Три алгоритма для вычисления чисел Фибоначчи

Обозначения и основные идеи взяты из книги [1].

Последовательность чисел Фибоначчи F_n определяется рекуррентным соотношением

$$F_n = F_{n-1} + F_{n-2} \qquad (n > 1)$$

и начальными условиями $F_0 = 0$, $F_1 = 1$ (иногда полагают $F_0 = 1$, но это менее удобно).

Хорошо известна *формула Бине* (Даниэль Бернулли, 1728; Жак Бине, 1843):

$$F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} \approx \frac{\phi^n}{\sqrt{5}}.$$

Здесь $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$ — «золотое сечение», $\hat{\phi} = 1 - \phi \approx -0.61803$. Формула Бине показывает, что F_n растёт экспоненциально (со скоростью геометрической прогрессии).

Опишем несколько способов вычисления F_n. Формулу Бине использовать не будем, так как при больших n возникают сложные вопросы, связанные с накоплением вычислительных погрешностей.

Программы будут записаны на языке C (см. [2]). Честно говоря, для больших n типа `int` недостаточно и нужно использовать числа произвольной длины. Но здесь, ради простоты, будем рассматривать лишь такие n, для которых F_n помещается в `int`.

1. Рекурсия

Сразу из определения получается рекурсивный алгоритм:

```
int fib(int n) {
    return (n > 1) ? (fib(n - 1) + fib(n - 2)) : n;
}
```

Но он работает слишком долго при больших n. С помощью математической индукции легко доказать, что для этого алгоритма время работы и объём используемой памяти (стека) пропорциональны F_{n+1}, т. е. растут экспоненциально при увеличении n.

2. Последовательное вычисление (динамическое программирование)

Значения F_n можно вычислять последовательно, сохраняя значение в массиве:

```
int fib(int n) {
    int result, i;
    int *f = (int*) malloc((n + 2) * sizeof(int));
    f[0] = 0; f[1] = 1;
    for (i = 2; i <= n; ++i)
        f[i] = f[i - 1] + f[i - 2];
    result = f[n];
    free(f);
    return result;
}
```

Конечно, здесь можно обойтись и без массива, так как для вычисления F_n используются лишь два предыдущих элемента последовательности:

```
int fib(int n) {
    if (n <= 1) return n;
    int fprev = 0, fcur = 1, fnext, i;
    for (i = 2; i <= n; ++i) {
        fnext = fcur + fprev;
        fprev = fcur; fcur = fnext;
    }
    return fcur;
}
```

Для этого алгоритма время работы линейно зависит от n.

3. Быстрый алгоритм (через степень матрицы)

Предыдущий способ показывает, что элементы последовательности Фибоначчи удобно хранить и вычислять парами. Если известна пара элементов (F_{n-1}, F_n), то следующую пару (F_n, F_{n+1}) можно вычислить так:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n-1} + F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix}$$

Обозначим матрицу $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ через M. С помощью математической индукции легко доказать, что

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = M^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} = M^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Вспомнив правило умножения матриц, можно сделать такой вывод: F_n есть правый верхний элемент матрицы M^n.

Теперь напомним быстрый алгоритм для вычисления степени. Если число n записывается в двоичной системе как $(a_m \dots a_1 a_0)_2$, то M^n можно найти как произведение всех степеней вида M^{2^k}, где $a_k = 1$. Например,

$$13 = 8 + 4 + 1 = (1101)_2, \qquad M^{13} = M^8 \cdot M^4 \cdot M^1 = M^{2^3} \cdot M^{2^2} \cdot M^{2^0}.$$

Матрицы вида M^{2^k} легко считать по очереди, путём возведения в квадрат.

Сначала напишем функцию для перемножения матриц размера 2×2. Будем хранить такие матрицы как массивы длины 4.

```
// Функция mul умножает матрицу dest на матрицу src
void mul(int *dest, const int* src) {
    int r0, r1, r2, r3;
    r0 = dest[0] * src[0] + dest[1] * src[2];
    r1 = dest[0] * src[1] + dest[1] * src[3];
    r2 = dest[2] * src[0] + dest[3] * src[2];
    r3 = dest[2] * src[1] + dest[3] * src[3];
    dest[0] = r0; dest[1] = r1;
    dest[2] = r2; dest[3] = r3;
}
```

С использованием функции `mul` уже нетрудно записать алгоритм вычисления F_n, работающий за время порядка $\log n$:

```
int fib(int n) {
    int Mpower[] = { 0, 1, 1, 1}; // = M
    int Mresult[] = { 1, 0, 0, 1}; // = E
    while (n) {
        // Сейчас Mpower есть M^{2^k},
        // где k - номер итерации цикла (начиная с 0).
        // Если двоичная запись n заканчивается на 1,
        // то умножаем Mresult на Mpower:
        if (n & 1) mul(Mresult, Mpower);
        // Возводим Mpower в квадрат:
        mul(Mpower, Mpower);
        // Убираем младшую двоичную цифру числа n:
        n >>= 1;
    }
    // Теперь Mresult есть искомая матрица M^n,
    // и можно вернуть её правый верхний элемент:
    return Mresult[1];
}
```

4. Заключение. Сходство с другими задачами

Мы рассмотрели 3 алгоритмы для нахождения чисел Фибоначчи:

1) рекурсивный (очень лаконичный, но очень долго работает);
2) динамическое программирование (несложно получить из рекуррентных соотношений, работает довольно быстро);
3) очень быстрый хитроумный алгоритм, который можно придумать лишь после глубокого постижения сути задачи.

Аналогично обстоят дела с некоторыми другими задачами:

1) вычисление чисел Каталана;
2) сравнение строки с шаблоном, содержащим звёздочки;
3) поиск кратчайших путей в графе.

Список литературы

[1] **Грэхем Р., Кнут Д., Паташник О.** Конкретная математика. Основание информатики: Пер. с англ. — М.: Мир, 1998. — 703 с., ил.

[2] **Kernighan B., Ritchie D.** The C programming language. Second edition. — AT&T Bell Laboratories. Murray Hill, New Jersey, 1988. — 272 p.