

05506005

ใบงาน Dynamic Programming

วัตถุประสงค์ เพื่อ ศึกษา

technique การแก้ปัญหาด้วย DP

Bottom-up computation: Computing the table using

$$V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$$

row by row.

| V[i,w] | w=0 | 1 | 2 | 3 | ... | W |
|--------|-----|---|---|---|-----|---|
| i=0 | 0 | 0 | 0 | 0 | ... | 0 |
| 1 | | | | | | |
| 2 | | | | | | |
| ... | | | | | | |
| n | | | | | | |

bottom
up

ในการแก้ปัญหา knapsack 0,1 ด้วย

dynamic programming เราจะสร้าง

อาร์เรย์เก็บคำตอบ ขนาด

[n+1][weight+1] ซึ่งหมายถึง

knapsack ยั้วาง (ใช้สำหรับกรณี row

ตั้งแต่ 1 เพราะต้องเทียบกับรอบก่อน

หน้า)

โอเดียของการประมวลผลคือ “มูลค่า”

ที่ดีที่สุดของรอบที่ผ่านมาของ

weightSoFar - น้ำหนักของ item นี้

(ไม่ว่าใส่อะไรไว้) หากเปลี่ยนเป็นใส่

item นี้แล้วน้ำหนักกลายเป็นเท่านี้แล้ว

มูลค่ามากกว่า ย่อมหมายถึงว่าได้

solution ที่ดีกว่าที่ผ่านมา (else จึง

บอกว่านั้นเอา solution ก่อน ซึ่งก็คือ

บรรทัดเหนือนั่น)

loop for item เป็นการหิยทีละชิ้น โดย

คำตอบที่ได้เป็นคำตอบที่ถูกต้องเพราะ

item นั้นๆจะถูกเทียบกับ solution ที่

ดีที่สุดของบรรทัดเหนือนั่น ทำให้หากได้

update อาร์เรย solution นั้นย่อม

เป็น solution ที่ดีที่สุดที่รวม item นี้

ด้วย

KnapSack(v, w, n, W)

```
{
  for (w = 0 to W) V[0, w] = 0;
  for (i = 1 to n)
    for (w = 0 to W)
      if (w[i] ≤ w)
        V[i, w] = max{V[i-1, w], v[i] + V[i-1, w-w[i]]};
      else
        V[i, w] = V[i-1, w];
  return V[n, W];
}
```

<https://www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf>

```
int C = 16;
int[] itemsW = { 2, 10, 5, 5 };
int[] itemsV = { 20, 50, 30, 10 };
int[][] table = new int[itemsW.length + 1][C + 1];
int item = 0;
int weightSoFar = 0;
int curWeight = 0;
int curValue = 0;

//initialize row0 and col0 to zero (omitted here)
for (item = 1; item <= itemsW.length; item++) {
  for (weightSoFar = 0;
        weightSoFar <= C; weightSoFar++) {
    curWeight = itemsW[item - 1];
    curValue = itemsV[item - 1];
    table[item][weightSoFar]
      = table[item - 1][weightSoFar];

    if (weightSoFar - curWeight >= 0) {
      if (curValue + table[item - 1][weightSoFar
        - curWeight] > table[item - 1][weightSoFar]) {
        table[item][weightSoFar] = curValue +
          table[item - 1][weightSoFar - curWeight];
      }
      // else println("negative index");
    }
  }
}
for (int i = 0; i < table.length; i++) {
  for (int j = 0; j < table[0].length; j++)
    System.out.print(table[i][j] + " ");
  System.out.println();
}
```

| | | | | | | | | | | | | | | | | | | |
|----|----|-------------|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| | | weightSoFar | | | | | | | | | | | | | | | | |
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| V | W | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 2 | 0 | 0 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| 50 | 10 | 0 | 0 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 50 | 50 | 70 | 70 | 70 | 70 | 70 |
| 30 | 5 | 0 | 0 | 20 | 20 | 20 | 30 | 30 | 50 | 50 | 50 | 50 | 50 | 70 | 70 | 70 | 70 | 70 |
| 10 | 5 | 0 | 0 | 20 | 20 | 20 | 30 | 30 | 50 | 50 | 50 | 50 | 50 | 70 | 70 | 70 | 70 | 70 |

สังเกตผลที่ได้หลังจากพิจารณาไป สอง item เมื่อพิจารณาเฉพาะ item แรก มูลค่าที่ดีที่สุดคือ 20 ตั้งแต่น้ำหนักเป็น 2 (ดังนั้นดีที่สุดหากพิจารณาเฉพาะ item นี้) และ เมื่อพิจารณา item ถัดมา โปรแกรมก็พบว่าที่ น้ำหนัก 10 ใส่ item 2 ดีกว่า item แรกขึ้นเดียว และที่ weightSoFar เป็น 12 ก็คือใส่ทั้ง 2 item

คำสั่ง

คำนวณค่าที่เหลือ

กำหนดส่ง TBA