Лекция 7

Динамическое программирование

Динамическое программирование

в теории управления и теории вычислительных систем — способ решения сложных задач путём разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. В этом случае время вычислений, по сравнению с «наивными» методами, можно значительно сократить.

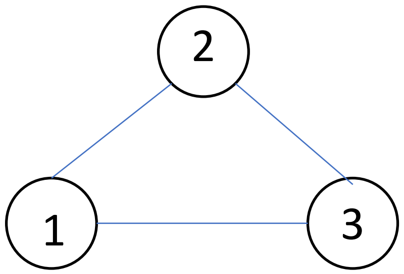
Будут рассмотрены задачи

- о путевом графе;
- о рюкзаке;
- о выравнивании последовательностей.

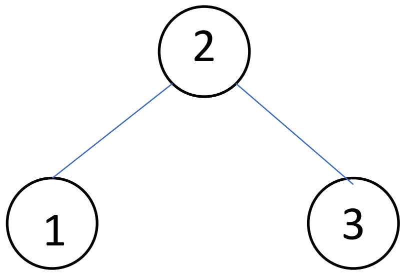
Задача на путевом графе

- G=(V, E). Независимое множество графа G – это подмножество взаимно-несмежных вершин.

Пример:



4 независимых подмножества
$\emptyset, \{1\}, \{2\}, \{3\}$



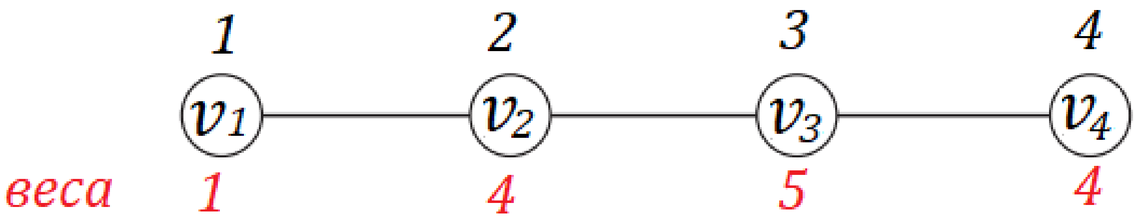
5 независимых подмножеств
$\emptyset, \{1\}, \{2\}, \{3\}$, {1,3}

Каждая вершина v_i имеет вес $w_i \geq 0$.

Задача MWIS (Max Weighted Independent Set) : найти в гафе G независимое множество с максимальным суммарным весом входящих в него вершин.

Путевой граф (простейший случай)

Пример.



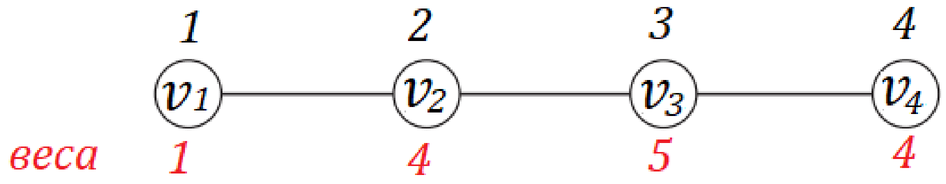
$$n = |V| = 4, \qquad m = |E| = 3$$

8 независимых множеств: $\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1,3\}, \{2,4\}, \{1,4\}$

Количество подмножеств растет экспоненциально!!!

- Жадная стратегия – выбрать самую тяжелую вершину. Тогда $W_{max} = 5 + 1 = 6$. Не оптимально! Эта жадная стратегия не подходит.
- Рекурсивно делить пополам и потом соединять («Разделяй и властвуй») тоже не подойдет.

Решение



Рассмотрим $G = (V, E)$ – путевой граф с ребрами $(v_1, v_2), (v_2, v_3), \dots,$ (v_{n-1}, v_n) и весами w_i для каждой v_i. Пусть $n \geq 2$.

Пусть уже найдено S – MWIS и W – его вес. Тогда

Случай 1: $v_n \notin S \Rightarrow S$ для G совпадает с S для G_{n-1} и W_n=W_{n-1};

Случай 2: $v_n \in S \Rightarrow v_{n-1} \notin S$, S для G совпадает с S для $(G_{n-2} \cup v_n)$ и W_n=$W_{n-2} + w_n$

Лемма 1

(оптимальная подструктура взвешенного независимого множества). *Пусть S равно независимому множеству с максимальным весом (MWIS) путевого графа G с не менее чем 2 вершинами. Пусть G_i обозначает подграф графа G, содержащий его первые i вершин и $i-1$ ребер. Тогда S является либо*:

(*i*) независимым множеством с максимальным весом (MWIS) G_{n-1},

либо

(*ii*) независимым множеством с максимальным весом (MWIS) G_{n-2}, дополненным конечной вершиной v_n графа G.

Следствие. (рекуррентное соотношение взвешенного независимого множества)

С допущениями и обозначением леммы 1 пусть W_i обозначает суммарный вес независимого множества с максимальным весом (MWIS) графа G_i. При $i = 0$ интерпретируем W_i как 0. Тогда

$$W_n = \max\{\underbrace{W_{n-1}}_{\text{Случай 1}}, \underbrace{W_{n-2} + w_n}_{\text{Случай 2}}\}.$$

В более общем случае для каждого $i = 2, 3, \ldots, n$

$$W_i = \max\{W_{i-1}, W_{i-2} + w_i\}.$$

Алгоритм нахождения суммарного **веса** независимого множества максимального веса (восходящая реализация)

ВЗВЕШЕННОЕ НЕЗАВИСИМОЕ МНОЖЕСТВО (WIS)

Вход: путевой граф G с множеством вершин $\{v_1, v_2, ..., v_n\}$ и неотрицательным весом для каждой вершины v_i.

Выход: суммарный вес независимого множества с максимальным весом графа G.

A := массив длиной $(n + 1)$ `// решения подзадач`

$A[0] := 0$ `// базовый случай #1`

$A[1] := w_1$ `// базовый случай #2`

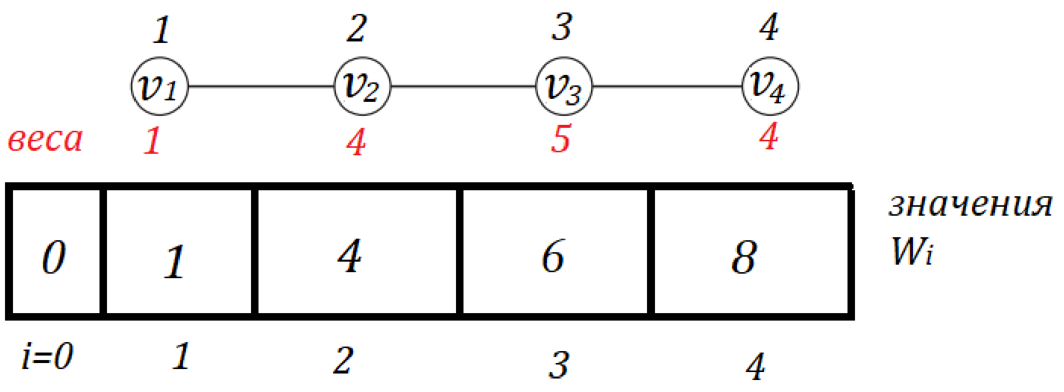
for $i = 2$ to n **do**

    `// использовать рекуррентное соотношение`
    `// из следствия 16.2`

$$A[i] := \max\{\underbrace{A[i-1]}_{\text{Случай 1}}, \underbrace{A[i-2]+w_i}_{\text{Случай 2}}\}$$

return $A[n]$ `// решение наибольшей подзадачи`

Используем кеш.
Будем вычислять последовательно W_i, сохраняя результаты в глобальном массиве *A*.



Алгоритм реконструкции WIS (получение MWIS)

WIS_RECONSTRUCTION

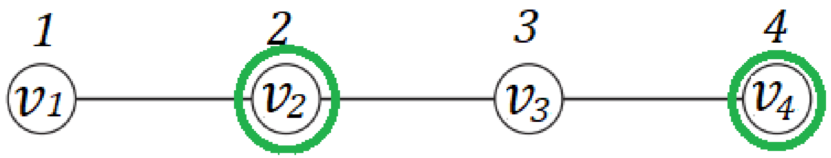
Вход: массив A, вычисленный алгоритмом WIS для путевого графа G с множеством вершин $\{v_1, v_2, ..., v_n\}$, и неотрицательный вес w_i для каждой вершины v_i.

Выход: независимое множество с максимальным весом графа G.

$S := \emptyset$ // вершины в множестве MWIS
$i := n$
while $i \geq 2$ **do**
    if $A[i-1] \geq A[i-2] + w_i$ **then** // Случай 1 побеждает
        $i := i - 1$ // исключить v_i
    else // Случай 2 побеждает
        $S := S \cup \{v_i\}$ // включить v_i
        $i := i - 2$ // исключить v_{i-1}
if $i = 1$ **then** // базовый случай #2
    $S := S \cup \{v_1\}$
return S

веса		1	4	5	4
значения W_i	0	1	$\boxed{4}$	6	$\boxed{8}$
	i=0	1	2	3	4

Таким образом, $S = \{2, 4\}$



Парадигма динамического программирования

- Определить относительно малую коллекцию подзадач

$(n+1)$ подзадач MWIS для G_i

- Показать, как быстро и правильно решать «более крупные» подзадачи с учетом решения «более мелких»

$$W_n = \max\{W_{i-1}, W_{i-2} + w_i\}$$

- Показать, как быстро и правильно выводить окончательное решение из решений всех подзадач

$$W = W_n$$

Сложность: $$\underbrace{f(n)}_{\#\ \text{подзадачи}} \times \underbrace{g(n)}_{\substack{\text{время на подзадачу} \\ (\text{с учетом предыдущих решений})}} + \underbrace{h(n)}_{\text{постобработка}} .$$

$$\left.\begin{aligned} f(n) &= O(n) \\ G(n) &= O(1) \\ H(n) &= O(n) \end{aligned}\right\} \Rightarrow O(n)$$

Задача о рюкзаке

Имеется n предметов.

$0 \le v_1, v_2, \dots, v_n$ - стоимости

$0 \le s_1, s_2, \dots, s_n$ - размеры

C – емкость рюкзака

Найти $S \subseteq \{1, 2, \dots, n\}$: $\sum_{i \in S} v_i \to max$ при $\sum_{i \in S} s_i \le C$

Пример: задача о рюкзаке с $n = 4$ предметами и $C = 6$

Предмет	Значение	Размер
1	3	4
2	2	3
3	4	2
4	4	3

Каково суммарное значение оптимального решения?

а) 6

б) 7

в) 8

г) 10

Решение

Пусть уже имеется S – оптимальное решение. $V = \sum_{i \in S} v_i$. Тогда

Случай 1: $n \notin S$. $\Rightarrow S$ – оптимальное решение для задачи с предметами $\{1, 2, \dots, n-1\}$ и $V = \sum_{i \in S} v_i$

Случай 2: $n \in S$. $\Rightarrow S - \{n\}$ – оптимальное решение для меньшей подзадачи для емкости $C - s_n$ и $V' = V - v_n$

Лемма. Пусть S равно оптимальному решению задачи о рюкзаке с $n \geq 1$ с $v_1, v_2, \dots, v_n$ - стоимостями, $s_1, s_2, \dots, s_n$ - размерами и емкостью C. Обозначим $V_{i,c}$ – максимальное суммарная стоимость подмножества первых i предметов с размером рюкзака не более c.

При $i = 0$ $V_{i,c} = 0, \forall i = 1,2, \dots, n$ и $c = 0, 1, \dots, C$

$$V_{i,c} = \begin{cases} \underbrace{V_{i-1,\,c}}_{\text{Случай 1}} & \text{если } s_i > c \\ \max\{\underbrace{V_{i-1,\,c}}_{\text{Случай 1}}, \underbrace{V_{i-1,\,c-s_i} + v_i}_{\text{Случай 2}}\} & \text{если } s_i \leq c. \end{cases} \qquad (**)$$

Поскольку и C, и размеры предметов являются целыми числами, остаточная емкость $c - s_i$ во втором случае также является целым числом.

Подзадачи задачи о рюкзаке

Вычислить $V_{i,c}$, суммарное значение оптимального решения задачи о рюкзаке с первыми i элементами и вместимостью рюкзака c.

Алгоритм

KNAPSACK

Вход: значения предметов $v_1, v_2, ..., v_n$, размеры предметов $s_1, s_2, ..., s_n$ и емкость ранца C (все положительные целые числа).

Выход: максимальное суммарное значение подмножества $S \subseteq \{1, 2, ..., n\}$, где $\Sigma_{i \in S}\, s_i \le C$.

```
// решения подзадач (проиндексированы от 0)
A := (n + 1) × (C + 1) двумерный массив
// базовый случай (i = 0)
for c = 0 to C do
    A[0][c] = 0
```

```
// систематически решить все подзадачи
for i = 1 to n do
    for c = 0 to C do
        // использовать рекуррентное соотношение
        // из **
        if s_i > c then
            A[i][c] := A[i − 1][c]
        else
            A[i][c] :=
```

$$\max\{\underbrace{A[i-1][c]}_{\text{Случай 1}},\ \underbrace{A[i-1][c-s_i]+v_i}_{\text{Случай 2}}\}$$

```
return A[n][C] // решение наибольшей подзадачи
```

Подзадач: $(n + 1)(C + 1)$, $O(1)$ – на каждую
Имеем сложность $O(n \cdot C)$
Задача квазиполиномиальна.

Теорема (свойство рюкзака)

Для каждого экземпляра задачи о рюкзаке алгоритм KNAPSACK возвращает суммарное значение оптимального решения и выполняется за время $O(n \cdot C)$, где n – число предметов, C – емкость рюкзака.

Доказательство: индукция по числу предметов с использованием (**) .

Продолжение примера. (C=6). Заполняем по столбцам.

Предмет	Значение	Размер
1	3	4
2	2	3
3	4	2
4	4	3

Остаточная емкость c

6	0				
5	0				
4	0				
3	0	0			
2	0	0			
1	0	0	0	0	0
0	0	0	0	0	0
	0	1	2	3	4

префиксная длина *i*

Остаточная емкость c

6	0	3			
5	0	3			
4	0	3			
3	0	0			
2	0	0	0		
1	0	0	0	0	0
0	0	0	0	0	0
	0	1	2	3	4

префиксная длина *i*

Остаточная емкость c

6	0	3	3	7	8
5	0	3	3	6	8
4	0	3	3	4	4
3	0	0	2	4	4
2	0	0	0	4	4
1	0	0	0	0	0
0	0	0	0	0	0
	0	1	2	3	4

префиксная длина *i*

Реконструкция (элементы множества *S*)

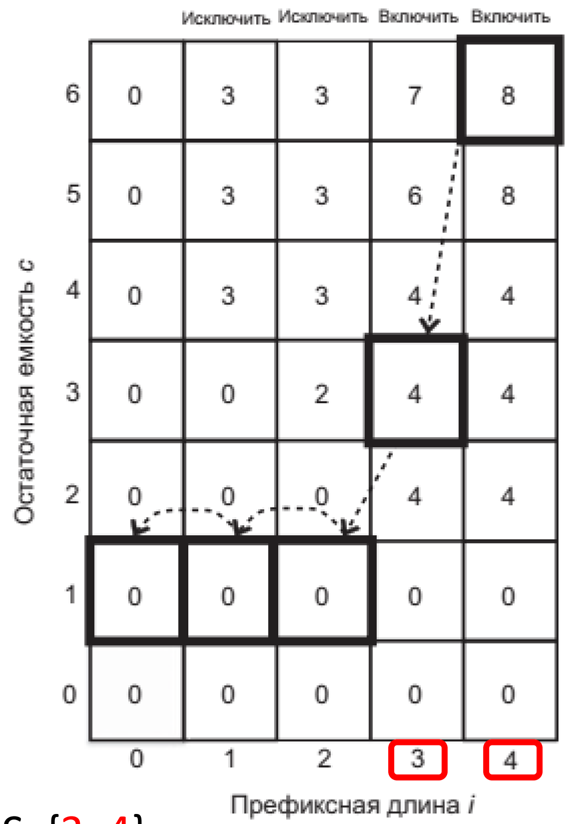
Предмет	Значение	Размер
1	3	4
2	2	3
3	4	2
4	4	3

KNAPSACK_RECONSTRUCTION

Вход: массив A, вычисленный алгоритмом Knapsack, со значениями предметов $v_1, v_2, \dots, v_n$, размерами предметов $s_1, s_2, \dots, s_n$ и емкостью ранца C.

Выход: оптимальное решение задачи о ранце.

$S := \emptyset$ `// предметы в оптимальном решении`
$c := C$ `// остаточная емкость`
for $i = n$ downto 1 **do**
  if $s_i \le c$ and $A[i-1][c-s_i] + v_i \ge A[i-1][c]$ **then**
    $S := S \cup \{i\}$ `// случай 2 побеждает, включить i`
    $c := c - s_i$ `// резервировать для него место`
  `// в противном случае пропустить i, емкость остается`
  `// прежней`
return S



Выбрали предметы *S*={3, 4}