4.2 節末問題 4.2 の解答

問題 4.2.1

この問題は、各経由地間の距離を単純なやり方で計算量 $O(N)$ かけて求めると、全体の計算量が $O(NM)$ となり、本問題の実行時間制限には間に合いませんが、累積和（→**4.2.1項**）のアイデアを使うとアルゴリズムを改善することができます。

まず、$X < Y$ の場合は以下の性質が成り立ちます。

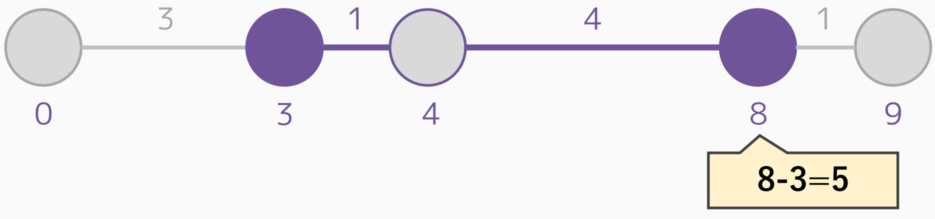
（駅 X から駅 Y までの距離）

＝（駅 1 から駅 Y までの距離 S_Y）－（駅 1 から駅 X までの距離 S_X）

下図は、$N = 5, (A_1, A_2, A_3, A_4) = (3, 1, 4, 1)$ の場合の具体例を示しています。駅 2 から駅 4 までの距離は $1 + 4 = 5$ と計算できますが、その代わりに

- 駅 1 から駅 4 までの距離： 8
- 駅 1 から駅 2 までの距離： 3

であることを使って、$8 - 3 = 5$ と求めることも可能です。$X > Y$ の場合は、X と Y を逆にして考えれば良いです。



そこで、駅 1 から駅 i までの距離は $S_i = A_1 + \cdots + A_{i-1}$ であるため、列 $[A_1, A_2, \ldots, A_N]$ に累積和をとると列 $[S_1, S_2, \ldots, S_N]$ が得られます。したがって、以下のように実装すると、正しい答えが求められます。計算量は $O(N + M)$ です。

```
#include <iostream>
using namespace std;

int N;
int A[200009], B[200009]; // 駅間距離、累積和
```

```
int main() {
    // 入力
    cin >> N;
    for (int i = 1; i <= N - 1; i++) cin >> A[i];
    cin >> M;
    for (int i = 1; i <= M; i++) cin >> B[i];

    // 累積和をとる
    S[1] = 0;
    for (int i = 2; i <= N; i++) S[i] = S[i - 1] + A[i - 1];

    // 答えを求める
    long long Answer = 0;
    for (int i = 1; i <= M - 1; i++) {
        if (B[i] < B[i + 1]) {
            Answer += (S[B[i + 1]] - S[B[i]]);
        }
        else {
            Answer += (S[B[i]] - S[B[i + 1]]);
        }
    }

    // 出力
    cout << Answer << endl;
    return 0;
}
```

※ Python などのソースコードは chap4-2.md をご覧ください。

問題 4.2.2

この問題は、以下のような手順で解くことができます。

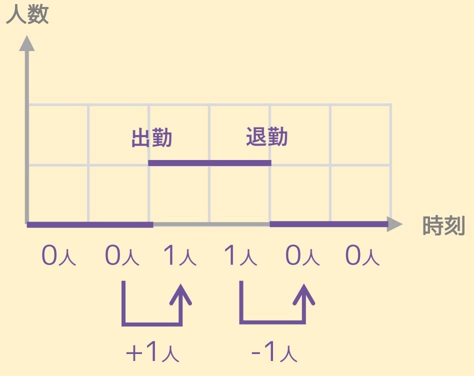
- 時刻 $t - 0.5$ と時刻 $t + 0.5$ の従業員数の差 B_t を計算する。
- 列 $[B_1, B_2, \ldots, B_T]$ に累積和をとると、列 $[A_1, A_2, \ldots, A_T]$ となる。

ここで、差 B_i については次のようにして計算すれば良いです。

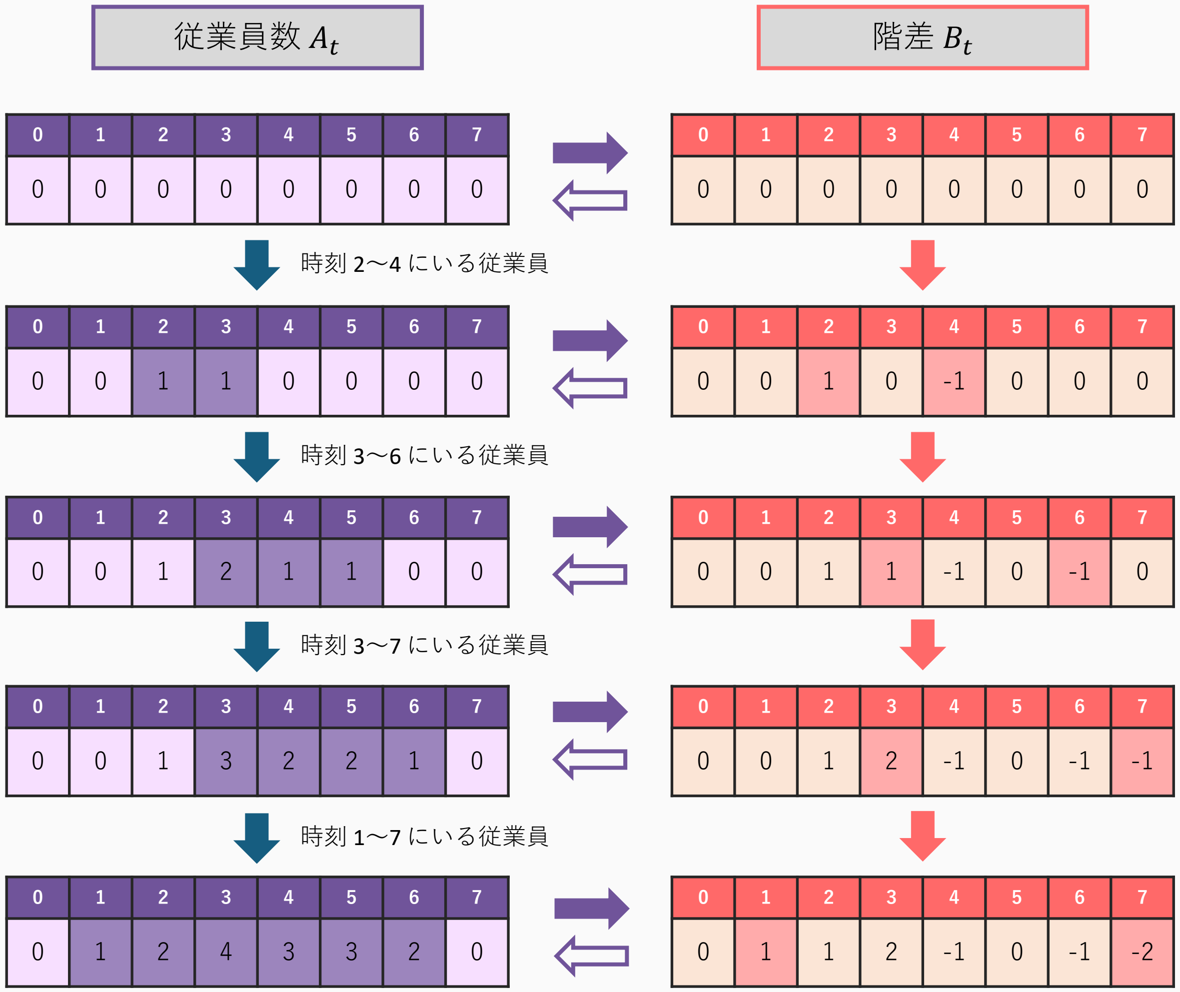
時刻 L に出勤し、時刻 R に退勤する従業員については、以下の操作を行う。

- B_L に 1 を足す。
- B_{R+1} に 1 を引く。

なお、操作を行う前は $B_i = 0$ に初期化しておく。



たとえば、$T=8, (L_i, R_i) = (2,4), (3,6), (3,7), (1,7)$ の場合における、差 B_t と従業員数 A_t の変化は以下のようになります。



したがって、各 i $(1 \leq i \leq N)$ について $B[L_i]$ に $+1$ して $B[R_i]$ に -1 した後、配列 B の累積和を出力する以下のようなプログラムを書くと、正解が得られます。計算量は $O(N+T)$ です。

```
#include <iostream>
using namespace std;

int N, T;
int L[500009], R[500009];
int A[500009], B[500009];

int main() {
    // 入力
    cin >> T >> N;
    for (int i = 1; i <= N; i++) cin >> L[i] >> R[i];

    // 階差 B[i] を計算する
    for (int i = 0; i <= T; i++) B[i] = 0;
```

```
    for (int i = 1; i <= N; i++) {
        B[L[i]] += 1;
        B[R[i]] -= 1;
    }

    // 累積和 A[i] を計算する
    A[0] = B[0];
    for (int i = 1; i <= T; i++) {
        A[i] = A[i - 1] + B[i];
    }

    // 答えを出力する
    for (int i = 0; i < T; i++) cout << A[i] << endl;
    return 0;
}
```

※ Python などのソースコードは chap4-2.md をご覧ください。

問題 4.2.3

まず、以下の 2 つのことを証明できれば、$f(x) = ax^2 + bx + c$ の形で表される時に限り `true` を返すことが分かります。

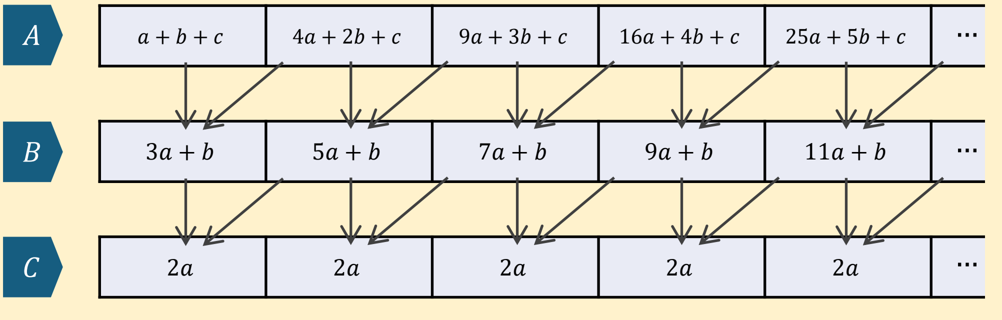
1. $f(x) = ax^2 + bx + c$ の形であれば `true` を返す
2. `true` を返した場合は $f(x) = ax^2 + bx + c$ の形で表される

それぞれについて、証明していきましょう。

1. の証明

$f(x) = ax^2 + bx + c$ のとき、$A[1] = a + b + c$、$A[2] = 4a + 2b + c$、$A[3] = 9a + 3b + c$、$\cdots$ と続きます。

そこで $B[1] = A[2] - A[1]$ なので、$B[1] = 3a + b$ となります。それ以降についても計算していくと、以下の表の通りになります。

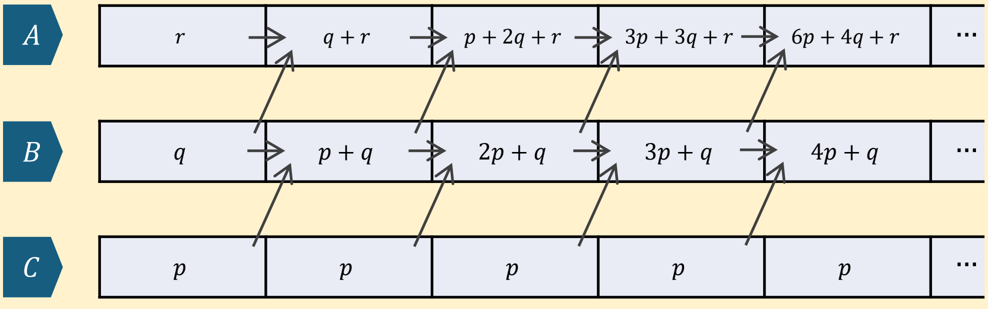


$C = [2a, 2a, \ldots, 2a]$ になっているため、`true` を返します。

2. の証明

関数 `func` が `true` を返す、すなわち $C[1] = C[2] = \cdots = C[N] = p$ となる場合を考えましょう。

階差は累積和の逆ですので、B は C の累積和、A は B の累積和となります。したがって、$A[1] = r, B[1] = q$ とすると、たとえば $B[2] = p + q$ となります。それ以降についても計算していくと、以下の表の通りになります。



そこで、$\big[A[1], A[2], \ldots, A[N]\big] = [r, q + r, p + 2q + r, 3p + 3q + r, \ldots]$ のとき、

$$f(x) = A[x] = \left(\frac{1}{2}x^2 - \frac{3}{2}x + 1\right)p + (x - 1)q + r$$

と表すことができます。これは二次関数（$f(x) = ax^2 + bx + c$ の形）ですので、2. が証明できました。

証明は以上です。なお、$f(x)$ が K 次関数のとき、1 回階差をとると $K - 1$ 次関数になることが知られているため、K 回階差をとると全ての要素が同じになります。（本問題は $K = 2$ の場合です）