

Contents

1	Basic
1.1	BIT
1.2	Black Magic
1.3	DJS
1.4	ST
1.5	DFS
1.6	BFS
2	DP
2.1	LCS
2.2	LIS
2.3	迴文
3	Graph
3.1	Bellman Ford
3.2	Dijk
3.3	Edges
3.4	Floyd
3.5	KM
4	Math
4.1	GCDhjackh
4.2	Prime
5	String
5.1	KMP

1 Basic

1.1 BIT

```

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 * (l - 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 }

```

1.2 Black Magic

```

1 #include <bits/extc++.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>; //紅黑樹(set)
10 using map_t =
11     tree<int, int, less<int>, rb_tree_tag,

```

```

12     tree_order_statistics_node_update>; //紅黑樹(map)
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); //在裡面第幾大
33     cout << index << '\n'; // print 3
34
35     // check if there exists 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     // 支援合併
59     // 404
60
61     //hash-table-----
62     ht_t ht;
63     ht[85] = 5;
64     ht[89975] = 234;
65     for (auto i : ht) {
66         cout << i.first << " " << i.second << '\n';
67     }
68     //比較強的unordered map

```

1.3 DJS

```

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 | }

```

1.4 ST

```

1 | #include <./basic/Template.h>
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 | }

```

1.5 DFS

```

1 | struct Edge {
2 |     int bi, color; // a 連接到的 bi, 通道顏色
3 |     bool operator < (const Edge &other) const {
4 |         return color < other.color;
5 |     }
6 | };
7 | vector<Edge> G[maxn];
8 |
9 | void DFS(int me, int mydad, int distance) {
10 |     if (dist[me] < distance) return;
11 |     dist[me] = distance;
12 |     for (int i = 0; i < G[me].size(); i++) {
13 |         int v = G[me][i].bi;
14 |         DFS(v, me, distance + 1);
15 |     }
16 | }

```

1.6 BFS

```

1 | bool visit[maxn]; // 訪問過的
2 | void BFS(int point) {
3 |     queue<int> q;
4 |     q.push(point);
5 |     while (!q.empty()) {
6 |         int u = q.front();
7 |         if (visit[u]) continue; // 訪問過就下一個
8 |         visit[u] = true;
9 |         for (int i =
10 |             0; i < edge[u][i]; i++) { // 連出去的線丟到 queue
11 |             q.push(edge[u][i]);
12 |         }
13 |     }

```

2 DP

2.1 LCS

```

1 | int LCS(string s1, string s2) {
2 |     int n1 = s1.size(), n2 = s2.size();
3 |     vector<vector<int>> dp(n1 + 1, vector<int>(n2 + 1,
4 |         0));
5 |     for (int i = 1; i <= n1; i++) {
6 |         for (int j = 1; j <= n2; j++) {
7 |             if (s1[i - 1] == s2[j - 1]) {
8 |                 dp[i][j] = dp[i - 1][j - 1] + 1;
9 |             } else {
10 |                 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
11 |             }
12 |         }
13 |     }
14 |     return dp[n1][n2];

```

2.2 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 | }

```

2.3 迴文

```

1 | bool isPalindrome[100][100];
2 | // Find the palindromes of a string in O(n^2)
3 |
4 | int main()
5 | {
6 |     ios_base::sync_with_stdio(0);
7 |     // freopen("in.txt", "r", stdin);
8 |     string s;
9 |     cin >> s;
10 |     int len = s.size();
11 |     for (int i = 0; i < len; i++)
12 |         isPalindrome[i][i] = true;
13 |
14 |     for (int k = 1; k < len; k++) {
15 |         for (int i = 0; i + k < len; i++) {
16 |             int j = i + k;
17 |             isPalindrome[i][j] = (s[i] == s[j]) &&
18 |                 (isPalindrome[i + 1][j - 1] || i + 1 == j - 1);

```

```

19     }
20 }
21 return 0;
22 }

```

3 Graph

3.1 Bellman Ford

```

1 #include <./basic/Template.h>
2 bool bellman(int src)
3 {
4     // Nodes are indexed from 1
5     for (int i = 1; i <= n; i++)
6         dist[i] = INF;
7     dist[src] = 0;
8     for(int i = 2; i <= n; i++)
9     {
10         for (int j = 0; j < edges.size(); j++)
11         {
12             int u = edges[j].first;
13             int v = edges[j].second;
14             ll weight = adj[u][v];
15             if (dist[u]!=INF && dist[u] + weight <
16                 dist[v])
17                 dist[v] = dist[u] + weight;
18         }
19     }
20     for (int i = 0; i < edges.size(); i++)
21     {
22         int u = edges[i].first;
23         int v = edges[i].second;
24         ll weight = adj[u][v];
25         // True if neg-cylce exists
26         if (dist[u]!=INF && dist[u] + weight < dist[v])
27             return true;
28     }
29     return false;
30 }

```

3.2 Dijk

```

1 #include <./basic/Template.h>
2 const long long int INF = 1e18;
3 const int MAXN = 1000000;
4 struct Edge {
5     int to;
6     long long int cost;
7     Edge(int v, long long int c) : to(v), cost(c) {}
8     bool operator < (const Edge &other) const {
9         return cost > other.cost;
10    }
11 };
12
13 int n;
14 long long int 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) {
24     priority_queue<Edge> pq;
25     pq.emplace(st, 0);
26     dis[st] = 0;
27     while (!pq.empty()) {
28         auto now = pq.top();
29         pq.pop();
30         if (now.to == ed) {

```

```

31             return;
32         }
33         if (now.cost > dis[now.to]) {
34             continue;
35         }
36         for (auto &e : G[now.to]) {
37             if (dis[e.to] > now.cost + e.cost) {
38                 dis[e.to] = now.cost + e.cost;
39                 pq.emplace(e.to, dis[e.to]);
40             }
41         }
42     }
43 }

```

3.3 Edges

```

1 struct Edge
2 {
3     int from, to, w;
4     bool operator<(const Edge& rhs) // optional
5     {
6         return w < rhs.w;
7     }
8 };

```

3.4 Floyd

```

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

```

3.5 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 |         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 | };

```

4 Math

4.1 GCDhjackh

```

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

```

```

12 | }

```

4.2 Prime

```

1 | const int maxn = ;
2 | int arr[maxn];
3 | int prime[maxn];
4 | void init(){
5 |     for (int i = 0; i < maxn; ++i){
6 |         arr[i] = 0;
7 |     }
8 | }
9 | void find(){
10 |    int num = 0;
11 |    for(int i = 2;i<maxn;i++){
12 |        if(arr[i] == 0){
13 |            prime[num] = 0;
14 |            num++;
15 |            for(int j = i*i;j<maxn;j+=i){
16 |                arr[j] = 1;
17 |            }
18 |        }
19 |    }
20 | }

```

5 String

5.1 KMP

```

1 | void failure(string s, int len, int *f)
2 | {
3 |     f[ 0 ] = -1;
4 |     for(int i = 1; i < len; i++)
5 |     {
6 |         int k = f[ i-1 ];
7 |
8 |         while(s[i] != s[k+1] && k >= 0)
9 |             k = f[k];
10 |
11 |         if(s[i] == s[k+1])f[i] = k+1;
12 |         else f[i] = -1;
13 |     }
14 | }
15 |
16 | int compare(string big, string little, int *f)
17 | {
18 |     int Blen = big.length(), Llen = little.length();
19 |     int i = 0, j = 0;
20 |
21 |     while(i < Blen && j < Llen)
22 |     {
23 |         if(big[i] == little[j])
24 |         {
25 |             i++;
26 |             j++;
27 |         }
28 |         else if(j == 0)i++;
29 |         else j = f[j-1] + 1;
30 |     }
31 |
32 |     if(j == Llen)return 1;
33 |     else return 0;
34 | }

```

