Восстановление ответа

На этом занятии мы изучим, как восстанавливать ответ при решении задач методом динамического программирования.

Начнем со знакомой нам задачи про Марио, который прыгает по платформам и собирает игровые очки. В прошлый раз, решив эту задачу методом динамического программирования, мы получили вот такую матрицу ответов `dp`:

	0	1	2	3	4
0	0	INF	INF	INF	INF
1	INF	2	-5	3	4
2	INF	-8	7	14	11
3	INF	-4	5	9	10

Теперь мы хотим понять, по каким платформам нужно прыгать Марио, чтобы набрать максимальное количество баллов в ячейке (n, m).

Первый метод, который используется для решения задач на восстановление динамики – это **обратная динамика**. Как этот метор работает? Разберемся, как мы попали в ячейку (n, m) – мы сделали лучший выбор из $(n-1, m)$, $(n-1, m-1)$, $(n, m-1)$. Давайте выберем из этих значений лучшее и перейдем в эту позицию. В нашей матрице максимальное значение 14 в $(n-1, m-1) = (2, 3)$, туда и отправимся:

	0	1	2	3	4
0	0	INF	INF	INF	INF
1	INF	2	-5	3	4
2	INF	-8	7	14	11
3	INF	-4	5	9	10

Далее повторим наши рассуждения и перейдем на предыдущий шаг. При поиске ответа для $(2, 3)$ мы обнаружили, что нам выгоднее всего попасть туда через $(2, 2)$:

	0	1	2	3	4
0	0	INF	INF	INF	INF
1	INF	2	-5	3	4
2	INF	-8	7	14	11
3	INF	-4	5	9	10

Повторяя этот процесс, мы рано или поздно придем в точку $(1, 1)$ и восстановим весь путь:

	0	1	2	3	4
0	0	INF	INF	INF	INF
1	INF	2	-5	3	4
2	INF	-8	7	14	11
3	INF	-4	5	9	10

Опишем этот алгоритм в коде. Обозначим за `i` и `j` координаты текущей позиции. Мы начинаем восстановление ответа с позиции `i = n`, `j = m`.

Мы знаем, что наш маршрут начался с координаты $(1, 1)$, а пришли мы туда из $(0, 0)$. Условием окончания нашего пути можем считать `i = j = 0`.

Для хранения координат передвижения Марио создадим список `Path`, и по мере восстановления пути будем заносить туда координаты платформ:

```
i, j = n, m
path = []
while i > 0 and j > 0:
    path.append((i, j))
    if dp[i-1][j] > dp[i-1][j-1] and dp[i-1][j] > dp[i][j-1]:
      i -= 1
    elif dp[i][j-1] > dp[i-1][j-1] and dp[i][j-1] > dp[i-1][j]:
      j -= 1
    else:
      i-=1
      j-=1
print(path)
```

```
[(3, 4), (2, 3), (2, 2), (1, 1)]
```

Запустим код и посмотрим на результат. Получили правильный путь Марио, но только из конца в начало, поскольку мы разворачивали его путь с конца. Развернем список и получим нужную последовательность.

Может ли так случиться, что оптимальный маршрут не единственный? Рассмотрим пример входных данных для нашей задачи про платформы. Допустим мы получили следующую матрицу ответов динамического программирования:

	0	1	2	3
0	2	-5	5	16
1	1	4	12	13
2	5	12	15	11

Заметим, что при такой конфигурации существуют два различных маршрута, которые оба набирают максимальное количество очков:

	0	1	2	3
0	2	-5	5	16
1	1	4	12	13
2	5	12	15	11

	0	1	2	3
0	2	-5	5	16
1	1	4	12	13
2	5	12	15	11

Если следовать нашему алгоритму восстановления ответа, то находясь в точке $(2, 2)$, можно пойти как по одному маршруту (вверх), так и по-другому (влево). Оба маршрута приведут от начального положения персонажа в конечное и позволят собрать по пути максимальное количество игровых очков. Какой из маршрутов выведет ваш алгоритм зависит от того, в каком порядке вы поставите условия перехода вверх, влево или по диагонали.

Какова сложность алгоритма восстановления ответа? Каждый переход – это посещение очередной координаты оптимального маршрута, никаких посторонних движений мы не делаем, а значит, восстанавливаем ответ за длину ответа.

Еще один способ восстановить ответ для динамики - сразу при построении динамики сохранять не только новое состояние, но и способ его получения. Список, для хранения таких значений часто называют **массивом предков**.

Добавим восстановление ответа в одномерную версию задачи про Марио и платформы. Посмотрим на код решения задачи, который мы обсудили на занятии по составлению плана:

```
dp[1] = a[1]
for i in range(2, n+1):
    if dp[i-1] > dp[i-2]:
        dp[i] = dp[i-1] + a[i]
    else:
        dp[i] = dp[i-2] + a[i]
print(dp[n])
```

В условном операторе мы проверяем, откуда нам было бы выгоднее прийти: из позиции `i-2` или из `i-1`. В этот же момент мы можем зафиксировать наш выбор в отдельном списке. Сохраним в `prev[i]` координату платформы, из которой нам выгодно прийти на платформу `i`:

```
if dp[i-1] > dp[i-2]:
    dp[i] = dp[i-1] + a[i]
    prev[i] = i - 1
else:
    dp[i] = dp[i-2] + a[i]
    prev[i] = i - 2
```

Изначально весь список `prev` можно заполнить значениями `-1`, которые будут означать, что у этой платформы нет предка. Выставляя начальные значения `dp[1] = a[1]` добавим и тот факт, что попасть на первую платформу можно только с `0`:

```
prev = [-1] * (n+1)
dp[1] = a[1]
prev[1] = 0
```

У платформы номер 0 предок останется равным `-1`. Используем этот факт для правильного завершения процесса восстановления ответа.

Создадим список `path` для хранения оптимального пути. Начнем восстановление с конца, установив `i = n`. Добавим эту координату в ответ. Заметим, что в `prev[i]` находится та координата, из которой мы пришли в координату `i`. Мы сохранили это значение, когда получали ответ для `dp[i]`. Теперь мы должны перейти на эту позицию, чтобы восстановить всю цепочку перемещений Марио, что можно сделать командой `i = prev[i]`. Повторяем восстановление ответа, пока `i != -1`, так как `i == -1` означает, что мы пришли на нулевую позицию и закончили восстановление ответа:

```
path = []
i = n
while i != -1:
    path.append(i)
    i = prev[i]

 print(path[::-1])
```

Сравним два рассмотренных метода восстановления ответа. Восстановление по предкам происходит так же быстро, как и в обратной динамике – за длину ответа. Однако при сохранении предков не нужно думать, откуда мы пришли в эту позицию `i` – ответ записан в `prev[i]`. В обмен на однозначность требуется хранить массив предков. Для многомерной динамики такая трата памяти иногда бывает критичной.

Сегодня мы рассмотрели два самых популярных способа хранения и восстановления ответа в динамике. Есть задачи, в которых восстановление ответа требует более сложных действий, чем построение самой динамики. Например, интересное восстановление ответа можно увидеть в алгоритме поиска кратчайшего пути во взвешенном графе. Но о графах мы поговорим на следующих занятиях.