HCPC 勉強会（2017/09/26）

「繰り返し 2 乗法（バイナリ法）」

北海道大学工学部 情報エレクトロニクス学科
情報理工学コース B4 杉江 祐哉

繰り返し 2 乗法とは？

- 繰り返し 2 乗法とは、**指数部を 2 冪の和に分解**することで計算を高速化する手法のこと
- たとえば、 a^x を計算したいときはどうする？
- 普通にやると、 1 に a を x 回かける必要があるが、今回の手法を使うと $\log x$ 回で済むようになる！

繰り返し 2 乗法とは？

- $x = 2^k$（指数が 2 のべき乗）のときは、2 乗の計算を k 回することで簡単に求まる
- a^{2^i} を順次求めながら、必要なものを答えに反映させていけば高速に計算できる
- 例：$3^{22} = 3^{16} \times 3^4 \times 3^2$

$$(3^{22} = 3^{(2^4)} \times 3^{(2^2)} \times 3^{(2^1)})$$

2 進数への招待

- 必要なものを答えに反映させていけば、とあったが、「必要なもの」がどれなのか知りたい
- 指数を 2 進数で表してみると、ここで言う「必要なもの」とは、**ビットが立っているもの**である
- さっきの例だと、 $22_{(10)} = 10110_{(2)}$
- 2 進数はビット演算と相性が良く、扱いやすい！

2 進数への招待

- 2 進数はビット演算と相性が良く、扱いやすい！

```
#include <cstdio>
int main() {
    int N, i; scanf("%d%d", &N, &i);
    // 数 N の i ビット目 (0-indexed) が立っている時に OK を返す
    // N を右に i ビットシフトさせて、1 との論理積をとっている
    // → i ビット目が 1 であれば、1 & 1 -> 1 が返るため true
    if(N >> i & 1) printf("OK\n");
    else printf("NG\n");
    return 0;
}
```

繰り返し 2 乗法（手順おさらい）

- 指数部を 2 進数として見る (2 冪の和に分解するため）
- 指数部の i ビット目が立っているならば、対応する 2 冪乗の値を答えに掛け合わせる
- ビット操作に慣れる必要がある

繰り返し 2 乗法（実装例）

```
// n^k mod p を計算する
ll mod_pow(int n, int k, int p) {
    // ret := 答え、mul := n の 2 冪乗 (最初は 2^0 乗)
    ll ret = 1, mul = n;
    for(int i=0; i<31; i++) {
        // ビットが立っているならば、mul を掛け合わせる
        if(k >> i & 1) ret = (ret * mul) % p;
        // 2 冪乗を作る (n の 2^x 乗 -> n の 2^(x+1) 乗)
        mul = (mul * mul) % p;
    }
    return ret;
}

int main() {
    int N, k; scanf("%d%d", &N, &k);
    printf("%lld\n", mod_pow(N, k, MOD));
    return 0;
}
```

問題集

- **Power**（AOJ NTL_1_B)
  今の話そのままです。
- **累乗の加算**（yukicoder No. 16)
  上ができればたぶんできます。
- **掛け算**（ARC051 C)
  初心者向けではない（普通に難しい）けど、面白い問題。
- **2^2^2** (yukicoder No. 403)
  これも初心者向けではない。 $A^{(B \wedge C)}$ の計算をどうするかがカギ。

繰り返し二乗法の応用

- 今まで説明してきた手法の応用として、２つ紹介します。
  - **行列累乗**（$N \times N$ 行列を k 乗する）
  - **ダブリング**（2 冪の和に分解して k 個先の状態を高速に求める。ある要素のすぐ次は容易にわかるが k 個先を求めるのに時間をかけられない時に有効）

行列累乗

- やりたいこと：$N \times N$ 行列を k 乗したい
- 積の計算時に、各要素で $O(N)$ の計算が発生
- よって、行列積の計算は $O(N^3)$
- 繰り返し二乗法を使うことによって、行列累乗を $O(N^3 logk)$ で処理できる
- 発展 :Kitamasa 法（今回の内容を逸脱するため省略）

行列累乗（実装例）

```cpp
template <typename T>
using Matrix = vector< vector<T> >;

template <typename T>
void init_mat(Matrix<T> &A, int h, int w) {
    A.resize(h, vector<T>(w, 0));
}

template <typename T>
Matrix<T> calc_mat(Matrix<T> A, Matrix<T> B) {
    Matrix<T> C(A.size(), vector<T>(B[0].size()));
    for(int i=0; i<A.size(); i++) {
        for(int k=0; k<B.size(); k++) {
            for(int j=0; j<B[0].size(); j++) {
                // C[i][j] += A[i][k] * B[k][j]; // modなし
                C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD; // modあり
            }
        }
    }
    return C;
}
```

```cpp
template <typename T>
Matrix<T> mat_pow(Matrix<T> A, ll n) {
    Matrix<T> B(A.size(), vector<T>(A.size()));
    for(int i=0; i<A.size(); i++) B[i][i] = 1;
    while(n > 0) {
        if(n & 1) B = calc_mat(B, A);
        A = calc_mat(A, A);
        n >>= 1;
    }
    return B;
}

// Matrix Library End
```

問題集

- **Blocks** (POJ No. 3734)
- **Matrix Power Series** (POJ No. 3233)

  この 2 つは蟻本に載っています。 2 つめは行列の累乗和を求める問題です。
- **漸化式** (ABC009 D)

  ちょっと変わった行列累乗。演算子に注意です。
- **ChristmasBinaryTree** (TopCoder SRM 705 Div.2 Hard)

  木と行列累乗が融合した、個人的に思い出深い問題。

ダブリング

- やりたいこと：k 個先の状態を高速に求めたい
- これを実現させるには、各要素から 2^i 個先の要素が何であるかを配列に記憶させれば良い
- あとは、k を 2 冪の和に分解して、上で作った配列を元に現在の状態からうつっていけば良い
- 計算量は配列構築 $O(N \log N)$、クエリ $O(\log k)$

ダブリング（配列の構築）

- 用意すべきはこのような配列。先程述べたように、ある要素のすぐ隣は既知（= 既に配列に値が入っている）であるものとする

要素の値

2^i個先

	3	10	1	4	9
0	10	1	4	9	-1
1					
2					

ダブリング（配列の構築）

- 次の行を埋める際には、今までの結果が利用できる
- (例: 2^i 個先の要素 → 2^{i-1} 個先の要素の 2^{i-1} 個先の要素)

要素の値

2^i 個先					
	3	10	1	4	9
0	10	1	4	9	-1
1	**1**	**要素 3 の 2^1 個先 → (3 の 2^0 個先) の 2^0 個先**			
2					

ダブリング（配列の構築）

- 次の行を埋める際には、今までの結果が利用できる
- (例 : 2^i 個先の要素 → 2^{i-1} 個先の要素の 2^{i-1} 個先の要素)

要素の値

2^i 個先

	3	10	1	4	9
0	10	1	4	9	-1
1	**1**	**4**	**9**	**-1**	**-1**
2					

ダブリング（配列の構築）

- 次の行を埋める際には、今までの結果が利用できる
- (例 : 2^i 個先の要素 → 2^{i-1} 個先の要素の 2^{i-1} 個先の要素)

要素の値

2^i 個先

	3	10	1	4	9
0	10	1	4	9	-1
1	1	4	9	-1	-1
2	**9**				

要素 3 の 2^2 個先 → (3 の 2^1 個先) の 2^1 個先

ダブリング（クエリ処理）

- 2 冪の和にしてたどっていけば良い
- (例：要素 3 から 3 つ先 → $2^0 + 2^1$ 個先)

要素の値

2^i 個先

	3	10	1	4	9
0	10	1	4	9	-1
1	1	4	9	-1	-1
2	9				

要素 3 の 3 個先 → (3 の 2^1 個先) の 2^0 個先

問題集

- **LCA: Lowest Common Ancestor** (AOJ GRL_5_C)

  蟻本 P.292 にも載っています。

- **阿弥陀** (ABC013 D)
- **高橋君とホテル** (ARC060 E)

  この 2 題を解くとダブリングの威力が大体分かると思います。

- **Friends and Subsequences** (Codeforces #361 Div.2 D)

  Sparse Table (ダブリングに似たことをするデータ構造。途中で数列が変わらない、かつクエリが多い場合に有効) でも解けます

まとめ

- 繰り返し二乗法（バイナリ法）とは、2冪の和に分解して処理を高速にする方法である
- ビット操作をよく使うため、そこの慣れが必要
- 行列に適用したり、先の状態を高速に求めたりなど、様々な形で応用が可能である