ABC 051 解説

writer: Hec

2017 年 1 月 6 日

A: Haiku

文字列を入力して、文字列の 6 文字目と 14 文字目をスペースに置き換えて出力すれば良いです。置き換える方法の例として、次の 2 つが挙げられます。

- 文字列の 6 文字目と 14 文字目をスペースに変更してから、文字列を出力する。
- 文字列をループ処理で 1 文字ずつ見ていき、その文字がカンマだったらスペース、そうでなければそのまま文字を出力する。

C++のコード例 (1 つ目の例)

```
int main(void){
    string s;
    cin >> s;
    s[5]=' ',s[13]=' ';
    cout << s << endl;
    return 0;
}
```

B: Sum of Three Integers

まず、変数 X, Y, Z の組を全探索して、$X + Y + Z = S$ となる組を数え上げることを考えます。各変数を 0 から K まで回す 3 重ループだと、時間計算量は $O(K^3)$ となります。ただし、K の上限は 2500 であるため、このままでは間に合いません。

そこで、$X + Y + Z = S$ に注目します。変数 X と Y が決まると、変数 Z の値は $S - X - Y$ に決まります。これを利用すると、変数 X, Y の組を全探索して、変数 Z の値が 0 以上かつ K 以下であるような組を数えると、答えが求まります。この解法の時間計算量は、$O(K^2)$ となるため間に合います。

C++のコード例

```
int main(void){
    int K,S;
    cin >> K >> S;

```

```
    int ans = 0;

    for(int x = 0; x <= K; ++x){
        for(int y = 0; y <= K; ++y){
            int z = S - x - y;
            if(0 <= z and z <= K){
                ans = ans + 1;
            }
        }
    }

    cout << ans << endl;
    return 0;
}
```

C: Back and Forth

まず、点 $s = (s_x, s_y)$ から点 $t = (t_x, t_y)$ への経路について考えます。問題文の条件より、点 s と点 t を除いた全ての格子点で交わらない 4 つの経路を見つける必要があります。この条件から、図 1 で示すような点 $s = (s_x, s_y)$ と点 $t = (t_x, t_y)$ のそれぞれ上下左右に距離 1 進んだ点は必ず訪れる必要があります。そうでなければ、先に述べた 4 つの経路の条件に反してしまうからです。

次に、図 1 で示した点 s の周辺の 4 点から点 t の周辺の 4 点への交差しない 4 つの経路を見つけることを考えます。このとき、図 2 のような 4 つの経路を見つけることができ、これらの経路は全て最短距離になっています。なぜなら、s の周辺の 4 点から t の周辺の 4 点へ結ぶ移動は右移動と上移動で行えるからです。あとは、4 つの経路を合体させ、$s \to t \to s \to t \to s$ と移動するような文字列を出力すれば良いです。

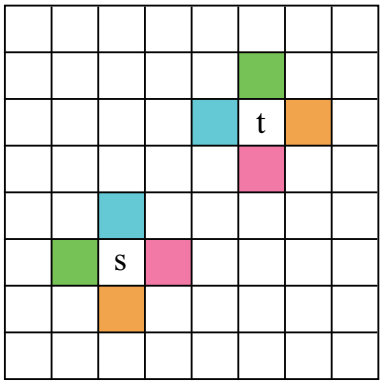

図 1:

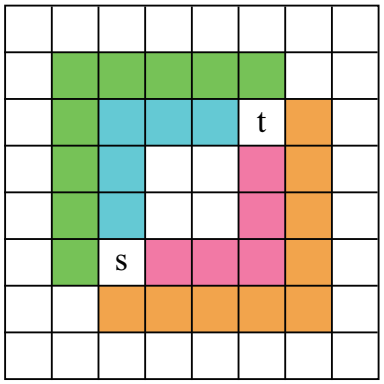

図 2:

C++のコード例

```
int main(void){
    int sx,sy,tx,ty;
    cin >> sx >> sy >> tx >> ty;
```

```
    const int dx=tx-sx,dy=ty-sy;

    // Path 1
    cout << string(dy,'U') << string(dx,'R');

    // Path 2
    cout << string(dy,'D') << string(dx,'L');

    // Path 3
    cout << 'L' << string(dy+1,'U') << string(dx+1,'R') << 'D';

    // Path 4
    cout << 'R' << string(dy+1,'D') << string(dx+1,'L') << 'U';

    // EndLine
    cout << endl;
    return 0;
}
```

D: Candidates of No Shortest Paths

まず、$\mathrm{edge}(i,j)$ を頂点 i と j を結ぶ辺のコスト、$\mathrm{dist}(i,j)$ を頂点 i から頂点 j までの最短距離とします。最短経路の候補となる辺を求めるために、全頂点間の最短経路を求める必要があります。これは、ワーシャルフロイド法 (時間計算量 $O(N^3)$) やダイクストラ法 (時間計算量 $O(NM\log N)$) を使って求めることができます。

次に、頂点 i と頂点 j を結ぶ辺が頂点 s から頂点 t への最短経路に含まれるとき、次の式が成り立ちます。

$$\mathrm{dist}(s,i)+\mathrm{edge}(i,j)+\mathrm{dist}(j,t)=\mathrm{dist}(s,t) \tag{1}$$

式 (1) を満たさない場合、この辺を使った経路は頂点 s から頂点 t への最短経路とならないことが分かります。したがって、頂点 s,t の組を全探索を行い、式 (1) を満たす頂点 s,t の組が 1 つでもあれば、その辺はある最短経路に含まれます。各辺に対してこのチェックを行うことにより、時間計算量 $O(MN^2)$ で答えを求めることができ、間に合います。

なお、さらに時間計算量を減らすことができます。全頂点間の最短距離が求まっているため、最短経路の判定に式 (2) を使うことができます。

$$\mathrm{dist}(s,i)+\mathrm{edge}(i,j)=\mathrm{dist}(s,j) \tag{2}$$

この場合、頂点 s を全探索すれば良いので、時間計算量 $O(MN)$ で答えを求めることができます。

また、ある頂点 s からダイクストラ法を行い、その最短経路木に含まれる辺は当然ながら最短経路に含まれます。全頂点を 1 回ずつ始点にしたダイクストラ法を行い、その最短経路木に含まれる辺をチェックすることでも、答えを求めることが可能です。(証明略)

C++のコード例

```
int main(void){
    const int inf = 100000000;
```

```
3
4       int n,m;
5       cin >> n >> m;
6
7       int a[1000],b[1000],c[1000],dist[100][100];
8
9       for(int i = 0; i < m; ++i){
10          cin >> a[i] >> b[i] >> c[i];
11          a[i]--,b[i]--;
12      }
13
14      for(int i = 0; i < n; ++i){
15          for(int j = 0; j < n; ++j){
16              if(i==j)
17                  dist[i][j]=0;
18              else
19                  dist[i][j]=inf;
20          }
21      }
22
23      for(int i = 0; i < m; ++i){
24          dist[a[i]][b[i]]=min(dist[a[i]][b[i]],c[i]);
25          dist[b[i]][a[i]]=min(dist[b[i]][a[i]],c[i]);
26      }
27
28      for(int k = 0; k < n; ++k){
29          for(int i = 0; i < n; ++i){
30              for(int j = 0; j < n; ++j){
31                  dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
32              }
33          }
34      }
35
36      int ans=m;
37      for(int i = 0; i < m; ++i){
38          bool shortest=false;
39          for(int j = 0; j < n; ++j) if(dist[j][a[i]]+c[i]==dist[j][b[i]]) shortest=true;
40          if(shortest==true){
41              ans=ans-1;
42          }
43      }
44
45      cout << ans << endl;
46  }
```