


# Integrated Exercise for Software I

## Mid-term Presentations

6.12 (wed)

# 開発環境

- ・チーム名 : powerpuff
  - ・メンバー : s1250110 Mai Kawauchi  
s1250132 Sakurako Kimura  
s1240072 Taiga Takahagi
  - ・開発プラットフォーム : Mac, Ubuntu
  - ・言語 : C++
  - ・ツール : emacs
- 

# 開発状況

- ・実装したアルゴリズムの解説
- ・クラス図等による概要
- ・ソースコード(必要な部分のみ)
- ・作成したテストデータ
- ・データ生成器・検証器

※今プレゼンでは小課題1及び、小課題2ー1の進捗部分までを対象とする



# 実装したアルゴリズムの解説/ソースコード

- ・ダイクストラ法
- ・距離計算
- ・交差判定
- ・Qデータ判定



# ダイクストラ法

参考 <http://nw.tsuda.ac.jp/lec/dijkstra/>

```
//*****  
// 最短経路(Dijkstra)  
//*****
```

```
double dijkstra(int start, int goal)
```

```
{  
    double min;  
    int target;  
    COST[start] = 0;  
  
    while(1){  
        /* 未確定の中から距離が最も小さい地点(a)を選んで、  
        その距離を その地点の最小距離として確定 */  
        min = INF;  
        for(int i = 1; i <= N+r; i++){  
            if( !USED[i] && min > COST[i] ){  
                min = COST[i];  
                target = i;  
            }  
        }  
    }  
}
```

```
/* 全ての地点の最短経路が確定 */
```

```
if(target == goal)
```

```
return COST[goal];
```

```
/* 今確定した場所から「直接つながっている」かつ  
「未確定の」 地点に関して、  
今確定した場所を経由した場合の距離を計算し、  
今までの距離よりも小さければ書き直します。 */
```

```
for(int next = 1; next <= N+r; next++){  
    if(COST[next] >= DIST[target][next] + COST[target]) {  
        COST[next] = DIST[target][next] + COST[target];  
        VIA[next] = target;  
    }  
}  
USED[target] = TRUE;  
}
```

# 距離計算

```
/**
// 距離計算
**/
void Distance(){

    for(int i=0; i<SIZE; i++){
        double dx2 = ((result[list[i].hen[0]-1].rx - result[list[i].hen[1]-1].rx) * (result[list[i].hen[0]-1].rx - result[list[i].hen[1]-1].rx) +
            (result[list[i].hen[0]-1].ry - result[list[i].hen[1]-1].ry) * (result[list[i].hen[0]-1].ry - result[list[i].hen[1]-1].ry));
        //((a-c)^2+(b-d)^2

        list[i].kyori = sqrt(dx2);    //ダイクストラ用の辺の重み

        for(int k=1; k<=N+r; k++){
            if(list[i].hen[0] == k){
                MM[k][list[i].hen[1]] = list[i].kyori;
                MM[list[i].hen[1]][k] = list[i].kyori;
            }
        }    //ex.線分4,6の距離ならMM[4][6]に格納
    }
}
```

# 交差判定

```
/*******
```

```
// 交差判定(crossing detection)
```

```
/*******
```

```
void CrossPoint(){
```

```
    double e = std::numeric_limits<double>::epsilon(); //EPS誤差
```

```
    double A, vx, vy, s, t;
```

```
    int u=0; //線分の組み合わせの数
```

```
    /*(5)式*/
```

```
        for(int i=0; i<M; i++){
```

```
            for(int j=0; j<M; j++){
```

```
                A = fabs((l[i].p2.x - l[i].p1.x)*(l[j].p1.y - l[j].p2.y)
                    + (l[j].p2.x - l[j].p1.x)*(l[i].p2.y - l[i].p1.y));
```

```
                if( -e<A && A<e ){
```

```
                    //cout << "NA" << endl; //2つの線分は触れ合わない
```

```
                }else{
```

```
                    /*(6)式*/
```

```
                    s = ((l[j].p1.y-l[j].p2.y)*(l[i].p1.x-l[i].p1.x)
                        + (l[j].p2.x-l[j].p1.x)*(l[i].p1.y-l[i].p1.y))/A;
```

```
                    t = ((l[i].p1.y-l[i].p2.y)*(l[j].p1.x-l[j].p1.x)
                        + (l[i].p2.x-l[i].p1.x)*(l[j].p1.y-l[j].p1.y))/A;
```

```
                /*(1)(3)式*/
```

```
                if((0<s && s<1) && (0<t && t<1)){ //2つの線分は交わる
```

```
                    /*-----交点の座標を計算してresultに格納-----*/
```

```
                    vx = l[i].p1.x + (l[i].p2.x-l[i].p1.x) * s;
```

```
                    vy = l[i].p1.y + (l[i].p2.y-l[i].p1.y) * s;
```

```
                    result[r+N].rx = vx;
```

```
                    result[r+N].ry= vy;
```

```
                    result[r+N].rn = r+N+1;
```

```
                /*--端点と交点からなる全ての線分をダイクストラ用の辺とする--*/ }
```

```
                list[u].hen[0] = l[i].m1;
```

```
                list[u].hen[1] = l[i].m2;
```

```
                u++;
```

```
                list[u].hen[0] = l[i].m1;
```

```
                list[u].hen[1] = result[r+N].rn;
```

```
                u++;
```

```
                list[u].hen[0] = l[i].m2;
```

```
                list[u].hen[1] = result[r+N].rn;
```

```
                u++;
```

```
                list[u].hen[0] = l[j].m1;
```

```
                list[u].hen[1] = l[j].m2;
```

```
                u++;
```

```
                list[u].hen[0] = l[j].m1;
```

```
                list[u].hen[1] = result[r+N].rn;
```

```
                u++;
```

```
                list[u].hen[0] = l[j].m2;
```

```
                list[u].hen[1] = result[r+N].rn;
```

```
                u++;
```

```
                /*-----*/
```

```
                r++; //交点の数++
```

```
            }else{ //他方の端点がある一方の線分上にある
```

```
                list[u].hen[0] = l[i].m1;
```

```
                list[u].hen[1] = l[i].m2;
```

```
                u++;
```

```
                list[u].hen[0] = l[j].m1;
```

```
                list[u].hen[1] = l[j].m2;
```

```
                u++;
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

# 交差判定

```
/*-----sort-----*/
for(int i=N; i<N+r; ++i){    //x座標を参考にする
    for(int j = i+1; j<r+N; ++j){
        if(result[i].rx > result[j].rx){
            float tmpx = result[i].rx;
            float tmpy = result[i].ry;
            int tmpn = result[i].rn;
            result[i].rx = result[j].rx;
            result[i].ry = result[j].ry;
            result[i].rn = result[j].rn;
            result[j].rx = tmpx;
            result[j].ry = tmpy;
            result[j].rn = tmpn;
        }
    }
}
```

```
if(result[i].rx == result[j].rx){    //x座標が同じならy座標を参考にする
    for(int i=N; i<r+N; ++i){
        for(int j = i+1; j<r+N; ++j){
            if(result[i].ry > result[j].ry){
                float tmpx = result[i].rx;
                float tmpy = result[i].ry;
                int tmpn = result[i].rn;
                result[i].rx = result[j].rx;
                result[i].ry = result[j].ry;
                result[i].rn = result[j].rn;
                result[j].rx = tmpx;
                result[j].ry = tmpy;
                result[j].rn = tmpn;
            }
        }
    }
}

/*-----端点同士からなる線分をダイクストラ用の辺とする-----*/
for(int i=N+1; i<N+r; i++){
    list[u].hen[0]=i;
    list[u].hen[1]=i+1;
    u++;
}
}
```



# Qデータ判定

```
/**
// Qデータ判定
void Judgement(){
    int road[N];    //始点から終点までの道順

    for(int i=0; i<Q; i++){    //Qデータが存在していればflag++
        int flag = 0;
        for(int j=0; j<N+r; j++){
            if(qdata[i].qq1 == result[j].rn) flag++;    //始点あり
            if(qdata[i].qq2 == result[j].rn) flag++;    //終点あり
        }
        if(flag==2) {    //始点終点のQデータが存在する
            /*-----ダイクストラ用に初期化-----*/
            for(int i = 0; i < SIZE; i++) {
                COST[i] = INF;
                USED[i] = FALSE;
                VIA[i] = -1;
                for(int j = 0; j < SIZE; j++)
                    DIST[i][j] = INF;    //道がない状態
            }
        }
    }
}
```

```
/*-----2線分とその重みの情報を格納-----*/
for(int i = 1; i <= N+r; i++) {
    for(int j = 1; j <= N+r; j++){
        if(MM[i][j] != 0) DIST[i][j]=MM[i][j];    //道がなければINFのまま
    }
}

cout<<dijkstra(qdata[i].qq1,qdata[i].qq2)<<endl;

/* 経路を表示(ゴールから) */
int node = qdata[i].qq2;
int noode = 1;
road[0]=node;

while(1){
    node = VIA[node];
    road[noode] = node;
    noode++;
    if (node == qdata[i].qq1) break;
}
```

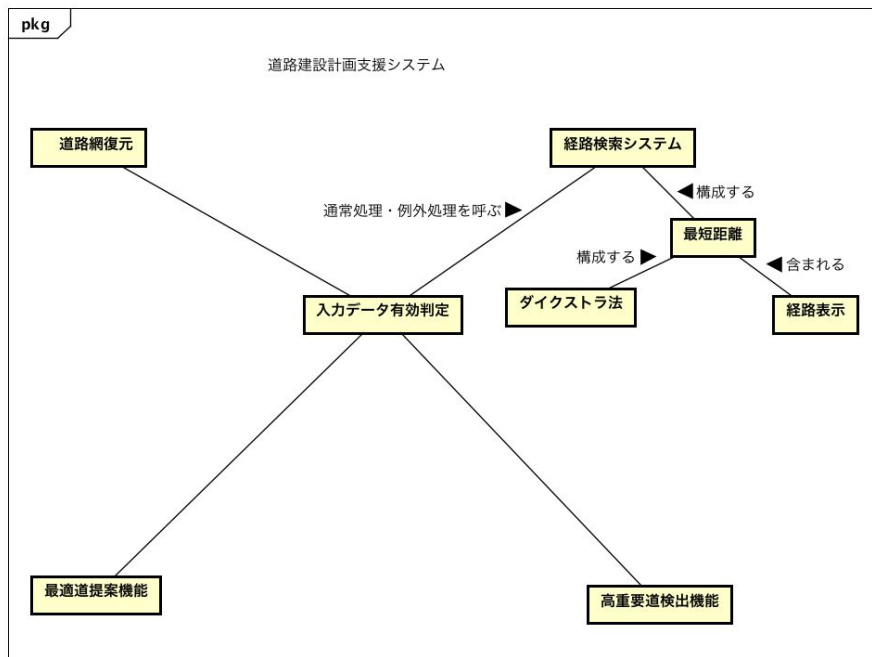
## Qデータ判定

```
for(int a=noode-1; a>=0; a--){
    if(road[a]<=N) printf("%d ", road[a]); //端点表示用
    else printf("C%d ",road[a]-N);      //交点表示用
}
cout<<endl;
}
/*-----*/
else cout << "NA" << endl; //始点終点の2データがない
}
}
```

.....

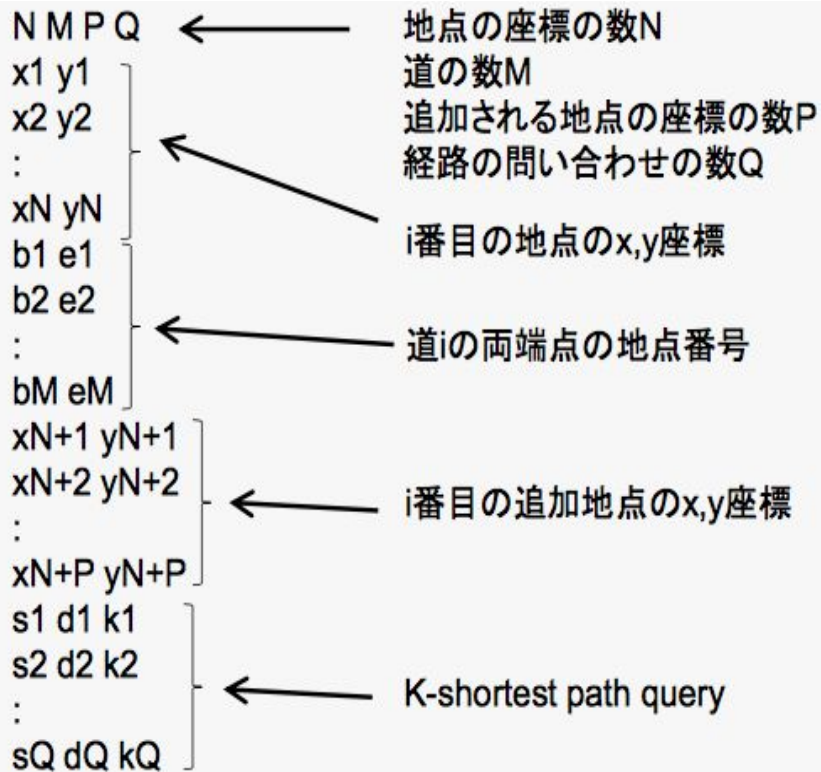
# クラス図等による概要

<http://www.itsenka.com/contents/development/uml/class.html>



※現地点で実装中の経路検索システム  
についてのみの概要とする

# 作成したテストデータ



## 全体の制約

- ・入力は左のような形式で与えられる
- ・入力の座標は全て整数  
(交点の座標は実数になる場合がある)
- ・線分が線で重なることはない  
(端点のみが他の線分と接する場合はある)

# 作成したテストデータ

## TASK 3, 4 制約

$2 \leq N \leq 1000$ ,  $1 \leq M \leq 500$ ,  $0 \leq x, y \leq 10^4$ ,  $P=0$ ,  $0 \leq Q \leq 100$ ,  $1 \leq b, e \leq N$ ,  $b \neq e$

※TASK1, 2についてはTASK3, 4に含まれる中間処理であるため、テストデータ作成及びデモンストレーションを省く

### 通常ケース

1. すべてが最適に交差している

### コーナーケース

1. 線を辿って互いに到達できない地点がある
2. 3つ以上の複数の線が交わる地点や交差地点がある

### min,maxケース

1. (min)交差地点はないが有効なクエリはある
2. (max)地点数、道数共に、またはどちらかが最大数

# データ生成器・検証器

Plot Points -Desmos

<https://www.desmos.com/calculator/mhq4hsncnh>

Generate Data

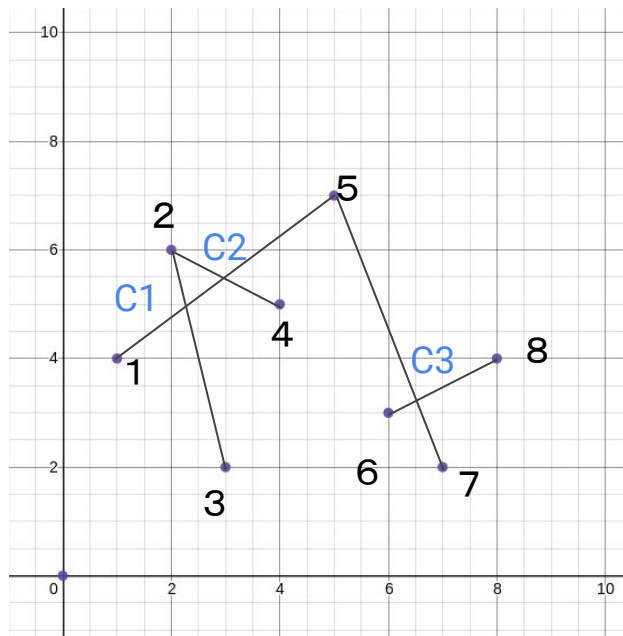
<https://www.generatedata.com/>

The screenshot displays the generatedata.com website. At the top, there's a header with the logo and navigation links: Generate, About, News, and Donate. Below the header, a progress bar indicates "Generated 100 of 100 results". A list of generated data points is shown on the left, with columns for x and y values. On the right, a sidebar menu is visible with options like "アカウント作成" (Create Account) or "サインイン" (Sign In), and "グラフを保存しましょう！" (Save your graph!). Below these, there's a section for "新しい空のグラフ" (New empty graph) and a list of graphing options: Lines (Slope Intercept Form, Point Slope Form, Two Point Form), Parabolas (Standard Form, Vertex Form), and Trigonometry (Period and A...). The main content area on the right shows a "Plot Points" interface with a table of data points.

$x_1$	$y_1$
0	0
1	4
4	5
5	7
6	2
4	2
8	4
2	1
2	6

# デモンストレーション(通常ケース)

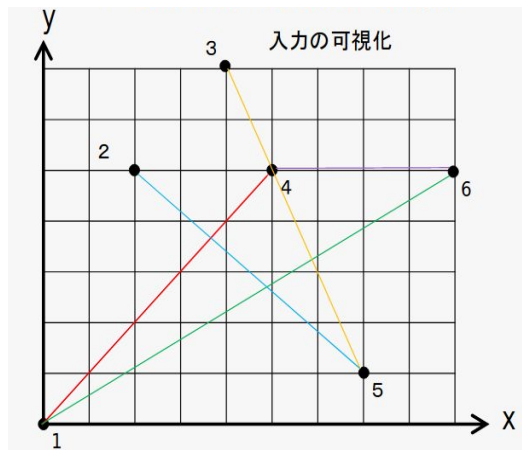
1. すべてが最適に交差している



最短距離とその経路を求める

入力: 1→C1  
7→C2  
3→4

※地点4(端点が線分上にある)  
のような点は含まない



期待する出力

1.57895

1 C1

8.11436

7 C3 5 C2

5.07716

3 C1 C2 4

実際の出力

1.57895

1 C1

5.27886

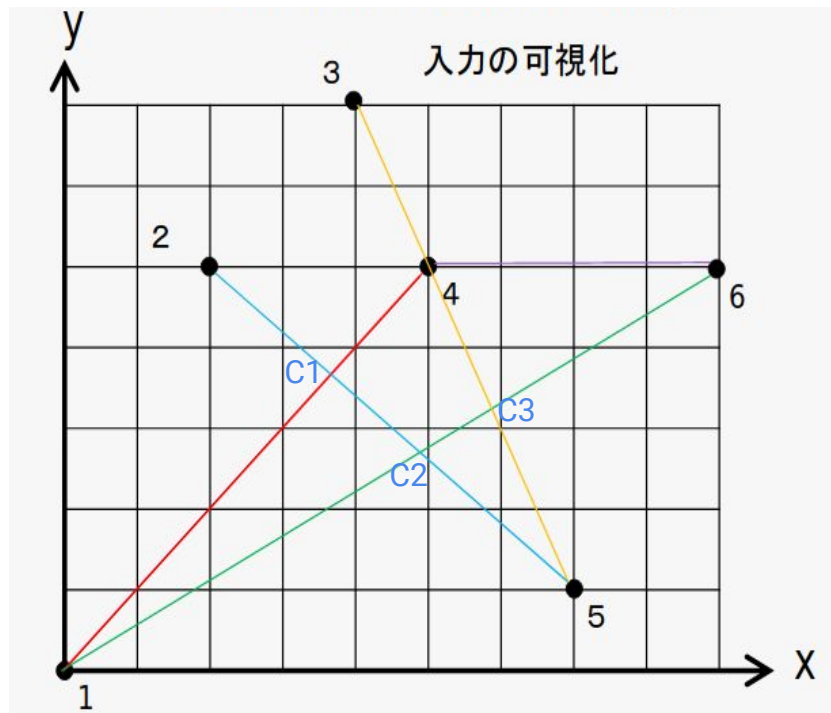
7 C3 C2

5.07716

3 C1 C2 4

不一致!?

# デモンストレーション



## <問題点>

この Sample Input を満たすためにプログラムを書いてしまったため、ほかの要件を満たさない場合が出てしまった。

- ・地点と交点の距離を出すために、地点—地点が出ない。

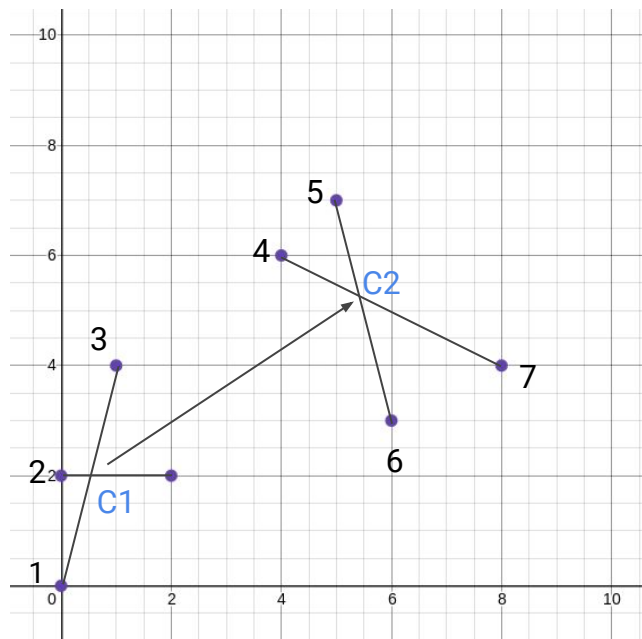
- ・交点—交点の距離について  
CN と CN+1 はつながっているとして距離を出した。  
CN と CN+2 はつながっていない判定

以下のテストケースでうまくいかないのはこれらが原因と考察。



# デモンストレーション(コーナーケース)

1. 線を辿って互いに到達できない地点がある

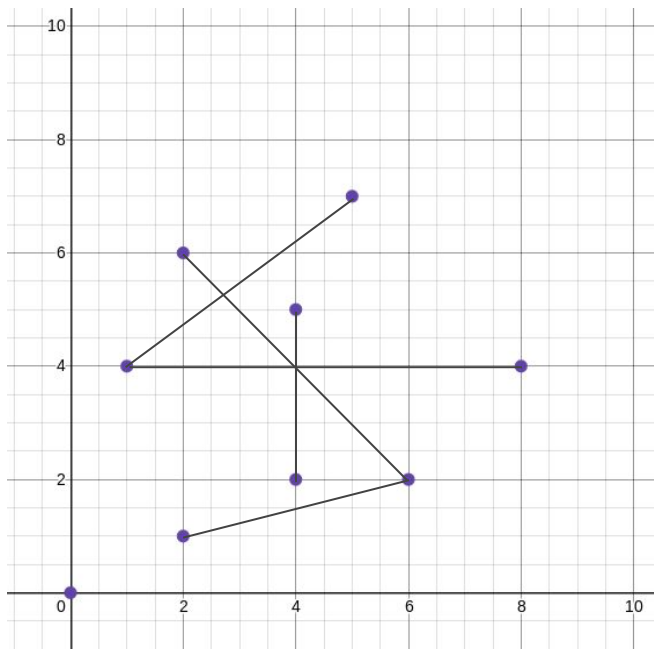


## ＜問題点＞

交差点と交差点を辺として設定してしまったため、到達できないはずの地点に到達できてしまう。

# デモンストレーション(コーナーケース)

2. 3つ以上の複数の線が交わる地点や交差点がある



理想の出力

4

1 C4 5

**5.65685**

3 C1 C2 7

**9.77996**

2 7 C2 C1 3

一致！！

実際の出力

4

1 C4 5

**5.65685**

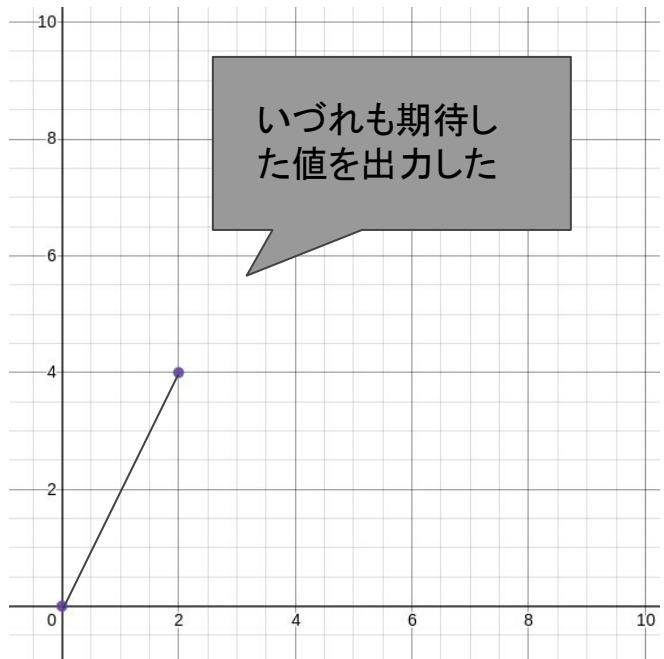
3 C1 C2 7

**9.77996**

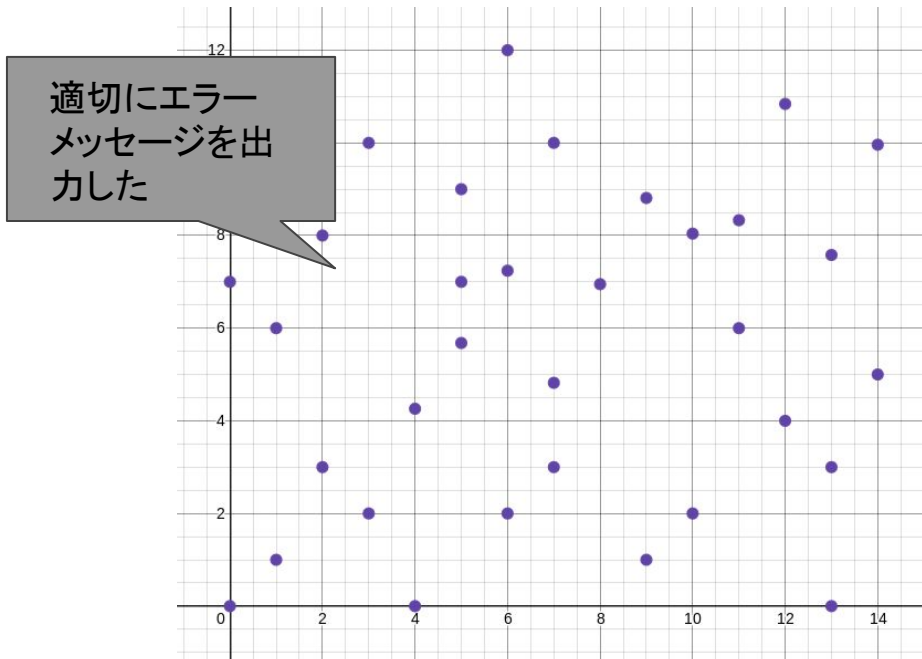
2 7 C2 C1 3

# デモンストレーション(min,maxケース)

1. (min)交差点はないが有効なクエリはある



2. (max)地点数、道数共に、またはどちらかが最大数



# 各メンバーの役割と貢献度

s1250110:

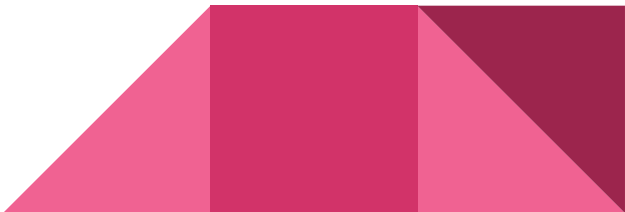
- ・コード行数 116/370  
(主に交差判定と距離計算・ソートについて )
- ・テストデータ・入出力例・サンプル座標グラフの作成  
(通常処理・コーナー処理について )
- ・テスト結果の考察  
(s1250132と同様)

s1250132:

- ・コード行数 254/370  
(主に最短距離・ダイクストラ法と経路表示について )
- ・テスト結果の考察  
(重み付きグラフの出次数決定時に、  
①交点同士を辺とみなしてしまっているため、余分な経路が含まれてしまう。  
②端点が線分上にある場合、交点になれず、必要な経路が含まれない。 )

s1240072:

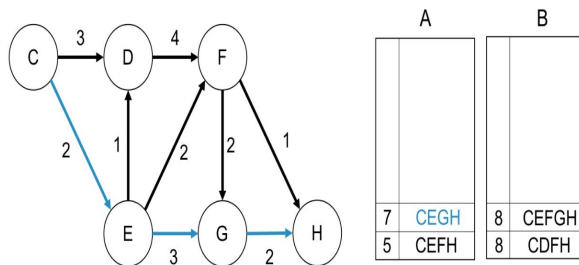
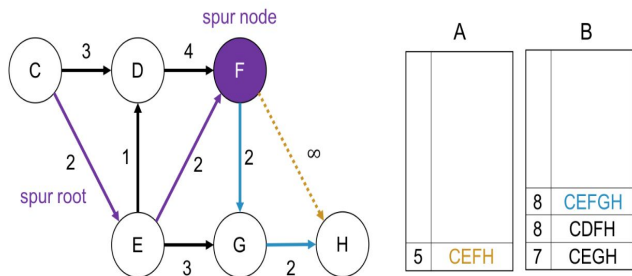
- ・アルゴリズムについての情報収集
- ・プレゼン資料のテンプレート作成



# 今後の計画

- ・小課題1について、重み付きグラフの修正
- ・小課題2について、k-最短経路問題の解決

参考: yenのアルゴリズム <https://qiita.com/nariaki3551/items/821dc6ffdc552d3d5f22>



第n最短経路を元に、spur-rootとspur-nodeをシフトしながら配列 Bに第n+1最短経路の候補を格納していく

配列Bの中から、最も適切な経路を取り出し、配列Aに格納する。