

## Contents

1	Data Structure	
1.1	BIT	
1.2	Segment tree	
1.3	Trie	
2	Dynamic Programming	
2.1	Josephus	
2.2	LCS	
2.3	LIS	
3	Graph	
3.1	Dijkstra	
3.2	Floyd Warshall	
3.3	SPFA	
3.4	Kruskal	
3.5	KM	
3.6	Dicnic	
4	Math	
4.1	Inversion	
5	Function	
5.1	strstr	
5.2	substr	
5.3	map set	
5.4	vector	
5.5	setprecision	
5.6	GCD LCM	
5.7	reverse	
5.8	CHAR	
5.9	sort	
5.10	struct	
5.11	deque	

## 1 Data Structure

### 1.1 BIT

```

1 #define lowbit(k) (k & -k)
2 void add(vector<int> &tr, int id, int val) {
3     for (; id <= n; id += lowbit(id)) {
4         tr[id] += val;
5     }
6 }
7 int sum(vector<int> &tr, int id) {
8     int ret = 0;
9     for (; id >= 1; id -= lowbit(id)) {
10        ret += tr[id];
11    }
12    return ret;
13 }

```

### 1.2 Segment tree

```

1 int dfs(int lef, int rig){
2     if(lef + 2 == rig){
3         if(num[lef] > num[rig-1]){
4             return lef;
5         }
6         else{
7             return rig-1;
8         }
9     }
10    int mid = (lef + rig)/2;
11    int p1 = dfs(lef, mid);
12    int p2 = dfs(mid, rig);
13    if(num[p1] > num[p2]){
14        return p1;
15    }
16    else{
17        return p2;
18    }
19 }

```

### 1.3 Trie

```

1 const int MAXL = ; // 自己填
2 const int MAXC = ;
3 struct Trie {
4     int nex[MAXL][MAXC];
5     int len[MAXL];
6     int sz;
7     void init() {
8         memset(nex, 0, sizeof(nex));
9         memset(len, 0, sizeof(len));
10        sz = 0;
11    }
12    void insert(const string &str) {
13        int p = 0;
14        for (char c : str) {
15            int id = c - 'a';
16            if (!nex[p][id]) {
17                nex[p][id] = ++sz;
18            }
19            p = nex[p][id];
20        }
21        len[p] = str.length();
22    }
23    vector<int> find(const string &str, int i) {
24        int p = 0;
25        vector<int> ans;
26        for (; i < str.length(); i++) {
27            int id = str[i] - 'a';
28            if (!nex[p][id]) {
29                return ans;
30            }
31            p = nex[p][id];
32            if (len[p]) {
33                ans.pb(len[p]);
34            }
35        }
36        return ans;
37    }
38 };

```

## 2 Dynamic Programming

### 2.1 Josephus

```

1 int josephus (int n, int k) {
2     // 有 n 個人圍成一圈，每 k 個一次
3     return n > 1 ? (josephus(n-1, k) + k) % n : 0;
4 }
5 // 回傳最後一人的編號，0 index

```

### 2.2 LCS

```

1 int LCS(string s1, string s2) {
2     int n1 = s1.size(), n2 = s2.size();
3     int dp[n1 + 1][n2 + 1];
4     memset(dp, 0, sizeof(dp));
5     // dp[i][j] = s1的前 i 個字元和 s2 的前 j 個字元
6     for (int i = 1; i <= n1; i++) {
7         for (int j = 1; j <= n2; j++) {
8             if (s1[i - 1] == s2[j - 1]) {
9                 dp[i][j] = dp[i - 1][j - 1] + 1;
10            }
11            else {
12                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
13            }
14        }
15    }
16    return dp[n1][n2];
17 }

```

## 2.3 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         }
7         else {
8             *lower_bound(s.begin(), s.end(), a[i]) =
              a[i];
9         }
10    }
11    return s.size();
12 }

```

```

5     }
6     G[i][i] = 0;
7 }
8 for (int k = 0; k < n; k++){ //
    嘗試每一個中繼點
9     for (int i = 0; i < n; i++){ //
        計算每一個i點與每一個j點
10        for (int j = 0; j < n; j++){
11            G[i][j] = min(G[i][j], G[i][k] +
              G[k][j]);
12        }
13    }
14 }
15 }

```

## 3 Graph

### 3.1 Dijkstra

```

1 int M; // number of vertex
2 int N; // number of edge
3 int s, t, w;
4
5 struct Edge{
6
7     int t, w;
8
9     bool operator < (const Edge &rhs) const {
10         return w > rhs.w;
11     }
12 };
13
14 int dis[100001];
15 vector<Edge> G[100001];
16
17 void Dijkstra(int s){
18
19     for(int i = 0; i <= M; i++){
20         dis[i] = 1e9;
21     }
22
23     dis[s] = 0;
24     priority_queue<Edge> pq;
25
26     pq.push({s, 0});
27
28     while(!pq.empty()){
29
30         Edge now = pq.top();
31         pq.pop();
32
33         if(now.w > dis[now.t]){
34             continue;
35         }
36
37         // relax
38         for(Edge e: G[now.t]){
39             if(dis[e.t] > now.w + e.w){
40                 dis[e.t] = now.w + e.w;
41                 pq.push({e.t, dis[e.t]});
42             }
43         }
44     }
45 }

```

### 3.2 Floyd Warshall

```

1 void floyd_warshall(){
2     for(int i = 0; i < n; i++){
3         for(int j = 0; j < n; j++){
4             G[i][j] = INF;

```

### 3.3 SPFA

```

1 bool SPFA(int s){
2     // 記得初始化這些陣列
3     int cnt[1000+5], dis[1000+5];
4     bool inqueue[1000+5];
5     queue<int> q;
6
7     q.push(s);
8     dis[s] = 0;
9     inqueue[s] = true;
10    cnt[s] = 1;
11    while(!q.empty()){
12        int now = q.front();
13        q.pop();
14        inqueue[now] = false;
15
16        for(auto &e : G[now]){
17            if(dis[e.t] > dis[now] + e.w){
18                dis[e.t] = dis[now] + e.w;
19                if(!inqueue[e.t]){
20                    cnt[e.t]++;
21                    if(cnt[e.t] > m){
22                        return false;
23                    }
24                    inqueue[e.t] = true;
25                    q.push(e.t);
26                }
27            }
28        }
29    }
30    return true;
31 }

```

### 3.4 Kruskal

```

1 struct Edge{
2     int u, v, w;
3     // 用權重排序 由大到小
4     bool operator < (const Edge &other) const{
5         return w > other.w;
6     }
7 }edge[maxn];
8 // disjoint set
9 int find(int x){
10    if(parent[x] < 0){
11        return x;
12    }
13    else{
14        return parent[x] = find(parent[x]);
15    }
16 }
17 void unite(int a, int b){
18     a = find(a);
19     b = find(b);
20
21     if(a != b){
22         if(parent[a] < parent[b]){

```

```

23     parent[a] += parent[b];
24     parent[b] = a;
25 }
26 else{
27     parent[b] += parent[a];
28     parent[a] = b;
29 }
30 }
31 }
32 void kruskal(){
33     memset(parent, -1, sizeof(parent));
34     sort(edge, edge + m);
35     int i, j;
36     for(i = 0, j = 0; i < n - 1 && j < m; i++){
37         // 如果 u 和 v 的祖先相同，則 j++
38         // (祖先相同代表會產生環 所以不要)
39         while(find(edge[j].u) == find(edge[j].v)) j++;
40         // 若都會產生環 則讓兩點之間產生橋
41         // (連接兩顆子生成樹)
42         unite(edge[j].u, edge[j].v);
43         j++;
44     }
45 }

```

### 3.5 KM

```

1  const int X = 50; // x的點數，等於y的點數
2  const int Y = 50; // y的點數
3  int adj[X][Y]; // 精簡過的adjacency matrix
4  int lx[X], ly[Y]; // vertex labeling
5  int mx[X], my[Y]; //
6  // x各點的配對對象、y各點的配對對象
7  int q[X], *qf, *qb; // BFS queue
8  int p[X]; // BFS
9  // parent，交錯樹之偶點，指向上一個偶點
10 bool vx[X], vy[Y]; // 記錄是否在交錯樹上
11 int dy[Y], pdy[Y]; // 表格
12 void relax(int x){ // relaxation
13     for (int y=0; y<Y; ++y)
14         if (adj[x][y] != 1e9)
15             if (lx[x] + ly[y] - adj[x][y] < dy[y]){
16                 dy[y] = lx[x] + ly[y] - adj[x][y];
17                 pdy[y] = x; //
18                 // 記錄好是從哪個樹葉連出去的
19             }
20 }
21 void reweight(){ // 調整權重、調整表格
22     int d = 1e9;
23     for (int y=0; y<Y; ++y) if (!vy[y]) d = min(d, dy[y]);
24     for (int x=0; x<X; ++x) if (vx[x]) lx[x] -= d;
25     for (int y=0; y<Y; ++y) if (vy[y]) ly[y] += d;
26     for (int y=0; y<Y; ++y) if (!vy[y]) dy[y] -= d;
27 }
28 void augment(int x, int y){ // 擴充路徑
29     for (int ty; x != -1; x = p[x], y = ty){
30         ty = mx[x]; my[y] = x; mx[x] = y;
31     }
32 }
33 bool branch1(){ // 延展交錯樹：使用既有的等邊
34     while (qf < qb)
35         for (int x=*qf++, y=0; y<Y; ++y)
36             if (!vy[y] && lx[x] + ly[y] == adj[x][y]){
37                 vy[y] = true;
38                 if (my[y] == -1){
39                     augment(x, y);
40                     return true;
41                 }
42                 int z = my[y];
43                 *qb++ = z; p[z] = x; vx[z] = true;
44                 relax(z);
45             }
46     return false;

```

```

44 }
45 bool branch2(){ // 延展交錯樹：使用新添的等邊
46     for (int y=0; y<Y; ++y){
47         if (!vy[y] && dy[y] == 0){
48             vy[y] = true;
49             if (my[y] == -1){
50                 augment(pdy[y], y);
51                 return true;
52             }
53             int z = my[y];
54             *qb++ = z; p[z] = pdy[y]; vx[z] = true;
55             relax(z);
56         }
57     }
58     return false;
59 }
60 int Hungarian(){
61     // 初始化vertex labeling
62     // memset(lx, 0, sizeof(lx)); // 任意值皆可
63     memset(ly, 0, sizeof(ly));
64     for (int x=0; x<X; ++x)
65         for (int y=0; y<Y; ++y)
66             lx[x] = max(lx[x], adj[x][y]);
67     // x側每一個點，分別建立等邊交錯樹。
68     memset(mx, -1, sizeof(mx));
69     memset(my, -1, sizeof(my));
70     for (int x=0; x<X; ++x){
71         memset(vx, false, sizeof(vx));
72         memset(vy, false, sizeof(vy));
73         memset(dy, 0x7f, sizeof(dy));
74         qf = qb = q;
75         *qb++ = x; p[x] = -1; vx[x] = true; relax(x);
76         while (true){
77             if (branch1()) break;
78             reweight();
79             if (branch2()) break;
80         }
81     }
82     // 計算最大權完美匹配的權重
83     int weight = 0;
84     for (int x=0; x<X; ++x)
85         weight += adj[x][mx[x]];
86     return weight;
87 }

```

### 3.6 Dicnic

```

1 // Maximum Flow
2 const int V = 100, E = 1000;
3 int adj[V]; // adjacency lists，初始化為-1。
4 struct Element {int b, r, next;} e[E*2];
5 int en = 0;
6 void addedge(int a, int b, int c){
7     e[en] = (Element){b, c, adj[a]}; adj[a] = en++;
8     e[en] = (Element){a, 0, adj[b]}; adj[b] = en++;
9 }
10 int d[V]; // 最短距離
11 bool visit[V]; // BFS/DFS visit record
12 int q[V]; // queue
13 int BFS(int s, int t){ // 計算最短路徑，求出容許圖
14     memset(d, 0x7f, sizeof(d));
15     memset(visit, false, sizeof(visit));
16     int qn = 0;
17     d[s] = 0;
18     visit[s] = true;
19     q[qn++] = s;
20
21     for (int qf=0; qf<qn; ++qf){
22         int a = q[qf];
23         for (int i = adj[a]; i != -1; i = e[i].next){
24             int b = e[i].b;
25             if (e[i].r > 0 && !visit[b]){
26                 d[b] = d[a] + 1;
27                 visit[b] = true;

```

```

28         q[qn++] = b;
29         if (b == t) return d[t];
30     }
31 }
32 }
33 return V;
34 }
35 int DFS(int a, int df, int s, int t){ //
    求出一條最短擴充路徑，並擴充流量
36 if (a == t) return df;
37 if (visit[a]) return 0;
38 visit[a] = true;
39 for (int i = adj[a]; i != -1; i = e[i].next){
40     int b = e[i].b;
41     if (e[i].r > 0 && d[a] + 1 == d[b]){
42         int f = DFS(b, min(df, e[i].r), s, t);
43         if (f){
44             e[i].r -= f;
45             e[i^1].r += f;
46             return f;
47         }
48     }
49 }
50 return 0;
51 }
52 int dinitz(int s, int t){
53     int flow = 0;
54     while (BFS(s, t) < V)
55         while (true){
56             memset(visit, false, sizeof(visit));
57             int f = DFS(s, 1e9, s, t);
58             if (!f) break;
59             flow += f;
60         }
61     return flow;
62 }

```

## 4 Math

### 4.1 Inversion

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int n; // number of integer
5 int a[1000001];
6 int buf[1000001];
7
8 long long Inversion(int left, int right){
9
10     if (right - left <= 1){ // finish split
11         return 0;
12     }
13
14     int middle = (right + left) / 2;
15     long long ans = Inversion(left, middle) +
16                     Inversion(middle, right);
17     int i = left, j = middle, k = left;
18     while (i < middle || j < right){
19         if (i >= middle){
20             buf[k] = a[j++];
21         }
22         else if (j >= right){
23             buf[k] = a[i++];
24         }
25         else{
26             if (a[i] <= a[j]){
27                 buf[k] = a[i++];
28             }
29             else{
30                 buf[k] = a[j++];
31                 ans += middle - i;
32             }
33         }
34     }
35 }

```

```

33     }
34     k++;
35 }
36
37 for (int k = left; k < right; k++){
38     a[k] = buf[k];
39 }
40
41 return ans;
42 }
43
44 int main(){
45
46     while(cin >> n && n){
47
48         ans = 0;
49         memset(a, 0, sizeof(a));
50         memset(buf, 0, sizeof(buf));
51
52         for(int i = 0; i < n; i++){
53             cin >> a[i];
54         }
55
56         cout << Inversion(0, n) << endl;
57     }
58
59     return 0;
60 }

```

## 5 Function

### 5.1 strstr

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main(){
5     char * c;
6     char str1[1005], str2[1005];
7     scanf("%s %s", str1, str2);
8     c = strstr(str1, str2);
9     if (c != NULL){
10         printf("Yes\n");
11     }
12     else printf("No\n");
13 }
14 // Input : Hello eLl
15 // Output : No

```

### 5.2 substr

```

1 int main(){
2     string str; //abcdef
3     cin >> str;
4     string tmp;
5     tmp = str.substr(0, 2); //ab
6     str = str.substr(2); //cdef
7     cout << tmp << " " << str;
8     return 0;
9 }

```

### 5.3 map set

```

1 .begin( ) // Return iterator to beginning
2 .end( ) // Return iterator to end
3 .empty( ) // 檢查是否為空
4 .size( ) // 回傳大小
5 mp.insert(pair<char, int>('a', 100))
6 st.insert(100) // 插入key、value
7 .erase( ) // 刪掉指定key和他的value

```

```

8 | .clear( ) // 清空整個 map
9 | m.find( )
10 | cout << "a => " << mymap.find('a')->second << endl;
11 | // 找出 map 裡 key
    // 有沒有在裡面，如果有的話會回傳元素所在的 iterator，
12 | s.count() // 返回某個值元素在 set 的個數
13 | while( !mymap.empty()){
14 |     cout << mymap.begin()->first << " => " <<
        mymap.begin()->second << endl;
15 |     mymap.erase(mymap.begin());
16 | }
17 | for (auto it = mymap.begin(); it != mymap.end(); ++it)
18 |     cout << it->first << " => " << it->second << endl;

```

## 5.4 vector

```

1 | v.erase(v.begin() + 5) // 拿掉第六個數
2 | v.erase (v.begin(), v.begin() + 3); // 拿掉前三個數

```

## 5.5 setprecision

```

1 | // 將數字的小數部分設定為固定長度
2 | cnt = 3.5555;
3 | cout << fixed << setprecision(3) << cnt ;
4 | // output : 3.555

```

## 5.6 GCD LCM

```

1 | int gcd(int a, int b){
2 |     return (b == 0 ? a : gcd(b, a % b));
3 | }
4 | int lcm(int a, int b){
5 |     return a * b / gcd(a, b);
6 | }
7 |
8 | /* 輾轉相除法 - 求兩數是否互質
9 | 如果兩數互質 最終結果其中一方為0時 另一方必為1
10 | 若兩數有公因數 最終結果其中一方為0時 另一方必不為1 */
11 | while ( ( num1 % num2 ) != 0 && ( num2 % num1 ) !=
    0 );

```

## 5.7 reverse

```

1 | int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
2 | reverse(a, a+5) // 轉換0~5
3 |
4 | vector<int> v;
5 | reverse(v.begin(), v.end());
6 |
7 | string str = "123";
8 | reverse(str.begin(), str.end());
9 | cout << str << endl; //321

```

## 5.8 CHAR

```

1 | isdigit()
2 | isalnum() //判斷字母 // 數字
3 | isalpha()
4 | islower()
5 | isupper()
6 | isblank() //判斷是否為空格，或者 tab 健制表符，即
    space 和 \t
7 | toupper()
8 | tolower()

```

## 5.9 sort

```

1 | priority_queue<int, vector<int>, less<int>> //大到小
2 | priority_queue<int, vector<int>, greater<int>>
    //小到大
3 |
4 | int arr[] = {4, 5, 8, 3, 7, 1, 2, 6, 10, 9};
5 | sort(arr, arr+10);
6 |
7 | vector<int> v;
8 | sort(v.begin(), v.end()); //小到大
9 |
10 | int cmp(int a, int b){
11 |     return a > b;
12 | }
13 | sort(v.begin(), v.end(), cmp); //大到小

```

## 5.10 struct

```

1 | struct area{
2 |     int a, b;
3 |     bool operator<(const area rhs) const{
4 |         return a > rhs.a || ( a == a && b > rhs.b);
5 |     }
6 |     bool operator!=(const area rhs) const{
7 |         return a != rhs.a || b != rhs.b;
8 |     }
9 | };

```

## 5.11 deque

```

1 | deque<int> que;
2 | que.push_back(10);
3 | que.push_front(20);
4 | que.front()
5 | que.back()
6 | que.pop_front()
7 | que.pop_back()
8 | cout << "Element at position 2 : " << que.at(2) <<
    endl;

```