プログラミングコンテスト入門

AtCoder Beginners Selection を解いてみよう

tsutaj (@_TTJR_) ▸ Link

Hokkaido University M2

Apr 15, 2019

1. はじめに
2. 基本
   - 入出力の練習
3. AtCoder Beginners Selection を解いてみよう
   - Product
   - Placing Marbles
   - Shift Only
   - Coins
   - Some Sums
   - Card Game for Two
   - Kagami Mochi
   - Otoshidama
   - 白昼夢 / Daydream
   - Traveling
4. 参考になる書籍
5. さいごに

プログラミングコンテストとは？

プログラミングコンテスト (プロコン)

- 与えられた問題に対して、「実行時間が速く、確実に解答できる」プログラムを書いて正答することを競う競技
- プログラミングやアルゴリズムのスキルが身につく
- 就活などで行われるコーディング試験に有利になる (と思います)

順位	ユーザ	得点	A	B	C	D	E	F
1	ksun48 Massachusetts Institute of Technology	4700 (2) 76:48	400 32:23	700 4:24	800 (1) 17:32	1000 (1) 36:15	(2)	1800 66:48
2	ecnerwal	4700 (3) 102:20	400 14:37	700 (1) 24:33	800 (2) 38:33	1000 44:43	-	1800 87:20
3	apiad unknown	4700 108:23	400 108:23	700 104:49	800 104:57	1000 105:08	-	1800 105:16
4	Petr	4700 (4) 128:53	400 4:32	700 8:55	800 (4) 52:05	1000 14:51	-	1800 108:53
5	tourist ITMO University	4100 39:15	400 16:20	700 16:16	800 24:20	1000 16:26	1200 39:15	-

Figure: プロコン (AtCoder) の順位表。世界中から参加しています

プログラミングコンテストの問題を解く流れ

プロコンの問題は、大きく分けて次の 3 ステップを踏んで解きます

1. 問題について考察

出題される問題は、見てすぐに解法がわかるようなものばかりではありません。問題を「実行時間が速く、確実に」解くために必要な手続きを考察します。

2. 実装に必要な処理を考察

特に最初のうちは、解法はわかっても実装面で詰まることが多いことでしょう。自分がやりたい処理を実現するために必要なことを考察します。

3. 実装・ジャッジに提出

実装方針が立てば、パソコンの出番です。実際に書いてみましょう。書いたコードがテストケースに正答できるかどうかは、オンラインジャッジが判定してくれます。

本スライドの目的

- このスライドは、「初心者が、オンラインジャッジにプログラムを提出してプログラミングを練習する」ことの一助を目的としたスライドです
  - AtCoder Beginners Selection Link を実際に解きます！ このセットは、「AtCoder に登録したら次にやること ～これだけ解けば十分戦える！過去問精選 10 選～」(drken さん) の記事 Link で選出された問題を集めたものです。
  - プログラムの書き方については深くまでは掘り下げません
  - 必要であれば他の記事なども参考にしてください (スライドの最後に参考になるものをまとめています)
- まだ AtCoder のアカウントを持っていない人は、登録しよう！ Link
- スライド内のコードは全て C++ で書いています

はじめてのあっとこーだー

はじめてのあっとこーだー (Welcome to AtCoder)

整数 a, b, c と文字列 s が与えられるので、整数 $a + b + c$ と文字列 s を空白区切りで並べて表示し、出力の最後に改行しなさい。

入力例 1

```
1
2 3
test
```

出力例 1

```
6 test
```

- $1 + 2 + 3$ は 6 なので、上記のように出力します。

はじめてのあっとこーだー

これを実現するプログラムを書くには・・・以下の 3 つの処理が必要

1. 入力を受け取る
2. 足し算をする
3. 出力する

C++ ではどう書くか？

はじめてのあっとこーだー

これを実現するプログラムを書くには・・・以下の 3 つの処理が必要

1. 入力を受け取る
   - cin を使って入力を受け取れます
2. 足し算をする
   - 普通の四則演算のように '+' 記号で足し算ができます
3. 出力する
   - cout を使って出力できます
   - 空白は " " を、改行は endl を使えば可能です

C++ ではどう書くか？
文章でいわれてもピンとこないと思うので、実際にコードを見てみよう

はじめてのあっとこーだー

これと同じ提出は AtCoder 上でも確認できます ▸ Link

```cpp
#include <iostream>  // cin, cout を利用するために必要です！
#include <string>    // string 型の利用に必要です
using namespace std; // 説明は省きますが cin, cout を利用する際のタイプ数が減ります

int main() { // 主な処理は、main で囲まれた範囲内に書きます
    // ### 1: 入力を受け取る ###
    // int (Integer, 整数) 型の変数 a, b, c を宣言 (変数を使うときはまず宣言！)
    // 注意: カンマ (',') は後に処理が続くことを、
    //       セミコロン (';') は文の終わりを表す、いわば句読点のようなものです
    //       これらがないとエラーが出てしまいます！
    int a, b, c;

    // 入力を受け取る
    // 受け取るときは、cin >> (変数1) >> (変数2) >> ...; と書きます
    cin >> a;
    cin >> b >> c;

    // string (文字列) 型の変数 s を宣言
    string s;

    // 入力を受け取る (整数のときと同様に cin でできます)
    cin >> s;

    // ### 2: 足し算をする ###
    // a + b + c の結果を格納する変数 sum を宣言・代入
    int sum = a + b + c;

    // ### 3: 出力する ###
    // 出力するときは、cout << (変数や文字列1) << (変数や文字列2) << ...; と書きます
    // 注意 1: sum と s の間に空白が必要なので " " が挟まっています
    // 注意 2: "<<" の後に endl と続けると改行できます
    cout << sum << " " << s << endl;
    // プログラムが正常に終了したことを知らせる文です
    return 0;
}
```

Product

2 つの正整数 a, b が与えられるので、a と b の積が奇数であれば “Odd” と、偶数であれば “Even” と出力しなさい。

入力例 1 Copy

```
3 4
```

出力例 1 Copy

```
Even
```

$3 \times 4 = 12$ は偶数なので 'Even' を出力してください。

入力例 2 Copy

```
1 21
```

出力例 2 Copy

```
Odd
```

$1 \times 21 = 21$ は奇数なので 'Odd' を出力してください。

Product

これを実現するプログラムを書くには・・・以下の 3 つの処理が必要

1. 入出力 (これ以降省略します)
2. 掛け算をする
3. 条件分岐
   - 奇数なら “Odd”、偶数なら “Even” のように、整数によって処理を変える必要がある

C++ ではどう書くか？

Product

これを実現するプログラムを書くには・・・以下の 3 つの処理が必要

1. 入出力 (これ以降省略します)
2. 掛け算をする
   - 普通の四則演算のように '*' 記号で掛け算ができます
3. 条件分岐
   - 奇数なら “Odd”、偶数なら “Even” のように、整数によって処理を変える必要がある
   - if - else 文を使うと、このような条件分岐ができます
   - 偶奇の判定には 2 で割り切れるかどうかが必要ですが、これには剰余計算 (剰余記号 '%') が有効です

C++ ではどう書くか？
実際にコードを見てみよう

Product

実際の提出: ▸ Link

```cpp
#include <iostream> // cin, cout を利用
using namespace std;

int main() {
    // 変数 a, b を宣言して入力を受け取る
    int a, b; cin >> a >> b;
    // a と b の積を格納する変数 product を宣言・代入
    int product = a * b;

    // '%' は剰余計算。下の例の場合、product を 2 で割った余りが返る
    // 2 で割ったあまりが 0 と等しい（偶数）場合は、if の中に入る
    // 注意: 代入のときとは異なり、等しいかどうかの判定には "==" を使う！
    if(product % 2 == 0) {
        cout << "Even" << endl;
    }
    // 2 で割ったあまりが 0 と等しくない（奇数）場合は、else の中に入る
    else {
        cout << "Odd" << endl;
    }
    return 0;
}
```

Placing Marbles

'0' と '1' のみからなる、長さ 3 の文字列 s が与えられます。この文字列に '1' がいくつ含まれているか出力しなさい。

入力例 1 Copy

```
101
```

出力例 1 Copy

```
2
```

- マス 1, 3 にビー玉が置かれます

入力例 2 Copy

```
000
```

出力例 2 Copy

```
0
```

- ビー玉はどのマスにも置かれません

Placing Marbles

これを実現するプログラムを書くには・・・以下の 2 つの処理が必要

1. 文字列に含まれるそれぞれの文字を見る
2. 条件分岐
   - それぞれの文字について、それが '1' であれば答えに 1 を足す

C++ ではどう書くか？

Placing Marbles

これを実現するプログラムを書くには・・・以下の 2 つの処理が必要

1. 文字列に含まれるそれぞれの文字を見る
   - 特定の範囲で処理を繰り返す際には、for 文が有効です
   - 文字列 s の i 文字目は、$s[i]$ のように書くと見ることができます。数列の添字のようなイメージ
2. 条件分岐
   - それぞれの文字について、それが '1' であれば答えに 1 を足す
   - if - else 文を使うと、このような条件分岐ができます
   - 文字の一致判定は、先程の剰余と同様に "==" で結べば可能です

C++ ではどう書くか？
実際にコードを見てみよう

Placing Marbles

実際の提出: ▸ Link

```cpp
#include <iostream> // cin, cout
#include <string>   // string 型
using namespace std;

int main() {
    string s; cin >> s;

    // 答えを格納する変数 (初期値は 0 としておく)
    int ans = 0;

    // for 文: for(初期条件; 継続条件; 次のイテレーションにうつる前に処理する内容) と書きます
    // 下の例だと初期条件が i = 0 で、i < 3 を満たす限り処理を継続 (i == 3 になると処理が終わる)、
    // 次のイテレーションにうつる前に i を 1 増やす、ということを 1 つの文で表現しています

    // i という変数を [0, 3) の範囲で 1 ずつ増やしながら、for 内の処理を行う
    for(int i=0; i<3; i++) {
        // s[i] は 「s の i 文字目」!
        // 注意: 添字は 0 から始まります
        if(s[i] == '1') {
            // s の 1 文字目が '1' と等しければ、答えを増やす
            ans = ans + 1;
        }
    }

    cout << ans << endl;
    return 0;
}
```

Shift Only

N 個の正整数 $A_1, A_2, \ldots, A_N$ がある。これらの整数がすべて偶数であるときに限り、以下の操作を行うことができる。操作回数の最大値を求めよ。

1. それぞれの整数 A_i について、2 で割ったものに置き換える。

- $1 \leq N \leq 200$
- $1 \leq A_i \leq 10^9$

入力例 1 Copy

```
3
8 12 40
```

出力例 1 Copy

```
2
```

最初，黒板には $[8, 12, 40]$ が書かれています．このとき，書かれている整数はすべて偶数なので，操作を行うことができます．

1 回操作を行った後，黒板には $[4, 6, 20]$ が書かれています．再び，書かれている整数はすべて偶数なので，操作を行うことができます．

2 回操作を行った後，黒板には $[2, 3, 10]$ が書かれています．この時，奇数 3 が書かれているため，これ以上操作を行うことはできません．

よって，すぬけ君は最大で 2 回操作を行うことができます．

Shift Only

これを実現するプログラムを書くには・・・以下の 2 つの処理が必要

1. それぞれの整数をみて、すべて偶数であるかどうか調べる
2. 条件分岐
   - 奇数がひとつでもあれば処理を終了
   - すべて偶数であれば、各整数を 2 で割る

C++ ではどう書くか？

Shift Only

これを実現するプログラムを書くには・・・以下の 2 つの処理が必要

1. それぞれの整数をみて、すべて偶数であるかどうか調べる
   - 特定の範囲で処理を繰り返すので、これも for 文が有効
   - 数列は、プログラミングの世界では「配列」という構造を用いて表します。文字列の時と同様、配列 A の i 番目の値は $A[i]$ で参照できます
2. 条件分岐
   - 奇数がひとつでもあれば処理を終了
   - すべて偶数であれば、各整数を 2 で割る
   - if - else 文を使いましょう
   - すべて偶数の場合の処理に関しては、for 文を使うと効果的に書けるでしょう

C++ ではどう書くか？
実際にコードを見てみよう

Shift Only

実際の提出: ▸ Link

```cpp
#include <iostream>
#include <vector>   // 配列を利用する際に必要！
using namespace std;

int main() {
    int N, ans = 0; cin >> N;

    // 長さ N の int 型の（可変長）配列 A を宣言！
    vector<int> A(N);
    // A の i 番目の値を入力する（for 文が効果的）
    for(int i=0; i<N; i++) cin >> A[i];

    // while 文: 括弧内の条件が満たされている間は処理し続ける
    //           true を入れているので無限ループになる
    while(true) {
        // 奇数が存在するかどうかを管理する変数を宣言
        bool exist_odd_number = false;
        for(int i=0; i<N; i++) {
            // 奇数ならば変数を true に変更
            if(A[i] % 2 == 1) exist_odd_number = true;
        }

        if(exist_odd_number) {
            // break 文を書くとループを抜けられる
            break;
        }
        else {
            // 1 回操作することができる
            ans++;
            // 配列内のそれぞれの値について、2 で割った値を代入
            for(int i=0; i<N; i++) {
                A[i] = A[i] / 2;
            }
        }
    }
    cout << ans << endl;
    return 0;
}
```

Shift Only

ところで、数列内の整数値は 10^9 までとりうるが、この解法は本当に間に合うのか・・・？

- 実はちゃんと間に合う
- 操作を行うと数列の値が半分になり、これの繰り返し
  - 操作回数 $\approx \max_i \log_2 A_i$
- 最悪でも、全体でだいたい $N \times \max_i \log_2 A_i$ 回しか計算ステップがないため、これで OK

Shift Only

ところで、数列内の整数値は 10^9 までとりうるが、この解法は本当に間に合うのか・・・？

- 実はちゃんと間に合う
- 操作を行うと数列の値が半分になり、これの繰り返し
  - 操作回数 $\approx \max_i \log_2 A_i$
- 最悪でも、全体でだいたい $N \times \max_i \log_2 A_i$ 回しか計算ステップがないため、これで OK

Shift Only

ところで、数列内の整数値は 10^9 までとりうるが、この解法は本当に間に合うのか・・・？

- 実はちゃんと間に合う
- 操作を行うと数列の値が半分になり、これの繰り返し
  - 操作回数 $\approx \max_i \log_2 A_i$
- 最悪でも、全体でだいたい $N \times \max_i \log_2 A_i$ 回しか計算ステップがないため、これで OK

Shift Only

ところで、数列内の整数値は 10^9 までとりうるが、この解法は本当に間に合うのか・・・？

- 実はちゃんと間に合う
- 操作を行うと数列の値が半分になり、これの繰り返し
  - 操作回数 $\approx \max_i \log_2 A_i$
- 最悪でも、全体でだいたい $N \times \max_i \log_2 A_i$ 回しか計算ステップがないため、これで OK

計算ステップ数の評価

プログラムを書くときは、それが最悪でどれだけのステップ実行されるかを吟味しよう。もっと詳しく言うと、プログラムの計算量を考えよう。計算量については以下に示す記事が参考になると思います。

- 計算量オーダーの求め方を総整理！ (drken さん) Link
- 初心者向け プログラムの計算量を求める方法 (cotrpepe さん) Link

Coins

500 円玉を A 枚、100 円玉を B 枚、50 円玉を C 枚持っている。これらから何枚か選び、合計金額をちょうど X 円にする方法は何通りあるか求めなさい。

- $0 \leq A, B, C \leq 50, A + B + C \geq 1$
- $50 \leq X \leq 20,000$
- X は 50 の倍数である

入力例 1 Copy

```
2
2
2
100
```

出力例 1 Copy

```
2
```

条件を満たす選び方は以下の 2 通りです。

- 500 円玉を 0 枚、100 円玉を 1 枚、50 円玉を 0 枚選ぶ。
- 500 円玉を 0 枚、100 円玉を 0 枚、50 円玉を 2 枚選ぶ。

Coins

これを実現するプログラムを書くには・・・以下の 2 つの処理が必要

1. それぞれのコインについて、使う枚数を全て試す
2. 使う枚数を決めた時の金額を計算し、X を比較
   - 金額が X と等しければ答えの 1 つである

C++ ではどう書くか？

Coins

これを実現するプログラムを書くには・・・以下の 2 つの処理が必要

1. それぞれのコインについて、使う枚数を全て試す
   - 特定の範囲で処理を繰り返すので、これも for 文が有効
   - 3 つのコインについて for 文を回すことになりますが、for は多重でも大丈夫です
2. 使う枚数を決めた時の金額を計算し、X を比較
   - 金額が X と等しければ答えの 1 つである
   - 等しいかどうかで処理が変わるため、if - else 文を使いましょう

C++ ではどう書くか？
実際にコードを見てみよう

Coins

実際の提出: ▸ Link

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

int main() {
    int A, B, C, X; cin >> A >> B >> C >> X;
    int ans = 0;

    // for 文を 3 重にして、コインの枚数の組み合わせを全て試す
    for(int i=0; i<=A; i++) {
        for(int j=0; j<=B; j++) {
            for(int k=0; k<=C; k++) {
                // 合計金額を計算し、X を等しければ答えを増やす
                int sum = 500*i + 100*j + 50*k;
                if(sum == X) ans++;
            }
        }
    }
    cout << ans << endl;
    return 0;
}
```

Some Sums

Some Sums

1 以上 N 以下の整数のうち、10 進法での各桁の和が A 以上 B 以下であるものの総和を求めなさい。

入力例 1 Copy

```
20 2 5
```
Copy

出力例 1 Copy

```
84
```
Copy

20 以下の整数のうち、各桁の和が 2 以上 5 以下なのは $2, 3, 4, 5, 11, 12, 13, 14, 20$ です。これらの合計である 84 を出力します。

Some Sums

これを実現するプログラムを書くには・・・以下の 2 つの処理が必要

1. 1 以上 N 以下の各整数について、桁和を調べる
2. 求めた桁和が A 以上 B 以下であるか調べる
   - 条件を満たすならばその整数を答えに足す必要がある

Some Sums

これを実現するプログラムを書くには・・・以下の 2 つの処理が必要

1. 1 以上 N 以下の各整数について、桁和を調べる
   - 桁和を調べるパートはどう書けるでしょうか？ この後詳しく見ていきます
   - 桁和を調べる処理は「関数」を使って実装するとよいでしょう
2. 求めた桁和が A 以上 B 以下であるか調べる
   - 条件を満たすならばその整数を答えに足す必要がある
   - 条件分岐なので if 文で書くとよさそうですね

まず桁和を求める処理のコードを示し、その後にこの問題に正答できるコードを示します

Some Sums

今やりたい処理

正整数 N が与えられるので、N を 10 進法で表したときの各桁の和がいくつになるか求めよ。

これを実現する処理を考えてみましょう

Some Sums

今やりたい処理

正整数 N が与えられるので、N を 10 進法で表したときの各桁の和がいくつになるか求めよ。

これを実現する処理を考えてみましょう

- $\mathrm{sum} \leftarrow 0$ (sum に 0 を代入)
- N が 0 になるまで以下を実行
  - $\mathrm{sum} \leftarrow \mathrm{sum}+$ (N を 10 で割った余り)
  - $N \leftarrow$ (N を 10 で割って小数点以下切り捨てしたもの)

実際にこれで動くのか、例で確認しましょう

Some Sums

処理のイメージをつかんでみてください。次で実装を示します

実行例

$N = 758$ でやってみよう

- $N = 758, \mathrm{sum} = 0$
  - $\mathrm{sum} \leftarrow 0 + 8 = 8$
  - $N \leftarrow \lfloor \frac{758}{10} \rfloor = 75$
- $N = 75, \mathrm{sum} = 8$
  - $\mathrm{sum} \leftarrow 8 + 5 = 13$
  - $N \leftarrow \lfloor \frac{75}{10} \rfloor = 7$
- $N = 7, \mathrm{sum} = 13$
  - $\mathrm{sum} \leftarrow 13 + 7 = 20$
  - $N \leftarrow \lfloor \frac{7}{10} \rfloor = 0$
- $N = 0, \mathrm{sum} = 20$
  - $N = 0$ になったため、これで処理は終了

Some Sums

処理のイメージをつかんでみてください。次で実装を示します

実行例

$N = 758$ でやってみよう

- $N = 758, \mathrm{sum} = 0$
  - $\mathrm{sum} \leftarrow 0 + 8 = 8$
  - $N \leftarrow \lfloor \frac{758}{10} \rfloor = 75$
- $N = 75, \mathrm{sum} = 8$
  - $\mathrm{sum} \leftarrow 8 + 5 = 13$
  - $N \leftarrow \lfloor \frac{75}{10} \rfloor = 7$
- $N = 7, \mathrm{sum} = 13$
  - $\mathrm{sum} \leftarrow 13 + 7 = 20$
  - $N \leftarrow \lfloor \frac{7}{10} \rfloor = 0$
- $N = 0, \mathrm{sum} = 20$
  - $N = 0$ になったため、これで処理は終了

Some Sums

処理のイメージをつかんでみてください。次で実装を示します

実行例

$N = 758$ でやってみよう

- $N = 758, \mathrm{sum} = 0$
  - $\mathrm{sum} \leftarrow 0 + 8 = 8$
  - $N \leftarrow \lfloor \frac{758}{10} \rfloor = 75$
- $N = 75, \mathrm{sum} = 8$
  - $\mathrm{sum} \leftarrow 8 + 5 = 13$
  - $N \leftarrow \lfloor \frac{75}{10} \rfloor = 7$
- $N = 7, \mathrm{sum} = 13$
  - $\mathrm{sum} \leftarrow 13 + 7 = 20$
  - $N \leftarrow \lfloor \frac{7}{10} \rfloor = 0$
- $N = 0, \mathrm{sum} = 20$
  - $N = 0$ になったため、これで処理は終了

Some Sums

処理のイメージをつかんでみてください。次で実装を示します

実行例

$N = 758$ でやってみよう

- $N = 758, \mathrm{sum} = 0$
  - $\mathrm{sum} \leftarrow 0 + 8 = 8$
  - $N \leftarrow \lfloor \frac{758}{10} \rfloor = 75$
- $N = 75, \mathrm{sum} = 8$
  - $\mathrm{sum} \leftarrow 8 + 5 = 13$
  - $N \leftarrow \lfloor \frac{75}{10} \rfloor = 7$
- $N = 7, \mathrm{sum} = 13$
  - $\mathrm{sum} \leftarrow 13 + 7 = 20$
  - $N \leftarrow \lfloor \frac{7}{10} \rfloor = 0$
- $N = 0, \mathrm{sum} = 20$
  - $N = 0$ になったため、これで処理は終了

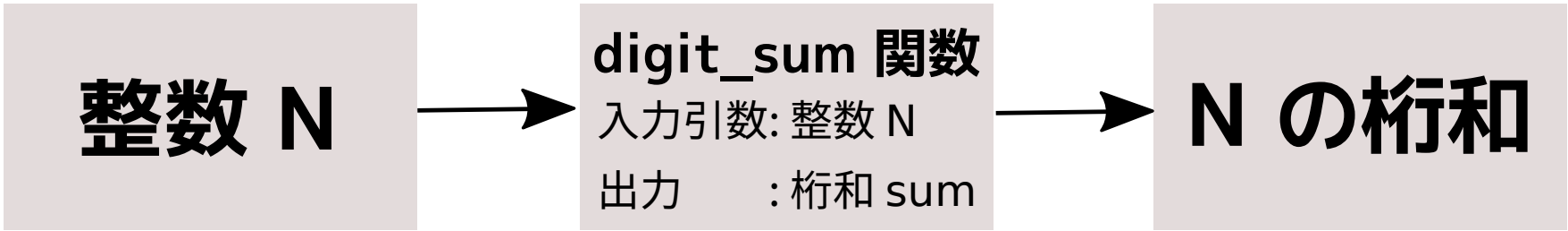
Some Sums

桁和を計算する処理の実装です。次は、これを関数で書いてみましょう

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

// 整数 N が与えられたら、桁和 sum を返すプログラム
int main() {
    int N; cin >> N;
    int sum = 0;

    // N が 0 になったらやめる (0 でない限り処理をつづける)
    while(N != 0) {
        sum = sum + (N % 10);

        // C++ での割り算は小数点以下切り捨てなのでこれで OK
        N = N / 10;
    }

    cout << sum << endl;
    return 0;
}

```

Some Sums

整数 N が与えられると、桁和 sum を返す関数




```
// 関数を定義しよう！
// 入力引数: 整数型の変数 N
// 出力    : 整数型の変数 sum

// [出力の型] [関数名] [入力引数] の順に書きます
int digit_sum(int N) {
    int sum = 0;
    while(N != 0) {
        sum = sum + (N % 10);
        N = N / 10;
    }

    // 関数で返すのは、桁和を表す整数 sum
    return sum;
}
```

Some Sums

関数を書いておくと main 内の処理が簡潔に書けますし、何度も同じ処理を実行しやすくなります

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

// 関数を定義しよう！
// 入力引数: 整数型の変数 N
// 出力    : 整数型の変数 sum

// [出力の型] [関数名] [入力引数] の順に書きます
int digit_sum(int N) {
    int sum = 0;
    while(N != 0) {
        sum = sum + (N % 10);
        N = N / 10;
    }

    // 関数で返すのは、桁和を表す整数 sum
    return sum;
}

// 整数 N が与えられたら、桁和 sum を返すプログラム
int main() {
    int N; cin >> N;

    // 関数を利用！
    cout << digit_sum(N) << endl;
    return 0;
}
```

Some Sums

元の問題を再掲します

Some Sums

1 以上 N 以下の整数のうち、10 進法での各桁の和が A 以上 B 以下であるものの総和を求めなさい。

1. 1 以上 N 以下の各整数について、桁和を調べる
   - 桁和を調べる処理は「関数」を使って実装するとよいでしょう
2. 求めた桁和が A 以上 B 以下であるか調べる
   - 条件を満たすならばその整数を答えに足す必要がある
   - 条件分岐なので if 文で書くとよさそうですね

関数を使って、これを解くプログラムを書いてみましょう

Some Sums

実際の提出: ▸ Link

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

// 桁和を求める関数
int digit_sum(int N) {
    int sum = 0;
    while(N != 0) {
        sum = sum + (N % 10);
        N = N / 10;
    }
    return sum;
}

int main() {
    int N, A, B; cin >> N >> A >> B;

    int ans = 0;
    for(int i=1; i<=N; i++) {
        // 複数の条件を全て満たすものがほしければ
        // 条件を and (&&) で結べば良いです
        if(A <= digit_sum(i) and digit_sum(i) <= B) {
            ans += i;
        }
    }
    cout << ans << endl;
    return 0;
}
```

Card Game for Two

Card Game for Two

N 枚のカードがあり、i 枚目のカードには a_i という数が書かれている。2 人のプレイヤーが交互にカードを 1 枚ずつ取っていき、2 人が全てのカードを取ったときにゲームは終了する。各プレイヤーの得点は、取ったカードの数の合計である。2 人が自分の得点を最大化するように最適な戦略を取ったとき、先手 (Alice) は後手 (Bob) より何点多く取るか求めよ。

入力例 2 Copy

```
3
2 7 4
```

出力例 2 Copy

```
5
```

最初, Alice は 7 が書かれたカードを取ります. 次に, Bob は 4 が書かれたカードを取ります. 最後に, Alice は 2 が書かれたカードを取ります. 得点差は, 7 - 4 + 2 = 5 点となります.

Card Game for Two

これを実現するプログラムを書くには・・・「最適な戦略」をシミュレーションできれば良いです

- 「最適な戦略」とは何か？
  - プレイヤーの得点は、得たカードに書かれた数の合計
  - それぞれは自分の得点を最大にしたい
- この場合は「その時点で存在するカードのうち数が最大であるもの」を取ることが最適な戦略にあたる！
- これをシミュレーションするにはどうすればよいか？

Card Game for Two

これを実現するプログラムを書くには・・・「最適な戦略」をシミュレーションできれば良いです

- 「最適な戦略」とは何か？
  - プレイヤーの得点は、得たカードに書かれた数の合計
  - それぞれは自分の得点を最大にしたい
- この場合は「その時点で存在するカードのうち数が最大であるもの」を取ることが最適な戦略にあたる！
- これをシミュレーションするにはどうすればよいか？

考察

- 与えられた数列 A が $A = \{3, 10, 2, 7, 13, 2, 51, 7\}$ であったとき、このままでシミュレーションはしやすいか？
- 数列に対して、どのような処理を施すとシミュレーションしやすくなるか？

Card Game for Two

- 数列内の要素を降順に並べてみる

$$A = \{3, 10, 2, 7, 13, 2, 51, 7\}$$
$$\downarrow$$
$$A = \{51, 13, 10, 7, 7, 3, 2, 2\}$$

Card Game for Two

- 数列内の要素を降順に並べてみる

$$A = \{3, 10, 2, 7, 13, 2, 51, 7\}$$
$$\downarrow$$
$$A = \{51, 13, 10, 7, 7, 3, 2, 2\}$$

- 降順に並べた状態だと、その時点で最も大きいものは簡単に分かるので、簡単にシミュレーションできる！
  - 赤字: 先手 (Alice) が取る要素
  - 青字: 後手 (Bob) が取る要素

$$A = \{\mathbf{51}, \mathbf{13}, \mathbf{10}, \mathbf{7}, \mathbf{7}, \mathbf{3}, \mathbf{2}, \mathbf{2}\}$$

Card Game for Two

- 数列内の要素を降順に並べてみる

$$A = \{3, 10, 2, 7, 13, 2, 51, 7\}$$

$$\downarrow$$

$$A = \{51, 13, 10, 7, 7, 3, 2, 2\}$$

- 降順に並べた状態だと、その時点で最も大きいものは簡単に分かるので、簡単にシミュレーションできる！
  - 赤字: 先手 (Alice) が取る要素
  - 青字: 後手 (Bob) が取る要素

$$A = \{\mathbf{51}, \mathbf{13}, \mathbf{10}, \mathbf{7}, \mathbf{7}, \mathbf{3}, \mathbf{2}, \mathbf{2}\}$$

Card Game for Two の解法

1. 与えられた数列を降順に並べる
2. (0-indexed で) 偶数番目を先手が、奇数番目を後手が取るものとして総和を計算し、答えを出す

Card Game for Two

1. 数列を降順に並べるにはどうすればいいの？
   - C++ には昇順ソートを行う関数[1] (sort) と、数列を逆順に並べる関数 (reverse) があります
   - sort 関数で昇順に並べたものを、reverse 関数で逆順にすると、降順に並んだ数列が得られます
2. sort や reverse を使うために必要な記述はあるの？
   - `#include <algorithm>` が必要なので、`#include <iostream>` の次の行にでも書いておきましょう

[1] いろいろ指定すれば昇順以外のソートもできますが、ここでは省略します。デフォルトでは昇順ソートです。

Card Game for Two

実際の提出: ▸ Link

```cpp
#include <iostream>
#include <vector>
#include <algorithm> // sort, reverse を使うために必要
using namespace std;

int main() {
    int N; cin >> N;
    vector<int> A(N);
    for(int i=0; i<N; i++) cin >> A[i];

    // 数列を昇順にソート
    sort(A.begin(), A.end());
    // 数列を逆順に並べる（これで降順になる）
    reverse(A.begin(), A.end());

    // （先手が取るものの総和）-（後手が取るものの総和）を計算
    int ans = 0;
    for(int i=0; i<N; i++) {
        if(i % 2 == 0) {
            // 偶数番目なら先手が取る
            ans += A[i];
        }
        else {
            // 奇数番目なら後手が取る
            ans -= A[i];
        }
    }
    cout << ans << endl;
    return 0;
}
```

Kagami Mochi

N 枚の円形の餅があり、i 枚目の餅の直径は d_i センチメートルです。これらの餅の一部または全部を使って、下段にある餅に行くほど直径が大きくなるように (同じ直径を持つものを複数使うことはできません) 鏡餅を作ります。最大で何段重ねの鏡餅を作れるか求めなさい。

- $1 \leq N \leq 100$
- $1 \leq d_i \leq 100$

入力例 1 Copy

```
4
10
8
8
6
```

出力例 1 Copy

```
3
```

直径 10、8、6 センチメートルの餅をこの順に下から積み重ねると 3 段重ねの鏡餅になり、これが最大です。

Kagami Mochi

鏡餅を作れる餅の組み合わせには、どのような制約があるか？

- 直径が d_i, d_j であるような 2 つの餅 i, j について、同時に使えるかどうか考えよう
  - 直径が異なる $(d_i \neq d_j)$ 場合は、大きい方を下にし、小さい方を上にすることでどちらも使用可能
  - 直径が等しい $(d_i = d_j)$ 場合は、どちらか一方しか使用できない
- つまりこの問題は次のように言い換えられる

Kagami Mochi

鏡餅を作れる餅の組み合わせには、どのような制約があるか？

- 直径が d_i, d_j であるような 2 つの餅 i, j について、同時に使えるかどうか考えよう
  - 直径が異なる $(d_i \neq d_j)$ 場合は、大きい方を下にし、小さい方を上にすることでどちらも使用可能
  - 直径が等しい $(d_i = d_j)$ 場合は、どちらか一方しか使用できない
- つまりこの問題は次のように言い換えられる

Kagami Mochi

鏡餅を作れる餅の組み合わせには、どのような制約があるか？

- 直径が d_i, d_j であるような 2 つの餅 i, j について、同時に使えるかどうか考えよう
  - 直径が異なる $(d_i \neq d_j)$ 場合は、大きい方を下にし、小さい方を上にすることでどちらも使用可能
  - 直径が等しい $(d_i = d_j)$ 場合は、どちらか一方しか使用できない
- つまりこの問題は次のように言い換えられる

Kagami Mochi の問題文言い換え

N 個の整数からなる数列があり、i 番目の値は d_i です。この数列に含まれる整数は何種類あるか求めなさい。

- 何種類あるか数えるにはどうすればよいか？

Kagami Mochi

色々なやりかたがありますが、今回は配列を使ってみましょう！

- 長さ 110 で、初期値が全て 0 の配列 A を用意する
  - 長さが 110 なのは d_i の最大値が 100 であって、それに少し余裕を持たせるためです (配列外参照を防ぐためにも余裕をもたせることをおすすめします)
  - この配列は、「d_i という値が数列内にいくつある」という情報を覚えるためのものです
- 各要素 d_i について、対応する A の要素に対して 1 加算する
  - 具体的には $A[d_i]$ に 1 足します
- A の各要素について、0 でない箇所がいくつあるか数えて、それを答えとして出力
  - 要素の値が 0 でないということは、その要素に対応している d_i の値が 1 つ以上存在することを意味します

Kagami Mochi

色々なやりかたがありますが、今回は配列を使ってみましょう！

- 長さ 110 で、初期値が全て 0 の配列 A を用意する
  - 長さが 110 なのは d_i の最大値が 100 であって、それに少し余裕を持たせるためです (配列外参照を防ぐためにも余裕をもたせることをおすすめします)
  - この配列は、「d_i という値が数列内にいくつある」という情報を覚えるためのものです
- 各要素 d_i について、対応する A の要素に対して 1 加算する
  - 具体的には $A[d_i]$ に 1 足します
- A の各要素について、0 でない箇所がいくつあるか数えて、それを答えとして出力
  - 要素の値が 0 でないということは、その要素に対応している d_i の値が 1 つ以上存在することを意味します

Kagami Mochi

色々なやりかたがありますが、今回は配列を使ってみましょう！

- 長さ 110 で、初期値が全て 0 の配列 A を用意する
  - 長さが 110 なのは d_i の最大値が 100 であって、それに少し余裕を持たせるためです (配列外参照を防ぐためにも余裕をもたせることをおすすめします)
  - この配列は、「d_i という値が数列内にいくつある」という情報を覚えるためのものです
- 各要素 d_i について、対応する A の要素に対して 1 加算する
  - 具体的には $A[d_i]$ に 1 足します
- A の各要素について、0 でない箇所がいくつあるか数えて、それを答えとして出力
  - 要素の値が 0 でないということは、その要素に対応している d_i の値が 1 つ以上存在することを意味します

Kagami Mochi

実際の提出: ▸ Link

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

int main() {
    int N; cin >> N;
    vector<int> d(N);
    for(int i=0; i<N; i++) cin >> d[i];

    // d[i] の値をカバーできる程度の数にしておく
    const int MAXD = 110;
    // A[i] := 値が i である要素がいくつあるか を覚える配列
    vector<int> A(MAXD);

    for(int i=0; i<N; i++) {
        // 値が d[i] である要素が 1 増える
        // x に対して 1 増やすのは x++; と書けます
        A[ d[i] ]++;
    }

    int ans = 0;
    // A について、0 でない要素がいくつあるか求める
    for(int i=0; i<MAXD; i++) {
        if(A[i] != 0) ans++;
    }
    cout << ans << endl;
    return 0;
}
```

Kagami Mochi

- 他にも解法はいろいろあります
  - ソートをする
  - set (C++ 標準ライブラリ) を使う
  - map (C++ 標準ライブラリ) を使う
  - unique (C++ 標準ライブラリ) を使う
  - などなど・・・
- 問題制約に応じて、どの方法が適しているか考えましょう
  - 例えば、先程コードで紹介した方法は制約が大きいと使えません
  - 仮に制約が $1 \leq d_i \leq 10^9$ であったときに、どのような問題点が発生するか考えてみましょう

Otoshidama

お年玉袋にはお札 (10000 円札、5000 円札、1000 円札) が N 枚入っていて、合計 Y 円であるそうですが、これは嘘かもしれません。このような状況があり得るか判定し、ありうる場合はお年玉袋の中身の候補を 1 つ見つけなさい。存在しない場合は “-1 -1 -1” と出力しなさい。

- $1 \leq N \leq 2000$
- $1 \leq Y \leq 2 \times 10^7$

入力例 1 Copy

```
9 45000
```

Copy

出力例 1 Copy

```
4 0 5
```

Copy

お年玉袋に 10000 円札 4 枚と 1000 円札 5 枚が入っていれば、合計枚数が 9 枚、合計金額が 45000 円になります。5000 円札 9 枚という可能性も考えられるため、`0 9 0` も正しい出力です。

Otoshidama

- 問題設定が “Coins” に似ている・・・？
  - 10000 円札、5000 円札、1000 円札それぞれについてループを回せば解けるのでは？
- 例えば、次のような解答が考えられる
  - この提出は TLE (Time Limit Exceeded) となってしまう！ ▸ Link

```cpp
#include <iostream>
using namespace std;
int main() {
    int N, Y; cin >> N >> Y;
    for(int i=0; i<=N; i++) {
        for(int j=0; j<=N; j++) {
            for(int k=0; k<=N; k++) {
                if(i + j + k == N and 10000 * i + 5000 * j + 1000 * k == Y) {
                    cout << i << " " << j << " " << k << endl;
                    return 0;
                }
            }
        }
    }
    cout << "-1 -1 -1" << endl;
    return 0;
}
```

ここで、前のスライドでも書いた内容を再掲

計算ステップ数の評価

プログラムを書くときは、それが最悪でどれだけのステップ実行されるかを吟味しよう。もっと詳しく言うと、プログラムの計算量を考えよう。計算量については以下に示す記事が参考になると思います。

- 計算量オーダーの求め方を総整理！ (drken さん) Link
- 初心者向け プログラムの計算量を求める方法 (cotrpepe さん) Link

さきほどのプログラムについて

- 0 から $N-1$ までの範囲でループを回すことを、3 重にして行っている
- このようなループだと N の 3 乗に比例した計算回数がかかる
  - 正確な説明ではありませんが、このような計算量を $O(N^3)$ と言ったりします
  - 今回は $N \leq 2000$ なので、3 乗に比例した計算回数は多すぎて実行制限時間に間に合いません

ここで、前のスライドでも書いた内容を再掲

計算ステップ数の評価

プログラムを書くときは、それが最悪でどれだけのステップ実行されるかを吟味しよう。もっと詳しく言うと、プログラムの計算量を考えよう。計算量については以下に示す記事が参考になると思います。

- 計算量オーダーの求め方を総整理！ (drken さん) Link
- 初心者向け プログラムの計算量を求める方法 (cotrpepe さん) Link

さきほどのプログラムについて

- 0 から $N-1$ までの範囲でループを回すことを、3 重にして行っている
- このようなループだと N の 3 乗に比例した計算回数がかかる
  - 正確な説明ではありませんが、このような計算量を $O(N^3)$ と言ったりします
  - 今回は $N \leq 2000$ なので、3 乗に比例した計算回数は多すぎて実行制限時間に間に合いません

Otoshidama

計算回数が減るように工夫してみよう！

- 10000 円札を x 枚、5000 円札を y 枚、1000 札を z 枚使うとき、以下の条件が成り立っていればよい
  - $x + y + z = N$
  - $10000x + 5000y + 1000z = Y$
- 先程は x, y, z を全部ループで探していたが、無駄な点はないだろうか？

Otoshidama

計算回数が減るように工夫してみよう！

- 10000 円札を x 枚、5000 円札を y 枚、1000 札を z 枚使うとき、以下の条件が成り立っていればよい
  - $x + y + z = N$
  - $10000x + 5000y + 1000z = Y$
- 先程は x, y, z を全部ループで探していたが、無駄な点はないだろうか？
  - $z = N - x - y$ なので、x, y を決めると z も決まる！
  - つまり x, y, z に対してループを回す必要はなく、x, y に対してのみループを回せば良い！

Otoshidama

計算回数が減るように工夫してみよう！

- 10000 円札を x 枚、5000 円札を y 枚、1000 札を z 枚使うとき、以下の条件が成り立っていればよい
  - $x + y + z = N$
  - $10000x + 5000y + 1000z = Y$
- 先程は x, y, z を全部ループで探していたが、無駄な点はないだろうか？
  - $z = N - x - y$ なので、x, y を決めると z も決まる！
  - つまり x, y, z に対してループを回す必要はなく、x, y に対してのみループを回せば良い！
- この解法だと 2 重ループで済むので、N の 2 乗に比例した時間しかかからない
  - これだと間に合う！
  - 実装してみましょう

Otoshidama

実際の提出: ▸ Link

```cpp
#include <iostream>
using namespace std;
int main() {
    int N, Y; cin >> N >> Y;

    // z は引き算で求めて、2 重ループにする
    for(int x=0; x<=N; x++) {
        for(int y=0; y<=N; y++) {
            int z = N - x - y;
            if(z >= 0 and 10000*x + 5000*y + 1000*z == Y) {
                cout << x << " " << y << " " << z << endl;
                return 0;
            }
        }
    }
    cout << "-1 -1 -1" << endl;
    return 0;
}
```

白昼夢 / Daydream

英小文字からなる文字列 S が与えられる。T が空文字列である状態から始め、以下の操作を好きな回数繰り返すことで $S = T$ とすることができるか判定しなさい。

1. T の末尾に “dream” “dreamer” “erase” “eraser” のいずれかを追加する。

- $1 \leq |S| \leq 10^5$
- S は英小文字からなる

入力例 2 Copy

```
dreameraser
```

出力例 2 Copy

```
YES
```

'dream' 'eraser' の順で T の末尾に追加することで $S = T$ とすることができます。

白昼夢 / Daydream

- 方針を立てましょう
- 例えば $S =$ “dreameraser” という文字列を先頭から順に見ると、“dream” で切れば良いのか “dreamer” で切れば良いのかの判断が難しい
- 他のアプローチはあるか？ 考えてみましょう

白昼夢 / Daydream

- 方針を立てましょう
- 例えば $S =$ “dreameraser” という文字列を先頭から順に見ると、“dream” で切れば良いのか “dreamer” で切れば良いのかの判断が難しい
- 他のアプローチはあるか？ 考えてみましょう

- 実は S を 後ろから 順に切っていこうと思うと、一気に見通しが良くなる！
- 前から読むと “dream” が “dreamer” に完全に被っていたが、後ろから読むとどの 2 つを取っても被ることはない
  - “maerd”, “remaerd”, “esare”, “resare” となるので
- したがって、後ろから切っていくと芋づる式に決まる

白昼夢 / Daydream

いくつかの文字列操作が必要

1. 文字列を左右反転したい！
   - C++ に標準で用意されています。reverse をつかいましょう
2. 文字列の長さが知りたい！
   - これも標準で用意されています。length をつかいましょう
3. 部分文字列 (元の文字列の l 文字目から r 文字目までをトリミングしてできる文字列) が欲しい！
   - これも標準で用意されています。substr をつかいましょう

次ページでこれらの使い方含め実装を示します

白昼夢 / Daydream

実際の提出: ▸ Link

```cpp
#include <iostream>  // 入出力に必要
#include <string>    // string の使用と substr に必要
#include <vector>    // vector に必要
#include <algorithm> // reverse に必要
using namespace std;

int main() {
    vector<string> pattern = {"dream", "dreamer", "erase", "eraser"};
    string s; cin >> s;

    // 文字列を反転する (reverse を使えばできます)
    reverse(s.begin(), s.end());
    for(int i=0; i<4; i++) reverse(pattern[i].begin(), pattern[i].end());

    // s.length() で、文字列 s の長さが得られます
    int i = 0, N = s.length();
    bool divide_all = true;
    for(; i<N; ) {
        // パターンがどれか 1 つ見つかったかどうかを表す変数
        bool found_pattern = false;
        for(int k=0; k<4; k++) {
            int len = pattern[k].size();
            // s の i 文字目から、長さ len の部分文字列を substr で得る
            // これが pattern[k] と一致しているなら、そこで分割できる
            if(s.substr(i, len) == pattern[k]) {
                found_pattern = true;
                i += len;
                break;
            }
        }

        // パターンが見つけられなかった場合、分割失敗 (ループを抜ける)
        if(found_pattern == false) {
            divide_all = false;
            break;
        }
    }

    if(divide_all == true) cout << "YES" << endl;
    else cout << "NO" << endl;
    return 0;
}
```

Traveling

AtCoDeer くんは二次元平面上におり、時刻 0 では $(0, 0)$ にいます。いま、この点を出発し、1 以上 N 以下の各 i に対して、時刻 t_i に点 (x_i, y_i) を訪れる予定です。
AtCoDeer くんが時刻 t に点 (x, y) にいるとき、時刻 $t + 1$ では $(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)$ のいずれかに存在できます (その場にとどまることはできません)。AtCoDeer くんが立てた予定が実行可能かどうか判定しなさい。

- $1 \leq N \leq 10^5$
- $0 \leq x_i \leq 10^5$
- $0 \leq y_i \leq 10^5$
- $1 \leq t_i \leq 10^5$
- $t_i < t_{i+1}$ $(1 \leq i \leq N - 1)$

Traveling

入力例 1 Copy

```
2
3 1 2
6 1 1
```

出力例 1 Copy

```
Yes
```

例えば、$(0,0), (0,1), (1,1), (1,2), (1,1), (1,0), (1,1)$ と移動すればよいです。

入力例 2 Copy

```
1
2 100 100
```

出力例 2 Copy

```
No
```

$(0,0)$ にいる状態から 2 秒後に $(100,100)$ にいるのは不可能です。

Traveling

- 方針を立てましょう
- t_i と t_{i+1} の間にする移動について考える
  - $dt = t_{i+1} - t_i, \mathrm{dist} = |x_{i+1} - x_i| + |y_{i+1} - y_i|$ とおく
  - 移動しなければならない距離が移動時間より長い、つまり $\mathrm{dist} > dt$ である場合は明らかに不可能 (以降、$\mathrm{dist} \leq dt$ を仮定します)
- $x_i + y_i$ という値の偶奇に着目
  - 毎秒 x_i か y_i のいずれかについて 1 ずれることから、$x_i + y_i$ の偶奇は毎秒入れ替わる ことがわかる！
  - この偶奇の値が異なっていれば不可能であるし、そうでなければ可能である
    - いったん目的地にたどり着いた後は、そこを行ったり来たりすることで必ず達成できる

Traveling

- 方針を立てましょう
- t_i と t_{i+1} の間にする移動について考える
  - $dt = t_{i+1} - t_i, \mathrm{dist} = |x_{i+1} - x_i| + |y_{i+1} - y_i|$ とおく
  - 移動しなければならない距離が移動時間より長い、つまり $\mathrm{dist} > dt$ である場合は明らかに不可能 (以降、$\mathrm{dist} \leq dt$ を仮定します)
- $x_i + y_i$ という値の偶奇に着目
  - 毎秒 x_i か y_i のいずれかについて 1 ずれることから、$x_i + y_i$ の偶奇は毎秒入れ替わる ことがわかる！
  - この偶奇の値が異なっていれば不可能であるし、そうでなければ可能である
    - もう少し自明にならないかな？

Traveling

- 方針を立てましょう
- t_i と t_{i+1} の間にする移動について考える
  - $dt = t_{i+1} - t_i, \mathrm{dist} = |x_{i+1} - x_i| + |y_{i+1} - y_i|$ とおく
  - 移動しなければならない距離が移動時間より長い、つまり $\mathrm{dist} > dt$ である場合は明らかに不可能 (以降、$\mathrm{dist} \leq dt$ を仮定します)
- $x_i + y_i$ という値の偶奇に着目
  - 毎秒 x_i か y_i のいずれかについて 1 ずれることから、$x_i + y_i$ の偶奇は毎秒入れ替わる ことがわかる！
  - この偶奇の値が異なっていれば不可能であるし、そうでなければ可能である
    - いったん目的地にたどり着いた後は、そこを行ったり来たりすることで必ず達成できる

Traveling

- 方針を立てましょう
- t_i と t_{i+1} の間にする移動について考える
  - $dt = t_{i+1} - t_i, \mathrm{dist} = |x_{i+1} - x_i| + |y_{i+1} - y_i|$ とおく
  - 移動しなければならない距離が移動時間より長い、つまり $\mathrm{dist} > dt$ である場合は明らかに不可能 (以降、$\mathrm{dist} \leq dt$ を仮定します)
- $x_i + y_i$ という値の偶奇に着目
  - 毎秒 x_i か y_i のいずれかについて 1 ずれることから、$x_i + y_i$ の偶奇は毎秒入れ替わる ことがわかる！
  - この偶奇の値が異なっていれば不可能であるし、そうでなければ可能である
    - いったん目的地にたどり着いた後は、そこを行ったり来たりすることで必ず達成できる

Traveling

実際の提出: ▸ Link

$t_0 = 0, (x_0, y_0) = (0, 0)$ という情報を追加して実装すると楽です

```cpp
#include <iostream>
#include <vector>
#include <cstdlib> // abs を使うために必要
using namespace std;

int main() {
    int N; cin >> N;
    vector<int> t(N+1), x(N+1), y(N+1);

    t[0] = x[0] = y[0] = 0; // 初期状態
    for(int i=1; i<=N; i++) cin >> t[i] >> x[i] >> y[i];

    bool can_travel = true;
    for(int i=0; i<N; i++) {
        int dt = t[i+1] - t[i];
        // 絶対値は abs で取得可能
        int dist = abs(x[i+1] - x[i]) + abs(y[i+1] - y[i]);

        if(dist > dt) can_travel = false;
        if((x[i+1] + y[i+1]) % 2 == (x[i] + y[i]) % 2) {
            // 偶奇が同じ場合は dt が偶数でなければならない
            if(dt % 2 != 0) can_travel = false;
        }
        else {
            // 偶奇が異なる場合は dt が奇数でなければならない
            if(dt % 2 != 1) can_travel = false;
        }
    }
    if(can_travel) cout << "Yes" << endl;
    else cout << "No" << endl;
    return 0;
}
```

参考になる書籍

おそらくすべて北大の図書館にあります。一応 Amazon へのリンクもつけておきます。

1. プログラミングコンテスト攻略のためのアルゴリズムとデータ構造 Link
   - Aizu Online Judge というオンラインジャッジの問題を使いながら、基礎的なアルゴリズムの演習をしていく本です
   - コード例が豊富に載っているので実装の参考にもなると思います
2. プログラミングコンテストチャレンジブック Link
   - 競技プログラマーのバイブル的存在である本です。基礎的なトピックから応用的なものまで実に様々な話題が詰まっています。
   - 最初にこれを勉強すると少しハードルが高く感じるかもしれません (わからないところがあれば質問してきてくださいね！)
3. 最強最速アルゴリズマー養成講座 Link
   - AtCoder (株) 代表取締役社長の chokudai さんが著した本です
   - TopCoder の問題演習をする実践的な内容で、C# ・ C++ ・ Java の解答例が載っています (特に C# の実装例が欲しい方におすすめ)

さいごに

1. 今回はプログラミングコンテストの入り口の部分を紹介しました
2. 実際にもっと問題を解いてみましょう！ AtCoder の過去問を探すには以下のサイトが便利です
   - AtCoder Problems (過去問一覧) Link
   - AtCoder Scores (問題を得点順に見る) Link
3. コンテストにも参加しましょう！
   - AtCoder Link
   - Codeforces Link
   - CSAcademy Link
   - TopCoder Link
   - LeetCode Link
4. 興味があれば HCPC の他の活動・スライドもチェック！
   - ホームページ: `http://hcpc.web.fc2.com/` Link
   - Twitter: `https://twitter.com/hcpc_hokudai` Link
   - SlideShare: `https://www.slideshare.net/hcpc_hokudai` Link

- END -