ABC 054 解説

writer: Hec

2017 年 2 月 11 日

A: One Card Poker

2 つの整数を入力として受け取り、条件文を用いてプレイヤーの勝敗判定を行い、適切な出力を行えば良いです。勝敗判定の際に、厄介となる点は 1 の扱いです。1 を特別扱いして条件文で場合分けする方針でも解くことは可能です。より簡単な方法として、1 を 14 として扱うと勝敗判定を整数の大小判定で行う事ができます。

C++のコード例

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

    if(a==1) a+=13;
    if(b==1) b+=13;

    if(a>b)
        cout << "Alice" << endl;
    else if(a<b)
        cout << "Bob" << endl;
    else
        cout << "Draw" << endl;

    return 0;
}
```

B: Template Matching

実際に、テンプレート画像 B を平行移動させて画像 A に含まれるかを調べます。まず、画像 A の中における $M \times M$ の正方形領域とテンプレート画像 B の同一判定は、二重ループで行うことが可能です。次に、画像 A の中における $M \times M$ の正方形領域は何通りあるか考えます。これは $(N-M+1) \times (N-M+1)$ 通り考えられます。以上のことから、全探索を行ったときの時間計算量は $O(N^2M^2)$ となり、これは間に合います。

より高速な解法として、ハッシュを用いた解法が存在します。(詳細は略)

C++のコード例

```cpp
int main(void){
    int N,M;
    cin >> N >> M;

    const int nmmax=50;
    char A[nmmax][nmmax],B[nmmax][nmmax];


    for(int y=0;y<N;++y){
        for(int x=0;x<N;++x){
            cin >> A[y][x];
        }
    }

    for(int y=0;y<M;++y){
        for(int x=0;x<M;++x){
            cin >> B[y][x];
        }
    }

    bool exist=false;

    for(int ly=0;ly<N;++ly){
        for(int lx=0;lx<N;++lx){
            if(ly+M-1>=N or lx+M-1>=N) continue;

            bool match=true;
            for(int y=0;y<M;++y){
                for(int x=0;x<M;++x){
                    if(A[ly+y][lx+x]!=B[y][x]) match=false;
                }
            }
            if(match) exist=true;
        }
    }

    if(exist)
        cout << "Yes" << endl;
    else
        cout << "No" << endl;

    return 0;
}
```

C: One-stroke Path

簡単のために、与えられた無向グラフを隣接行列を用いて受け取ります。与えられた無向グラフに対して、頂点 1 を始点とする深さ優先探索 (DFS) を用いて、候補となるパスを数え上げることを考えます。まず、頂点 1 を訪問済みとします。その後に、頂点 1 を引数として以下の再帰関数を呼びます。この再帰関数は、隣接頂点が全て訪問済みとなるまでグラフを辿っていくことで、パスを列挙します。

1: **procedure** DFS(現在の頂点 v)
2: **if** 全ての頂点を訪問済み **then**
3: 答えを 1 増やす。
4: **return**
5: **end if**
6: **for** 頂点 i: 頂点 v に隣接しているかつ未訪問 **do**
7: 頂点 i を訪問済みとする。
8: DFS(i)
9: 頂点 i を未訪問とする。
10: **end for**
11: **return**
12: **end procedure**

ここで、候補となるパスの数の上限を考えます。パスの始点は頂点 1 かつ全ての頂点は 1 度しか訪れないため、2 番目から N 番目までに訪れる頂点は $\{2, \ldots, N\}$ の順列で表されます。このため、パスの候補は最大で $(N-1)!$ 通りです。頂点数の上限は $N \leq 8$ であるため、最大で $7! = 5040$ 通りしか存在しません。この解法の時間計算量は、$O(N!)$ となるため間に合います。

別解としては、$\{2, \ldots, N\}$ の順列を列挙を行い、条件を満たしたパスを数え上げます。先ほどと同様に、この解法の時間計算量は、$O(N!)$ となります。

より高速な解法として、bitDP を用いた時間計算量 $O(2^N N^2)$ となる解法が存在します。(詳細は略)

C++のコード例 (DFS 解)

```cpp
const int nmax=8;
bool graph[nmax][nmax];

int dfs(int v,int N,bool visited[nmax]){

    bool all_visited=true;

    for(int i=0;i<N;++i){
        if(visited[i]==false)
            all_visited=false;
    }

    if(all_visited){
        return 1;
    }
```

```
16
17      int ret=0;
18
19      for(int i=0;i<N;++i){
20          if(graph[v][i]==false) continue;
21          if(visited[i]) continue;
22
23          visited[i]=true;
24          ret+=dfs(i,N,visited);
25          visited[i]=false;
26      }
27
28      return ret;
29  }
30
31  int main(void){
32      int N,M;
33      cin >> N >> M;
34
35      for(int i=0;i<M;++i){
36          int A,B;
37          cin >> A >> B;
38          graph[A-1][B-1]=graph[B-1][A-1]=true;
39      }
40
41      bool visited[nmax];
42      for(int i=0;i<N;++i){
43          visited[i]=false;
44      }
45
46      visited[0]=true;
47      cout << dfs(0,N,visited) << endl;
48      return 0;
49  }
```

D: Mixing Experiment

まず、物質 C を生成するための条件に注目します。タイプ A の物質とタイプ B の物質の混合比が $M_a : M_b$ と買った薬品は全て使うという条件から、購入した薬品の金額とそれら全てを混ぜた溶液に含まれる物質 A と物質 B の重さが重要になります。

この考察をもとに、次のような 3 次元 DP を考えます。

$dp[i][ca][cb]$:= 1–i 番目までの薬品の組み合わせで、物質 A が ca グラム、物質 B が cb グラムとなる溶液の最小コスト

まず、dp テーブル全体を ∞ で初期化します。(∞ の具体的な値は、全ての薬品を買ったコスト $N\max(C_i)$ より大きくなるように決めます。) また、$i = 0$ ときイルカは薬品を 1 つも持たないので、$\mathrm{dp}[0][0][0] = 0$ と初期化します。次に、dp テーブルの更新を考えます。ここでは、$\mathrm{dp}[i][ca][cb]$ に注目して、i から $i+1$ $(0 \leq i \leq N-1)$ への具体的な更新方法を考えます。このとき、$i+1$ 番目の薬品を加える・加えないの 2 つの処理が考えられて、更新方法は次の通りです。(ここでは、入力の A,B,C は 1-indexed ではなく、0-indexed で表されます。)

- 薬品 i を加えない場合: $\mathrm{dp}[i+1][ca][cb] = \min(\mathrm{dp}[i+1][ca][cb], \mathrm{dp}[i][ca][cb])$
- 薬品 i を加える場合: $\mathrm{dp}[i+1][ca+\mathrm{A}[i]][cb+\mathrm{B}[i]] = \min(\mathrm{dp}[i+1][ca+\mathrm{A}[i]][cb+\mathrm{B}[i]], \mathrm{dp}[i][ca][cb]+\mathrm{C}[i])$

以上の更新を i と ca と cb に関する三重ループを用いて行います。結果として、全ての薬品の組み合わせは $\mathrm{dp}[N][\cdot][\cdot]$ で表されます。

最後に、$ca : cb = M_a : M_b$ を満たしている $\mathrm{dp}[N][ca][cb]$ に注目して最小コストを求めて出力します。このとき、$ca = 0$ かつ $cb = 0$ は条件を満たさないことに気をつけて下さい。なお、$ca : cb = M_a : M_b$ の判定には $caM_b = cbM_a$ という比の等式を使うのが簡単です。また、求めた最小コストが ∞ であるならば、物質 C を生成できないので代わりに -1 を出力します。

ここからは、時間計算量について考えていきます。まず、三重ループの各変数が取りうる上限を考えていきます。薬品の個数の上限は、制約から $N \leq 40$ です。物質 A の重さの上限は、$N\max(A_i) \leq 400$ で表されます。同様に、物質 B の重さの上限は、$N\max(B_i) \leq 400$ で表されます。また、dp テーブルの更新は $O(1)$ で行うことができます。以上のことから、この解法の時間計算量は、$O(N^3\max(A_i)\max(B_i))$ となり、間に合います。

なお、比の等式と半分全列挙を用いた時間計算量 $O(2^{\frac{N}{2}}N)$ となる解法が存在します。(詳細は略)

C++のコード例

```
const int nmax=40,abmax=10,inf = 1000000;

int a[nmax],b[nmax],c[nmax];
int dp[nmax+1][nmax*abmax+1][nmax*abmax+1];

int main(void){
    int n,ma,mb;
    cin >> n >> ma >> mb;

    for(int i=0;i<n;++i){
        cin >> a[i] >> b[i] >> c[i];
    }

    for(int i = 0; i <= n; ++i){
        for(int ca = 0; ca <= nmax*abmax; ++ca){
            for(int cb = 0; cb <= nmax*abmax; ++cb){
                dp[i][ca][cb]=inf;
            }
        }
    }

    dp[0][0][0]=0;
```

```
23
24      for(int i = 0; i < n; ++i){
25          for(int ca = 0; ca <= nmax*abmax; ++ca){
26              for(int cb = 0; cb <= nmax*abmax; ++cb){
27                  if(dp[i][ca][cb]==inf) continue;
28                  dp[i+1][ca][cb]=min(dp[i+1][ca][cb],dp[i][ca][cb]);
29                  dp[i+1][ca+a[i]][cb+b[i]]=min(dp[i+1][ca+a[i]][cb+b[i]],dp[i][ca][cb]+c[i]);
30              }
31          }
32      }
33
34      int ans=inf;
35      for(int ca = 1; ca <= nmax*abmax; ++ca){
36          for(int cb = 1; cb <= nmax*abmax; ++cb){
37              if(ca*mb==cb*ma) ans=min(ans,dp[n][ca][cb]);
38          }
39      }
40
41      if(ans==inf) ans=-1;
42      cout << ans << endl;
43
44      return 0;
45  }
```