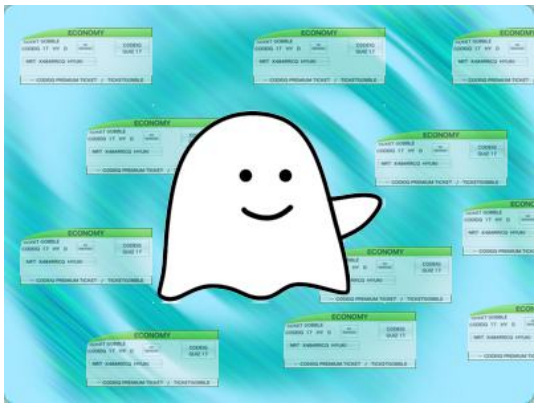
チケットゴブル問題

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

2014 年 5 月



目次

1 概要

あなたはチケットゴブル社にプログラマとして雇われました。チケットゴブル社は、国際線の安売り航空チケットを販売している会社で、格安な世界旅行プランとセットにして個人向けのプロモーションをしています。

あなたの目の前に並べられたのは、世界各国に旅行ができるたくさんの航空チケット。この航空チケットを使って `できるだけ多くの国に旅行するプラン` を立てることがあなたのミッションなのです！

2 会話

あなた「旅行プランを立てる仕事と聞きましたが、いったいどういうことでしょうか……」

依頼者「順を追って説明しましょう。わが社では、個人向けに格安な世界旅行プランを提案しています」

あなた「はい」

依頼者「国際線の航空チケットは期間に自由度が高ければ値段も高く、期間が限定されていれば値段は安くなります。それはご存じですね」

あなた「それはそうでしょうね。いつでも乗れる航空チケットは便利ですから、それだけ高くなる」

依頼者「そういうことです。ですから、わが社が扱う航空チケットはすべて、日本から出発する日付と、日本に戻ってくる日付が定まっています」

あなた「なるほど。行く日と帰る日が決まっている」

依頼者「そこで、あなたのお仕事になるわけですが、航空チケットを複数まとめてパックにし、個人のお客様向けの世界旅行を《プラン》として提案していただきたい」

あなた「はあ……たとえば《ヨーロッパ十日間の旅プラン》のようなものを？」

依頼者「違います。わが社が求めているのは、 できるだけ多くの航空チケットをさばく プランです」

あなた「いや、ちょっと意味がわかりませんが」

依頼者「具体的にお話ししましょう。たとえば、ここに 5 枚の航空チケットがあります（five.txt）」

```
Austria 1/1-1/5
Belgium 1/6-1/10
Canada 1/7-1/8
Denmark 1/9-1/10
Egypt 1/18-1/30
```

あなた「これはオーストリア(Austria)、ベルギー(Belgium)、カナダ(Canada)、デンマーク(Denmark)、それからエジプト(Egypt)の各国に、それぞれ日本から往復するための航空チケットということでしょうか」

依頼者「そうです。一つの国に対して、航空チケットは一枚ずつ。そしてそれぞれの航空チケットは出国日と帰国日が明記されています。この 5 枚の場合、航空チケットの期間は以下のように読みます。年は省略してありますが、いずれも同じ年です」

```
Austria 1/1-1/5       1 月 01 日出国→ 1 月 05 日帰国
Belgium 1/6-1/10      1 月 06 日出国→ 1 月 10 日帰国
Canada 1/7-1/8        1 月 07 日出国→ 1 月 08 日帰国
Denmark 1/9-1/10      1 月 09 日出国→ 1 月 10 日帰国
Egypt 1/18-1/30       1 月 18 日出国→ 1 月 30 日帰国
```

あなた「ちょっと待ってください。おかしいですよ。だってこの旅程では、1 月 7 日から 1 月 8 日まで、Belgium と Canada に同時に滞在することになってしまいます。期間がだぶっています！」

依頼者「ですから、この航空チケットからプランを作るのです」

あなた「わかりました。 期間がだぶらないように航空チケットを選ぶ わけですね……こうでしょうか」

```
Austria 1/1-1/5       1 月 01 日出国→ 1 月 05 日帰国
Belgium 1/6-1/10      1 月 06 日出国→ 1 月 10 日帰国
Egypt 1/18-1/30       1 月 18 日出国→ 1 月 30 日帰国
```

依頼者「確かにこれですと、期間にだぶりはありません。しかしながら、これでは航空チケットが 3 枚しか売れません。Belgium をあきらめて、Canada と Denmark を入れれば 4 枚の航空チケットが売れます。そうですね。3 枚よりも 4 枚の方が多い。 航空チケットの枚数が多い方がいいプラン です」

```
Austria 1/1-1/5        1 月 01 日出国→ 1 月 05 日帰国
Canada 1/7-1/8         1 月 07 日出国→ 1 月 08 日帰国
Denmark 1/9-1/10       1 月 09 日出国→ 1 月 10 日帰国
Egypt 1/18-1/30        1 月 18 日出国→ 1 月 30 日帰国
```

あなた「なるほど。確かにそうですね。(そうか、Belgium は期間が長くて、Canada と Denmark の両方とだぶっていたからまずいんだな。ということは期間が短い航空チケットを選べば……いや、待てよ。たくさんの航空チケットと期間がだぶってしまうヤツを避けるべきかな。ふむむむ……どっちだ？)」

依頼者「何かおっしゃいましたか。大丈夫ですか。ともかく、航空チケットの枚数ができるだけ多いプランを立てていただきたい。もちろん、プランに含まれる航空チケットの期間にだぶりがあっては絶対にいけません」

あなた「そうだ。たとえば、『1 月 3 日に帰国して、その足で 1 月 3 日に出国する』というプランを作っても構いませんか」

依頼者「いいえ、だめです。出国日と帰国日が同日になるのも《だぶり》です」

あなた「わかりました。では、たとえば、『できるだけ合計の旅行期間が長いほうがよい』といった別の条件はありますか」

依頼者「いえ、ありません。プランに含まれる 航空チケットの枚数が多ければいい のです」

あなた「もしも、枚数が同じ複数のプランがあった場合にはどうしましょうか」

依頼者「プランはひとつだけ作っていただければ結構です」

あなた「整理させてください。こういうことですね」

- 国名と期間が書かれた航空チケットの一覧が与えられる。
- そこから何枚かを選択してプランをひとつだけ作る。
- 航空チケットはできるだけ多い方がよい。
- プランに含まれる航空チケットの期間がだぶってはいけない。

依頼者「その通りです」

あなた「では、プランを作るために、航空チケットの一覧をいただけますか」

依頼者「これです。ファイル `tickets.txt` としてお渡しします」

ミッションはこのようにして始まりました……。

3　解答編

3.1　はじめに

こんにちは、結城浩です。チケットゴブル問題へ多数のご解答ありがとうございました。今回のチケットゴブル問題は、たくさんのチケットから、期間がだぶらないものをできるだけたくさん集めるというものでした。楽しんでいただけたでしょうか。

今回は特に大きなトラップもなく素直な問題だったと思います。多くの方が最高評価 5 を得ることができました。その分、形式上の不備は厳しくチェックさせていただきました。ご了承ください（もっとも、「厳しく」とはいえ、前もって `answer.txt` に述べている通りのチェックですが)。

3.2　評価基準

評価基準は以下の通りです。

- 評価 5 チケットゴブル賞（62 チケット、最善プランを作成）
- 評価 3 準チケットゴブル賞（1〜61 チケット）
- 評価 1 残念賞（だぶりのあるプランを作成。形式上の不備を含む）

複数回投稿した場合は、最終投稿のみを評価しました。
以下の二点は CodeIQ 運営さんからの連絡になります。

- 評価 5 の方全員には チケットゴブルバッジ がサイト上で付与されます。
- 評価 5 の方には抽選で書籍 『プログラマの数学』 が当たります。

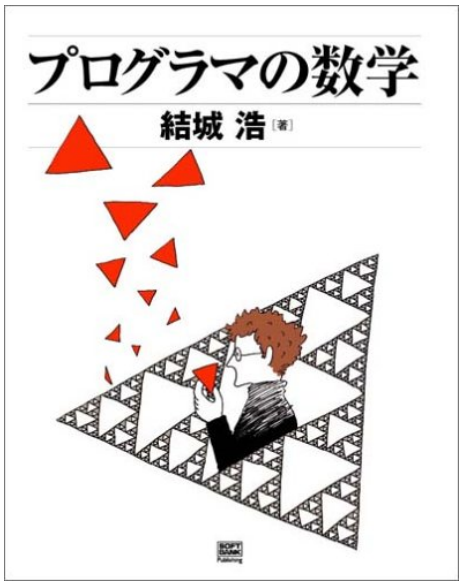


図 1　『プログラマの数学』

3.3 正解例

以下に正解例を示します。便宜上、複数行で表示してありますが、実際にはスペース区切りで一行です。

これは一例です。他のチケットの組み合わせでも、62 枚のチケットで期間のだぶりがなく、形式上の不備がなければ評価 5 になります。

```
62 Afghanistan Angola Antarctica Austria Azerbaijan Barbados Belarus Botswana
Burundi Canada China Cyprus Denmark Ecuador Eritrea Ethiopia Finland Georgia
Germany Ghana Greece Guadeloupe Jordan Kiribati Liberia Libya Liechtenstein Macao
Malawi Malaysia Martinique Mongolia Montenegro Montserrat Myanmar Namibia Narnia
Niger Nigeria Oman Palau Poland Portugal Romania Rwanda Samoa Senegal Serbia
Seychelles Slovakia Slovenia Somalia Sudan Sweden Switzerland Tajikistan Tokelau
Turkey Uganda Ukraine Zambia Zimbabwe
```

ファイルは Gist にも `answer.txt` として置いてあります。
https://gist.github.com/hyuki0000/6273a52b9454ccfca095

3.4 評価の分布

評価の分布は以下の表の通りです。

評価 5	230 名
評価 3	34 名
評価 1	70 名
合計	334 名

チケットの枚数やエラー原因による人数分布は以下の通りです。

62 チケット	230 名
61 チケット	9 名
60 チケット	3 名
59 チケット	2 名
58 チケット	4 名
57 チケット	3 名
56 チケット	4 名
55 チケット	2 名
47 チケット	1 名
43 チケット	1 名
42 チケット	1 名
4 チケット	3 名
1 チケット	1 名
期間がだぶっている	19 名
国名が未ソート	34 名
国数が書かれていない	8 名
国名の誤り	2 名
コンマなどの不正な文字が含まれている	4 名
国数が不一致	3 名
合計	334 名

3.5 アルゴリズムの考え方

それでは今回の問題を一緒に考えていきましょう。今回の問題は 区間スケジューリング 問題と呼ばれているアルゴリズムの問題になります。

問題の物語の中に、こんなセリフがありました。

> そうか、Belgium は期間が長くて、Canada と Denmark の両方とだぶっていたからまずいんだな。ということは期間が短い航空チケットを選べば……いや、待てよ。たくさんの航空チケットと期間がだぶってしまうヤツを避けるべきかな。ふむむむ……どっちだ？

このセリフで軽く提示されているアルゴリズムは以下の二種類です。

方法 1. 期間が短い航空チケットから順番に（だぶらないようにしつつ）選んでいく方法。
方法 2. だぶりが少ない航空チケットから順番に選んでいく方法。

これは自然な発想から出てくるものですが、どちらの方法も最適解を生み出す保証はありません。
なぜなら、 小さな例を作って試してみる とわかるように、最適解を見つけない場合があるからです。

方法 1 で最適解にならない例

方法 1 は期間が短いチケットから選びます。

以下のような 3 枚のチケットが与えられたとしましょう。

```
A 1/1-1/4
B 1/5-1/7
C 1/4-1/5
```

1 月 1 日から 1 月 7 日までの重なり具合はこうなっています。

```
1234567
AAAABBB
...CC..
```

このとき、方法 1 では、期間が短いチケットから選びますから、まっさきに C を選びます。ところがその結果、A も B も（C とだぶるために）選べなくなってしまいますね。最適解は A,B を選ぶことなのですが、方法 1 では、C だけを選んでしまいます。

ですから、期間が短い順に選ぶ方法 1 が最適解になるとは限りません。

方法 2 で最適解にならない例

方法 2 は期間のだぶりが少ないチケットから選びます。

以下のような 11 枚のチケットが与えられたとしましょう。

```
A 1/1-1/2
B 1/3-1/4
C 1/5-1/6
D 1/7-1/8
E 1/1-1/3
F 1/4-1/5
G 1/6-1/7
H 1/1-1/3
I 1/6-1/7
J 1/1-1/3
K 1/6-1/7
```

重なり具合はこうです。

```
12345678
AABBCCDD
EEEFFGG.
HHH..II.
JJJ..KK.
```

この場合、最適解は A, B, C, D を選ぶことなのですが、方法 2 では、重なりが少ない順番に選ぶので、A, D, F を選んでしまいます。B,C の代わりに F を選んでしまい、最適解を逃します。

ですから、だぶりが少ない順に選ぶ方法 2 が最適解になるとは限りません。

最適解を見つけるアルゴリズム

今回の問題で最適解を見つけるアルゴリズムは実は単純です。それは「帰国日の早い順にチケットを選ぶ」という方法です。もう少し具体的に書けば以下の手順です。これを便宜上「アルゴリズム **A**」と呼ぶことにしましょう。

ステップ 1 残っているチケットの中から最も帰国日の早いものを選び、プランに組み込む。
ステップ 2 ステップ 1 で選んだチケットとだぶっているチケットをすべて捨てる。
ステップ 3 残っているチケットがなくなったら終了する。まだ残っていたらステップ 1 に戻る。

最適解になることの証明

前述したアルゴリズム A は「帰国日の早い順にチケットを選ぶ」という単純な方法ですが、これで最適解が得られるというのはなかなか驚きですね。

アルゴリズム A が最適解の一つを見つけ出すことは数学的に証明できます。

まず最初に、アルゴリズム A が作り出すプランに含まれるチケットのどの二枚をとってもだぶりがないことはすぐにわかります。なぜなら、プランに組み込まれていくチケットとだぶるチケットは、ステップ 2 ですべて捨てているからです。

ですからあとは、アルゴリズム A が作り出すプランが含むチケットの枚数よりも多いプランは作れないことを証明すればいいですね。

ここで、アルゴリズム A が作り出すプランを、

$$a = \langle\, a_1, a_2, a_3, \ldots, a_m \,\rangle$$

と表すことにします。それぞれの a_k はチケットを表し、旅行日でソートされているとします（だぶりがないので、出国日でソートしても帰国日でソートしてもかまいません)。このプランのチケット数は m 枚です。

これとは別に最適解、つまり最も多くチケットを含むベストプランがあったとしましょう。これを

$$b = \langle\, b_1, b_2, b_3, \ldots, b_n \,\rangle$$

とします。先ほどと同じように b_k はチケットを表し、旅行日でソートされているとします。ベストプランのチケット数は n 枚です。

b はベストプランなのですから、$m \leqq n$ は明らかですね。私たちが証明すべきことは、$m = n$ です。言い換えれば「$m < n$ ではない」と証明すればよいことになります。

いま、プランの帰国日に着目したいので、

$$\mathrm{end}\,\big\langle \text{チケットの列} \big\rangle = \text{チケットの列で表されるプランの最終帰国日}$$

と書くことにしましょう。たとえば、

$$\mathrm{end}\,\langle\, a_1, a_2, a_3, \ldots, a_m \,\rangle = \mathrm{end}\,\langle\, a_m \,\rangle$$

が成り立ちます。

また、チケットの列にだぶりがないことを、

$$\mathrm{valid}\,\langle\ \text{チケットの列}\ \rangle$$

と書くことにします。たとえばプラン a にはだぶりがありませんから、

$$\mathrm{valid}\,\langle\, a_1, a_2, a_3, \ldots, a_m \,\rangle$$

です。もちろんベストプラン b にもだぶりはありませんから、

$$\mathrm{valid}\,\langle\, b_1, b_2, b_3, \ldots, b_n \,\rangle$$

ですね。

さて、これからアルゴリズム A が作り出すプラン a と、ベストプラン b を比べていきます。

まずは 1 枚目のチケットを比べます。アルゴリズム A は、帰国日が早いものからチケットを選びますから、

$$\mathrm{end}\,\langle\, a_1 \,\rangle \leqq \mathrm{end}\,\langle\, b_1 \,\rangle$$

が成り立ちます。

k 枚目まで比較してきて、

$$\mathrm{end}\,\langle\, a_1, a_2, a_3, \ldots, a_k \,\rangle \leqq \mathrm{end}\,\langle\, b_1, b_2, b_3, \ldots, b_k \,\rangle \qquad \cdots (K)$$

が成り立っていたとしましょう（$1 \leqq k < m$ および $1 \leqq k < n$ とします)。このとき、$k+1$ 枚目まで比較して、

$$\mathrm{end}\,\langle\, a_1, a_2, a_3, \ldots, a_k, a_{k+1} \,\rangle \leqq \mathrm{end}\,\langle\, b_1, b_2, b_3, \ldots, b_k, b_{k+1} \,\rangle \qquad \cdots (K')$$

が成り立ちます。なぜなら、もしも、(K') が成り立たないとするなら、

$$\mathrm{end}\,\langle\, a_1, a_2, a_3, \ldots, a_k, a_{k+1} \,\rangle > \mathrm{end}\,\langle\, b_1, b_2, b_3, \ldots, b_k, b_{k+1} \,\rangle$$

ということですから、$\mathrm{end}\,\langle\, a_{k+1} \,\rangle > \mathrm{end}\,\langle\, b_{k+1} \,\rangle$ が成り立つことになってしまいます（下図参照)。

$$\begin{array}{cc} \overset{a_k}{\cdots \longrightarrow} & \overset{a_{k+1}}{\cdots \longrightarrow} \\ \overset{b_k}{\cdots \longrightarrow} & \overset{b_{k+1}}{\cdots \longrightarrow} \end{array}$$

しかも (K) から、$\mathrm{valid}\,\langle\, a_1, a_2, a_3, \ldots, a_k, b_{k+1} \,\rangle$ がいえますから、帰国日が早いチケットを選ぶアルゴリズム A は $k+1$ 枚目のチケットとして a_{k+1} ではなく b_{k+1} を選んでいたはずです。これは矛盾です。ですから (K') は成り立ちます。

以上より、$k = 1, 2, 3, \ldots, m$ について、

$$\mathrm{end}\,\langle\, a_1, a_2, a_3, \ldots, a_k \,\rangle \leqq \mathrm{end}\,\langle\, b_1, b_2, b_3, \ldots, b_k \,\rangle$$

が成り立つといえます。

ここで、プラン a がチケットをすべて使い尽くしたとき（m 枚目までベストプランと比較してきたとき)、以下の式が成り立っているはずです。

$$\mathrm{end}\,\langle\, a_1, a_2, a_3, \ldots, a_m \,\rangle \leqq \mathrm{end}\,\langle\, b_1, b_2, b_3, \ldots, b_m \,\rangle$$

さてここで、もしも $m < n$ だと仮定しましょう。もしそうだとするなら、ベストプランには b_{m+1} というチケットがあるはずです。このチケットはプラン a のどのチケットともだぶりません。すなわち、

$$\mathrm{valid}\,\langle\, a_1, a_2, a_3, \ldots, a_m, b_{m+1} \,\rangle$$

が成り立ちます。ということは、プラン a にチケット b_{m+1} を追加して、a よりもさらにチケット数の多いプランがつくれてしまうことになります。これはアルゴリズム A によるチケットの選び方と矛盾します。ですから、$m < n$ ではないことがいえます。

以上より、$m = n$ が証明できました。すなわち、私たちのアルゴリズム A（帰国日が早い順にチケットを選ぶ）は、ベストプランと同じだけのチケット数を含んでいることになります。

以上の証明は Kleinberg, Tardos『アルゴリズムデザイン』（共立出版）を参考にしています。

この書籍では上のアルゴリズムを グリーディ (greedy) なアルゴリズムの例として紹介しています。グリーディな（貪欲な）アルゴリズムというのは、あとになって困るかもしれないなどと考えずに、各時点でいちばんほしいものをどんどん取っていくようなアルゴリズムの総称です。今回の場合、帰国日の早いチケットをどんどん貪欲に選んでいますね。

チケットに重みが入ってくると、動的計画法を使うことになります。

3.6 解答に使ったコード

解答に使ったコードは Gist に置いてあります。

このコードでの日付の計算では、月を 100 倍して日に加えています。これは、今回の問題では日付の順序関係だけが問題で、チケットの正確な期間は不要だからです。

https://gist.github.com/hyuki0000/6273a52b9454ccfca095

3.7 挑戦者が使った言語

挑戦者が使った言語などは以下の通りです。

行末の数は挑戦者数を表しています。

- Action Script 3.0, Perl 5 1
- C 8
- C + Cygwin 1
- C# 7
- C# 4.0 1
- C# 4.5 1
- C# MicrosoftVisualStudio2010 1
- C#(.NET Framework 4.5) 1
- C#,Excel 2
- C++ 20
- C++ (Xcode on mac mini) 1
- C++ (g++ 4.8.2, -O2 -std=c++1y -Wall, cygwin) 1
- C++ + gcc 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5) 1
- C++ VS2013 1
- C++(gcc-4.8.1 (ideone)) 1
- C++11 2

- C/C++ 1
- Clojure 1
- Common Lisp 1
- C 言語 1
- C 言語 gcc 4.8.1 1
- C 言語　Mac OSX Xcode 5.1.1 1
- D 2
- Elixir 1
- Excel 11
- Excel VBA 1
- Excel+ 手計算 1
- Excel, 秀丸エディタ 1
- Excel2003,Excel VBA 1
- Excel2010 VBA 1
- Excel と手作業 1
- Excel シート 1
- F# 1
- Go 1
- Haskell 8
- Haskell2010 1
- JAVA 2
- Java 34
- Java 1.6 1
- Java 1.7 2
- Java 8 1
- Java SE 8 1
- Java jdk1.7 1
- Java1.8.0_05 1
- Java7 1
- Java8 3
- JavaScript 6
- JavaScript(Node) 1
- JavaScript+ 手作業 1
- LibreOffice Calc 1
- Microsoft Visual C++ 2010 Express 1
- Node.js 1
- OCaml 1
- PHP 11
- PHP 5.3.3 1
- PHP 5.4, Netbeans 7.4 1
- PHP5.3 1
- Perl 13
- Perl 5.14.2 1
- Processing (Java) 1
- Python 15
- Python 2.7 3
- Python 2.7.2+ 1
- Python 2.7.3 2
- Python 2.7.5 1

- Python 2.7.6 1
- Python 3.3 1
- Python 3.4.0 1
- Python 3.4.0 (Windows 7 64bit) 1
- Python2.7 2
- Python3 2
- Python3.3 1
- R 3
- Ruby 55
- Ruby + Excel 1
- Ruby + 紙と鉛筆 1
- Ruby 2.0 1
- Ruby 2.0.0 1
- Ruby 2.0.0p451 1
- Ruby 2.1.0 1
- Ruby 2.1.1 1
- Ruby, Excel（CSV データの読み込み） 1
- Ruby,Excel,Windows,Adobe Reader,Lunascape. 1
- Ruby、表計算ソフト (OpenOffice) 1
- Scala 4
- Scala 2.10 1
- Scala 2.10.4 1
- Scala, Haskell 1
- Scala、Java、紙とペン 1
- Scheme (gauche) 1
- TeX (TeX on LaTeX) 1
- VB 1
- VB6 1
- VBA 1
- VimL 1
- Visual C++ 2013 1
- VisualStodio2012 で VB.Net を使用 1
- XMind 1
- c++ 1
- google apps script 1
- groovy(version 2.2) 1
- haskell 1
- java 1
- java8 1
- javascript 1
- node 1
- nodejs 1
- objective-c 1
- perl 2
- php 1
- php 5.4.15 1
- python 1
- python2.7 1
- python3 1

- ruby -v / ruby 2.1.0p0 (2013-12-25 revision 44422) [x86_64-darwin12.0] 1
- ruby 1.9.3p194 (2012-04-20 revision 35410) [x86_64-linux] 1
- ruby 2.0.0p247 (2013-06-27) [x64-mingw32] 1
- ruby 2.0.0p451 1
- tr + sort + vi + awk + sort + tr 1
- なし 1
- 手作業 1
- 手計算、Java 8 1
- 日本語プログラミング言語「なでしこ」 1
- 蛍光ペンとルーズリーフ 1

3.8 挑戦者の声から

ではおまちかね、挑戦者のみなさんの声をいくつかピックアップしてご紹介します！ 全員の分は掲載していません。また、掲載しているものも全文ではなく、さらに、一部編集している部分もあります。ご了承ください。

ENV: Ruby
POINT: 貪欲法。帰国日がはやいものを選び続ければいい。
注意点は、帰国日と出国日を重複しないようにすること。

ENV: C
POINT: 出発日と到着日の関係
感想:
前回のスペーストーキーでは、残念ながら評価 3 だったので再挑戦させて頂きました。
どうしてもポイントなる点に気付けず、いつもなら何かしら引っかけがあるはずなのにと
疑いながら解きました。結局、終わりまで気付けず終いでした。

---

1 Afghanistan
ENV: C
POINT: 間に合いませんでした (-_-)

次は解いて見せます！

（結城：1 チケットのプランで評価 3 をゲットした方ですね (^^)）

ENV: Ruby
POINT: 帰国日が最も早いチケットを選んでいけばいい

最初、最も帰国日の早いチケットを選び、

以降、最後に選んだチケットより出国日が後のチケットのうち、
最も帰国日の早いチケットを選んでいく。
こうしてできたプランがチケットの枚数が 62 枚で最大となる（最大性の証明は省略)。

蛇足だが、チケット枚数最大のプランが何通りあるかも求めてみた。
結果は 59929893273600 通りとなった。

ENV: Excel
POINT: 帰国日の早いチケットを次に選ぶ。

コード化してる余裕が無かったため、取り急ぎ Excel で手作業で頑張ってみました。
「残りのチケットの中で帰国日が最も早いチケットを選び、それと期間が重複してるものを取り除く」
という作業をくり返して上記の解を作ってみました。

ENV: Java
POINT: 帰国日が早いチケットを選ぶことがポイントです。
「航空チケットの枚数が多ければいい」という条件だったので、
帰国日が早いものから、期間が被らないようにチケットを選択しました。

ENV: Ruby
POINT: 帰る日の早い仕事を選び，それと期間が重なるっているチケットを取り除くという操作の繰り返しにより最適解が得られる
クリプタンで 2 つ目が解けず、そのリベンジに挑戦したスペーストーキーでは境界条件のチェックを失念しておりバッジ獲得ならず……なので、今回は確認！ よかった、はず……。

ENV:Excel
POINT:開始日がかぶらないもののなかで終了日が一番早いものを選んでいくことがポイントです。

ENV: Ruby
POINT: 問題を良く読んで、ミスのないように。

ENV: C 言語　Mac OSX Xcode 5.1.1
POINT: 行き先を帰国日の昇順に並べ、帰国日の早いものから帰国->出発が可能なものを順に選ぶ。この方法で最多の組み合わせを選ぶ事ができる事がポイント。

ENV: Ruby
POINT: 最適化範囲の分割
チケットの組み合わせが良心的だったので、思っていたよりスムーズに解けました。
プラン自体は旅行者にまったく良心的でないですが・・・こんなプラン、どうやって売りさばくのやら（笑）
旅行期間は気にしないとありましたが、同一枚数の解がある場合は、期間の長いものを選択しました。

ENV: Ruby
POINT: 出国日にはとらわれずに帰国日が最も早いチケットを順に選んでいくのがポイントです。

ENV: Ruby
Point: 帰国日が早いチケットから greedy に選択していくのがポイントです

ENV:C++11
POINT:区間スケジューリング

ENV: PHP 5.3.3
POINT: 次の行き先候補のうち出発可能なチケットの数が最も多い行き先をひとつ選び枝刈りして探索をすること
プログラミングの基礎力を向上させたいなと思い、少しずつデータ構造とアルゴリズムを勉強して
いますが、このような問題があるとモチベーションの維持につながりますのでとてもありがたく思っております。

ENV: Python
POINT: 早い順に選ぶなら、できるだけ帰国の日付が早いものを選べば良いこと、がポイントです。

今回は、アルゴリズムを思いついてしまえばすぐでした。
以下、プランを考える時、日付が早いチケットから選んでいくことにします。
このとき重要なのは,、期間の短さでもチケットの被り方でもなく、帰国の日付の早さであることに気付くのがポイントだと思います。

ENV: python3
POINT: 貪欲法で帰国日が現時点から近いものを選び続ける。

CodeIQ 初挑戦でした

ENV: Java
POINT: サイクルのない有向グラフ上の最長経路の問題と捉え、エッジの重みを負として Bellman-Ford 法で解きまし

た。問題中の日付が閏年であることが何かのトラップかと思いましたが、解を精査したところ影響はありませんでした。

ENV: C++
POINT: DP で解きました。

ENV: Ruby
POINT: 不要な情報を適宜削除することで組み合わせが爆発しないようにする

ENV: Java 1.7
POINT:一番早く終わるチケット（もしくは一番遅く始まるチケット、どちらでも可）を順次追加していけばいいと思います。

ENV: Python2.7
POINT: とにかく早く帰国する

今回はあまり時間がなく、きちんと検証できませんでした。。
行ける国の中からもっとも早く帰国できるものを選択すれば、直感的に最大化できそうな気がしますがはたして。。
そして危うくフォーマットを間違えるところでした。(プランに含まれる国の並び順)
確認大事。

ENV: Ruby
POINT: 問題を適切に分割すること
そのまま眺めていても何も思いつかなさそうだったので、取り敢えず添付の ConvertDate.rb を作り、中身の形式を変えたファイルを作りました。
それを見ていたら、いくつかの独立な組に分割できることに気づき、小サイズの問題なら力押しでなんとかなるのではと思って、実装した感じです。

ENV: Java jdk1.7
POINT: 旅行期間の重なりをグラフのエッジで考えて、一番多いエッジを含むチケットを切っていき、
すべてのノードが独立するまでループしました。

ENV: Java
POINT: 帰国日を基準としてプランのアルゴリズムを考えたのがポイントです。
感想：Java を使ったことがなかったので、今回勉強のために Java を使用しました。普段使った言語ではなかったので、プログラムの作成に四苦八苦しました。

ENV: Ruby
POINT: うしろから

年末から逆順に何種類のチケットを使え得るかと確認していきました。
チケットを利用すると「それ自身＋帰国後翌日の最大利用枚数」がわかるので
選択肢の中から最大のものを選べばよい。

確認用に、チケットの期間が重なっていない日で区切り、重なっているものに対し
その期間における総当りも行い、最大枚数を調べました。

その枚数にするときの組み合わせが 59,929,893,273,600 通りとでて……

ENV: Java
POINT: 動的計画法で解きました。
別解がいくつもありそうですね。
私の解答が正しければ、という仮定付ですが、ざっと見た感じ、Mozambique は Finland でも可。Colombia のところは、Uzbekistan、Georgia、Anguilla でも可など。
全パターンを求めるプログラムを作るのも面白そうです。

ENV: Ruby
POINT: 愚直に他のチケットとだぶる数が多いチケットを排除していきました。

ENV: C#
POINT: アルゴリズム（貧欲法）がポイントです。
感想:探索アルゴリズムがわからず悩みましたが、たまたまみつけた本にヒントがありました。選択可能なチケットのうちに、最もはやく旅行が終わるものを探すことにしました。方針がきまれば実装は平易だったと思います。うるう年であることの影響は不明でした。

ENV: Ruby
POINT: 1/1〜12/31 の配列を用意し、前から順に「その時点で最も多くの国へ旅行できるプラン」を求める方法を用いました（動的計画法）
CodeIQ 初挑戦です。結城さんの問題は Twitter でよく話題になっているのでいつか挑戦したいと思っていました。
動的計画法で問題を解くのも初めてなので一部非効率な部分があるかもしれません。

ENV: Java
POINT: すべての組合せを考えると 2^162 通りの組合せになるため、組み合わせる範囲を分割し解を求めました。

162 通りのチケットについてプランに含まれる/含まれないの 2 通りの状態を考えると 2^162 通りとなり、
計算が終了しないため工夫をする必要があることがわかりました。
このため 162 のチケットについて組合せ範囲を分割し分割範囲それぞれの最適解の和を回答としました。
具体的には「1/26] は旅行中の状態になることは全くないため、1/1 から 1/25 までで最もチケットが多くなる組合せを求め、
1/27 以降についても同様に、1/27〜2/12、2/13〜3/5…というように組合せ範囲を分割して解を求めました。

ENV: Ruby
POINT: 全部のパターンを数えあげようとしないこと。

ENV: Python 3.3
POINT: 各旅程をノードとした、有向グラフの探索問題として考えることが最重要だと思います。
DAG の最長経路探索問題として DP で解きました。

ENV: 日本語プログラミング言語「なでしこ」
POINT: 帰国日順に並び替えて、重複しないように出国日より後で帰国日が早い日の行き先を順に選んでいくのがポイントです。

ENV: Ruby
POINT: 難しく考え過ぎないこと？

使った方法は「選択可能なものの中から一番早く帰国するものを選ぶ」です。
単純すぎて迷いましたが、最終的にはもういいや、といった気持ちになり
このまま提出することにしました。

ENV: Perl
POINT: 出国日順に並べ替え、(1) 一部重複の場合は期間が遅いものを削除し、(2) 包含している場合は期間が長いものを削除する。

ENV: Haskell
POINT: すべて-1 の重みを持つ非循環有向グラフの最短経路を説くアルゴリズムを使ったのがポイントです。
各チケットを頂点、チケットを使用した後に使用可能なチケットを辺としたグラフだと気付きました。
そのためそのようなグラフの最長経路を求める問題ととらえて、すべての辺が-1 の重みをもつ非循環有向グラフの最短経路をとく方法としてベルマン・フォード法を使用しました。
Haskell はほとんど使用したことないですが、わりと簡単にアルゴリズムの実装ができたのでひとりで勝手に驚きました。

また帰国した日には出発できない制約のおかげで非循環グラフにできたと思うので助かりました。

ENV: Java
POINT: 日数を減らした部分最適化問題を利用して全体最適化問題を解くことができる点

ENV:Python
POINT:ポイントは帰国日が早いものから選んで、一つ前の帰国日がその選んだものの出国日に間に合うかどうかを判定することです。
今回はじめて挑戦をしました。プログラミングもアルゴリズムも勉強をはじめたばかりなので、わからないことばかりでしたが、楽しく挑戦できました。結城先生の『数学ガール』愛読しています。今回も「例示は理解の試金石」の精神でがんばりました。

ENV: Perl
POINT: 予め要素の数を減らすのがポイントです。

いつも楽しい問題をありがとうございます。

さて、今回の問題ですが、総当たりで試すには要素が多く、全ての組み合わせを考えるのは現実的ではないので、
下記順番で組み合わせを考える要素を減らしてから、チケットを抽出しました。

1．チケットの期間が包含関係にあるもののうち、期間の長いものを除外。
2．残ったチケットを出発日程の早いものから順に並べ、開始から終了までがオーバーラップするチケットだけを
　集めたグループを作る。
3．グループ内で最もチケット枚数の多い組み合わせを求める。
4．使用するチケットを国名でソート

ENV: C
POINT: 区間スケジューリング問題
最初日付をどう扱おうか悩んで、最終的に 1 月 1 日を基準にしてそこから何日目なのかで統一しました。
統一したら、あとは区間スケジューリング問題だったので、帰国日時の早い順に国を選択していきました。
プログラムのミスがなければこれであってるはず。

ENV: VB
POINT: 探索ロジックの設計がポイント。

1.Ticket 一覧を出国日でソートした上で、期間がだぶらない組み合わせで Tree 状に順にぶら下げる。
　その際、以降の Ticket をぶら下げる可能性のあるノード一覧をリーフとして管理しておく。
2．全てのチケットをぶら下げ終わった後、リーフから順に親を辿った結果が旅行プランとなる。
3．ある Ticket をぶら下げ可能なリーフの内、Tree の長さが最長となるノード 1 つのみに Ticket をぶら下げる。

1,2 が基本的な探索ロジックだが、このままではリーフの数が指数的に増加する。
回答は 1 組だけで良い事から、3 の条件を導入する事で回答の正当性を損なわずに計算量を抑える事が出来る。

ENV: TeX（TeX on LaTeX）
POINT: 帰国日順にソートし，帰国日が早いものから順に貪欲に選んでゆく。

ENV: Perl
POINT: 帰国日でソートして貪欲でポン
引っ掛けありませんように！

ENV: PHP
POINT: 出発日でソートして、最後から探索しました。

ENV: R
POINT: 国と国が期間が重なっているかどうかをマトリックスとして管理して、そのマトリックスだけを見て航空チケットの枚数に不利な国を取り除いて行き、残ったものが一番よいプランです。

ENV: OCaml
POINT: 選べるチケットの中で帰国日が最も早いものを選ぶことを繰り返す。Greedy。アリ本 P.43 と同じ問題。

（結城：そうですね。アリ本＝『プログラミングコンテストチャレンジブック［第 2 版］』秋葉拓哉、岩田陽一、北川宜稔）

ENV: Scala
POINT: 前提条件として，tickets.txt に記された各チケットの行き先の国に重複は存在しなかったため，本問題は最も多くのチケットを使って旅行することと等価である．このためには，各時点で最も早くに帰れるチケットを貪欲的に選択し続ければ良い．何故なら，ある地点で最も早くに帰れるチケット x を選択したとき，x を選択した場合より多くのチケットを選べるケースが存在するとしたら，そのケースには x より早くに帰ってこられるチケットが無ければならず矛盾であるからである．従って，入力を一旦帰国日時の昇順でソートし，あとは選択可能なチケットを順に選び続ければ良い．

ENV:Ruby
POINT:シンプルな貪欲法

ENV: C++ + gcc 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)
POINT: 区間スケジューリングの問題として、到着日の早いものから選ぶ貪欲法により解きました

ENV: XMind
POINT:日付順にソート

これまでの先生の出題された問題の中で最速の短時間で解けた問題でした。

ENV: Excel2010 VBA
POINT:多くのチケットを捌くには「帰国日から次の出発日までの期間＋旅行期間」が一番短いチケットを選ぶこと。

ENV: Ruby 2.0.0
POINT:闇雲に組合せを試す前に下処理をするのが吉

1) 同一期間のチケットが複数ある時は 1 枚だけ残しあとは使わない。
2) あるチケットの期間内に別のチケットの期間が丸ごと含まれている場合、そのチケットは使わない。
3) 残ったチケットを期間が重複しないグループに分類する。
4) 各グループ内でチケットの組合せを総当りで調べる。
　というやり方で調べました。

ENV: Java
POINT: dp[i] を「i 日目までに使えるチケットの最大数」と定義し、年の初めの日から順に更新していく DP

ENV: tr + sort + vi + awk + sort + tr
POINT:　日付順ソート。期間が包含関係にあるとき含む側のチケットは諦めて良い。
問題の理解のために編集してたら、プログラム書く前に終わってしまった。

ENV: C#
POINT: 帰国日を使用して動的計画法で答えを求める。ナルニア国行きのチケットがある。

ENV: C++
POINT: 蟻本

問題を見た瞬間、プログラミングコンテストチャレンジブック 第 2 版の 43 ページに載っている
「区間スケジューリング問題」の仕事をチケットに変えただけの問題だと気づきました。

従って、そこで紹介されている通りに、
使用可能なチケットで、到着の日付が一番早いものを順番に採用するプログラムを書きました。

ENV: Java
POINT: 無駄な計算を減らすための工夫をするのがポイントでしょうか。
なかなか OutOfMemoryError が止められず苦労しました。

ENV: javascript
POINT: グループ分けがポイントです。

方針
できるだけ、たくさんの旅行先に行きながら、
日本にいられる時間を長くする！

62 件の旅行先を旅しながら、最短の旅行日数は
197 日でした。
GOOGLE CHROME にて実行しました。

Ticket の枚数は 162 枚。
これらの Ticket 全てに渡って、行く、行かないを
調べると、2^162 という途方もないパターン数を
計算する必要があり、速度的に難しい。
そこで、高速化のために、旅行期間が重複しない、
小さなグループに旅行先を分割して、計算を行いました。

ENV: C
POINT: グループ分けして考えることがポイントです。期間が被っている航空チケット同士で 1 つのグループを作る。そうすると、いくつかのグループが出来上がるので、各グループに対して問題を解けば良い。

ENV: Java
POINT: 全チケットからなる集合をお互いに期間がだぶらない複数のパーティションに分割し、パーティション毎の最多チケットプランを求め、それらを直和すればそれが最多チケットプランとなる。

久しぶりにアルゴリズムで頭をひねらせて楽しかったです。
さすがに総当たりで調べたら計算量が爆発しそうでしので、出国日でソートしたデータを眺めていたら、期間の切れ目のところで分割する作戦を思いつきました。

ENV: D
POINT: 隣接リストを用いたメモ化再帰で解を探すが、最初の訪問国となりうる国が絞られること、さらに解はどれ

か 1 つでよいことから計算量を減らすことができる。

ENV: Ruby
POINT: 帰国日でソートしチケット組合せの対象を絞り込むこと、チケットとチケット枚数が最大となるプランを保持しておくこと

毎回楽しく解かせていただいております。
スペーストーキー問題では見事に引っかかってしまったため、リベンジということで今回も参加させていただきました。

ENV: Ruby
POINT: 自分で考えたポイントとしては出国日時順に並べてから考えることでした。あとは余分なパターンを省きつつ使うチケットを追加していくことで答えを得られました。

ENV: LibreOffice Calc
POINT: ガントチャート風の表にまとめ、選択できるチケットの中から最も早く日本に帰ってくるチケットを選択を繰り返す。

ENV: Ruby
POINT: 他のチケットの期間を包含するチケットをまず排除する。

楽しそうなプランだけど、お高いんでしょ？

ENV: C++
POINT: 現在の日付より後で、帰国の日付が最も早いものを選ぶことが一番よい方法だと考えました。

ENV: Java
POINT: 距離が負（または 0）の最短経路問題

ENV: Java
POINT: チケットを出国日の順にソートして、若い順からそれ以前の帰国日のチケット中でもっともチケット消費量の多いものとエッジを結ぶことでグラフを作る。そのようにしてできたグラフの最長パスが選ぶべきチケットセットになる。

ENV: objective-c

POINT: 手動でコツコツ

判定するロジックが解らなかったので、
どのチケットが何回重複するところまでプログラムで行い、
後は手動で判定しました。

重複している回数が一番多く、
期間が長いチケットから除外していけばいいのかな？
とも思いましたが、確信が持てませんでした。

ENV: Ruby + 紙と鉛筆
POINT: Ruby で日付順にソートして、紙と鉛筆で解きました。

今回は無理にプログラミングする必要がないような気がしましたので、
無謀にも紙と鉛筆で解きました。そのため、ミスがあるかもしれません。

（結城：ちなみにこの方は評価 5 でした）

ENV: haskell
POINT: 可視化

期間が被るノード同士にエッジを引いた無向グラフにおいての最大独立集合問題として
解きました。

このノード数だと相当頑張らないと（頑張っても？ ）解が得られないと考え、取り敢えず
可視化してみたところ、各独立頂点集合はたかだか 18 ノードからなる事がわかったため、
各独立頂点集合に対して DFS を行い、解を導きました。

ぱっと見ではノード数の多さにげんなりしましたが、可視化することで行けそうと感じ
たため、可視化の重要性を感じました。

ENV: Python 2.7.3
POINT:解答形式を守る。
各チケットの旅行期間を 1/1 を 0 としてナンバリングして、帰国日時の早いものから順に並べ、
動的計画法を用いて解いた。

ENV: Haskell
POINT: 区間スケジューリング問題

このチケットの数だと総当たりでは時間的にダメだろうな〜、
うーんどうしよう〜、ナップザック問題の応用かな〜。

などと思いながら色々調べていると、区間スケジューリング問題の存在を知りました。

問題は複雑な（ように見える）のに、アルゴリズムはとても簡単なのに驚きでした。

ENV: Python 2.7.3
POINT: 有向グラフの最長経路問題に帰着できれば後は簡単でした。
こっそり閏年が入っているのが楽しいですね (笑
ところで、解が複数あるはずなのですが、どうやって"機械的に"答え合わせしているのでしょう。。。

（結城：最大数はわかっているので、あとはだぶりがないかどうかを調べています）

ENV: Java1.8.0_05
POINT: 組合せ最適化へのアプローチとして遺伝的アルゴリズム（GA）を採用したことがポイントです。
不得手ながらなんとか回答らしきものを探り当ててきましたので，提出してみました。

ENV: Java
POINT: 区間列から選ぶ最長 chain．終端昇順にソートして、区間 i の始端を終端集合から二分探索すると、i の始端より前に終端があるものが二分探索の結果の先頭側すべてに固まっているので、そこでの最大値を見て伝播すればよい。O(Nlog N)．
ソートをバケットソートで行って O(N)，始端終端をイベント化してソートし、始端イベントの直前の終端イベントを記録するようにしてから処理を行うと、全体で O(N) でいけそうだけど、めんどくさそう。

ENV: Ruby
POINT: ZDD を使って最適解を探しました。変数の順序を出発日順とすることで、接点の数がチケットの数に等しいコンパクトな ZDD を生成することができました。

ENV: python2.7
POINT: 帰国日が早いものから順に選択していけばよい

ENV: Python
POINT: 動的計画法？

ENV: Scala 2.10.4

POINT: なるべく早く帰国させると，たくさんの国を訪問できるようになります．

感想: 解答は単にチケットの日付でソートするだけなのですが，ソートするだけでこんな問題も解けるのだというのが勉強になりました．出題ありがとうございます．

ENV: Scala
POINT: 旅程が重複しない集団を見付けること

最初は何も考えずに整数計画問題に落とし込みましたが、
計算が終了する気配が見えませんでした。そこで、旅程を
概観するためにガントチャートに書き下してみました。すると、
いくつか旅程が重複しない集団（Afghanistan は 4/18 発、4/19 着ですが
この間を旅程に含む他の国はない）があることに気付きました。
これらの部分集団で訪問する場所を最大化することで、上記の解を
得ました。ナルニアや南極が含まれていますね :-)

ENV: C#
POINT: 帰国日が早い順にチケットを選んでゆくのがポイントです。

アルゴリズムは知っていました。
日付計算をするので日付型（DateTime 構造体）でデータを取り扱おうと思い、適当に 2014 年としたのですが、
うるう日があったのできちんと Parse してくれなかったのは、なるほどと思いました。

ENV: Ruby
POINT: 「帰国日」が早いものを探すことがポイントです。
今回も面白い問題を出題してくださりありがとうございます。楽しくうんうん考えられました。

ENV: Python 2.7
POINT: 各チケットについて、そのチケットを使う場合、それを使うまでに最大何枚のチケットを使えるかはいつも決まっているので、動的計画法で答えを求めることができるのがポイントだと思います。

ENV: ruby 1.9.3p194 (2012-04-20 revision 35410) [x86_64-linux]
POINT: o(N^2) 以上の計算の N がなるべく少なくなるようにまず独立なチケットのグループを作りました。分割と統治!!

ENV: C#,Excel
POINT: スケジューリング結果は目視確認するしかない点がポイントです。

ENV: Python2.7
POINT: 1 年を丁度いい期間でいくつか区切れるところを探し、それぞれの期間にチケットを振り分け期間ごとに最大のチケットパターンを見つける

最初は期間を区切らずにコーディングしました。
処理が終わりそうにないので、期間で区切れるか検証し、区切ることが出来るのが判ったので、
期間分割処理を付け加えました。
もしスムーズに区切れなかったら大分実装に時間がかかったと思います。

ENV: java8
POINT: N 日までに何回旅行に行けるかではなく、N 回目までの旅行で一番早く帰ってこれるのはいつかをメモった点がポイントです。

ENV: Ruby
POINT: チケットの先行/後続関係をグラフ構造で持ち、適切な先行チケットを更新していく。

先行するチケットから後続するチケットへ有向辺を持つ有向グラフにおいて、最長路を求める問題として考えました。

チケットをグラフへ追加する際、そのチケットに先行するチケットの中で、もっとも長いパスを持つチケットを親とすることで、
追加されたチケットまでのもっとも長いパスが分かります。
そして、チケットのグラフへの追加により、追加されたチケットに後続するチケットのもっとも長いパスが更新される可能性があるので、
後続するチケットに対して再帰的に親の更新を行いました。
（ここでいう「親」は、有向グラフ上に張られた最長路のパスのツリーでの親という意味です）

最初、グラフ構造をまず作ってから走査を行うのかと思っていたのですが、
上記のように考えることで、グラフ構造を作りながら最長路の解決も行えることが分かって、
意外とスッキリしたアルゴリズムになりました。

ENV: nodejs
POINT: 最短経路問題のアルゴリズムを参考に最長経路問題として実装しました

ENV: Java 1.7
POINT: 各行き先毎に重複リストを作成し、重複関係がなくなるまで重複リストの大きい行き先から削除していきます

ENV: Ruby
POINT: 重複が少ないチケットから順次選びとっていきました。他に、最初は乱数等でチケットを選び、何度もチケットを入れ替えては重複を排除し、総枚数が増えたら採用…を繰り返しているだけでも割とすぐに収束するようでした（乱択アルゴリズム？ あまり効率は良くありませんが…）。

ENV: Java SE 8
POINT: ダイクストラ法。２点間の距離を用い、総距離を最小化する目的でよく使われるが、この問題の場合、２チケット間の距離が 1（続けて入国できる）または 0(期間が重複している）で、総距離を最大化する、という形に問題を読み替えることができる。面倒なので次の２チケットを追加し、（１）から（２）の総距離を最大化する。（１）国名 pre_first、入国日 100 年前の 1/1、出国日 100 年前の 1/1。（２）国名 post_last、入国日 100 年後の 12/31、出国日 100 年後の 12/31。　最後に、小さな注意点として、問題のチケットは 4/29 が含まれるため、暗黙にうるう年であることが想定されている。日付として扱う際は 2000 年などうるう年を設定すること。誤って 2014 年を補完してから java.textDateFormat#parse() すると、（デフォルトが isLenient == true なので）勝手に 2014/3/1 になる。

ENV: C++11
POINT: なるだけ早く帰ってこれるチケットから順に選ぶ，貪欲法です．

ENV: Ruby
POINT: スケジューリング問題に帰着すること（終了日時が最も短いものを選んでいけ
ばよい）
ゴールデンウィークの余裕から挑戦させていただきました。
１時間くらい悩んだあげく、スケジューリング問題に帰着できることに気づきました。

ENV: Ruby
POINT: 帰国日でソートして使えるチケットを順番に詰めていくのがポイントです。

感想:
いろいろ探索したり組み合わせたりする必要があるのかと思いきや、
案外簡単な方法に落ち着いたのが意外でした。
書いたコードが正しく最大のチケットが得られるのか、証明する方法が知りたいです。

ENV: JavaScript
POINT: 帰国の日付が早い順に取っていくだけです。

合っているか心配だったので、下のように可視化して調べました。目が痛いです。

```
00000000 33333333  77777  11111111111  55555558888 00  222 4444444444444 888
  111  222   55555 88888    222222222  6666666  99999999 3333333333333   9999
```

```
          44444 66666666        333333333377          1111        55555555
                    9999             444444                                66
                       000                                                  7777
```

ENV: VimL
POINT: 動的計画法とアルファベット順に解答すること

ENV: Python
POINT: 帰国日が早い順に、期間が重複しないようにチケットを選ぶ

ENV: Python
POINT: 到着日が遅いチケットから順に計算していくことがポイントです。
また、2/29 の日付があったので、1 年を 366 日としています。

ENV: Ruby
POINT: greedy

#区間スケジューリングやるだけ
#ソートに分布数えソートを使えば O(n) でも解ける

ENV: C++
POINT: 区間スケジューリング問題なので、帰国日時が早い順に選択する。

ENV: Ruby 2.0.0p451
POINT: 日が重ならず直近（間に他国へのフライトを挟む事なく）で続けて飛べる国のペアを組む。
そうしたらば、有向グラフができたのですが、グラフ理論系のアルゴリズム等も使う事なく解けてしまったので不安です。
これから検算しますが取り敢えず提出します。

ENV: Python 2.7
POINT: トポロジカルソートと動的計画法

ENV: Haskell
POINT: 日付をヒントに有向グラフを作成し動的計画法による最長パス探索で解きます。

前回は見事に罠にはまって評価 3 でしたが、今回は大丈夫……だと思います。

ENV: 蛍光ペンとルーズリーフ
POINT: テトラ式「例示は理解の試金石」(全部書く)
領域分割してから評価値 (チケットの枚数) が最大になるように書けばいいのかなーと思い、どう領域分割するかをルーズリーフに書きながら考えていたらそのままゴリ押しで回答になりました・・・。

(結城：この方は評価 5 でした)

ENV: Python 3.4.0
POINT: 帰国日の早い国から選んでいくこと。国の名前を出発日順で回答しないこと (しそうになった)。
日付の比較のために Python の日付型を使用しました。この型では年も与えなければいけないため，適当に 2014 年を指定しましたが，プログラム実行中になぜかエラーになりました。出発日にうるう日が含まれていることが原因でした。年をうるう年である 2012 年に変えて実行したらうまくいきました。
国の名前を出発日順で回答したものを提出しそうになりましたが，提出形式を確認している時に間違いに気づきました。危なかったです。

ENV: Perl
POINT: ソートの利用と罠に怯えない心
とりあえず出発日順に並べてあとから出てきたやつの帰国日が早ければ上書きしてみました
何か罠がありそうで現段階ではイマイチ確信が持てないです…

ENV: Python 2.7.5
POINT: 出力条件を見落とさないこと、または蟻本を積読しないこと
1. 旅行期間を長くする必要はない
2. 一つだけ出力すればよい
という条件から、「重み付け無しの区間スケジューリング問題」になるため、帰国日についての貪欲法で再帰的に解きました。
出力はアルファベット順指定のため国名でソートしても良かったのですが、
「tickets.txt に書かれている順」という補足があったため、
念のため初期順序を保持しておいて出力時にそれをキーにソートしています。

ENV: C
POINT: 年の後ろから動的計画法。(その日がある国に出発可能なら、帰国後にいける国の数 +1 が現状最良かをメモしていく)
日付は月*31+ 日で integer に変換 (今回は特に期間が問題になるわけではないので、1 日以上の差と複数日の差が特にない)
帰国日をあえて入力時に 1 日ずらすことで、「次の出発可能日」として解釈した。

最初は間違えて出発可能日をそのままやってしまったので 73 になったが、デバッグ用に日付を出した際に気づいた。

とりあえず、こんな旅行プランは要らない。疲労で死んでしまう・・・
あと、オーストリアに行って次の日帰ってくるってなんですかこれ。それだけで死ねますよ・・・

ENV: Python
POINT: 帰国日が早い順に選んでいく。出力するときに国名をアルファベット順にするのを忘れそうになった

ENV: Perl
POINT: 問題を適度なサイズに分割して解くこと。

・単純な brute force で解こうとすると O(2^n) になってしまい現実的ではないため、
  まずチケット全体を期間が重なり合うチケットの独立なグループに分割し、
  そのグループ内で brute force で探索した結果を最後にまとめる戦略を取っている。

ENV: Haskell
POINT: オーバーラップしていない日程群を分けることで組み合わせ爆発を抑える.

ENV: C
POINT: 出発日と到着日の関係
感想:
前回のスペーストーキーでは、残念ながら評価 3 だったので再挑戦させて頂きました。
どうしてもポイントなる点に気付けず、いつもなら何かしら引っかけがあるはずなのにと
疑いながら解きました。結局、終わりまで気付けず終いでした。

ENV: Haskell
POINT: オーバーラップしていない日程群を分けることで組み合わせ爆発を抑える.

「挑戦者の声から」は以上です！

3.9 コード集

以下に、〆切後、結城浩（@hyuki）あてに送っていただいた挑戦者さんのコードをまとめました。

- http://togetter.com/li/669617

3.10　最後に

さて今回のチケットゴブル問題はお楽しみいただけたでしょうか。
結城が出題した問題については、ブログやツイッターなどでの言及、参照、引用は大歓迎です。
ぜひ、このフィードバックに対するご感想も `@hyuki` までお聞かせくださいね。
結城はこれからも CodeIQ で出題していきますので、またチャレンジしてください。
今回は挑戦ありがとうございました。ではまた！

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