AtCoder Beginner Contest 123 解説

April 6th, 2019

Problem Setters : E869120, square1001

Notice for Internatioal Readers: Currently, AtCoder is not preparing editorial for ABC-only contests.

問題 A : Five Antennas

この問題は、問題文の通りに、次の 10 個の条件をすべて満たしているか判定し、すべて満たしているならば “Yay!”、そうでないならば “:(” と出力すると解くことができます。

1. $-k \leq a - b \leq k$
2. $-k \leq a - c \leq k$
3. $-k \leq a - d \leq k$
4. $-k \leq a - e \leq k$
5. $-k \leq b - c \leq k$
6. $-k \leq b - d \leq k$
7. $-k \leq b - e \leq k$
8. $-k \leq c - d \leq k$
9. $-k \leq c - e \leq k$
10. $-k \leq d - e \leq k$

しかし、これをすべて判定していると実装をするのにかなり手間がかかります。

そこで、「$a < b < c < d < e$」という制約に注目します。そこで、最も遠いアンテナは A と E なので、結果的に A と E の座標しか気にしなくてもよいのです！よって、次のたった 1 つの条件を満たしているか判定すればよいです。

1. $e - a \leq k$

よって、簡単にこの問題を正解することができます。入力・出力は、C++ では cin/cout を使う、Python では input、print を使うなどをすると実装することができます。また、条件分岐は、多くのプログラミング言語では if/else を使うと実装することができます。

分からない人は、AC している他の人のソースコードを見ると勉強になるかもしれません。

ソースコード（C++）: https://atcoder.jp/contests/abc123/submissions/4840685

問題 B：Five Dishes

主に、この問題には 2 通りの解法があります。1 つ目の方法は「5 つの料理を注文する順番を $5! = 120$ 通り全探索する」もので、2 つ目の方法は「最後に注文した方が良さそうなものを決める」ものです。そのふたつの解法について説明していきます。

解法 #1: 5 つの料理を注文する順番を全探索する

注文する順番が決まると、「料理が来たときに、次に注文できるタイミングですぐ次の料理を注文する」という方法で 5 つの料理を注文していくのが最適です。

例えば $A = 13, B = 17, C = 21, D = 25, E = 29$ で、「ABC 丼 → ARC カレー → AGC パスタ → APC ラーメン → ATC ハンバーグ」の順に注文するとき、次のようになります。

- 時刻 0 に ABC 丼を注文し、13 に届く。次に注文できるのは時刻 20
- 時刻 20 に ARC カレーを注文し、37 に届く。次に注文できるのは時刻 40
- 時刻 40 に AGC パスタを注文し、61 に届く。次に注文できるのは時刻 70
- 時刻 70 に APC ラーメンを注文し、95 に届く。次に注文できるのは時刻 100
- 時刻 100 に ATC ハンバーグを注文し、129 に届く。(次に注文できるのは時刻 130)

「注文の順番」は $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$ 通りあります。この全てに対して時刻いくつで 5 つの料理すべてが届くかを計算し、そのうち最も速いものが答えになります。

解法 #2: 最後に注文した方が良さそうなものを求める

先ほどの例に着目してみましょう。ABC 丼は時刻 0 に注文され、時刻 13 に届き、次に注文できるのは時刻 20 なので、「注文から次の注文までの時間は 20 分」です。一方、AGC パスタは時刻 40 に注文され、時刻 61 に届き、次に注文できるのは時刻 70 なので、「注文か次の注文までの時間は 30 分」です。

実は、この「注文から次の注文までの時間」は、どのような順番で注文しても変わりません！これは、注文してから次の注文までの時刻が 10 の倍数であるからです。つまり、調理時間 X 分の料理について、注文してから次の注文までの時刻は「X 以上の最小の 10 の倍数分」になります。

つまり、最後に注文する料理以外は「X 以上の最小の 10 の倍数分」かかるとみなしてよいです。最後に注文する料理は、調理時間そのままの時間かかるので、「(X 以上の最小の 10 の倍数) と X の差」が最大になるような料理を選べば、最短の時間になります。

解法 1 のソースコード (C++)： https://atcoder.jp/contests/abc123/submissions/4840052
解法 2 のソースコード (C++)： https://atcoder.jp/contests/abc123/submissions/4840010

問題 C：Five Transportations

◆ ステップ #1：貪欲的な方法でシミュレーション

人を都市 6 に向かって「限界の人数まで進ませられるだけ進ませる」貪欲的な方法を考えます。この問題の入力例 1 では、まさにこの「貪欲的な方法」で人を動かしています。直感的に、これが最適そうな感じがします (そしてこれは実に最適な方法です)。

しかし、これを単純にシミュレーションしていると、移動時間に比例する計算時間がかかります。個の問題の制約を見ると $N, A, B, C, D, E \leq 10^{15}$ となっていますので、最悪ケースで $10^{15} + 4$ 分の移動時間がかかります ($N = 10^{15}, A = 1, B = 1, C = 1, D = 1, E = 1$ の場合)。つまり、貪欲的な方法でシミュレーションをすると「実行時間超過 (TLE)」となり、不正解となります！

もっと高速な方法を考えてみましょう・・・が、しかし、単純にシミュレーションを工夫して高速化するのは難しいので、別の方向からこの問題を考えてみることにします。

◆ ステップ #2：最も「きつい」交通機関を考える

最も通りにくい、つまり容量が最も小さい交通機関を考えます。直感的には、最短時間で移動する場合でも、1 分でたどり着けるのは X 人 (交通機関の容量の最小を X とする) になります。これは、「1 秒に 15 L のペースで水を送れるポンプと、1 秒に 12 L のペースで水を送れるポンプをつなげたときに、1 秒に送れる水の量は小さい方を取って 12 L」になるのと同じ原理です。

つまり、先ほどの貪欲的な方法ではなくて、1 分に X 人ずつ 1 つ先に進めてやっても、直感的にはかかる時間は変わらなさそうです。また、移動時間は「($N \div X$ を切り上げた値) + 4 分」と、簡単に計算できるようになります。

これが最短である証明が気になる人もいるかもしれないので書いておきます。

都市 i と $i+1$ を結ぶ交通機関が最小容量 X であるとき、この交通機関にはスタートから $i-1$ 分後から人が乗り始め、ゴールから $5-i$ 分前には人が全員この交通機関を使い終わっていなければなりません。合計で $(i-1)+(5-i)=4$ 分間は“誰も乗らない時間”があります。また、N 人が容量 X の交通機関を使うので最小でも ($N \div X$ を切り上げた値) 回使うことになるので、最低でも“($N \div X$ を切り上げた値) + 4 分”必要になることが分かります。

よって、$ceil(x)$ を実数 x を切り上げた値として、答えは“$ceil(N \div \min(A, B, C, D, E)) + 4$ 分”と、簡単な式で表すことができました！

ソースコード (C++)： https://atcoder.jp/contests/abc123/submissions/4840678

問題 D : Cake 123

この問題では、一番単純な解法として、XYZ 通りを全探索して、全てのコストの合計をソートして、大きい順から K 番目までを順に出力する、という方法が考えられます。

しかし、N 要素のソートの計算量は $O(N \log N)$ です。[1] そのため、この単純な解法の計算量は $O(XYZ \log XYZ)$ となり、$X, Y, Z = 1000$ の時 3×10^{10} 回程度の計算がかかり、実行時間制限の 2 秒に間に合いません。

そこで、ある工夫をするとこの問題が解けます。解法は色々とあり、今回はこのうちの 4 通りを説明したいと思います。

解法 #1 – 工夫したソートで $O(K^2 \log K)$

1 つ目の解法として、以下のような方法が考えられます。

- まず、最初の 2 種類のケーキ（'1', '2' の形のキャンドルが付いたもの）だけを考えて、XY 通りの中で上位 K 通りを順に求める。そこで、$A_i + B_j$ でソートし、その中の k 番目 $(1 \leq k \leq K)$ を E_k とする。
- 次に、最後のケーキ（'3' の形のキャンドルが付いたもの）を考える。最終的な美味しさの合計が上位 K 番目までに入るようなケーキの選び方は、最初の 2 種類のケーキだけを考えても上位 K 番目までに入らなければならないので、$E_i + C_j$ $(1 \leq \mathrm{i}, \mathrm{j} \leq \mathrm{K})$ でソートして、$1, 2, 3, \ldots, \mathrm{K}$ 番目の値が答えである。

例えば、$A = \{1, 2, 3\}, B = \{1, 5, 7\}, C = \{1, 9, 3\}, K = 5$ のとき、

- $A_i + B_j$ でソートした時の上位 5 番目までの値（$E_1 \sim E_5$）は $\{10, 9, 8, 8, 7\}$ である
  - 上位 9 番目までを見ると、$\{10, 9, 8, 8, 7, 6, 4, 3, 2\}$
- 次に、$E_i + C_j$ でソートした時の上位 5 番目までの値は、$\{19, 18, 17, 17, 16\}$ である。
  - 上位 15 番目までを見ると、$\{19, 18, 17, 17, 16, 13, 12, 11, 11, 11, 10, 10, 9, 9, 8\}$
- よって、XYZ 個の中の上位 5 個は $\{19, 18, 17, 17, 16\}$ である。

この解法を用いると、高々 $O(XY + KZ)$ 個をソートする必要がありますが、$X, Y, Z \leq 1000, K \leq 3000$ より実行時間制限に間に合います。

解法 #2 –「ある条件」を満たす最大 106307 通りを全部ソート

2 つ目の解法として、以下のような定理を使います。

【定理】
以下のような選び方は、上位 K 個には絶対入っていない。

- 高橋君が選んだ‘1’の形のキャンドルが付いたケーキが、それらのケーキの中で a 番目に美味しいとする。$(1 \leq a \leq X)$
- 高橋君が選んだ‘2’の形のキャンドルが付いたケーキが、それらのケーキの中で b 番目に美味しいとする。$(1 \leq b \leq Y)$
- 高橋君が選んだ‘3’の形のキャンドルが付いたケーキが、それらのケーキの中で c 番目に美味しいとする。$(1 \leq c \leq Z)$
- そのとき、$abc > K$ の場合、絶対に上位 K 個には入っていない。具体的には、一番順位が高い場合でも上から abc 番目である。

実際に、上の定理の条件を満たさない（つまり、$a \times b \times c \leq K$ である）ようなケーキの選び方は、$X, Y, Z = 1000, K = 3000$ の場合 106,307 通りあります。そうすると、条件を満たす 106,307 通り全てをソートしても、計算量的には全く問題がありません。

また、一般の K に対して、$a \times b \times c \leq K$ であるケーキの選び方は高々 $O(K \log^2 K)$ 通りです。つまり、この解法の計算量は $O(K \log^3 K)$ です。

以下のようなコードを書くと正解します。

```cpp
#include <bits/stdc++.h>
using namespace std;

long long X, Y, Z, K, A[1009], B[1009], C[1009];
vector<long long>vec;

int main() {
	cin >> X >> Y >> Z >> K;
	for (int i = 0; i < X; i++) cin >> A[i]; sort(A, A + X, greater<long long>());
	for (int i = 0; i < Y; i++) cin >> B[i]; sort(B, B + Y, greater<long long>());
	for (int i = 0; i < Z; i++) cin >> C[i]; sort(C, C + Z, greater<long long>());

	for (int i = 0; i < X; i++) {
		for (int j = 0; j < Y; j++) {
			for (int k = 0; k < Z; k++) {
				if ((i + 1) * (j + 1) * (k + 1) <= K) vec.push_back(A[i] + B[j] + C[k]);
				else break;
			}
		}
	}
	sort(vec.begin(), vec.end(), greater<long long>());
	for (int i = 0; i < K; i++) cout << vec[i] << endl;
	return 0;
}
```

解法 #3 – さらに高速な解法、priority-queue を用いて $O(K \log K)$

この問題は、以下のようなアルゴリズムによって $O(K \log K)$ で解くことができます。

- 数列 A, B, C (各形のキャンドルが付いたケーキ) を降順ソートする。そして、A_i が i 番目に美味しい '1' の形のキャンドルが付いたケーキとなるようにする。B_j, C_k に関しても同様とする。
- Priority Queue (優先度付きキュー) である Q を定義する。
- Q に $(A_1 + B_1 + C_1, 1, 1, 1)$ を追加する。なお、$A_1 + B_1 + C_1$ が 1 番目の美味しさの合計であることが保証されている。
- 以下の操作を K 回繰り返す。
  - Q の中で一番大きい要素を (p, i, j, k) とする。この組が t 番目の美味しさの合計である。ただし、ここで行っている操作が t 回目の操作であるとする。
  - Q の中で一番大きい要素を削除する。
  - $(A_{i+1} + B_j + C_k, i + 1, j, k)$ がまだ追加されたことが無ければ Q に追加する。
  - $(A_i + B_{j+1} + C_k, i, j + 1, k)$ がまだ追加されたことが無ければ Q に追加する。
  - $(A_i + B_j + C_{k+1}, i, j, k + 1)$ がまだ追加されたことが無ければ Q に追加する。

実際に、このアルゴリズムは正しく動作します。

理由は、$f(a, b, c)$ の値を「$A_a + B_b + C_c$ という選び方の美味しさの合計が上から何番目か、という値」とするとき、$f(a, b, c) < f(a + 1, b, c), f(a, b, c) < f(a, b + 1, c), f(a, b, c) < f(a, b, c + 1)$ が成り立つからです。

Priority Queue の挿入・削除・一番大きい値を求めるといった操作は $O(\log K)$ で動作するので、この問題を $O(K \log K)$ と、前 2 つの解法より大幅に高速に解くことができます。

なお、Priority Queue (優先度付きキュー) を知らない人は参考文献[2] を参考にしてください。

解法 #4 – 二分探索で $O(K \log max)$

まず、「上から K 番目の美味しさにランクインするボーダーの値」を求めることを考えます。

前提として、ここでは数列 A, B, C がソートされているものとします。

ボーダーの値は、二分探索によって求めることができます。

さて、「美味しさの合計が P 以上である個数が K 個以上あるかどうかを高速判定する方法」はどのように求めればよいのでしょうか。これは、単純な枝刈り全探索で $O(K^2)$ あるいは $O(K + X + Y + Z)$ で解くことができます。

以下は、枝刈り全探索の疑似コードです。

```
bool solve(int P) {
    int cnt = 0;
    for (int i = 1; i <= X; i++) {
        for (int j = 1; j <= Y; j++) {
            for (int k = 1; k <= Z; k++) {
                if (A[i] + B[j] + C[k] < P) break;
                cnt++;
                if (cnt >= K) return true;
            }
        }
    }
    return false;
}
```

上の太字部分が求まれば、P を二分探索するだけで、K 位以内にランクインするためのボーダーの美味しさの合計値が求まります。

あとは、ボーダーの値を $Border$ とするとき、以下の方法で上位 K 個を求めることができます。

- $Border + 1$ 以上であるものを全部列挙する。なお、列挙する方法は上の疑似コードにおいて、$Border + 1$ 以上であるものすべてについて $A_i + B_j + C_k$ を記録すればよい。
- 列挙した個数が K 個に満たなければ残りは全部美味しさの合計が $Border$ である。
- これらを降順ソートしたものが、上位 K 番目まで、つまり答えである。

計算量は $O(K^2 \log max)$ です。実装を工夫すると、$O((K + X + Y + Z) \log max)$ 程度で解くことができます。

ソースコード（C++, 解法 2）: https://atcoder.jp/contests/abc123/submissions/4841936

参考文献

[1] https://qiita.com/drken/items/18b3b3db5735241465ef の 3-2. 節

[2] https://ja.wikipedia.org/wiki/優先度付きキュー