

大手前プロコン 2019 F

天秤とコイン 解説

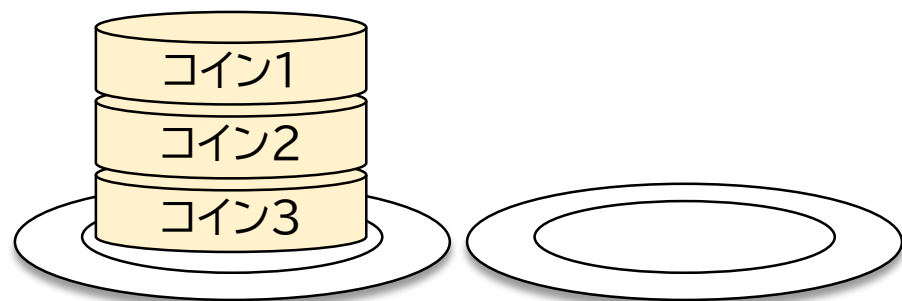
Pro_ktmr

問題概要

- 天秤がある
- 左側の皿に $N (\leq 2000)$ 枚のコインが積まれていて、上から $1 \sim N$ の名前がついていて、質量 $M (\leq 10^9)$ は毎日変化する
- いつでも左側の皿の一番上のコインを右側の皿に移動させられる（右から左への移動や一番上以外のコインの移動は不可）
- $D (\leq 2000)$ 日間、左右の皿の合計質量の差の絶対値をコストとして支払う
- 支払うコストの最小値を求めたい

入出力例1

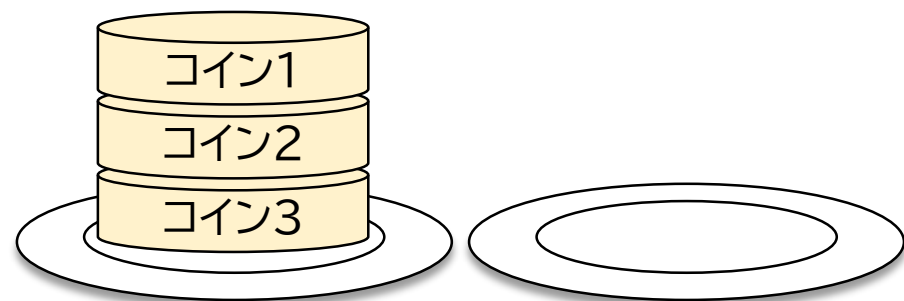
	コイン1	コイン2	コイン3
1日目	10[g]	8[g]	3[g]
2日目	5[g]	7[g]	8[g]
3日目	9[g]	6[g]	1[g]
4日目	2[g]	8[g]	9[g]



- 質量は質量だとわかるよう、勝手に単位[g]をつけることにする
- コストの単位も[g]にしちゃう

入出力例1

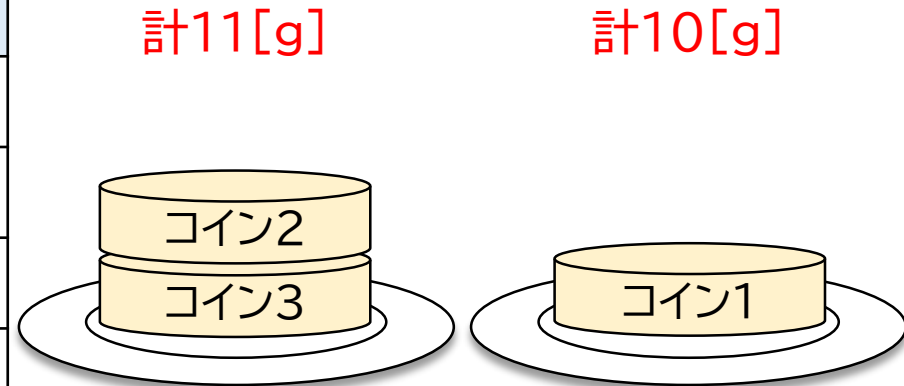
	コイン1	コイン2	コイン3
1日目	10[g]	8[g]	3[g]
2日目	5[g]	7[g]	8[g]
3日目	9[g]	6[g]	1[g]
4日目	2[g]	8[g]	9[g]



- 以下のように操作を行うと, 払う合計コストは 14 となり, このときが最小である.
 - 1 日目にはコインを 1 枚移動する. 払うコストは $|10 - (8+3)| = 1$ である.
 - 2 日目にはコインを 1 枚も移動しない. 払うコストは $|5 - (7+8)| = 10$ である.
 - 3 日目にはコインを 1 枚も移動しない. 払うコストは $|9 - (6+1)| = 2$ である.
 - 4 日目にはコインを 1 枚移動する. 払うコストは $|(2+8) - 9| = 1$ である.
- だそうなので実際にやってみる

入出力例1 : 1日目

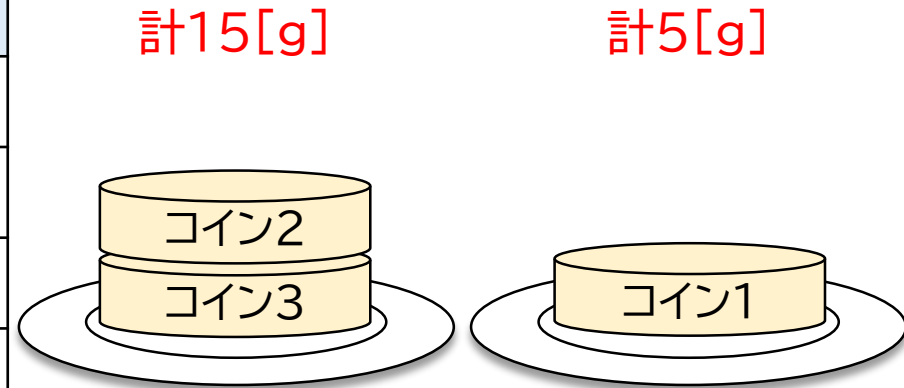
	コイン1	コイン2	コイン3	コスト
1日目	10[g]	8[g]	3[g]	1[g]
2日目	5[g]	7[g]	8[g]	
3日目	9[g]	6[g]	1[g]	
4日目	2[g]	8[g]	9[g]	



- 以下のように操作を行うと、払う合計コストは 14 となり、このときが最小である。
 - 1 日目にはコインを 1 枚移動する. 払うコストは $|10 - (8+3)| = 1$ である.
 - 2 日目にはコインを 1 枚も移動しない. 払うコストは $|5 - (7+8)| = 10$ である.
 - 3 日目にはコインを 1 枚も移動しない. 払うコストは $|9 - (6+1)| = 2$ である.
 - 4 日目にはコインを 1 枚移動する. 払うコストは $|(2+8) - 9| = 1$ である.
- だそうなので実際にやってみる

入出力例1 : 2日目

	コイン1	コイン2	コイン3	コスト
1日目	10[g]	8[g]	3[g]	1[g]
2日目	5[g]	7[g]	8[g]	10[g]
3日目	9[g]	6[g]	1[g]	
4日目	2[g]	8[g]	9[g]	

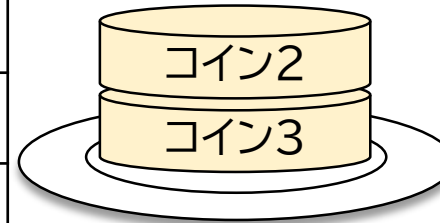


- 以下のように操作を行うと, 払う合計コストは 14 となり, このときが最小である.
 - 1 日目にはコインを 1 枚移動する. 払うコストは $|10 - (8+3)| = 1$ である.
 - 2 日目にはコインを 1 枚も移動しない. 払うコストは $|5 - (7+8)| = 10$ である.
 - 3 日目にはコインを 1 枚も移動しない. 払うコストは $|9 - (6+1)| = 2$ である.
 - 4 日目にはコインを 1 枚移動する. 払うコストは $|(2+8) - 9| = 1$ である.
- だそうなので実際にやってみる

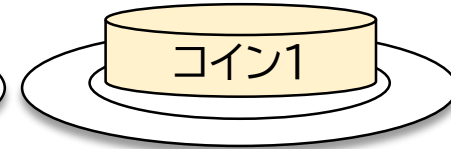
入出力例1 : 3日目

	コイン1	コイン2	コイン3	コスト
1日目	10[g]	8[g]	3[g]	1[g]
2日目	5[g]	7[g]	8[g]	10[g]
3日目	9[g]	6[g]	1[g]	2[g]
4日目	2[g]	8[g]	9[g]	

計7[g]



計9[g]



- 以下のように操作を行うと, 払う合計コストは 14 となり, このときが最小である.
 - 1 日目にはコインを 1 枚移動する. 払うコストは $|10 - (8+3)| = 1$ である.
 - 2 日目にはコインを 1 枚も移動しない. 払うコストは $|5 - (7+8)| = 10$ である.
 - 3 日目にはコインを 1 枚も移動しない. 払うコストは $|9 - (6+1)| = 2$ である.
 - 4 日目にはコインを 1 枚移動する. 払うコストは $|(2+8) - 9| = 1$ である.
- だそうなので実際にやってみる

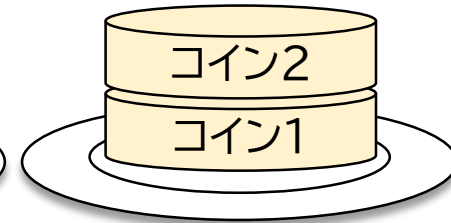
入出力例1 : 4日目

	コイン1	コイン2	コイン3	コスト
1日目	10[g]	8[g]	3[g]	1[g]
2日目	5[g]	7[g]	8[g]	10[g]
3日目	9[g]	6[g]	1[g]	2[g]
4日目	2[g]	8[g]	9[g]	1[g]

計9[g]



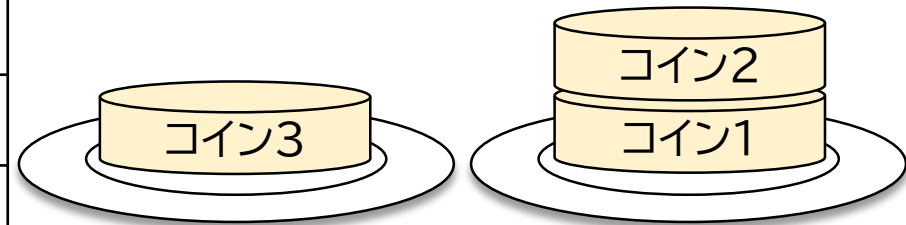
計10[g]



- 以下のように操作を行うと, 払う合計コストは 14 となり, このときが最小である.
 - 1 日目にはコインを 1 枚移動する. 払うコストは $|10 - (8+3)| = 1$ である.
 - 2 日目にはコインを 1 枚も移動しない. 払うコストは $|5 - (7+8)| = 10$ である.
 - 3 日目にはコインを 1 枚も移動しない. 払うコストは $|9 - (6+1)| = 2$ である.
 - 4 日目にはコインを 1 枚移動する. 払うコストは $|(2+8) - 9| = 1$ である.
- だそうなので実際にやってみる

入出力例1 : 4日目

	コイン1	コイン2	コイン3	コスト
1日目	10[g]	8[g]	3[g]	1[g]
2日目	5[g]	7[g]	8[g]	10[g]
3日目	9[g]	6[g]	1[g]	2[g]
4日目	2[g]	8[g]	9[g]	1[g]



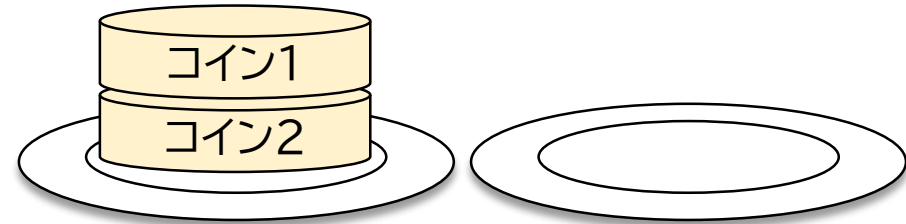
- 以下のように操作を行うと, 払う合計コストは 14 となり, このときが最小である.
 - 1 日目にはコインを 1 枚移動する. 払うコストは $|10 - (8+3)| = 1$ である.
 - 2 日目にはコインを 1 枚も移動しない. 払うコストは $|5 - (7+8)| = 10$ である.
 - 3 日目にはコインを 1 枚も移動しない. 払うコストは $|9 - (6+1)| = 2$ である.
 - 4 日目にはコインを 1 枚移動する. 払うコストは $|(2+8) - 9| = 1$ である.
- 確かに上の操作をすると合計コストは14である

小課題1 (8点)

$N=2$

小課題1

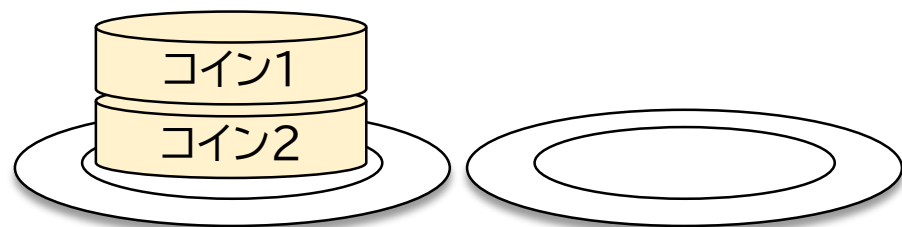
	コイン1	コイン2
1日目		
2日目	?	
3日目		
⋮		⋮



- コインの枚数が2枚で確定
- 何かいい性質があるのだろうか・・・？

小課題1

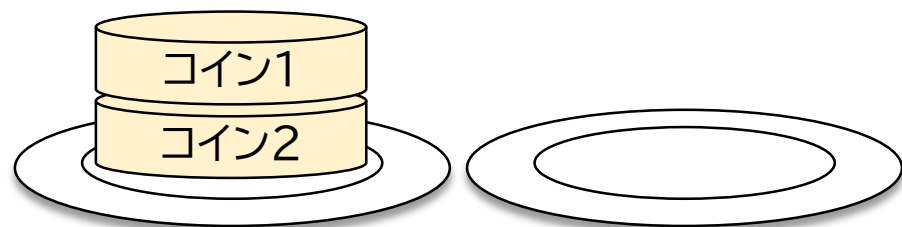
	コイン1	コイン2
1日目		
2日目	?	
3日目		
⋮	⋮	⋮



- ある日の質量がそれぞれ $X[g]$, $Y[g]$ だとする
- いずれも移動していない: コスト $X+Y[g]$
- コイン1だけ移動している: コスト $|X-Y|[g]$
- 両方移動している: コスト $X+Y[g]$

小課題1

	コイン1	コイン2
1日目		
2日目	?	
3日目		
⋮	⋮	⋮



- ある日の質量がそれぞれ $X[g]$, $Y[g]$ だとする
 - いずれも移動していない: コスト $X+Y[g]$
 - コイン1だけ移動している: コスト $|X-Y|[g]$
 - 両方移動している: コスト $X+Y[g]$
- 明らかにコイン1だけが移動しているときが最小

小課題1

- コインが2枚しかないとき、それぞれの質量がなんであれ、2枚のコインは別の皿に乗っている方がいい
- これはすべての日について当てはまる
- よって、1日目にコイン1を移動させて、そのあとは放置するのが最適
- このとき各日のコストは $|X-Y|$ になる

実装例(C++):O(DN)

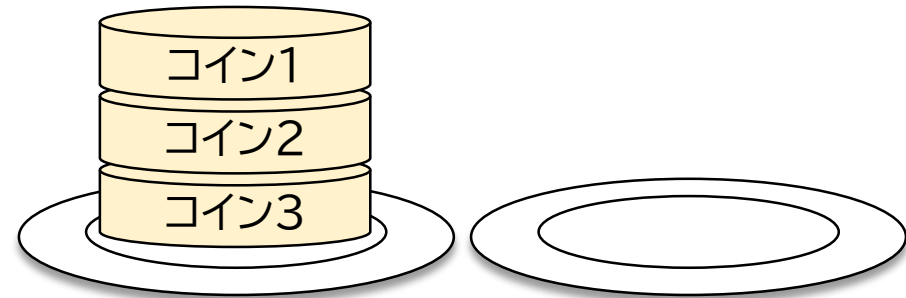
```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int N, D;
signed main(){
    cin >> N >> D;
    int ans = 0;
    for(int i=0; i<D; i++){
        int X, Y;
        cin >> X >> Y;
        ans += abs(X-Y);
    }
    cout << ans << endl;
}
```

小課題2(8点)

$N=3$

小課題2

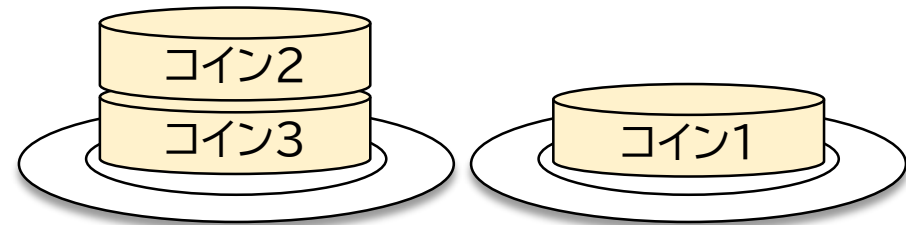
	コイン1	コイン2	コイン3
1日目		?	
2日目			
3日目			
⋮	⋮	⋮	⋮



- $N=3$
- 小課題1と同じで N が小さい値で固定されている
- 小課題1と似たような感じでなんとかならないか・・・？

小課題2

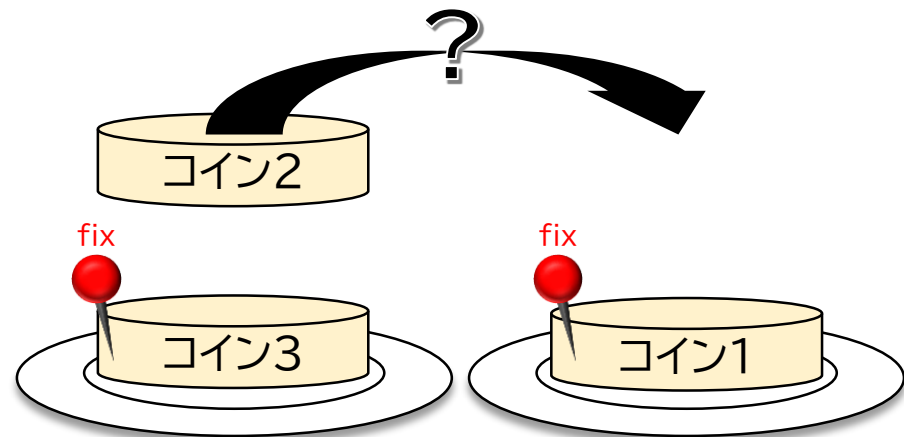
	コイン1	コイン2	コイン3
1日目		?	
2日目			
3日目			
⋮	⋮	⋮	⋮



- 小課題1と同じ理屈で、片方の皿の上にコインが載っていないのは明らかに損
- 1日目の時点でひとまずコイン1を右に移動させるのがよい

小課題2

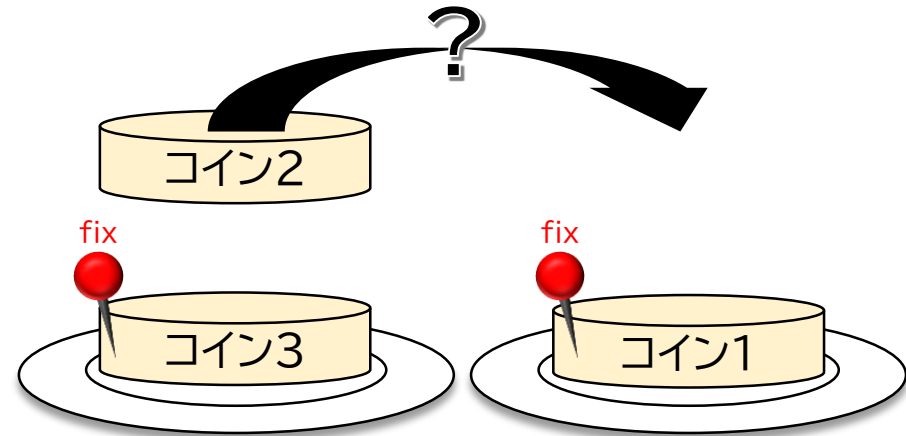
	コイン1	コイン2	コイン3
1日目			
2日目		?	
3日目		?	
⋮	⋮	⋮	⋮



- 小課題1と同じ理屈で、片方の皿の上にコインが載っていないのは明らかに損
- コイン3を右側に移すのも損
- よって、コイン2だけ動かせると考えてもよい

小課題2

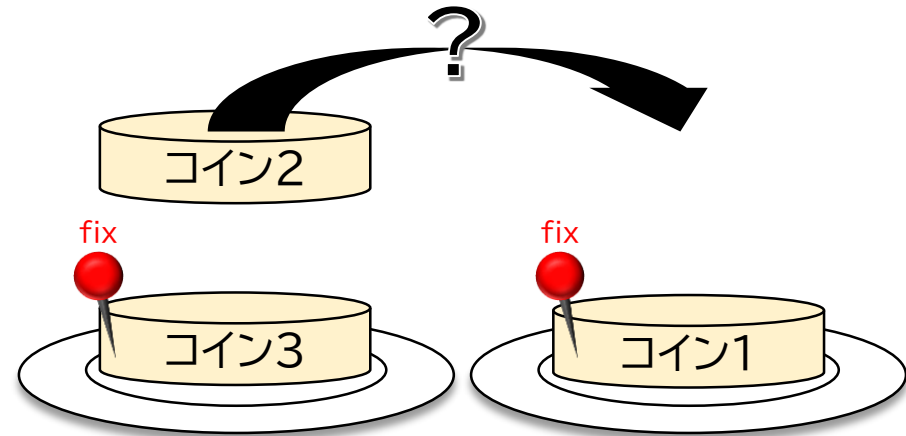
	コイン1	コイン2	コイン3
1日目			
2日目		?	
3日目			
⋮	⋮	⋮	⋮



- コイン2はどのタイミングで移動させるのが最適・・・？

小課題2

	コイン1	コイン2	コイン3
1日目			
2日目		?	
3日目			
⋮	⋮	⋮	⋮



- コイン2はどのタイミングで移動させるのが最適・・・？
- これはわからないので、全パターン(D通り)を確かめる

実装例(C++): $O(DN)$

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int N, D;
int M[2000][2000];
signed main(){
    cin >> N >> D;
    for(int i=0; i<D; i++){
        for(int j=0; j<N; j++){
            cin >> M[i][j];
        }
    }
    int tmp = 0;
    for(int i=0; i<D; i++){
        tmp += abs(M[i][0] + M[i][1] - M[i][2]);
    }
    int ans = tmp;
    for(int i=0; i<D; i++){
        tmp -= abs(M[i][0] + M[i][1] - M[i][2]);
        tmp += abs(M[i][0] - M[i][1] - M[i][2]);
        ans = min(ans, tmp);
    }
    cout << ans << endl;
}
```

実装例(C++):O(DN)

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int N, D;
int M[2000][2000];
signed main(){
    cin >> N >> D;
    for(int i=0; i<D; i++){
        for(int j=0; j<N; j++){
            cin >> M[i][j];
        }
    }
    int tmp = 0;
    for(int i=0; i<D; i++){
        tmp += abs(M[i][0] + M[i][1] - M[i][2]); 1日目にコイン2を移動させる場合
    }
    int ans = tmp;
    for(int i=0; i<D; i++){
        tmp -= abs(M[i][0] + M[i][1] - M[i][2]); i+2日目にコイン2を移動させる場合
        tmp += abs(M[i][0] - M[i][1] - M[i][2]); に修正するために差分計算
        ans = min(ans, tmp);
    }
    cout << ans << endl;
}
```

小課題3 (14点)

$N \leq 10, D \leq 10$

小課題3

- N も D も大きくなって、今までのような特定の場合にだけ使える性質はなさそう
- N も D も十分小さいので全パターンを試せそう
(全探索)

全探索とは

- 全部を探索すること
- 今回の問題の場合、場合分けが非常に多い
- 樹形図が描きたくなる
- そういう場合は再帰関数を使うとよい

再帰関数による全探索

- 引数の決め方が超重要
- なるべく引数以外の値を触らないようにする
- 引数をなるべく少なくするとよい
- 今回の問題の場合、「今何日目？」「今左側の皿の一番上のコインの名前は？」があれば各状況を十分表現できる
- 日数もコインの名前も0-originにする

再帰関数の中身

- 引数は「今何日目？」「今左側の皿の一番上のコインの名前は？」
- 次の日になるまでにコインをどこまで動かすかでループを回すことにする
- 小課題2と同様に差分計算を行う
- D日間が終わったときの処理を忘れずに

実装例(C++) 1ページ目

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int N, D;
int M[2000][2000];

int solve(int day, int coin){
    if(day == D) return 0;
    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1, coin);
    for(int i=coin; i<N; i++){
        tmp += 2*M[day][i];
        ret = min(ret, abs(tmp) + solve(day+1, i+1));
    }
    return ret;
}
```

実装例(C++) 2ページ目

```
signed main(){
    cin >> N >> D;
    for(int i=0; i<D; i++){
        for(int j=0; j<N; j++){
            cin >> M[i][j];
        }
    }
    cout << solve(0, 0) << endl;
}
```

時間計算量

- ちょっと難しい(読み飛ばしてもよい)
- $\text{day} = D$ に到達するまでの経路数を考える
 - 必ずすべての日について処理しなければならない(各経路について、すべての日を順に1回ずつ通っている)
 - coinの値はすべての値を取る必要がないが、dayが増加するにつれて、広義単調増加である
 - 例) $(\text{day}, \text{coin}): (0, 0) \rightarrow (1, 3) \rightarrow (2, 3) \rightarrow (3, 5)$
 - dayの値の取り方はどの経路も同じで、coinが特殊
 - よって、経路数は0以上N以下の整数がD個並んでいる広義単調増加な数列の数と等しい

時間計算量

- 要素数がDですべての要素が0以上N以下の広義単調増加な数列は何種類ある？
 - 数学A(場合の数)を勉強する
 - $N=5$ とする
 - 答えは $\bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \mid \mid \mid \mid \mid$ の並べ方と同値である
 - 左端が0で、右に進み、 $\mid$ を1本超えたら1大きくなると考える
 - 例) $\bigcirc \mid \mid \bigcirc \mid \bigcirc \bigcirc \mid \bigcirc \mid$ は $\{0, 2, 3, 3, 4\}$ を意味する
 - この並べ方は同じものを含む順列で ${}_{10}C_5$ である

時間計算量

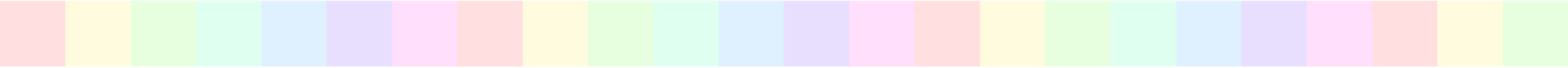
- 一般化すると、 $\text{day} = D$ にたどり着くまでの経路数は $_{N+D}C_N$ 通りである
- `solve` 関数の中には $O(N)$ のループが存在している
- よってソースコード全体では $O(_{N+D}C_N N)$ である

小課題4(24点)

$N \leq 200, D \leq 200$

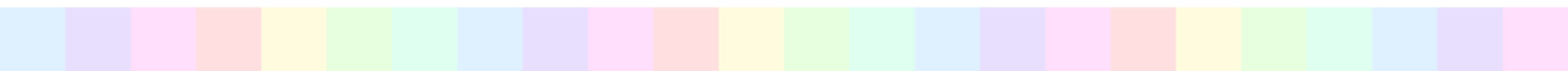
小課題4

- NもDもちょっと大きくなった
- 小課題3の解法の効率化を考えられないか？



動的計画法

-Dynamic Programming-



メモ化再帰

- 再帰関数を書いたらメモ化できないか考える
- 引数の値が小さく、配列を作れる
- 引数の値が同じなら何回その関数を呼び出しても値が変わらない
- →メモ化できる！！！！
- メモ化再帰を一般に動的計画法(DP)と呼ぶ
 - ・ 厳密にはDPの実装方法の1つがメモ化再帰

小課題3の実装例

```
int solve(int day, int coin){
    if(day == D) return 0;

    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1, coin);
    for(int i=coin; i<N; i++){
        tmp += 2*M[day][i];
        ret = min(ret, abs(tmp) + solve(day+1, i+1));
    }
    return ret;
}
```

実装例(C++)

```
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return memo[day][coin];
    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1, coin);
    for(int i=coin; i<N; i++){
        tmp += 2*M[day][i];
        ret = min(ret, abs(tmp) + solve(day+1, i+1));
    }
    return memo[day][coin] = ret;
}
```

実装例(C++)

```
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return memo[day][coin];
    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1, coin);
    for(int i=coin; i<N; i++){
        tmp += 2*M[day][i];
        ret = min(ret, abs(tmp) + solve(day+1, i+1));
    }
    return memo[day][coin] = ret;
}
```

ちょっとメモ処理を書き足すだけでめっちゃ高速化！！
DP最高！！

実装例の全貌

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int N, D;
int M[2000][2000];
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return memo[day][coin];
    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1, coin);
    for(int i=coin; i<N; i++){
        tmp += 2*M[day][i];
        ret = min(ret, abs(tmp) + solve(day+1, i+1));
    }
    return memo[day][coin] = ret;
}
signed main(){
    cin >> N >> D;
    for(int i=0; i<D; i++){
        for(int j=0; j<N; j++){
            cin >> M[i][j];
            memo[i][j] = -1;
        }
        memo[i][N] = -1;
    }
    cout << solve(0, 0) << endl;
}
```

実装例の全貌

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int N, D;
int M[2000][2000];
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return memo[day][coin];
    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1, coin);
    for(int i=coin; i<N; i++){
        tmp += 2*M[day][i];
        ret = min(ret, abs(tmp) + solve(day+1, i+1));
    }
    return memo[day][coin] = ret;
}
signed main(){
    cin >> N >> D;
    for(int i=0; i<D; i++){
        for(int j=0; j<N; j++){
            cin >> M[i][j];
            memo[i][j] = -1;
        }
        memo[i][N] = -1;
    }
    cout << solve(0, 0) << endl;
}
```

メモ用配列の初期化を忘れずに

時間計算量

- 配列、つまり表を埋めていくようなイメージ
- 再帰関数の中にN回のループがあるから、表を1マス埋めるのに $O(N)$
- 表は $O(DN)$ マスある
- よって全体では $O(DN^2)$

```
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return
memo[day][coin];
    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1,
coin);
    for(int i=coin; i<N; i++){
        tmp += 2*M[day][i];
        ret = min(ret, abs(tmp) +
solve(day+1, i+1));
    }
    return memo[day][coin] = ret;
}
```

小課題5 (46点)

追加制約なし ($N \leq 2000, D \leq 2000$)

小課題5

- N も D もさらにちょっと大きくなった
- 小課題4の解法の効率化を考えられないか？
- メモ化再帰の配列部分で $O(DN)$
- 再帰の中身部分で $O(N)$

小課題5

- NもDもさらにちょっと大きくなった
- 小課題4の解法の効率化を考えられないか？
- メモ化再帰の配列部分で $O(DN)$
- 再帰の中身部分で $O(N)$
- 実はまだ引数を減らせる??

小課題5

- NもDもさらにちょっと大きくなった
- 小課題4の解法の効率化を考えられないか？
- メモ化再帰の配列部分で $O(DN)$
- 再帰の中身部分で $O(N)$
- 実はまだ引数を減らせる??

小課題5

- NもDもさらにちょっと大きくなった
- 小課題4の解法の効率化を考えられないか？
- メモ化再帰の配列部分で $O(DN)$
- 再帰の中身部分で $O(N)$
- 実は中身のループをなくせる??

小課題5

- NもDもさらにちょっと大きくなった
- 小課題4の解法の効率化を考えられないか？
- メモ化再帰の配列部分で $O(DN)$
- 再帰の中身部分で $O(N)$
- 実は中身のループをなくせる??

中身のループとは

- 今、再帰関数の中にはループが2つある
- ① とりあえずコインを動かさなかった場合のコストを計算
- ② コインを1枚ずつ動かしてみて差分計算
- 両方のループを取り除かなければ時間計算量は変わらない！ 1つずつなくしていく

さっきの実装例

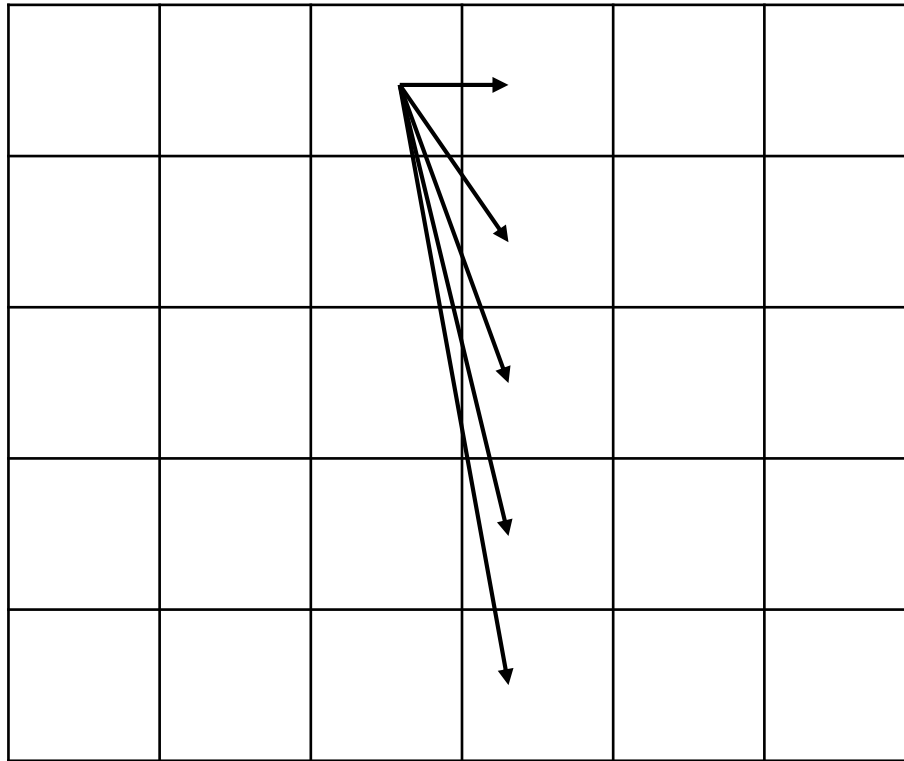
```
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return memo[day][coin];
    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1, coin);
    for(int i=coin; i<N; i++){
        tmp += 2*M[day][i];
        ret = min(ret, abs(tmp) + solve(day+1, i+1));
    }
    return memo[day][coin] = ret;
}
```

①

②

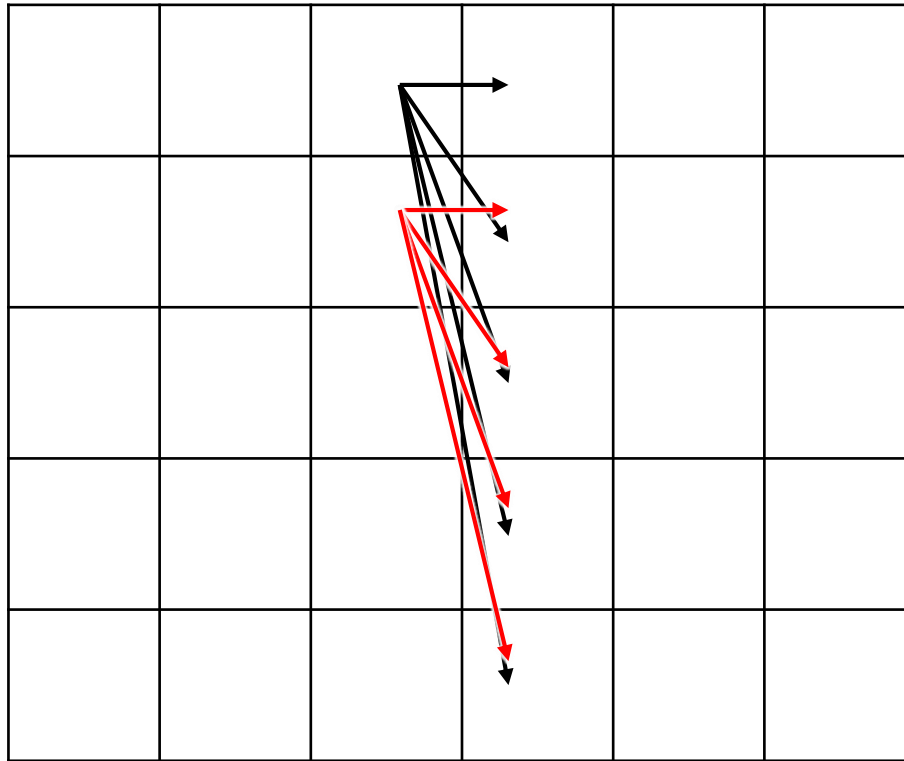
② コインを動かして差分計算

- 今の処理のままでは遷移先がたくさんある



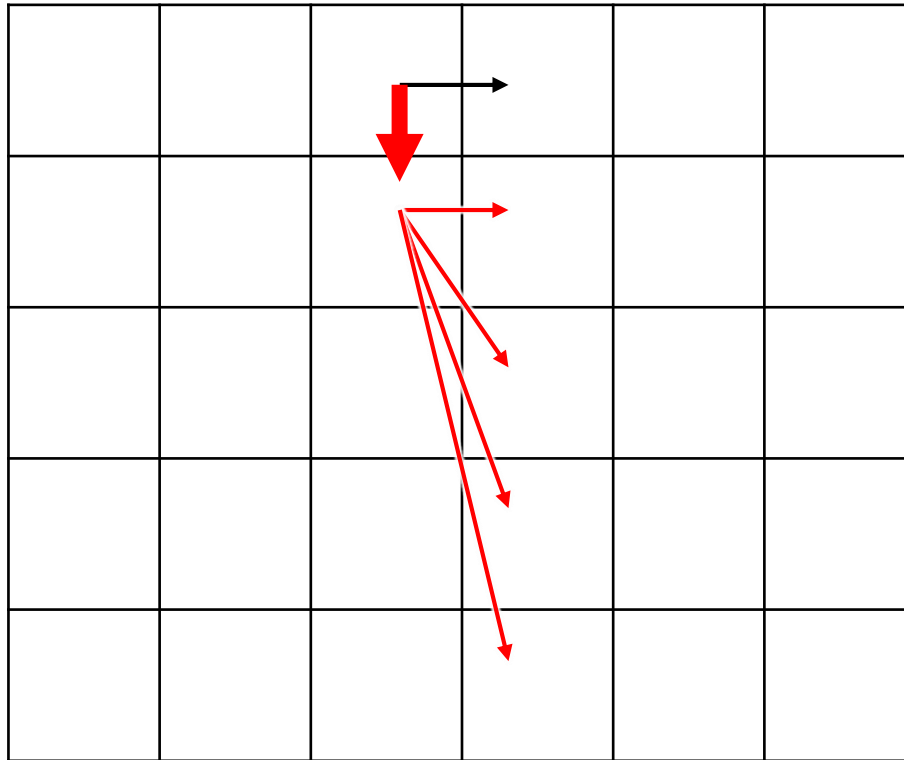
② コインを動かして差分計算

- 今の処理のままでは遷移先がたくさんある
- もちろん($\text{day}, \text{coin} + 1$)でも



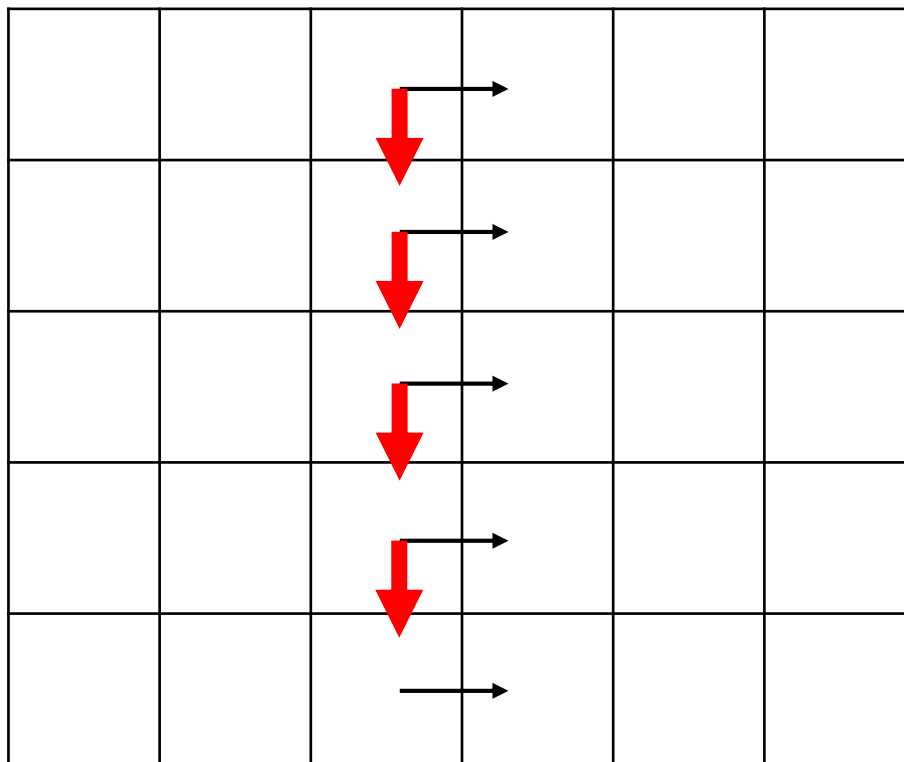
② コインを動かして差分計算

- 今の処理のままでは遷移先がたくさんある
- 遷移先が被っているからまとめられる



② コインを動かして差分計算

- 今の処理のままでは遷移先がたくさんある
- 最終的にはこんな感じに



遷移をまとめる

- 遷移先が被っていたらまとめられないか考えてみる
- 自分で処理せずにとりよりのマスに託す感覚

さっきの実装例

```
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return memo[day][coin];
    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1, coin);
    for(int i=coin; i<N; i++){
        tmp += 2*M[day][i];
        ret = min(ret, abs(tmp) + solve(day+1, i+1));
    }
    return memo[day][coin] = ret;
}
```

①

②

さっきの実装例

```
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return memo[day][coin];
    int tmp = 0;
    for(int i=0; i<N; i++){
        if(i < coin) tmp += M[day][i];           ①
        else tmp -= M[day][i];
    }
    int ret = abs(tmp) + solve(day+1, coin);

    if(coin < N) ret = min(ret, solve(day, coin+1)); ②

    return memo[day][coin] = ret;
}
```

① コインを動かさないときのコスト

- コインを動かさなかった場合のコストを計算している

```
for(int i=0; i<N; i++){  
    if(i < coin) tmp += M[day][i];  
    else tmp -= M[day][i];  
}
```

① コインを動かさないときのコスト

- コインを動かさなかった場合のコストを計算している

```
for(int i=0; i<N; i++){  
    if(i < coin) tmp += M[day][i];  
    else tmp -= M[day][i];  
}
```

- 式で書くと

$$tmp = \sum_{i=0}^{N-1} \begin{cases} M_{d,i}, & i < c \\ -M_{d,i}, & otherwise \end{cases}$$

① コインを動かさないときのコスト

- $tmp = \sum_{i=0}^{N-1} \begin{cases} M_{d,i}, & i < c \\ -M_{d,i}, & otherwise \end{cases}$
- $tmp = \sum_{i=0}^{N-1} \begin{cases} M_{d,i}, & i < c \\ -M_{d,i}, & i \geq c \end{cases}$
- $tmp = \sum_{i=0}^{c-1} M_{d,i} + \sum_{i=c}^{N-1} (-M_{d,i})$
- $tmp = \sum_{i=0}^{c-1} M_{d,i} - \sum_{i=c}^{N-1} M_{d,i}$
- 式がごちゃっとならなうときはきれいになるよう変形することも大切

① コインを動かさないときのコスト

- $tmp = \sum_{i=0}^{c-1} M_{d,i} - \sum_{i=c}^{N-1} M_{d,i}$
- 結局、 i が $[a,b]$ における $M_{d,i}$ の和がわかればよい

① コインを動かさないときのコスト

- $tmp = \sum_{i=0}^{c-1} M_{d,i} - \sum_{i=c}^{N-1} M_{d,i}$
- 結局、 i が $[a,b]$ における $M_{d,i}$ の和がわかればよい
- 累積和を取ればよい

累積和とは

- 累積した和
- もとの数列をA、累積和を取った後の数列をBとする
- $B_i = \sum_{j=0}^i A_j = B_{i-1} + A_i$
- このとき、区間[a,b]におけるAの部分 and を簡単に求められる
- $\sum_{i=a}^b A_i = B_b - B_{a-1}$

累積和の具体例

	0	1	2	3	4	5	6
A		2	3	5	1	7	3
B	0	2	5	10	11	18	21

- 実際に累積和を取ってみた
- 例えば $B_4 = A_1 + A_2 + A_3 + A_4 = 2 + 3 + 5 + 1 = 11$

累積和の具体例

	0	1	2	3	4	5	6
A		2	3	5	1	7	3
B	0	2	5	10	11	18	21

- $A_2 \sim A_5$ の和は？
- $B_5 - B_1 = 18 - 2 = 16$
- 実際に計算してみると、 $3 + 5 + 1 + 7 = 16$ でちゃんと動作している

累積和の実装

- $B_i = B_{i-1} + A_i$ を用いることで $O(\text{要素数})$ で累積和を取ることができる
- この問題では累積和をそれぞれの日について取らないといけない(D 回累積和をとる)
- クエリ処理は $O(1)$

実装例(C++)

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int N, D;
int M[2000][2000];
int sum[2000][2000];
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return memo[day][coin];
    int tmp = (coin > 0 ? sum[day][coin-1] : 0);
    tmp -= sum[day][N-1] - (coin > 0 ? sum[day][coin-1] : 0);
    int ret = abs(tmp) + solve(day+1, coin);
    if(coin < N) ret = min(ret, solve(day, coin+1));
    return memo[day][coin] = ret;
}
signed main(){
    cin >> N >> D;
    for(int i=0; i<D; i++){
        for(int j=0; j<N; j++){
            cin >> M[i][j];
            memo[i][j] = -1;
            sum[i][j] = (j > 0 ? sum[i][j-1] : 0) + M[i][j];
        }
        memo[i][N] = -1;
    }
    cout << solve(0, 0) << endl;
}
```

実装例(C++)

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int N, D;
int M[2000][2000];
int sum[2000][2000];
int memo[2000][2001];
int solve(int day, int coin){
    if(day == D) return 0;
    if(memo[day][coin] != -1) return memo[day][coin];
    int tmp = (coin > 0 ? sum[day][coin-1] : 0);
    tmp -= sum[day][N-1] - (coin > 0 ? sum[day][coin-1] : 0);
    int ret = abs(tmp) + solve(day+1, coin);
    if(coin < N) ret = min(ret, solve(day, coin+1));
    return memo[day][coin] = ret;
}
signed main(){
    cin >> N >> D;
    for(int i=0; i<D; i++){
        for(int j=0; j<N; j++){
            cin >> M[i][j];
            memo[i][j] = -1;
            sum[i][j] = (j > 0 ? sum[i][j-1] : 0) + M[i][j];
        }
        memo[i][N] = -1;
    }
    cout << solve(0, 0) << endl;
}
```

時間計算量

- 入力を受け取るのに $O(DN)$
- メモ用配列を初期化するのに $O(DN)$
- 累積和を取るのに $O(DN)$
- メモ化再帰で表を埋めるのに $O(DN)$
 - 表1マスを埋めるのに $O(1)$
 - 累積和のクエリ処理は $O(1)$
- よって全体では $O(DN)$

備考

- 36枚目のスライドはJOI2013/2014春合宿かかしの解説スライドを~~パクリ~~ました参考にした
- <https://www.ioi-jp.org/camp/2014/2014-sp-tasks/2014-sp-d3-scarecrows-review.pdf>