

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 剪枝回溯, 狀態空間搜尋	12
18	Graph DFS 回溯找尋拓撲排序, 建相鄰有向邊	12
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	MST Kuskal 計算最小樹新增無向邊權和	16
23	MST prim 計算權和, 線性掃描最小邊, 密稠圖	16
24	SP Warshell 閉包遞移, 二分法計算最長邊最小路徑	17
25	SP Dijkstra 重邊判斷, 找最短路徑	18
26	SP Dijkstra 二分搜尋最佳初始值	18
27	SP SPFA 求負權最短路徑	20
28	BG HA (匈牙利算法) 二分圖最大匹配	20
29	BG 最大匹配數求邊覆蓋	21
30	BG 二分圖匹配最大化最小值	22
31	BG KM 求二分圖最小權和 (負邊)	23
32	Flow EK 求最大流	24
33	Flow SPFA 求最小費用流, 帶權二分圖轉網路圖	25

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; //
    字母表的大小 (假設只有小寫字母)

```

```

8 char str[MAXN][25]; // 儲存單詞的陣列
9 int ch[maxnode][sigma_size]; // Trie 樹的子節點指標
10 int val[maxnode]; // Trie 樹節點的取值次數
11
12 // 定義 Trie 結構
13 struct Trie {
14     int sz; // Trie 樹的節點數量
15
16     // 初始化 Trie 樹
17     Trie() { sz = 1; memset(ch[0], 0,
18         sizeof(ch[0])); } // 根節點初始化
19
20     // 將字母轉成數字索引
21     int idx(char c) { return c - 'a'; }
22
23     // 插入單詞到 Trie 樹
24     void insert(char *s) {
25         int u = 0, n = strlen(s); // 起始於根節點 u
26         = 0
27         for (int i = 0; i < n; i++) {
28             int c = idx(s[i]); // 計算字母的索引值
29             if (!ch[u][c]) { //
30                 若該節點不存在, 則創建新節點
31                 memset(ch[sz], 0,
32                     sizeof(ch[sz])); // 初始化新節點
33                 ch[u][c] = sz++; //
34                 設置子節點並增加節點數量
35             }
36             u = ch[u][c]; // 移動到下一個節點
37             val[u]++; // 計算到達該節點的次數
38         }
39     }
40
41     // 查詢單詞在 Trie 中的最短前綴
42     void query(char *s) {
43         int u = 0, n = strlen(s); // 起始於根節點 u
44         = 0
45         for (int i = 0; i < n; i++) {
46             putchar(s[i]); // 輸出當前字母
47             int c = idx(s[i]); // 計算字母的索引值
48             if (val[ch[u][c]] == 1) return; //
49             若當前子節點的次數為 1, 則找到最短前綴
50             u = ch[u][c]; // 移動到下一個節點
51         }
52     }
53 };
54
55 int main() {
56     int tot = 0; // 單詞數初始化
57     Trie trie; // 建立 Trie 的結構體變數
58
59     // 讀取每個單詞並插入到 Trie
60     while (scanf("%s", str[tot]) != EOF) {
61         trie.insert(str[tot]); // 插入單詞到 Trie
62         tot++; // 單詞數累加
63     }
64
65     // 查詢每個單詞的最短唯一前綴
66     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 // 節點結構體定義
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] =
            nullptr;
18     }
19 };
20 node *root;
21 char key[70];
22 char pattern[MAXN];
23 int N;
24
25 // 插入模式字串到 Trie 樹中
26 void Insert(char *s)
27 {
28     node *p = root; // 開始於 Trie 樹的根節點
29     for (int i = 0; s[i]; i++) {
30         int x = s[i] - 'a'; // 計算字元索引
31         if (p->next[x] == nullptr) {
32             p->next[x] = new node(); // 創建新節點
33         }
34         p = p->next[x]; // 移動到子節點
35     }
36     p->sum++; // 該節點匹配次數加 1
37 }
38
39 // 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     else {
88         //
89         // 若當前節點缺少某字母的邊, 將其指向失敗指針的相應子節點
90         current->next[i] =
91         current->fail->next[i];
92     }
93 }
94 }
95
96 // 在目標字串中運行 AC 自動機, 進行多模式匹配
97 void ac_automation(char *ch) {
98     node *p = root; // 從根節點開始
99     int len = strlen(ch); // 目標字串的長度
100    for (int i = 0; i < len; i++) {
101        int x = ch[i] - 'a'; // 當前字元索引
102        while (p->next[x] == root && p != root)
103            p = p->fail;
104        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
140 5
141 she
142 he
143 say
144 shr
145 her
146 yasherhs

```

```

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

```

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
10     // 是左子樹和右子樹的範圍
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
27         return p[a].t > p[b].t; // If pri equal,
            earlier t has higher priority
28     }
29 };
30
31 int main(void) {
32     char command[maxs];
33
34     // Define the priority queue with the custom comparator
35     priority_queue<int, vector<int>, Compare>
36         pq;
37
38     // Read commands until end of file
39     while (scanf("%s", command) != EOF) {

```

```

38     if (strcmp(command, "GET") == 0) { // GET
39         command
40         if (!pq.empty()) {
41             int top_idx = pq.top(); // Get the
42                 index of the highest priority element
43             pq.pop(); // Remove it from the queue
44             printf("%s %d\n",
45                 p[top_idx].name,
46                 p[top_idx].para);
47         } else {
48             printf("EMPTY QUEUE!\n");
49         }
50     } else { // Insert command
51         // Read the name, para, and pri for the new
52         // Info object
53         scanf("%s %d %d", p[used].name,
54             &p[used].para, &p[used].pri);
55         p[used].t = cnt++; // Assign and
56             increment the insertion order
57         pq.push(used++); // Push the index into
58             the priority queue
59     }
60 }
61
62 return 0;
63
64 /*
65 GET
66 PUT msg1 10 5
67 PUT msg2 10 4
68 GET
69 GET
70 GET
71 */

```

BST Treap 樹堆積左旋右旋, 插入刪除

```

1 #include <cstdio>
2 #include <cstdlib>
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
8         是客戶的編號, r 由 rand() 產生, 作為節點的優先順序
9
10     Node(int v, int info) : v(v), info(info) {
11         r = rand(); // 隨機產生節點優先順序
12         ch[0] = ch[1] = NULL; // 左右指標為空
13     }
14
15     int cmp(int x) { // 客戶優先順序 v 與 x 比較大小
16         if (v == x) return -1;
17         return x < v ? 0 : 1;
18     }
19 };

```

```

20 void rotate(Node *&o, int d) { // 節點 o 旋轉, 方向
21     d = 0 左旋, 1 右旋
22     Node *k = o->ch[d^1];
23     o->ch[d^1] = k->ch[d];
24     k->ch[d] = o;
25     o = k;
26 }
27
28 void insert(Node *&o, int v, int info) { //
29     插入一個節點 info, 優先順序為 v
30     if (o == NULL) {
31         o = new Node(v, info); //
32         若找到插入位置, 則客戶作為節點插入
33     }
34     else {
35         // **Corrected Insertion Direction**: Place higher
36         // 'v' to the right
37         int d = v < o->v ? 0 : 1;
38         insert(o->ch[d], v, info);
39         if (o->ch[d]->r > o->r) rotate(o, d^1);
40         // 若方向 d 的子樹優先順序較小, 則進行旋轉
41     }
42 }
43
44 void remove(Node *&o, int v) { // 在 o
45     為根的樹狀堆棧中, 刪除優先順序 v 的節點
46     if (!o) return;
47     int d = o->cmp(v);
48     if (d == -1) { // 如果找到該節點
49         Node *u = o;
50         if (o->ch[0] && o->ch[1]) { // 若 o
51             有左右子樹, 則計算被刪除節點的方向
52             int d2 = o->ch[0]->r < o->ch[1]->r ?
53                 1 : 0;
54             rotate(o, d2);
55             remove(o->ch[d2], v);
56         } else { // 若 o 節點僅有一個子樹, 則將其子樹取代 o
57             o = o->ch[0] ? o->ch[0] : o->ch[1];
58             delete u;
59         }
60     } else {
61         remove(o->ch[d], v); // 若 o
62         節點為葉節點, 直接將其刪除
63     }
64 }
65
66 int find_max(Node *o) { // 在 o
67     為根的樹狀堆棧中尋找最大優先順序
68     if (!o) return -1; // Handle empty treap
69     while (o->ch[1] != NULL) o = o->ch[1];
70     printf("%d\n", o->info);
71     return o->v;
72 }
73
74 int find_min(Node *o) { // 在 o
75     為根的樹狀堆棧中尋找最小優先順序
76     if (!o) return -1; // Handle empty treap
77     while (o->ch[0] != NULL) o = o->ch[0];
78     printf("%d\n", o->info);
79 }

```



```

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]);
23     // 將序列元素加入 q1
24
25     ll sum = 0;
26     while (1) {
27         ll tem = 0;
28         for (int i = 1; i <= x; i++) { // 每次取出
29             // x 個元素
30             if (q1.empty() && q2.empty()) break;
31             if (q1.empty()) {
32                 tem += q2.front();
33                 q2.pop();
34             } else if (q2.empty()) {
35                 tem += q1.front();
36                 q1.pop();
37             } else if (q1.front() < q2.front()) {
38                 tem += q1.front();
39                 q1.pop();
40             } else {
41                 tem += q2.front();
42                 q2.pop();
43             }
44         }
45         sum += tem;
46         if (q1.empty() && q2.empty()) break;
47         q2.push(tem);
48         if (sum > t) return 0;
49     }
50     return sum <= t;
```

```
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
58         sort(a + 1, a + 1 + n); //
59         // 排序以便進行霍夫曼合併
60
61         int st = 2, en = n;
62         while (st < en) { // 使用二分法找最小的 k 值
63             int mid = (st + en) / 2;
64             if (Huffman(mid)) en = mid;
65             else st = mid + 1;
66         }
67         printf("%d\n", st); // 輸出最小的 k 值
68     }
69     return 0;
70 }
```

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
5 // pow
6 using namespace std;
7
8 // Define the node struct with an explicit constructor
9 struct node {
10     int k, step;
11     node(int k_val, int step_val) : k(k_val),
12         step(step_val) {}
13 };
14
15 // Precompute powers of 10 for digit manipulation
16 const int power10_arr[4] = {1000, 100, 10, 1};
17
18 bool p[10000], s_array[10000]; // Renamed 's' to
19 // 's_array' to avoid confusion
20
21 // Sieve of Eratosthenes to generate prime numbers up to n
22 void make_sieve(int n) {
23     memset(p, 0, sizeof(p));
24     p[0] = 1; // 0 is not a prime
25     p[1] = 1; // 1 is not a prime
26     for (int i = 2; i <= n; i++) {
27         if (!p[i]) {
28             for (int j = i * i; j <= n; j += i)
29                 p[j] = 1; // Mark multiples of i as
30                 // non-prime
31         }
32     }
33 }
```

```

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-- > 0) {
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         memset(s_array, 0, sizeof(s_array));
61         s_array[x] = 1;
62         int ans = -1;
63
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++) { //
        依照節點順序訪問相鄰節點
29         if (map[x][y]) {

```

```

30     map[x][y] = map[y][x] = 0; //
        設定邊為已訪問
31     dfs(y, k + 1, s); // 遞迴搜索
32     map[x][y] = map[y][x] = 1; //
        恢復邊的狀態
33 }
34 }
35 }
36
37 int main() {
38     makemap(); // 產生無向圖的相鄰矩陣
39     dfs(1, 0, ""); // 從節點 1 開始計算所有可能的存取順序
40     return 0;
41 }

```

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; //
    木棍的數量, 木棍的目標長度, 所有木棍的總和
8
9 // 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
10 bool dfs(int i, int l, int t) {
11     int j;
12
13     // 若長度達到 len, 則成功構成一根木棍
14     if (l == 0) {
15         t -= len;
16         if (t == 0) return true;
17         for (i = 0; used[i]; ++i); //
            找到下一個未使用的木棍
18         used[i] = 1; // 標記該木棍為已使用
19         if (dfs(i + 1, len - sticks[i], t))
20             return true; // 遞迴切割
21         used[i] = 0;
22         return false;
23     } else {
24         for (int j = i; j < n; ++j) { //
            從長度遞減順序尋找木棍 j 到木棍 n - 1
25             if (j > 0 && sticks[j] == sticks[j - 1] && !used[j - 1]) continue; //
                若長度相同且木棍 j - 1 沒有被使用則跳過
26             if (!used[j] && sticks[j] <= l) { //
                若木棍沒有被使用且長度小於等於當前長度
27                 used[j] = 1;
28                 if (dfs(j + 1, l - sticks[j], t)) return true; // 遞迴嘗試
29                 used[j] = 0;
30                 if (sticks[j] == l) break; //
                    若木棍無法完成切割則跳過
31             }
32         }
33         return false;
34     }
35 }

```

```

33     }
34 }
35
36 // 木棍長度的比較函數
37 bool cmp(const int a, const int b) {
38     return a > b;
39 }
40
41 int main() {
42     while (cin >> n && n) { // 讀取木棍數量, 直至輸入
        0 為止
43         int sum = 0;
44         for (int i = 0; i < n; ++i) {
45             cin >> sticks[i];
46             sum += sticks[i]; // 計算木棍總長度
47             used[i] = 0; // 初始化使用標記
48         }
49         sort(sticks, sticks + n, cmp); //
            按木棍長度降序排列
50         bool flag = false;
51         for (len = sticks[0]; len <= sum / 2;
            ++len) { // 在 [sticks[0]..sum/2] 區間搜尋
52             if (sum % len == 0) { // 若總長度能被 len
                整除
53                 if (dfs(0, len, sum)) { // 若長度為
                    len 的木棍能夠切成 n 根木棍, 則標記成功
54                     flag = true;
55                     cout << len << endl; //
                        輸出木棍的最小可能長度並結束計算
56                     break;
57                 }
58             }
59         }
60         if (!flag) cout << sum << endl; //
            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 回溯找尋拓撲排序, 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,
16             visited))
17             return true;
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
29                                     // when N and K are both zero
30
31         // Initialize adjacency matrix and in-degree count
32         int g[100][100];
33         memset(g, 0, sizeof(g));
34         vector<int> in_degree(N, 0);
35
36         bool determined = false;
37         bool inconsistent = false;
38         string relation;
39
40         for (int i = 0; i < K; i++) {
41             cin >> relation;
42             if (relation.size() < 3 ||
43                 (relation[1] != '<' &&
44                  relation[1] != '>')) {
45                 // Handle invalid input format
46                 cout << "Invalid relation
47                     format." << endl;
48                 inconsistent = true;
49                 break;
50             }
51
52             char a = relation[0];
53             char b = relation[2];
54             char op = relation[1];
55             int x, y;
56
57             // Determine the direction of the relation
58             if (op == '<') {
59                 x = a - 'A';
60                 y = b - 'A';
61             }
62             else { // op == '>'
63                 x = b - 'A';
64                 y = a - 'A';
65             }
66
67             // Check if adding edge x -> y creates a cycle
68             bool visited[100];
69             memset(visited, false,
70                 sizeof(visited));
71             if (is_reachable(y, x, N, g,
72                 visited)) {

```

```

67         // Adding edge x->y would create a cycle
68         cout << "Inconsistency found
69             after " << (i + 1) << "
70             relations." << endl;
71
72         // Read and discard remaining relations
73         for this test case
74         for (int j = i + 1; j < K; j++) {
75             cin >> relation;
76         }
77
78         inconsistent = true;
79         break;
80     }
81
82     // Add the edge x -> y
83     g[x][y] = 1;
84     in_degree[y]++;
85
86     // Perform Topological Sort to check if a
87     unique sequence is determined
88     // Create a copy of in_degree to manipulate
89     vector<int> in_copy = in_degree;
90     queue<int> Q;
91     vector<int> result;
92
93     // Enqueue all nodes with in-degree 0
94     for (int node = 0; node < N; node++)
95     {
96         if (in_copy[node] == 0) {
97             Q.push(node);
98         }
99     }
100
101     bool multiple = false; // Flag to check
102                             // if multiple sequences are possible
103
104     while (!Q.empty()) {
105         if (Q.size() > 1) {
106             multiple = true; // More than one
107                             // node with in-degree 0 implies
108                             // multiple sequences
109         }
110
111         int current = Q.front();
112         Q.pop();
113         result.push_back(current);
114
115         // Decrease in-degree of neighboring nodes
116         for (int neighbor = 0; neighbor
117             < N; neighbor++) {
118             if (g[current][neighbor]) {
119                 in_copy[neighbor]--;
120                 if (in_copy[neighbor] ==
121                     0) {
122                     Q.push(neighbor);
123                 }
124             }
125         }

```

```

116     }
117
118     // Check if a unique sorted sequence is
119     // determined
120     if (result.size() == N && !multiple)
121     {
122         cout << "Sorted sequence
123             determined after " << (i +
124             1) << " relations: ";
125
126         // Replace range-based for loop with
127         // traditional for loop
128         for (size_t idx = 0; idx <
129             result.size(); idx++) {
130             int node = result[idx];
131             cout << char('A' + node);
132         }
133         cout << "." << endl;
134
135         // Read and discard remaining relations
136         // for this test case
137         for (int j = i + 1; j < K; j++) {
138             cin >> relation;
139         }
140
141         determined = true;
142         break;
143     }
144 }
145
146 // If no inconsistency or unique sequence was found
147 // after processing all relations
148 if (!inconsistent && !determined) {
149     cout << "Sorted sequence cannot be
150         determined." << endl;
151 }
152
153 return 0;
154 }

```

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);

```

```

15     if (x - 1 >= 0 && y < m && !vis[x - 1][y]
16         && map[x - 1][y]) dfs(x - 1, y);
17     if (x < n && y + 1 < m && !vis[x][y + 1] &&
18         map[x][y + 1]) dfs(x, y + 1);
19     if (x < n && y - 1 >= 0 && !vis[x][y - 1]
20         && map[x][y - 1]) dfs(x, y - 1);
21     if (x + 1 < n && y + 1 < m && !vis[x + 1][y
22         + 1] && map[x + 1][y + 1]) dfs(x + 1,
23         y + 1);
24     if (x - 1 >= 0 && y - 1 >= 0 && !vis[x -
25         1][y - 1] && map[x - 1][y - 1]) dfs(x
26         - 1, y - 1);
27     if (x + 1 < n && y - 1 >= 0 && !vis[x +
28         1][y - 1] && map[x + 1][y - 1]) dfs(x
29         + 1, y - 1);
30     if (x - 1 >= 0 && y + 1 < m && !vis[x -
31         1][y + 1] && map[x - 1][y + 1]) dfs(x
32         - 1, y + 1);
33 }
34
35 // Function to initialize visited array to zero
36 void init() {
37     memset(vis, 0, sizeof(vis));
38 }
39
40 int main() {
41     char ch;
42     while (cin >> n >> m) { // Read grid dimensions
43         if (n == 0 && m == 0) break;
44
45         init(); // Clear the visited markers
46
47         for (int i = 0; i < n; i++) {
48             for (int j = 0; j < m; j++) {
49                 cin >> ch; // Read each cell of the grid
50                 if (ch == '*')
51                     map[i][j] = 0; // Mark empty spot
52                 else
53                     map[i][j] = 1; // Mark oil spot
54             }
55         }
56
57         int count = 0; // Initialize oil deposit count
58
59         for (int i = 0; i < n; i++) {
60             for (int j = 0; j < m; j++) {
61                 if (!vis[i][j] && map[i][j]) {
62                     dfs(i, j); // Run DFS from each
63                     // unvisited oil spot
64                     count++; // Increment oil deposit
65                     // count
66                 }
67             }
68         }
69         cout << count << endl; // Output the count
70         // of different oil deposits
71     }
72     return 0;
73 }

```

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
    避免與標準庫衝突
8
9 int m, n; // 網格的行數 m 和列數 n
10 char map[101][101]; // 相鄰矩陣, '*' 表示牆, '@'
    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] == '@') { // 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) { //
        反覆輸入行數 m 和列數 n, 直到 m 為 0
43         num = 0;
44         for (i = 0; i < m; i++)
45             scanf("%s", map[i]); //
                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,
                    j) 為油田
50                     num++; // 不同的油田數量 +1
51                     BFS(i, j); // 透過 BFS 將 (i, j)
                        可達的所有油田設為無油狀態
52                 }
53
54             printf("%d\n", num); // 輸出油田數
55         }
56
57         return 0;
58     }

```

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 }
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",
55                     idx++);
56             else
57                 printf("Case %d is not a
58                     tree.\n", idx++);
59
60             is_tree = true;
61             range = 0;
62             make_set();
63             continue;
64         }
65         if (x >= MAX_SIZE || y >= MAX_SIZE) { //
66             檢查 x 和 y 是否在範圍內
67             is_tree = false;
68             continue;
69         }
70         range = x > range ? x : range;
71         range = y > range ? y : range;
72         flag[x] = flag[y] = true;
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 }

```

MST Kuskal 計算最小樹新增無向邊權和

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 vector<int> Fa;
6
7 int Get_father(int x) {
8     return Fa[x] == x ? x : Fa[x] =
9         Get_father(Fa[x]);
10 }
11 int main() {

```

```

12 int N, M;
13 while (cin >> N) { // read N (number of nodes)
14     vector<vector<int>> P(N + 1,
15         vector<int>(N + 1, 0));
16     Fa.resize(N + 1);
17
18     for (int i = 0; i < N; i++) { // input
19         adjacency matrix
20         for (int j = 0; j < N; j++) {
21             cin >> P[i][j];
22         }
23
24     for (int i = 0; i <= N; i++) Fa[i] = i;
25
26     cin >> M;
27     while (M--) { // process M edges
28         int a, b;
29         cin >> a >> b;
30         Fa[Get_father(a - 1)] = Get_father(b
31             - 1);
32     }
33
34     int ans = 0;
35     for (int k = 1; k <= 1000; k++) {
36         for (int i = 0; i < N; i++) {
37             for (int j = 0; j < N; j++) {
38                 if (P[i][j] == k &&
39                     Get_father(i) !=
40                     Get_father(j)) {
41                     Fa[Get_father(i)] =
42                         Get_father(j);
43                     ans += k;
44                 }
45             }
46         }
47     }
48
49     cout << ans << "\n";
50 }
51 return 0;
52 }

```

MST prim 計算權和, 線性掃描最小邊, 密碼圖

```

1 #include <iostream>
2 #include <vector>
3 #include <climits>
4 using namespace std;
5
6 int min(int i, int j) { // Return the index with the
7     minimum value
8     return i < j ? i : j;
9 }
10 int main() {
11     int n;
12     while (cin >> n) {

```



```

13     int tot = 0;
14     vector<vector<int>> v(n,
        vector<int>(n)); // Adjacency matrix for
        the graph
15     vector<int> dist(n, INT_MAX); // Distance
        array, initialized to maximum
16     vector<bool> use(n, false); // Boolean
        array to mark visited nodes
17
18     // Reading the adjacency matrix
19     for (int i = 0; i < n; i++) {
20         for (int j = 0; j < n; j++) {
21             cin >> v[i][j];
22         }
23     }
24
25     dist[0] = 0; // Starting node has distance 0
26
27     // Prim's algorithm to find MST
28     for (int i = 1; i < n; i++) {
29         dist[i] = v[0][i]; // Initialize the
        distance from the starting node
30     }
31
32     for (int i = 1; i < n; i++) { // Expand
        the MST with n - 1 edges
33         int tmp = -1;
34         for (int k = 1; k < n; k++) { // Find
        the minimum edge weight to add to MST
35             if (!use[k] && (tmp == -1 ||
                dist[k] < dist[tmp])) {
36                 tmp = k;
37             }
38         }
39
40         use[tmp] = true; // Mark the node as part
        of MST
41         tot += dist[tmp]; // Add the minimum edge
        weight to the total weight
42
43         // Update distances to nodes outside the MST
44         for (int k = 1; k < n; k++) {
45             if (!use[k]) {
46                 dist[k] = min(dist[k],
                    v[k][tmp]);
47             }
48         }
49     }
50
51     cout << tot << endl; // Output the total
        weight of the MST
52 }
53
54 return 0;
55 }

```

SP Warshell 閉包遞移, 二分法計算最長邊最小路徑

```

1  #include <iostream>
2  #include <cmath>
3  #include <iomanip>
4  #include <vector>
5
6  using namespace std;
7
8  const int MAX_N = (1 << 9) + 1;
9  double L[MAX_N][MAX_N]; // Distance matrix
10 bool con[MAX_N][MAX_N]; // Connection matrix
11
12 int main() {
13     int N, testCase = 0;
14     while (cin >> N && N != 0) {
15         vector<double> x(N), y(N);
16
17         // Read coordinates of each stone
18         for (int i = 0; i < N; ++i) {
19             cin >> x[i] >> y[i];
20         }
21
22         // Calculate the distance matrix L
23         for (int i = 0; i < N; ++i) {
24             for (int j = 0; j < N; ++j) {
25                 L[i][j] = sqrt(pow(x[i] - x[j],
                    2) + pow(y[i] - y[j], 2));
26             }
27         }
28
29         // Binary search for the minimum distance
30         double l = 0, r = 1e5;
31         while (r - l > 1e-5) {
32             double mid = (l + r) / 2;
33
34             // Initialize the connection matrix based on
        mid distance
35             for (int i = 0; i < N; ++i) {
36                 for (int j = 0; j < N; ++j) {
37                     con[i][j] = (L[i][j] <= mid);
38                 }
39             }
40
41             // Floyd-Warshall algorithm to determine
        reachability
42             for (int k = 0; k < N; ++k) {
43                 for (int i = 0; i < N; ++i) {
44                     for (int j = 0; j < N; ++j) {
45                         con[i][j] = con[i][j] ||
                            (con[i][k] &&
                                con[k][j]);
46                     }
47                 }
48             }
49
50             // Check if the first and second stones are
        connected
51             if (con[0][1]) {
52                 r = mid;
53             } else {

```

```

54         l = mid;
55     }
56 }
57
58 // Output the result with three decimal precision
59 cout << "Scenario #" << ++testCase <<
    endl;
60 cout << "Frog Distance = " << fixed <<
    setprecision(3) << l << endl;
61 cout << endl;
62 }
63 return 0;
64 }

```

SP Dijkstra 重邊判斷, 找最短路徑

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4
5 #define MAX_N 1010
6 #define MAX_M 2010
7 #define INF 1e9
8
9 using namespace std;
10
11 int w[MAX_N][MAX_N]; // Weight matrix for the graph
12 int d[MAX_N]; // Distance array for Dijkstra's algorithm
13 bool visited[MAX_N]; // Visited array for Dijkstra's
    algorithm
14 int n, m; // n = number of nodes, m = number of edges
15
16 void dijkstra(int s) { // Dijkstra's algorithm
    starting from node s
17     for (int i = 1; i <= n; ++i) {
18         d[i] = INF; // Initialize all distances to
            infinity
19         visited[i] = false; // Mark all nodes as
            unvisited
20     }
21     d[s] = 0; // Starting node distance is 0
22
23     for (int i = 1; i <= n; ++i) {
24         int x = -1;
25         for (int j = 1; j <= n; ++j) {
26             if (!visited[j] && (x == -1 || d[j]
                < d[x])) x = j;
27         }
28         visited[x] = true;
29
30         for (int j = 1; j <= n; ++j) {
31             if (!visited[j] && w[x][j] != INF) {
32                 d[j] = min(d[j], d[x] + w[x][j]);
33             }
34         }
35     }
36 }
37

```

```

38 int main() {
39     scanf("%d%d", &m, &n); // Input the number of
        edges and nodes
40     for (int i = 1; i <= n; ++i) {
41         for (int j = 1; j <= n; ++j) {
42             w[i][j] = INF; // Initialize weight matrix
                with INF
43         }
44     }
45
46     for (int i = 0; i < m; ++i) {
47         int a, b, c;
48         scanf("%d%d%d", &a, &b, &c); // Input edge
            endpoints and weight
49         if (w[a][b] > c) w[a][b] = w[b][a] = c;
            // Update to minimum weight for undirected
            graph
50     }
51
52     dijkstra(1); // Run Dijkstra's algorithm starting
        from node 1
53
54     printf("%d\n", d[n]); // Output the shortest path
        distance to node n
55     return 0;
56 }

```

SP Dijkstra 二分搜尋最佳初始值

```

1 ##include <bits/stdc++.h>
2 using namespace std;
3
4 // Function to convert character to index
5 int turn(char x){
6     if(x >= 'A' && x <= 'Z') return x - 'A' +
        1; // 'A'-'Z' -> 1-26
7     if(x >= 'a' && x <= 'z') return x - 'a' +
        27; // 'a'-'z' -> 27-52
8     return -1; // Invalid character
9 }
10
11 // Check function: Determines the maximum cargo that can
    reach 'to' from 'from' with starting cargo 'o'
12 int check(int from, int to, int o, bool
    go[][55]){
13     int g[55];
14     memset(g, 0, sizeof(g)); // Initialize cargo for
        each node to 0
15     bool flag[55];
16     memset(flag, false, sizeof(flag)); //
        Initialize visit flags to false
17     g[from] = o; // Set starting cargo at 'from' node
18
19     while(true){
20         int w = 0, next = -1;
21         // Find the unflagged node with the highest cargo
22         for(int i = 1; i <= 52; i++){
23             if(!flag[i] && g[i] > w){

```

```

24         next = i;
25         w = g[i];
26     }
27 }
28
29 if(next == -1) break; // No more nodes to
    process
30 flag[next] = true; // Mark the node as
    processed
31
32 // Update cargo for connected nodes
33 for(int i = 1; i <= 52; i++){
34     if(go[next][i]){
35         int reduction;
36         if(i < 27){
37             reduction = (w + 19) / 20; //
                Equivalent to ceil(w / 20)
38         }
39         else{
40             reduction = 1;
41         }
42         int tmp = w - reduction;
43         if(tmp > g[i]){
44             g[i] = tmp; // Update cargo if the
                new value is higher
45         }
46     }
47 }
48 }
49
50 return g[to]; // Return the cargo at the
    destination node
51 }
52
53 int main(){
54     ios::sync_with_stdio(false);
55     cin.tie(0); // Fast I/O
56
57     int T; // Number of connections
58     int tot = 0; // Test case counter
59
60     while(cin >> T){
61         if(T == -1) break; // Termination condition
62
63         // Initialize adjacency matrix
64         bool go[55][55];
65         memset(go, false, sizeof(go));
66
67         // Read T connections
68         for(int i = 0; i < T; i++){
69             char x, y;
70             cin >> x >> y;
71             int a = turn(x);
72             int b = turn(y);
73             if(a == -1 || b == -1){
74                 // Invalid characters, skip this connection
75                 continue;
76             }

```

```

77         go[a][b] = go[b][a] = true; //
                Bidirectional connection
78     }
79
80     // Read Tot (required cargo)
81     int Tot;
82     cin >> Tot;
83
84     // Read source and destination characters
85     char fromChar, toChar;
86     cin >> fromChar >> toChar;
87     int from = turn(fromChar);
88     int to = turn(toChar);
89
90     if(from == -1 || to == -1){
91         // Invalid source or destination nodes
92         tot++;
93         cout << "Case " << tot << ":
                Impossible\n";
94         continue;
95     }
96
97     // Binary search to find the minimal starting cargo
    'o' such that check(from, to, o) >= Tot
98     int l = 1, r = (1 << 20); // Search range:
    1 to 1048576
99     while(l < r){
100         int mid = (l + r - 1) / 2; // Midpoint
                calculation
101         int cargo = check(from, to, mid, go);
102         if(cargo >= Tot){
103             r = mid; // Possible answer found,
                search lower half
104         }
105         else{
106             l = mid + 1; // Need a higher starting
                cargo, search upper half
107         }
108     }
109
110     // After binary search, 'l' should be the minimal
    'o' such that check(from, to, l) >= Tot
111     // Verify the result
112     int finalCargo = check(from, to, l, go);
113     if(finalCargo >= Tot){
114         tot++;
115         cout << "Case " << tot << ": " << l
                << "\n";
116     }
117     else{
118         // If even the maximum cargo doesn't meet the
                requirement
119         tot++;
120         cout << "Case " << tot << ":
                Impossible\n";
121     }
122 }
123
124 return 0;

```

```

125 }
126
127 /*
128 1
129 a Z
130 19 a Z
131 5
132 A D
133 D X
134 A b
135 b c
136 c X
137 39 A X
138 -1
139 */

```

SP SPFA 求負權最短路徑

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 #include <queue>
5 using namespace std;
6
7 const int MAX = 501;
8 const int INF = 1e9; // Define a large constant for
    infinity
9 int map[MAX][MAX]; // Adjacency matrix to store graph
    weights
10 int dis[MAX]; // Distance array
11 int n, m, w, s, e, t; // Variables for nodes, edges,
    and endpoints
12 bool spfa(); // Function declaration for SPFA algorithm
13
14 // SPFA function to detect if there is a negative cycle in
    the graph
15 bool spfa() {
16     bool flag[MAX] = {0}; // Flags to mark nodes in
    the queue
17     int count[MAX] = {0}; // Counter for each node's
    occurrences in queue
18     queue<int> q;
19     q.push(s); // Start from source node
20     dis[s] = 0; // Distance to source is 0
21     int curr;
22     int i;
23
24     while (!q.empty()) {
25         int curr = q.front();
26         q.pop();
27
28         for (int i = 1; i <= n; i++) {
29             if (map[curr][i] < 100000)
30                 {
31                     if (dis[i] > map[curr][i] +
32                         dis[curr])

```

```

33             dis[i] = map[curr][i] +
34                 dis[curr];
35             if (flag[i] == 0)
36                 q.push(i);
37             count[i]++;
38             flag[i] = 1;
39             if (count[i] >= n)
40                 return 0;
41         }
42     }
43     flag[curr] = 0;
44 }
45 return true;
46 }
47
48 int main() {
49     int f;
50     scanf("%d", &f);
51     while (f--) {
52         // Reset arrays between test cases
53         memset(dis, 63, sizeof(dis)); //
54         // Initialize distance array to a large number
55         memset(map, 127, sizeof(map));
56
57         scanf("%d %d %d", &n, &m, &w); // Read
58         // number of nodes, edges, and start/end points
59         int i;
60         for (i = 0; i < m; i++) {
61             scanf("%d %d %d", &s, &e, &t);
62             // Update to the smallest weight
63             // if multiple edges exist
64             map[s][e] = map[s][e] > t? t:
65                 map[s][e];
66             map[e][s] = map[e][s] > t? t:
67                 map[e][s];
68         }
69
70         for (int i = 0; i < w; i++) {
71             scanf("%d %d %d", &s, &e, &t);
72             map[s][e] = -t; // Set the edge with
73             // negative weight for wormhole
74         }
75
76         if (spfa()) // Use SPFA to detect negative
77             // cycles
78             printf("NO\n");
79         else
80             printf("YES\n");
81     }
82     return 0;
83 }

```

BG HA(匈牙利算法) 二分圖最大匹配

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>

```

```

4 #include <queue>
5 using namespace std;
6
7 const int MAX = 501;
8 const int INF = 1e9; // Define a large constant for
    infinity
9 int map[MAX][MAX]; // Adjacency matrix to store graph
    weights
10 int dis[MAX]; // Distance array
11 int n, m, w, s, e, t; // Variables for nodes, edges,
    and endpoints
12 bool spfa(); // Function declaration for SPFA algorithm
13
14 // SPFA function to detect if there is a negative cycle in
    the graph
15 bool spfa() {
16     bool flag[MAX] = {0}; // Flags to mark nodes in
    the queue
17     int count[MAX] = {0}; // Counter for each node's
    occurrences in queue
18     queue<int> q;
19     q.push(s); // Start from source node
20     dis[s] = 0; // Distance to source is 0
21     int curr;
22     int i;
23
24     while (!q.empty()) {
25         int curr = q.front();
26         q.pop();
27
28         for (int i = 1; i <= n; i++) {
29             if(map[curr][i]<100000)
30             {
31                 if(dis[i] > map[curr][i] +
                    dis[curr])
32                 {
33                     dis[i] = map[curr][i] +
                        dis[curr];
34                     if(flag[i] == 0)
35                         q.push(i);
36                     count[i]++;
37                     flag[i] = 1;
38                     if(count[i]>=n)
39                         return 0;
40                 }
41             }
42         }
43         flag[curr] = 0;
44     }
45     return true;
46 }
47
48 int main() {
49     int f;
50     scanf("%d", &f);
51     while (f--) {
52         // Reset arrays between test cases
53         memset(dis, 63, sizeof(dis)); //
    Initialize distance array to a large number

```

```

54     memset(map, 127, sizeof(map));
55
56     scanf("%d %d %d", &n, &m, &w); // Read
    number of nodes, edges, and start/end points
57     int i;
58     for (i = 0; i < m; i++) {
59         scanf("%d %d %d", &s, &e, &t);
60         // Update to the smallest weight
    if multiple edges exist
61         map[s][e] = map[s][e]>t? t:
            map[s][e];
62         map[e][s] = map[e][s]>t? t:
            map[e][s];
63     }
64
65     for (int i = 0; i < w; i++) {
66         scanf("%d %d %d", &s, &e, &t);
67         map[s][e] = -t; // Set the edge with
    negative weight for wormhole
68     }
69
70     if (spfa()) // Use SPFA to detect negative
    cycles
71         printf("NO\n");
72     else
73         printf("YES\n");
74 }
75 return 0;
76 }

```

BG 最大匹配数求邊覆蓋

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 #include <queue>
5 using namespace std;
6
7 const int MAX = 501;
8 const int INF = 1e9; // Define a large constant for
    infinity
9 int map[MAX][MAX]; // Adjacency matrix to store graph
    weights
10 int dis[MAX]; // Distance array
11 int n, m, w, s, e, t; // Variables for nodes, edges,
    and endpoints
12 bool spfa(); // Function declaration for SPFA algorithm
13
14 // SPFA function to detect if there is a negative cycle in
    the graph
15 bool spfa() {
16     bool flag[MAX] = {0}; // Flags to mark nodes in
    the queue
17     int count[MAX] = {0}; // Counter for each node's
    occurrences in queue
18     queue<int> q;
19     q.push(s); // Start from source node
20     dis[s] = 0; // Distance to source is 0

```

```

21     int curr;
22     int i;
23
24     while (!q.empty()) {
25         int curr = q.front();
26         q.pop();
27
28         for (int i = 1; i <= n; i++) {
29             if(map[curr][i]<100000)
30             {
31                 if(dis[i] > map[curr][i] +
32                     dis[curr])
33                 {
34                     dis[i] = map[curr][i] +
35                         dis[curr];
36                     if(flag[i] == 0)
37                         q.push(i);
38                     count[i]++;
39                     flag[i] = 1;
40                     if(count[i]>=n)
41                         return 0;
42                 }
43             }
44             flag[curr] = 0;
45         }
46     }
47
48     int main() {
49         int f;
50         scanf("%d", &f);
51         while (f--) {
52             // Reset arrays between test cases
53             memset(dis, 63, sizeof(dis)); //
54             // Initialize distance array to a large number
55             memset(map, 127, sizeof(map));
56
57             scanf("%d %d %d", &n, &m, &w); // Read
58             // number of nodes, edges, and start/end points
59             int i;
60             for (i = 0; i < m; i++) {
61                 scanf("%d %d %d", &s, &e, &t);
62                 // Update to the smallest weight
63                 // if multiple edges exist
64                 map[s][e] = map[s][e]>t? t:
65                     map[s][e];
66                 map[e][s] = map[e][s]>t? t:
67                     map[e][s];
68             }
69
70             for (int i = 0; i < w; i++) {
71                 scanf("%d %d %d", &s, &e, &t);
72                 map[s][e] = -t; // Set the edge with
73                 // negative weight for wormhole
74             }
75
76             if (spfa()) // Use SPFA to detect negative
77                 // cycles

```

```

71         printf("NO\n");
72     else
73         printf("YES\n");
74 }
75 return 0;
76 }

```

BG 二分圖匹配最大化最小值

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 #include <queue>
5 using namespace std;
6
7 const int MAX = 501;
8 const int INF = 1e9; // Define a large constant for
9 // infinity
10 int map[MAX][MAX]; // Adjacency matrix to store graph
11 // weights
12 int dis[MAX]; // Distance array
13 int n, m, w, s, e, t; // Variables for nodes, edges,
14 // and endpoints
15 bool spfa(); // Function declaration for SPFA algorithm
16
17 // SPFA function to detect if there is a negative cycle in
18 // the graph
19 bool spfa() {
20     bool flag[MAX] = {0}; // Flags to mark nodes in
21     // the queue
22     int count[MAX] = {0}; // Counter for each node's
23     // occurrences in queue
24     queue<int> q;
25     q.push(s); // Start from source node
26     dis[s] = 0; // Distance to source is 0
27     int curr;
28     int i;
29
30     while (!q.empty()) {
31         int curr = q.front();
32         q.pop();
33
34         for (int i = 1; i <= n; i++) {
35             if(map[curr][i]<100000)
36             {
37                 if(dis[i] > map[curr][i] +
38                     dis[curr])
39                 {
40                     dis[i] = map[curr][i] +
41                         dis[curr];
42                     if(flag[i] == 0)
43                         q.push(i);
44                     count[i]++;
45                     flag[i] = 1;
46                     if(count[i]>=n)
47                         return 0;
48                 }
49             }
50         }
51     }
52 }

```

```

42     }
43     flag[curr] = 0;
44 }
45 return true;
46 }
47
48 int main() {
49     int f;
50     scanf("%d", &f);
51     while (f--) {
52         // Reset arrays between test cases
53         memset(dis, 63, sizeof(dis)); //
54         // Initialize distance array to a large number
55         memset(map, 127, sizeof(map));
56
57         scanf("%d %d %d", &n, &m, &w); // Read
58         // number of nodes, edges, and start/end points
59         int i;
60         for (i = 0; i < m; i++) {
61             scanf("%d %d %d", &s, &e, &t);
62             // Update to the smallest weight
63             // if multiple edges exist
64             map[s][e] = map[s][e] > t ? t :
65                 map[s][e];
66             map[e][s] = map[e][s] > t ? t :
67                 map[e][s];
68         }
69
70         for (int i = 0; i < w; i++) {
71             scanf("%d %d %d", &s, &e, &t);
72             map[s][e] = -t; // Set the edge with
73             // negative weight for wormhole
74         }
75
76         if (spfa()) // Use SPFA to detect negative
77             // cycles
78             printf("NO\n");
79         else
80             printf("YES\n");
81     }
82     return 0;
83 }

```

BG KM 求二分圖最小權和 (負邊)

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 #include <queue>
5 using namespace std;
6
7 const int MAX = 501;
8 const int INF = 1e9; // Define a large constant for
9 // infinity
10 int map[MAX][MAX]; // Adjacency matrix to store graph
11 // weights
12 int dis[MAX]; // Distance array

```

```

11 int n, m, w, s, e, t; // Variables for nodes, edges,
12 // and endpoints
13 bool spfa(); // Function declaration for SPFA algorithm
14 // SPFA function to detect if there is a negative cycle in
15 // the graph
16 bool spfa() {
17     bool flag[MAX] = {0}; // Flags to mark nodes in
18     // the queue
19     int count[MAX] = {0}; // Counter for each node's
20     // occurrences in queue
21     queue<int> q;
22     q.push(s); // Start from source node
23     dis[s] = 0; // Distance to source is 0
24     int curr;
25     int i;
26
27     while (!q.empty()) {
28         int curr = q.front();
29         q.pop();
30
31         for (int i = 1; i <= n; i++) {
32             if (map[curr][i] < 1000000)
33             {
34                 if (dis[i] > map[curr][i] +
35                     dis[curr])
36                 {
37                     dis[i] = map[curr][i] +
38                         dis[curr];
39                     if (flag[i] == 0)
40                         q.push(i);
41                     count[i]++;
42                     flag[i] = 1;
43                     if (count[i] >= n)
44                         return 0;
45                 }
46             }
47         }
48         flag[curr] = 0;
49     }
50     return true;
51 }
52
53 int main() {
54     int f;
55     scanf("%d", &f);
56     while (f--) {
57         // Reset arrays between test cases
58         memset(dis, 63, sizeof(dis)); //
59         // Initialize distance array to a large number
60         memset(map, 127, sizeof(map));
61
62         scanf("%d %d %d", &n, &m, &w); // Read
63         // number of nodes, edges, and start/end points
64         int i;
65         for (i = 0; i < m; i++) {
66             scanf("%d %d %d", &s, &e, &t);
67             // Update to the smallest weight
68             // if multiple edges exist

```

```

61     map[s][e] = map[s][e]>t? t:
        map[s][e];
62     map[e][s] = map[e][s]>t? t:
        map[e][s];
63 }
64
65 for (int i = 0; i < w; i++) {
66     scanf("%d %d %d", &s, &e, &t);
67     map[s][e] = -t; // Set the edge with
        negative weight for wormhole
68 }
69
70 if (spfa()) // Use SPFA to detect negative
        cycles
71     printf("NO\n");
72 else
73     printf("YES\n");
74 }
75 return 0;
76 }

```

Flow EK 求最大流

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 #include <queue>
5 using namespace std;
6
7 const int MAX = 501;
8 const int INF = 1e9; // Define a large constant for
        infinity
9 int map[MAX][MAX]; // Adjacency matrix to store graph
        weights
10 int dis[MAX]; // Distance array
11 int n, m, w, s, e, t; // Variables for nodes, edges,
        and endpoints
12 bool spfa(); // Function declaration for SPFA algorithm
13
14 // SPFA function to detect if there is a negative cycle in
        the graph
15 bool spfa() {
16     bool flag[MAX] = {0}; // Flags to mark nodes in
        the queue
17     int count[MAX] = {0}; // Counter for each node's
        occurrences in queue
18     queue<int> q;
19     q.push(s); // Start from source node
20     dis[s] = 0; // Distance to source is 0
21     int curr;
22     int i;
23
24     while (!q.empty()) {
25         int curr = q.front();
26         q.pop();
27
28         for (int i = 1; i <= n; i++) {
29             if(map[curr][i]<100000)

```

```

30         {
31             if(dis[i] > map[curr][i] +
                dis[curr])
32             {
33                 dis[i] = map[curr][i] +
                    dis[curr];
34                 if(flag[i] == 0)
35                     q.push(i);
36                 count[i]++;
37                 flag[i] = 1;
38                 if(count[i]>=n)
39                     return 0;
40             }
41         }
42     }
43     flag[curr] = 0;
44 }
45 return true;
46 }
47
48 int main() {
49     int f;
50     scanf("%d", &f);
51     while (f--) {
52         // Reset arrays between test cases
53         memset(dis, 63, sizeof(dis)); //
            Initialize distance array to a large number
54         memset(map, 127, sizeof(map));
55
56         scanf("%d %d %d", &n, &m, &w); // Read
            number of nodes, edges, and start/end points
57         int i;
58         for (i = 0; i < m; i++) {
59             scanf("%d %d %d", &s, &e, &t);
60             // Update to the smallest weight
                if multiple edges exist
61             map[s][e] = map[s][e]>t? t:
                map[s][e];
62             map[e][s] = map[e][s]>t? t:
                map[e][s];
63         }
64
65         for (int i = 0; i < w; i++) {
66             scanf("%d %d %d", &s, &e, &t);
67             map[s][e] = -t; // Set the edge with
                negative weight for wormhole
68         }
69
70         if (spfa()) // Use SPFA to detect negative
            cycles
71             printf("NO\n");
72         else
73             printf("YES\n");
74     }
75     return 0;
76 }

```


Flow SPFA 求最小費用流, 帶權二分圖轉網路圖

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 #include <queue>
5 using namespace std;
6
7 const int MAX = 501;
8 const int INF = 1e9; // Define a large constant for
    infinity
9 int map[MAX][MAX]; // Adjacency matrix to store graph
    weights
10 int dis[MAX]; // Distance array
11 int n, m, w, s, e, t; // Variables for nodes, edges,
    and endpoints
12 bool spfa(); // Function declaration for SPFA algorithm
13
14 // SPFA function to detect if there is a negative cycle in
    the graph
15 bool spfa() {
16     bool flag[MAX] = {0}; // Flags to mark nodes in
    the queue
17     int count[MAX] = {0}; // Counter for each node's
    occurrences in queue
18     queue<int> q;
19     q.push(s); // Start from source node
20     dis[s] = 0; // Distance to source is 0
21     int curr;
22     int i;
23
24     while (!q.empty()) {
25         int curr = q.front();
26         q.pop();
27
28         for (int i = 1; i <= n; i++) {
29             if(map[curr][i]<100000)
30             {
31                 if(dis[i] > map[curr][i] +
                    dis[curr])
32                 {
33                     dis[i] = map[curr][i] +
                        dis[curr];
34                     if(flag[i] == 0)
35                         q.push(i);
36                     count[i]++;
37                     flag[i] = 1;

```

```

38                 if(count[i]>=n)
39                     return 0;
40             }
41         }
42     }
43     flag[curr] = 0;
44 }
45 return true;
46 }
47
48 int main() {
49     int f;
50     scanf("%d", &f);
51     while (f--) {
52         // Reset arrays between test cases
53         memset(dis, 63, sizeof(dis)); //
    Initialize distance array to a large number
54         memset(map, 127, sizeof(map));
55
56         scanf("%d %d %d", &n, &m, &w); // Read
    number of nodes, edges, and start/end points
57         int i;
58         for (i = 0; i < m; i++) {
59             scanf("%d %d %d", &s, &e, &t);
60             // Update to the smallest weight
    if multiple edges exist
61             map[s][e] = map[s][e]>t? t:
                map[s][e];
62             map[e][s] = map[e][s]>t? t:
                map[e][s];
63         }
64
65         for (int i = 0; i < w; i++) {
66             scanf("%d %d %d", &s, &e, &t);
67             map[s][e] = -t; // Set the edge with
    negative weight for wormhole
68         }
69
70         if (spfa()) // Use SPFA to detect negative
    cycles
71             printf("NO\n");
72         else
73             printf("YES\n");
74     }
75     return 0;
76 }

```