Dynamic Programming (I)

Chin Ka Wang {rina__owo}

2025-03-19

Content

1. Introduction to DP
2. 0-1 Knapsack
3. Unbounded Knapsack UKP
4. Output solution, no. of solution

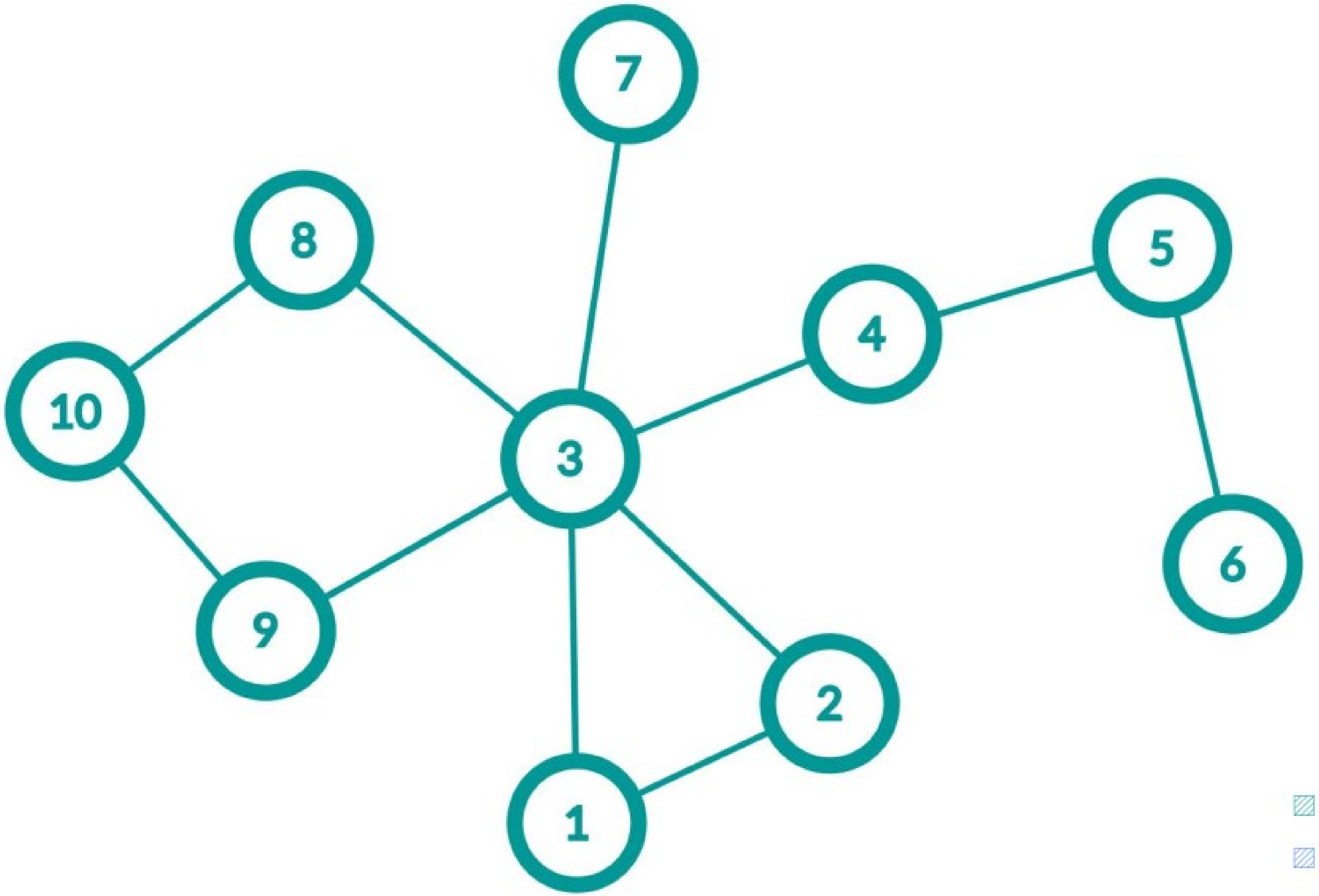
Content

1. **Introduction to DP**
2. 0-1 Knapsack
3. Unbounded Knapsack UKP
4. Output solution, no. of solution

What is Dynamic programming?

- Dynamic programming (DP) is a method for solving complex problems by breaking down the original problem into relatively simpler sub-problems.

- It is not a specific algorithm but a method to solve specific problems, it appears in a variety of data structures, and the types of questions related to it are more complicated.


7
8
5
4
10
3
6
9
2
1

Key steps to do a DP question:

1. Observe that the question doesn’t have aftereffect（無後效性）
2. Define state
3. Find the state transition equation
4. Find the value of target state using the previous state

Content

There are N items and a knapsack with capacity M.

The weight of the i^{th} item is w_i , the value is v_i .

Find out the largest total cost of the items that the knapsack can afford.

Let us observe what will happen before and after we put the i^{th} item into the knapsack:

Let v be the current value, w be the current weight.

Before: (v, w)

After: $(v + v_i, w + w_i)$

For a knapsack with infinite capacity, assume we have processed the previous $i-1$ items, there are only two situations:

(m_i means the max value for the first i items)

Value of putting the i^{th} item $= m_{i-1} + v_i$

Value of not putting the i^{th} item $= m_{i-1}$

By combining two situations, max value the knapsack can carry for the first i item is:

$\max(m_{i-1} + v_i, m_{i-1})$

Note that no matter how we put the items, the total weight will always increase.

↓

When we process the i^{th} item with capacity j, it never affect the max value of capacity $< j$. (No aftereffect)

Define State

- Let DP[i][j] be the max value of putting the first i items in a bag of capacity j.

- The answer of the question is DP[N][M].

Define State Transition Equation

Val	Wt	Item	Max Weight 0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1	1	1
4	3	2	0	1	1	4	5	5	5	5
5	4	3	0	1	1	4	5	6	6	9
7	5	4	0	1	1	4	5	7	8	9

Define State Transition Equation

State transition equation: Link the relationship between the states of the first $i - 1$ items and the first i item.

Define State Transition Equation

$$\mathrm{DP[i][j]} = \max(\mathrm{DP}[i-1][j], \mathrm{DP}[i-1][j - w_i] + \mathrm{v}_i)$$

Enumerate Order

Val	Wt	Item	Max Weight 0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1	1	1
4	3	2	0	1	1	4	5	5	5	5
5	4	3	0	1	1	4	5	6	6	9
7	5	4	0	1	1	4	5	7	8	9

Content

1. Introduction to DP
2. 0-1 Knapsack
3. **Unbounded Knapsack UKP**
4. Output solution, no. of solution

Unbounded Knapsack

UKP is basically a knapsack problem which allows unlimited repetition of items.

Unbounded Knapsack

Naive solution:

For the i^{th} item, loop till the current weight exceed the capacity.

(Treat the i^{th} item as many different items)

$$f(i,j) = max\begin{cases} f(i-1,j) \\ f(i-1,j-v[i]*1)+w[i]*1 \\ \dots \\ f(i-1,j-v[i]*k)+w[i]*k & k*v[i] \leq j \end{cases}$$

Unbounded Knapsack

$$f(i,j) = max \begin{cases} f(i-1,j) \\ f(i-1,j-v[i]*1)+w[i]*1 \\ \dots \\ f(i-1,j-v[i]*k)+w[i]*k \quad k*v[i] \le j \end{cases}$$

Note that the enumerate of j is from 0 to M →
we can get all info of $j - k * v[i]$ before j →
$j - v[i]$ is already updated using the info of $j - 2 * v[i]$ →
We just need to compare j and $j - v[i]$

Unbounded Knapsack

State Transition Equation:

Unbounded Knapsack

State Transition Equation:

$$\mathrm{DP[i][j]} = \max(\mathrm{DP}[i-1][j], \mathrm{DP}[i][j - w_i] + \mathrm{v}_i)$$

Review

Unbounded Knapsack:

$$\text{DP[i][j]} = \max(\text{DP}[i-1][j], \text{DP}[i][j - w_i] + \text{v}_i)$$

0-1 Knapsack:

$$\text{DP[i][j]} = \max(\text{DP}[i-1][j], \text{DP}[i-1][j - w_i] + \text{v}_i)$$

Content

1. Introduction to DP
2. 0-1 Knapsack
3. Unbounded Knapsack UKP
4. **Output solution, no. of solution**

Output Solution

We have to record how a certain state in the backpack is derived →

Use another bool array G[i][v] to record whether the i^{th} item is chosen when the knapsack has v remaining capacity →

Loop from the last item till the first item, $v\ -= w_i$ if it is chosen.

Output Number of Solution

Easy.

Change the max function to sum.

For 0-1 knapsack:

$$\mathrm{DP}[\mathrm{i}][\mathrm{j}] = \mathrm{DP}[i-1][j] + \mathrm{DP}[i-1][j-w_i]$$

where DP[0] = 1 (The only sol is put nothing)

Practice Problem

- Z0057 0-1 Knapsack
- Z0058 UKP

Q&A