

Contents

1	Tree Vector 建樹, DFS 先序尋訪, 找尋最近 LCA 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	3
4	Trie 建樹, 修改, 查詢	4
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	6
7	BST Stack 後序轉二元樹	6
8	BST 前序加中序找出後序	7
9	BST 後序二分搜尋樹還原	7
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	8
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	10
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	11
16	Graph DFS 走訪, 建無向相鄰矩陣	12
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	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	16
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, 稠密圖	18
24	SP Warshell 閉包遞移, 二分法計算最長邊最小路徑	18
25	SP Dijkstra 重邊判斷, 找最短路徑	19
26	SP Dijkstra 二分搜尋最佳初始值	19
27	SP SPFA 求負權最短路徑	21
28	BG HA(匈牙利算法) 二分圖最大匹配	22
29	BG 最大匹配數求邊覆蓋	22
30	BG 二分圖匹配最大化最小值	23
31	BG KM 求二分圖最小權和 (負邊)	24
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, DFS 先序遍訪, 找尋最近 LCA 共同祖先

```

1 #include <iostream>
2 #include <vector>
3 #include <cstring>
4 using namespace std;
5
6 const int N = 10000; // 最大節點數
7 vector<int> tree[N]; // 用來儲存樹的鄰接表
8 int parent[N]; // 紀錄每個節點的父節點
9 int depth[N]; // 紀錄每個節點的深度
10
11 // 深度優先搜尋 (DFS) 計算每個節點的深度
12 void DFS(int node, int dep) {
13     depth[node] = dep; // 設置當前節點的深度
14     for (size_t i = 0; i < tree[node].size();
15         i++) { // 遍歷所有子節點
16         int child = tree[node][i];
17         DFS(child, dep + 1); // 遞迴處理子節點, 深度加 1
18     }
19 }
20
21 // 最近共同祖先 (LCA) 查找函數
22 int findLCA(int x, int y) {
23     // 讓深度較大的節點向上移動, 直到兩個節點在同一深度
24     while (x != y) {
25         if (depth[x] > depth[y]) {
26             x = parent[x]; // x 上移到其父節點
27         } else {
28             y = parent[y]; // y 上移到其父節點
29         }
30     }
31     return x; // 返回最近共同祖先
32 }
33
34 int main() {
35     int casenum, n, i, x, y;
36     scanf("%d", &casenum); // 輸入測試案例數量
37
38     for (int num = 0; num < casenum; num++) {
39         scanf("%d", &n); // 輸入節點數
40         for (i = 0; i < n; i++)
41             tree[i].clear(); // 清空鄰接表
42         memset(parent, -1, sizeof(parent)); // 初始化父節點為 -1
43
44         // 輸入 n-1 條邊來建樹
45         for (i = 0; i < n - 1; i++) {
46             scanf("%d %d", &x, &y);
47             x--; y--; // 將節點編號轉為 0 開始的索引
48             tree[x].push_back(y); // 將 y 加入 x 的子節點中
49             parent[y] = x; // 樹中 y 的父節點為 x
50         }
51
52         // 找到樹的根節點 (父節點為 -1 的節點)

```

```

51     int root = 0;
52     for (i = 0; i < n; i++) {
53         if (parent[i] == -1) {
54             root = i;
55             break;
56         }
57     }
58
59     DFS(root, 0); // 從根節點開始進行先序遍訪, 初始化深度
60
61     // 查詢最近共同祖先
62     scanf("%d %d", &x, &y);
63     x--; y--; // 將輸入的節點轉為 0 開始的索引
64     int lca = findLCA(x, y); // 找到最近共同祖先
65     printf("%d\n", lca + 1); // 輸出結果, 轉回 1 開始的編號
66 }
67
68 return 0;
69 }

```

Tree Disjoinset 並查集, 路徑壓縮

```

1 #include <iostream>
2 #include <vector>
3 #include <cstring>
4
5 const int maxn = 100000 + 5;
6
7 class DisjointSet {
8 public:
9     std::vector<int> parent;
10
11     // 初始化父節點為 -1
12     DisjointSet(int size) : parent(size, -1) {}
13
14     // 查找根節點
15     int find(int x) {
16         if (parent[x] < 0)
17             return x;
18         return parent[x] = find(parent[x]); // 路徑壓縮
19     }
20
21     // 合併兩個集合
22     void unionSets(int x, int y) {
23         int rootX = find(x);
24         int rootY = find(y);
25         if (rootX != rootY) {
26             parent[rootX] = rootY; // 將 rootX 合併到 rootY 中
27         }
28     }
29 };
30
31 int n, m;
32

```

```

1 #include <iostream>
2 #include <vector>
3 #include <cstring>
4
5 const int maxn = 100000 + 5;
6
7 class DisjointSet {
8 public:
9     std::vector<int> parent;
10
11     // 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023
```

```

17     return x;
18     return parent[x] = find(parent[x]); //
19     , /# Y; A±N x %±μ³s±μ" ` I
20 }
21 // ±N" °/X/X"
22 void unionSets(int x, int y) {
23     int rootX = find(x);
24     int rootY = find(y);
25     if (rootX != rootY) {
26         parent[rootX] = rootY; // ±N x °° s±μ"
27         y °°
28     }
29 }
30 };
31 int n, m;
32
33 int main() {
34     int loop;
35     scanf("%d", &loop); // ` , Ψ q
36     while (loop--) {
37         scanf("%d%d", &n, &m);
38
39         // ° l= id¶°; A±j±p~° 2 * n; A¥Ψ L Y
40         DisjointSet dsu(2 * n + 1);
41
42         for (int i = 0; i < m; i++) {
43             int a, b;
44             char s[5];
45             scanf("%s%d%d", s, &a, &b);
46
47             if (s[0] == 'A') { // ~d, OS_! P=00~L 0 L
48                 if (dsu.find(a) != dsu.find(b)
49                     && dsu.find(a) != dsu.find(b
50                     + n))
51                     printf("Not sure yet.\n");
52                 else if (dsu.find(a) ==
53                     dsu.find(b))
54                     printf("In the same gang.\n");
55                 else
56                     printf("In different
57                     gangs.\n");
58             } else { // % 0 a 0M b ~°%L 0~L
59                 if (dsu.find(a) != dsu.find(b +
60                     n)) {
61                     dsu.unionSets(a, b + n); // a
62                     °°%HH~0 b
63                     dsu.unionSets(b, a + n); // b
64                     °°%HH~0 a
65                 }
66             }
67         }
68     }
69     return 0;
70 }

```

Trie 建樹, 修改, 查詢

```

1 #include <cstdio>
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,
        sizeof(ch[0])); } // 根節點初始化
18
19     // 將字母轉成數字索引
20     int idx(char c) { return c - 'a'; }
21
22     // 插入單詞到 Trie 樹
23     void insert(char *s) {
24         int u = 0, n = strlen(s); // 起始於根節點 u
            = 0
25         for (int i = 0; i < n; i++) {
26             int c = idx(s[i]); // 計算字母的索引值
27             if (!ch[u][c]) { //
                若該節點不存在, 則創建新節點
28                 memset(ch[sz], 0,
                    sizeof(ch[sz])); // 初始化新節點
29                 ch[u][c] = sz++; //
                    設置子節點並增加節點數量
30             }
31             u = ch[u][c]; // 移動到下一個節點
32             val[u]++; // 計算到達該節點的次數
33         }
34     }
35
36     // 查詢單詞在 Trie 中的最短前綴
37     void query(char *s) {
38         int u = 0, n = strlen(s); // 起始於根節點 u
            = 0
39         for (int i = 0; i < n; i++) {
40             putchar(s[i]); // 輸出當前字母
41             int c = idx(s[i]); // 計算字母的索引值
42             if (val[ch[u][c]] == 1) return; //
                若當前子節點的次數為 1, 則找到最短前綴
43             u = ch[u][c]; // 移動到下一個節點
44         }
45     }
46 };
47
48 int main() {

```

```

49     int tot = 0; // 單詞數初始化
50     Trie trie; // 建立 Trie 的結構體變數
51
52     // 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             else {
87                 //
88                 // 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
89                 current->next[i] =
90                 current->fail->next[i];
91             }
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
105         p = p->next[x];
106         if (!p)
107             p = root;
108
109         node *temp = p;
110         while (temp != root) { //
111             // 往上沿失敗指針累計所有匹配結果
112             if (temp->sum >= 0) { // 如果是匹配節點
113                 cnt += temp->sum; // 累計匹配次數
114                 temp->sum = -1; // 設置為 -1
115                 // 以避免重複計算
116             }
117             else
118                 break;
119             temp = temp->fail; // 沿失敗指針往上跳轉
120         }
121     }
122 }
123
124 int main()
125 {
126     int T; // 測試案例數量
127     cin >> T;
128     while (T--)
129     {
130         // 建立根節點
131         root = new node();
132         // 讀取模式字串數量
133         cin >> N;
134         cin.ignore(); // 忽略換行符
135
136         for (int i = 1; i <= N; i++)
137         {
138             // 讀取模式字串
139             cin.getline(key, sizeof(key));
140             Insert(key); // 將模式字串插入到 Trie 樹中
141         }
142
143         // 讀取目標字串
144         cin.getline(pattern, sizeof(pattern));
145         cnt = 0;
146         build_fail_pointer(); // 建立失敗指針
147         ac_automation(pattern); // 使用 AC
148         // 自動機進行匹配
149         cout << cnt << "\n"; //
150         // 輸出匹配到的模式字串次數
151     }
152     return 0;
153 }

```

```

136 /*
137 輸入範例:
138 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 << ": " <<
            height1 << " => " << height2 <<
            endl;
25    }
26    return 0;
27 }

```

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     while (read_case()) solve_case();
39     return 0;
40 }
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++; //
15     // 找到分界點 i，使得左子樹的元素都小於 a[r]
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
9     // 插入二元搜尋樹
10     if (T == NULL) { // 若 T 為空，則找到插入位置
11         T = new binTreeNode(); //
12         // 申請記憶體，建構儲存 val 的葉節點
13         T->data = val;
14         T->lchild = T->rchild = NULL;
15     } else if (T->data > val) { // 若 val
16         // 小於根節點值，則沿左子樹方向尋找插入點
17         add(T->lchild, val);
18     } else { // 若 val
19         // 不小於根節點值，則沿右子樹方向尋找插入位置
20         add(T->rchild, val);
21     }
22 }
23
24 void preOrder(BT T, bool flag) { //
25     // 前序輸出樹的順序，參數 flag 為首節點標誌
26     if (T == NULL)
27         return;

```

```

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
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 (Hufman(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

```

```

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
27                     non-prime
28         }
29     }
30
31 // Function to change the digit at a specific position
32 int change_digit(int x, int pos, int
33     new_digit) {
34     int digits[4] = { x / 1000, (x / 100) % 10,
35         (x / 10) % 10, x % 10 };
36     digits[pos - 1] = new_digit;
37     return digits[0] * 1000 + digits[1] * 100 +
38         digits[2] * 10 + digits[3];
39 }
40
41 int main() {
42     // Optimize I/O operations
43     ios::sync_with_stdio(false);
44     cin.tie(NULL);
45
46     make_sieve(9999); // Generate primes up to 9999
47
48     int tot;
49     cin >> tot;
50     while (tot--) {
51         int x, y;
52         cin >> x >> y;
53
54         // Validate that both x and y are 4-digit primes
55         if (x < 1000 || x > 9999 || y < 1000 ||
56             y > 9999 || p[x] || p[y]) {
57             cout << "Impossible" << endl;
58             continue;
59         }
60
61         // Initialize the BFS queue and visited array
62         queue<node> q;
63         q.push(node(x, 0)); // Use constructor to
64             initialize node
65         memset(s_array, 0, sizeof(s_array));
66         s_array[x] = 1;
67         int ans = -1;
68
69         while (!q.empty()) {
70             node cur = q.front();
71             q.pop();
72
73             if (cur.k == y) {
74                 ans = cur.step;
75                 break;
76             }
77
78             for (int i = 1; i <= 4; i++) { //
79                 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

```

```

25     if (!used[j] && sticks[j] <= 1) { //
        若木棍沒有被使用且長度小於等於當前長度
26         used[j] = 1;
27         if (dfs(j + 1, 1 - sticks[j],
            t)) return true; // 遞迴嘗試
28         used[j] = 0;
29         if (sticks[j] == 1) break; //
            若木棍無法完成切割則跳過
30     }
31 }
32 return false;
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,
                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,
66        sizeof(visited));
67 if (is_reachable(y, x, N, g,
68                 visited)) {
69     // Adding edge x->y would create a cycle
70     cout << "Inconsistency found
71         after " << (i + 1) << "
72         relations." << endl;
73
74     // Read and discard remaining relations
75     for this test case
76     for (int j = i + 1; j < K; j++) {
77         cin >> relation;
78     }
79
80     inconsistent = true;
81     break;
82 }
83
84 // Add the edge x -> y
85 g[x][y] = 1;
86 in_degree[y]++;
87
88 // Perform Topological Sort to check if a
89 // unique sequence is determined
90 // Create a copy of in_degree to manipulate
91 vector<int> in_copy = in_degree;
92 queue<int> Q;
93 vector<int> result;
94
95 // Enqueue all nodes with in-degree 0
96 for (int node = 0; node < N; node++)
97 {
98     if (in_copy[node] == 0) {
99         Q.push(node);
100     }
101 }
102
103 bool multiple = false; // Flag to check
104 // if multiple sequences are possible
105
106 while (!Q.empty()) {
107     if (Q.size() > 1) {
108         multiple = true; // More than one
109         // node with in-degree 0 implies
110         // multiple sequences
111     }
112
113     int current = Q.front();
114     Q.pop();
115     result.push_back(current);
116
117     // Decrease in-degree of neighboring nodes

```

```

108     for (int neighbor = 0; neighbor
109         < N; neighbor++) {
110         if (g[current][neighbor]) {
111             in_copy[neighbor]--;
112             if (in_copy[neighbor] ==
113                 0) {
114                 Q.push(neighbor);
115             }
116         }
117     }
118
119     // Check if a unique sorted sequence is
120     // determined
121     if (result.size() == N && !multiple)
122     {
123         cout << "Sorted sequence
124             determined after " << (i +
125             1) << " relations: ";
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
                    // empty spots

```

```

6 int vis[105][105]; // Visited marker array
7 int n, m; // Rows and columns of the grid
8
9 // DFS function to explore connected components
10 void dfs(int x, int y) {
11     vis[x][y] = 1; // Mark current cell as visited
12
13     // Explore all 8 possible directions
14     if (x + 1 < n && y < m && !vis[x + 1][y] &&
15         map[x + 1][y]) dfs(x + 1, y);
16     if (x - 1 >= 0 && y < m && !vis[x - 1][y]
17         && map[x - 1][y]) dfs(x - 1, y);
18     if (x < n && y + 1 < m && !vis[x][y + 1] &&
19         map[x][y + 1]) dfs(x, y + 1);
20     if (x < n && y - 1 >= 0 && !vis[x][y - 1]
21         && map[x][y - 1]) dfs(x, y - 1);
22     if (x + 1 < n && y + 1 < m && !vis[x + 1][y
23         + 1] && map[x + 1][y + 1]) dfs(x + 1,
24         y + 1);
25     if (x - 1 >= 0 && y - 1 >= 0 && !vis[x -
26         1][y - 1] && map[x - 1][y - 1]) dfs(x
27         - 1, y - 1);
28     if (x + 1 < n && y - 1 >= 0 && !vis[x +
29         1][y - 1] && map[x + 1][y - 1]) dfs(x
30         + 1, y - 1);
31     if (x - 1 >= 0 && y + 1 < m && !vis[x -
32         1][y + 1] && map[x - 1][y + 1]) dfs(x
33         - 1, y + 1);
34 }
35
36 // Function to initialize visited array to zero
37 void init() {
38     memset(vis, 0, sizeof(vis));
39 }
40
41 int main() {
42     char ch;
43     while (cin >> n >> m) { // Read grid dimensions
44         if (n == 0 && m == 0) break;
45
46         init(); // Clear the visited markers
47
48         for (int i = 0; i < n; i++) {
49             for (int j = 0; j < m; j++) {
50                 cin >> ch; // Read each cell of the grid
51                 if (ch == '*')
52                     map[i][j] = 0; // Mark empty spot
53                 else
54                     map[i][j] = 1; // Mark oil spot
55             }
56         }
57
58         int count = 0; // Initialize oil deposit count
59
60         for (int i = 0; i < n; i++) {
61             for (int j = 0; j < m; j++) {
62                 if (!vis[i][j] && map[i][j]) {
63                     dfs(i, j); // Run DFS from each
64                     // unvisited oil spot
65                 }
66             }
67         }
68     }
69 }

```

```

52         count++; // Increment oil deposit
53         // count
54     }
55 }
56 cout << count << endl; // Output the count
57 // of different oil deposits
58 }
59 return 0;
60 }

```

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++) { // 8
29             // 個相鄰方向
30             int t1 = ii + a[k][0];
31             int t2 = jj + a[k][1];
32
33             if (map[t1][t2] == '@') { // 若 (t1,
34                 // t2) 是油田
35                 bfsQueue[rear].i = t1;
36                 bfsQueue[rear].j = t2;
37                 map[t1][t2] = '*'; // 將 (t1, t2)
38                 // 設為無油狀態
39                 rear++; // 佇列尾指標 +1
40             }
41         }
42     }
43 }
44
45 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 <stdio>
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",
    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 }

```

MST Kuskal 計算最小樹新增無向邊權和

```

1  #include <iostream>

```



```

2 #include <vector>
3 #include <algorithm>
4 using namespace std;
5
6 // μ²°c  ψ v«
7 struct Edge {
8     int u;
9     int v;
10    int weight;
11 };
12
13 // Union-Find (Disjoint Set Union - DSU)  $0
14 class UnionFind {
15 private:
16     vector<int> parent;
17 public:
18     // ° l° DSU_i A,` I% ° n
19     UnionFind(int n) : parent(n + 1) { //
20         // °²³],` I±q 1 ¶]°l
21         for(int i = 0; i <= n; ++i)
22             parent[i] = i;
23     }
24     // ~d$% /X°° A`°i /#Y
25     int find_set(int x) {
26         if(parent[x] != x)
27             parent[x] = find_set(parent[x]);
28         return parent[x];
29     }
30
31     // /X` °/X
32     void union_set(int x, int y) {
33         int fx = find_set(x);
34         int fy = find_set(y);
35         if(fx != fy)
36             parent[fx] = fy;
37     }
38 };
39
40 // ° °  °  A%Q
41 bool compare_edges(const Edge &a, const Edge
42     &b) {
43     return a.weight < b.weight;
44 }
45
46 // ° Kruskal °t° k`ē`°p%`° `v«
47 int kruskal(int N, vector<Edge> &edges,
48     UnionFind &uf) {
49     // « v«¹ i/ ā]±q²p` j_i`
50     sort(edges.begin(), edges.end(),
51         compare_edges);
52
53     int total_weight = 0;
54     for(int i = 0; i < edges.size(); ++i) {
55         Edge edge = edges[i];
56         // /p°G 類` ° I P°°¶°/X_i A«h% [=J³p%`°
57         if(uf.find_set(edge.u) !=
58             uf.find_set(edge.v)) {
59             uf.union_set(edge.u, edge.v);

```

```

56         total_weight += edge.weight;
57     }
58 }
59 return total_weight;
60 }
61
62 int main() {
63     ios::sync_with_stdio(false);
64     cin.tie(0); // ° 0 ¶N` nullptr
65
66     int N;
67     while(cin >> N) { // Ū` ,` I% N
68         // Ū` %F±μ`x°}
69         vector<vector<int>> P(N + 1,
70             vector<int>(N + 1, 0));
71         for(int i = 1; i <= N; ++i) { // 1-based
72             // indexing
73             for(int j = 1; j <= N; ++j) {
74                 cin >> P[i][j];
75             }
76         }
77
78         // ° l° Union-Find
79         UnionFind uf(N);
80
81         // Ū` °w, g³s±μ°° M` °X` °`
82         int M;
83         cin >> M;
84         for(int i = 0; i < M; ++i) {
85             int a, b;
86             cin >> a >> b;
87             uf.union_set(a, b);
88         }
89
90         // /~¶°° ° A K<°σ]μL/V` °
91         vector<Edge> edges;
92         for(int i = 1; i <= N; ++i) {
93             for(int j = i + 1; j <= N; ++j) { //
94                 // j ±q i+1 ¶]°l K«%
95                 if(P[i][j] > 0) { // °²³] 0 °S!³
96                     edges.push_back(Edge{ i, j,
97                         P[i][j] });
98                 }
99             }
100         }
101
102         // ° Kruskal °t° k°Hhp° p%`° `v«
103         int total_MST_weight = kruskal(N,
104             edges, uf);
105
106         // ° std::endl ¶HXT°O`C , °Y X
107         cout << total_MST_weight << endl;
108     }
109     return 0;

```

```

110 0 990 692
111 990 0 179
112 692 179 0
113 1
114 1 2
115
116 'w'-X:
117 179
118 */

```

MST prim 計算權和，線性掃描最小邊，密稠圖

```

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
    minimum value
7     return i < j ? i : j;
8 }
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' +
7         1; // 'A'-'Z' -> 1-26
8     if(x >= 'a' && x <= 'z') return x - 'a' +
9         27; // 'a'-'z' -> 27-52
10    return -1; // Invalid character
11 }
12 // Check function: Determines the maximum cargo that can
13 // reach 'to' from 'from' with starting cargo 'o'
14 int check(int from, int to, int o, bool
15     go[][55]){
16     int g[55];
17     memset(g, 0, sizeof(g)); // Initialize cargo for
18     // each node to 0
19     bool flag[55];
20     memset(flag, false, sizeof(flag)); //
21     // Initialize visit flags to false
22     g[from] = o; // Set starting cargo at 'from' node
23     while(true){
24         int w = 0, next = -1;
25         // Find the unflagged node with the highest cargo
26         for(int i = 1; i <= 52; i++){
27             if(!flag[i] && g[i] > w){
28                 next = i;
29                 w = g[i];
30             }
31         }
32         if(next == -1) break; // No more nodes to
33         // process
34         flag[next] = true; // Mark the node as
35         // processed
36         // Update cargo for connected nodes
37         for(int i = 1; i <= 52; i++){
38             if(go[next][i]){
39                 int reduction;
40                 if(i < 27){
41                     reduction = (w + 19) / 20; //
42                     // Equivalent to ceil(w / 20)
43                 }
44                 else{
45                     reduction = 1;
46                 }
47                 int tmp = w - reduction;
48                 if(tmp > g[i]){
49                     g[i] = tmp; // Update cargo if the
50                     // new value is higher
51                 }
52             }
53         }
54     }
55     return g[to]; // Return the cargo at the
56     // destination node
57 }

```

```

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; //
78             // Bidirectional connection
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 << ":
94                     // Impossible\n";
95                 continue;
96             }
97
98             // Binary search to find the minimal starting cargo
99             // 'o' such that check(from, to, o) >= Tot
100            int l = 1, r = (1 << 20); // Search range:
101            // 1 to 1048576
102            while(l < r){
103                int mid = (l + r - 1) / 2; // Midpoint
104                // calculation
105                int cargo = check(from, to, mid, go);
106                if(cargo >= Tot){
107                    r = mid; // Possible answer found,
108                    // search lower half
109                }
110            }
111        }
112    }

```

```

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] +
                    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 HA(匈牙利算法) 二分圖最大匹配

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4
5 using namespace std;
6
7 int a[110][310];
8 int n, m, vis[310], pre[310];
9
10 bool dfs(int x) {
11     int t;
12     for (t = 1; t <= m; ++t) {
13         if (a[x][t] && !vis[t]) { // if there is an
            edge and t is not visited
14             vis[t] = 1; // mark t as visited
15             if (pre[t] == 0 || dfs(pre[t])) { //
                if t has no previous match or previous
                match can find an alternate path
16                 pre[t] = x; // set (x, t) as a match
17                 return true; // matching success
18             }
19         }
20     }
21     return false; // matching failed
22 }
23
24 int main() {
25     int T, i, t, j, s;
26     scanf("%d", &T); // input number of test cases
27     while (T--) {
28         scanf("%d%d", &n, &m); // input number of
            courses and students
29         memset(a, 0, sizeof(a)); // initialize
            adjacency matrix for bipartite graph
30         memset(pre, 0, sizeof(pre)); // initialize
            matching storage array
31

```

```

32     for (i = 1; i <= n; ++i) {
33         scanf("%d", &t); // input number of
            students corresponding to each course
34         while (t--) {
35             scanf("%d", &j); // input each student
36             a[i][j] = 1; // mark relation between
                course i and student j as an edge
37         }
38     }
39
40     s = 0;
41     for (i = 1; i <= n; ++i) {
42         memset(vis, 0, sizeof(vis)); //
            initialize visit array for each course
43         if (dfs(i)) s++; // if course i can be
            matched, increment match count
44     }
45
46     if (s == n) printf("YES\n"); // if maximum
            match count equals course count, team can be
            formed
47     else printf("NO\n"); // otherwise, team
            formation fails
48 }
49 return 0;
50 }

```

BG 最大匹配數求邊覆蓋

```

1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 const int V = 1100;
6
7 int n, m, k, x, y, pre[V];
8 bool v[V], a[V][V];
9
10 bool dfs(int i) {
11     for (int j = 1; j <= m; j++) {
12         if (!v[j] && a[i][j]) { // check if j is
            unvisited and there is an edge between i and j
13             v[j] = 1; // mark j as visited
14             if (pre[j] == 0 || dfs(pre[j])) { //
                if j has no previous match or previous
                match can find an alternate path
15                 pre[j] = i; // set (i, j) as a match
16                 return 1; // matching success
17             }
18         }
19     }
20     return 0; // matching failed
21 }
22
23 int main() {
24     cin >> n >> m >> k; // input number of
            representatives from A and B, and the number of
            pairs

```

```

25  memset(a, 0, sizeof(a)); // initialize adjacency
    matrix for bipartite graph
26  memset(pre, 0, sizeof(pre)); // initialize
    matching storage array
27
28  for (int i = 1; i <= k; i++) {
29      cin >> x >> y; // input pair representing
        relationship between A's representative x and
        B's representative y
30      a[x][y] = 1; // mark the relationship as an edge
31  }
32
33  int ans = 0;
34  for (int i = 1; i <= n; i++) {
35      memset(v, 0, sizeof(v)); // initialize visit
        array for each representative
36      if (dfs(i)) ans++; // if representative i can
        be matched, increment match count
37  }
38
39  cout << n + m - ans << endl; // output minimum
    number of telephone lines required, N + M - max
    matching
40  return 0;
41 }

```

BG 二分图匹配最大化最小值

```

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
    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 KM 求二分圖最小權和 (負邊)

```

1  #include <iostream>
2  #include <algorithm>
3  #include <cstring>
4  #include <climits>
5
6  using namespace std;
7
8  #define MAXN 102
9  #define max(x, y) ((x) > (y) ? (x) : (y))
10
11 int n, m, a[MAXN][MAXN], lx[MAXN], ly[MAXN],
    slack[MAXN], maty[MAXN];
12 int lenx, leny;
13 bool vx[MAXN], vy[MAXN];
14 char map[MAXN][MAXN];
15
16 bool search(int u) {
17
18     int i, t;
19     vx[u] = 1;
20     for (int i = 0; i < leny; ++i) {
21         if (!vy[i]) {
22             int t = lx[u] + ly[i] - a[u][i];
23             if (t == 0) {
24                 vy[i] = 1;
25                 if (maty[i] == -1 ||
26                     search(maty[i])) {
27                     maty[i] = u;
28                     return 1;
29                 }
30             } else if (slack[i] > t) {
31                 slack[i] = t;
32             }
33         }
34     }
35     return 0;
36 }
37
38 int KM() {
39     int i, j, ans = 0;
40     for (i = 0; i < lenx; ++i) {
41         lx[i] = -INT_MAX;
42         for (j = 0; j < leny; ++j) {
43             lx[i] = max(lx[i], a[i][j]);
44         }
45     }
46     memset(maty, -1, sizeof(maty));
47     memset(ly, 0, sizeof(ly));
48     for (i = 0; i < lenx; ++i) {
49         fill(slack, slack + leny, INT_MAX);
50         while (1) {
51             memset(vx, 0, sizeof(vx));

```

```

52             memset(vy, 0, sizeof(vy));
53             if (search(i)) break;
54             int d = INT_MAX;
55             for (j = 0; j < leny; ++j) {
56                 if (!vy[j] && d > slack[j]) d =
                    slack[j];
57             }
58             for (j = 0; j < lenx; ++j) {
59                 if (vx[j]) lx[j] -= d;
60             }
61             for (j = 0; j < leny; ++j) {
62                 if (vy[j]) ly[j] += d;
63             }
64         }
65     }
66     for (i = 0; i < leny; ++i) {
67         if (maty[i] != -1) ans += a[maty[i]][i];
68     }
69     return -ans;
70 }
71
72 int main() {
73     int i, j;
74     while (~scanf("%d%d", &n, &m) && n + m) {
75         lenx = leny = 0;
76         for (i = 0; i < n; ++i) {
77             scanf("%s", map[i]);
78             for (j = 0; j < m; ++j) {
79                 if (map[i][j] == 'H') {
80                     lx[lenx] = i;
81                     slack[lenx++] = j;
82                 } else if (map[i][j] == 'm') {
83                     ly[leny] = i;
84                     maty[leny++] = j;
85                 }
86             }
87         }
88         for (i = 0; i < lenx; ++i) {
89             for (j = 0; j < leny; ++j) {
90                 a[i][j] = -abs(lx[i] - ly[j]) -
                    abs(slack[i] - maty[j]);
91             }
92         }
93         printf("%d\n", KM());
94     }
95     return 0;
96 }

```

Flow EK 求最大流

```

1  #include <stdio.h>
2  #include <math.h>
3  #include <memory.h>
4
5  int n, np, nc, m, s, t;
6  int fa[104], q[104], f[104][104], c[104][104];
7  //
8  // fa[]儲存路徑, q[]為佇列, f[i][j]記錄流量, c[i][j]記錄容量

```



```

7
8 void Edmonds_Karp() { // Edmonds-Karp演算法求最大流
9     int qs, qt, d, doo, i, j, ans = 0;
10    fa[t] = 1;
11    while (fa[t] != 0) { // 若增廣路徑存在
12        qs = 0; qt = 1;
13        q[qt] = s;
14        memset(fa, 0, sizeof(fa)); // 初始化增廣路徑
15        fa[s] = s;
16        while (qs < qt && fa[t] == 0) {
17            i = q[++qs];
18            for (j = 1; j <= t; j++) {
19                if (fa[j] == 0) {
20                    if (f[i][j] < c[i][j]) { //
21                        若(i, j)的流量可增加
22                        fa[j] = i;
23                        q[++qt] = j;
24                    }
25                    else if (f[j][i] > 0) { //
26                        若(i, j)的反向流量可減少
27                        fa[j] = -i;
28                        q[++qt] = j;
29                    }
30                }
31            }
32            if (fa[t] != 0) { // 如果找到增廣路徑
33                doo = 1000000000;
34                i = t;
35                while (i != s) { //
36                    從終點逆向尋找增廣路徑中最小容量
37                    if (fa[i] > 0) {
38                        if ((d = c[fa[i]][i] -
39                             f[fa[i]][i]) < doo)
40                            doo = d;
41                    }
42                    else {
43                        if (f[i][-fa[i]] < doo)
44                            doo = f[i][-fa[i]];
45                    }
46                    i = abs(fa[i]);
47                }
48                ans += doo;
49                i = t;
50                while (i != s) { // 增加增廣路徑上流量
51                    if (fa[i] > 0)
52                        f[fa[i]][i] += doo;
53                    else
54                        f[i][-fa[i]] -= doo;
55                    i = abs(fa[i]);
56                }
57            }
58            printf("%d\n", ans); // 輸出最大流
59        }
60    }
61    int main() {
62        int i, u, v, cc;
63        while (scanf("%d%d%d%d", &n, &np, &nc, &m)
64                == 4) { // 輸入節點數、源點數、匯點數和邊數

```

```

61        s = n + 2; t = n + 1;
62        memset(f, 0, sizeof(f));
63        memset(c, 0, sizeof(c));
64        for (i = 1; i <= m; i++) { // 讀取邊的資訊
65            while (getchar() != '(');
66            scanf("%d,%d%d", &u, &v, &cc);
67            c[u + 1][v + 1] = cc;
68        }
69        for (i = 1; i <= np; i++) { //
70            讀取每個源點的流量容量
71            while (getchar() != '(');
72            scanf("%d)%d", &u, &cc);
73            c[s][u + 1] = cc;
74        }
75        for (i = 1; i <= nc; i++) { //
76            讀取每個匯點的流量容量
77            while (getchar() != '(');
78            scanf("%d)%d", &u, &cc);
79            c[u + 1][t] = cc;
80        }
81        Edmonds_Karp(); // 執行 Edmonds-Karp
82        演算法求最大流
83    }
84    return 0;
85 }

```

Flow SPFA 求最小費用流, 帶權二分圖轉網路圖

```

1 #include <iostream>
2 #include <vector>
3 #include <queue>
4 #include <cstring>
5 #include <climits>
6 using namespace std;
7
8 const int MAXN = 505;
9 const int INF = INT_MAX;
10
11 struct Edge {
12     int to, cap, cost;
13     Edge(int _to, int _cap, int _cost) :
14         to(_to), cap(_cap), cost(_cost) {}
15 };
16
17 class MinCostMaxFlow {
18 private:
19     int n, s, t;
20     vector<Edge> edges;
21     vector<vector<int>>> adj;
22     vector<int> dist, prev, prevEdge;
23     vector<bool> inQueue;
24
25     bool SPFA() {
26         dist.assign(n + 1, INF);
27         prev.assign(n + 1, -1);
28         prevEdge.assign(n + 1, -1);
29         inQueue.assign(n + 1, false);

```

```

30     queue<int> q;
31     q.push(s);
32     dist[s] = 0;
33     inQueue[s] = true;
34
35     while (!q.empty()) {
36         int u = q.front();
37         q.pop();
38         inQueue[u] = false;
39
40         for (int eid : adj[u]) {
41             Edge& e = edges[eid];
42             int v = e.to;
43
44             if (e.cap > 0 && dist[v] >
45                 dist[u] + e.cost) {
46                 dist[v] = dist[u] + e.cost;
47                 prev[v] = u;
48                 prevEdge[v] = eid;
49
50                 if (!inQueue[v]) {
51                     q.push(v);
52                     inQueue[v] = true;
53                 }
54             }
55         }
56
57         return dist[t] != INF;
58     }
59
60     void augment() {
61         int minFlow = INF;
62         for (int v = t; v != s; v = prev[v]) {
63             Edge& e = edges[prevEdge[v]];
64             minFlow = min(minFlow, e.cap);
65         }
66
67         for (int v = t; v != s; v = prev[v]) {
68             Edge& e = edges[prevEdge[v]];
69             e.cap -= minFlow;
70             edges[prevEdge[v] ^ 1].cap +=
71                 minFlow;
72         }
73
74     public:
75         MinCostMaxFlow(int _n) : n(_n) {
76             adj.resize(n + 1);
77         }
78
79         void addEdge(int from, int to, int cap, int
80             cost) {
81             adj[from].push_back(edges.size());
82             edges.emplace_back(to, cap, cost);
83             adj[to].push_back(edges.size());
84             edges.emplace_back(from, 0, -cost);
85
86             int solve(int _s, int _t) {
87                 s = _s;
88                 t = _t;
89                 int totalCost = 0;
90
91                 while (SPFA()) {
92                     totalCost += dist[t];
93                     augment();
94                 }
95
96                 return totalCost;
97             }
98 };
99
100 int main() {
101     ios_base::sync_with_stdio(false);
102     cin.tie(nullptr);
103
104     int n;
105     cin >> n;
106
107     MinCostMaxFlow mcmf(2 * n + 2);
108     vector<vector<int>>> cost(n + 1,
109         vector<int>(n + 1));
110
111     // 讀入成本矩陣
112     for (int i = 1; i <= n; i++) {
113         int sum = 0;
114         for (int j = 1; j <= n; j++) {
115             cin >> cost[i][j];
116             sum += cost[i][j];
117         }
118         // 調整成本
119         for (int j = 1; j <= n; j++) {
120             cost[i][j] = sum - cost[i][j];
121         }
122     }
123
124     // 建圖
125     int s = 1, t = 2 * n + 2;
126     for (int i = 1; i <= n; i++) {
127         mcmf.addEdge(s, i + 1, 1, 0); //
128             源點到左側節點
129         mcmf.addEdge(i + n + 1, t, 1, 0); //
130             右側節點到匯點
131
132         for (int j = 1; j <= n; j++) {
133             mcmf.addEdge(i + 1, j + n + 1, 1,
134                 cost[i][j]); // 左側到右側的連接
135         }
136     }
137
138     cout << mcmf.solve(s, t) << endl;
139
140     return 0;

```