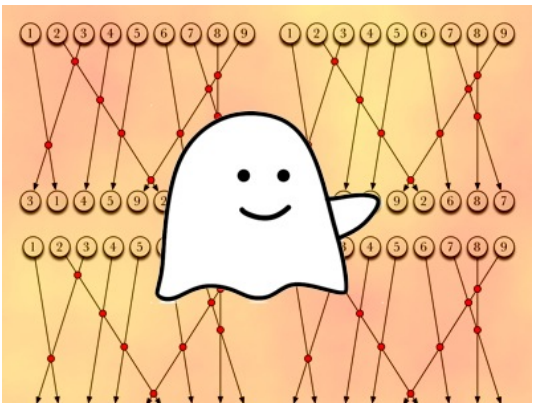
クロッシング問題

http://www.hyuki.com/codeiq/

2013 年 8 月



1　概要

あなたはクロッシング社のプログラマとして雇われました。あなたの仕事は、研究開発している光学素子（クロッシング素子）の性能測定です。クロッシング素子内部では数多くの光線が飛び交って情報伝達を行いますが、**光線 2 本の交差点の個数**が性能に大きく影響を与えます。あなたにはクロッシング素子内部で光線がどのように飛んでいるかのデータファイルが与えられます。そのデータファイルから、**何個の交差点があるかを求めるプログラムを作る**のがあなたのミッションなのです！

2　会話

あなた「仕事は光学素子の性能測定と聞きました。私はあまり物性には詳しくないのですが……」

依頼者「順を追って説明しましょう。わがクロッシング社ではまったく新しい光学素子である『クロッシング素子』を研究開発しています。その素子の内部では数多くの光線が飛び交って情報伝達を行います」

あなた「はい」

依頼者「高速で正確な情報処理に使える素子なのですが、詳しい原理は省略します。あなたの仕事に関係するクロッシング素子の重要な性質は**『光線 2 本が交差すると性能が劣化する』**という点です」

あなた「なるほど」

依頼者「だから素子の設計段階で、**光線 2 本が作り出す交差点の数**を把握しておくことはたいへん重要なのです」

あなた「はい、よく理解できます」

依頼者「実は設計のかなりの部分はプログラムが行います。何百種類、何千種類というクロッシング素子のプロトタイプを作り、その性能をシミュレーションし、どの設計を選ぶべきかを自動的に判断します」

あなた「すばらしいですね！」

依頼者「ところが問題が発生しました。クロッシング素子が複雑になるにつれ、交差点の数を数えるプログラムの遅さがボトルネックになってきたのです。わがクロッシング社のプログラマではもう対処できません。そこで……」

あなた「私にお声がけくださったということですね」

依頼者「そういうことです」

あなた「交差点の数を数えるプログラムを作る……そこまではいいのですが、念のために例を挙げていただけますか」

依頼者「たとえば図 1 をごらんください（p. 3）。上に 1 から 9 までの番号が一直線に並び、下にも番号が一直線に並んでいます。そして、同じ番号の間に矢印が引かれています。これが光線です」

あなた「矢印が光線」

依頼者「そして、矢印が交わっている部分に●印が置かれていますが、これが問題となっている交差点です」

あなた「●印が交差点……この個数が問題なのですね」

依頼者「そうです。図 1 では交差点が 9 個あります」

あなた「はい、わかります」

依頼者「クロッシング素子の内部はもっと複雑なのですが、あなたにお願いしたいのは、このような状況で交差点の個数を求めるプログラムを作ることです。この図 1 に示した例の場合、あなたには図 2 のようなファイルが与えられます。ファイルの各行に番号が書かれています」

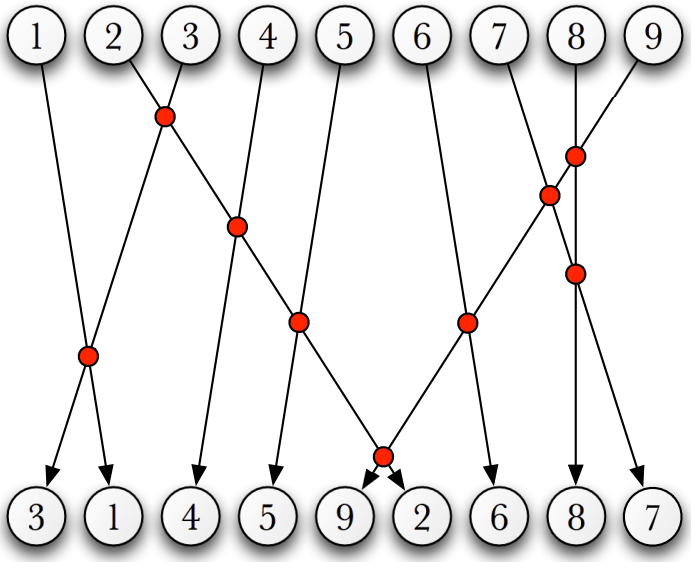


図 1　光線が 9 点で交差する例

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

図 2　例の場合に与えられるデータファイル (`sample.txt`)

あなた「なるほど。光線の行き先の番号がデータファイルとして与えられるわけですね」

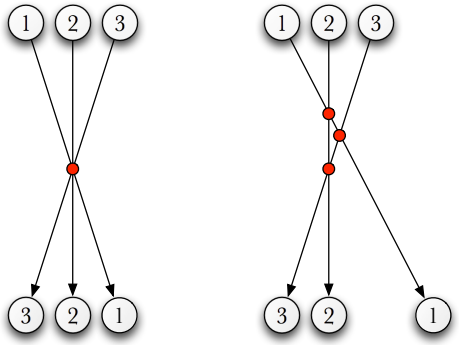
依頼者「そうです。この場合は 9 本の光線なので 9 行のファイルが与えられます。もしも n 本の光線の場合には、n 行からなるファイルが与えられます。そしてそのファイルには 1 から n までの番号が一つずつ含まれていることになります」

あなた「わかりました。さっそくプログラミングに掛かりたいと思います」

依頼者「重要な注意があります。光線が 3 本以上重なる場合です」

あなた「おっと、確かに」

依頼者「たとえば、図 3 のようになる場合には、交差点は 1 個ではなく 3 個と数えてください。つまり、た



1 個は誤り　　**3 個が正しい**

図 3　交差点は必ず 2 本の光線で判断すること

とえ光線が 3 本以上重なった場合でも、交差点はあくまで 2 本の光線ごとに 1 個と数えるのです。図 3 の右側のようにちょっとずらした様子を考えていただければわかると思います」

あなた「なるほど。それはプログラムがシンプルになりそうで助かります。ところで今回の報酬はどうなりますか」

依頼者「いうまでもないですが、与えられたデータファイルに対して**正しい交差点の数を得ること**は必須事項です。正しい交差点の数が得られないなら最低の**評価 1** になります。その場合、報酬はありません」

あなた「はい、わかりました」

依頼者「それからもう一つ。肝心なポイントとして**性能評価**があります。お渡しする本番のデータファイル（`crossing.txt`）から正しい交差点の数を得るまでの**実行時間は 3 秒を切ること**。実行時間は CPU 時間ではなく実時間 (wallclock time) で、あなたのマシンで計ってください。要するにプログラムの最初と最後で現在時刻を得て、その差を計算するということです」

あなた「はい、わかりました……ちょっと待ってください。自己申告でかまわないのですか」

依頼者「かまいません。信用しておりますから。実行時間が 3 秒を切った場合には最高の**評価 5** になります。実行時間が 3 秒以上になる場合には正しい交差点の数を得るプログラムでも**評価 3** になります」

あなた「測定の精度は……？」

依頼者「精度ですか？ 標準的なライブラリを利用して、秒単位の計測をしていただければ結構ですよ。何度か実行して、一度でも実行時間が 0 秒か、1 秒か、2 秒のいずれかに収まればよろしい。当然ながら、複数回実行するときでも毎回データファイルから計算するのですよ。何度やっても 3 秒以上かかるなら、正しい交差点の数を得るプログラムでも**評価 3** どまりです」

あなた「よくわかりました」

依頼者「それではこちらが本番のデータファイル `crossing.txt` になります」

あなた「あ、意外に小さいですね。これならすぐにできそうです」

ミッションはこのようにして始まりました……。