

Contents

1	Tree Vector 建樹, 先序尋訪, 找尋最近共同祖先	2
2	Tree Disjoinset 並查集, 路徑壓縮	2
3	Tree Fenwick tree 鄰接表樹, 時間戳樹, 權值陣列, lowbit 修改查詢區間和	2
4	Trie 建樹, 修改, 查詢	3
5	Trie AC 自動機, 多模式字串數	4
6	BST 有序樹轉二元樹, 數組模擬樹	5
7	BST Stack 後序轉二元樹	5
8	BST 前序加中序找出後序	6
9	BST 後序二分搜尋樹還原	6
10	BST 建立結構指標二元樹, 前序尋訪	7
11	BST Heap priority queue 插入取出調整	7
12	BST Treap 樹堆積左旋右旋, 插入刪除	8
13	BST Treap rope 結構操作字串修改 Treap	9
14	BST 霍夫曼樹最小代價 Min Heap	10
15	Graph BFS 狀態空間搜尋最短路徑	10
16	Graph DFS 走訪, 建無向相鄰矩陣	11
17	Graph DFS 剪枝回朔, 狀態空間搜尋	12
18	Graph DFS 回朔找尋拓撲排序, 建相鄰有向邊	13
19	Graph DFS 計算圖連接性	14
20	Graph BFS 計算圖連接性	15
21	Graph 有向邊並查集, 速通性檢查, 樹判斷	15
22	Graph BFS 計算圖連接性	16

Tree Vector 建樹, 先序尋訪, 找尋最近共同祖先

```

1  // #include <bits/stdc++.h>
2  #include <iostream>
3  #include <vector>
4  #include <cstring>
5  using namespace std;
6  const int N = 10000;
7  vector<int> a[N];
8  int f[N], r[N];
9  void DFS(int u, int dep)
10 {
11     r[u] = dep;
12     for(vector<int>::iterator it =
13         a[u].begin(); it != a[u].end(); it++)
14         DFS(*it, dep + 1);
15 }
16 int main()
17 {
18     int casenum, num, n, i, x, y;
19     scanf("%d", &casenum);
20     for(num=0; num<casenum; num++)
21     {
22         scanf("%d", &n);
23         for(i=0; i<n; i++) a[i].clear();
24         memset(f, 255, sizeof(f));
25         for(i=0; i<n-1; i++)
26         {
27             scanf("%d %d", &x, &y);
28             a[x-1].push_back(y-1);
29             f[y-1] = x-1;
30         }
31         for(i=0; f[i]>=0; i++);
32         DFS(i, 0);
33         scanf("%d %d", &x, &y);
34         x--; y--;
35         while(x != y)
36         {
37             if(r[x] > r[y]) x = f[x];
38             else y = f[y];
39         }
40         printf("%d\n", x+1);
41     }
42     return 0;
43 }

```

Tree Disjoinset 並查集, 路徑壓縮

```

1  #include <iostream>
2  #include <vector>
3  #include <cstring>
4  const int maxn = 100000+5;
5  int n, m;
6  int set[maxn + maxn];

```

```

7  int set_find(int d)
8  {
9
10     if(set[d] < 0)
11         return d;
12     return set[d] = set_find(set[d]);
13 }
14 int main(void)
15 {
16
17     int loop;
18     scanf("%d", &loop);
19     while(loop--)
20     {
21         scanf("%d%d", &n, &m);
22         memset(set, -1, sizeof(set));
23         for(int i=0; i<m; i++)
24         {
25             int a, b;
26             char s[5];
27             scanf("%s%d%d", s, &a, &b);
28             if(s[0] == 'A')
29             {
30                 if(set_find(a) != set_find(b) &&
31                     set_find(a) != set_find(b+n))
32                     printf("%s\n", "Not sure
33                         yet.");
34                 else if(set_find(a) ==
35                     set_find(b))
36                     printf("%s\n", "In the same
37                         gang.");
38                 else
39                     printf("%s\n", "In different
40                         gangs.");
41             }
42             else
43             {
44                 if(set_find(a) != set_find(b+n))
45                 {
46                     set[set_find(a)] =
47                         set_find(b+n);
48                     set[set_find(b)] =
49                         set_find(a+n);
50                 }
51             }
52         }
53     }
54     return 0;
55 }

```

Tree Fenwick tree 鄰接表樹, 時間戳樹, 權值陣列, lowbit 修改查詢區間和

```

1  #include <iostream>
2  #include <vector>

```

```

3 #include<cstring>
4 const int maxn = 100000+5;
5 int n, m;
6 int set[maxn + maxn];
7 int set_find(int d)
8 {
9
10     if(set[d] < 0)
11         return d;
12     return set[d] = set_find(set[d]);
13 }
14 int main(void)
15 {
16
17     int loop;
18     scanf("%d", &loop);
19     while(loop--)
20     {
21         scanf("%d%d", &n, &m);
22         memset(set, -1, sizeof(set));
23         for(int i=0; i<m; i++)
24         {
25             int a, b;
26             char s[5];
27             scanf("%s%d%d", s, &a, &b);
28             if(s[0] == 'A')
29             {
30                 if(set_find(a) != set_find(b) &&
31                    set_find(a) != set_find(b+n))
32                     printf("%s\n", "Not sure
33                        yet.");
34                 else if(set_find(a) ==
35                    set_find(b))
36                     printf("%s\n", "In the same
37                        gang.");
38                 else
39                     printf("%s\n", "In different
40                        gangs.");
41             }
42             else
43             {
44                 if(set_find(a) != set_find(b+n))
45                 {
46                     set[set_find(a)] =
47                         set_find(b+n);
48                     set[set_find(b)] =
49                         set_find(a+n);
50                 }
51             }
52         }
53     }
54     return 0;
55 }

```

Trie 建樹, 修改, 查詢

```

1 #include <stdio>

```

```

2 #include <algorithm>
3 using namespace std;
4
5 const int MAXN = 1000 + 10; // 單詞的最大長度
6 const int maxnode = 100005; // Trie 樹的最大節點數量
7 const int sigma_size = 26; //
8     字母表的大小 (假設只有小寫字母)
9 char str[MAXN][25]; // 儲存單詞的陣列
10 int ch[maxnode][sigma_size]; // Trie 樹的子節點指標
11 int val[maxnode]; // Trie 樹節點的取值次數
12
13 // 定義 Trie 結構
14 struct Trie {
15     int sz; // Trie 樹的節點數量
16
17     // 初始化 Trie 樹
18     Trie() { sz = 1; memset(ch[0], 0,
19         sizeof(ch[0])); } // 根節點初始化
20
21     // 將字母轉成數字索引
22     int idx(char c) { return c - 'a'; }
23
24     // 插入單詞到 Trie 樹
25     void insert(char *s) {
26         int u = 0, n = strlen(s); // 起始於根節點 u
27         = 0
28         for (int i = 0; i < n; i++) {
29             int c = idx(s[i]); // 計算字母的索引值
30             if (!ch[u][c]) { //
31                 若該節點不存在, 則創建新節點
32                 memset(ch[sz], 0,
33                     sizeof(ch[sz])); // 初始化新節點
34                 ch[u][c] = sz++; //
35                 設置子節點並增加節點數量
36             }
37             u = ch[u][c]; // 移動到下一個節點
38             val[u]++; // 計算到達該節點的次數
39         }
40     }
41
42     // 查詢單詞在 Trie 中的最短前綴
43     void query(char *s) {
44         int u = 0, n = strlen(s); // 起始於根節點 u
45         = 0
46         for (int i = 0; i < n; i++) {
47             putchar(s[i]); // 輸出當前字母
48             int c = idx(s[i]); // 計算字母的索引值
49             if (val[ch[u][c]] == 1) return; //
50             若當前子節點的次數為 1, 則找到最短前綴
51             u = ch[u][c]; // 移動到下一個節點
52         }
53     }
54 }
55
56 int main() {
57     int tot = 0; // 單詞數初始化
58     Trie trie; // 建立 Trie 的結構體變數
59
60     // 讀取每個單詞並插入到 Trie

```

```

53 while (scanf("%s", str[tot]) != EOF) {
54     trie.insert(str[tot]); // 插入單詞到 Trie
55     tot++; // 單詞數累加
56 }
57
58 // 查詢每個單詞的最短唯一前綴
59 for (int i = 0; i < tot; i++) {
60     printf("%s ", str[i]); // 輸出單詞
61     trie.query(str[i]); // 查詢單詞的最短前綴
62     printf("\n"); // 換行
63 }
64
65 return 0;
66 }

```

Trie AC 自動機, 多模式字串數

```

1 #include <iostream>
2 #include <cstring>
3 #include <string>
4 #include <queue>
5 using namespace std;
6
7 const int MAXN = 1e6 + 6; // 適當調整大小, 根據需要
8 int cnt; // 記錄匹配模式字串的次數
9
10 // 節點結構定義
11 struct node {
12     int sum; // 該節點的匹配次數
13     node *next[26]; // 指向子節點的指標陣列
14     node *fail; // 失敗指針
15
16     node() : sum(0), fail(nullptr) {
17         for(int i = 0; i < 26; i++) next[i] =
18             nullptr;
19     };
20
21     node *root;
22     char key[70];
23     char pattern[MAXN];
24     int N;
25
26     // 插入模式字串到 Trie 樹中
27     void Insert(char *s)
28     {
29         node *p = root; // 開始於 Trie 樹的根節點
30         for (int i = 0; s[i]; i++) {
31             int x = s[i] - 'a'; // 計算字元索引
32             if (p->next[x] == nullptr) {
33                 p->next[x] = new node(); // 創建新節點
34             }
35             p = p->next[x]; // 移動到子節點
36         }
37         p->sum++; // 該節點匹配次數加 1
38     }
39
40     // 建立失敗指針

```

```

40 void build_fail_pointer()
41 {
42     queue<node*> q; // 使用 C++ 的隊列
43     root->fail = nullptr; // 根節點的失敗指針指向空
44
45     // 將根節點的子節點加入隊列並設置失敗指針
46     for (int i = 0; i < 26; i++) {
47         if (root->next[i] != nullptr) {
48             //
49             // 如果根的某子節點存在, 將該子節點的失敗指針設為根節點
50             root->next[i]->fail = root;
51             q.push(root->next[i]); //
52             // 將該子節點加入隊列
53         }
54         else {
55             root->next[i] = root; //
56             // 優化, 缺失的邊指向根節點
57         }
58     }
59
60     // BFS 建立失敗指針
61     while (!q.empty()) {
62         node* current = q.front(); q.pop(); //
63         // 取出隊首節點
64         for (int i = 0; i < 26; i++) {
65             if (current->next[i] != nullptr) {
66                 // 若存在子節點
67                 // 設置失敗指針
68                 node* fail_node = current->fail;
69                 // 從當前節點的失敗指針開始
70                 // 找到某個祖先節點的匹配邊
71                 while (fail_node != nullptr &&
72                     fail_node->next[i] ==
73                     nullptr)
74                     fail_node = fail_node->fail;
75                 // 繼續沿著失敗指針向上
76                 if (fail_node == nullptr)
77                     current->next[i]->fail =
78                     root; // 若找不到則指向根節點
79                 else
80                     current->next[i]->fail =
81                     fail_node->next[i]; //
82                 // 否則設置為找到的節點
83                 q.push(current->next[i]); //
84                 // 將該子節點加入隊列
85             }
86         }
87     }
88
89     // 在目標字串中運行 AC 自動機, 進行多模式匹配
90     void ac_automation(char *ch) {

```

```

84  node *p = root; // 從根節點開始
85  int len = strlen(ch); // 目標字串的長度
86  for (int i = 0; i < len; i++) {
87      int x = ch[i] - 'a'; // 當前字元索引
88      while (p->next[x] == root && p != root)
89          p = p->fail;
90
91      p = p->next[x];
92      if (!p)
93          p = root;
94
95      node *temp = p;
96      while (temp != root) { //
97          往上沿失敗指針累計所有匹配結果
98          if (temp->sum >= 0) { // 如果是匹配節點
99              cnt += temp->sum; // 累計匹配次數
100             temp->sum = -1; // 設置為 -1
101             以避免重複計算
102         }
103         else
104             break;
105         temp = temp->fail; // 沿失敗指針往上跳轉
106     }
107 }
108 int main()
109 {
110     int T; // 測試案例數量
111     cin >> T;
112     while (T--)
113     {
114         // 建立根節點
115         root = new node();
116         // 讀取模式字串數量
117         cin >> N;
118         cin.ignore(); // 忽略換行符
119
120         for (int i = 1; i <= N; i++)
121         {
122             // 讀取模式字串
123             cin.getline(key, sizeof(key));
124             Insert(key); // 將模式字串插入到 Trie 樹中
125         }
126         // 讀取目標字串
127         cin.getline(pattern, sizeof(pattern));
128         cnt = 0;
129         build_fail_pointer(); // 建立失敗指針
130         ac_automation(pattern); // 使用 AC
131         自動機進行匹配
132         cout << cnt << "\n"; //
133         輸出匹配到的模式字串次數
134     }
135     return 0;
136 }
137 /*
138 輸入範例:
139 1

```

```

139 5
140 she
141 he
142 say
143 shr
144 her
145 yasherhs
146
147 預期輸出:
148 3
149 */

```

BST 有序樹轉二元樹，數組模擬樹

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  string s;
6  int i, n = 0, height1, height2;
7
8  void work(int level1, int level2) {
9      int tempson = 0;
10     while (s[i] == 'd') {
11         i++;
12         tempson++;
13         work(level1 + 1, level2 + tempson);
14     }
15     height1 = max(height1, level1);
16     height2 = max(height2, level2);
17     if (s[i] == 'u') i++;
18 }
19
20 int main() {
21     while (cin >> s && s != "#") {
22         i = height1 = height2 = 0;
23         work(0, 0);
24         cout << "Tree " << ++n << ": " <<
25             height1 << " => " << height2 <<
26             endl;
27     }
28     return 0;
29 }

```

BST Stack 後序轉二元樹

```

1  #include <iostream>
2  #include <stack>
3  #include <queue>
4  using namespace std;
5
6  const int maxn = 11000;
7
8  struct node {
9      int l, r;
10     char c;

```

```

11 } e[maxn];
12
13 int cnt;
14 char s[maxn];
15
16 void initial() {
17     int len = strlen(s);
18     for (int i = 0; i <= len; i++) {
19         e[i].l = e[i].r = -1;
20     }
21     cnt = 0;
22 }
23
24 void solve() {
25     int len = strlen(s);
26     stack<int> v;
27     for (int i = 0; i < len; i++) {
28         if (s[i] >= 'a' && s[i] <= 'z') {
29             e[cnt].c = s[i];
30             v.push(cnt);
31             cnt++;
32         } else {
33             int r = v.top();
34             v.pop();
35             int l = v.top();
36             v.pop();
37             e[cnt].l = l;
38             e[cnt].r = r;
39             e[cnt].c = s[i];
40             v.push(cnt);
41             cnt++;
42         }
43     }
44 }
45
46 void output() {
47     string ans;
48     queue<int> q;
49     q.push(cnt - 1);
50     while (!q.empty()) {
51         int st = q.front();
52         q.pop();
53         ans.push_back(e[st].c);
54         if (e[st].l != -1) q.push(e[st].l);
55         if (e[st].r != -1) q.push(e[st].r);
56     }
57     reverse(ans.begin(), ans.end());
58     printf("%s\n", ans.c_str());
59 }
60
61 int main() {
62     while (scanf("%s", s) != EOF) {
63         initial();
64         solve();
65         output();
66     }
67     return 0;
68 }

```

BST 前序加中序找出後序

```

1 #include <stdio.h>
2 #include <string.h>
3
4 char preord[30], inord[30];
5
6 int read_case() {
7     if (scanf("%s %s", preord, inord) != 2)
8         return 0;
9     return 1;
10 }
11
12 void recover(int preleft, int preright, int
13             inleft, int inright) {
14     // 首先根據前序字串中的根節點判斷樹結構，計算左右子樹
15     int root, leftsize, rightsize;
16
17     for (root = inleft; root <= inright;
18         root++) {
19         if (preord[preleft] == inord[root])
20             break; // 找到根的位置
21     }
22
23     leftsize = root - inleft;
24     rightsize = inright - root;
25
26     if (leftsize > 0) // 遞迴左子樹
27         recover(preleft + 1, preleft +
28               leftsize, inleft, root - 1);
29
30     if (rightsize > 0) // 遞迴右子樹
31         recover(preleft + leftsize + 1,
32               preright, root + 1, inright);
33
34     printf("%c", inord[root]); // 輸出根節點
35 }
36
37 void solve_case() {
38     int n = strlen(preord);
39     recover(0, n - 1, 0, n - 1);
40     printf("\n");
41 }
42
43 int main() {
44     while (read_case()) solve_case();
45     return 0;
46 }

```

BST 後序二分搜尋樹還原

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 using namespace std;

```

```

5
6 int n; // 節點總數
7 int a[3010]; // 儲存後序遍歷
8
9 void solve(int l, int r) { // l 和 r
    是左子樹和右子樹的範圍
10
11     if (l > r) return; // 如果範圍無效則返回
12
13     int i = l;
14     while (i < r && a[i] < a[r]) i++; //
        找到分界點 i, 使得左子樹的元素都小於 a[r]
15
16     if (i < r) solve(i, r - 1); // 遞迴處理右子樹
17     if (l < i) solve(l, i - 1); // 遞迴處理左子樹
18
19     printf("%d\n", a[r]); // 輸出根節點
20 }
21
22 int main() {
23     scanf("%d", &n); // 輸入節點總數
24     for (int i = 0; i < n; ++i) // 輸入後序遍歷
25         scanf("%d", &a[i]);
26
27     solve(0, n - 1); // 計算並輸出右子樹-左子樹-根的遍歷
28     return 0;
29 }

```

BST 建立結構指標二元樹, 前序尋訪

```

1 #include <stdio.h>
2
3 typedef struct binTreeNode { // 定義二元搜尋樹的結構
4     int data;
5     struct binTreeNode *lchild, *rchild;
6 } *BT;
7
8 void add(BT &T, int val) { // 將順序值 val
    插入二元搜尋樹
9     if (T == NULL) { // 若 T 為空, 則找到插入位置
10         T = new binTreeNode(); //
            申請記憶體, 建構儲存 val 的葉節點
11         T->data = val;
12         T->lchild = T->rchild = NULL;
13     } else if (T->data > val) { // 若 val
        小於根節點值, 則沿左子樹方向尋找插入點
14         add(T->lchild, val);
15     } else { // 若 val
        不小於根節點值, 則沿右子樹方向尋找插入位置
16         add(T->rchild, val);
17     }
18 }
19
20 void preOrder(BT T, bool flag) { //
    前序輸出樹的順序, 參數 flag 為首節點標誌
21     if (T == NULL)

```

```

22         return;
23     else {
24         if (!flag) // 若節點非首節點, 則尾隨空格
25             printf(" ");
26         printf("%d", T->data); // 輸出 T 的順序值
27         preOrder(T->lchild, 0); //
            分別遞迴左子樹和右子樹
28         preOrder(T->rchild, 0);
29     }
30 }
31
32 int main() {
33     BT T;
34     int n, v;
35     while (~scanf("%d", &n)) { //
        二元搜尋樹的根節點數量
36         T = NULL; // 初始化二元搜尋樹
37         for (int i = 0; i < n; i++) {
38             scanf("%d", &v); // 輸入一個個順序值
39             add(T, v); // 插入二元搜尋樹
40         }
41         preOrder(T, 1); // 按照前序搜尋的順序輸出樹的順序
42         printf("\n");
43     }
44     return 0;
45 }

```

BST Heap priority queue 插入取出調整

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 #include <queue>
5
6 using namespace std;
7
8 const int maxn = 60000 + 10;
9 const int maxs = 100;
10
11 // Structure to hold the information
12 struct Info {
13     char name[maxs];
14     int para, pri, t;
15 };
16
17 Info p[maxn];
18 int used = 0; // Next available index in p[]
19 int cnt = 0; // Counter to maintain insertion order
20
21 // Comparator for the priority queue
22 struct Compare {
23     bool operator()(const int a, const int b)
24         const {
25             if (p[a].pri != p[b].pri)
26                 return p[a].pri > p[b].pri; //
                Min-heap based on pri

```

```

26     return p[a].t > p[b].t; // If pri equal,
           earlier t has higher priority
27 }
28 };
29
30 int main(void) {
31     char command[maxs];
32
33     // Define the priority queue with the custom comparator
34     priority_queue<int, vector<int>, Compare>
        pq;
35
36     // Read commands until end of file
37     while (scanf("%s", command) != EOF) {
38         if (strcmp(command, "GET") == 0) { // GET
           command
39             if (!pq.empty()) {
40                 int top_idx = pq.top(); // Get the
           index of the highest priority element
41                 pq.pop(); // Remove it from the queue
42                 printf("%s %d\n",
           p[top_idx].name,
           p[top_idx].para);
43             } else {
44                 printf("EMPTY QUEUE!\n");
45             }
46         } else { // Insert command
47             // Read the name, para, and pri for the new
           Info object
48             scanf("%s %d %d", p[used].name,
           &p[used].para, &p[used].pri);
49             p[used].t = cnt++; // Assign and
           increment the insertion order
50             pq.push(used++); // Push the index into
           the priority queue
51         }
52     }
53
54     return 0;
55 }
56
57 /*
58 GET
59 PUT msg1 10 5
60 PUT msg2 10 4
61 GET
62 GET
63 GET
64 */

```

BST Treap 樹堆積左旋右旋, 插入 刪除

```

1 #include <stdio>
2 #include <stdlib>
3 using namespace std;
4

```

```

5 struct Node {
6     Node *ch[2]; // 左右指標
7     int v, r, info; // v 是客戶優先順序, info
           是客戶的編號, r 由 rand() 產生, 作為節點的優先順序
8
9     Node(int v, int info) : v(v), info(info) {
10         r = rand(); // 隨機產生節點優先順序
11         ch[0] = ch[1] = NULL; // 左右指標為空
12     }
13
14     int cmp(int x) { // 客戶優先順序 v 與 x 比較大小
15         if (v == x) return -1;
16         return x < v ? 0 : 1;
17     }
18 };
19
20 void rotate(Node *&o, int d) { // 節點 o 旋轉, 方向
           d = 0 左旋, 1 右旋
21     Node *k = o->ch[d^1];
22     o->ch[d^1] = k->ch[d];
23     k->ch[d] = o;
24     o = k;
25 }
26
27 void insert(Node *&o, int v, int info) { //
           插入一個節點 info, 優先順序為 v
28     if (o == NULL) {
29         o = new Node(v, info); //
           若找到插入位置, 則客戶作為節點插入
30     }
31     else {
32         // **Corrected Insertion Direction**: Place higher
           'v' to the right
33         int d = v < o->v ? 0 : 1;
34         insert(o->ch[d], v, info);
35         if (o->ch[d]->r > o->r) rotate(o, d^1);
           // 若方向 d 的子樹優先順序較小, 則進行旋轉
36     }
37 }
38
39 void remove(Node *&o, int v) { // 在 o
           為根的樹狀堆棧中, 刪除優先順序 v 的節點
40     if (!o) return;
41     int d = o->cmp(v);
42     if (d == -1) { // 如果找到該節點
43         Node *u = o;
44         if (o->ch[0] && o->ch[1]) { // 若 o
           有左右子樹, 則計算被刪除節點的方向
45             int d2 = o->ch[0]->r < o->ch[1]->r ?
           1 : 0;
46             rotate(o, d2);
47             remove(o->ch[d2], v);
48         } else { // 若 o 節點僅有一個子樹, 則將其子樹取代 o
49             o = o->ch[0] ? o->ch[0] : o->ch[1];
50             delete u;
51         }
52     } else {
53         remove(o->ch[d], v); // 若 o
           節點為葉節點, 直接將其刪除

```



```

54     }
55 }
56
57 int find_max(Node *o) { // 在 o
    為根的樹狀堆棧中尋找最大優先順序
58     if (!o) return -1; // Handle empty treap
59     while (o->ch[1] != NULL) o = o->ch[1];
60     printf("%d\n", o->info);
61     return o->v;
62 }
63
64 int find_min(Node *o) { // 在 o
    為根的樹狀堆棧中尋找最小優先順序
65     if (!o) return -1; // Handle empty treap
66     while (o->ch[0] != NULL) o = o->ch[0];
67     printf("%d\n", o->info);
68     return o->v;
69 }
70
71 int main() {
72     int op;
73     Node *root = NULL;
74     while (scanf("%d", &op) == 1 && op) {
75         if (op == 1) { // 若輸入為新增客戶
76             int v, info;
77             scanf("%d%d", &info, &v);
78             insert(root, v, info);
79         } else if (op == 2) { // 若輸入為最大優先順序
80             if (root == NULL) {
81                 printf("0\n");
82                 continue;
83             }
84             int v = find_max(root);
85             if (v != -1) remove(root, v);
86         } else if (op == 3) { // 若輸入為最小優先順序
87             if (root == NULL) {
88                 printf("0\n");
89                 continue;
90             }
91             int v = find_min(root);
92             if (v != -1) remove(root, v);
93         }
94     }
95     return 0;
96 }
97
98 /*
99 Sample Input:
100 2
101 1 20 14
102 1 30 3
103 2
104 1 10 99
105 3
106 2
107 2
108 0
109
110 Expected Output:

```

```

111 0
112 20
113 30
114 10
115 0
116 */

```

BST Treap rope 結構操作字串修改 Treap

```

1 #include <iostream>
2 #include <ext/rope> // 使用 GNU C++ rope 函式庫
3 using namespace std;
4 using namespace __gnu_cxx; // rope 所在的命名空間
5
6 /*
7 rope 函式庫提供的基本操作有:
8 list.insert(p, str); // 將字串 str 插入到 rope 的 p 位置
9 list.erase(p, c); // 刪除 rope 中從 p 位置開始的 c 個字元
10 list.substr(p, c); // 擷取 rope 中從 p 位置開始長度為 c
    的子字串
11 list.copy(q, p, c); // 將 rope 中從 p 位置開始長度為 c
    的子字串複製到 q
12 */
13
14 using namespace std;
15
16 rope<char> ro, tmp; // 定義 rope 物件
17 rope<char> l[50005]; // 紀錄每個版本
18
19 char str[205]; // 用於暫存輸入的字串
20
21 int main() {
22     int n, op, p, c, d, v, cnt;
23     cin >> n;
24     d = 0;
25     cnt = 1;
26     while (n-->0) {
27         cin >> op;
28
29         if (op == 1) { // 插入命令
30             cin >> p >> str;
31             p -= d; // 計算相對位置
32             ro.insert(p, str); // 將字串 str 插入
                rope 的 p 位置
33             l[cnt++] = ro; // 將版本 ro 儲存到
                l[cnt], 版本計數 + 1
34         } else if (op == 2) { // 刪除命令
35             cin >> p >> c;
36             p -= d; c -= d; // 計算相對位置和長度
37             ro.erase(p-1, c); // 刪除 rope 中從 p
                位置開始的 c 個字元
38
39             l[cnt++] = ro; // 將版本 ro 儲存到
                l[cnt], 版本計數 + 1
40         } else { // 列印命令
41             cin >> v >> p >> c;

```

```

42
43     p -= d; v -= d; c -= d; //
        計算相對位置和長度
44     tmp = l[v].substr(p-1, c); // 擷取版本
        v 中從 p 位置開始長度為 c 的子字串
45     d += count(tmp.begin(), tmp.end(),
        'c'); // 計算子字串 tmp 中 'c' 的出現次數
46     cout << tmp << "\n"; // 輸出子字串 tmp
47 }
48 }
49
50     return 0;
51 }
52
53 /*
54 6
55 1 0 abcdefgh
56 2 4 3
57 3 1 2 5
58 */

```

BST 霍夫曼樹最小代價 Min Heap

```

1 #include <iostream>
2 #include <queue>
3 #include <algorithm>
4 using namespace std;
5
6 const int maxn = 1e5 + 100;
7 typedef long long ll;
8 queue<ll> q1, q2;
9 ll a[maxn];
10 ll t, n;
11 bool Huffman(int x) {
12     // 清空 q1 和 q2 佇列
13     while (!q1.empty()) q1.pop();
14     while (!q2.empty()) q2.pop();
15
16     int tt = 0;
17     // 模擬 k 元霍夫曼樹: 計算要使用的虛葉節點數
18     if ((n - 1) % (x - 1) != 0) tt = (x - 1) -
        (n - 1) % (x - 1);
19
20     // 將虛葉節點加入 q1
21     for (int i = 1; i <= tt; i++) q1.push(0);
22     for (int i = 1; i <= n; i++) q1.push(a[i]);
        // 將序列元素加入 q1
23
24     ll sum = 0;
25     while (1) {
26         ll tem = 0;
27         for (int i = 1; i <= x; i++) { // 每次取出
            x 個元素
28             if (q1.empty() && q2.empty()) break;
29             if (q1.empty()) {
30                 tem += q2.front();
31                 q2.pop();

```

```

32             } else if (q2.empty()) {
33                 tem += q1.front();
34                 q1.pop();
35             } else if (q1.front() < q2.front()) {
36                 tem += q1.front();
37                 q1.pop();
38             } else {
39                 tem += q2.front();
40                 q2.pop();
41             }
42         }
43         sum += tem;
44         if (q1.empty() && q2.empty()) break;
45         q2.push(tem);
46         if (sum > t) return 0;
47     }
48     return sum <= t;
49 }
50
51 int main() {
52     int T;
53     scanf("%d", &T); // 測試案例數量
54     while (T--) {
55         scanf("%lld%lld", &n, &t); // 輸入 n 和 t
56         for (int i = 1; i <= n; i++)
57             scanf("%lld", &a[i]); // 輸入每個序列元素
        排序以便進行霍夫曼合併
58
59         int st = 2, en = n;
60         while (st < en) { // 使用二分法找最小的 k 值
61             int mid = (st + en) / 2;
62             if (Huffman(mid)) en = mid;
63             else st = mid + 1;
64         }
65         printf("%d\n", st); // 輸出最小的 k 值
66     }
67     return 0;
68 }

```

Graph BFS 狀態空間搜尋最短路徑

```

1 #include <iostream>
2 #include <queue>
3 #include <cstring>
4 // #include <cmath> // Removed since we're no longer using
    pow
5 using namespace std;
6
7 // Define the node struct with an explicit constructor
8 struct node {
9     int k, step;
10     node(int k_val, int step_val) : k(k_val),
        step(step_val) {}
11 };
12

```

```

13 // Precompute powers of 10 for digit manipulation
14 const int power10_arr[4] = {1000, 100, 10, 1};
15
16 bool p[10000], s_array[10000]; // Renamed 's' to
    's_array' to avoid confusion
17
18 // Sieve of Eratosthenes to generate prime numbers up to n
19 void make_sieve(int n) {
20     memset(p, 0, sizeof(p));
21     p[0] = 1; // 0 is not a prime
22     p[1] = 1; // 1 is not a prime
23     for (int i = 2; i <= n; i++) {
24         if (!p[i]) {
25             for (int j = i * i; j <= n; j += i)
26                 p[j] = 1; // Mark multiples of i as
                    non-prime
27         }
28     }
29 }
30
31 // Function to change the digit at a specific position
32 int change_digit(int x, int pos, int
    new_digit) {
33     int digits[4] = { x / 1000, (x / 100) % 10,
        (x / 10) % 10, x % 10 };
34     digits[pos - 1] = new_digit;
35     return digits[0] * 1000 + digits[1] * 100 +
        digits[2] * 10 + digits[3];
36 }
37
38 int main() {
39     // Optimize I/O operations
40     ios::sync_with_stdio(false);
41     cin.tie(NULL);
42
43     make_sieve(9999); // Generate primes up to 9999
44
45     int tot;
46     cin >> tot;
47     while (tot--) {
48         int x, y;
49         cin >> x >> y;
50
51         // Validate that both x and y are 4-digit primes
52         if (x < 1000 || x > 9999 || y < 1000 ||
            y > 9999 || p[x] || p[y]) {
53             cout << "Impossible" << endl;
54             continue;
55         }
56
57         // Initialize the BFS queue and visited array
58         queue<node> q;
59         q.push(node(x, 0)); // Use constructor to
            initialize node
60
61         memset(s_array, 0, sizeof(s_array));
62         s_array[x] = 1;
63         int ans = -1;
64         while (!q.empty()) {

```

```

65         node cur = q.front();
66         q.pop();
67
68         if (cur.k == y) {
69             ans = cur.step;
70             break;
71         }
72
73         for (int i = 1; i <= 4; i++) { //
            Change each digit position
74             for (int j = 0; j <= 9; j++) {
75                 // Skip if trying to set the first
                    digit to 0 or if the digit is
                    the same
76                 if ((i == 1 && j == 0) ||
                    ((cur.k / power10_arr[i -
                        1]) % 10) == j)
77                     continue;
78
79                 int tk = change_digit(cur.k,
                    i, j);
80
81                 // Check if the new number is a prime,
                    within range, and hasn't been
                    visited
82                 if (tk >= 1000 && tk <= 9999
                    && !p[tk] &&
                    !s_array[tk]) {
83                     s_array[tk] = 1;
84                     q.push(node(tk, cur.step
                        + 1)); // Use constructor
                        to initialize node
85                 }
86             }
87         }
88     }
89
90     if (ans >= 0)
91         cout << ans << "\n";
92     else
93         cout << "Impossible\n";
94 }
95
96 return 0;
97 }

```

Graph DFS 走訪, 建無向相鄰矩陣

```

1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 int map[6][6]; // 無向圖的相鄰矩陣
6
7 // 產生無向圖的相鄰矩陣
8 void makemap() {
9     memset(map, 0, sizeof(map));

```

```

10     for (int i = 1; i <= 5; i++) {
11         for (int j = 1; j <= 5; j++) {
12             if (i != j) map[i][j] = 1;
13         }
14     }
15     map[4][1] = map[1][4] = 0;
16     map[4][2] = map[2][4] = 0;
17 }
18
19 // 深度優先搜索，找到所有可能的存取路徑
20 void dfs(int x, int k, string s) {
21     s += char(x + '0'); // 將當前節點 x 加入存取序列
22
23     if (k == 8) { // 如果已完成一筆順序
24         cout << s << endl;
25         return;
26     }
27
28     for (int y = 1; y <= 5; y++) { //
29         // 依照節點順序訪問相鄰節點
30         if (map[x][y]) {
31             map[x][y] = map[y][x] = 0; //
32             // 設定邊為已訪問
33             dfs(y, k + 1, s); // 遞迴搜索
34             map[x][y] = map[y][x] = 1; //
35             // 恢復邊的狀態
36         }
37     }
38 }
39
40 int main() {
41     makemap(); // 產生無向圖的相鄰矩陣
42     dfs(1, 0, ""); // 從節點 1 開始計算所有可能的存取順序
43     return 0;
44 }

```

Graph DFS 剪枝回溯，狀態空間搜尋

```

1 #include <iostream>
2 #include <algorithm>
3 using namespace std;
4
5 int sticks[65]; // 用來存放木棍的長度
6 int used[65]; // 標記木棍是否被使用
7 int n, len, sum; //
8     // 木棍的數量，木棍的目標長度，所有木棍的總和
9
10 // 深度優先搜尋，檢查是否可以用目前的木棍切成長度為 len 的木棍
11 bool dfs(int i, int l, int t) {
12     int j;
13
14     // 若長度達到 len，則成功構成一根木棍
15     if (l == 0) {
16         t -= len;
17         if (t == 0) return true;

```

```

18     for (i = 0; used[i]; ++i); //
19     // 找到下一個未使用的木棍
20     used[i] = 1; // 標記該木棍為已使用
21     if (dfs(i + 1, len - sticks[i], t))
22         return true; // 遞迴切割
23     used[i] = 0;
24     return false;
25 } else {
26     for (int j = i; j < n; ++j) { //
27         // 從長度遞減順序尋找木棍 j 到木棍 n - 1
28         if (j > 0 && sticks[j] == sticks[j - 1] && !used[j - 1]) continue; //
29         // 若長度相同且木棍 j - 1 沒有被使用則跳過
30         if (!used[j] && sticks[j] <= l) { //
31             // 若木棍沒有被使用且長度小於等於當前長度
32             used[j] = 1;
33             if (dfs(j + 1, l - sticks[j], t)) return true; // 遞迴嘗試
34             used[j] = 0;
35             if (sticks[j] == l) break; //
36             // 若木棍無法完成切割則跳過
37         }
38     }
39     return false;
40 }
41
42 // 木棍長度的比較函數
43 bool cmp(const int a, const int b) {
44     return a > b;
45 }
46
47 int main() {
48     while (cin >> n && n) { // 讀取木棍數量，直至輸入
49         // 0 為止
50         int sum = 0;
51         for (int i = 0; i < n; ++i) {
52             cin >> sticks[i];
53             sum += sticks[i]; // 計算木棍總長度
54             used[i] = 0; // 初始化使用標記
55         }
56         sort(sticks, sticks + n, cmp); //
57         // 按木棍長度降序排列
58         bool flag = false;
59         for (len = sticks[0]; len <= sum / 2;
60             ++len) { // 在 [sticks[0]..sum/2] 區間搜尋
61             if (sum % len == 0) { // 若總長度能被 len
62                 // 整除
63                 if (dfs(0, len, sum)) { // 若長度為
64                     // len 的木棍能夠切成 n 根木棍，則標記成功
65                     flag = true;
66                     cout << len << endl; //
67                     // 輸出木棍的最小可能長度並結束計算
68                     break;
69                 }
70             }
71         }
72         if (!flag) cout << sum << endl; //
73         // 若找不到符合條件的木棍長度，則輸出木棍的總長度

```

```

61     }
62     return 0;
63 }

```

Graph DFS 回溯找尋拓模排序, 建相鄰有向邊

```

1  #include <iostream>
2  #include <vector>
3  #include <cstring>
4  #include <queue>
5  #include <string> // Ensure you have included <string>
6
7  using namespace std;
8
9  // Function to perform DFS to check if target is reachable
   from current
10 bool is_reachable(int current, int target, int
   N, int g[][100], bool visited[]) {
11     if (current == target) return true;
12     visited[current] = true;
13     for (int i = 0; i < N; i++) {
14         if (g[current][i] && !visited[i]) {
15             if (is_reachable(i, target, N, g,
   visited))
16                 return true;
17         }
18     }
19     return false;
20 }
21
22 int main() {
23     ios::sync_with_stdio(false);
24     cin.tie(0);
25
26     int N, K;
27     while (cin >> N >> K) {
28         if (N == 0 && K == 0) break; // Terminate
   when N and K are both zero
29
30         // Initialize adjacency matrix and in-degree count
31         int g[100][100];
32         memset(g, 0, sizeof(g));
33         vector<int> in_degree(N, 0);
34
35         bool determined = false;
36         bool inconsistent = false;
37         string relation;
38
39         for (int i = 0; i < K; i++) {
40             cin >> relation;
41             if (relation.size() < 3 ||
   (relation[1] != '<' &&
   relation[1] != '>')) {
42                 // Handle invalid input format
43                 cout << "Invalid relation
   format." << endl;

```

```

44         inconsistent = true;
45         break;
46     }
47
48     char a = relation[0];
49     char b = relation[2];
50     char op = relation[1];
51     int x, y;
52
53     // Determine the direction of the relation
54     if (op == '<') {
55         x = a - 'A';
56         y = b - 'A';
57     }
58     else { // op == '>'
59         x = b - 'A';
60         y = a - 'A';
61     }
62
63     // Check if adding edge x -> y creates a cycle
64     bool visited[100];
65     memset(visited, false,
   sizeof(visited));
66     if (is_reachable(y, x, N, g,
   visited)) {
67         // Adding edge x->y would create a cycle
68         cout << "Inconsistency found
   after " << (i + 1) << "
   relations." << endl;
69
70         // Read and discard remaining relations
   for this test case
71         for (int j = i + 1; j < K; j++) {
72             cin >> relation;
73         }
74
75         inconsistent = true;
76         break;
77     }
78
79     // Add the edge x -> y
80     g[x][y] = 1;
81     in_degree[y]++;
82
83     // Perform Topological Sort to check if a
   unique sequence is determined
84     // Create a copy of in_degree to manipulate
85     vector<int> in_copy = in_degree;
86     queue<int> Q;
87     vector<int> result;
88
89     // Enqueue all nodes with in-degree 0
90     for (int node = 0; node < N; node++)
91     {
92         if (in_copy[node] == 0) {
93             Q.push(node);
94         }
95     }

```

```

96     bool multiple = false; // Flag to check
97         if multiple sequences are possible
98     while (!Q.empty()) {
99         if (Q.size() > 1) {
100             multiple = true; // More than one
101                 node with in-degree 0 implies
102                 multiple sequences
103         }
104         int current = Q.front();
105         Q.pop();
106         result.push_back(current);
107
108         // Decrease in-degree of neighboring nodes
109         for (int neighbor = 0; neighbor
110             < N; neighbor++) {
111             if (g[current][neighbor]) {
112                 in_copy[neighbor]--;
113                 if (in_copy[neighbor] ==
114                     0) {
115                     Q.push(neighbor);
116                 }
117             }
118         }
119
120         // Check if a unique sorted sequence is
121         determined
122         if (result.size() == N && !multiple)
123         {
124             cout << "Sorted sequence
125                 determined after " << (i +
126                 1) << " relations: ";
127
128             // Replace range-based for loop with
129             traditional for loop
130             for (size_t idx = 0; idx <
131                 result.size(); idx++) {
132                 int node = result[idx];
133                 cout << char('A' + node);
134             }
135             cout << "." << endl;
136
137             // Read and discard remaining relations
138             for this test case
139             for (int j = i + 1; j < K; j++) {
140                 cin >> relation;
141             }
142
143             determined = true;
144             break;
145         }
146     }
147
148     // If no inconsistency or unique sequence was found
149     after processing all relations
150     if (!inconsistent && !determined) {
151         cout << "Sorted sequence cannot be
152             determined." << endl;

```

```

141     }
142 }
143
144 return 0;
145 }

```

Graph DFS 計算圖連接性

```

1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 int map[105][105]; // Grid map indicating oil and
6     empty spots
7 int vis[105][105]; // Visited marker array
8 int n, m; // Rows and columns of the grid
9
10 // DFS function to explore connected components
11 void dfs(int x, int y) {
12     vis[x][y] = 1; // Mark current cell as visited
13
14     // Explore all 8 possible directions
15     if (x + 1 < n && y < m && !vis[x + 1][y] &&
16         map[x + 1][y]) dfs(x + 1, y);
17     if (x - 1 >= 0 && y < m && !vis[x - 1][y]
18         && map[x - 1][y]) dfs(x - 1, y);
19     if (x < n && y + 1 < m && !vis[x][y + 1] &&
20         map[x][y + 1]) dfs(x, y + 1);
21     if (x < n && y - 1 >= 0 && !vis[x][y - 1]
22         && map[x][y - 1]) dfs(x, y - 1);
23     if (x + 1 < n && y + 1 < m && !vis[x + 1][y
24         + 1] && map[x + 1][y + 1]) dfs(x + 1,
25         y + 1);
26     if (x - 1 >= 0 && y - 1 >= 0 && !vis[x -
27         1][y - 1] && map[x - 1][y - 1]) dfs(x
28         - 1, y - 1);
29     if (x + 1 < n && y - 1 >= 0 && !vis[x +
30         1][y - 1] && map[x + 1][y - 1]) dfs(x
31         + 1, y - 1);
32     if (x - 1 >= 0 && y + 1 < m && !vis[x -
33         1][y + 1] && map[x - 1][y + 1]) dfs(x
34         - 1, y + 1);
35 }
36
37 // Function to initialize visited array to zero
38 void init() {
39     memset(vis, 0, sizeof(vis));
40 }
41
42 int main() {
43     char ch;
44     while (cin >> n >> m) { // Read grid dimensions
45         if (n == 0 && m == 0) break;
46
47         init(); // Clear the visited markers
48
49         for (int i = 0; i < n; i++) {

```

```

37     for (int j = 0; j < m; j++) {
38         cin >> ch; // Read each cell of the grid
39         if (ch == '*')
40             map[i][j] = 0; // Mark empty spot
41         else
42             map[i][j] = 1; // Mark oil spot
43     }
44 }
45
46 int count = 0; // Initialize oil deposit count
47
48 for (int i = 0; i < n; i++) {
49     for (int j = 0; j < m; j++) {
50         if (!vis[i][j] && map[i][j]) {
51             dfs(i, j); // Run DFS from each
52                         // unvisited oil spot
53             count++; // Increment oil deposit
54                       // count
55         }
56     }
57 }
58 cout << count << endl; // Output the count
59 // of different oil deposits
60
61 return 0;
62 }

```

Graph BFS 計算圖連接性

```

1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 struct Position {
6     int i, j; // 網格位置
7 } bfsQueue[10000]; // BFS 的佇列, 重新命名為 bfsQueue
8 // 避免與標準庫衝突
9
10 int m, n; // 網格的行數 m 和列數 n
11 char map[101][101]; // 相鄰矩陣, '*' 表示牆, '@'
12 // 表示油田
13 int a[8][2] = {{-1, 0}, {1, 0}, {0, -1}, {0,
14 1}, {-1, -1}, {-1, 1}, {1, -1}, {1, 1}};
15 // 8 個方向的移動
16
17 void BFS(int i, int j) {
18     int front = 0, rear = 1; // 佇列的首尾標誌初始化
19     bfsQueue[front].i = i;
20     bfsQueue[front].j = j;
21     map[i][j] = '*'; // 將起點設為無油狀態
22
23     while (front != rear) {
24         int ii = bfsQueue[front].i;
25         int jj = bfsQueue[front].j;
26         front++; // 佇列首指標 +1
27
28         for (int k = 0; k < 8; k++) {
29             int t1 = ii + a[k][0];
30             int t2 = jj + a[k][1];
31
32             if (map[t1][t2] == '@') { // 若 (t1,
33                                     // t2) 是油田
34                 bfsQueue[rear].i = t1;
35                 bfsQueue[rear].j = t2;
36                 map[t1][t2] = '*'; // 將 (t1, t2)
37                                     // 設為無油狀態
38                 rear++; // 佇列尾指標 +1
39             }
40         }
41     }
42 }
43
44 int main() {
45     int i, j;
46     int num;
47
48     while (scanf("%d %d", &m, &n) && m) { //
49         // 反覆輸入行數 m 和列數 n, 直到 m 為 0
50         num = 0;
51         for (i = 0; i < m; i++)
52             scanf("%s", map[i]); //
53         // 自上而下, 從左至右讀取每個網格
54
55         for (i = 0; i < m; i++)
56             for (j = 0; j < n; j++)
57                 if (map[i][j] == '@') { // 若 (i,
58                                         // j) 為油田
59                     num++; // 不同的油田數量 +1
60                     BFS(i, j); // 透過 BFS 將 (i, j)
61                                 // 可達的所有油田設為無油狀態
62                 }
63
64         printf("%d\n", num); // 輸出油田數
65     }
66
67     return 0;
68 }

```

```

24     for (int k = 0; k < 8; k++) { // 8
25         // 個相鄰方向
26         int t1 = ii + a[k][0];
27         int t2 = jj + a[k][1];
28
29         if (map[t1][t2] == '@') { // 若 (t1,
30                                     // t2) 是油田
31             bfsQueue[rear].i = t1;
32             bfsQueue[rear].j = t2;
33             map[t1][t2] = '*'; // 將 (t1, t2)
34                                     // 設為無油狀態
35             rear++; // 佇列尾指標 +1
36         }
37     }
38 }
39
40 int main() {
41     int i, j;
42     int num;
43
44     while (scanf("%d %d", &m, &n) && m) { //
45         // 反覆輸入行數 m 和列數 n, 直到 m 為 0
46         num = 0;
47         for (i = 0; i < m; i++)
48             scanf("%s", map[i]); //
49         // 自上而下, 從左至右讀取每個網格
50
51         for (i = 0; i < m; i++)
52             for (j = 0; j < n; j++)
53                 if (map[i][j] == '@') { // 若 (i,
54                                         // j) 為油田
55                     num++; // 不同的油田數量 +1
56                     BFS(i, j); // 透過 BFS 將 (i, j)
57                                 // 可達的所有油田設為無油狀態
58                 }
59
60         printf("%d\n", num); // 輸出油田數
61     }
62
63     return 0;
64 }

```

Graph 有向邊並查集, 速通性檢查, 樹判斷

```

1 #include <cstdio>
2 #include <memory>
3
4 const int MAX_SIZE = 105;
5 int parent[MAX_SIZE]; // 每個點的根節點
6 bool flag[MAX_SIZE]; // 標記每個點是否被取用
7
8 void make_set() { // 初始化
9     for (int x = 1; x < MAX_SIZE; x++) {
10         parent[x] = x;
11         flag[x] = false;
12     }
13 }

```



```

12     }
13 }
14
15 int find_set(int x) { // 尋找根節點，帶路徑壓縮
16     if (x != parent[x])
17         parent[x] = find_set(parent[x]);
18     return parent[x];
19 }
20
21 void union_set(int x, int y) { // 合併兩個節點的集合
22     if (x < 1 || x >= MAX_SIZE || y < 1 || y >=
        MAX_SIZE) return; // 加入範圍檢查
23     x = find_set(x);
24     y = find_set(y);
25     if (x != y)
26         parent[y] = x;
27 }
28
29 bool single_root(int n) { // 檢查是否只有一個根
30     int i = 1;
31     while (i <= n && !flag[i]) i++;
32     if (i > n) return true; //
        如果範圍內沒有使用的節點
33     int root = find_set(i);
34     while (i <= n) {
35         if (flag[i] && find_set(i) != root)
36             return false;
37         ++i;
38     }
39     return true;
40 }
41
42 int main() {
43     int x, y;
44     bool is_tree = true;
45     int range = 0;
46     int idx = 1;
47     make_set();
48
49     while (scanf("%d %d", &x, &y) != EOF) {
50         if (x < 0 && y < 0)
51             break;
52         if (x == 0 && y == 0) {
53             if (is_tree && single_root(range))
54                 printf("Case %d is a tree.\n",
                    idx++);
55             else
56                 printf("Case %d is not a
                    tree.\n", idx++);
57
58             is_tree = true;
59             range = 0;
60             make_set();
61             continue;
62         }
63
64         if (x >= MAX_SIZE || y >= MAX_SIZE) { //
            檢查 x 和 y 是否在範圍內
65             is_tree = false;

```

```

66         continue;
67     }
68
69     range = x > range ? x : range;
70     range = y > range ? y : range;
71     flag[x] = flag[y] = true;
72
73     if (find_set(x) == find_set(y))
74         is_tree = false;
75     else
76         union_set(x, y);
77 }
78
79 return 0;
80 }

```

Graph BFS 計算圖連接性

```

1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 struct Position {
6     int i, j; // 網絡位置
7 } bfsQueue[10000]; // BFS 的佇列，重新命名為 bfsQueue
        避免與標準庫衝突
8
9 int m, n; // 網絡的行數 m 和列數 n
10 char map[101][101]; // 相鄰矩陣，'*' 表示牆，'0'
        表示油田
11 int a[8][2] = {{-1, 0}, {1, 0}, {0, -1}, {0,
        1}, {-1, -1}, {-1, 1}, {1, -1}, {1, 1}};
        // 8 個方向的移動
12
13 void BFS(int i, int j) {
14     int front = 0, rear = 1; // 佇列的首尾標誌初始化
15     bfsQueue[front].i = i;
16     bfsQueue[front].j = j;
17     map[i][j] = '*'; // 將起點設為無油狀態
18
19     while (front != rear) {
20         int ii = bfsQueue[front].i;
21         int jj = bfsQueue[front].j;
22         front++; // 佇列首指標 +1
23
24         for (int k = 0; k < 8; k++) { // 8
            個相鄰方向
25             int t1 = ii + a[k][0];
26             int t2 = jj + a[k][1];
27
28             if (map[t1][t2] == '0') { // 若 (t1,
                t2) 是油田
29                 bfsQueue[rear].i = t1;
30                 bfsQueue[rear].j = t2;
31                 map[t1][t2] = '*'; // 將 (t1, t2)
                    設為無油狀態
32                 rear++; // 佇列尾指標 +1

```



```
33     }
34 }
35 }
36 }
37
38 int main() {
39     int i, j;
40     int num;
41
42     while (scanf("%d %d", &m, &n) && m) { //
43         // 反覆輸入行數 m 和列數 n, 直到 m 為 0
44         num = 0;
45         for (i = 0; i < m; i++)
46             scanf("%s", map[i]); //
47         // 自上而下, 從左至右讀取每個網格
```

```
46
47     for (i = 0; i < m; i++)
48         for (j = 0; j < n; j++)
49             if (map[i][j] == '@') { // 若 (i,
50                 // j) 為油田
51                 num++; // 不同的油田數量 +1
52                 BFS(i, j); // 透過 BFS 將 (i, j)
53                 // 可達的所有油田設為無油狀態
54             }
55     printf("%d\n", num); // 輸出油田數
56 }
57 return 0;
58 }
```