Рассмотрим ещё одну задачу, на первый взгляд, не похожую на предыдущую.

Задача про жучка

Имеется прямоугольное поле размера $n \times m$. Поле разделено на единичные клетки. В левой верхней клетке сидит жучок. Он может перемещаться по полю ходами вправо или вниз на одну клетку. Для каждой клетки задано целое положительное число — количество бонусов, которое получит жучок при проходе через эту клетку. Нужно добраться в правую нижнюю клетку, собрав как можно больше бонусов. Бонусы в начальной и конечной клетках жучок тоже собирает (см. пример на рис. 1).

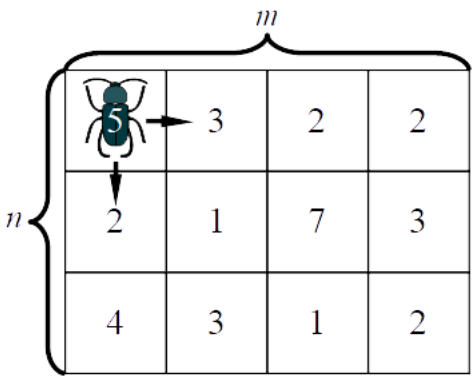

Рис. 1. Пример к задаче про жучка

Оптимальный путь жучка для примера на рис. 1 представлен на рис. 2. Можно посчитать, что по пути жучок соберёт 22 бонуса.

Пусть бонусы заданы в двумерном массиве или векторе векторов a, число a_{ij} равно количеству бонусов в клетке с координатами (i, j). В массиве 0-индексация: номер строки i принимает значения от 0 до $(n-1)$, номер столбца j — от 0 до $(m-1)$. Координаты $(0, 0)$ соответствуют начальной клетке, $(n-1, m-1)$ — конечной.

Будем решать задачу методом динамического программирования. Для этого нужно выделить какие-то однотипные подзадачи, чтобы решение каждой подзадачи выражалось через меньшие подзадачи.

Подзадача будет следующая: набрать наибольшую сумму бонусов, дойдя до клетки с координатами (i, j) (см. рис. 3). Будем считать ответ для каждой клетки на поле. В прошлой задаче про замощение доминошками подзадача

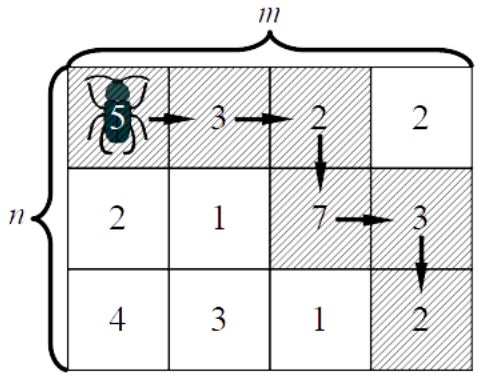

Рис. 2. Оптимальный путь

описывалась одним числом — размером поля. В данном случае подзадача описывается двумя числами — координатами клетки i и j. Эта пара (i, j) будет называться *состоянием динамики*. Для каждого состояния посчитаем d_{ij} — наибольшую сумму бонусов, которые наберёт жучок, дойдя до этой клетки.

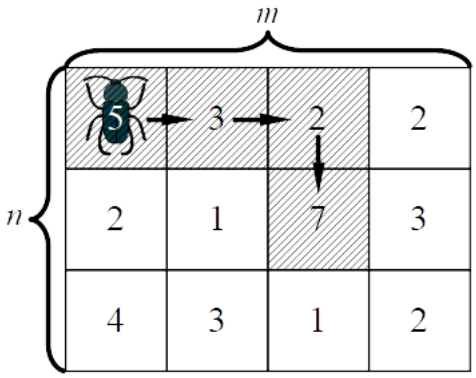

Рис. 3. Состояние динамики

Как посчитать d_{ij} через предыдущие состояния? В состояние (i, j) жучок может прийти из состояния $(i-1, j)$ или из состояния $(i, j-1)$ (см. рис. 4).

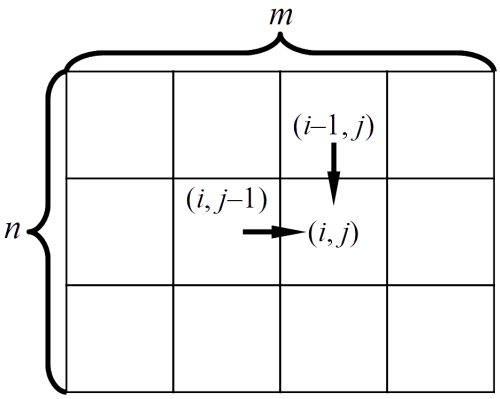

Рис. 4. Переходы динамики

Посмотрим, в каком из этих состояний набирается бóльшая сумма бонусов и прибавим число бонусов в клетке (i, j). Получим следующую формулу перехода:

$$d_{ij} = \max(d_{i-1,j}, d_{i,j-1}) + a_{ij}.$$

Заметим, что чтобы ее применять, должны существовать клетка сверху и клетка слева от (i, j). Если же какой-то из них не существует (жучок в самом левом столбце или в верхней строке), вместо максимума будет только одно число.

Рассмотрим программу на рис. 5.

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
    {
        d[i][j] = a[i][j];
        if (i > 0)
            d[i][j] = max(d[i][j], d[i - 1][j] + a[i][j]);
        if (j > 0)
            d[i][j] = max(d[i][j], d[i][j - 1] + a[i][j]);
    }
cout << d[n - 1][m - 1] << endl;
```

Рис. 5. Решение задачи про жучка

Двумя вложенными циклами перебираем клетки на поле:

for (**int** i = 0; i < n; i++)
    for (**int** j = 0; j < m; j++)

Выполняем следующее присваивание, потому что жучок точно наберёт не меньше a[i][j] бонусов.

d[i][j] = a[i][j];

Затем нам предстоит выбирать максимум. Делаем две проверки условий. Если $i > 0$, то есть корректна позиция $(i - 1, j)$, то мы делаем обновление значения d[i][j] из неё.

if (i > 0)
    d[i][j] = max(d[i][j], d[i - 1][j] + a[i][j]);

Аналогично проверяем $j > 0$:

if (j > 0)
    d[i][j] = max(d[i][j], d[i][j - 1] + a[i][j]);

Для ячейки d[0][0] оба условия не выполнятся, и в ней останется значение a[0][0], то есть она корректно инициализируется. В результате ответ на задачу будет в ячейке массива d[n-1][m-1].

Нетрудно оценить время работы программы. Она выполнит $O(nm)$ действий.

В этом решении очень важно, что, когда мы собираемся обновить значение в ячейке (i, j), предыдущие значения в ячейках $(i-1, j)$ и $(i, j-1)$ уже посчитаны. Поэтому условие о том, что жучок ходит только вправо и вниз, играет принципиальную роль. Если он мог бы ходить во всех четырёх направлениях, такая задача уже бы не решалась динамическим программированием.

Обсудим еще один важный вопрос — как найти оптимальный путь жучка. Пусть нужно вывести не только ответ на задачу, но и так называемый *сертификат* — последовательность перемещений жучка, подтверждающую этот ответ. Создадим массив p с оптимальными переходами. Пусть p[i][j] = 0, если оптимальный путь в клетку (i, j) проходит через $(i-1, j)$, и p[i][j] = 1, если оптимальный путь в клетку (i, j) проходит через клетку $(i, j-1)$.

Добавим вычисление значений массива p в код (см. рис. 6).

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
    {
        d[i][j] = a[i][j];
        if (i > 0 && d[i - 1][j] + a[i][j] > d[i][j])
        {
            d[i][j] = d[i - 1][j] + a[i][j];
            p[i][j] = 0;
        }
        if (j > 0 && d[i][j - 1] + a[i][j] > d[i][j])
        {
            d[i][j] = d[i][j - 1] + a[i][j];
            p[i][j] = 1;
        }
    }
cout << d[n - 1][m - 1] << endl;
recout(n - 1, m - 1);
```

Рис. 6. Нахождение сертификата

Если реализуется обновление значения в массиве d, то мы нашли более оптимальный переход, чем был раньше, и должны изменить значение в массиве p. Значение p[0][0] никак не заполняется и использоваться не будет. В конце вызывается функцию recout для вывода пути. Сначала в неё передаётся позиция конечной клетки $(n-1, m-1)$.

Функцию recout удобно реализовать рекурсивно (см. рис. 7). Ей передаются координаты текущей клетки (i, j), и её задача — вывести путь до этой клетки на основе массива p.

Если мы дошли до начальной клетки — путь пустой, поэтому просто выходим.

```
if (i == 0 && j == 0)
    return;
```

```
void recout(int i, int j)
{
    if (i == 0 && j == 0)
        return;
    if (p[i][j] == 0)
    {
        recout(i - 1, j);
        cout << 'D';
    }
    if (p[i][j] == 1)
    {
        recout(i, j - 1);
        cout << 'R';
    }
}
```

Рис. 7. Нахождение сертификата

Далее выполняется проверка значения p[i-1][j].

if (p[i][j] == 0)

Если оно равно нулю, мы пришли в клетку (i, j) из клетки $(i-1, j)$. Поэтому сначала рекурсивно выводим путь до клетки $(i-1, j)$:

recout(i - 1, j);

Затем нужно вывести последний шаг пути в требуемом формате. Например, пусть нужно вывести последовательность букв R и D. R означает движение вправо, D — вниз. В данном случае выводим D:

cout << 'D';

Аналогично разбирается случай, когда p[i][j] = 1, и мы пришли из клетки $(i, j-1)$. Вывод программы для примера на рис. 2 имеет вид RRDRD. Жучок делает два шага вправо, потом — шаг вниз, потом — вправо и вниз.

Аналогичным образом — при помощи рекурсивной функции — можно реализовать вывод сертификата в других задачах.

Упражнение 3.2.1
Найдите оптимальный путь жучка для примера на рис. 8.

1	**4**	**1**	**2**	**3**
2	**3**	**2**	**1**	**4**
1	**1**	**1**	**2**	**4**
2	**5**	**1**	**7**	**1**

Рис. 8. Нахождение сертификата