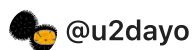


 More than 1 year has passed since last update.



posted at 2020-05-11 updated at 2020-05-11

【AtCoder解説】PythonでABC167のA, B, C問題を制する !

 Python, AtCoder

AtCoder Beginners Contest 167のA,B,C問題を、Python3でなるべく丁寧に解説していきます。

ただ解けるだけの方法ではなく、次の3つのポイントを満たす解法を解説することを目指しています。

- シンプル：余計なことを考えずに済む
- 実装が楽：ミスやバグが減ってうれしい
- 時間がかからない：パフォが上がって、後の問題に残せる時間が増える


5/11 C問題に『`itertools.product(range(2), repeat=n)`』を使う方法を追記しました。

AtCoder Beginners Contest 167

全提出人数: 11930人

パフォ	AC	点数	時間	順位	目安
400	AB----	300	14分	7391位	茶パフォ
600	ABC---	600	87分	5971位	8回で茶レート
800	AB-D--	700	94分	4523位	緑パフォ
1000	ABCD--	1000	82分	3218位	8回で緑レート
1200	ABCD--	1000	50分	2183位	水パフォ

(参考)私：パフォ1515 ABCDE- 87:43 1114位

1114	 unidayo	1500 87:43	100 1:33	200 4:55	300 11:03	400 29:57	500 87:43	(1)
最速正解者			Yu_212 0:29	enjapma 0:53	fuppy0716 2:23	startcpp 1:52	LayCurse 3:48	riantkb 8:59
正解者数 / 提出者数			11507 / 11798	10836 / 11474	5652 / 6388	4666 / 7436	1393 / 1828	415 / 1649

A問題 (11507人AC) 『Registration』

与えられるTが、必ずSより1文字長いことを利用すると楽です。

B問題 (10836人AC) 『Easy Linear Programming』

貪欲に1から先に取ればいいです。

C問題 (5652人AC) 『Skill Up』

参考書は最大12冊なので、それぞれ買うか買わないか、2の12乗=4096通り全てを確かめます。

D問題 (4666人AC) 『Teleporter』 [この記事では解説しません]

町の数がそんなに多くないので、どこかでループになることを利用します。ダブリングというテクニックを使うとより楽です。

A問題『Registration』

問題ページ：[A - Registration](#)

むずかしさ：☆☆☆☆☆

ポイント：文字列の扱い

サンプルを見ると、問題文が何を言っているかわかりやすいです。

解き方

1. 実装を考える

ステップ1:実装を考える

要するに、 T が S になんでもいいから1文字付け加えたものになっていればいいです。

与えられる T は、必ず S より1文字だけ長いです。よって、 S の文字数を l として、 T の l 文字目までが S と全く同じなら良いです。

サンプル1の `chokudai` と `chokudaiz` なら、`chokudaiz` の8文字目までが `chokudai` ならいいので、Yes です。

サンプル2の `snuke` と `snekee` なら、`snekee` の5文字までが `snuke` ならいいのですが、`sneke` なので No です。すね毛とは

コード

```
s = input()
t = input()
l = len(s)  #sの長さl

if s == t[:l]:  #tのl文字目までが、sと同じか確かめる
    print("Yes")
else:
    print("No")
```

B問題『Easy Linear Programming』

問題ページ： [B - Easy Linear Programming](#)

むずかしさ：★★☆☆☆

ポイント：貪欲法

最大値にしたいので、貪欲に1から先にとっていけばいいです。カードの枚数が最大で20億枚もあることに気をつけましょう。

解き方

1. 解き方を考える

ステップ1: 解き方を考える

最大値にしたいので、1のカードはできるだけ取りたいです。逆に-1のカードはできるだけ取りたくないです。ですので、1のカードから先にとって、なくなったら0のカードを取って、それもなくなったら仕方なく-1のカードを取ればいいです。

例えば、 $A = 1, B = 2, C = 3$ なら、1, 0, 0, -1, -1, -1の順番に取ると最大値になります。

そのままシミュレートして、 $s = [1] * a + [0] * b + [-1] * c$ というリストを作り、 $\text{sum}(s[:k])$ とすると解けそうな気がします。しかし、カードは最大で20億枚あるので、この方法ではリストが巨大すぎてMLE（メモリ上限超過）になります。（たぶん数十GB）

ですので、算数をして求める必要があります。といっても難しいことはありません。「1だけ取れる場合」「1と0を取る場合」「1と0と-1を取る場合」の3パターンを考えます。

$A \leq K$ なら1のカードだけ取れるので、答えは $1 \times K = K$ です。

$A < K \leq A + B$ なら、1のカードを全部取って、残りは0のカードを取ります。0のカードは何枚取っても0なので、答えは $1 \times A = A$ です。

最後に、 $A + B < K$ なら、1と0のカードを全部取って、残りは仕方なく-1のカードを取ります。-1を取らないといけない枚数は $K - A - B$ 枚なので、答えは $A - 1 \times (K - A - B)$ です。

コード

少し考えてみると、前の2パターンは一緒にしていいことがわかります。取れる1の枚数は「1の枚数 A 」と「取る枚数 K 」の小さい方なので、 $\min(a, k)$ とすればいいです。

このように、数値に上限があるときは \min を使うと楽です。よく使うので、覚えておくと便利です。

```
a, b, c, k = list(map(int, readline().split()))

if a + b >= k:
    print(min(a, k)) # 取れるだけ1を取る
else:
    print(a - (k - a - b)) # 仕方なく-1を取る
```

C問題『Skill Up』

問題ページ： [C - Skill Up](#)

むずかしさ：★★★★★★

ポイント：bit全探索

アルゴリズムと参考書は、どちらも最大12個です。参考書を買うか買わないか、すべての組み合わせを試しても間に合いそうです。

こういう問題は『**bit全探索**』というアルゴリズムを使うと解けます。強そうな名前が怖いかもしれませんが、結局やることは全部の組み合わせを総当りするだけです。

『bit全探索』は総当りするだけといえそうなので、C問題でもたまに出てきます。しかし、知らないとまず書けないうえ、実装がやや複雑なので、C問題としては最高レベルに難しくなります。(先週のABC165のC問題ほどではありませんが)

『bit全探索』は慣れないと難しいですが、逆に慣れればそこまで難しくはありません。この機会に覚えておいて、類題を解いて少しずつ慣れていきましょう。

私もはじめは、このレベルの問題を解くのに、調べながらやって1時間以上かかりました。何度も使って慣れたおかげで、今回は7分で解けました。本当に慣れだと思えます。

注釈:この解説では、0番目、0桁目からスタートします

わかりやすさのため、参考書の番号は0番目から数え、後で出てくる2進数についても、一番下の桁を0桁目として説明します。

『問題文で1番目』が、Python的には `c[0]` といういつものアレです。

解き方

1. 問題文を理解する
2. 解法を考える
3. 「総当たりの方法」を考える
4. 『bit全探索』の実装を考える

ステップ1:問題文を理解する

アルゴリズムが M 個あります。最大で12個です。各アルゴリズムには「理解度」という数値があって、初期値は0です。

アルゴリズムの参考書が N 冊あります。最大で12冊です。各参考書には「値段 C 」と「読むと各アルゴリズムに加算される理解度 A 」が設定されています。

すべてのアルゴリズムの理解度を X 以上にしたいです。かかる金額の最小値を求めてください。どうしても無理なら -1 を出力してください。

ステップ2: 解法を考える

参考書の数 N が最大で12冊ということに着目しましょう。各参考書について「買う場合」「買わない場合」の組み合わせをすべて考えても、それほど多くならなそうです。

計算してみると、組み合わせ数は最大で $2^{12} = 4096$ 通りです。アルゴリズムも最大12個だけなので、「最大4096通りの買い方に全てについて、全アルゴリズムの理解度が X 以上になるか確かめて、なる場合は金額の最小値を更新する」としても余裕で間に合います。

ステップ3 「総当たりの方法」を考える

「全ての買い方について理解度が X 以上になるか確かめて、なるならそのときの値段で答えを更新すればいい」ということはわかりました。

しかし、どうやって実装するかが問題です。

参考書は1から12冊で決まっていなくて、forループを使うのは無理そうです。決まっていたとしても、12重ループを書くなんてことはやってられません。

この問題のように「使う」「使わない」の2通りについて、全パターンを確かめたいときに都合がいい方法が、はじめに書いた『bit全探索』です。

『bit全探索』はどんなコードになる？

説明が長くなるので、先に『bit全探索』がどんなコードになるか載せておきます。最後に同じものをもう一度載せます。これは『bit全探索』のテンプレなので、使うときは毎回同じようなコードを書けばできます。

コードは2重ループにif文が組み合わさったものですが、`for bit in range(1 << n):` や `if (bit >> 1) & 1` が見慣れないと思います。これらについても説明します。それほど難しいことはしていないので、安心してください。

```
for bit in range(1 << n):
    for i in range(n):
        if (bit >> i) & 1:
            # i冊目を買うときの処理
        else:
            # i冊目を買わないときの処理（今回は何もしないので省略）
```

追記『itertools.product(range(2), repeat=n)』を使う方法

せっくなので、ビット演算なしで『bit全探索』をする方法も書いておきました。

ステップ4B 『itertools.product(range(2), repeat=n)』を試してみる

『bit全探索』の考え方:買うなら『1』、買わないなら『0』

買う場合を 1、買わない場合を 0 と数字にしてみます。すると、110 や 010 のように、1 と 0 を参考書の数 n 個だけ並べた、 n 桁の2進数の数字で買い方を表現できます。

参考書の番号と2進数の数字の桁数を対応させるとわかりやすいので、そうすることになります。1 冊目を買うなら、1 桁目が 1 になるということです。

参考書が3冊で、0冊目、1冊目、2冊目とあるとします。例えば、1,2冊目を買って、0冊目を買わないパターンを表す2進数は、1,2桁目が 1 で、0桁目が 0 なので、110 です。同じように考えて、全て買うなら 111、全て買わないなら 000 です。

ステップ4『bit全探索』の実装を考える

2進数を使うと、数字で組み合わせを表現できることはわかりました。では、どうすれば全部の組み合わせを抜けなしに列挙できるのでしょうか。

参考書が3冊の場合を考えてみます。全部買わない場合は 000 で、全部買う場合は 111 です。

000 から 111 まで 1 ずつ足していくと、000、001、010、011、100、101、110、111 となって、2の3乗=8通り、全ての組み合わせが現れていることがわかります。

つまり、000 からはじめて、111 まで 1 ずつ足していだけでできます。

2進数も結局ただの数字

「2進数なんて使ったことないけど、どうすればいいんだよ」となると思います。

2進数は普段使っている10進数と表し方が違うだけで、同じ数字を表していることに違いはありません。そこで、10進数の整数を普通に使って、操作だけは2進数としてやるというです。

つまり、000 ~ 111 を全探索したいなら、0 から 7 までをforループで回すだけです。これはおなじみの `range(8)` です。この 0 ~ 7 にちょっとした2進数の操作をすれば、bit全探索ができます。

rangeの指定に『左ビットシフト』を使う

参考書の手数は決まっていないので、`range` の範囲は計算で求める必要があります。参考書の手数は `n` なので、「1 が `n` 個続いた2進数を、10進数で表した数字」が範囲の最大値です。

再び、参考書を3冊として説明します。

`range(8)` は0〜7までの8個だというように、`range(x)` は x を含みません。ですので、範囲の最大値の「1 が 3 個続いた 111 を10進数にした 7」に 1 を足して「1 のあとに 0 が 3 個続いた2進数 1000 を、10進数で表した 8」を指定する必要があります。

「1000 を10進数で表した 8」を簡単に作るために、『ビットシフト演算子』というものを使います。『ビットシフト演算子』には、『左シフト `<<`』と『右シフト `>>`』の2種類があります。

「1000 を10進数で表した 8」を作るには、『左シフト `<<`』を使います。`1 << 3` と書くだけです。「1 を2進数で表したものを、左に 3 ビット分ズラす（シフトする）」という計算です。空いたスペースには 0 が入ります。

`1 << 3` は2進数で 1000 なので、10進数の 8 になります。「2進数の 1000 を10進数で表した 8」を作れました。

『ビット演算子』の考え方は2進数ですが、得られる結果は普通の10進数の整数です。`1 << 3` で出てくるのはいつもの使っている10進数の 8 であって、1000 ではありません。

やっていることは2の累乗

実は、`1 << 3` でやっていることは `2 ** 3` と全く同じです。いつも使っているものと同じだとわかると、安心感があると思います。

考えてみると、3冊を買う買わないの組み合わせ数は2の3乗で8通りなので、そうなるのは必然です。

「じゃあわざわざ `<<` なんかわからなくても」と思うかもしれませんが、『ビット演算子』を使ったほうがわかりやすいです。最後に『ビット演算子』を使わない例を載せます。それを見ると、『ビット演算子』を使ったほうがいいと感じると思います。

『i番目を買うか買わないか』確かめるには『右ビットシフト』と『ビットの論理積』を使う

全組み合わせを2進数で表す方法はわかりました。ですが、作った数字から「0番目の参考書は買うのか買わないのか」「1番目は買うのか買わないのか」.....を取り出せないという意味がありません。

つまり「数字を2進数にしたとき、 i 桁目が1なのか0なのか」知りたいです。

これは、さっきと反対の『右ビットシフト \gg 』と『論理積 $\&$ 』合わせるとできます。

左シフト \ll では、空いたスペースに 0 が入りました。反対の右シフト \gg では、右に詰めた分の桁が消滅します。

例えば $6 \gg 1$ は、「 110 を1桁右にズラした2進数 11 」を10進数にした 3 になります。右1桁の 0 は消滅しました。

こうやって「 i 桁右ビットシフトして得られた数字の一番下の桁」が「 1 なら i 桁目は1」「 0 なら i 桁目は 0 」だとわかります。

「 i 桁右ビットシフトして得られた数字」の「一番下の桁が 1 か確認する」にはどうするのかというと、『ビットの論理積 $\&$ 』を利用します。論理積 $\&$ は、「どちらも1なら1、どちらか一方でも0なら0」を桁ごとに行う計算です。

2つを組み合わせると $(bit \gg i) \& 1$ と書くと、 i 桁目が 1 なら 1 、 0 なら 0 が返ってきます。

例えば、 $(6 \gg 1) \& 1$ は、 $11 \& 1$ です。右の 1 は左の 11 より1桁少ないので、足りない部分に 0 が入ります。つまり、 $11 \& 01$ を桁ごとに計算します。0桁目は $1 \& 1 = 1$ 、1桁目は $1 \& 0 = 0$ となるので、 $6 \& 1 = 11 \& 01 = 01 = 1$ です。

1 の0桁目以外は全て 0 なので、どんな数字でも 1 と論理積を取ると、0桁目と同じ 0 か 1 しか返ってきません。これが0桁目が 0 か 1 か取り出せる仕組みです。

forループですべての桁を確認すればいい

for i in range(n) として、すべての桁を『右ビットシフト >>』と『論理積 &』で確認すれば、全ての参考書をそれぞれを買うか買わないか確認できます。

具体的には、最初に載せたこのコードになります。

```
for bit in range(1 << n):
    for i in range(n):
        if (bit >> i) & 1:
            # i冊目を買うときの処理
        else:
            # i冊目を買わないときの処理（今回は何もしないので省略）
```

ビット演算子を使わないでやるとどうなる？

ビット演算子を使わないで同じことをすると、こういうコードになります。かなりわかりづらいです。

```
for x in range(2 ** n):
    for i in range(n):
        if ((x // (2 ** i)) % 2) == 1:
            # i冊目を買うときの処理
        else:
            # i冊目を買わないときの処理（今回は何もしないので省略）
```

コード

『bit全探索』の説明が長くなりました。ようやくこの問題を解くコードの解説に入ります。といっても、この問題はほとんど『bit全探索』をするだけで解けます。

長いので、本体の main 関数、組み合わせを表す数字 bit を受け取って価格を計算する calc_price 関数、理解度の条件を満たすか確認する check 関数に分けて書きます。

関数に渡している変数と、渡していない変数について

作った関数に渡している変数と、渡していない変数があります。

main 関数内で定義した変数は、どの関数内でも使えるので、渡す必要はありません。一方、main 関数以外で定義した変数は、引数として渡す必要があります。『関数のスコープ』というものです。

main関数

まずは本体の main 関数です。最初に入力を受け取ります。空のリストを作って、そこにappendで追加していきます。

次に、答えの初期値を正の無限大にします。これから何度も打つのが面倒なので、`INF = float("inf")` として、INF だけで使えるようにしておきます。

その後、bit全探索のテンプレ通り `for bit in range(1 << n):` で全探索します。

価格の計算は、calc関数に買い方を表す整数 bit を渡して行います。「全ての理解度が x 以上」の条件を満たすならそのときの値段、満たさないなら INF を返します。

最後に、1つでも条件を満たしているなら、ans は INF でないので、ans を出力します。ans が INF のままなら、1つも条件を満たさなかったなので、-1 を出力します。

```
if __name__ == '__main__':
    n, m, x = list(map(int, input().split()))

    # 空のリストを作って、appendで追加していきます
    c = [] # 参考書の値段です
    a = [] # 各参考書に入る理解度です

    for i in range(n):
        c_temp, *a_temp = list(map(int, input().split())) # こうすると、2つ目以降を
        c.append(c_temp)
        a.append(a_temp)

    INF = float("inf")
    ans = INF
```

```

for bit in range(1 << n):
    price = calc_price(bit) # 条件を満たすならそのときの値段、満たさないならINFが返
    ans = min(ans, price) # 答えを更新します

if ans == INF:
    print(-1) # どうやっても条件を満たせなかった
else:
    print(ans)

```

calc_price関数

次は、値段を計算する calc_price 関数です。bit (int型の整数) を受け取って、値段 price_total と各アルゴリズムの理解度 learn_total を計算します。

ここもテンプレ通りに、for i in range(n) と if (bit >> i) & 1 を組み合わせて、どの参考書を買うか確認して計算します。

計算が終わったら、各アルゴリズムの理解度 learn_total がすべて x 以上か判定するcheck関数に、learn_total を渡します。

条件を満たすなら、main関数に値段 price_total を返します。満たさないなら、正の無限大 INF を返します。

```

def calc_price(bit):
    price_total = 0 # 値段の合計
    learn_total = [0] * m # 理解度

    for i in range(n):
        if (bit >> i) & 1: # i桁目が0か1か見て、i冊目を買うか買わないか判定します
            price_total += c[i] # 買うので、i冊目の値段を加算します
            learn = a[i] # 「i冊目の参考書を読んで増える、各アルゴリズムの理解度」のリスト
            for j, la in enumerate(learn): # 理解度を加算します enumerateを使うと、r
                learn_total[j] += la

    # 条件を満たすか、check関数で確認します
    if check(learn_total):
        return price_total ... # 条件を満たすので、値段を返します

```

```
else:
    return INF # 条件を満たさないので、INFを返します
```

check関数

最後に、条件を満たすならTrue、満たさないならFalseを返すcheck関数です。そのままforループを使って確認しているだけです。

ここまで分けなくても大丈夫ですが、読みやすさのために分けておきました。

```
def check(learn_total):
    for i in range(m):
        if learn_total[i] < x:
            return False # 1個でもx未満ならFalse
    return True # 全部x以上ならTrue
```

『bit全探索』の類題

『bit全探索』は慣れです。何回も解いているとわかってきます。ということで、『bit全探索』を使う問題をいくつか探してきました。

どの問題も、制約の数字のうち1つが10～16くらいという特徴があります。やたらと小さい制約があるときは、bit全探索で総当たりして解けないか考えてみましょう。

基本

bit全探索に慣れるのに向いていそうな、それほど複雑でない問題です。とはいえ、bit全探索自体が少し複雑なので、それなりの難易度はあります。

[ABC129C - switches](#)

[ABC147C - HonestOrUnkind2](#)

Donutsプロコンチャレンジ2015B - Tokyo 7th シスターズ

ABC002D - 派閥

応用

少し難しくなりますが、やることはbit全探索です。これができれば、bit全探索に関しては緑後半～水色前半レベルだと思っています。

ABC104C - AllGreen

ABC119C - Synthetic Kadomatsu

チャレンジ

これが1時間くらいで解けたら、私より強いです。考え方が複雑というわけではなく、ひたすら実装が重い問題です。

ABC159E - Dividing Chocolate

ステップ4B『itertools.product(range(2), repeat=n)』を使ってみる

ビット演算に慣れておくとその先有利ですが、Pythonにはもっとわかりやすい方法があります。

それは、 `itertools` モジュールの `product` メソッドを使う方法です。

`product(range(2), repeat=n)` と書くと、 `0` か `1` (`range(2)`) を `n` 回繰り返してできる組み合わせがすべて作れます。

どんなものができるのか、中身を見てみましょう。

```
from itertools import product
```



```
n = 3
product(range(2), repeat=n)
<itertools.product object at 0x03C973A8>
```

range と同じようなオブジェクトなので、このままでは中身を見れません。中身をすべて見たい場合は list に渡すといいです。

```
list(product(range(2), repeat=n))
[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)]
```

確かに長さ3の8組が全て生成されています。各要素はタプルなので、添字でアクセスできます。タプルの i 番目が 1 なら買う、0 なら買わないとすればいいです。

これをforループで回せば、『bit全探索』と同じことができます。forループで回す場合は、リストに変換する必要はありません。

```
for bit in product(range(2), repeat=n):
```

コード

入力と出力部は省略しました。それほど長くならなかったなので、別関数に分けずに書きました。

```
from itertools import product

# 入力部は省略

for bit in product(range(2), repeat=n):
    price_total = 0
    learn_total = [0] * m
    for book, is_buy in enumerate(bit):
        if is_buy:
            price_total += c[book]
            for algo, le in enumerate(a[book]):
                learn_total[algo] += le
    is_ok = True
```

```
for i in range(n):  
    if learn_total[i] < x:  
        is_ok = False  
if is_ok:  
    ans = min(ans, price_total)
```

出力部も省略



Why not register and get more from Qiita?

Sign up

Login



@u2dayo

AtCoder (Python 青) | TOEIC980点 (2019/05) | データベーススペシャリスト (R3秋) | ネットワークスペシャリスト (R3春) | 応用情報技術者 (R2秋) | つよくなりたいむしょく

Follow



Comments

No comments

Sign up for free and join this conversation.

[Sign Up](#)

If you already have a Qiita account [Login](#)

Being held events



2022年に流行る技術予想

2022/01/05~2022/01/31

[View Event Details](#)



How developers code is here.

© 2011-2022 Qiita Inc.

Guide & Help

- About
- Terms
- Privacy
- Guideline
- Design Guideline
- Feedback
- Help
- Advertisement

Our service

- Qiita Team
- Qiita Jobs
- Qiita Zine
- Official Shop

Contents

- Release Note
- Event
- Advent Calendar
- Qiita Award
- API
- Qiitadon (β)

Company

- About Us
- Careers
- Qiita Blog

SNS

- @Qiita
- @qiita_milestone
- @qiitapoi
- @Qiita