ソフトウェア総合演習I
中間レビュー

team_tower
s1250125 Takahisa Watanabe
s1250183 Yuki Homma

1. 開発環境

-> 開発環境

チーム

チーム名: **team_tower** https://github.com/ie03-aizu-2019/ie03project-team_tower

メンバー: **s1250125 渡邉 貴久**

s1250183 本間 祐樹

使用言語

C言語

2. 開発状況 (小課題1 ~ 8)

小課題1 交差地点の検出

-> 開発状況 小課題1 交差地点の検出　データ構造

線分を表現する配列

```
int road[MMAX+1][2];  // 0: 端点Pのid, 1: 端点Qのid
```

ex) ID 1 の線分の場合

road[1][1] = 座標Qのid

road[1][0] = 座標Pのid

座標を表現する構造体

```
typedef struct {
  double x;
  double y;
  int roadA;
  int roadB;
  int id;
} point_t;
```

座標x, y

座標が交差地点だったら、
交差している2本の線分idがA, Bに入る

座標のid (1 ~ n)

-> 開発状況 小課題1 交差地点の検出　アルゴリズム

Step 1.	If 式(5) = 0 Else	交差無し Step 2 へ
Step 2.	式(6)から s, t を求める	Step 3 へ
Step 3.	If 0≦s≦1 かつ 0≦t≦1 Else	交差有り Step 4 へ 交差無し
Step 4.	交差地点の x,y 座標を求める. 式(1)か式(3)に s を代入して x 座標を求める 式(2)か式(4)に t を代入して y 座標を求める	

$$x = x_{P1} + (x_{Q1} - x_{P1})s \quad (1) \qquad x = x_{P2} + (x_{Q2} - x_{P2})t \quad (3)$$

$$y = y_{P1} + (y_{Q1} - y_{P1})s \quad (2) \qquad y = y_{P2} + (y_{Q2} - y_{P2})t \quad (4)$$

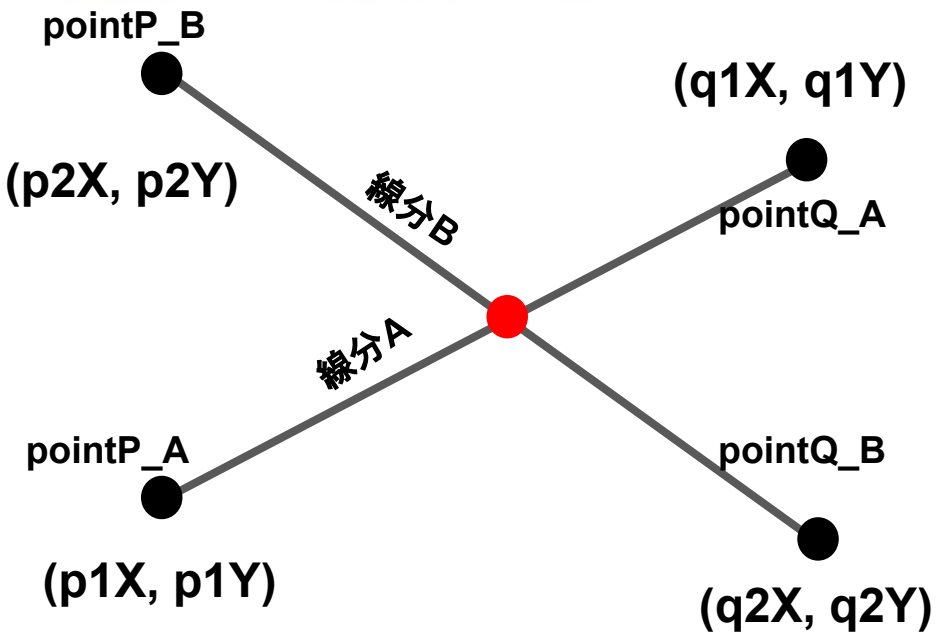
$$|A| = (x_{Q1} - x_{P1})(y_{P2} - y_{Q2}) + (x_{Q2} - x_{P2})(y_{Q1} - y_{P1}) \quad (5)$$

$$\begin{Bmatrix} s \\ t \end{Bmatrix} = \frac{1}{|A|} \begin{bmatrix} y_{P2} - y_{Q2} & x_{Q2} - x_{P2} \\ y_{P1} - y_{Q1} & x_{Q1} - x_{P1} \end{bmatrix} \begin{Bmatrix} x_{P2} - x_{P1} \\ y_{P2} - y_{P1} \end{Bmatrix} \quad (6)$$

-> 開発状況 小課題1 交差地点の検出 ソースコード

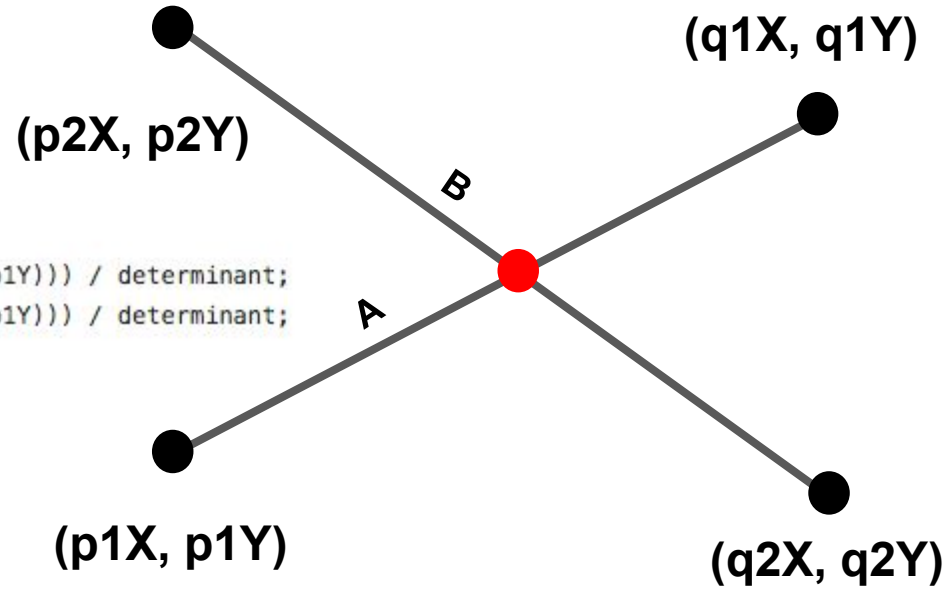
```
// 交差地点を返す関数，ない場合はNotExist{-1, -1}を返す
point_t detectCrossing(point_t pointP_A, point_t pointQ_A, point_t pointP_B, point_t pointQ_B)
```

```
6   // 交差地点を返す関数，ない場合はNotExist{-1, -1}を返す
7   point_t detectCrossing(point_t pointP_A, point_t pointQ_A, point_t pointP_B, point_t pointQ_B) {
8     double p1X, p1Y, q1X, q1Y, p2X, p2Y, q2X, q2Y;
9     double s, t;
10    double x, y;
11    double determinant;
12    point_t crossing;
13    point_t notExist = {-1, -1};
14    int i, index;
15
16    p1X = pointP_A.x;
17    p1Y = pointP_A.y;
18    q1X = pointQ_A.x;
19    q1Y = pointQ_A.y;
20    p2X = pointP_B.x;
21    p2Y = pointP_B.y;
22    q2X = pointQ_B.x;
23    q2Y = pointQ_B.y;
24
```



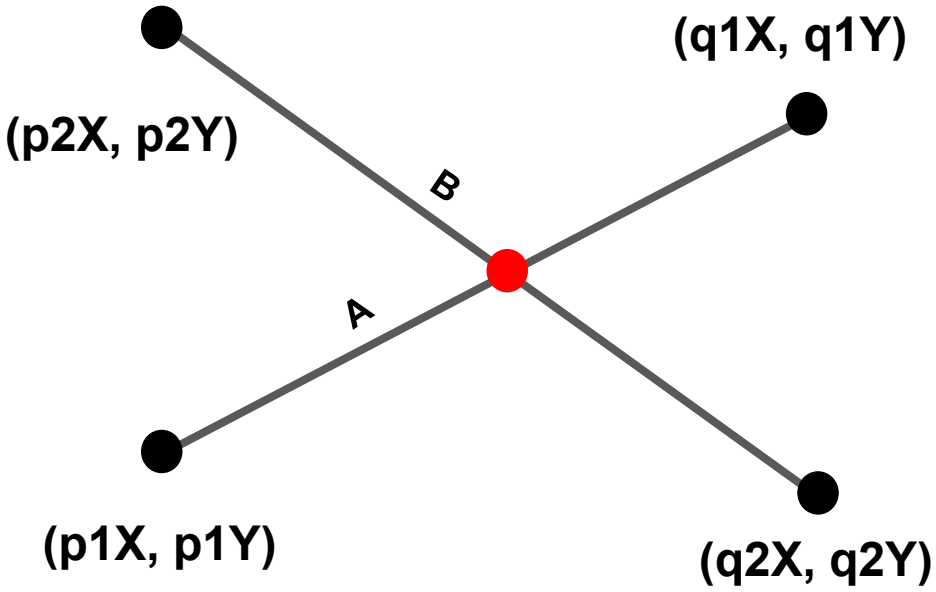
-> 開発状況 小課題1 交差地点の検出　ソースコード

```
25    // 行列式を求める
26    determinant = fabs(( (q1X - p1X)*(p2Y - q2Y) + (q2X - p2X)*(q1Y - p1Y) ));
27
28    // Step1
29    if( (determinant <= EPS) && (determinant >= EPS) ) {
30      return notExist;
31    }
32
33    // Step2
34    // パラメータを求める
35    s = fabs(((q2Y - p2Y)*(p2X - p1X) - (q2X - p2X)*(p2Y - p1Y))) / determinant;
36    t = fabs(((q1Y - p1Y)*(p2X - p1X) - (q1X - p1X)*(p2Y - p1Y))) / determinant;
37
```



-> 開発状況 小課題1 交差地点の検出　ソースコード

```
38    // Step3
39    // 線分の端点にある場合(s == 0, 1 or t == 0, 1)は交差地点でない
40    if( ((s <= EPS)&&(s >= EPS)) || ((t >= EPS)&&(t <= EPS)) ||
41        ((s-1 <= EPS)&&(s-1 >= EPS)) || ((t-1 <= EPS)&&(t-1 >= EPS)) ) {
42      return notExist;
43    }
44
45    else if( ((s > 0)&&(s < 1)) && ((t > 0)&&(t < 1))) {
46      // Step 4
47      // 交差地点を求める
48      x = p1X + (q1X - p1X) * s;
49      y = p1Y + (q1Y - p1Y) * s;
50
51      crossing.x = x;
52      crossing.y = y;
53
54      return crossing;
55    }
56
57    else {
58      return notExist;
59    }
60  }
```



小課題2 交差地点の列挙

-> 開発状況

小課題2 交差地点の列挙

→ **小課題1の拡張**

```
26    roadA_P = road[1][0];
27    roadA_Q = road[1][1];
28    roadB_P = road[2][0];
29    roadB_Q = road[2][1];
30
31    tmpPoint =
32      detectCrossing(point[roadA_P], point[roadA_Q], point[roadB_P], point[roadB_Q]);
33
34    if( (tmpPoint.x != -1) && (tmpPoint.y != -1) ) {
35      tmpPoint.roadA = 1;
36      tmpPoint.roadB = 2;
37      printf("%f %f\n", tmpPoint.x, tmpPoint.y);
38    } else {
39      printf("NA\n");
40    }
```

-> 開発状況

小課題2 交差地点の列挙

main → **小課題1の拡張**

```
/* 交差地点を探し出す部分 */
for(i = 1; i < m; i++) {
  for(j = i + 1; j <= m; j++) {
    roadA_P = road[i][0];
    roadA_Q = road[i][1];
    roadB_P = road[j][0];
    roadB_Q = road[j][1];

    tmpPoint =
      detectCrossing(point[roadA_P], point[roadA_Q], point[roadB_P], point[roadB_Q]);
    if( (tmpPoint.x != -1) && (tmpPoint.y != -1) ) {
      crossing[crossCount] = tmpPoint;
      crossCount++;
    }
  }
}

/* xが小さい順にソート */
sortCrossing(crossing, crossCount, n);
```

入力された全ての線分の組み合わせにおいて、交差地点を探し出す！

小課題3 最短経路の距離
小課題4 最短経路

-> 開発状況 小課題3,4 最短経路の距離　アルゴリズム

ダイクストラ法

グラフの2頂点間の最短経路を求めるアルゴリズム

グラフ $G = (V, E)$ 、スタートとなる頂点 s 、および u から v への辺の長さ $\mathrm{length}(u, v)$ を入力として受け取る

// 初期化
各頂点 $v \in V$ に対し、$d(v) \leftarrow (v = s$ ならば 0 、それ以外は $\infty)$
各頂点 $v \in V$ に対し、$\mathrm{prev}(v) \leftarrow$ 「無し」
Q に V の頂点を全て追加

// 本計算
while (Q が空ではない)
  $u \leftarrow Q$ から $d(u)$ が最小である頂点を取り除く
  for each (u からの辺がある各頂点 $v \in V$)
    if ($d(v) > d(u) + \mathrm{length}(u, v)$)
      $d(v) \leftarrow d(u) + \mathrm{length}(u, v)$
      $\mathrm{prev}(v) \leftarrow u$

Wikipediaより
https://ja.wikipedia.org/wiki/%E3%83%80%E3%82%A4%E3%82%AF%E3%82%B9%E3%83%88%E3%83%A9%E6%B3%95

-> 開発状況 小課題3,4 最短経路の距離　データ構造

```
3   // 座標の構造体
4   typedef struct {
5     double x;
6     double y;
7     int roadA;
8     int roadB;
9     double cost;     // コスト
10    int done;     // 訪問済みのフラグ
11    int preNode; // 最短経路として選択された自身の前のノード
12    int id;
13  } point_t;
```

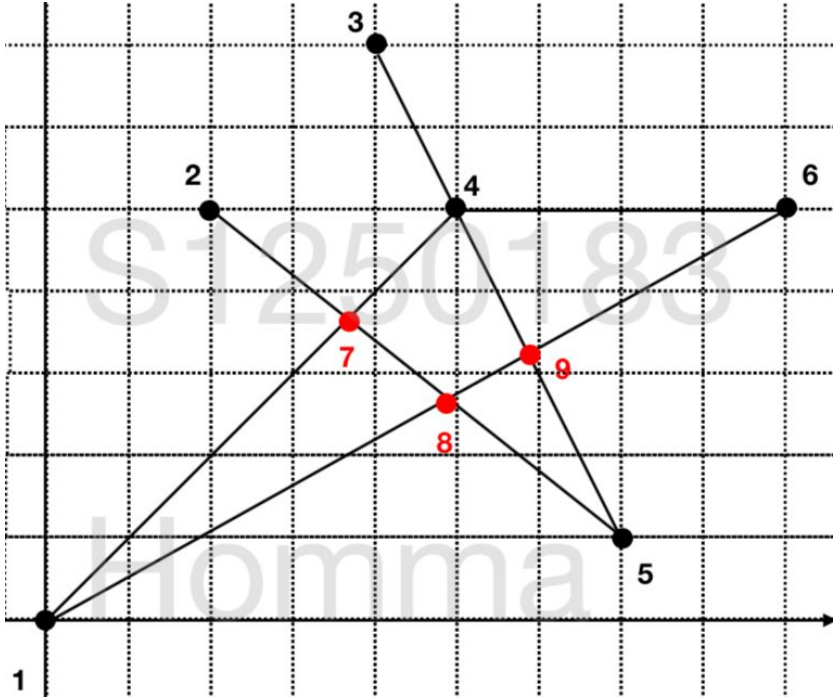
座標構造体をノードとする

-> 開発状況

小課題3,4 最短経路の距離　データ構造

隣接行列

```
double edge[NMAX][NMAX]
```



	1	2	3	4	5	6	7	8	9
1	-1	-1	-1	-1	-1	-1	w	w	-1
2	-1	-1	-1	-1	-1	-1	w	-1	-1
3	-1	-1	-1	w	-1	-1	-1	-1	-1
4	-1	-1	w	-1	-1	w	w	-1	w
5	-1	-1	-1	-1	-1	-1	-1	w	w
6	-1	-1	-1	w	-1	-1	-1	-1	w
7	w	w	-1	w	-1	-1	-1	w	-1
8	w	-1	-1	-1	w	-1	w	-1	w
9	-1	-1	-1	w	w	w	-1	w	-1

小課題5 第k最短路
小課題6 複数経路の提案

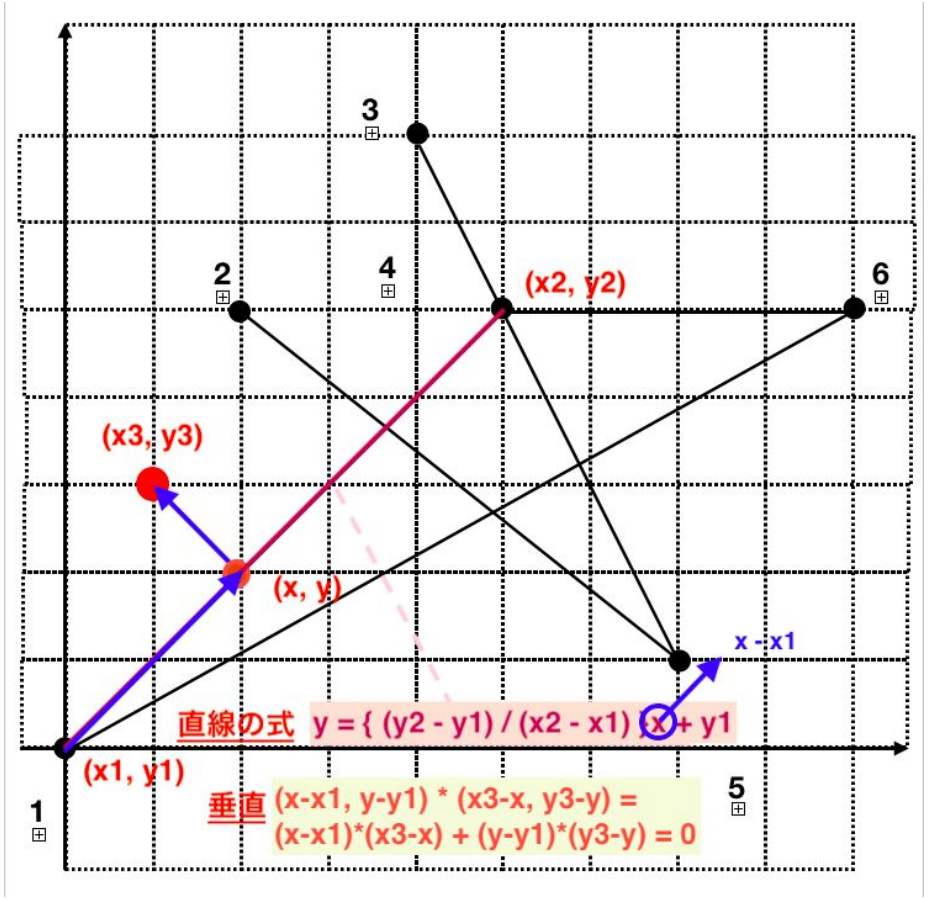
-> 開発状況 小課題5,6

実装中

小課題7 最適な道の建設提案

-> 開発状況 小課題7 最適な道の建設提案　アルゴリズム

ベクトルの内積の考え方を使った



-> 開発状況 小課題7 最適な道の建設提案　アルゴリズム

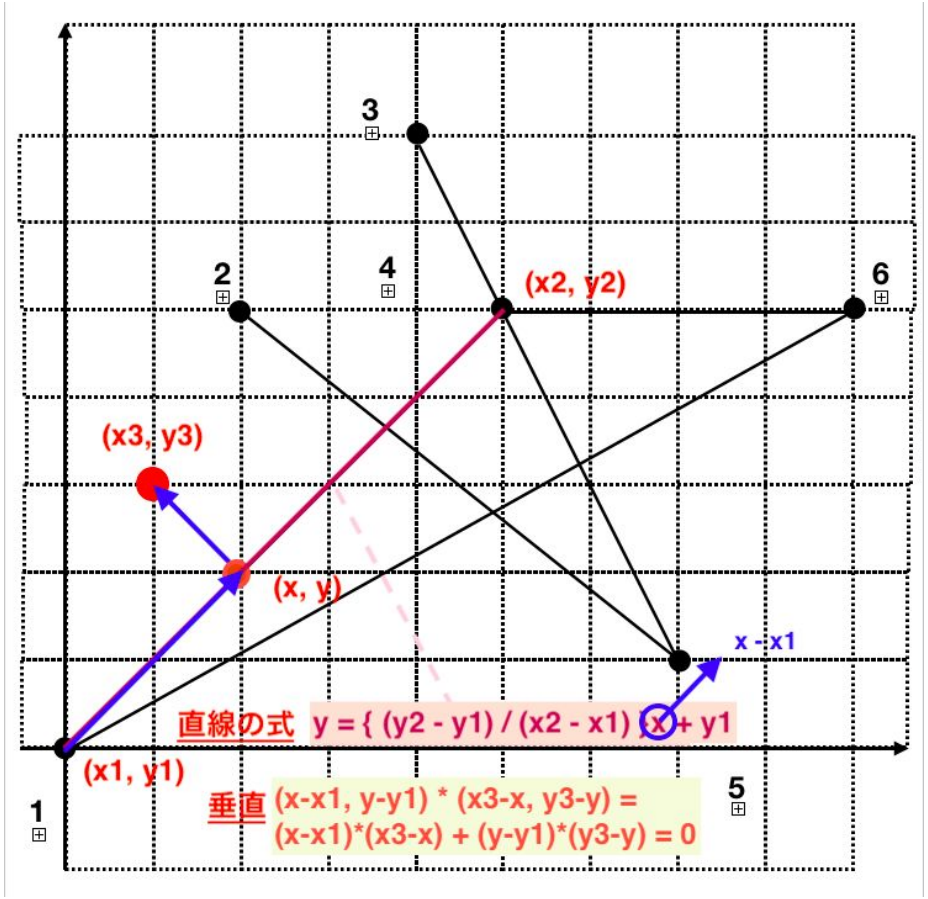
それぞれの線分に対して、

内積の式 = 0　かつ　線分上にある

x, y を求め、

各線分のx, yのうち、
新しい座標との距離が最短である

x, y が最適である



-> 開発状況 小課題7 最適な道の建設提案　アルゴリズム

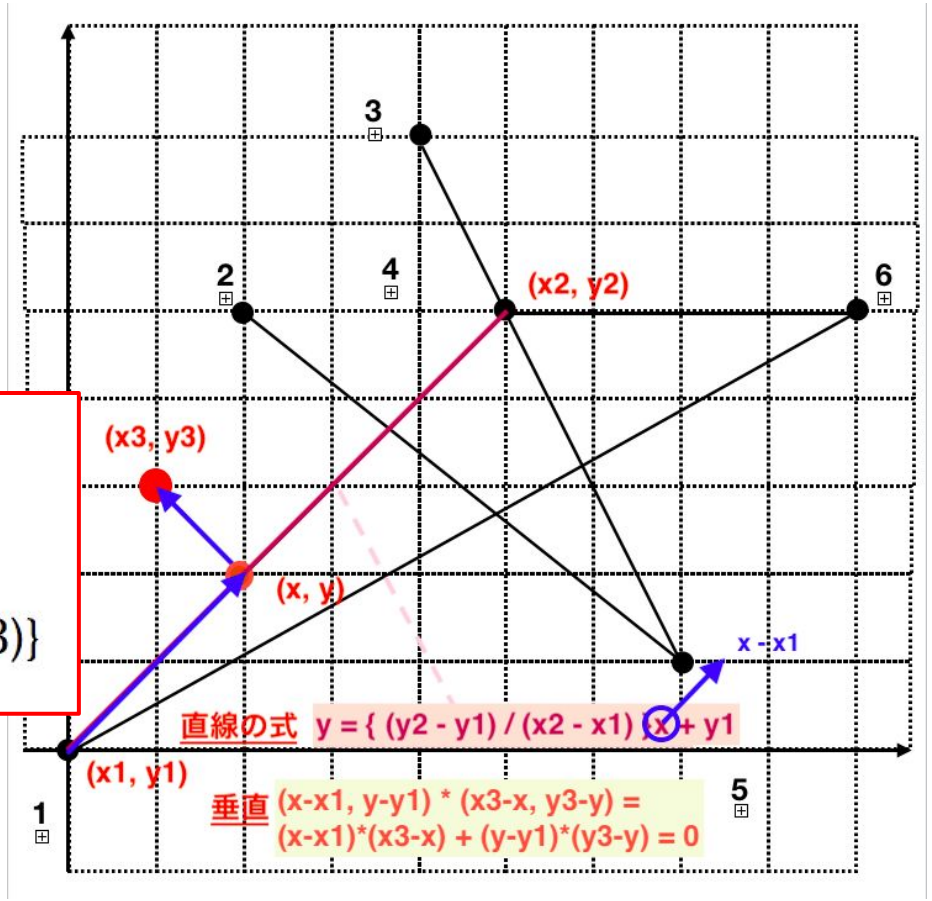
$$y = \frac{(y2 - y1)}{(x2 - x1)}(x - x1) + y1$$

線分の式

$$(x - x1)(x3 - x) + (y - y1)(y3 - y) = 0$$

内積の式

$$x = \frac{1}{(x2 - x1)^2 + (y2 - y1)^2}$$
$$\{(x2 - x1)^2 x3 + (y2 - y1)^2 x1$$
$$- (x2 - x1)(y2 - y1)(y1 - y3)\}$$



-> 開発状況 小課題7 最適な道の建設提案　ソースコード

```
46    /*
47     * 全ての道に対して内積を求め、
48     * 内積が0の時の座標と距離を出す(その道までの最短距離)
49     * そのなかで一番距離が小さいものを採用
50     */
51    for(i = 1; i <= m; i++) {
52      pointPIndex = searchPointIndex(point, n, road[i][0]);
53      pointQIndex = searchPointIndex(point, n, road[i][1]);
54      // x1, y1 は 線分端点Pの座標
55      x1 = point[pointPIndex].x;
56      y1 = point[pointPIndex].y;
57      // x2, y2 は 線分端点Qの座標
58      x2 = point[pointQIndex].x;
59      y2 = point[pointQIndex].y;
60
61      coef1 = pow(x2-x1, 2.0);
62      coef2 = pow(y2-y1, 2.0);
63
64      // 方程式を解く
65      x[i] = (1 / (coef1 + coef2)) * ( coef1 * x3 + coef2 * x1 - (x2-x1)*(y2-y1)*(y1-y3) );
66      y[i] = ( ( (y2 - y1)/(x2 - x1) ) * (x[i] - x1) ) + y1;
67      // 距離を求める
68      distance[i] = sqrt(fabs( (x3 - x[i])*(x3 - x[i]) + (y3 - y[i])*(y3 - y[i]) ));
69    }
70
```

-> 開発状況 小課題7 最適な道の建設提案　ソースコード

```
78    // 最短距離の座標を返す
79    min = INF;
80    minIndex = 0;
81    for(i = 1; i <= m; i++) {
82      if( (min > distance[i]) && (x[i] != x3) && (y[i] != y3) ) {
83        min = distance[i];
84        minIndex = i;
85      }
86    }
87
88    connectPoint.x = x[minIndex];
89    connectPoint.y = y[minIndex];
90
91    return connectPoint;
92  }
```

小課題8 幹線道路の検出

-> 開発状況 小課題8 幹線道路の検出

実装中

3. デモンストレーション

小課題1 交差地点の検出

-> デモ

小課題1 交差地点の検出

テストケース 1

入力

```
4 2 0 0
0 0
5 5
2 5
7 1
1 2
3 4
```

出力

```
3.66667 3.66667
```

-> デモ

小課題1 交差地点の検出

テストケース 2

入力

```
4 2 0 0
0 0
5 5
2 5
7 1
1 3
2 4
```

出力

```
NA
```

-> デモ

小課題1 交差地点の検出

テストケース 3

入力

```
4 2 0 0
5 5
9 5
4 7
7 1
1 2
3 4
```

出力

NA

小課題2 交差地点の列挙

-> デモ

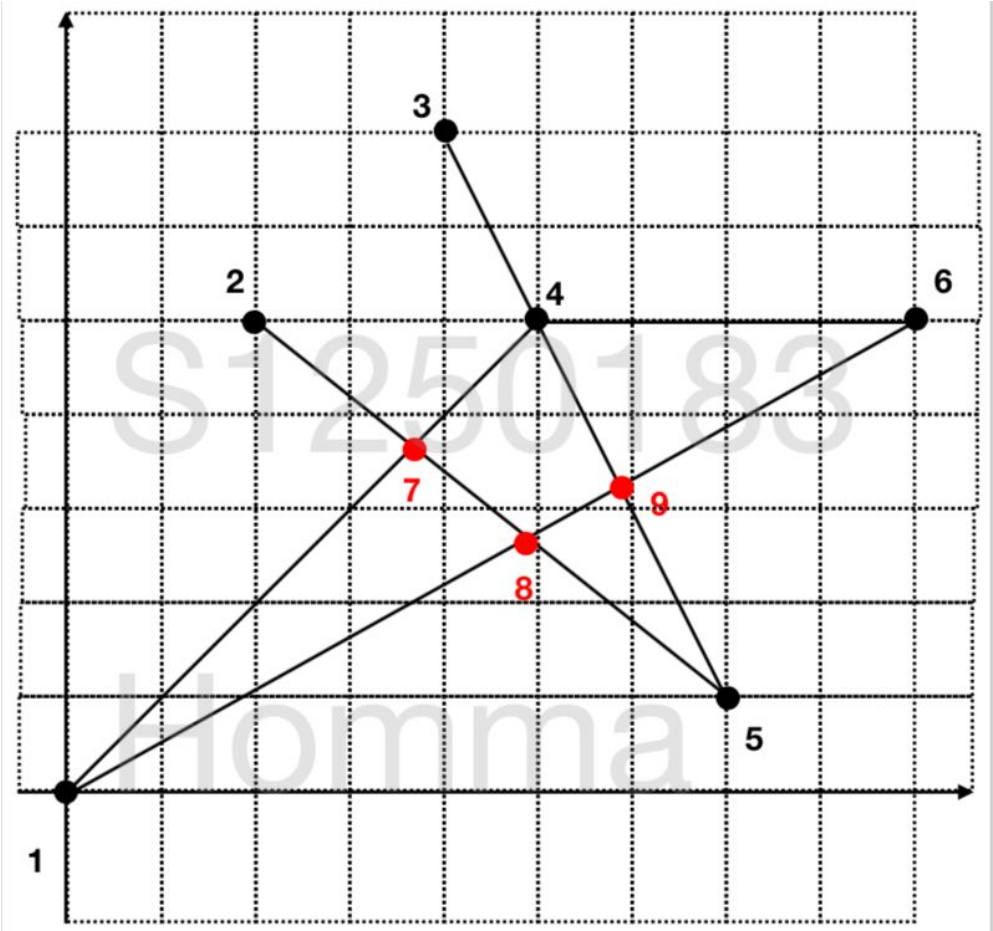
小課題2 交差地点の列挙

テストケース 1

入力
6 5 0 0
0 0
2 5
4 7
5 5
7 1
9 5
1 4
1 6
2 5
3 5
4 6

出力

3.66667 3.66667
4.86885 2.70492
5.86957 3.26087



-> デモ

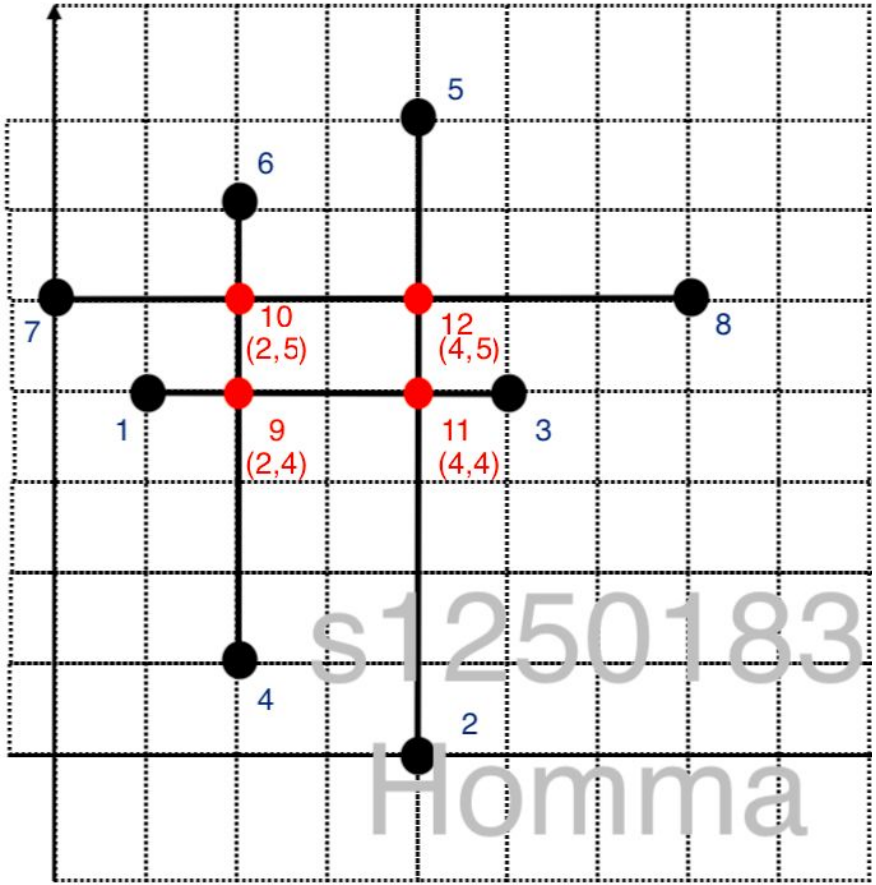
小課題2 交差地点の列挙

テストケース 2(x座標が同じ場合)

入力
8 4 0 5
1 4
4 0
5 4
2 1
4 7
2 6
0 5
7 5
1 3
7 8
6 4
5 2

出力

2.000000 4.000000
2.000000 5.000000
4.000000 4.000000
4.000000 5.000000



-> デモ

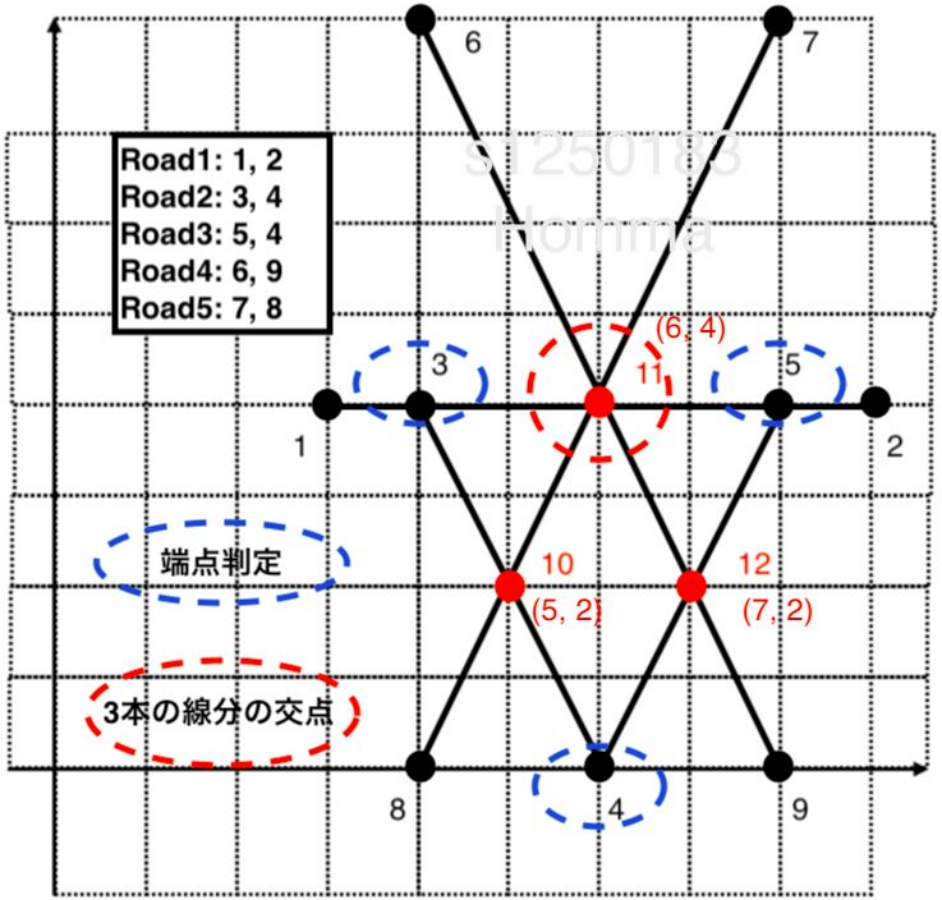
小課題2 交差地点の列挙

テストケース 3(線分3本交差)

入力
9 5 0 0
3 4
9 4
4 4
6 0
8 4
4 8
8 8
4 0
8 0
1 2
3 4
5 4
6 9
7 8

出力

5.000000 2.000000
6.000000 4.000000
7.000000 2.000000



-> デモ

小課題2 交差地点の列挙

テストケース 4(n = 200 m = 100)

入力
200 100 0 0
148 886
107 87
449 387
609 351
…..
184 59
87 59
152 30
198 73
154 31
150 47
121 99
181 45
52 65

出力

???
(random)

小課題3 最短経路の距離

-> デモ

小課題3 最短経路の距離

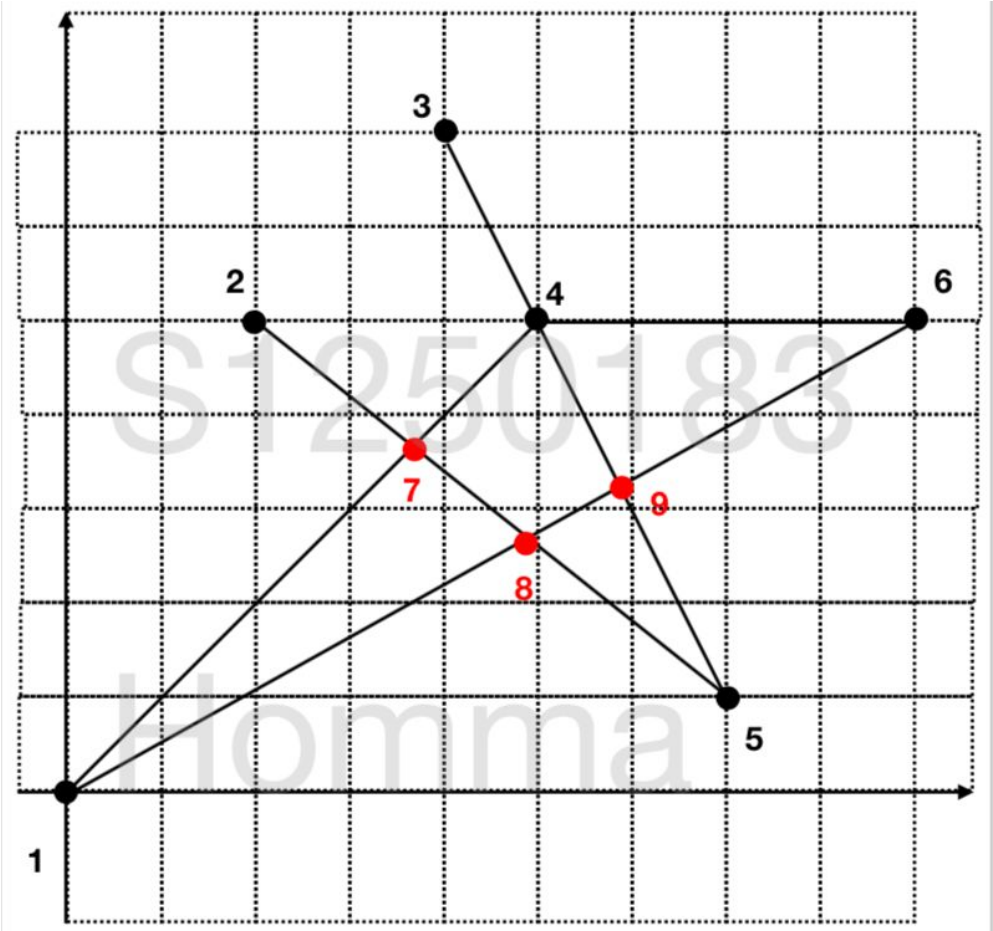
テストケース 1

入力

```
6 5 0 5
0 0
2 5
4 7
5 5
7 1
9 5
1 4
1 6
2 5
3 5
4 6
1 4 1
5 6 1
C1 6 1
C1000 1 1
C1 C3 1
```

出力

```
7.07107
6.10882
5.88562
NA
2.68432
```



-> デモ

小課題3 最短経路の距離

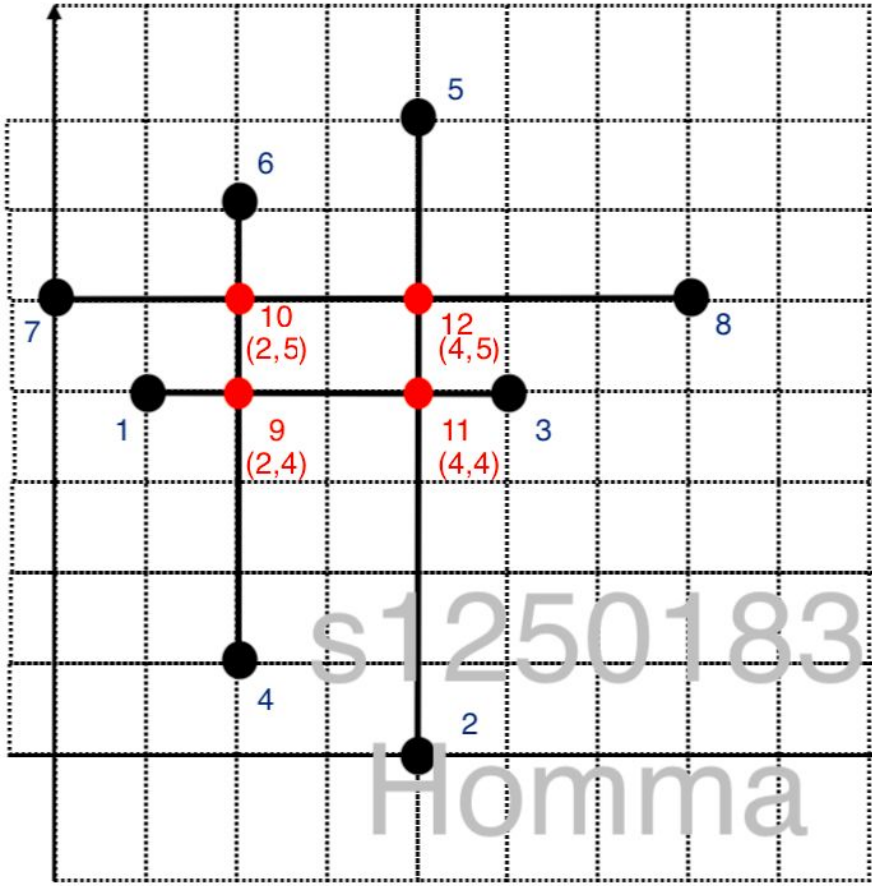
テストケース 2(複数の経路)

入力

```
8 4 0 5
1 4
4 0
5 4
2 1
4 7
2 6
0 5
7 5
1 3
7 8
6 4
5 2
4 5 1
7 C3 1
4 8 1
2 6 1
1 C1 1
```

出力

```
8.000000
5.000000
9.000000
8.000000
1.000000
```



-> デモ

小課題3 最短経路の距離

テストケース 3(端点同士で重なる)

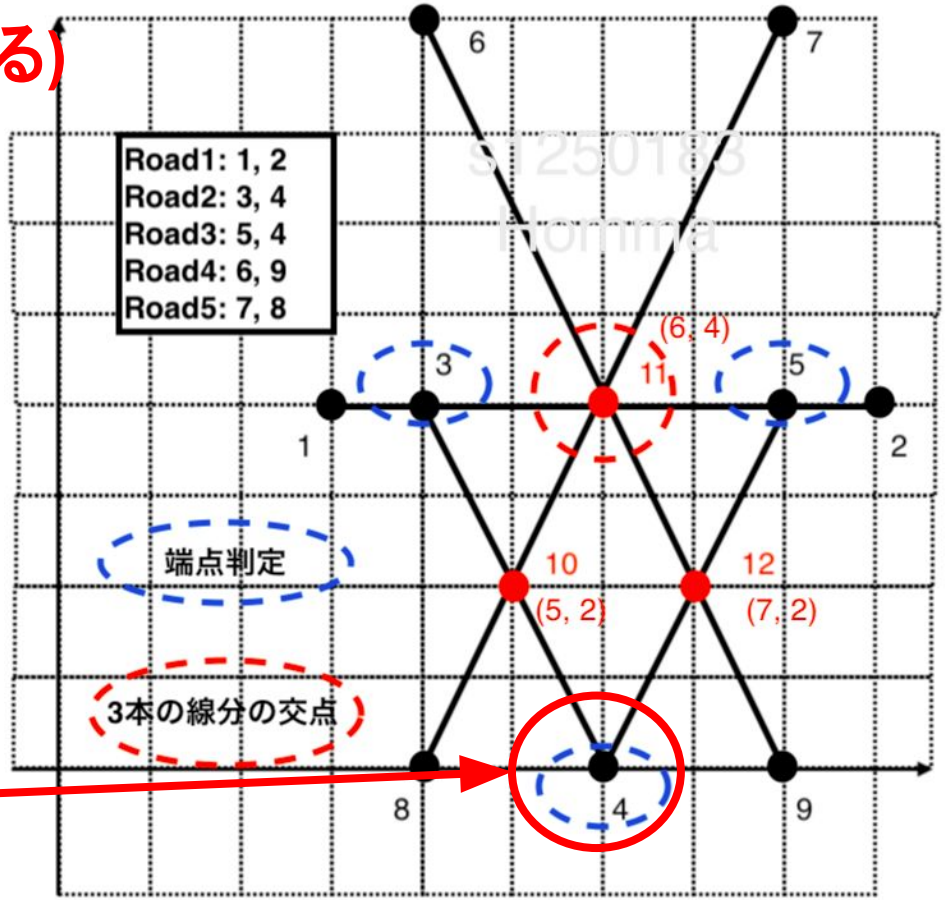
入力
```
9 5 0 4
3 4
9 4
4 4
6 0
8 4
4 8
8 8
4 0
8 0
1 2
3 4
5 4
6 9
7 8
8 7 1
1 2 1
1 6 1
8 9 1
```

出力

8.944272
6.000000
7.472136
8.944272

edge[4][10] と edge[4][12] があるため、繋がっていると判断される

ここが問題！



-> デモ

小課題3 交差地点の列挙

テストケース 4(n = 200 m = 100)

入力
200 100 0 100
239 296
92 163
148 903
578 669
557 656
322 559
…..
371 895 1
917 C589 1
83 C406 1
99 C760 1
C487 542 1
C124 C903 1

出力

???
(random)

メモリが足りない！

小課題4 最短経路

-> デモ

小課題4 最短経路

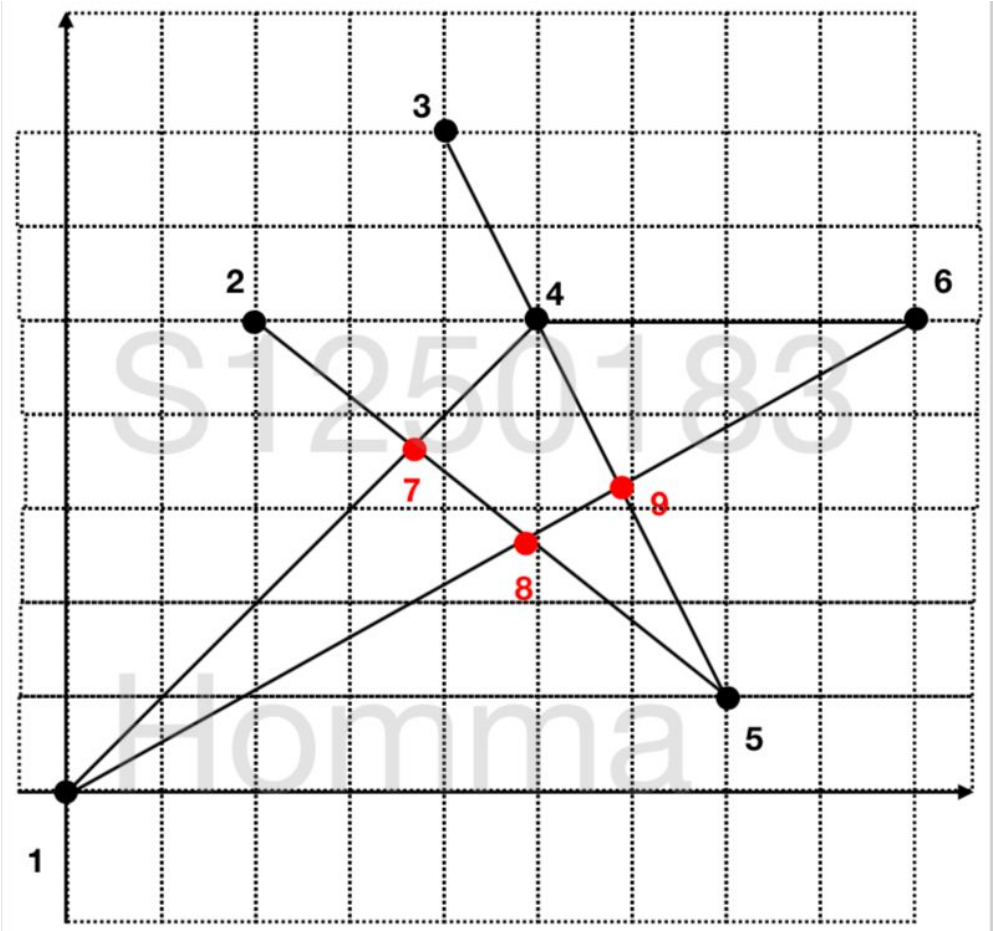
テストケース 1

入力

```
6 5 0 5
0 0
2 5
4 7
5 5
7 1
9 5
1 4
1 6
2 5
3 5
4 6
1 4 1
5 6 1
C1 6 1
C1000 1 1
C1 C3 1
```

出力

```
7.07107
1 C1 4
6.10882
5 C3 6
5.88562
C1 4 6
NA
2.68432
C1 C2 C3
```



-> デモ

小課題4 最短経路

テストケース 2(複数の経路)

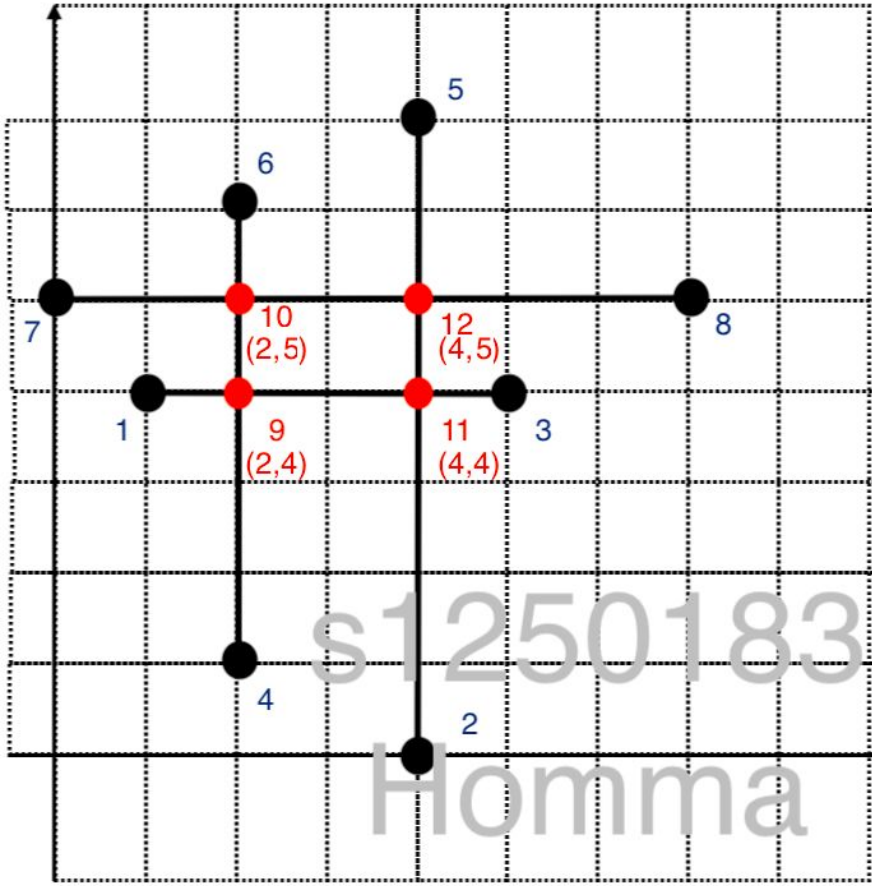
入力

```
8 4 0 5
1 4
4 0
5 4
2 1
4 7
2 6
0 5
7 5
1 3
7 8
6 4
5 2
4 5 1
7 C3 1
4 8 1
2 6 1
1 C1 1
```

出力

辞書順で小

```
8.000000
4 C1 C3 C4 5  or  4 C1 C2 C4 5
5.000000
7 C2 C4 C3  or  7 C2 C1 C3
9.000000
4 C1 C2 C4 8  or  4 C1 C3 C4 8
8.000000
2 C3 C4 C2 6  or  2 C3 C1 C2 6
1.000000
1 C1
```



小課題5 第k最短路
小課題6 複数経路の提案

-> デモ

小課題5,6

実装中

小課題7 最適な道の建設提案

-> デモ

小課題7 最適な道の建設提案

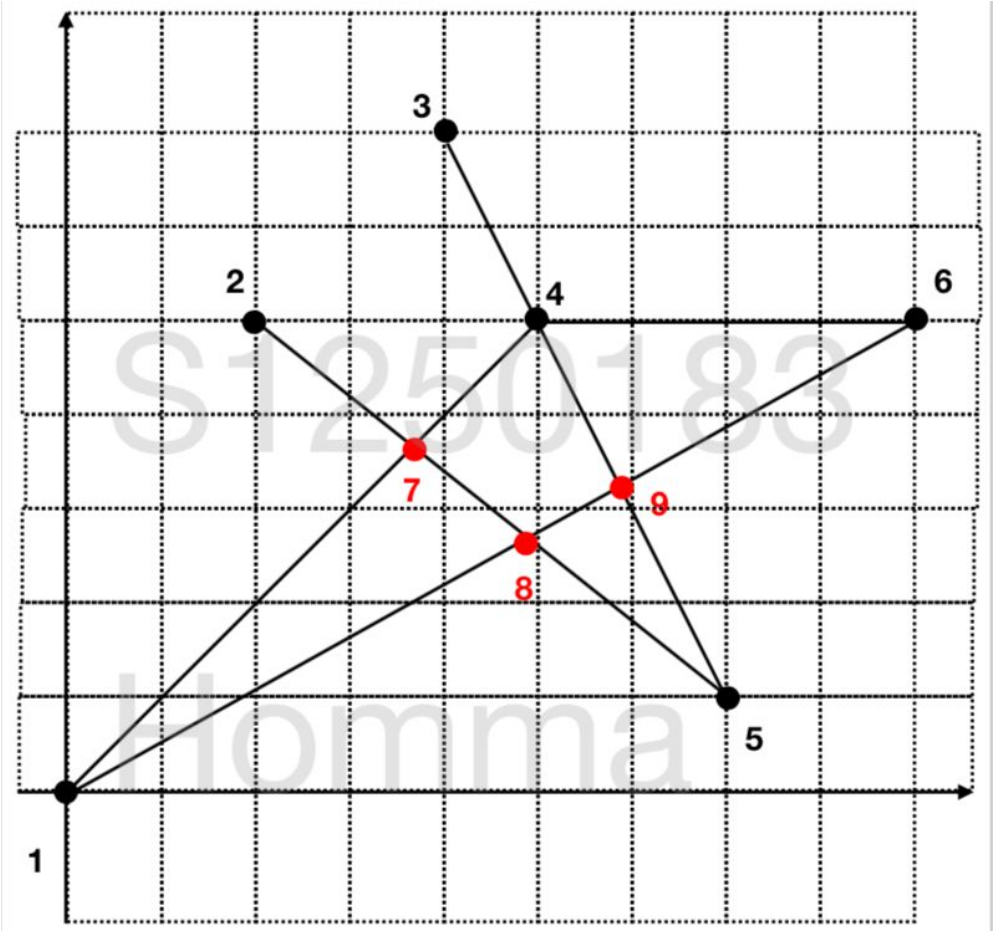
テストケース 1

入力

```
6 5 0 5
0 0
2 5
4 7
5 5
7 1
9 5
1 4
1 6
2 5
3 5
4 6
5 1
11 5
5 4
3 6
```

出力

```
5.78049 1.97561
9 5
5.4 4.2
4.2 6.6
```



小課題8 幹線道路の検出

-> デモ

小課題8 幹線道路の検出

実装中

4. 各メンバーの役割と貢献度

-> 各メンバーの役割と貢献度

s1250125: Takahisa Watanabe

- mainのコーディング
- アルゴリズム設計 (小課題1, 5, 6)
- コードのテスト
- 毎週のレポートの提出

s1250183: Yuki Homma

- 関数のコーディング
- アルゴリズム設計 (小課題2, 3, 4, 7, 8)
- テストケース作成
- 中間発表スライド作成

-> 各メンバーの役割と貢献度

s1250125: Takahisa Watanabe, s1250183 Yuki Homma

	設計(アルゴリズム考案)	実装(プログラミング)	テストケース作成	テスト
小課題 1	s1250125	s1250125	s1250125	s1250125
小課題 2	s1250183	s1250183	s1250183	s1250125
小課題 3	s1250183	s1250183	s1250183	s1250125
小課題 4	s1250183	s1250183	s1250183	s1250125
小課題 5	s1250125	s1250125 (実装中)	s1250125	s1250125
小課題 6	s1250125	s1250125 (実装中)	s1250125	s1250125
小課題 7	s1250183	s1250183	s1250183	s1250183
小課題 8	s1250183	s1250183 (実装中)	s1250183	s1250183

5. 今後の計画

-> 今後の計画

- データ構造を見直す（メモリを節約するため）
- できる限りのテストケースを作成し、小課題1からテストを再びやり直す
- 各小課題でのテストが成功したのちに、次の小課題に取り掛かることを徹底する