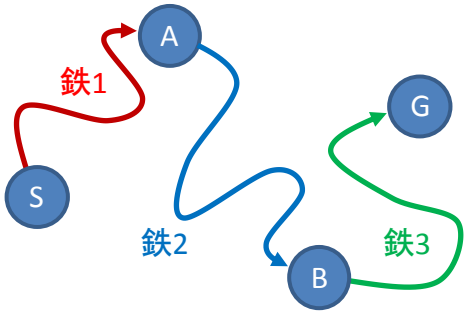
ACM ICPC 2012 国内予選問題 D 「鉄道乗り継ぎ」

- S 駅から G 駅に至る最小運賃の経路が右図のように求まったとする。この時、駅 S から駅 A へは鉄道会社「鉄 1」による図示の経路が最小運賃であり、駅 A から駅 B へは鉄道会社「鉄 2」による図示の経路が最小運賃であり、駅 B から駅 G へは鉄道会社「鉄 3」による図示の経路が最小運賃になっている。したがって、鉄道会社ごとに駅間の最小運賃を求めておき、それらの最小値を駅間の運賃として S 駅から G 駅へ至る最小運賃の経路を求めれば良い。



- アルゴリズムの概要
  1. 各鉄道会社ごとに、各駅間の最小距離を求める。この問題は全対最短経路問題なので、ワーシャル・フロイド法を用いる。
  2. 上記の最小距離を用いて、各鉄道会社ごとに、各駅間の最小運賃を求める。
  3. 各駅間について、全鉄道会社の最小運賃の最小値を求める。これは、どの鉄道会社でも良いが一つの鉄道会社のみを利用する場合の、各駅間の最小運賃となる。
  4. 上記で得られた各駅間の最小運賃を用いて、駅 S から駅 G への最小運賃を求めれば良い。この問題は、ダイクストラ法で効率よく解くことができるが、上記 1 でワーシャル・フロイド法を実装しているので、それを再利用することにする。
- 駅間に経路が無い事を表すために、経路が無い駅間の距離や運賃を大きな整数値 INF で表現することにする。路線数の最大値は 10,000、路線の長さの最大値は 200、距離あたりの運賃単価の最大値は 100 であるので、2 駅間の最小運賃は、$10{,}000 \times 200 \times 100 = 2 \times 10^8$ を超えることは無い。したがって、INF として、例えば 10^9 を用いることが出来る。
- 鉄道会社 c のみを利用した時の駅間の最小距離や最小運賃は、
  3 次元配列 dist[c][駅 1][駅 2]を用いるものとする。任意の鉄道会社を利用する場合の最小運賃は dist[0][駅 1][駅 2]に上書きする形で求めるものとする。
- 料金計算が容易に行えるように、料金グラフの折れ目位置の運賃を事前に計算し、2 次元配列 sc[c][j]にセットしておく。sc[c][j]は会社 c の j 番目の折れ目位置の運賃を表す。

【プログラム例】

```
// ICPC 2012 国内予選問題 D
// http://www.cs.titech.ac.jp/icpc2012/icpc2012-mondai/icpc2012/contest/D_ja.html
// Filename = pd.c
// Compile:    cc pd.c
// Execution: ./a.out < D0 > D0.result
// Check:       diff D0.ans D0.result
```

```
//
#include <stdio.h>
#define MAX_N 100          // 駅の数の最大値
#define MAX_M 10000        // 路線の数の最大値
#define MAX_D 200          // 路線の長さの最大値
#define MAX_C 20           // 鉄道会社の数の最大値
#define MAX_AREA 50        // 運賃表の区間の数の最大値
#define MAX_Q 10000        // 運賃表の折れ目の位置を示す距離の最大値
#define MAX_R 100          // 運賃表の距離単価の最大値
#define INF 1000000000     // 無限大を表す = 10^9  >
                           //      MAX_M * MAX_D * MAX_R = 200000000 = 2 * 10^8
int n;     // 駅の数（2 <= n <= 100)
int m;     // 路線数（0 <= m <= 10000
int nc;    // 鉄道会社の数（1 <= nc <= 20)
int s;     // 出発地の駅番号
int g;     // 目的地の駅番号
int p[MAX_C];                      // 料金表の区間の数（1 <= p[i] <= 50)
int q[MAX_C][MAX_AREA];            // 料金グラフの折れ目の位置を示す距離
int r[MAX_C][MAX_AREA];            // その領域の距離単価
int sc[MAX_C][MAX_AREA];           // 折れ目位置での運賃
int dist[MAX_C][MAX_N][MAX_N];

// ワーシャルーフロイド法により、鉄道会社 c だけを使う場合の全対最短距離を求める
void wf(int c)
{
  int i,j,k;

  for(k=0; k<n; k++){
    for(i=0; i<n; i++){
      for(j=0; j<n; j++){
        // 駅 i から駅 j への距離として駅 k 経由の方が短ければその距離を採用
        if( dist[c][i][j] > dist[c][i][k] + dist[c][k][j])
          dist[c][i][j] = dist[c][i][k] + dist[c][k][j];
      }
    }
  }
}

// 鉄道会社 c の距離 d に対する料金を求める
int cost(int c, int d)
{
  int i;
  int uc = 0;;
  if(d==0) return 0;          // 距離 0 の運賃は 0
  if(d==INF) return INF;  // 距離が無限（到達不能）の時は運賃を無限としておく
  for(i=p[c]-1; i>=0; i--){    // 距離 d の直前の折れ目位置を見つける
    if(d > q[c][i]) break;
  }
  // i が距離 d の直前の折れ目位置
  // sc[c][i] は i 番目の折れ目位置の運賃
  // q[c][i] は i 番目の折れ目位置の距離なので
  // d - q[c][i] は i 番目の折れ目位置から d までの距離を表す。
```

```
    // r[c][i] は d が含まれる区間の距離当たりの運賃
    return sc[c][i] + (d - q[c][i])*r[c][i];
}

// 鉄道会社ごとの最短距離マップを最小料金マップに変換
void dtoc(int c)
{
    int i,j,d;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            dist[c][i][j] = cost(c, dist[c][i][j]);
}

void solve()
{
    int i,j,c;
    for(i=0; i<nc; i++){
        wf(i);              // 鉄道会社 i のみを使用する場合の各駅間の最短距離を求める
        dtoc(i);            // 鉄道会社 i のみを使用する場合の各駅間の最低運賃を求める
    }
    // 一つの鉄道会社のみを利用する場合の各駅間の最低運賃を求め、dist[0][i][j]にセット
    for(i=0; i<n; i++)
        for(j=0; j<n; j++){
            for(c = 1; c<nc; c++)
                if(dist[c][i][j] < dist[0][i][j])
                    dist[0][i][j] = dist[c][i][j];
        }
    // dist[0][i][j] の情報に基づき各駅間の最低運賃を計算すると
    // 複数の鉄道会社の利用も含めた場合の最低運賃が求まる。
    // ・ 出発駅と目的駅が与えられているので、ダイクストラ法で与えられた２駅間の
    //      最低運賃を求める方が計算量は少なくなるが、ワーシャルーフロイド法を
    //      既に実装しているので、それを再利用する。
    wf(0); // ワーシャルーフロイド法で全ての２駅間の最低運賃を求める
    if(dist[0][s-1][g-1] == INF)                    // INF は到達不能を意味するので
        printf("-1¥n");                             // -1 を印刷
    else
        printf("%d¥n", dist[0][s-1][g-1]);          // s から g への最低運賃を印刷
}

void init()
{
    int i,j,k;
    // 運賃表のデータを初期化
    for(i=0; i<MAX_C; i++){
        p[i] = 0;
        for(j=0; j<MAX_AREA; j++)
            q[i][j] = r[i][j] = 0;
    }
    // 対角要素は 0 に、それ以外は「無限」に初期化
    for(i=0; i<MAX_C; i++){
        for(j=0; j<MAX_N; j++){
            for(k=0; k<MAX_N; k++){
                if(j==k) dist[i][j][k] = dist[i][j][k] = 0;
```

```
            else dist[i][j][k] = dist[i][j][k] = INF;
        }
      }
    }
}

int main()
{
    int i,j;
    int x,y,d,c;
    while(1){
      // 駅の数 n, 路線の数 m, 鉄道会社の数 nc, 出発駅 s, 目的駅 g を読み込む
      scanf("%d %d %d %d %d", &n, &m, &nc, &s, &g);
      if(n==0 && m==0 && nc==0 && s==0 && g==0) break;     // 全て 0 なら終了
      init();             // 各配列を初期化
      for(i=0; i<m; i++){                    // 路線の数だけ路線情報を読み込む
        // x, y: 路線の駅番号,  d: 路線の距離,  c:鉄道会社番号
        scanf("%d %d %d %d", &x, &y, &d, &c);
        // 同一の鉄道会社で駅間に複数の路線がある場合、最小距離のものを採用
        if(dist[c-1][x-1][y-1] > d)
          dist[c-1][x-1][y-1] = dist[c-1][y-1][x-1] = d;      // 双方向に移動可能なので
                                                              // 対称位置もセット
      }
      // 料金表の区間数を読み込む
      for(i=0; i<nc; i++)
        scanf("%d", &p[i]);   // p[i]: 会社 i の料金表の区間数
      for(i=0; i < nc; i++){    // 各会社 i に対して
        for(j=1; j < p[i]; j++)       // 各区間ごとに
          scanf("%d", &q[i][j]);          // 料金表の折れ目位置の距離を読み込む
        q[i][0] = 0;                // 最も左側に「0」に対応する折れ目位置を設ける
        for(j=0; j<p[i]; j++)   // 各区間ごとに
          scanf("%d", &r[i][j]);          // 料金表の距離単価を読み込む
        sc[i][0] = 0;
        for(j=1; j<p[i]; j++)   // 料金表の折れ目位置の距離に対する運賃を求める
          sc[i][j] = sc[i][j-1]+(q[i][j]-q[i][j-1])*r[i][j-1];
      }
      solve();                      // 解を求める
    }
}
```