力試し問題前半

（問題 1～10）

1 問題 C01：Tax Rate (Lv. 1)

消費税率は 10% ですので、税抜価格が N 円のときの税込価格は $1.1N$ 円です※1。したがって、`1.1 * N` を出力する以下のプログラムを提出すると、正解が得られます。

実装上の注意

C++ の場合、`(int)(1.1 * N)` のように整数型に変換してから出力しなければ不正解となる可能性があります。なぜなら、たとえば $N = 1000000$ のような大きな値の場合、`1.1e+06` のように指数表記で出力されるからです。

解答例（C++）

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

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

    // 出力
    cout << (int)(1.1 * N)  << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

※1　制約より N は 100 の倍数であるため、税込価格 $1.1N$ は必ず整数になります。

2 問題 C02：Two Balls (Lv. 10)

この問題は、**選ぶ 2 つのボールを全探索**することによって、計算量 $O(N^2)$ で解くことができます。

実装例を以下に示します。このプログラムでは、選ぶボールのうち 1 つ目の番号を i とし、2 つ目の番号を j としています。

解答例（C++）

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

int N, A[109];
int Answer = 0;

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

    // 答えを求める（全探索）
    for (int i = 1; i <= N; i++) {
        for (int j = i + 1; j <= N; j++) Answer = max(Answer, A[i] + A[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

想定される別解

実は、**「最も重いボール」と「2 番目に重いボール」を選ぶ**のが絶対に最適になります。

そのため、$A = [A_1, \ldots, A_N]$ を大きい順にソートし、$A_1 + A_2$ を出力するという方法もあります。計算量は $O(N \log N)$ であり、全探索よりも高速です。

3 問題 C03：Stock Queries (Lv. 20)

この問題は、次のようなアルゴリズムで解くことができます。

[手順1]

- まず、各 i に対して i 日目の株価 Price[i] を計算する。

[手順2]

- その後、それぞれの質問に答える。
- S_j, T_j 日目の株価のどちらが高いかは、Price[S[j]]<Price[T[j]] かどうかで判定できる。

ここで Price[i] の値は、以下の式にしたがって 1 日目から順番に計算すると求められます。計算方法が、累積和を高速に計算するとき（本の 54 ページ参照）とよく似ていますね。

- Price[1]=A[1]
- Price[i]=Price[i-1]+A[i] (i>=2)

※ここで、A[i] は i 日目の前日比

以上の内容を実装すると解答例のようになります。計算量は $O(N+Q)$ です。

解答例（C++）

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

// 入力で与えられる変数
int D, X, A[200009];
int Q, S[200009], T[200009];

// 各日の株価
int Price[200009];

int main() {
    // 入力
    cin >> D >> X;
    for (int i = 2; i <= D; i++) cin >> A[i];
```

```
    cin >> Q;
    for (int i = 1; i <= Q; i++) cin >> S[i] >> T[i];

    // 各日の株価を求める（累積和）
    Price[1] = X;
    for (int i = 2; i <= D; i++) Price[i] = Price[i - 1] + A[i];

    // 答えを出力する
    for (int i = 1; i <= Q; i++) {
        if (Price[S[i]] > Price[T[i]]) cout << S[i] << endl;
        else if (Price[S[i]] < Price[T[i]]) cout << T[i] << endl;
        else cout << "Same" << endl;
    }
    return 0;
}
```

※Python のコードはサポートページをご覧ください

4 問題 C04：Divisor Enumeration (Lv. 20)

まず考えられる方法は、「1 は N の約数か？」「2 は N の約数か？」といったように一つずつ調べていく方法です。しかし、計算量は $O(N)$ と遅く、本問題の制約では実行時間制限に間に合いません。

	約数
1	○
2	○
3	○
4	○
5	×
6	○
7	×
8	×
9	○

	約数
10	×
11	×
12	○
13	×
14	×
15	×
16	×
17	×
18	○

	約数
19	×
20	×
21	×
22	×
23	×
24	×
25	×
26	×
27	×

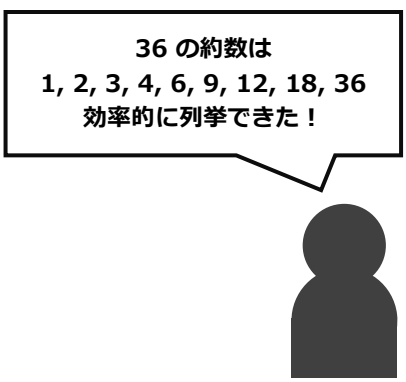
	約数
28	×
29	×
30	×
31	×
32	×
33	×
34	×
35	×
36	○

効率的な解法

実は、1 から N までを全部調べなくても、**1 から $\sqrt{N}$ までを調べれば、すべての約数を列挙することが可能です**。

なぜなら、整数 i が約数であると分かったら、整数 N/i も約数であることが分かるからです（$i \times (N/i) = N$ であるため）※2。たとえば $N = 36$ のとき、3 が約数とわかれば 12 も約数であると分かります。

	約数	
1	○	→ 36÷1=36 も約数！
2	○	→ 36÷2=18 も約数！
3	○	→ 36÷3=12 も約数！
4	○	→ 36÷4=9 も約数！
5	×	
6	○	→ 36÷6=6 も約数！



※2 もしかしたら、$\sqrt{N}$ まで調べても、i にも N/i にも含まれない約数があるのではないかと思う方もいるかもしれません。しかし、N のすべての「$\sqrt{N}$ を超える約数 A」について、$A \times B = N$ を満たす整数 B（$\sqrt{N}$ 以下）が存在するため、そんなことは絶対にあり得ません。たとえば $N = 36$ の場合、$9 \times 4 = 36$ より、約数 9 は 4 を調べる時点で見つかってしまいます

したがって、以下のようなプログラムにより、すべての約数を列挙することができます。計算量は $O(\sqrt{N})$ です。

なお、この問題では約数を小さい順に出力しなければ不正解となることに注意してください（そのため、21 行目ではソートを使っています）。

解答例（C++）

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

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

    // 約数を列挙
    vector<long long> Yakusuu;
    for (long long i = 1; i * i <= N; i++) {
        if (N % i == 0) {
                Yakusuu.push_back(i);
                if (i != N / i) Yakusuu.push_back(N / i);
        }
    }

    // 小さい順にソート
    sort(Yakusuu.begin(), Yakusuu.end());

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

※Python のコードはサポートページをご覧ください

5 問題 C05：Lucky Numbers (Lv. 30)

まず、N 番目のラッキー数は**「$N-1$ の 2 進法表記の 0,1 を 4,7 に書き換えた値」**となります。40 番目までを以下に示します※3。

N	N 番目のラッキー数	$N-1$ の 2 進法
1	4444444444	0000000000
2	4444444447	0000000001
3	4444444474	0000000010
4	4444444477	0000000011
5	4444444744	0000000100
6	4444444747	0000000101
7	4444444774	0000000110
8	4444444777	0000000111
9	4444447444	0000001000
10	4444447447	0000001001
11	4444447474	0000001010
12	4444447477	0000001011
13	4444447744	0000001100
14	4444447747	0000001101
15	4444447774	0000001110
16	4444447777	0000001111
17	4444474444	0000010000
18	4444474447	0000010001
19	4444474474	0000010010
20	4444474477	0000010011

N	N 番目のラッキー数	$N-1$ の 2 進法
21	4444474744	0000000100
22	4444474747	0000000101
23	4444474774	0000000110
24	4444474777	0000010111
25	4444477444	0000011000
26	4444477447	0000011001
27	4444477474	0000011010
28	4444477477	0000011011
29	4444477744	0000011100
30	4444477747	0000011101
31	4444477774	0000011110
32	4444477777	0000011111
33	4444744444	0000100000
34	4444744447	0000100001
35	4444744474	0000100010
36	4444744477	0000100011
37	4444744744	0000100100
38	4444744747	0000100101
39	4444744774	0000100110
40	4444744777	0000100111

したがって、次ページの解答例のような実装をすると、正解が得られます。10 進法を 2 進法に変換する方法が分からない方は、本の 36 ページに戻って復習しましょう。

※3 「なぜ 2 進法というアイデアがひらめくのか？」と思った方は、ラッキー数で使われる数が 4, 7 ではなく 0, 1 である場合に単純化して考えてみると良いでしょう

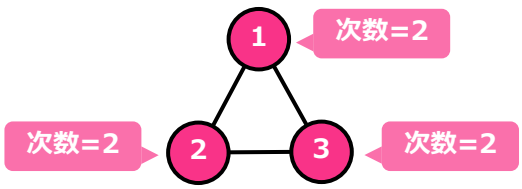
解答例（C++）

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

int main() {
    // 入力
    int N;
    cin >> N; N -= 1;

    // 出力
    for (int x = 9; x >= 0; x--) {
        int wari = (1 << x);
        if ((N / wari) % 2 == 0) cout << "4"; // 2 進法の 0 が 4 に対応
        if ((N / wari) % 2 == 1) cout << "7"; // 2 進法の 1 が 7 に対応
    }
    cout << endl;
    return 0;
}
```

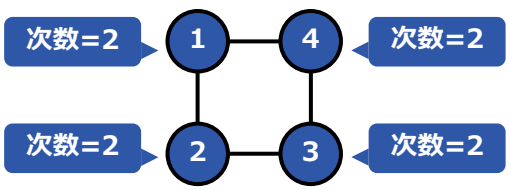
※Python のコードはサポートページをご覧ください

6 問題 C06：Regular Graph (Lv. 40)

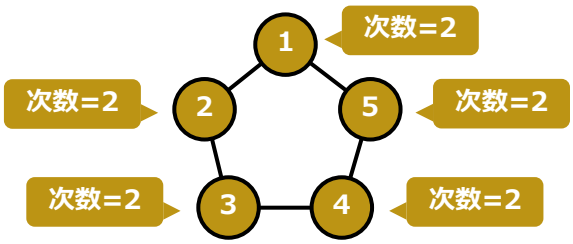
まずは $N = 3$ の場合を解いてみましょう。少し考察すると、すべての頂点が次数 2 のグラフとして、以下のものが見つかると思います。



次に $N = 4$ の場合を解いてみましょう。これも少し考察すると、すべての頂点が次数 2 のグラフとして、以下のものが見つかると思います。

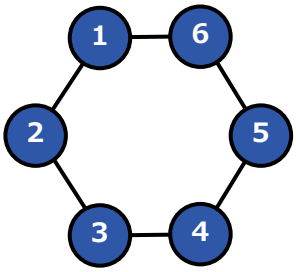


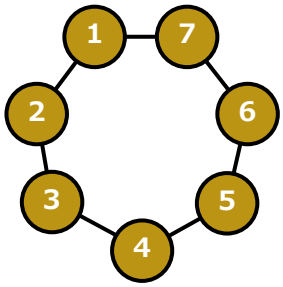
続いて $N = 5$ の場合を解いてみましょう。これはやや難しいですが、すべての頂点が次数 2 のグラフとして、以下のものが見つかると思います。

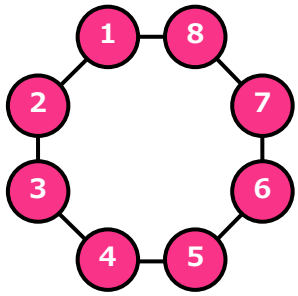


勘の良い人であれば、この時点で**「全頂点を一度ずつ通るループのような形にすれば、すべての頂点の次数が 2 になるのではないか」**という予測が立つのではないでしょうか。

実際、この予測は正しいです。したがって、ループのようなグラフを出力する次ページの解答例のようなプログラムを書くと、正解となります。







解答例（C++）

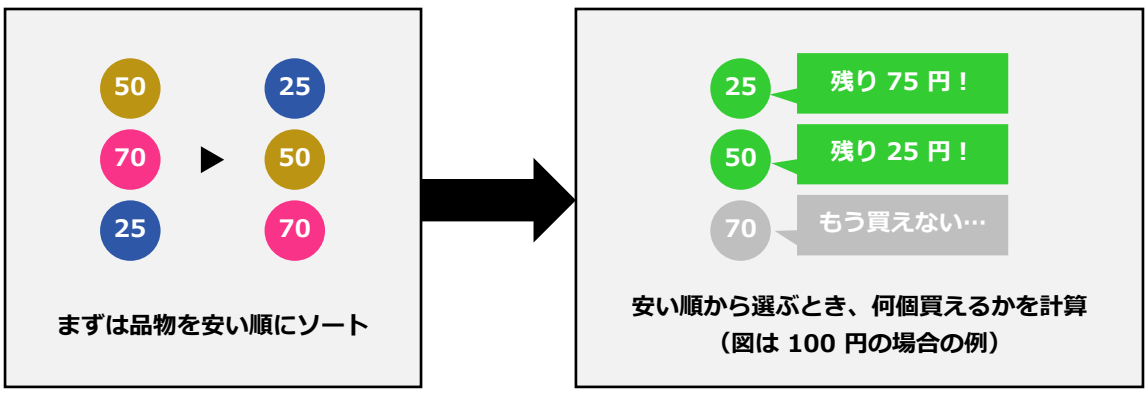
```cpp
#include <iostream>
using namespace std;

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

    // 出力
    cout << N << endl;
    for (int i = 1; i <= N; i++) {
        cout << i << " " << i % N + 1 << endl;
    }
    return 0;
}
```

※Python のコードはサポートページをご覧ください

7 問題 C07：ALGO-MARKET (Lv. 45)

まず、限られた値段の中で最も多くの品物を買う方法は、**「安い品物から順番に買っていくこと」**です。

そのため、あらかじめ品物を値段の安い順にソートしておき、その後各質問に対して「どこまで買えるか」を直接計算すると、正しい答えが分かります。



しかし、計算量は質問に答える部分がボトルネックとなって $O(NQ)$ となります。残念ながら実行時間制限に間に合いません。

効率的な解法

そこで、**i 個の品物を買うのに必要な最小の金 `Price[i]`** を前もって計算することを考えましょう。

> $A_1, \ldots, A_N$ が小さい順にソートされているとき、`Price[i]=A[1]+･･･+A[i]` となるため、`Price[i]` の値は累積和を求めるとき（本の 54 ページ）と同じようにして計算することができます。

すると、`X[j]` 円以内で買える品物の個数の最大値は、**`Price[i]<=X[j]` を満たす最大の `i`** となります。そしてこの値は、配列の二分探索によって計算量 $O(\log N)$ で計算することができます。

以上の考察により、アルゴリズム全体の計算量が $O((N+Q)\log N)$ まで削減されました。本問題の制約の上限値である $N, Q = 100000$ でも、もちろん実行時間制限に間に合います。

解答例（C++）

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

long long N, C[100009], S[100009];
long long Q, X[100009];

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

    // C[i] を小さい順にソート
    sort(C + 1, C + N + 1);

    // 累積和 S[i] をとる
    // S[i] は「i 個の品物を買うときの最小金額」
    S[0] = 0;
    for (int i = 1; i <= N; i++) S[i] = S[i - 1] + C[i];

    // 質問に答える
    for (int i = 1; i <= Q; i++) {
        int pos = upper_bound(S, S + N + 1, X[i]) - S;
        cout << pos - 1 << endl;
    }
    return 0;
}
```

※Python のコードはサポートページをご覧ください

8 問題 C08：ALGO4 (Lv. 50)

この問題は**全探索**によって解くことができます。具体的には、0000～9999 の全パターンについて、「もし当選番号が△△△△ならば抽選券 N 枚の等級はすべて正しいか？」ということを調べれば良いです。

実装は少し大変ですが、C++ の場合は数値を文字列に変換する関数 to_string などを使えば少し簡単になります。

解答例（C++）

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

int N;
string S[1009]; int T[1009];

// 当選番号が A2 のとき、A1 は何等かを返す
int Hantei(string A1, string A2) {
    int Diff = 0;
    for (int i = 0; i < 4; i++) {
        if (A1[i] != A2[i]) Diff += 1;
    }
    if (Diff == 0) return 1; // 全く同じとき 1 等
    if (Diff == 1) return 2; // 桁が 1 つだけ違うとき 2 等
    return 3;
}

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

    // 全探索
    vector<string> Answer;
    for (int num = 0; num <= 9999; num++) {
        // 整数 num を 4 桁の文字列に置き換える
        string ID = to_string(num);
        while (ID.size() < 4) ID = "0" + ID;

        // すべての情報が正しいかどうかを確認
        bool flag = true;
```

```
        for (int i = 1; i <= N; i++) {
                if (Hantei(S[i], ID) != T[i]) flag = false;
        }

        // もしすべての情報が正しかった場合
        if (flag == true) {
                Answer.push_back(ID);
        }
    }

    // 出力
    if (Answer.size() != 1) {
        cout << "Can't Solve" << endl;
    }
    else {
        cout << Answer[0] << endl;
    }
    return 0;
}
```

※Python のコードはサポートページをご覧ください

9 問題 C09：Taro’s Vacation (Lv. 55)

まず考えられる解法は、どの日に勉強するかを全探索することです。もちろんこの解法でも最終的には正しい答えを出すことができます。しかし、全部で 2^N 通りを探索することになるため、計算量の面で絶望的です。

そこで、以下のような動的計画法を考えます。前の日に勉強したかどうかで場合分けするために、配列を 2 つ持つというアイデアが基本です。

管理する配列

- `dp1[i]`：`i` 日目に勉強する場合の、`i` 日目までの実力アップの最大値
- `dp2[i]`：`i` 日目に勉強しない場合の、`i` 日目までの実力アップの最大値

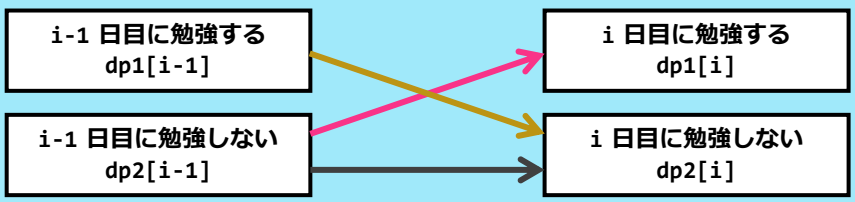
初期状態

1 日目が始まる前の時点では、何も実力が上がっていないため、`dp1[0]=0` および `dp2[0]=0` となる。

状態遷移

状態遷移のイメージは、下図に示すとおりである。

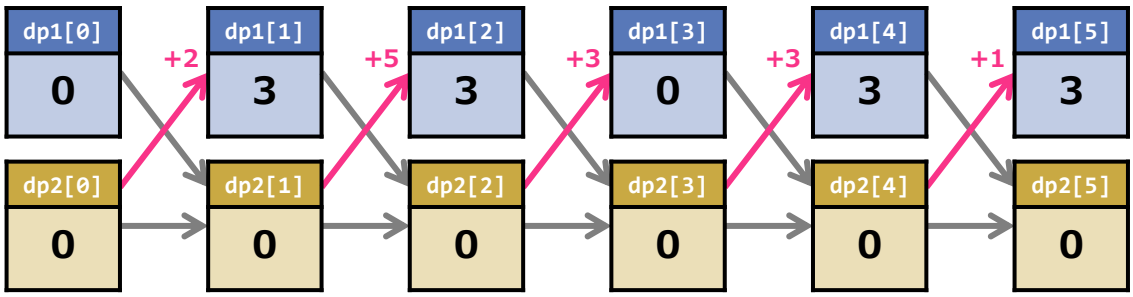
- `dp1[i]=dp2[i-1]+A[i]`
- `dp2[i]=max(dp1[i-1], dp2[i-1])`



求める答え

`max(dp1[N], dp2[N])`

たとえば、$N = 5, (A_1, A_2, A_3, A_4, A_5) = (2, 5, 3, 3, 1)$ の場合、dp1[i] および dp2[i] の値は以下のようになります。



ここまでの解法を実装すると、以下のようになります。計算量は $O(N)$ と高速です。なお、本問題の制約は $A_i \leq 10^9$ と大きく、答えが最大で 10^{14} を超える可能性もあるため、C++ の場合は long long 型などの 64 ビット整数を利用する必要があることに注意してください（注：Python の場合は関係ありません）。

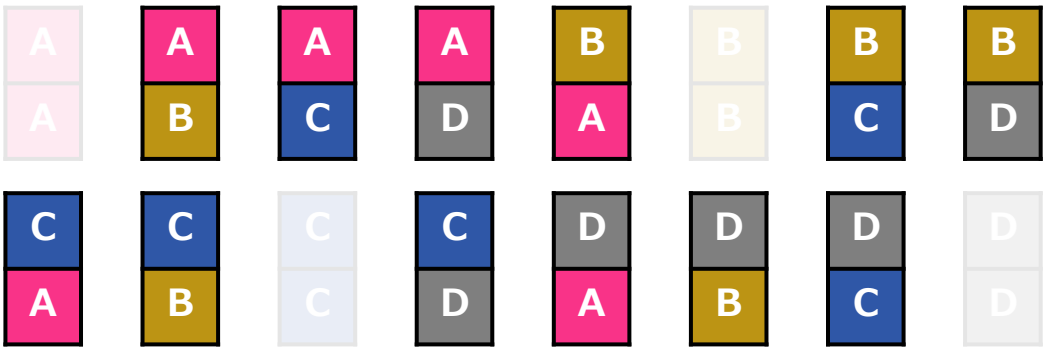
解答例（C++）

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

long long N, A[500009];
long long dp1[500009], dp2[500009];

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

    // 配列の初期化
    dp1[0] = 0;
    dp2[0] = 0;

    // 動的計画法
    for (int i = 1; i <= N; i++) {
        dp1[i] = dp2[i - 1] + A[i];
        dp2[i] = max(dp1[i - 1], dp2[i - 1]);
    }

    // 答えを出力
    cout << max(dp1[N], dp2[N]) << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください

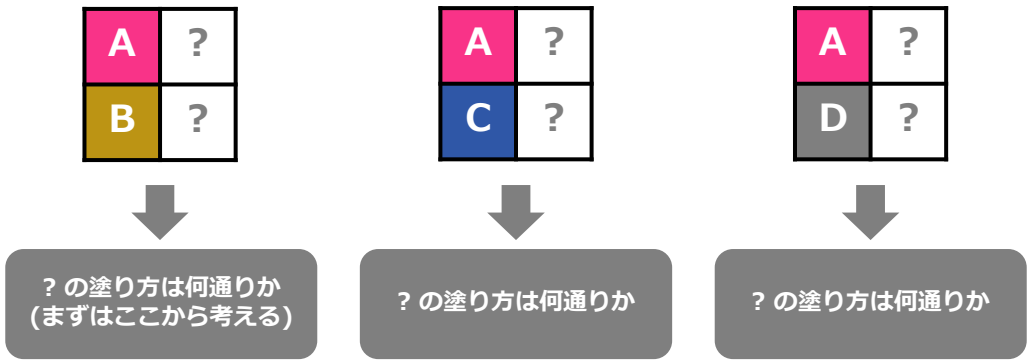
10 問題 C10：A Long Grid (Lv. 65)

まずは $W = 1$ の場合を考えましょう。マス目に色を塗る方法は全部で $4 \times 4 = 16$ 通りありますが、その中の 4 通りは「隣接する 2 マスが同じ色」になってしまうので、条件を満たす塗り方は **12 通り**です。

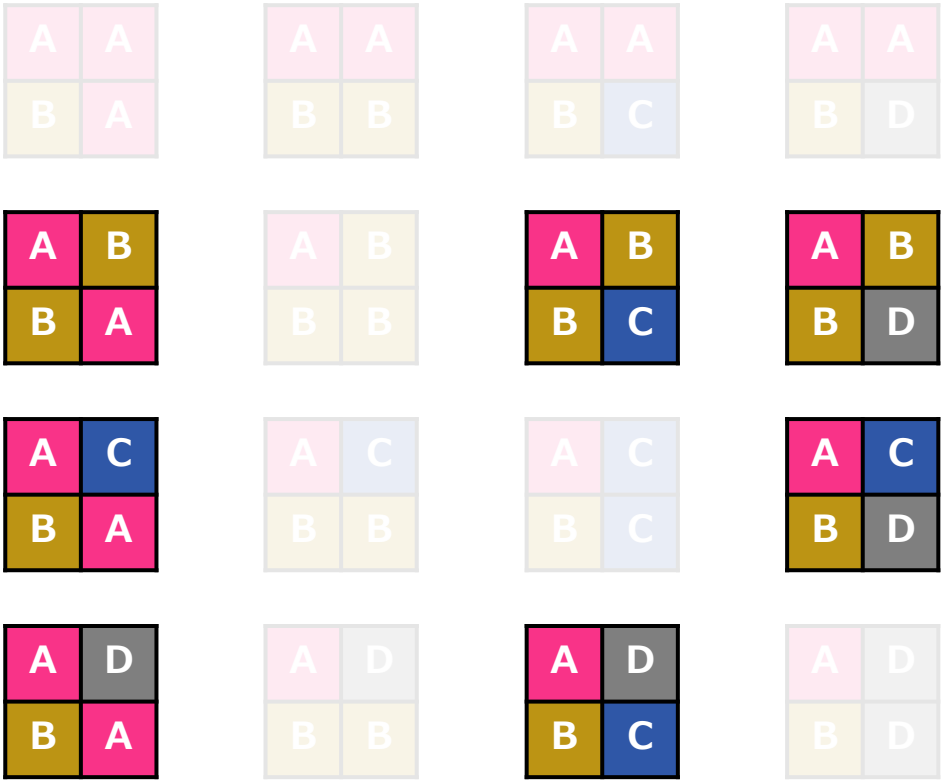


$W = 2$ の場合を考える

次に $W = 2$ の場合ですが、「どの隣接する 2 マスも同じ色にならないように 2 列目を塗る方法の数」は 1 列目の塗り方に依存します。そこで最初に、1 列目の塗り方が「A・B」の場合から考えましょう。



次ページの図に示す通り、1 列目が「A・B」のときの 2 列目の塗り方は全部で **7 通り**あります。



最初の列が「A・B」以外の場合

それでは、最初の列が「A・B」以外の 11 通りについてはどうでしょうか。1 列目の 2 マスの色が異なるという点で「A・B」の場合と一致しているので、**実質的には「A・B」の場合と同じように考える**ことができます。

したがって、全パターンについて 2 列目の塗り方は 7 通りです。このことから、$W = 2$ のときの答えが **$12 \times 7 = 84$ 通り**であると分かります。

W が 3 以上の場合は？

さて、$W = 3$ の場合はどうでしょうか。3 列目の塗り方は 2 列目の塗り方に依存しますが、先程述べたように、**前の列の 2 マスが異なる色の場合、その列の塗り方は必ず 7 通りだけあります。**したがって、$W = 3$ のときの答えは $84 \times 7 = 588$ 通りです。

$W \geq 4$ の場合も同様に考えると、答えが **$12 \times 7^{W-1}$ 通り**であると分かります。本問題の制約は $W \leq 10^{18}$ と非常に大きいので、答えを直接計算すると実行時間制限に間に合いませんが、繰り返し二乗法（本の 170 ページ）を使って計算すると、計算量が $O(\log W)$ となり効率的です。

解答例（C++）

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

// a の b 乗を m で割った余りを返す関数
long long Power(long long a, long long b, long long m) {
    long long p = a, Answer = 1;
    for (int i = 0; i < 60; i++) {
        long long wari = (1LL << i);
        if ((b / wari) % 2 == 1) {
                Answer = (Answer * p) % m;
        }
        p = (p * p) % m;
    }
    return Answer;
}

int main() {
    // 入力
    const long long mod = 1000000007;
    long long W;
    cin >> W;

    // 出力
    cout << 12LL * Power(7, W - 1, mod) % mod << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください