                            //最左邊 > k 的位置
3 | upper_bound(a, a + n, k) - 1; //最右邊 ≤ k 的位置
```

```
4 | lower_bound(a, a + n, k) - 1; //最右邊 < k 的位置</td>5 | [lower_bound, upper_bound)//等於 k 的範圍6 | equal_range(a, a + n, k);
```

2 Data Structure

2.1 Disjoint Set

```
1 // 0-base
2 const int MAXN = 1000;
3 int boss[MAXN];
4 void init(int n) {
    for (int i = 0; i < n; i++) {</pre>
       boss[i] = -1;
    }
7
8 }
9 int find(int x) {
10
   if (boss[x] < 0) {
11
      return x;
12
13
    return boss[x] = find(boss[x]);
14 }
15 bool uni(int a, int b) {
16
    a = find(a);
17
    b = find(b);
18
    if (a == b) {
       return false;
19
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) {
    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);
14
    add(B2, 1, val * (1 - 1));
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
21
      ret += tr[id];
    }
22
23
    return ret;
24 }
25 int prefix_sum(int id) {
    return sum(B1, id) * id - sum(B2, id);
27 }
28 int range_sum(int 1, int r) {
    return prefix_sum(r) - prefix_sum(l - 1);
29
```

2.3 zkw RMQ

```
1 // 0-base
  const int INF = 1e9;
  const int MAXN = ;
  int n;
 6 int a[MAXN], tr[MAXN << 1];</pre>
  // !!! 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) {
      tr[id >> 1] = max(tr[id], tr[id ^ 1]);
19
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) {
24
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 = ;
3
  struct Edge {
    int to:
    LL cost;
     Edge(int v, LL c) : to(v), cost(c) {}
 7
     bool operator < (const Edge &other) const {</pre>
9
      return cost > other.cost;
10
11 };
12
13 int n;
14 LL dis[MAXN];
15 vector < Edge > G[MAXN];
16
17 void init() {
18
    for (int i = 0; i < n; i++) {
19
      G[i].clear();
20
       dis[i] = INF;
21
    }
22 }
23
  void Dijkstra(int st, int ed = -1) {
    priority_queue < Edge > pq;
24
     pq.emplace(st, 0);
26
     dis[st] = 0;
27
     while (!pq.empty()) {
       auto now = pq.top();
28
       pq.pop();
29
       if (now.to == ed) {
30
31
         return;
32
33
       if (now.cost > dis[now.to]) {
34
         continue:
```

3.2 SPFA(negative cycle)

```
1 // 0-base
2 const LL INF = 1e18;
3 const int MAXN = ;
4 struct Edge {
    int to;
    LL cost:
     Edge(int v, LL c) : to(v), cost(c) {}
8 };
9
10 int n;
11 LL dis[MAXN];
12 vector < Edge > G[MAXN];
13
14 void init() {
     for (int i = 0; i < n; i++) {</pre>
15
16
       G[i].clear();
17
       dis[i] = INF;
18
     }
19 }
20 bool SPFA(int st) {
     vector < int > cnt(n, 0);
21
22
     vector<bool> inq(n, false);
     queue<int> q;
23
24
     q.push(st);
25
     dis[st] = 0;
26
27
     inq[st] = true;
28
     while (!q.empty()) {
29
       int now = q.front();
30
       q.pop();
       inq[now] = false;
31
32
       for (auto &e : G[now]) {
33
         if (dis[e.to] > dis[now] + e.cost) {
34
            dis[e.to] = dis[now] + e.cost;
35
           if (!inq[e.to]) {
36
              cnt[e.to]++;
              if (cnt[e.to] > n) {
37
                // negative cycle
38
39
                return false;
40
41
              inq[e.to] = true;
42
              q.push(e.to);
43
44
45
       }
46
     }
47
     return true;
```

3.3 Floyd Warshall

```
1  // 0-base
2  // G[i][i] < 0 -> negative cycle
3  const LL INF = 1e18;
4  const int MAXN = ;
5  int n;
7  LL G[MAXN][MAXN];
8  void init() {
10  for (int i = 0; i < n; i++) {</pre>
```

```
11
        for (int j = 0; j < n; j++) {
          G[i][j] = INF;
12
13
        G[i][i] = 0;
14
15
16 }
17
   void floyd() {
18
     for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
19
          for (int j = 0; j < n; j++) {
  if (G[i][k] != INF && G[k][j] != INF) {</pre>
20
21
               G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
22
23
24
          }
25
     }
26
27 }
```

3.4 Topological Sort

```
1 // 0-base
  // if ret.size < n -> cycle
  int n;
  vector<vector<int>> G;
6
  vector<int> topoSort() {
     vector<int> indeg(n), ret;
7
     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;
     for (int i = 0; i < n; i++) {</pre>
15
16
       if (!indeg[i]) {
17
         q.push(i);
18
19
     }
20
     while (!q.empty()) {
21
      int u = q.front();
       q.pop();
22
23
       ret.pb(u);
       for (int v : G[u]) {
24
25
         if (--indeg[v] == 0) {
           q.push(v);
26
27
28
29
     }
30
     return ret;
31 }
```

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;
6 int sccCnt;
7
  void dfs1(int u) {
    vis[u] = true;
    for (int v : G[u]) {
10
      if (!vis[v]) {
         dfs1(v);
11
12
      }
    }
13
    s.pb(u);
14
15 }
16 void dfs2(int u) {
17
    color[u] = sccCnt;
18
    for (int v : G2[u]) {
      if (!color[v]) {
```

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36 37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72 73

74 75

76

77

78

79 80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

return ret;

```
20
          dfs2(v);
       }
21
22
     }
23 }
24 void Kosaraju() {
     sccCnt = 0;
25
     for (int i = 0; i < n; i++) {
26
27
       if (!vis[i]) {
         dfs1(i);
28
29
30
     }
     for (int i = n - 1; i \ge 0; i - -) {
31
32
       if (!color[s[i]]) {
33
          ++sccCnt:
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;
    Edge(int v, int c) : to(v), cost(c) {}
8 };
9
10 | int n, d = 0;
11 int d1[MAXN], d2[MAXN];
12 vector < Edge > G[MAXN];
13 // dfs(0, -1);
14 void dfs(int u, int from) {
     d1[u] = d2[u] = 0;
15
     for (auto e : G[u]) {
16
       if (e.to == from) {
17
18
         continue;
19
20
       dfs(e.to, u);
       int t = d1[e.to] + e.cost;
21
       if (t > d1[u]) {
22
23
         d2[u] = d1[u];
         d1[u] = t;
24
25
       } else if (t > d2[u]) {
         d2[u] = t;
26
27
    }
28
29
     d = max(d, d1[u] + d2[u]);
30 }
```

3.7 Directed MST

```
1 // 0-base
2 const LL INF = 1e18;
  const int MAXN = ;
3
5 struct Edge {
6
    int from;
7
     int to;
8
     LL cost;
     Edge(int u, int v, LL c) : from(u), to(v), cost(c)
9
         {}
10 };
11
12 struct DMST {
13
    int n;
     int vis[MAXN], pre[MAXN], id[MAXN];
14
    LL in[MAXN];
15
16
     vector < Edge > edges;
17
     void init(int _n) {
18
      n = _n;
```

```
edges.clear();
void add_edge(int from, int to, LL cost) {
  edges.eb(from, to, cost);
LL run(int root) {
 LL ret = 0;
  while (true) {
    for (int i = 0; i < n; i++) {</pre>
      in[i] = INF;
    // find in edge
    for (auto &e : edges) {
      if (e.cost < in[e.to] && e.from != e.to) {</pre>
        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];
 }
```

4

```
4 Flow & Matching
```

95 }

96 };

4.1 Bipartite Matching

```
1 // 0-base
2 const int MAXN = ;
3 int n;
4 vector<int> G[MAXN];
5 int vy[MAXN], my[MAXN];
7 bool match(int u) {
8
    for (int v : G[u]) {
       if (vy[v]) {
9
10
         continue;
11
12
       vy[v] = true;
13
       if (my[v] == -1 || match(my[v])) {
14
         my[v] = u;
15
         return true;
16
17
18
    return false;
19 }
20 int sol() {
    int cnt = 0;
21
    memset(my, -1, sizeof(my));
23
    for (int i = 0; i < n; i++) {
       memset(vy, 0, sizeof(vy));
24
25
       if (match(i)) {
26
         cnt++;
27
       }
    }
28
29
    return cnt;
30 }
```

4.2 KM

```
1 const int INF = 1e9;
2 const int MAXN = ;
3 struct KM { //1-base
    int n, G[MAXN][MAXN];
     int lx[MAXN], ly[MAXN], my[MAXN];
     bool vx[MAXN], vy[MAXN];
7
     void init(int _n) {
       n = _n;
8
9
       for (int i = 1; i <= n; i++) {
         for (int j = 1; j <= n; j++) {</pre>
10
11
           G[i][j] = 0;
12
         }
       }
13
14
     }
     bool match(int i) {
15
       vx[i] = true;
16
       for (int j = 1; j <= n; j++) {</pre>
17
18
         if (lx[i] + ly[j] == G[i][j] && !vy[j]) {
19
           vy[j] = true;
           if (!my[j] || match(my[j])) {
20
21
             my[j] = i;
             return true;
22
23
         }
24
25
       }
26
       return false;
27
     void update() {
28
29
       int delta = INF;
       for (int i = 1; i <= n; i++) {</pre>
30
         if (vx[i]) {
31
           for (int j = 1; j \le n; j++) {
32
```

```
33
              if (!vy[j]) {
                delta = min(delta, lx[i] + ly[j] -
34
                     G[i][j]);
35
              }
36
           }
         }
37
38
39
       for (int i = 1; i <= n; i++) {
         if (vx[i]) {
40
41
           lx[i] -= delta;
42
         if (vy[i]) {
43
44
            ly[i] += delta;
45
46
       }
     }
47
48
     int run() {
49
       for (int i = 1; i <= n; i++) {
         lx[i] = ly[i] = my[i] = 0;
50
51
         for (int j = 1; j <= n; j++) {
            lx[i] = max(lx[i], G[i][j]);
52
53
       }
54
55
       for (int i = 1; i <= n; i++) {</pre>
56
         while (true) {
            for (int i = 1; i <= n; i++) {</pre>
57
              vx[i] = vy[i] = 0;
58
            }
59
60
            if (match(i)) {
61
              break:
            } else {
62
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 };
```

4.3 Dinic

```
1 #define eb emplace_back
2 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) {}
9|};
10
  struct Dinic {
11
     int n;
     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());
22
       // directed graph
       G[v].eb(u, 0, G[u].size() - 1);
23
24
       // undirected graph
25
       // G[v].eb(u, c, G[u].size() - 1);
26
27
     bool bfs(int st, int ed) {
28
       fill(level, level + n + 1, -1);
       queue<int> q;
29
30
       q.push(st);
       level[st] = 0;
```

```
32
       while (!q.empty()) {
                                                                29
         int u = q.front();
                                                                     bool SPFA(int st, int ed) {
33
                                                                30
         q.pop();
                                                                31
                                                                        for (int i = 0; i < n; i++) {</pre>
34
35
         for (const auto &e : G[u]) {
                                                                32
                                                                          pre[i] = -1;
36
           if (e.cap > 0 && level[e.to] == -1) {
                                                                33
                                                                          dis[i] = INF;
37
              level[e.to] = level[u] + 1;
                                                                34
                                                                          cnt[i] = 0;
              q.push(e.to);
                                                                35
                                                                          inq[i] = false;
38
39
           }
                                                                36
         }
40
                                                                37
                                                                        queue<int> q;
41
                                                                38
                                                                        bool negcycle = false;
42
       return level[ed] != -1;
                                                                39
                                                                        dis[st] = 0;
43
                                                                40
44
     LL dfs(int u, int ed, LL limit) {
                                                                41
                                                                        cnt[st] = 1;
45
       if (u == ed) {
                                                                        inq[st] = true;
                                                                42
46
         return limit;
                                                                43
                                                                        q.push(st);
47
                                                                44
       LL ret = 0;
                                                                 45
                                                                        while (!q.empty() && !negcycle) {
48
49
       for (int &i = now[u]; i < G[u].size(); i++) {</pre>
                                                                 46
                                                                          int u = q.front();
         auto &e = G[u][i];
                                                                47
50
                                                                          q.pop();
51
         if (e.cap > 0 && level[e.to] == level[u] + 1) {
                                                                48
                                                                          inq[u] = false;
           LL f = dfs(e.to, ed, min(limit, e.cap));
                                                                          for (int i : G[u]) {
52
                                                                49
           ret += f;
                                                                 50
                                                                            int v = edges[i].v;
53
54
           limit -= f;
                                                                51
                                                                            LL cost = edges[i].cost;
55
           e.cap -= f;
                                                                52
                                                                            LL cap = edges[i].cap;
           G[e.to][e.rev].cap += f;
                                                                53
56
57
           if (!limit) {
                                                                54
                                                                            if (dis[v] > dis[u] + cost && cap > 0) {
                                                                              dis[v] = dis[u] + cost;
58
              return ret;
59
                                                                56
                                                                              pre[v] = i;
                                                                57
                                                                              if (!inq[v]) {
60
61
                                                                58
                                                                                q.push(v);
       if (!ret) {
                                                                59
                                                                                cnt[v]++:
62
63
         level[u] = -1;
                                                                60
                                                                                inq[v] = true;
64
                                                                61
65
                                                                62
                                                                                if (cnt[v] == n + 2) {
       return ret;
66
                                                                63
                                                                                   negcycle = true;
67
     LL flow(int st, int ed) {
                                                                                   break:
                                                                64
       LL ret = 0;
                                                                65
                                                                                }
68
       while (bfs(st, ed)) {
                                                                66
                                                                              }
69
70
         fill(now, now + n + 1, 0);
                                                                67
71
         ret += dfs(st, ed, INF);
                                                                68
                                                                          }
72
                                                                69
73
                                                                70
       return ret;
     }
                                                                71
74
                                                                        return dis[ed] != INF;
75 };
                                                                72
                                                                73
                                                                     LL sendFlow(int v, LL curFlow) {
                                                                        if (pre[v] == -1) {
                                                                74
                                                                75
                                                                          return curFlow;
  4.4 MCMF
                                                                76
                                                                77
                                                                        int i = pre[v];
1 // 0-base
                                                                78
                                                                        int u = edges[i].u;
2 const LL INF = 1e18;
                                                                79
                                                                        LL cost = edges[i].cost;
  const int MAXN = ;
                                                                80
4 struct Edge {
                                                                81
                                                                        LL f = sendFlow(u, min(curFlow, edges[i].cap));
    int u, v;
                                                                82
     LL cost;
                                                                        ans_cost += f * cost;
                                                                83
7
     LL cap;
                                                                        edges[i].cap -= f;
8
     Edge(int _u, int _v, LL _c, LL _cap) : u(_u),
                                                                85
                                                                        edges[i ^ 1].cap += f;
         v(_v), cost(_c), cap(_cap) {}
                                                                86
                                                                        return f;
9 };
                                                                87
10 struct MCMF {
                      // inq times
                                                                     pair<LL, LL> run(int st, int ed) {
                                                                88
     int n, pre[MAXN], cnt[MAXN];
11
                                                                89
                                                                        ans_flow = ans_cost = 0;
     LL ans_flow, ans_cost, dis[MAXN];
12
                                                                        while (SPFA(st, ed)) {
                                                                90
13
     bool inq[MAXN];
                                                                91
                                                                          ans_flow += sendFlow(ed, INF);
     vector<int> G[MAXN];
14
                                                                92
15
     vector < Edge > edges;
                                                                93
                                                                        return make_pair(ans_flow, ans_cost);
     void init(int _n) {
16
                                                                94
                                                                     }
       n = _n;
17
                                                                95 };
18
       edges.clear();
19
       for (int i = 0; i < n; i++) {</pre>
         G[i].clear();
20
21
       }
```

5 String

22

23 24

25

26

27

28

void add_edge(int u, int v, LL c, LL cap) {

G[u].pb(edges.size());

G[v].pb(edges.size());

edges.eb(v, u, -c, \emptyset);

edges.eb(u, v, c, cap);

5.1 Manacher

```
1 int p[2 * MAXN];
2 int Manacher(const string &s) {
```

```
string st = "@#";
    for (char c : s) {
       st += c;
       st += '#';
6
7
     st += '$';
8
     int id = 0, mx = 0, ans = 0;
     for (int i = 1; i < st.length() - 1; i++) {</pre>
       p[i] = (mx > i ? min(p[2 * id - i], mx - i) : 1);
11
       for (; st[i - p[i]] == st[i + p[i]]; p[i]++);
12
       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

```
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();
}</pre>
```

6.2 LCS

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

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, y, x);
9   y -= (a / b) * x;
10  return d;
11 }
```

8 Geometry

8.1 Point

```
1 // notice point type!!!
2 using dvt = int;
  const double EPS = 1e-6;
  const double PI = acos(-1);
  struct Pt {
7
    dvt x:
    dvt y;
9 };
10 bool operator < (const Pt &a, const Pt &b) {
11
   return a.x == b.x ? a.y < b.y : a.x < b.x;</pre>
12 }
13 bool operator == (const Pt &a, const Pt &b) {
14
  return a.x == b.x && a.y == b.y;
15 }
16 Pt operator + (const Pt &a, const Pt &b) {
17
   return {a.x + b.x, a.y + b.y};
18 }
19 Pt operator - (const Pt &a, const Pt &b) {
20
   return {a.x - b.x, a.y - b.y};
21 }
22 // multiply constant
23 Pt operator * (const Pt &a, const dvt c) {
24
   return {a.x * c, a.y * c};
25 }
26 Pt operator / (const Pt &a, const dvt c) {
   return {a.x / c, a.y / c};
27
28 }
29
  // |a| x |b| x cos(x)
  dvt iproduct(const Pt &a, const Pt &b) {
30
   return a.x * b.x + a.y * b.y;
31
32 }
33 // |a| \times |b| \times \sin(x)
34 dvt cross(const Pt &a, const Pt &b) {
35
    return a.x * b.y - a.y * b.x;
36 }
37 dvt dis_pp(const Pt &a, const Pt, &b) {
38
  dvt dx = a.x - b.x;
    dvt dy = a.y - b.y;
39
40
    return sqrt(dx * dx, dy * dy);
```

8.2 Line

```
1 struct Line {
    Pt st;
    Pt ed;
4 };
  // return point side
  // left, on line, right -> 1, 0, -1
6
  int side(Line 1, Pt a) {
    dvt cross_val = cross(a - 1.st, 1.ed - 1.st);
    if (cross_val > EPS) {
9
10
      return 1;
    } else if (cross_val < -EPS) {</pre>
11
12
      return -1;
13
    } else {
14
       return 0;
15
16 }
  // AB infinity, CD segment
18 bool has_intersection(Line AB, Line CD) {
19
    int c = side(AB, CD.st);
20
    int d = side(AB, CD.ed);
    if (c == 0 || d == 0) {
21
      return true;
22
23
    } else {
       // different side
24
25
       return c == -d;
26
```

```
27 }
                                                                   12
28 // find intersection point, two line, not seg
                                                                        int k = m;
                                                                   13
29 pair<int, Pt> intersection(Line a, Line b) {
                                                                   14
                                                                        for (int i = sz - 2; i >= 0; i--) {
     Pt A = a.ed - a.st;
                                                                          while (m > k \&\&
30
                                                                   15
31
     Pt B = b.ed - b.st;
                                                                   16
                                                                            cross(ret[m - 1] - ret[m - 2], a[i] - ret[m -
32
     Pt C = b.st - a.st;
                                                                                2]) <= EPS) {
     dvt mom = cross(A, B);
                                                                   17
33
     dvt son = cross(C, B);
                                                                   18
                                                                          }
     if (std::abs(mom) <= EPS) {</pre>
                                                                   19
                                                                          ret[m++] = a[i];
35
       if (std::abs(son) <= EPS) {</pre>
36
                                                                   20
37
         return {1, {}}; // same line
                                                                   21
                                                                        if (sz > 1) {
       } else {
                                                                   22
38
                                                                          m - -;
39
         return {2, {}}; // parallel
                                                                   23
                                                                        }
       }
                                                                   24
                                                                        ret.resize(m);
40
41
     } else {
                                                                   25
                                                                        return ret;
       return {0, a.st + A * (son / mom)};
                                                                   26 }
42
43
44 }
45 // line to point distance
46 dvt dis_lp(Line 1, Pt a) {
   return area3x2(1.st, l.ed, a) / dis_pp(l.st, l.ed);
47
48 }
                       d(P,L) = \frac{|ax_0 + by_0 + c|}{2\sqrt{1 - c}}
                                  \sqrt[2]{a^2 + b^2}
```

8.3 Area

```
1 // triangle
2 dvt area3(Pt a, Pt b, Pt c) {
   return std::abs(cross(b - a, c - a) / 2);
3
4 }
5 dvt area3x2(Pt a, Pt b, Pt c) { // for integer
    return std::abs(cross(b - a, c - a));
6
7 }
8 // simple convex area(can in)
9 dvt area(vector<Pt> &a) {
    dvt ret = 0;
10
     for (int i = 0, sz = a.size(); i < sz; i++) {</pre>
11
12
       ret += cross(a[i], a[(i + 1) % sz]);
13
14
    return std::abs(ret) / 2;
15 }
16 // check point in/out a convex
int io_convex(vector<Pt> convex, Pt q) {
18
     // convex is Counterclockwise
19
     for (int i = 0, sz = convex.size(); i < sz; i++) {</pre>
       Pt cur = convex[i] - q;
20
       Pt nex = convex[(i + 1) % sz] - q;
21
       dvt cross_val = cross(cur, nex);
22
23
       if (std::abs(cross_val) <= EPS) {</pre>
24
         return 0; // on edge
25
26
       if (cross_val < 0) {</pre>
27
         return -1; // outside
28
    }
29
                    // inside
30
    return 1;
```

8.4 Convex Hull

```
1 vector<Pt> convex_hull(vector<Pt> &a) {
2
    sort(a.begin(), a.end());
    a.erase(unique(a.begin(), a.end()), a.end());
    int sz = a.size(), m = 0;
    vector<Pt> ret(sz + 5); // safe 1 up
6
    for (int i = 0; i < sz; i++) {
      while (m > 1 &&
7
        cross(ret[m - 1] - ret[m - 2], a[i] - ret[m -
            2]) <= EPS) {
      }
10
11
      ret[m++] = a[i];
```