AtCoder Beginner Contest 013
解説

はじめに

・競技プログラミングが初めての人
・まだまだ競技プログラミングに慣れていない人

最も基礎的なことは過去の解説に載っています！
特に ABC004 の解説が詳しいです。
http://www.slideshare.net/chokudai/abc004

AtCoder の練習用コンテストも活用しましょう！
http://practice.contest.atcoder.jp/

A問題

A

A - 問題概要

問題

英語のアルファベット X が与えられるので、

`A` から数えて何番目のアルファベットかを答えよ。

制限

X は `A`, `B`, `C`, `D`, `E` のどれか

A - プログラムの流れ

N を**入力**する

↓

答えを**計算**する

↓

答えを**出力**する

入力、**出力**でつまずく場合

→入出力について復習！

http://www.slideshare.net/chokudai/abc004

http://practice.contest.atcoder.jp/

A - プログラムの流れ

N を**入力**する

↓

答えを**計算**する

↓

答えを**出力**する

入力、**出力**でつまずく場合
→入出力について復習！

http://www.slideshare.net/chokudai/abc004

http://practice.contest.atcoder.jp/

残りは答えの**計算**

↓

答えを**計算する方法（＝アルゴリズム）**を考えよう！

A - 解き方

いろんな解き方がある

・条件分岐をたくさん書く
・ループを回して調べる
・文字コードを利用する

A - 解き方

いろんな解き方がある

・**条件分岐をたくさん書く**
・ループを回して調べる
・文字コードを利用する

```
if X == ‘A’:
  res = 1
else if X == ‘B’:
  res = 2
else if X == ‘C’:
  …
```

特徴

記述量が多くミスをしやすい

アルファベットの種類が増えたときに大変

A - 解き方

いろんな解き方がある

・条件分岐をたくさん書く
・**ループを回して調べる**
・文字コードを利用する

```
S = “ABCDE”
for i = 0 to 4:
  if S[i] == X:
    res = i + 1
```

特徴

さっきの条件分岐羅列の欠点を克服
（こっちの方がおすすめ）

A - 解き方

いろんな解き方がある

- 条件分岐をたくさん書く
- ループを回して調べる
- **文字コードを利用する**

C言語

```
res = X - ‘A’ + 1;
```

Python

```
res =
 ord(X) -ord(‘A’) + 1
```

特徴

文字コードに依存する方法だが、コンテストでは
ふつう ASCII 以外考えなくてよい。

B問題

錠

B - 問題概要

問題

1 桁のダイヤルロック錠がある。

数字を a から b に変えることを考える。

1 回の操作で数字を次のものか前のものに変えられるとき、必要な最低の操作回数は？

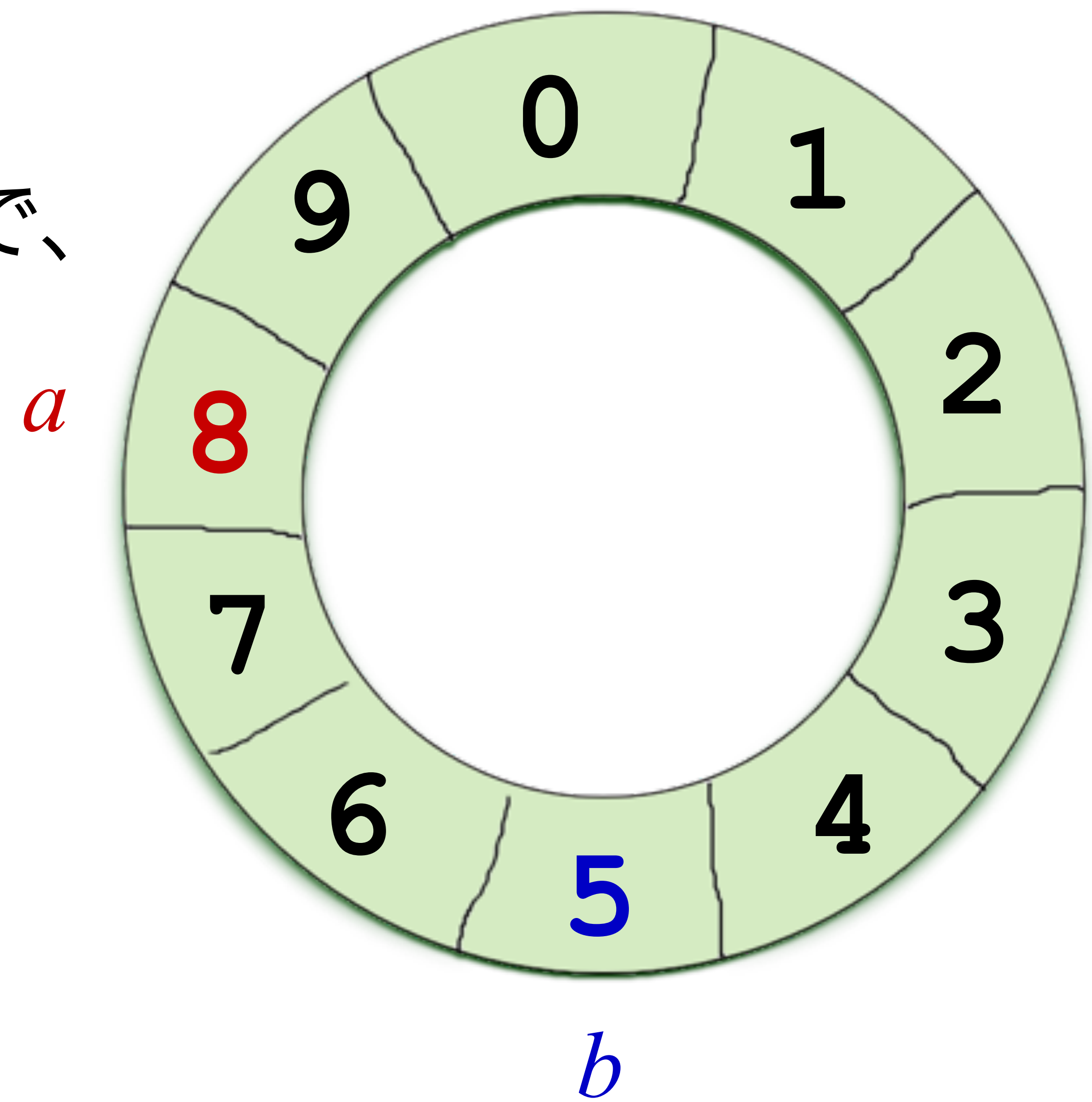
制限

$0 \leqq a \leqq 9, 0 \leqq b \leqq 9$
$a \neq b$

B - 問題言い換え

右図のようなリングの上で、
a から b への距離は？

ととらえると考えやすい



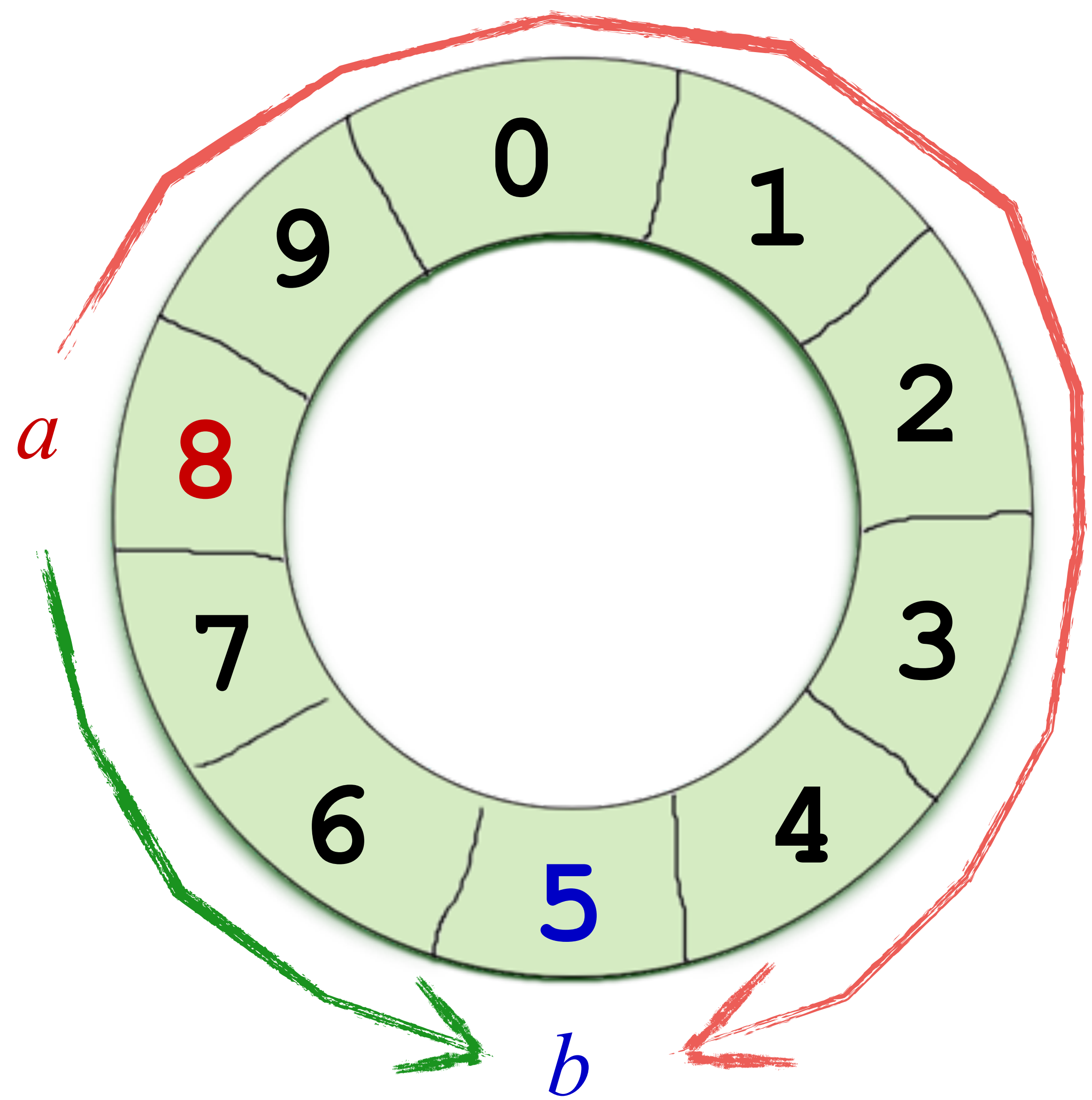
B - 解き方

a から b への行き方は

・**時計回り**に回る
（数字が大きくなる方向）

・**反時計回り**に回る
（数字が小さくなる方向）

の 2 通り考えられる。



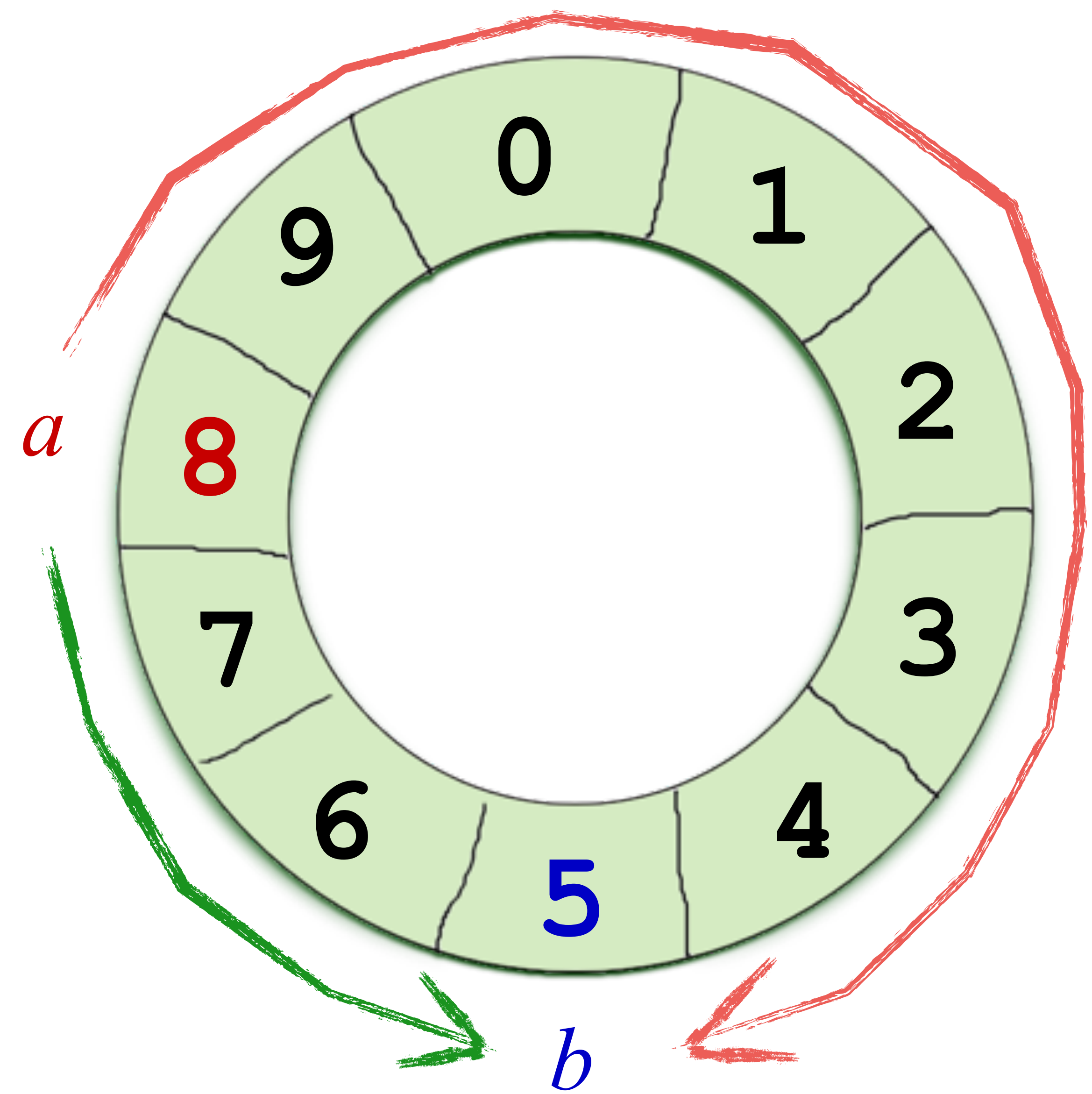
B - 解き方

a から b への行き方は

・**時計回り**に回る
（数字が大きくなる方向）

・**反時計回り**に回る
（数字が小さくなる方向）

の 2 通り考えられる。



両方の場合の距離を計算して、小さいほうが答え！

B - プログラム

```
int next_digit(int d) {
  if (d == 9) { return 0; }
  else        { return d + 1; }
}

int distance1(int a, int b) {
  int res = 0;
  while (a != b) {
    ++res;
    a = next_digit(a);
  }
  return res;
}
```

ループを回して何回で b になるかを調べる

B - おまけ

引き算をうまく使って
もっと簡潔に解くこともできます

```
diff = abs(a - b)
res = min(diff, 10 - diff)
```

C問題
節制

C - 問題概要

問題

N 日間の食事をそれぞれ次の中から 1 つ選ぶ:

・普通の食事（A 円出費、満腹度 +B）

・質素な食事（C 円出費、満腹度 +D）

・食事抜き（0 円出費、満腹度 -E）

最初の満腹度 H で、最低何円で N 日乗りきれるか？

N 日間で一度も満腹度が 0 以下になってはいけない

C - 問題概要

制約

$1 \leqq N \leqq 5\times10^5$

$1 \leqq H \leqq 10^9$

$1 \leqq C < A \leqq 10^6$

$1 \leqq D < B \leqq 10^6$

$1 \leqq E \leqq 10^6$

ただし、$N \leqq 5\times10^5$ はボーナス点の制約で、

$N \leqq 1{,}000$ のケースに正解すれば 100 点

C - 部分点解法

10点 ($N \leqq 10$)

N がとても小さいので、それぞれの日にどの種類の食事をとるかを全探索しても間に合う。

30点 ($N, H, B, D \leqq 50$)

n 日目までが終わった段階で満腹度が h という状態に至る最小の費用を**動的計画法**で計算する。

C - 重要な考察

N 日間の食事、計 N 回のうち

・X 回は普通の食事をとる

・Y 回は質素な食事をとる（食事抜きは N-X-Y 回）

と**回数**を決めたとする。

このとき、食事の順番をうまく工夫して N 日間を

乗り切ることができるかどうかを知りたいが、

最初の X+Y 日に全部食事をとる（食いだめをする）

と考えても構わない！

C - 食いだめができる器用な人

食事抜きの日のあとに普通の食事をとる場合

最終的に満腹度は $-E + B$ だけ変わるが、

1日目の $-E$ の時点で 0 以下になってしまうかも

普通の食事をとった日のあとに食事を抜く場合

最終的な満腹度は $B - E$ だけ変わり、

1日目の段階では満腹度が 0 以下になることはない

満腹度に上限がないので
先に食べておいたほうが絶対に得！

C - 100点解法

普通の食事をとる回数(X)と
質素な食事をとる回数(Y)を全部試す

各 (X, Y) について、N 日間を乗りきれるかチェック

$$\boldsymbol{H + BX + DY - (N - X - Y)E > 0}$$

が成り立つかどうかを調べればよい。
（食べる方を優先するので、最終的に満腹度が 0 より大きければ
途中でも絶対に大丈夫）

計算量は $O(N^2)$ なので $N \leqq 1{,}000$ で間に合う

C - ボーナス点解法

100 点解法において

普通の食事をとる回数(X)だけを決めると……

条件

$$H + BX + DY - (N - X - Y)E > 0$$

を式変形して

$$Y > \{(N - X)E - H - BX\} / (\mathrm{D} + \mathrm{E})$$

とすれば、

質素な食事を最低何回とればいいかが計算できる

→ これで計算量が O(N) となって 101 点

D問題
阿弥陀

D - 問題概要

問題

縦線 N 本、横線 M 本のあみだくじを縦に D 個
つなげてできる巨大なあみだくじを考える。
この巨大あみだくじにおいて、それぞれの縦線から
くじを引くとどこに行き着くかを求めよ。

制限

$2 \leqq N \leqq 10^5$, $0 \leqq M \leqq 2\times10^5$
$1 \leqq D \leqq 10^9$

D - 部分点解法

10点 ($D = 1$)

縦につなげないので、与えられたあみだくじを
そのままシミュレーションすればよい。

1 本ずつシミュレーションするより
まとめて考えたほうが楽

```
to = [1, 2, …, N]
for a in reversed(A):
  swap(to[a], to[a - 1])
```

D - 問題の言い換え

10点分の解法を使って、つなげる前のあみだくじについて、何番目の縦線で終わるかを計算できる。

x 番目の縦線から始めたときに $T[x]$ 番目の縦線で終わることを表す配列 T を用意する。

すると、2 個あみだくじをつなげたとき
x 番目の縦線から始めると $T[T[x]]$ 番目の縦線に着く

D - 部分点解法2

20点 ($N \leqq 1{,}000$, $D \leqq 1{,}000$)

10点の解法を使って T を計算したあと、各 k に対し

$T[T[...(T$ が全部で D 個$)...T[k]...]]$ を求める。

20点 ($N \leqq 8$)

$(1,2,\ldots,N) \rightarrow (T[1],T[2],\ldots,T[N]) \rightarrow (T[T[1]], \ldots$
という変換は少なくとも $N!$ 回でループするので、

最初に D を $N!$ で割った余りをとっておく。

D - 満点解法

$T[T[...(T$ が全部で D 個$)...T[k]...]]$ のことを

これ以降では $\boldsymbol{T_D[k]}$ と書くことにする。

目標

普通に計算するとひとつの k に対してですら

O(D) もの時間かかってしまうので、

なんとかして高速に $T_D[k]$ を計算したい。

D - ダブリング

$$T_1[k] = T[k]$$
$$T_2[k] = T_1[T_1[k]]$$
$$T_4[k] = T_2[T_2[k]]$$
$$T_8[k] = T_4[T_4[k]]$$
$$\cdots$$

と計算していくと、D が 2 の累乗のときには

$T_D[k]$ を $O(\log D)$ 時間で求めることができる。

D - ダブリング

D が 2 の累乗でないときも

D の 2 進数表示を考えれば、たとえば

$$\boldsymbol{T_{100}[k] = T_{64}[T_{32}[T_4[k]]]}$$

といった風に $O(\log D)$ で $T_D[k]$ が計算できる！

これを使えば全体で $O(N \log D + M)$ 時間となって

満点が得られる。

D - おまけ

実は少し違ったアプローチで $\mathrm{O}(N+M)$ 時間の
アルゴリズムもあります。

キーワード: 置換、巡回置換、置換の積

スライド作ってる時間がなかったので
詳細については読者への練習問題とする。