ABC 075 解説

writer: Hec

2017 年 10 月 14 日

A: One out of Three

入力で整数 A, B, C を受け取ります。$A = B$ の場合には C、$A = C$ の場合には B、$B = C$ の場合には A を出力します。

C++のコード例

```cpp
int main(void) {
    int a,b,c;
    cin >> a >> b >> c;

    if(a == b){
        cout << c << endl;
    }else if(a == c){
        cout << b << endl;
    }else{
        cout << a << endl;
    }

    return 0;
}
```

B: Minesweeper

問題文のとおりにマス目を書き換えることを考えます。まず、二重ループを使って各空きマスに注目します。各空きマスの周囲 8 方向に隣接するマスの種類を調べて、爆弾マスの個数を数えていきます。このとき、各方向に対応する差分を持った配列を利用して調べると実装が簡単になります。また、処理の途中で空白マスを表す文字「.」が数字に置き換わる点に注意してください。

C++のコード例

```cpp
int main(void) {
    int h, w;
    cin >> h >> w;

    string board[50];

    for (int i = 0; i < h; ++i)  cin >> board[i];

    const int dx[8] = {1, 0, -1, 0, 1, -1, -1, 1};
    const int dy[8] = {0, 1, 0, -1, 1, 1, -1, -1};

    for (int i = 0; i < h; ++i) {
        for (int j = 0; j < w; ++j) {
            if (board[i][j] == '#') continue;

            int num = 0;
            for (int d = 0; d < 8; ++ d) {
                const int ni = i + dy[d];
                const int nj = j + dx[d];

                if (ni < 0 or h <= ni) continue;
                if (nj < 0 or w <= nj) continue;
                if (board[ni][nj] == '#') num++;
            }

            board[i][j] = char(num + '0');

        }
    }

    for (int i = 0; i < h; ++i)  cout << board[i] << endl;

    return 0;
}
```

C: Bridge

まず、各辺について橋であるかを判定することを考えます。グラフから辺を取り除いたときに、非連結グラフとなる辺を橋と呼ぶので、この定義にそって調べていきます。したがって、各辺をグラフから取り除いて、グラフが連結であるかを判定します。グラフの連結判定には、深さ優先探索 (DFS)、幅優先探索 (BFS)、Union find、ワーシャルフロイド法などが使えます。今回の問題の制約であれば、ここで挙げたアルゴリズムのどれを使っても、十分に間に合います。

ここでは、グラフの連結判定のための DFS の実装について説明します。グラフの連結判定のための DFS は、以下の疑似コードで表わされる再帰関数で実装することができます。

```
1: procedure DFS(現在の頂点 v)
2:     頂点 v を訪問済みにする
3:     for 頂点 v2: 頂点 v に隣接しているかつ未訪問 do
4:         DFS(v2)
5:     end for
6:     return
7: end procedure
```

連結判定をする際には、全ての頂点を未訪問にしてから、任意の 1 つの頂点から DFS を呼びます。その後に全ての頂点が訪問済みならば連結であり、そうでない場合は非連結です。また、グラフの辺の繋がりは隣接行列で管理でき、隣接行列は 2 次元配列で実装することができます。

以上をまとめると、各辺をグラフから取り除いて、グラフが連結であるかを判定して、非連結となる場合を数え上げます。DFS を用いた場合の時間計算量は、グラフの連結判定に $O(N+M)$ でかかるため、全体の時間計算量は $O(M(N+M))$ となり間に合います。

C++のコード例 (隣接行列でグラフを管理して、DFS で連結判定する)

```cpp
const int limit = 50;

int n, m;
int a[limit], b[limit];

bool graph[limit][limit];
bool visited[limit];

void dfs(int v) {
    visited[v] = true;
    for (int v2 = 0; v2 < n; ++v2) {
        if (graph[v][v2] == false) continue;
        if (visited[v2] == true) continue;
        dfs(v2);
    }
}

int main(void) {
```

```
19      cin >> n >> m;
20
21      for (int i = 0; i < m; ++i) {
22          cin >> a[i] >> b[i];
23          a[i]--, b[i]--;
24          graph[a[i]][b[i]] = graph[b[i]][a[i]] = true;
25      }
26
27      int ans = 0;
28
29      for (int i = 0; i < m; ++i) {
30          graph[a[i]][b[i]] = graph[b[i]][a[i]] = false;
31
32          for (int j = 0; j < n; ++j) visited[j] = false;
33
34          dfs(0);
35
36          bool bridge = false;
37          for (int j = 0; j < n; ++j) if (visited[j] == false) bridge = true;
38          if (bridge) ans += 1;
39
40          graph[a[i]][b[i]] = graph[b[i]][a[i]] = true;
41      }
42
43      cout << ans << endl;
44
45      return 0;
46  }
```

D: Axis-Parallel Rectangle

まず、最小の面積となるような長方形の候補について考えてみます。ここで、図 1 のように適当に K 点以上含んでいるような長方形を考えます。このとき、図 2 のように各辺を点に接するまで長方形を小さくしても、含まれる点の数は変わりません。

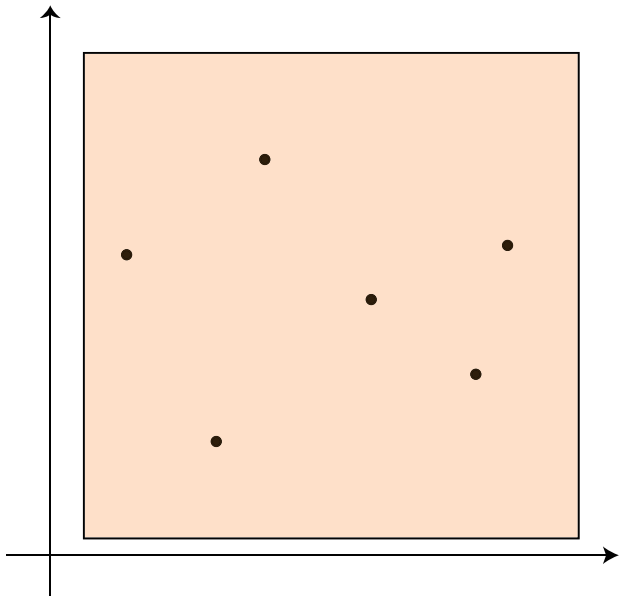

図 1: 6 つの点を含む長方形の例

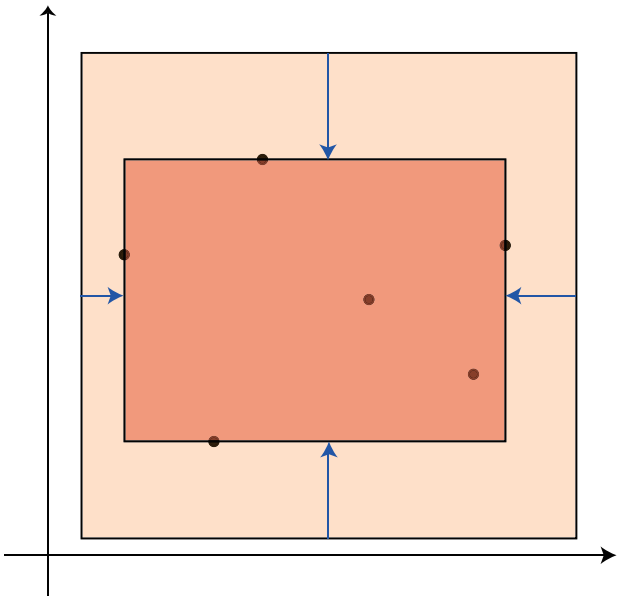

図 2: 内部に含む点の数を変えずに長方形を縮小

したがって、無限にある長方形の候補から、与えられた点集合の座標をもとにした有限の候補に絞り込めます。具体的には、長方形の左辺と右辺は点集合の x 座標、長方形の上辺と下辺は点集合の y 座標が候補となります。

そうすると、長方形の候補を全探索をすることが可能となります。点集合の x 座標から左辺と右辺を全探索して、点集合の y 座標から長方形の上辺と下辺を全探索します。ここで長方形の形が決まるので、あとは長方形の内部に含まれる点の数を数え上げて、K 点以上なら答えの候補です。

この解法の時間計算量は $O(N^5)$ となり、十分間に合います。また、長方形の内部に含まれる点の数の数え上げに 2 次元累積和を利用すると、時間計算量を $O(N^4)$ に改善することが可能です。

C++のコード例（時間計算量は $O(N^5)$）

```
using ll = long long;

int main(void){
    int n,k;
    cin >> n >> k;

    vector<int> x(n),y(n);
    vector<int> xary,yary;

    for(int i = 0; i < n; ++i){
        cin >> x[i] >> y[i];
```

```
12          xary.push_back(x[i]);
13          yary.push_back(y[i]);
14      }
15
16      sort(begin(xary),end(xary));
17      sort(begin(yary),end(yary));
18
19      ll ans = 1LL * (xary[n-1] - xary[0]) * (yary[n-1] - yary[0]);
20
21      for(int xi = 0; xi < n; ++xi){
22          for(int xj = xi + 1; xj < n;++xj){
23              for(int yi = 0; yi < n; ++yi){
24                  for(int yj = yi + 1; yj < n;++yj){
25
26                      int num = 0;
27                      ll lx = xary[xi],rx = xary[xj];
28                      ll by = yary[yi],uy = yary[yj];
29
30                      for(int i = 0; i < n;++i){
31                          if(lx <= x[i] and x[i] <= rx and by <= y[i] and y[i] <= uy){
32                              num++;
33                          }
34                      }
35
36                      if(num >= k) ans = min(ans,1LL * (rx - lx) * (uy - by));
37                  }
38              }
39          }
40      }
41
42      cout << ans << endl;
43
44      return 0;
45  }
```