Week 05: Combination

FanFly

April 5, 2020

Knapsack Problem

Today we are going to introduce a combinatorial optimzation problem, which is called the **knapsack problem**.

Knapsack Problem

Today we are going to introduce a combinatorial optimzation problem, which is called the **knapsack problem**.

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.

Knapsack Problem

Today we are going to introduce a combinatorial optimzation problem, which is called the **knapsack problem**.

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack.

Knapsack Problem

Today we are going to introduce a combinatorial optimzation problem, which is called the **knapsack problem**.

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack.

It is supposed that the weight capacity of knapsack and the weights and values of all items are positive integers.

An Instance of Knapsack Problem

Following is an instance of knapsack problem.

An Instance of Knapsack Problem

Following is an instance of knapsack problem.

Weight Capacity	10

Item	Weight	Value
A	2	5
B	3	7
C	5	10
D	8	19

An Instance of Knapsack Problem

Following is an instance of knapsack problem.

Weight Capacity	10

Item	Weight	Value
A	2	5
B	3	7
C	5	10
D	8	19

It can be found that the most valuable combination is A and D, whose total value is 24.

An Exponential Time Algorithm

Since the number of all possible combinations is finite, we can perform a brute force algorithm.

An Exponential Time Algorithm

Since the number of all possible combinations is finite, we can perform a brute force algorithm.

- There are 2^n different possible combinations of items.

An Exponential Time Algorithm

Since the number of all possible combinations is finite, we can perform a brute force algorithm.

- There are 2^n different possible combinations of items.
- For each combination, we need $O(n)$ time to compute the total weight and the total value.

An Exponential Time Algorithm

Since the number of all possible combinations is finite, we can perform a brute force algorithm.

- There are 2^n different possible combinations of items.
- For each combination, we need $O(n)$ time to compute the total weight and the total value.
- Thus, there exists an $O(2^n n)$-time algorithm.

An Exponential Time Algorithm

Since the number of all possible combinations is finite, we can perform a brute force algorithm.

- There are 2^n different possible combinations of items.
- For each combination, we need $O(n)$ time to compute the total weight and the total value.
- Thus, there exists an $O(2^n n)$-time algorithm.

The function $f(n) = 2^n n$ grows really fast as n increases, so it can only be used for small n.

n	1	5	10	50	100
$f(n)$	2	160	10240	5.6×10^{16}	1.3×10^{32}

Dynamic Programming

If we want to solve the knapsack problem for big n, we can use dynamic programming!

Dynamic Programming

If we want to solve the knapsack problem for big n, we can use dynamic programming!

Let $V(W, S)$ be the maximum value when

- the weight capacity is W, and
- the items can only be chosen from the set S.

Dynamic Programming

If we want to solve the knapsack problem for big n, we can use dynamic programming!

Let $V(W, S)$ be the maximum value when

- the weight capacity is W, and
- the items can only be chosen from the set S.

How to determine $V(W, S \cup \{k\})$?

Dynamic Programming

If we want to solve the knapsack problem for big n, we can use dynamic programming!

Let $V(W, S)$ be the maximum value when

- the weight capacity is W, and
- the items can only be chosen from the set S.

How to determine $V(W, S \cup \{k\})$?

- If item k is chosen, then we can choose items from S with weight up to $W - w_k$.

Dynamic Programming

If we want to solve the knapsack problem for big n, we can use dynamic programming!

Let $V(W, S)$ be the maximum value when

- the weight capacity is W, and
- the items can only be chosen from the set S.

How to determine $V(W, S \cup \{k\})$?

- If item k is chosen, then we can choose items from S with weight up to $W - w_k$.
- If item k is not chosen, then we can choose items from S with weight up to W.

Dynamic Programming

If we want to solve the knapsack problem for big n, we can use dynamic programming!

Let $V(W, S)$ be the maximum value when

- the weight capacity is W, and
- the items can only be chosen from the set S.

How to determine $V(W, S \cup \{k\})$?

- If item k is chosen, then we can choose items from S with weight up to $W - w_k$.
- If item k is not chosen, then we can choose items from S with weight up to W.
- Thus, we have

$$V(W, S \cup \{k\}) = \max\{v_k + V(W - w_k, S), V(W, S)\}.$$

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2				
3				
4				
5				
6				
7				
8				
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3				
4				
5				
6				
7				
8				
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4				
5				
6				
7				
8				
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4	5	7	7	7
5				
6				
7				
8				
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4	5	7	7	7
5	5	12	12	12
6				
7				
8				
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4	5	7	7	7
5	5	12	12	12
6	5	12	12	12
7				
8				
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4	5	7	7	7
5	5	12	12	12
6	5	12	12	12
7	5	12	15	15
8				
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4	5	7	7	7
5	5	12	12	12
6	5	12	12	12
7	5	12	15	15
8	5	12	17	19
9				
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4	5	7	7	7
5	5	12	12	12
6	5	12	12	12
7	5	12	15	15
8	5	12	17	19
9	5	12	17	19
10				

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4	5	7	7	7
5	5	12	12	12
6	5	12	12	12
7	5	12	15	15
8	5	12	17	19
9	5	12	17	19
10	5	12	17	24

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4	5	7	7	7
5	5	12	12	12
6	5	12	12	12
7	5	12	15	15
8	5	12	17	19
9	5	12	17	19
10	5	12	17	24

Dynamic Programming (cont.)

	A	B	C	D
Weight	2	3	5	8
Value	5	7	10	19

	A	B	C	D
0	0	0	0	0
1	0	0	0	0
2	5	5	5	5
3	5	7	7	7
4	5	7	7	7
5	5	12	12	12
6	5	12	12	12
7	5	12	15	15
8	5	12	17	19
9	5	12	17	19
10	5	12	17	24

Conclusion

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack.

Conclusion

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack.

For the knapsack problem, we have found a brute force algorithm and a dynamic programming algorithm.

Conclusion

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack.

For the knapsack problem, we have found a brute force algorithm and a dynamic programming algorithm.

- The brute force algorithm runs in $\Theta(2^n n)$ time.

Conclusion

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack.

For the knapsack problem, we have found a brute force algorithm and a dynamic programming algorithm.

- The brute force algorithm runs in $\Theta(2^n n)$ time.
- The dynamic programming algorithm runs in $\Theta(nC)$ time.

Conclusion

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack.

For the knapsack problem, we have found a brute force algorithm and a dynamic programming algorithm.

- The brute force algorithm runs in $\Theta(2^n n)$ time.
- The dynamic programming algorithm runs in $\Theta(nC)$ time.
  - There are nC entries to compute.

Conclusion

Knapsack Problem

- Input:
    - A set of n items, each with a weight w_i and a value v_i.
    - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack.

For the knapsack problem, we have found a brute force algorithm and a dynamic programming algorithm.

- The brute force algorithm runs in $\Theta(2^n n)$ time.
- The dynamic programming algorithm runs in $\Theta(nC)$ time.
    - There are nC entries to compute.
    - For each entry, we only need $\Theta(1)$ time to get the result.

Conclusion (cont.)

Is $\Theta(nC)$ considered polynomial?

Conclusion (cont.)

Is $\Theta(nC)$ considered polynomial? No!

Conclusion (cont.)

Is $\Theta(nC)$ considered polynomial? No!

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C, represented in m bits.
- Output: The most valuable combination of items that fits in the knapsack.

Conclusion (cont.)

Is $\Theta(nC)$ considered polynomial? No!

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C, represented in m bits.
- Output: The most valuable combination of items that fits in the knapsack.

The way to measure input size is really important.

Conclusion (cont.)

Is $\Theta(nC)$ considered polynomial? No!

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C, represented in m bits.
- Output: The most valuable combination of items that fits in the knapsack.

The way to measure input size is really important.

- If we use the number of bits to measure the input size, the time complexity of the dynamic programming method turns into $\Theta(n2^m)$.

Conclusion (cont.)

Is $\Theta(nC)$ considered polynomial? No!

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C, represented in m bits.
- Output: The most valuable combination of items that fits in the knapsack.

The way to measure input size is really important.

- If we use the number of bits to measure the input size, the time complexity of the dynamic programming method turns into $\Theta(n2^m)$.
- Thus, it is in fact an exponential-time algorithm.

Conclusion (cont.)

Is $\Theta(nC)$ considered polynomial? No!

Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C, represented in m bits.
- Output: The most valuable combination of items that fits in the knapsack.

The way to measure input size is really important.

- If we use the number of bits to measure the input size, the time complexity of the dynamic programming method turns into $\Theta(n2^m)$.
- Thus, it is in fact an exponential-time algorithm.
- Since its running time is polynomial in the numeric value of the input, we also say that it runs in **pseudo-polynomial time**.

Exercise #1

We have known that the brute force algorithm runs in $\Theta(2^n n)$ time, while the dynamic programming algorithm runs in $\Theta(nC)$ time.

Exercise #1

We have known that the brute force algorithm runs in $\Theta(2^n n)$ time, while the dynamic programming algorithm runs in $\Theta(nC)$ time.

Please find the necessary and sufficient condition such that $2^n n \leq nC$.
(In this case, the brute force algorithm does not run asymptotically slower than the dynamic programming algorithm.)

Exercise #1

We have known that the brute force algorithm runs in $\Theta(2^n n)$ time, while the dynamic programming algorithm runs in $\Theta(nC)$ time.

Please find the necessary and sufficient condition such that $2^n n \leq nC$.
(In this case, the brute force algorithm does not run asymptotically slower than the dynamic programming algorithm.)

Solution

The necessary and sufficient condition is $C \geq 2^n$.

Exercise #2

We have introduced the knapsack problem.

Exercise #2

We have introduced the knapsack problem.

Now let's introduce its variation, called the unbounded knapsack problem.

Exercise #2

We have introduced the knapsack problem.

Now let's introduce its variation, called the unbounded knapsack problem.

Unbounded Knapsack Problem

- Input:
  - A set of n types of items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack, where the number of each type of items can be any finite integer.

Exercise #2

We have introduced the knapsack problem.

Now let's introduce its variation, called the unbounded knapsack problem.

Unbounded Knapsack Problem

- Input:
  - A set of n types of items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack, where the number of each type of items can be any finite integer.

Can you find any algorithm that solves the unbounded knapsack problem?

Exercise #2 (cont.)

Solution

We can choose C items for each type, and it turns into the knapsack problem with nC items.
Thus, the problem can be solved in $O(nC^2)$ time.

Exercise #3

Another variation of knapsack problem is as follows.

Exercise #3

Another variation of knapsack problem is as follows.

Fractional Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack, where fractions of items can be taken.

Exercise #3

Another variation of knapsack problem is as follows.

Fractional Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack, where fractions of items can be taken.

Please find an algorithm that solves the fractional knapsack problem.

Exercise #3

Another variation of knapsack problem is as follows.

Fractional Knapsack Problem

- Input:
  - A set of n items, each with a weight w_i and a value v_i.
  - A knapsack with weight capacity C.
- Output: The most valuable combination of items that fits in the knapsack, where fractions of items can be taken.

Please find an algorithm that solves the fractional knapsack problem.

Solution

The problem can be solved by continually choosing the most valuable items (according to their values per unit weight) until the knapsack is full. If one uses sorting, the problem can be solved in $O(n \log n)$ time.