ソフトウェア総合演習I
中間レビュー

チーム

（仮）

各メンバーの役割と貢献度

- S1250111 杵鞭　俊
- 司令塔兼サポート。プログラムの間違いがあれば指摘した。

- S1250081 水口　裕貴
- メインでプログラムを作成。チームの要

- S1250178 田仲　早紀
- lldbなどのデバッグツールなど、便利な知識を提供

開発環境

プラットフォーム:演習室のsolarisをメインにしたLinux環境
　　足りない機能を補うためにmacも使用。
プログラム言語:主にC言語を使用。
　　理由は速度が速く、デバッグも容易なため。
ツール:テキストエディタは基本EMACS、状況に応じてlldbなどのデバッグツールを使用。
　　そのほかにも、github等を使用してソースコードやレポートなどのファイルを共有。

進捗状況

現在Phase1まで完了。
Phase2からは現在デバック作業に入っており、「不具合なし」と判断でき次第githubに公開する予定。

Phase1の要件をすべて満たしたプログラムは、一つのファイルにまとめており、次のスライドに必要な部分のみ抜粋し、ソースコードを添付する。

Phase1のソースコード

行列式やノードなど、「関連性のあるデータの塊」は構造体で表現。
それにより、単なる配列や変数を使うよりも開発やデバッグを容易にする。

```
typedef struct
{
int edges_to[1000];
double edges_cost[1000],cost;
int n_edges, done, from;
} node; //重み付きグラフのノード
typedef struct
{
double A1;/*行列式|A|を表す*/
int p1;
int q1;
int p2;
int q2;
int P1[2];
int P2[2];
int Q1[2];
int Q2[2];
int flag;
} Check;
```

アルゴリズムの解説

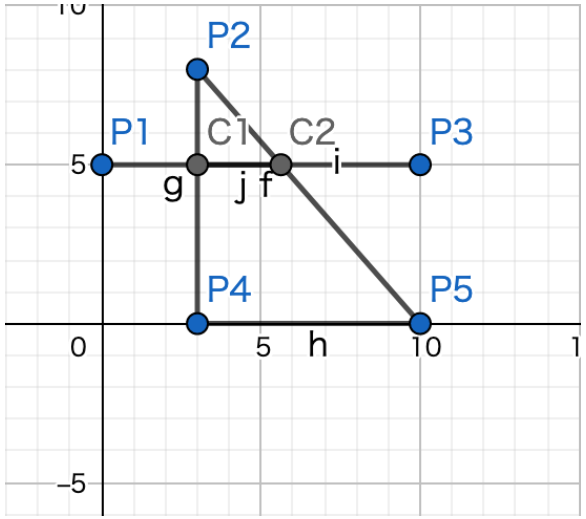
Phase1の小課題で培ってきたアルゴリズムをただ並べるのではなく、さらにそれぞれを細かく分割し、関数にして組み合わせてつかうことでデバッグを容易にし、さらに後のPhase2やPhase3でも応用できるようにする。

交点を並び変える関数や、最短経路を探索する関数など。

そのほかにも、道が交わるか交わらないかをフラグで表現する、最短経路を求めるためにダイクストラ法を用いたなどの工夫がある。

テストデータ(1)

通常のパターン
入力
5 4 0 1
0 5
3 8
10 5
3 0
10 0
1 3
2 4
2 5
4 5
1 4 1



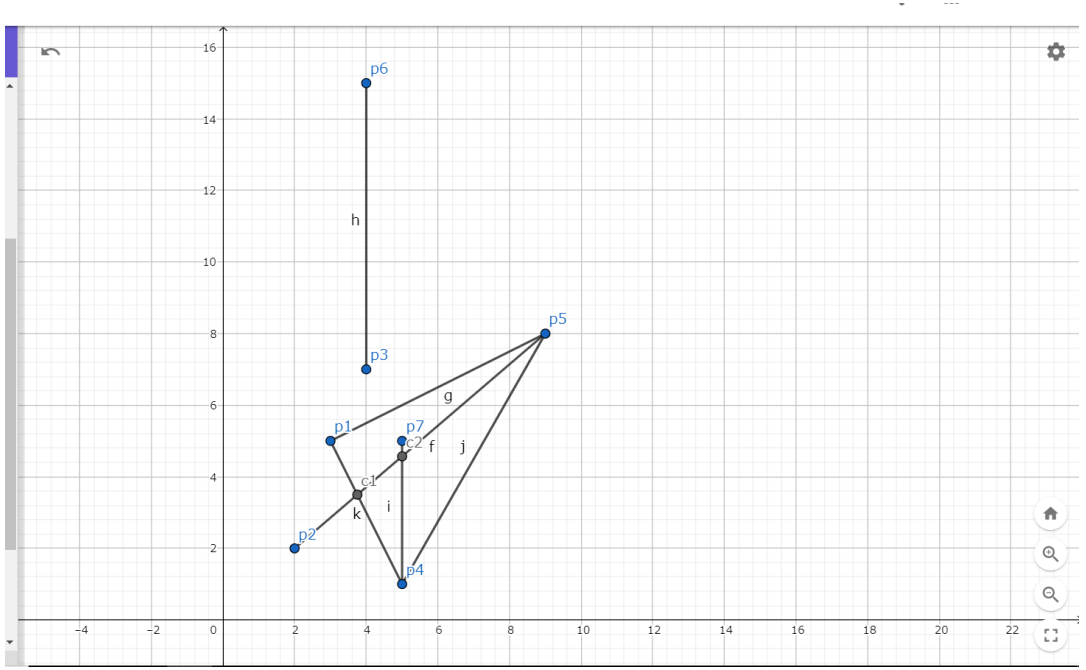
出力
最短経路:8.00000
経路:1-> C1-> 4

結果
通常のケースの場合、滞りなく実行できる。

テストデータ(例外的1)

例外的な入力パターン

7 6 0 3
3 5
2 2
4 7
5 1
9 8
4 15
5 5
1 5
1 4
2 5
3 6
4 7
4 5
6 7 1
2 5 1
4 1 1

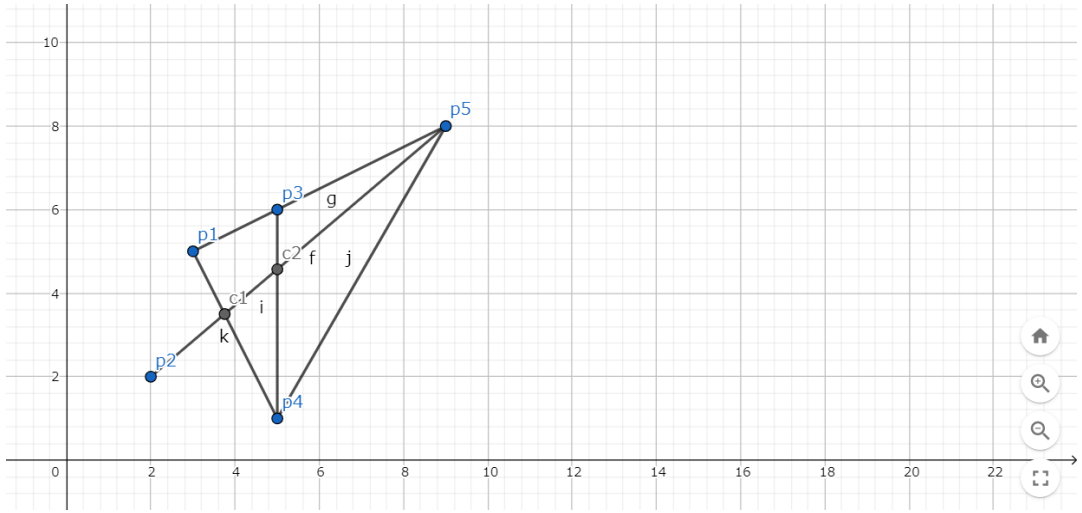


互いに到達できない地点が一つでもあると、全ての出力がNAになる。

出力
NA
NA
NA

テストデータ(例外的2)

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



一つでも三つ以上の複数の点が交わる場所があると、交差地点の検出がうまくできなくなってしまう。

出力
経路：1->4->C1->2
経路：2->C1->5(C2を通っていない)

以上、成功した通常のテストデーター一つと、課題の残るテストデータ二つをしょうかいしました。
これからの課題として、これら二つの例外的処理をうまくできるようにデバッグする必要があります。

今後の計画

当面の課題として、Phase2のデバッグを終わらせる。
それと並行し、Phase1で、例外処理のデバッグを終わらせる。
それが完了次第、Phase3の開発に移る。