ABC 119 解説

出題・解説: @evima0

2019 年 2 月 24 日

A: Still TBD

正解までの道のりを細かく分割すると次の通りです。

1. 標準入力から文字列 s を受け取る。(または、別の何らかの形で入力を受け取る。)
2. s が表す日付が 2019/04/30 以前であるか判定し、答えとなる文字列 Heisei または TBD を得る。
3. 得た文字列を標準出力に出力する。

手順 1, 3 については practice contest 問題 A の言語別サンプルコードが参考になります。手順 2 については、まず if 文などの何らかの条件分岐を行う機構を用いる必要があります。それに加えて、s と文字列 2019/04/30 を辞書式の順序で直接比較するのが一つの手です。文字列を辞書式順序で比較する機能は現代の多くの言語に標準で実装されており、「自前」で実装する必要はありません。Python3 での実装例を示します。

```
S = input()
print('Heisei' if S <= '2019/04/30' else 'TBD')
```

言語によっては、入力をはじめから整数と文字の並びとして受け取り、「月」の部分が 4 以下であるか否かを判定するという方針も大いに考えられます。C++ での実装例を示します。

```
#include <iostream>
using namespace std;
int main(){
    int y, m, d;
    char c1, c2;
    cin >> y >> c1 >> m >> c2 >> d;
    cout << (m <= 4 ? "Heisei" : "TBD") << endl;
}
```

B: Digital Gifts

親戚の人数が 2 人、3 人などと固定されていれば問題 A と大差ありませんが、実際の親戚の人数 N は 2 以上 10 以下と固定されておらず、それに対応するために何らかの「ループ機構」を使うことになります。その代表例が for 文で、この問題でも for 文を使うのが素直でしょう。以下、プログラムで答えを求めるための機械的な手順を述べます。

まず標準入力から N の値を読み込み、ans という値を宣言して 0 で初期化します。そして次に述べる処理を N 回繰り返すと、最終的に ans に求めたい値が入ります。i 回目の処理 $(1 \leq i \leq N)$ は次の通りです:「標準入力から小数 x_i (整数も小数の一種と考えられる) と文字列 u_i を読み込む。$u_i =$ `JPY` なら ans に x_i を足す。$u_i =$ `BTC` なら ans に $x_i \times 380000.0$ を足す」。

これを Python3 で実装したコードが以下です。

```
N = int(input())
rate = 380000.0
ans = 0.0
for i in range(N):
    x, u = input().split()
    x = float(x)
    ans += x * rate if u == 'BTC' else x
print('{:.11f}'.format(ans))
```

なお、小数を出力する際は基本的に桁数を多めに (最低でも 10 桁) 持たせることを勧めます。言語と問題によっては、小数を出力する際の標準の桁数に頼ると精度が足りないことがあるためです。

C: Synthetic Kadomatsu

竹の数は 8 本までと少ないですが、それでも三種類の魔法による行動パターンの総数は極めて多いです。

しかし、足し算の結果は順序を入れ替えても変わらないため、例えば「竹 X に延長魔法→竹 Y に延長魔法→竹 X, Y に合成魔法を使用して竹 XY とする→竹 XY に延長魔法」という一連の行動は、「竹 X, Y に合成魔法を使用して竹 XY とする→竹 XY に延長魔法を 3 回」という一連の行動と同等です。

このように、「以後合成魔法を使用しないような竹に対してのみ延長・縮小魔法を使用する」ような行動パターンのみを考えると、可能な行動パターンを本質的にすべて列挙することが可能になります。

具体的には、竹 $1, 2, \ldots, N$ のそれぞれに対し、その竹の扱いは本質的には次の 4 通りに限られます:「長さ A の竹の “材料” とする」「長さ B の竹の材料とする」「長さ C の竹の材料とする」「使わない」。この合計 4^N 通りの竹の扱い方をすべて試行します (最大で $4^8 = 65536$ 通りで、この数は十分現実的です)。

ただし、“無” に対して延長魔法を使って竹を得ることはできないことに注意します。すなわち、「長さ A の竹の材料」「長さ B の竹の材料」「長さ C の竹の材料」がそれぞれ一本以上必要で、これを満たさない竹の扱い方は棄却します。

上記の条件を満たす竹の扱い方に対しては、その場合の「長さ A の竹の材料」をすべて合成して竹を得て、延長魔法と縮小魔法のうち適切な方を適切な回数だけ使って長さ A の竹を得るためのコストを計算します。長さ B, C の竹についても同様に得るためのコストを計算して合計すれば、その竹の扱い方で達成できる最小のコストを計算できます。以上を 4^N 通りすべての竹の扱い方に対して行って得られた最小値が答えです。

```python
N, A, B, C = map(int, input().split())
l = [int(input()) for i in range(N)]
INF = 10 ** 9

def dfs(cur, a, b, c):
    if cur == N:
        return abs(a - A) + abs(b - B) + abs(c - C) - 30 if min(a, b, c) > 0 else INF
    ret0 = dfs(cur + 1, a, b, c)
    ret1 = dfs(cur + 1, a + l[cur], b, c) + 10
    ret2 = dfs(cur + 1, a, b + l[cur], c) + 10
    ret3 = dfs(cur + 1, a, b, c + l[cur]) + 10
    return min(ret0, ret1, ret2, ret3)

print(dfs(0, 0, 0, 0))
```

D: Lazy Faith

一つの問い x_i について考えましょう。地点 x_i より西にある神社のうち最も東にある神社を s_a、x_i より東にある神社のうち最も西にある神社を s_b とします。もとの設定のままでは神社 s_a, s_b のいずれかが存在しない可能性がありますが、道路のはるか西とはるか東、訪れる価値がないような位置 (例えば、道路の西端から 10^{18} メートル西と 10^{18} メートル東) にも神社があるとすると神社 s_a, s_b はどちらも必ず存在します。

このとき、道路上に s_a, s_b 以外の神社は存在しないとしても答えは変わりません。なぜなら、もしその他の神社 s_o を訪れることがあれば、その過程ですでに s_a, s_b のいずれかを通過しているはずで、s_o をあえて訪れる必要はないためです。同様に寺 t_c, t_d を定義すると、それら以外の寺は存在しないとしても答えは変わりません。よって、s_a, s_b, t_c, t_d のうちどれをどの順に訪れるかの選択 8 通り[*1]をすべて試せば答えが求まります。

あとは、s_a, s_b, t_c, t_d をどのように高速に求めるかが課題です。b が求められれば、$a = b - 1$ であり c, d も同様に求められるため、以下では b を求めることを考えます。

b をより数学的に定義すると「$s_j > x_i$ であるような最小の整数 j」となります ($s_1, \ldots, s_A$ がはじめから昇順に並んでいることに注意)。このような値を高速に求めるアルゴリズムに 二分探索 (リンク先は Wikipedia の同名の記事) があり、現代の多くの言語に標準で実装されています。これを用いると、$\lceil \log_2(A+2) \rceil$ ステップ[*2]の演算で b を求めることができ、10 万個の問いに対しても十分高速です。[*3]

```python
import bisect
A, B, Q = map(int, input().split())
INF = 10 ** 18
s = [-INF] + [int(input()) for i in range(A)] + [INF]
t = [-INF] + [int(input()) for i in range(B)] + [INF]
for q in range(Q):
    x = int(input())
    b, d = bisect.bisect_right(s, x), bisect.bisect_right(t, x)
    res = INF
    for S in [s[b - 1], s[b]]:
        for T in [t[d - 1], t[d]]:
            d1, d2 = abs(S - x) + abs(T - S), abs(T - x) + abs(S - T)
            res = min(res, d1, d2)
    print(res)
```

[*1] $s_a \to t_c, s_a \to t_d, s_b \to t_c, s_b \to t_d$ とこれらの逆順。これらの中には通り過ぎた寺社に戻ってくるような無意味な歩き方もありますが、最適解を求める上で害はありません

[*2] $\lceil x \rceil$ は x 以上の最小の整数です。なお、数式中の “+2” は西の果てと東の果てに足した神社に対応します

[*3] ただし “遅い” 言語ではあまり余裕がなく、上記のコードも最適化が甘くかろうじて間に合う程度です