ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Школа::Кода

Олимпиадное программирование

2020-2021 Таганрог

Что такое ДП?

- Динамическое программирование – это когда у нас есть задача, которую непонятно как решать, и мы разбиваем ее на меньшие задачи, которые тоже непонятно как решать. (с) А.Кумок
- Динамическое программирование – это метод оптимизации, который заключается в нахождении структуры оптимального решения.
  Этот метод применим, когда оптимальное решение задачи может быть составлено из оптимальных решений её подзадач.

Представление ДП

- ДП представляют в виде набора состояний.
- Каждое состояние имеет параметры. Каждому набору параметров соответствует одно состояние ДП.
- Каждое состояние имеет своё значение.
- Перед решением задачи состояниям задаются начальные значения.
- Между состояниями есть переходы, позволяющие вычислять значения одних состояний на основе значений других по определённой формуле.
- Ответом на задачу может быть значение одного из состояний ДП, сумма нескольких состояний, их минимум и т.д.

Задача 1

- Дано число N требуется вычислить N-е число Фибоначчи.
- По определению f(N) = f(N – 1) + f(N – 2), f(0) = 1, f(1) = 1.

```
int f(int n)
{
    if (n == 0)
        return 1;
    if (n == 1)
        return 1;
    return f(n - 1) + f(n - 2);
}
```

С точки зрения ДП:

n – это параметр состояния, f(n) – значение состояния;

В состояние k существуют переходы из состояний k – 1 и k – 2, такие что dp[k] = dp[k – 1] + dp[k – 2];

Начальные значения – dp[0] = 1, dp[1] = 1.

Реализация

Ленивая динамика

```
const int MAXN = 40;
vector<int> dp(MAXN, -1);

int f(int n)
{
    if (dp[n] != -1)
        return dp[n];
    return dp[n] = f(n - 1) + f(n - 2);
}

int main()
{
    dp[0] = dp[1] = 1;
```

Прямой пересчёт

```
const int MAXN = 40;

int main()
{
    vector<int> dp(MAXN);
    dp[0] = dp[1] = 1;
    for (int i = 2; i < MAXN; ++i)
        dp[i] = dp[i - 1] + dp[i - 2];
```

Обратный пересчёт

```
const int MAXN = 40;

int main()
{
    vector<int> dp(MAXN, 0);
    dp[0] = 1;
    for (int i = 0; i < MAXN - 1; ++i)
    {
        dp[i + 1] += dp[i];
        if (i < MAXN - 2)
            dp[i + 2] += dp[i];
    }
```

Задача 2?

- Есть лестница длиной N ступенек.
- За 1 шаг вы можете подняться на 1 или 2 ступеньки вверх.
- Изначально вы стоите перед первой ступенькой.
- Сколько существует различных способов подняться по лестнице?

- Номер ступеньки – параметр состояния.
- Количество способов подняться на ступеньку – значение состояния.
- Начальное значение dp[0] = 1.
- На ступеньку k можно перейти со ступенек k – 1 и k – 2.

Задача 2

- Есть лестница длиной N ступенек, на каждой ступеньке написано положительное число A_k.
- За 1 шаг вы можете подняться на 1 или 2 ступеньки вверх, когда вы становитесь на ступеньку, к результату прибавляется число, написанное на ней.
- Изначально вы стоите перед первой ступенькой.
- Какой максимальный результат вы можете получить при подъёме по лестнице?
- Номер ступеньки – параметр состояния.
- Максимальный результат, который можно набрать, поднявшись до определённой ступеньки – значение состояния.
- Начальное значение dp = {0, … 0}.
- На ступеньку k можно перейти со ступенек k – 1 и k – 2, следовательно $dp[k] = \max(dp[k - 1], dp[k - 2]) + A_k$.

Задача о рюкзаке

- Есть рюкзак объёмом V и N предметов, каждый из которых имеет объём A_i. Какой максимальный объём рюкзака можно заполнить?
- Идея: сначала определим, какой объём рюкзака можно заполнить используя только первый предмет. Затем определим, какой объём можно заполнить добавив второй предмет, и т.д.

dp[‘количество рассмотренных предметов’] [‘набранный объём’] =
= 0, если состояние недостижимо, иначе 1

Начальное значение: dp[0][0] = 1

Переход: dp[i][j] = max(dp[i – 1][j], dp[i – 1][j – A_i])

Ответ: максимальное j, такое что dp[N][j] = 1

Задача о рюкзаке

Создадим матрицу dp[N + 1][V + 1].
Пусть изначально её элементы равны 0, за исключением dp[0][0] = 1.
Будем проходить по всем элементам матрицы, начиная со строки 1 и пересчитывать значения, согласно переходам.

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
3	2	0	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0

Задача о рюкзаке. Шаг 1

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
3	2	0	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 2

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0
3	2	0	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 3

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0
3	2	0	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 4

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	0	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 8

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	0	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 9

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 10

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 11

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	0	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 12

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 13

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	0	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 14

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 15

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	0	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 16

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	1	0	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 17

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	1	1	0	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 18

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	1	1	1	0	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 19

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	1	1	1	1	0	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 20

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	1	1	1	1	1	0	0

Переход: $dp[i][j] = \max(dp[i-1][j], dp[i-1][j-A_i])$

Задача о рюкзаке. Шаг 21

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	1	1	1	1	1	0	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Шаг 22

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	1	1	1	1	1	1	0

Переход: $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - A_i])$

Задача о рюкзаке. Итоговая матрица

	j	0	1	2	3	4	5	6
	i							
A	0	1	0	0	0	0	0	0
2	1	1	0	1	0	0	0	0
3	2	1	0	1	1	0	1	0
1	3	1	1	1	1	1	1	1

Ответ: 6

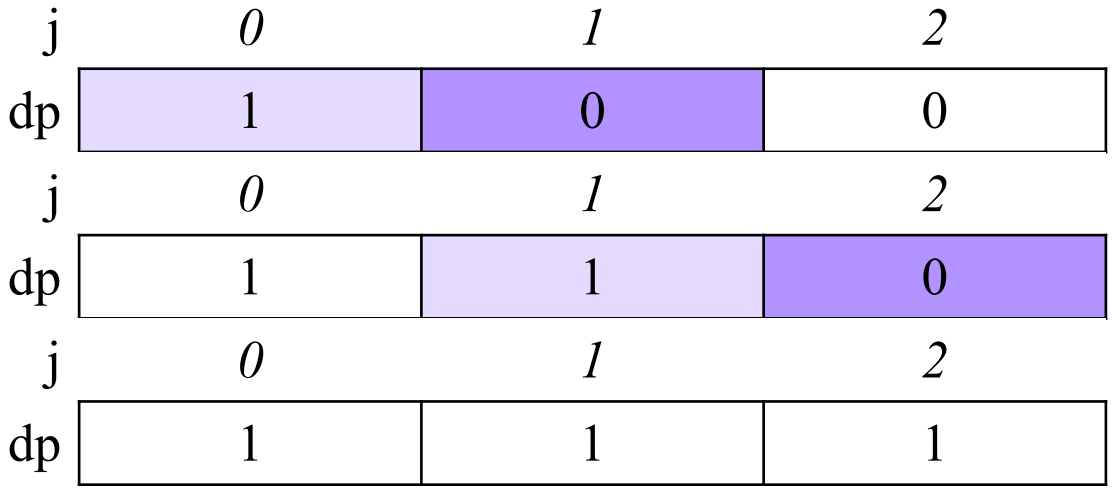
Задача о рюкзаке. Оптимизация

- Базовый алгоритм имеет асимптотику O(N*V) и требует N*V памяти. Однако, можно заметить, что при переходе мы не можем ухудшить уже имеющийся результат, что позволяет провести вычисления на одномерном массиве.
- Перейдём от матрицы dp[N + 1][V + 1] к массиву dp[V + 1].
- Теперь для пересчёта будем пользоваться формулой $dp[j] = max(dp[j], dp[j - A_i])$.

Оптимизированный рюкзак с повторениями

Заметим, что если мы будем перебирать j от 0 до V, то может произойти ситуация, когда значение dp[j - A_i] стало равно 1 при использовании предмета с номером i. В таком случае значение dp[j] так же станет равно 1, а это будет означать что мы использовали предмет с номером i уже несколько раз. Такой порядок обхода применяется, когда в условии сказано, что у вас есть неограниченное количество каждого предмета.

$A_0 = 1$



Оптимизированный рюкзак без повторений

Если же каждый предмет дан в единственном экземпляре, то для предотвращения повторений достаточно изменить порядок обхода массива и перебирать j от V до 0.

$A_0 = 1$

j	*0*	*1*	*2*
dp	1	0	0

j	*0*	*1*	*2*
dp	1	0	0

j	*0*	*1*	*2*
dp	1	1	0