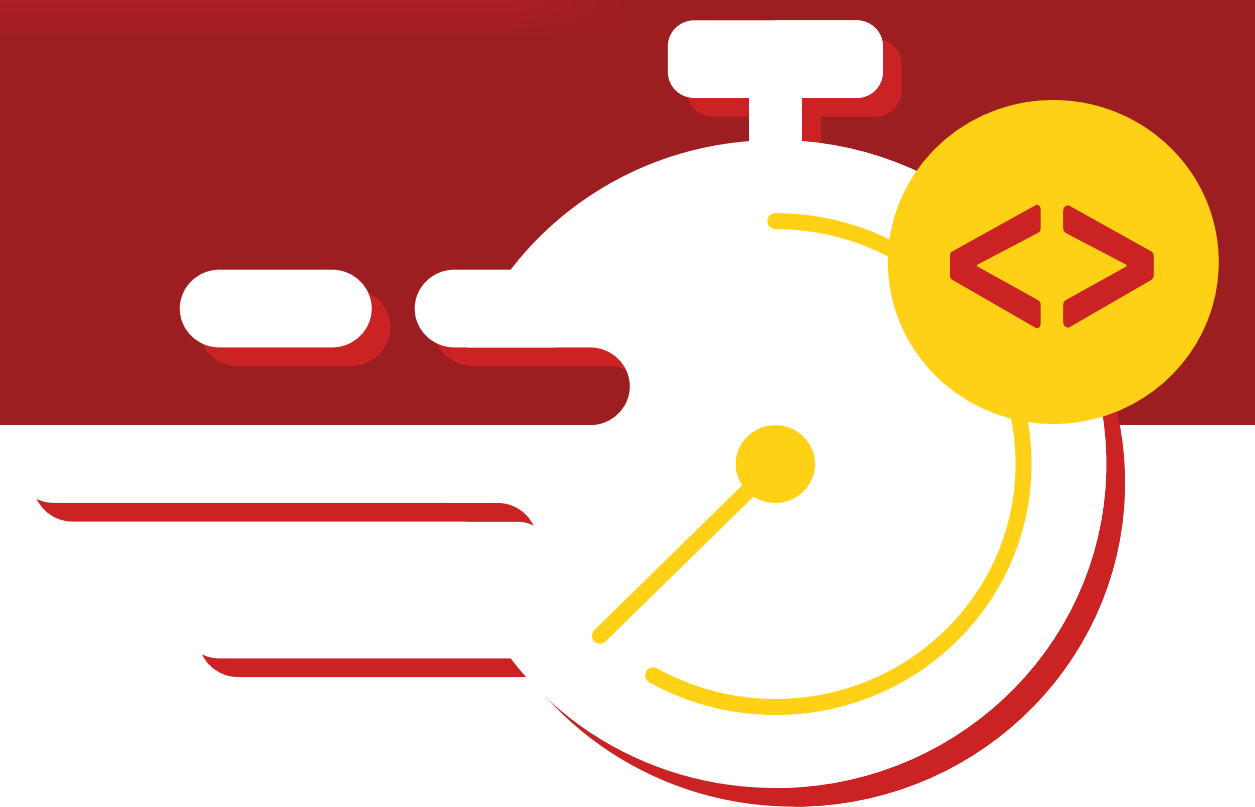
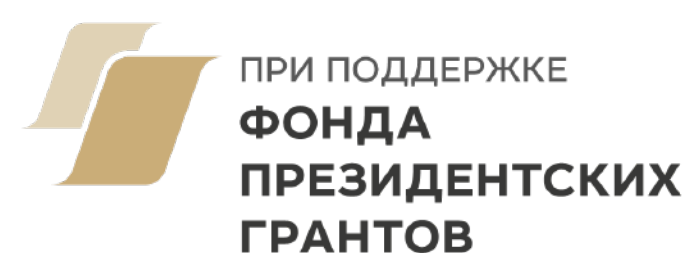


Линейное динамическо программирование. Кузейчик с препятствием

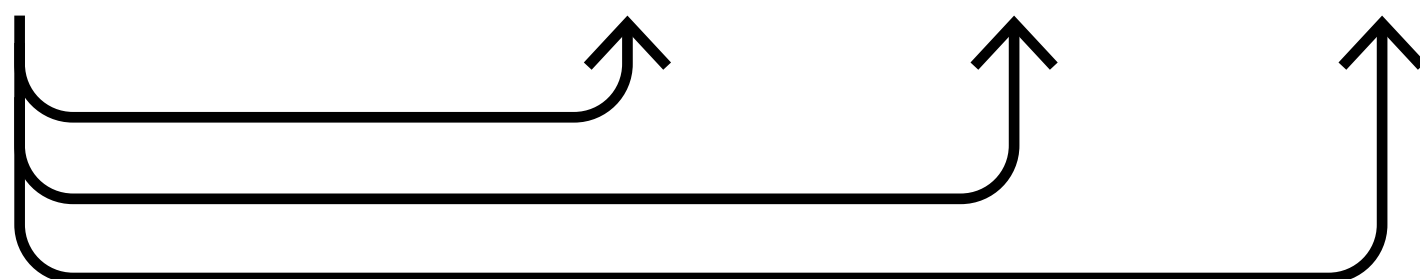
Урок 3.2.2



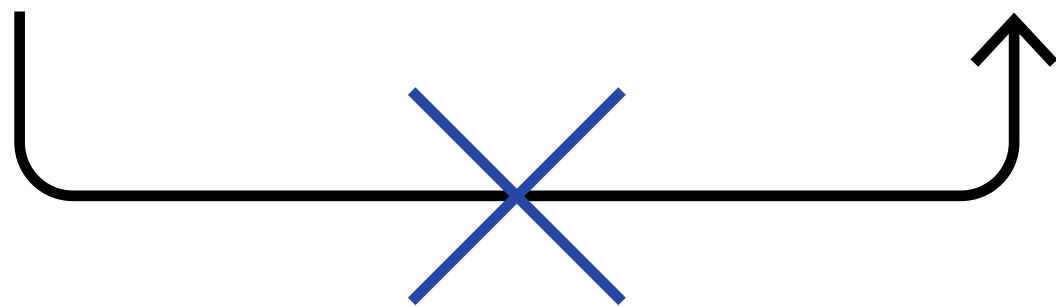
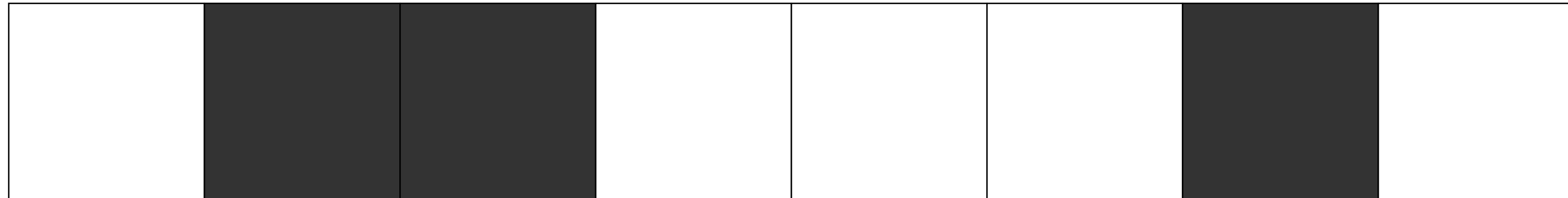
Постановка задачи_



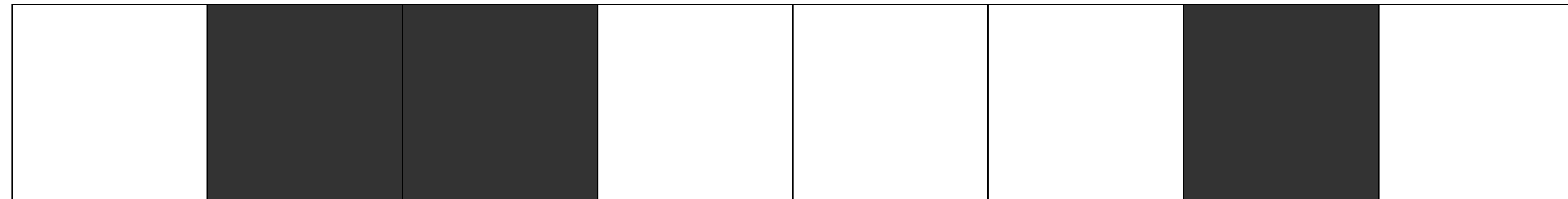
--	--	--	--	--	--	--	--



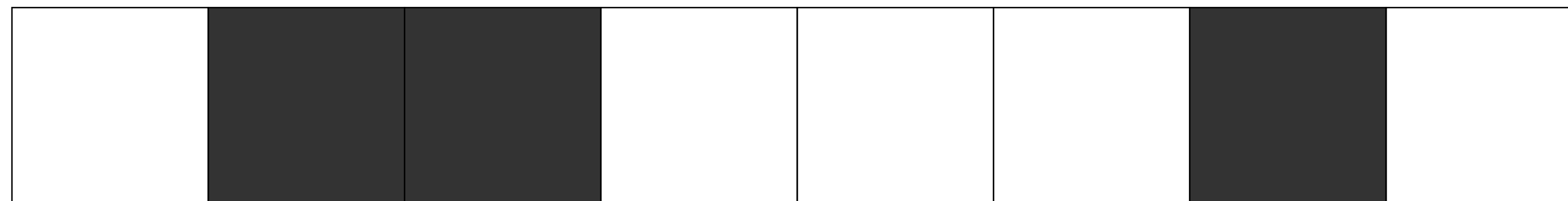
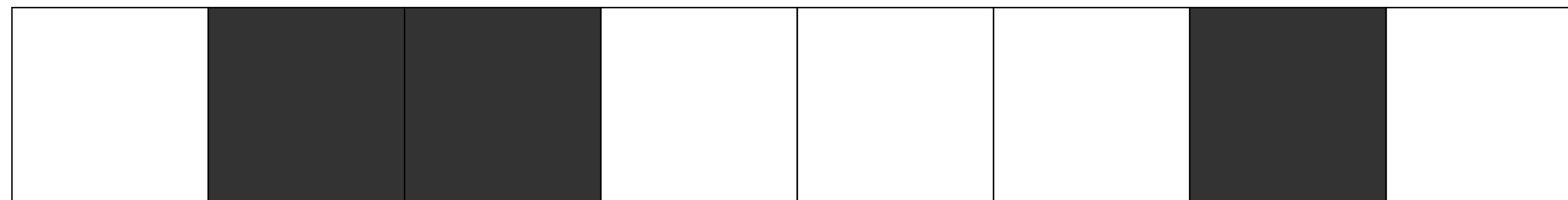
Постановка задачи_



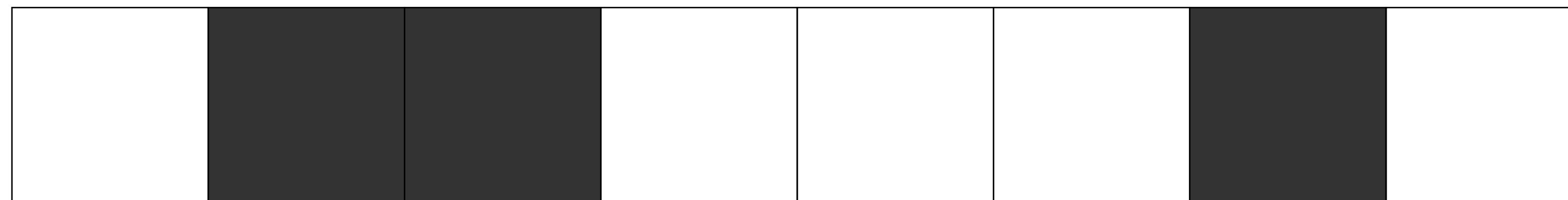
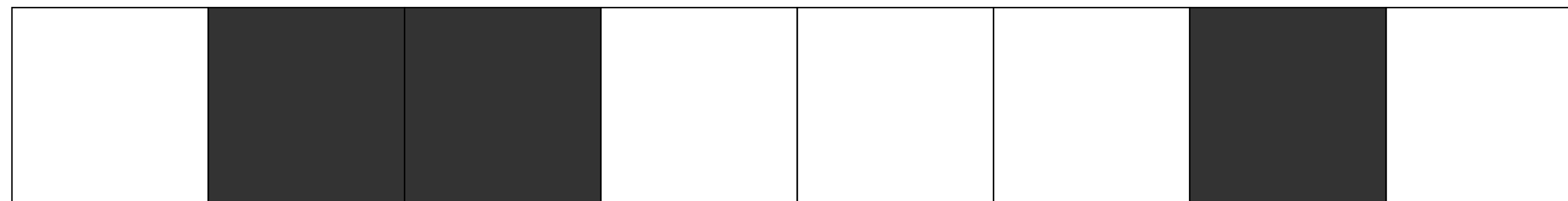
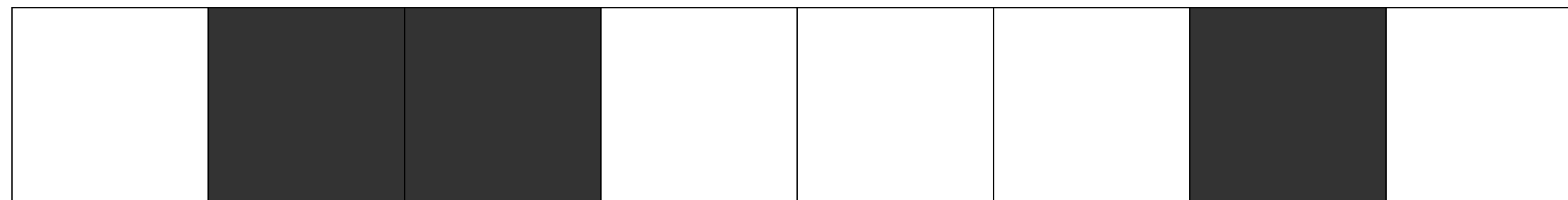
Возможные способы_



Возможные способы_



Возможные способы_



Иногда нельзя_



--	--	--	--	--	--	--	--



--	--	--	--	--	--	--	--

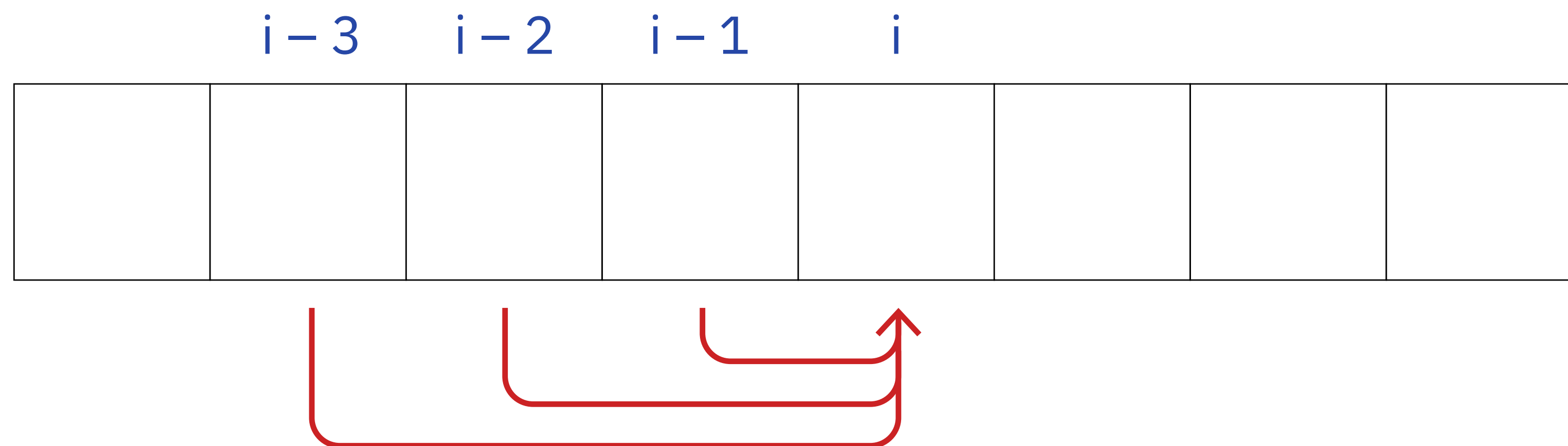
Решение_

1. Состояние
2. База
3. Формула
4. Порядок
5. Ответ

Решение_

1. Состояние: dp_i — количество способов добраться до i -й клетки
2. База
3. Формула
4. Порядок
5. Ответ: dp_n

Формула_



$$dp_i = dp_{i-1} + dp_{i-2} + dp_{i-3}$$

Решение_

1. **Состояние:** dp_i — количество способов добраться до i -й клетки

2. **База**

3. **Формула:** если клетка занята, то 0.
Иначе $dp_i = dp_{i-1} + dp_{i-2} + dp_{i-3}$

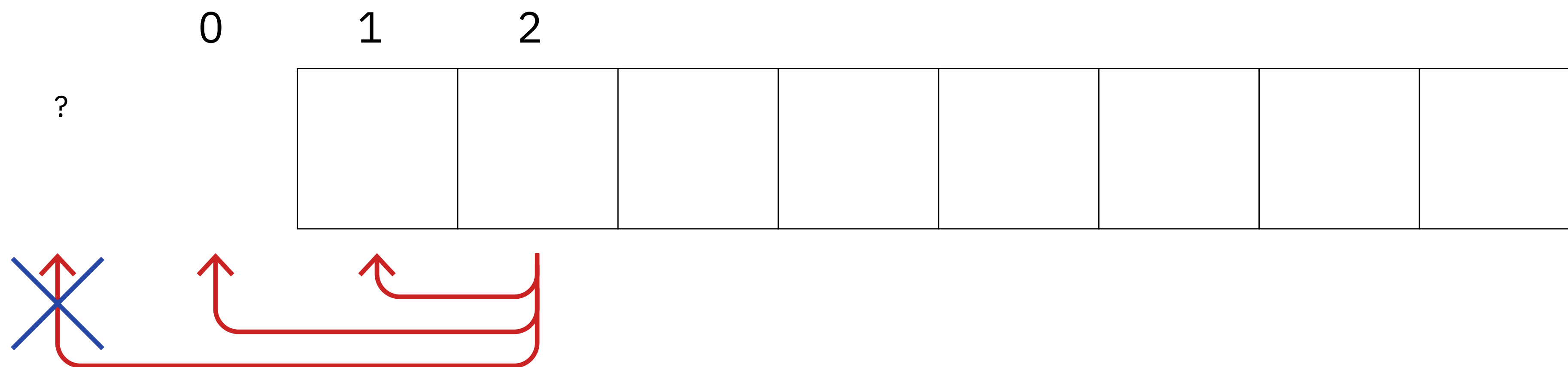
4. **Порядок**

5. **Ответ:** dp_n

Решение_

1. **Состояние:** dp_i — количество способов добраться до i -й клетки
2. **База**
3. **Формула:** если клетка занята, то 0.
Иначе $dp_i = dp_{i-1} + dp_{i-2} + dp_{i-3}$
4. **Порядок:** по возрастанию i
5. **Ответ:** dp_n

База_




Включим dp_2 в базу!

0	1	2							
1	1, если свободен, иначе 0								

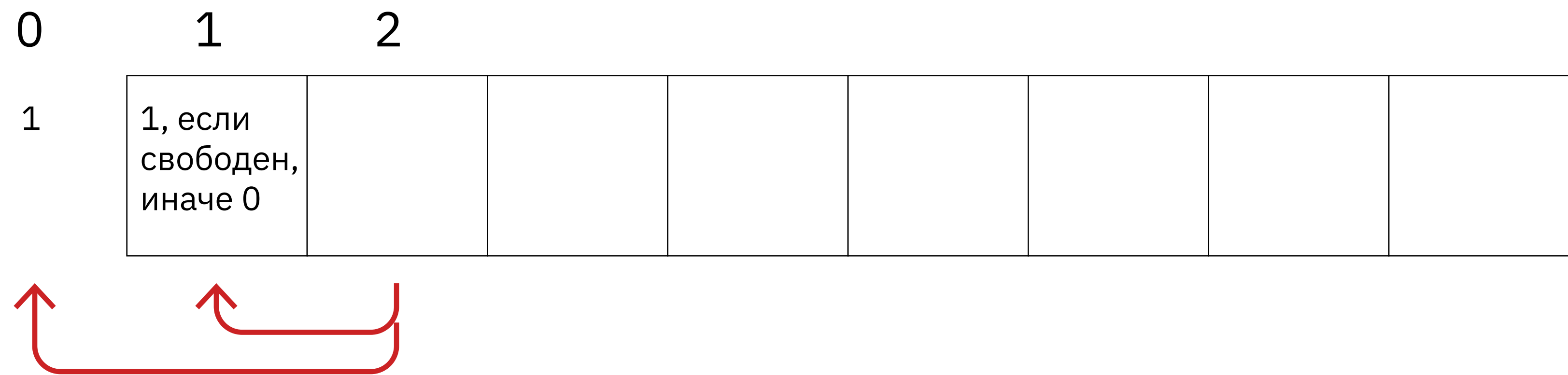


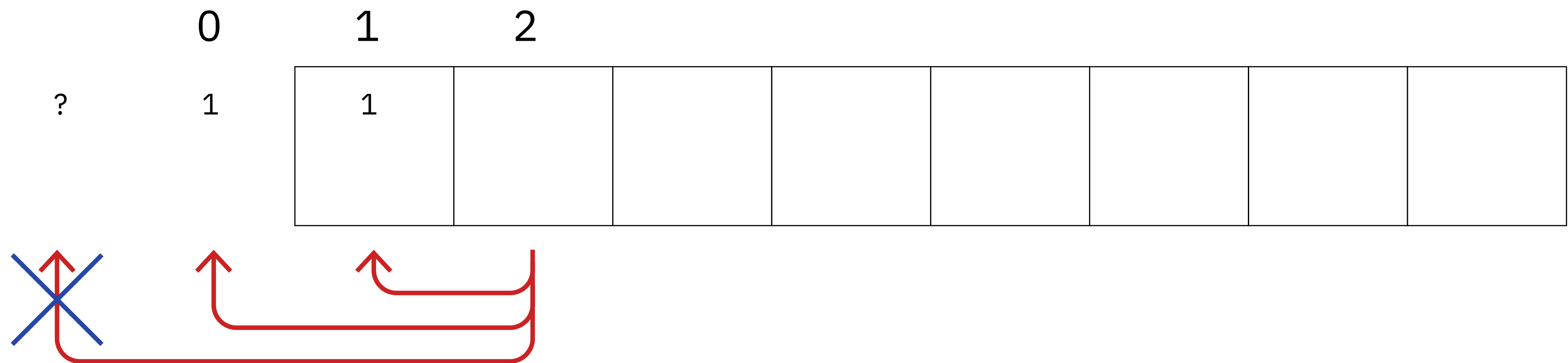
$$dp_1 = 1 - a_1$$

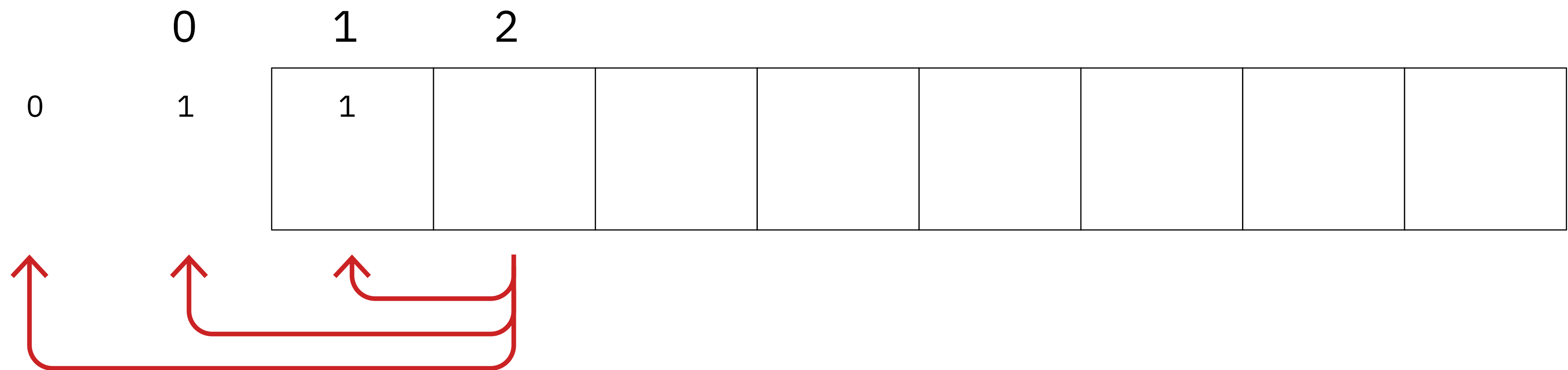
0	1	2						
1	1, если свободен, иначе 0							



0, если занята.
Иначе $dp_2 = dp_1 + dp_0$







Решение_

1. **Состояние:** dp_i — количество способов добраться до i -й клетки
2. **База:** $dp_0 = 1, dp_{<0} = 0$
3. **Формула:** если клетка занята, то 0.
Иначе $dp_i = dp_{i-1} + dp_{i-2} + dp_{i-3}$
4. **Порядок:** по возрастанию i
5. **Ответ:** dp_n

Реализация_

Ошибка!

```
1 vector <int> a;
2 vector <int> dp;
3
4 int f(int i) {
5     if (i < 0) {
6         return 0;
7     }
8     if (i == 0) {
9         return 1;
10    }
11    if (a[i] == 0) {
12        return 0;
13    } else {
14        return f(i - 1) + f(i - 2) + f(i - 3);
15    }
16 }
```

Реализация_

```
1 vector <int> a;
2 vector <int> dp; // Не используем!
3
4 int f(int i) {
5     if (i < 0) {
6         return 0;
7     }
8     if (i == 0) {
9         return 1;
10    }
11    if (a[i] == 0) {
12        return 0;
13    } else {
14        return f(i - 1) + f(i - 2) + f(i - 3);
15    }
16 }
```

Реализация_

```
1 vector <int> a;
2 vector <int> dp; // Не используем!
3
4 int f(int i) {
5     if (i < 0) {
6         return 0;
7     }
8     if (i == 0) {
9         return 1;
10    }
11    if (dp[i] != -1) { // проверяем, что уже считали
12        return dp[i];
13    }
14    if (a[i] == 0) {
15        return 0;
16    } else {
17        // Считаем, если еще этого не делали
18        dp[i] = f(i - 1) + f(i - 2) + f(i - 3);
19        return dp[i];
20    }
21 }
```

Реализация_

```
1 int n;  
2 cin >> n;  
3  
4 a.resize(n + 1);  
5 dp.assign(n + 1, -1);  
6  
7 // Чтение массива a  
8  
9 cout << f(n) << endl;
```

Итоги_

1. **Состояние:** dp_i — количество способов добраться до i -й клетки
2. **База:** $dp_0 = 1, dp_{<0} = 0$
3. **Формула:** если клетка занята, то 0.
Иначе $dp_i = dp_{i-1} + dp_{i-2} + dp_{i-3}$
4. **Порядок:** по возрастанию i
5. **Ответ:** dp_n

**Следующее занятие —
черепашка**
