第 6 章

考察テクニック

6.1 問題 B36：Switching Lights (難易度：★2相当)

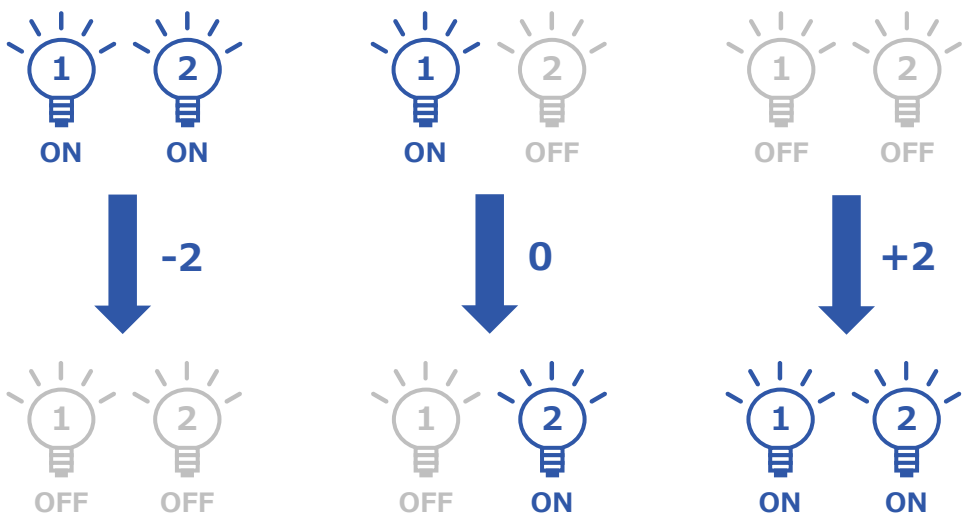
早速ですが結論から書きます。最初に ON になっている豆電球の個数を numON とするとき、答えは以下のようになります。

- numON の偶奇と K の偶奇が一致するとき：Yes
- numON の偶奇と K の偶奇が一致しないとき：No

したがって、次ページの解答例のようなプログラムにより、計算量 $O(N)$ で正しい答えを出すことができます。

解法の証明

それでは、なぜ偶奇が一致しなければ答えが No になるのでしょうか。この理由は、「2 つの豆電球の状態を反転させる」という操作を行っても、**ON になっている豆電球の個数の偶奇は変わらない**（個数の変化は -2/0/+2 のいずれかである）ことから分かります。

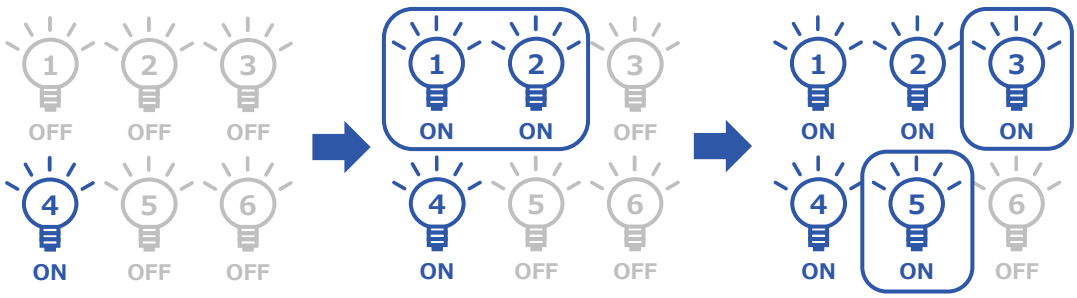


（次ページへ続く）

逆に、偶奇が一致している場合は、以下のような操作を行うことで K 個の豆電球を ON にできるので、答えは Yes です。

numON<K の場合	**OFF になっている豆電球を 2 つ選び、反転させる という操作を (K-numON)/2 回行う**
numON>K の場合	**ON になっている豆電球を 2 つ選び、反転させる という操作を (numON-K)/2 回行う**

たとえば $A=(0,0,0,1,0,0), K=5$ の場合、操作例は次のようになります。2 回の操作で、ON の豆電球が 5 個になっています。



解答例（C++）

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

int N, K;
string S;

int main() {
    // 入力
    cin >> N >> K;
    cin >> S;

    // ON となっているものの個数を数える
    int numON = 0;
    for (int i = 0; i < N; i++) {
        if (S[i] == '1') numON += 1;
    }

    // 答えを出力
    if (numON % 2 == K % 2) cout << "Yes" << endl;
    else cout << "No" << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

6.2 問題 B37：Sum of Digits (難易度：★5相当)

※この問題は例題と比べて相当難しいです。

この問題を解く最も単純な方法は、「1 の各桁の和は？」「2 の各桁の和は？」といったように 1 つずつ調べていくことです。もちろんこの方法でも正しい答えを出すことができますが、計算量は $O(N)$ と遅く※1、$N < 10^{15}$ の制約では実行時間制限に間に合いません。

そこで、**「i 桁目の数字が j であるような N 以下の整数」の個数 $R_{i,j}$** をすべての組 (i,j) に対して計算すると、高速に答えを求めることができます。

なぜなら、「i 桁目の数字が j のとき」が答えに与える寄与分が $j \times R_{i,j}$ となり、それらの値を合計すると答えになるからです。しかし、よく分からない方もいると思うので、具体例から入ります。

具体例を考えよう

まずは具体例として、$N = 23$ の場合を考えてみましょう。各桁の数字が 0～9 であるような 0 以上 23 以下※2 の整数の個数は次表の通りです（この程度であれば、手で計算できるでしょう）。

	0	1	2	3	4	5	6	7	8	9
1 の位	3	3	3	3	2	2	2	2	2	2
10 の位	10	10	4	0	0	0	0	0	0	0

そのため、各桁の数字が 0～9 のときの「答えへの寄与分」は以下の表のようになります。たとえば 10 の位が 2 のときの寄与分は、[20, 21, 22, 23] の 2 の部分が 4 回足されるので、寄与分は 2×4=8 です。

	0	1	2	3	4	5	6	7	8	9
1 の位	0	3	6	9	8	10	12	14	16	18
10 の位	0	10	8	0	0	0	0	0	0	0

したがって、$N = 23$ のときの答えは、表の値を合計した 114 となります。

※1　実装によっては、計算量が $O(N \log N)$ となります。

※2　1 以上より 0 以上の方がプログラムの実装が楽なので、0 以上の個数を考えています。なお、存在しない位は 0 であるものとして考えます（例：7 の 10 の位は 0）。

一般のケースで $R_{i,j}$ を求める（1）

さて、この問題では $R_{i,j}$ の値を高速に求める部分が難所になりますが、少しずつ考えていきましょう。

まずは 1 の位の個数 $R_{0,j}\ (0 \leq j \leq 9)$ についてはどうでしょうか。手始めに $N = 723$ のケースを考えると、次のようになります。

以下の理由で $R_{0,0} \sim R_{0,3}$ が 73、$R_{0,4} \sim R_{0,9}$ が 72 となる：

- 0〜719 の範囲では、1 の位に 0〜9 が 72 回ずつ出現
- 720〜723 の範囲では、1 の位に 0〜3 が 1 回ずつ出現

一般の N の場合も同じように考えると、次表のようになります（**0 以上 $[N \div 10] \times 10$ 未満**、**$[N \div 10] \times 10$ 以上 N 以下**の 2 つに分割して考えています）。

条件	$R_{i,j}$ の値
$j \leq$ (1 の位の値) の場合	$[N \div 10] + 1$ 個※3
$j >$ (1 の位の値) の場合	$[N \div 10]$ 個

次に 10 の位の個数 $R_{1,j}\ (0 \leq j \leq 9)$ についてはどうでしょうか。手始めに $N = 723$ のケースを考えると、次のようになります。

以下の理由で $R_{0,0} = R_{0,1} = 80$、$R_{0,2} = 74$、$R_{0,3} \sim R_{0,9}$ が 70 となる：

- 0〜699 の範囲では、10 の位に 0〜9 が 70 回ずつ出現
- 700〜719 の範囲では、10 の位に 0〜1 が 10 回ずつ出現
- 720〜723 の範囲では、10 の位に 2 が 4 回出現

一般の N の場合も同じように考えると、次表のようになります（**0 以上 $[N \div 100] \times 100$ 未満**、**$[N \div 100] \times 100$ 以上 $[N \div 10] \times 10$ 未満**、**$[N \div 10] \times 10$ 以上 N 以下**の 3 つに分割して考えています）。

条件	$R_{i,j}$ の値
$j <$ (10 の位の値) の場合	$[N \div 100] \times 10 + 10$ 個
$j =$ (10 の位の値) の場合	$[N \div 100] \times 10 + (N \bmod 10 + 1)$ 個※4
$j >$ (10 の位の値) の場合	$[N \div 100] \times 10$ 個

※3　$[x]$ は「x 以下の小数点以下切り捨て」です。
※4　$a \bmod b$ は「a を b で割ったときの余り」です。

一般のケースで $R_{i,j}$ を求める（2）

最後に、100 の位以上についても同様に考えることができます。整数の組 (i, j) に対する $R_{i,j}$ の値は以下の通りです。

条件	$R_{i,j}$ の値
$j <$ (10^i の位の値) の場合	$\left\lfloor N \div 10^{i+1} \right\rfloor \times 10^i + 10^i$ 個
$j =$ (10^i の位の値) の場合	$\left\lfloor N \div 10^{i+1} \right\rfloor \times 10^i + (N \bmod 10^i + 1)$ 個※4
$j >$ (10^i の位の値) の場合	$\left\lfloor N \div 10^{i+1} \right\rfloor \times 10^i$ 個

したがって、この問題を解くプログラムは以下のようにして実装できます。本問題の制約は $N < 10^{15}$ ですので、少なくとも $i = 14$ までの範囲で $R_{i,j}$ を計算しなければならないことに注意してください。

解答例（C++）

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

long long N;
long long R[18][10]; // R[i][j] は「下から i 桁目が j となるような N 以下の整数の個数」
long long Power10[18];

int main() {
    // 入力
    cin >> N;

    // 10 の N 乗を求める
    Power10[0] = 1;
    for (int i = 1; i <= 16; i++) Power10[i] = 10LL * Power10[i - 1];

    // R[i][j] の値を計算
    for (int i = 0; i <= 14; i++) {
        // 下から i 桁目の数字を求める
        long long Digit = (N / Power10[i]) % 10LL;

        // R[i][j] の値を求める
        for (int j = 0; j < 10; j++) {
                if (j < Digit) {
                        R[i][j] = (N/Power10[i+1])*Power10[i] + Power10[i];
                }
                if (j == Digit) {
                        R[i][j] = (N/Power10[i+1])*Power10[i] + (N%Power10[i]) + 1LL;
                }
                if (j > Digit) {
                        R[i][j] = (N/Power10[i+1])*Power10[i];
                }
```

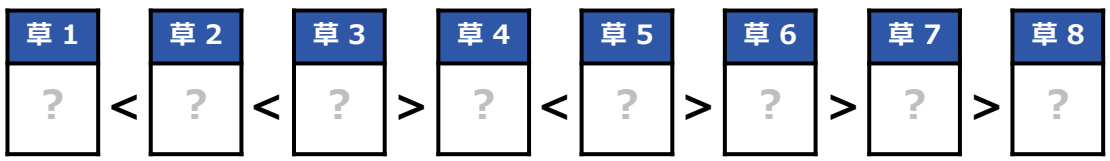
```
        }
    }

    // 答えを求める
    long long Answer = 0;
    for (int i = 0; i <= 15; i++) {
        for (int j = 0; j < 10; j++) Answer += 1LL * j * R[i][j];
    }

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

※Python のコードはサポートページをご覧ください

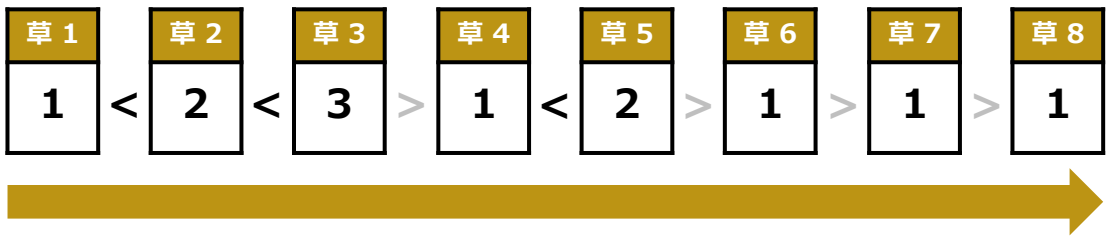
6.3 問題 B38：Height of Grass (難易度：★4相当)

この問題を解くための重要なポイントは、**「草の高さは絶対に〇〇以上である」という下限値を考えること**です。

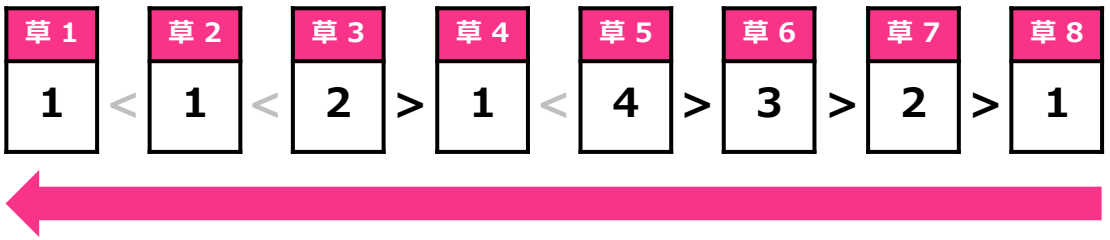
まずは手始めに、$N = 8, S = \text{“}AABABBB\text{“}$ というケースに対して下限値を求めることを考えましょう（イメージ図を以下に示します）。



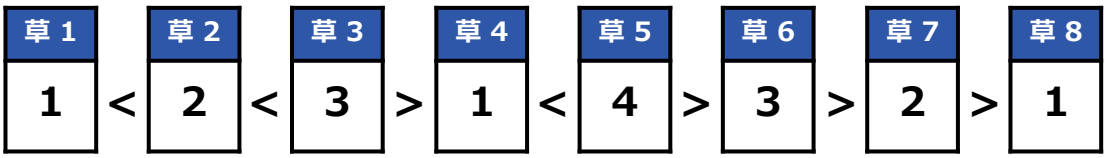
第一に、条件「<」から生まれる下限値を計算すると、以下のようになります。たとえば草 2 は草 1 より高くなければならないので、絶対に 2cm 以上であると分かります。また、草 3 は草 2 より高くなければならないので、絶対に 3cm 以上であると分かります。



第二に、条件「>」から生まれる下限値を計算すると、以下のようになります。たとえば草 7 は草 8 より高くなければならないので、絶対に 2cm 以上であると分かります。また、草 6 は草 7 より高くなければならないので、絶対に 3cm 以上であると分かります。



ここまでの 2 つをまとめると、草の高さは下図の値以上であることが分かります（2 つの上限値の max をとっています）。



それでは、草の高さが [1, 2, 3, 1, 4, 3, 2, 1] の場合は条件を満たすのでしょうか。答えは Yes です。そのため、「草の高さの合計としてあり得る最小値」は、1+2+3+1+4+3+2+1=17 であると分かります。

一般のケースを考える

一般のケースでも同様に下限値を求めてみましょう。

- 「<」の条件を考えたときの草 `i` の高さの下限値を `ret1[i]`
- 「>」の条件を考えたときの草 `i` の高さの下限値を `ret2[i]`

とするとき、それぞれの値は以下のようにして計算することができます。

```
// 「<」を考えたときの下限値を求める
int streak1 = 1; ret1[0] = 1; // streak1 は「何個 A が連続したか」+ 1
for (int i = 0; i < N - 1; i++) {
    if (S[i] == 'A') streak1 += 1;
    if (S[i] == 'B') streak1 = 1;
    ret1[i + 1] = streak1;
}

// 「>」を考えたときの下限値を求める
int streak2 = 1; ret2[N - 1] = 1; // streak2 は「何個 B が連続したか」+ 1
for (int i = N - 2; i >= 0; i--) {
    if (S[i] == 'B') streak2 += 1;
    if (S[i] == 'A') streak2 = 1;
    ret2[i] = streak2;
}
```

それでは、草 `i` の高さが `max(ret1[i], ret2[i])` のとき、すべての条件を満たすのでしょうか。証明は省略しますが※5、答えは Yes です。そのため、次ページの解答例のような実装により、正解を出すことができます。

※5　「<」と「>」の境界で分けて考えることが、証明のためのヒントになります。

解答例（C++）

```cpp
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int N, ret1[1 << 18], ret2[1 << 18];
string S;

int main() {
    // 入力
    cin >> N >> S;

    // 答えを求める
    int streak1 = 1; ret1[0] = 1; // streak1 は「何個 A が連続したか」+ 1
    for (int i = 0; i < N - 1; i++) {
        if (S[i] == 'A') streak1 += 1;
        if (S[i] == 'B') streak1 = 1;
        ret1[i + 1] = streak1;
    }
    int streak2 = 1; ret2[N - 1] = 1; // streak2 は「何個 B が連続したか」+ 1
    for (int i = N - 2; i >= 0; i--) {
        if (S[i] == 'B') streak2 += 1;
        if (S[i] == 'A') streak2 = 1;
        ret2[i] = streak2;
    }

    // 出力
    long long Answer = 0;
    for (int i = 0; i < N; i++) Answer += max(ret1[i], ret2[i]);
    cout << Answer << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

6.4 問題 B39：Taro’ s Job (難易度：★3相当)

実は次のような貪欲法を使うことで、最も多くの金額を稼ぐことができます。

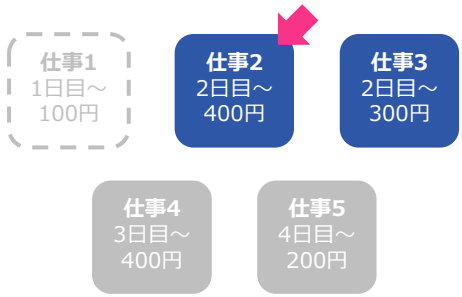
1 日目から順番に考えていく。それぞれの日について「今選べる仕事の中で最も給料の高いもの」を選ぶ。

たとえば $N = 5, D = 4, (X_i, Y_i) = (1, 100), (2, 400), (2, 300), (3, 400), (4, 200)$ の場合、下図のようにして最大値 1200 円を稼ぐことができます。

1

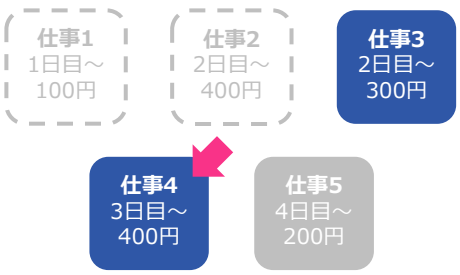
1 日目に選べるのは仕事 1
→ 仕事 1 を選ぶ（+100円）

2



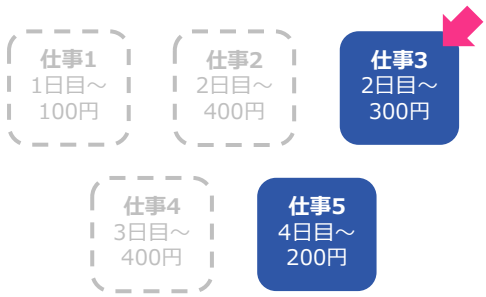
2 日目に選べるのは仕事 2, 3
→ 最も高い仕事 2 を選ぶ（+400円）

3



3 日目に選べるのは仕事 3, 4
→ 最も高い仕事 4 を選ぶ（+400円）

4



4 日目に選べるのは仕事 3, 5
→ 最も高い仕事 3 を選ぶ（+300円）

なぜ貪欲法が上手くいく？

それでは、なぜ「一番給料が高いものを選ぶ」という非常に直感的な方法が最適になるのでしょうか。少し難しいですが、以下のようにして証明することができます。

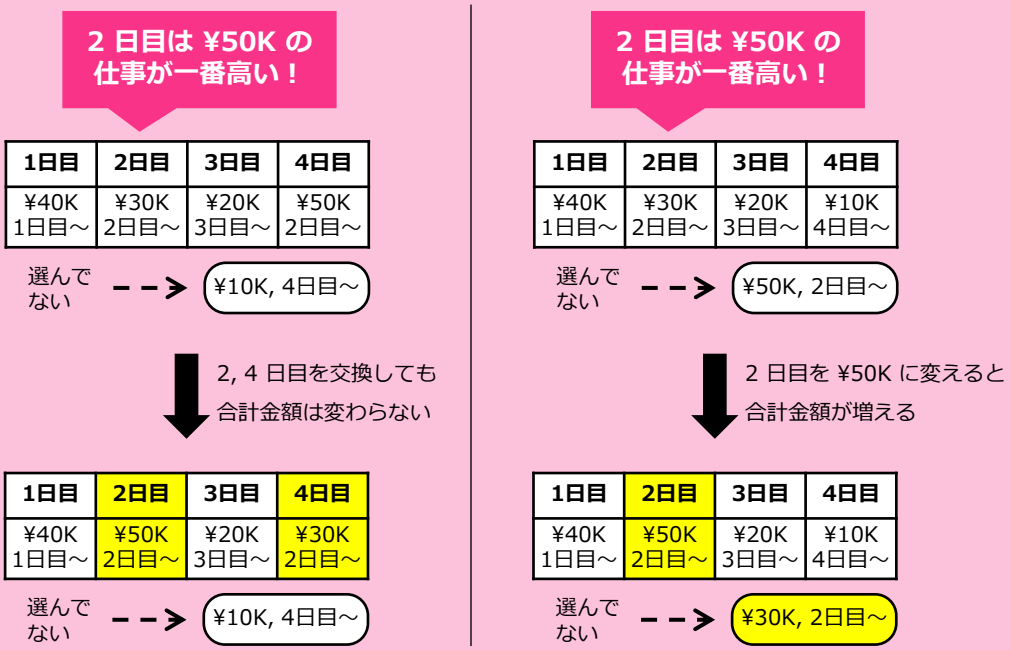
まず、「一番給料が高い仕事を選ぶ」以外の選び方をした場合、一番給料が高い仕事を選ばなかった“最初の日”が存在するので、これを d 日目とする。

また、d 日目に選んだ仕事の番号を a とし、その日に選べる最も給料の高い仕事の番号を b とする（ここで $Y_b > Y_a$ が成り立つ）。

このとき、次の 2 つのことから、**もし d 日目に仕事 b を選んでも、絶対に「稼げる合計金額」は減らない**といえる。

- もし $d+1$ 日目以降に仕事 b を選んでいる場合、仕事 a, b のやる日を交換しても合計金額は変わらない。
- もし $d+1$ 日目以降に仕事 b を選んでいない場合、d 日目に仕事 a の代わりに仕事 b をやると合計金額が増える。

したがって、すべての日について「一番給料が高い仕事」を選択しても損することは絶対にない。



解答例（C++）

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

int N, D;
int X[2009], Y[2009];
bool used[2009]; // used[i] は仕事 i を選んだかどうか
int Answer = 0;

int main() {
    // 入力
    cin >> N >> D;
    for (int i = 1; i <= N; i++) cin >> X[i] >> Y[i];

    // 答えを求める
    for (int i = 1; i <= D; i++) {
        int maxValue = 0; // 給料の最大値
        int maxID = -1;   // 給料が最大となる仕事の番号
        for (int j = 1; j <= N; j++) {
                if (used[j] == true) continue;
                if (maxValue < Y[j] && X[j] <= i) {
                        maxValue = Y[j];
                        maxID = j;
                }
        }

        // 選べる仕事がある場合
        if (maxID != -1) {
                Answer += maxValue;
                used[maxID] = true;
        }
    }

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

※Python のコードはサポートページをご覧ください

6.5 問題 B40：Divide by 100 (難易度：★3相当)

まず、$A_x + A_y$ の値が 100 の倍数になる条件を漏れなく列挙すると、以下のようになります。最初の 2 つ以外は逆順の場合（$A_x = 99, A_y = 1$ など）も含まれることに注意してください。

- **(100 で割った余りが 0) + (100 で割った余りが 0) の形**
- **(100 で割った余りが 50) + (100 で割った余りが 50) の形**
- (100 で割った余りが 1) + (100 で割った余りが 99) の形
- (100 で割った余りが 2) + (100 で割った余りが 98) の形
- (100 で割った余りが 3) + (100 で割った余りが 97) の形
- (100 で割った余りが 4) + (100 で割った余りが 96) の形

  ：
- (100 で割った余りが 49) + (100 で割った余りが 51) の形

そこで、A_i を 100 で割った余りが p となるような i の個数を cnt[p] とするとき、各条件を満たす組 (x, y) の個数は次表の通りになります。

条件	組 (x, y) の個数
0+0	cnt[0]*(cnt[0]-1)/2
50+50	cnt[50]*(cnt[50]-1)/2
1+99	cnt[1]*cnt[99]
2+98	cnt[2]*cnt[98]
3+97	cnt[3]*cnt[97]
4+96	cnt[4]*cnt[96]
：	：
49+51	cnt[49]*cnt[51]

この部分はそれぞれ ${}_{cnt[0]}C_2$ 個、${}_{cnt[50]}C_2$ 個になっている

これらを合計した値が答えですので、次ページに示すような実装をすると正解です。計算回数は $N + 100$ 回程度と高速です。

解答例（C++）

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

long long N, A[200009];
long long cnt[109];
long long Answer = 0;

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

    // 個数を数える
    for (int i = 0; i < 100; i++) cnt[i] = 0;
    for (int i = 1; i <= N; i++) cnt[A[i] % 100] += 1;

    // 答えを求める
    for (int i = 1; i < 50; i++) Answer += cnt[i] * cnt[100 - i];
    Answer += cnt[0] * (cnt[0] - 1LL) / 2LL;
    Answer += cnt[50] * (cnt[50] - 1LL) / 2LL;

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

※Python のコードはサポートページをご覧ください

6.6 問題 B41：Reverse of Euclid (難易度：★3相当)

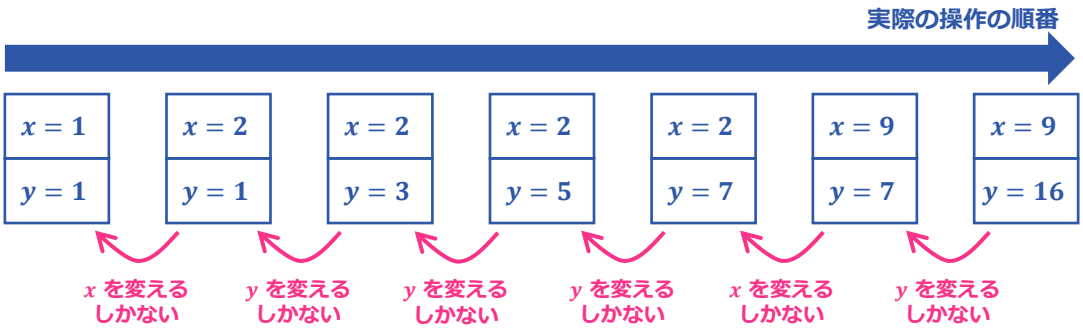
この問題を解く重要なポイントは、**最後の操作から順番に考えていく**ことです。まず、最後の一手は以下のようになります。

> $X < Y$ の場合、最後の一手として「x の値を $x+y$ に変更する」という操作を選べない。なぜなら y より x の方が大きくなるからである。そのため、最後の一手は残った**「y の値を $x+y$ に変更する」**となる。
>
> $X > Y$ の場合、最後の一手として「y の値を $x+y$ に変更する」という操作を選べない。なぜなら x より y の方が大きくなるからである。そのため、最後の一手は残った**「x の値を $x+y$ に変更する」**となる。

また、最後から 2 手目、3 手目、4 手目、… についても、大小関係によって「行える操作」が一通りに決まるので、最後の一手のときと同じような方針で操作の列を求めることができます。

具体例を考えよう

たとえば $(X,Y)=(9,16)$ の場合、最後の一手から順番に考えていくと以下のような操作の列が得られます。



補足

逆から考えたときの操作列は、ユークリッドの互除法とほぼ同じであるため、(X,Y) の最大公約数が 1 のときは $(x,y)=(1,1)$ で操作が終わります。

解答例（C++）

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int X, Y;

int main() {
    // 入力
    cin >> X >> Y;

    // 逆から考えていく
    vector<pair<int, int>> Answer;
    while (X >= 2 || Y >= 2) {
        Answer.push_back(make_pair(X, Y));
        if (X > Y) X -= Y;
        else Y -= X;
    }
    reverse(Answer.begin(), Answer.end());

    // 出力
    cout << Answer.size() << endl;
    for (int i = 0; i < Answer.size(); i++) {
        cout << Answer[i].first << " " << Answer[i].second << endl;
    }
    return 0;
}
```

※Python のコードはサポートページをご覧ください

6.7 問題 B42：Two Faced Cards (難易度：★5相当)

まず、(表の総和の絶対値)+(裏の総和の絶対値) を最大化する方法としては、以下の 4 つがあります。

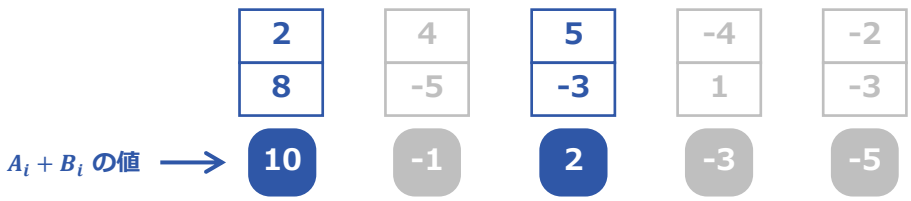
- (表の総和) + (裏の総和) を最大化する。
- (表の総和) - (裏の総和) を最大化する。
- - (表の総和) + (裏の総和) を最大化する。
- - (表の総和) - (裏の総和) を最大化する。

それぞれの場合について、どのようなカードの選び方を選べば最大になるかを考えてみましょう。

(表の総和)+(裏の総和) の場合

まず、(表の総和)+(裏の総和) の値は、**$A_i + B_i > 0$ であるカードをすべて選び、そうでないカードを全く選ばない**ときに最大になります。

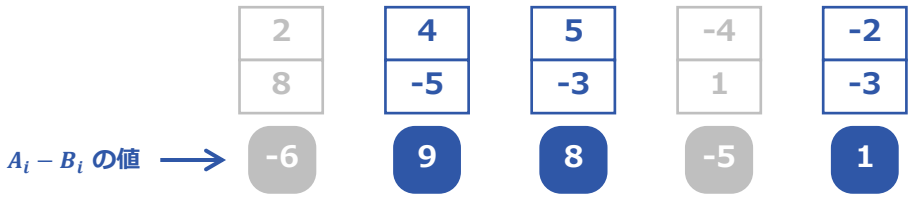
たとえば $(A_i, B_i) = (2, 8), (4, -5), (5, -3), (-4, 1), (-2, -3)$ であるときは、1 枚目と 3 枚目のカードを選べば良いです。 (表の総和)+(裏の総和) は 10+2=12 となります。



(表の総和)-(裏の総和) の場合

次に、(表の総和)-(裏の総和) の値は、**$A_i - B_i > 0$ であるカードをすべて選び、そうでないカードを全く選ばない**ときに最大になります。

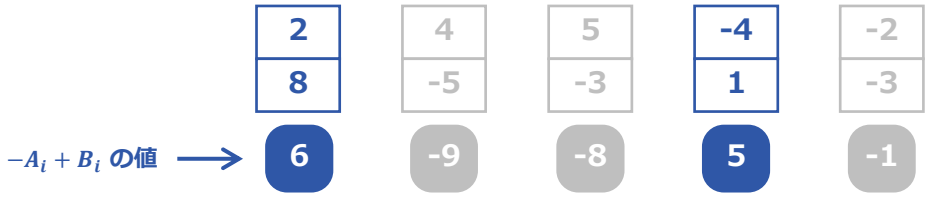
たとえば $(A_i, B_i) = (2,8), (4,-5), (5,-3), (-4,1), (-2,-3)$ であるときは、2・3・5 枚目のカードを選べば良いです。 (表の総和)-(裏の総和) は 9+8+1=18 となります。



-(表の総和)+(裏の総和) の場合

次に、-(表の総和)+(裏の総和) の値は、**$-A_i + B_i > 0$ であるカードをすべて選び、そうでないカードを全く選ばない**ときに最大になります。

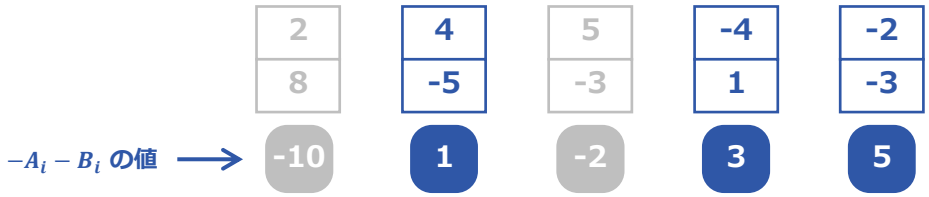
たとえば $(A_i, B_i) = (2,8), (4,-5), (5,-3), (-4,1), (-2,-3)$ であるときは、1 枚目と 4 枚目のカードを選べば良いです。 (表の総和)+(裏の総和) は 6+5=11 となります。



-(表の総和)-(裏の総和) の場合

次に、-(表の総和)-(裏の総和) の値は、**$-A_i - B_i > 0$ であるカードをすべて選び、そうでないカードを全く選ばない**ときに最大になります。

たとえば $(A_i, B_i) = (2,8), (4,-5), (5,-3), (-4,1), (-2,-3)$ であるときは、2・4・5 枚目のカードを選べば良いです。 (表の総和)+(裏の総和) は 1+3+5=9 となります。



結局求める答えは？

最後に、求めるべき答えは、4 つの方法で得られた合計値のうち最大の値となります。たとえば先程挙げた例では $\max(12, 18, 11, 9) = 18$ です。

このように、「表と裏の目標とする符号」を全探索することで、計算量 $O(N)$ で正しい答えを出すことができます。

解答例（C++）

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

long long N;
long long A[100009], B[100009];

// omote=1 のとき表の総和が正、ura=1 のとき裏の総和が正
// omote=2 のとき表の総和が負、ura=2 のとき裏の総和が負
long long solve(int omote, int ura) {
    long long sum = 0;
    for (int i = 1; i <= N; i++) {
        long long card1 = A[i]; if (omote == 2) card1 = -A[i];
        long long card2 = B[i]; if (ura == 2) card2 = -B[i];
        // カード i は選ぶべきか？
        if (card1 + card2 >= 0) {
                sum += (card1 + card2);
        }
    }
    return sum;
}

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

    // 表の総和の正負と、裏の総和の正負を全探索
    long long Answer1 = solve(1, 1);
    long long Answer2 = solve(1, 2);
    long long Answer3 = solve(2, 1);
    long long Answer4 = solve(2, 2);

    // 答えを出力
    cout << max({ Answer1,Answer2,Answer3,Answer4 }) << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

6.8 問題 B43：Quiz Contest (難易度：★2相当)

この問題を解くための重要なポイントは、**不正解数を数えること**です※6。生徒 i の不正解数を Incorrect[i] をするとき、この生徒の正解数は M-Incorrect[i] となるため、もし不正解数さえ分かれば正解数を計算することができます。

不正解数を数えるには？

それでは、不正解数を効率的に数えるにはどうすれば良いのでしょうか。各問題には 1 名しか不正解者がいないので、以下のようなプログラムにより計算量 $O(N+M)$ で全生徒の不正解数が分かります。

```
for (int i = 1; i <= N; i++) Incorrect[i] = 0;
for (int i = 1; i <= M; i++) Incorrect[A[i]] += 1;
```

最後に、入力や出力などを含めた全体のプログラムを以下に示します。

解答例（C++）

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

int N, M;
int A[200009];
int Incorrect[200009];

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

    // 不正解数を求める
    for (int i = 1; i <= N; i++) Incorrect[i] = 0;
    for (int i = 1; i <= M; i++) Incorrect[A[i]] += 1;

    // 答えを出力
    for (int i = 1; i <= N; i++) cout << M - Incorrect[i] << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

※6　なぜ「不正解数を数える」という発想になるのか：不正解は 1 名しかいないので、その方が数えるのが速いからです。テストで最下位に近い成績を取ったとき、下から数える方が速いのと同じです

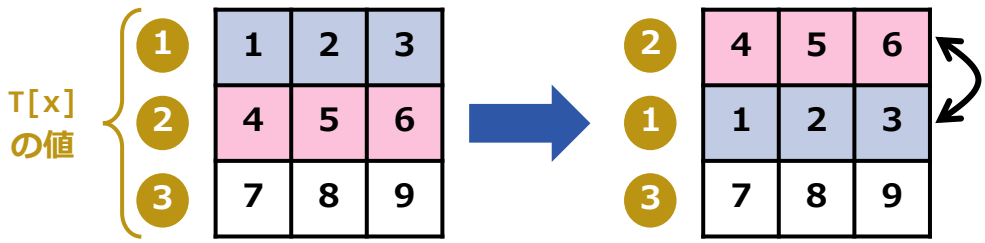
6.9 問題 B44：Grid Operations (難易度：★3相当)

まず考えられる方法は、交換操作を直接行うことです。1 回の操作で $2N$ 個のマスが変化するため、1 回当たりの計算量は $O(N)$ となり、全体の計算量は $O(NQ)$ です。C++ などの高速なプログラミング言語であれば、この解法でも十分間に合います。しかし、もう少し良い解法はあるのでしょうか。

工夫した解法

そこで、マス目の代わりに**現在の x 行目が元々の何行目であるか T[x]** を管理すると、より高速に解くことができます。まず x 行目と y 行目を入れ替える交換操作は、swap(T[x], T[y]) の一発で処理することができます。

また、上から x 行目・左から y 列目の値を答える取得操作では、A[T[x]][y] を出力すれば良いです（A は元々のマス目を表す二次元配列）。



この解法を実装すると以下の解答例のようになります。計算量は $O(N+Q)$ と非常に高速です。

解答例（C++）

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

int N, A[509][509];
int Q, QueryType[200009], x[200009], y[200009];
int T[509];

int main() {
    // 入力
    cin >> N;
```

※Python のコードはサポートページをご覧ください

```
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) cin >> A[i][j];
    }

    // 配列 T を初期化
    for (int i = 1; i <= N; i++) T[i] = i;

    // クエリの処理
    cin >> Q;
    for (int i = 1; i <= Q; i++) {
        cin >> QueryType[i] >> x[i] >> y[i];
        if (QueryType[i] == 1) {
                swap(T[x[i]], T[y[i]]);
        }
        if (QueryType[i] == 2) {
                cout << A[T[x[i]]][y[i]] << endl;
        }
    }
    return 0;
}
```

※Python のコードはサポートページをご覧ください

6.10 問題 B45：Blackboard 2 (難易度：★2相当)

まず、1 回の操作では **$a+b+c$ の値が変わりません。** なぜなら、操作によって 1 つの整数が +1 され、別の 1 つの整数が -1 されるため、合計して ±0 となるからです。そのため、$a+b+c$ の値が 0 でなければ、答えは絶対に No です。

◆ $a+b+c=0$ の場合

逆に、$a+b+c=0$ の場合は必ず Yes になります。これは次のようにして証明することができます。

$a+b+c=0$ の場合、「正の整数の絶対値の合計 s_+」と「負の整数の絶対値合計 s_-」は必ず一致する。たとえば $(a,b,c)=(13,7,-20)$ の場合、$s_+=s_-=20$ となり確かに一致する。

そのため、「正の整数を一つ選んで +1 し、負の整数を一つ選んで -1 する」という操作を s_+ 回行えば、必ず 3 つすべての整数が 0 になる。

◆ 解答例（C++）

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

long long a, b, c;

int main() {
    // 入力
    cin >> a >> b >> c;

    // 出力
    if (a + b + c == 0) cout << "Yes" << endl;
    else cout << "No" << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください