

SC2001 Unbounded Knapsack Problem

Presented by: Qi Yang, Felicia and Zheng Wei



Presentation Overview



1. Recursive Definition

1. Subproblem graph

1. Bottom-up DP code

1. Running Results

The background features large, abstract shapes in pink, blue, and yellow. Scattered around are various school supplies: a yellow crayon, a pencil with a blue eraser, an orange paperclip, a blue sharpener, and a red dotted line forming a spiral. A white hashtag symbol is also visible on the yellow background.

01

Recursive Definition



Understanding how to solve the Problem

GENERIC EXAMPLE



profit	weight	index	0	1	2	3	4	5	6	7	8
15	1	0	0	15	30	45	60	75	90	105	120
50	3	1	0	15	30	50	65	80	100	115	130
60	4	2	0	15	30	50	65	80	100	115	130
90	5	3	0	15	30	50	65	90	105	120	140

Only
considers
combinations
for these
items

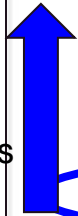


GENERIC EXAMPLE



profit	weight	index	0	1	2	3	4	5	6	7	8
15	1	0	0	15	30	45	60	75	90	105	120
50	3	1	0	15	30	50	65	80	100	115	130
60	4	2	0	15	30	50	65	80	100	115	130
90	5	3	0	15	30	50	65	90	105	120	140

Only considers combinations for these items



Possibilities to consider given capacity C and object of index n



1. Number of object of index n in optimal solution = 0
2. Number of object of index n in optimal solution ≥ 1



$P(n-1, C)$



$\text{Profit}[n] + P(n, C - \text{Weight}[n])$

Take
Maximum
value

GENERIC EXAMPLE

Our Base Cases

Weight of First item > Bag capacity

profit	weight	index	0	1	2	3	4	5	6	7	8
15	1	0	0	15	30	45	60	75	90	105	120
50	3	1	0	15	30	50	65	80	100	115	130
60	4	2	0	15	30	50	65	80	100	115	130
90	5	3	0	15	30	50	65	90	105	120	140

Bag Capacity=0

GENERIC EXAMPLE

Our Base Cases

Weight of First item > Bag capacity

profit	weight	index	0	1	2	3	4	5	6	7	8
15	3	0	0	0	0	45	60	75	90	105	120
50	3	1	0	15	30	50	65	80	100	115	130
60	4	2	0	15	30	50	65	80	100	115	130
90	5	3	0	15	30	50	65	90	105	120	140

Bag Capacity=0

Q1 Recursive Definition:

For N objects, N-1 is our Maximum object index, hence we have the following:

Let $P(n-1, C)$ be the maximum profit that can be made by selecting any combination of the n objects with knapsack capacity of C .

Base cases

$$P(0, 0) = P(1, 0) = P(2, 0) \dots = P(n-1, 0) = 0$$

$$P(0, 0) = P(0, 1) \dots P(0, \text{weight}[0] - 1) = 0$$

Recursive Definition

$$P(n-1, C) = \max(P(n-2, C), \text{profit}[n-1] + P(n-1, C - \text{weight}[n-1]))$$

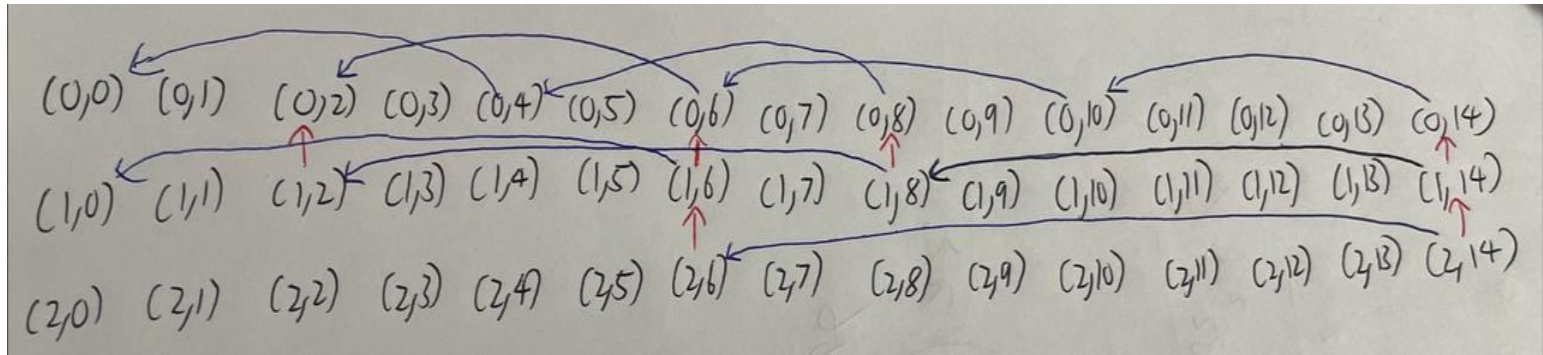
The background features large, abstract shapes in pink, blue, and yellow. Various school supplies are illustrated: a yellow crayon, a pencil, a paperclip, and a sharpener. A red dotted line forms a loop in the bottom right corner. A small white grid pattern is visible on the yellow background on the right.

02

Subproblem Graph

Q2 Subproblem graph for P(14) given the following items:

	0	1	2
w_i	4	6	8
p_i	7	6	9



The background features large, abstract shapes in pink, blue, and yellow. Scattered around are illustrations of school supplies: a yellow crayon, a pencil with a blue eraser, an orange paperclip, and a blue stapler. A red dotted line connects the paperclip to the stapler. A white hashtag symbol is visible on the yellow background on the right.

03

Algorithm

```
// Unbounded Knapsack function
public static int[][] Profit(int Capacity, int NoOfObjType, int[] weight, int[] profit){
    int[][] P = new int[NoOfObjType][Capacity + 1];

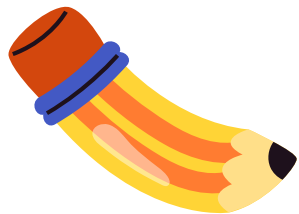
    for (int i = 0; i < NoOfObjType; i++) { //Initialise all first row, since when capacity=0, profit=0.
        P[i][0]=0;
    }

    for (int i = 0; i < NoOfObjType; i++) {
        for (int j = 1; j <= Capacity; j++) { //Start from 1 since first column alr initialised
            if (weight[i] > j)
            {
                //if first row, initialise to 0. If other rows, they can read from above.
                if(i==0)
                {
                    P[i][j]=0;
                }
                else
                {
                    P[i][j] = P[i-1][j];
                }
            }
        }
    }
}
```



```
    else
    {
        //If first row, can't compare with index above, so only get the leftSide case.
        if(i==0)
        {
            P[i][j]=P[i][j - weight[i]] + profit[i];
        }
        else
        {
            P[i][j] = Math.max(P[i-1][j], P[i][j - weight[i]] + profit[i]);
        }
    }
}
return P;
}
```





04

Results

a) Result of P(14): object table 1

i	0	1	2
W _i	4	6	8
p _i	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
7		4	0	0	0	0	7	7	7	7	14	14	14	14	21	21	21
6		6	0	0	0	0	7	7	7	7	14	14	14	14	21	21	21
9		8	0	0	0	0	7	7	7	7	14	14	14	14	21	21	21

a) Result of P(14): object table 1, steps 1-2

i	0	1	2
Wi	4	6	8
pi	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0													
6		6	0													
9		8	0													

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0										
6		6	0													
9		8	0													

```
for (int i = 0; i < NoOfObjType; i++) { //Initialise all
//first col, since when capacity=0, profit=0.
    P[i][0]=0;
}
```

```
for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
// first column alr initialised
        if (weight[i] > j)
        {
            //if first row, initialise to 0.
            //If other rows, they can read from above.
            if(i==0)
            {
                P[i][j]=0;
            }
        }
    }
}
```

a) Result of P(14): object table 1, step 3

i	0	1	2
Wi	4	6	8
pi	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	7										
6		6	0													
9		8	0													

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	7	7	7	7							
6		6	0													
9		8	0													

```
for (int i = 0; i < NoOfObjType; i++) {  
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since  
        // first column alr initialised  
        if (weight[i] > j)  
        {  
            ...  
        }  
        else  
        {  
            //If first row, can't compare with index above,  
            // so only get the leftSide case.  
            if(i==0)  
            {  
                P[i][j]=P[i][j - weight[i]] + profit[i];  
            }  
        }  
    }  
}
```

a) Result of P(14): object table 1, step 3(cont)

i	0	1	2
Wi	4	6	8
pi	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0	7	7	7	14	14	14	14			
6		6	0													
9		8	0													

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	7	7	7	7	14	14	14	14	21	21	21
6		6	0													
9		8	0													

```

for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        {
            ...
        }
        else
        {
            //If first row, can't compare with index above,
            // so only get the leftSide case.
            if(i==0)
            {
                P[i][j]=P[i][j - weight[i]] + profit[i];
            }
        }
    }
}
    
```

a) Result of P(14): object table 1, step 4

i	0	1	2
W _i	4	6	8
p _i	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0	7	7	7	7	14	14	14	14	21	21
6		6	0	0	0	0										
9		8	0													

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0	7	7	7	14	14	14	14	21	21	21
6		6	0	0	0	0	7	7								
9		8	0													

```

for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        {
            //if first row, initialise to 0.
            //If other rows, they can read from above.
            if(i==0)
            { ...
            else
            {
                P[i][j] = P[i-1][j];
            }
        }
    }
}

```

a) Result of P(14): object table 1, step 5

i	0	1	2
Wi	4	6	8
pi	7	6	9

$$\max(7, 6) = 7$$

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0	7	7	7	14	14	14	14	21	21	21
6		6	0	0	0	0	7	7	7							
9		8	0													

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0	7	7	7	14	14	14	14	21	21	21
6		6	0	0	0	0	7	7	7							
9		8	0													

```

for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        { ...
        }
        else
        { //If first row, can't compare with index above,
          // so only get the leftSide case.
            if(i==0)
            { ...
            }
            else
            {
                P[i][j] = Math.max(P[i-1][j], P[i][j - weight[i]] + profit[i]);
            }
        }
    }
}
    
```

a) Result of P(14): object table 1, step 5(cont)

i	0	1	2
Wi	4	6	8
pi	7	6	9

$$\max(21, 20) = 21$$

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0	7	7	7	7	14	14	14	14	21	21
6		6	0	0	0	0	7	7	7	7	14	14	14	14	21	21
9		8	0													

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0	7	7	7	7	14	14	14	14	21	21
6		6	0	0	0	0	7	7	7	7	14	14	14	14	21	21
9		8	0													

```

for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        { ...
        }
        else
        { //If first row, can't compare with index above,
          // so only get the leftSide case.
            if(i==0)
            { ...
            }
            else
            {
                P[i][j] = Math.max(P[i-1][j], P[i][j - weight[i]] + profit[i]);
            }
        }
    }
}
    
```

a) Result of P(14): object table 1, step 4

i	0	1	2
W _i	4	6	8
p _i	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	7	7	7	7	14	14	14	14	21	21	21
6		6	0	0	0	7	7	7	7	14	14	14	14	21	21	21
9		8	0	0	0	7										

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	7	7	7	7	14	14	14	14	21	21	21
6		6	0	0	0	7	7	7	7	14	14	14	14	21	21	21
9		8	0	0	0	7	7	7	7							

```

for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        {
            //if first row, initialise to 0.
            //If other rows, they can read from above.
            if(i==0)
            { ...
            else
            {
                P[i][j] = P[i-1][j];
            }
        }
    }
}

```


a) Result of P(14): object table 1, step 5

i	0	1	2
Wi	4	6	8
pi	7	6	9

$$\max(21, 16) = 21$$

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0	7	7	7	7	14	14	14	14	21	21
6		6	0	0	0	0	7	7	7	7	14	14	14	14	21	21
9		8	0	0	0	0	7	7	7	7	14	14	14	14	21	21

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		4	0	0	0	0	7	7	7	7	14	14	14	14	21	21
6		6	0	0	0	0	7	7	7	7	14	14	14	14	21	21
9		8	0	0	0	0	7	7	7	7	14	14	14	14	21	21

```

for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        { ...
        }
        else
        { //If first row, can't compare with index above,
          // so only get the leftSide case.
            if (i==0)
            { ...
            }
            else
            {
                P[i][j] = Math.max(P[i-1][j], P[i][j - weight[i]] + profit[i]);
            }
        }
    }
}
    
```

b) Result of P(14): object table 2

i	0	1	2
Wi	5	6	8
pi	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	0	7	7	7	7	7	14	14	14	14
6		6	0	0	0	0	0	7	7	7	7	7	14	14	14	14
9		8	0	0	0	0	0	7	7	7	9	9	14	14	14	16

a) Result of P(14): object table 2, steps 1-2

i	0	1	2
W _i	5	6	8
p _i	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0													
6		6	0													
9		8	0													

```
for (int i = 0; i < NoOfObjType; i++) { //Initialise all
//first col, since when capacity=0, profit=0.
    P[i][0]=0;
}
```

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	0									
6		6	0													
9		8	0													

```
for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
// first column alr initialised
        if (weight[i] > j)
        {
            //if first row, initialise to 0.
            //If other rows, they can read from above.
            if(i==0)
            {
                P[i][j]=0;
            }
        }
    }
}
```

a) Result of P(14): object table 1, step 3

i	0	1	2
Wi	5	6	8
pi	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	0	7	7	7	7					
6		6	0													
9		8	0													

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	7	7	7	7	7	14	14	14	14	14
6		6	0													
9		8	0													

```
for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        {
            // ...
        }
        else
        {
            //If first row, can't compare with index above,
            // so only get the leftSide case.
            if(i==0)
            {
                P[i][j]=P[i][j - weight[i]] + profit[i];
            }
        }
    }
}
```

a) Result of P(14): object table 1, step 4

i	0	1	2
Wi	5	6	8
pi	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	7	7	7	7	7	14	14	14	14	14
6		6	0	0	0	0	0									
9		8	0													

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	7	7	7	7	7	14	14	14	14	14
6		6	0	0	0	0	7									
9		8	0													

```

for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        {
            //if first row, initialise to 0.
            //If other rows, they can read from above.
            if(i==0)
            { ...
            else
            {
                P[i][j] = P[i-1][j];
            }
        }
    }
}

```

a) Result of P(14): object table 1, step 5

i	0	1	2
Wi	5	6	8
pi	7	6	9

$$\max(14, 13) = 14$$

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	7	7	7	7	7	14	14	14	14	14
6		6	0	0	0	0	7	7	7	7	7	14	14	14	14	14
9		8	0													

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	7	7	7	7	7	14	14	14	14	14
6		6	0	0	0	0	7	7	7	7	7	14	14	14	14	14
9		8	0													

```

for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        { ...
        }
        else
        { //If first row, can't compare with index above,
          // so only get the leftSide case.
            if(i==0)
            { ...
            }
            else
            {
                P[i][j] = Math.max(P[i-1][j], P[i][j - weight[i]] + profit[i]);
            }
        }
    }
}
    
```

a) Result of P(14): object table 1, step 4

i	0	1	2
Wi	5	6	8
pi	7	6	9

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	0	7	7	7	7	7	14	14	14	14
6		6	0	0	0	0	0	7	7	7	7	7	14	14	14	14
9		8	0	0	0	0	0									

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	0	7	7	7	7	7	14	14	14	14
6		6	0	0	0	0	0	7	7	7	7	7	14	14	14	14
9		8	0	0	0	0	0	7	7	7						

```
for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        {
            //if first row, initialise to 0.
            //If other rows, they can read from above.
            if(i==0)
            { ...
            else
            {
                P[i][j] = P[i-1][j];
            }
        }
    }
}
```

a) Result of P(14): object table 1, step 5

i	0	1	2
Wi	5	6	8
pi	7	6	9

$$\max(14, 16) = 16$$

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	0	7	7	7	7	14	14	14	14	14
6		6	0	0	0	0	0	7	7	7	7	14	14	14	14	14
9		8	0	0	0	0	0	7	7	9	9	14	14	14	16	16

Profit	Weight\Capacity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7		5	0	0	0	0	0	7	7	7	7	14	14	14	14	14
6		6	0	0	0	0	0	7	7	7	7	14	14	14	14	14
9		8	0	0	0	0	0	7	7	9	9	14	14	14	16	16

```

for (int i = 0; i < NoOfObjType; i++) {
    for (int j = 1; j <= Capacity; j++) { //Start from 1 since
        // first column alr initialised
        if (weight[i] > j)
        { ...
        }
        else
        { //If first row, can't compare with index above,
          // so only get the leftSide case.
            if (i==0)
            { ...
            }
            else
            {
                P[i][j] = Math.max(P[i-1][j], P[i][j - weight[i]] + profit[i]);
            }
        }
    }
}
    
```


Thank You!

