Алгоритмы и структуры данных. Динамическое программирование

densine

22 ноября 2025 г.

Содержание

1. Примеры

1.1. Числа Фибоначчи

$$f_i = f_{i-1} + f_{i-2}$$

$$f_0 = 0, f_1 = 1$$

Код на C++ без рекурсии с вектором:

```cpp
vector<int> f;
f[0] = 0;
f[1] = 1;
for (int i = 2; i <= n; i++) {
    f[i] = f[i - 1] + f[i - 2];
}
```

Код на C++ без рекурсии и без вектора:

```
int f_p = 1;
int f_pp = 0;
int f;

for (int i = 2; i <= n; i++) {
    f = f_p + f_pp;
    f_pp = f_p;
    f_p = f;
}
```

1.2. Кузнечик

Суть задачи – кузнечик (зайчик) находится в самой левой клетке числовой прямой. Он может передвигаться на 1 или 2 клетки вперед и только вперед. Сколько у него способов добраться до n клетки?

Пусть k_i – количество попасть в клетку i. Тогда мы можем обозначить $k_1 = 1, k_2 = 1$. Затем, для i клетки можно добраться из предыдущей и предпредыщей клетки, т.е:

$$k_i = k_{i-1} + k_{i-2}$$

Знакомая формула, правда? ;)

1.3. Закрытые клетки

Теперь наш кузнечик научился прыгать еще и на 3 клетки вперед, но в некоторые клетки попасть нельзя – они заблокированы. Для заблокированных клеток $k_i = 0$. Тогда формула остается прежней!

$$k_i = k_{i-1} + k_{i-2} + k_{i-3}$$

Тогда просто не считаем k_i для заблокириванных клеток. Код на C++:

```
vector<int> k(n + 1);
k[1] = 1;
for (int i = 2; i <= n; i++) {
    if (blocked[i]) continue;
    k[i] += k[i - 1];
    if (i >= 2) k[i] += k[i - 2];
    if (i >= 3) k[i] += k[i - 3];
}

cout << k[n] << endl;
```

Или, если сделать сдвиг по индексам:

```
vector<int> k(n + 2);
k[0] = 0;
k[1] = 0;
k[2] = 1;

for (int i = 3; i <= n + 1; i++) {
```

```
    if (blocked[i]) continue;
    k[i] = k[i - 1] + k[i - 2] + k[i - 3];
}

cout << k[n + 1] << endl;
```

1.4. Минимальный путь

Та же задача, но теперь каждая клетка "стоит" какое-то количество денег. Какое минимальное количество денег можно потратить?

Пусть $dp[i]$ – минимальная стоимость пути в i клетку. Тогда:

$$dp_i = min(dp_{i-1}, dp_{i-2}) + cost_i$$

Причем из значений в dp можно восстановить минимальный путь. Просто начинаем с конца и выбираем минимальную из 2 предыдущих клеток. Если они равны, то неважно какую выберем (следует из определения dp). Код на C++ для восстановления пути:

```
vector<int> way;
int i = n;
while (i > 1) {
    way.push_back(i);
    if (dp[i - 1] < d[i - 2]) i--;
    else i -= 2;
}

way.push_back(i);
reverse(way.begin(), way.end());
// way - ответ
```

2. Собственно, динамическое программирование

Обобщим подход, применненый нами в предыдущих задачах. Какие вопросы нужно задать себе?

1. Состояние
   dp_i – это ... (дать определение, что значит dp_i?)
2. База (стартовые значения)
3. Переход (формула, правило)
   Например $dp_i = dp_{i-1} + dp_{i-2}$.
4. Порядок обхода
   - ДП назад – то, что мы писали;
   - ДП вперед – ...;
   - Псевдорекурсия – рекурсия с меморизацией.
5. Ответ (где он лежит, как он связан с dp, как его в целом вычислить?)
6. Восстановление (как восстановить путь в оптимальное состояние?)

2.1. Без двух 0 подряд

Сколько существует бинарных строк длины n, где нет двух 0 подряд?

1. $dp[i][1/0]$ – количество строк длины i, оканчивающихся на 1 или 0.
2. $dp[1][1] = 1$, $dp[1][0] = 1$.
3. $dp[i][0] = dp[i-1][1]; dp[i][1] = dp[i-1][0] + dp[i-1][1]$.
4. От 1 до n
5. $dp[n][0] + dp[n][1]$

2.2. Калькулятор

Имеется калькулятор, который может совершать 3 действия: прибавить 1, умножить на 2 и умножить на 3. За сколько действий можно получить x, если изначальное число – 1?

1. Состояние – dp_i – количество действий для получения i.
2. $dp_i = 0$
3. Переход... Т.к. заданы лишь 3 операции, из текущего i можно получить лишь $i+1; 2 \cdot i, 3 \cdot i$. Тогда переход выглядит так (если в ячейку уже записано значение, сравниваем с тем, что уже в ячейке, и выбираем минимальное):

$$dp[i+1] \leq dp_i + 1$$

$$dp[2 \cdot i] \leq dp_i + 1$$

$$dp[3 \cdot i] \leq dp_i + 1$$

4. ДП вперед!
5. dp_i
6. Либо проверяем $x-1; x/2; x/3$,

Код на C++:

```
vector<int> dp(n + 1, 1e9 + 1);
dp[1] = 0;
for (int i = 1; i <= n; i++) {
    dp[i + 1] = min(dp[i + 1], dp[i] + 1);
    dp[2 · i] = min(dp[2 · i], dp[i] + 1);
    dp[3 · i] = min(dp[3 · i], dp[i] + 1);
}
```

Подход, можно сказать, другой. Это называют "динамическим программированием вперед".

2.3. Черепашка

Дано поле $n \cdot m$. В левом вернем углу находится черепашка. Сколькими способами черепашка может добраться до правого нижнего угла, если она может передвигаться только вниз и вправо.

1. dp_{ij} – количество способов попасть в (i, j).
2. $dp_{1j} = dp_{i1} = 1$.
3. $dp_{ij} = dp_{i-1j} + dp_{ij-1}$.
4. ДП назад – для следующей клетки необходимо, чтобы предыдущие значения были посчитаны.
5. dp_{nm} – это ответ.

Код на C++:

```
vector<vector<int>> dp(n, vector<int>(m, 1));
for (int i = 2; i <= n; i++) {
    for (int j = 2; j <= m; j++) {
        dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
    }
}
```

2.4. Закрытые клетки

То же самое... Но есть клетки, на которые наступать нельзя. Тут уж справимся сами ;). Только есть заметка – добавляем один ряд и столбец сверху и слева соответственно, и инициализируем $dp[1][1] = 1$, а все остальное нулями.

```
vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
dp[1][1] = 1; // (2)
// Либо dp[1][0] = 1 (dp[0][1] = 1) (1)
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        // Но тогда не нужно проверять i = 1, j = 1 (1)
        // Чтобы не перезаписать (1, 1)
        if (i == 1 && j == 1) continue; // (2)
        dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
    }
}
```

2.5. Минимальная стоимость

(было утверждено, что мы можем сами без проблем и сложностей это написать со знанием всего предыдущего... ;))

2.6. Коняшка

Дано поле $n \cdot m$. В клетке $(1, 1)$ стоит конь из шахмат, но ходить он может только по $(i-1, j+2); (i+1; j+2); (i+2; j+1); (i+2; j-1)$. Сколько способов у него способов попасть в (n, m)?

Чем это отличается от черепашки? Если мы будем линейно проходиться по рядам, в какой-то момент окажется, что не все маршруты просчитаны для какой-то клетки. Нужно поменять обход... Ходим диагоналями из верхнего ряда влево вниз (кода не будет... задачка повышенной сложности ;)).