

# CS101

A Mars rover, likely a Curiosity rover, is shown on a rocky, reddish-brown landscape. The rover is white with various instruments and cameras. It has six large, treaded wheels. The background shows a hazy, orange-tinted sky and distant hills. The overall scene is a typical Mars surface environment.

信奥  
算法

课件下载地址:

<http://pan.baidu.com/s/1o885tz0>

作业网站:

<http://120.132.18.213:8080/thrall-web/main#home>

# 01背包问题

## 动态规划

# 作业01 - 01背包简单版

```
1  #include <iostream>
2  #define MAXC 1001
3  #define MAXN 501
4  using namespace std;
5  int n,C,w[MAXN],v[MAXN],f[MAXN][MAXC];
6  int main(){
7      cin>>C>>n;
8      for(int i=1;i<=n;++i) cin>>w[i]>>v[i];
9      // f[i][j]代表从前i件物品中选, 背包载重不超过j, 最大价值
10     for (int i=1;i<=n;++i)    //循环查看物品i
11         for (int j=0;j<=C;++j)    //循环查看背包剩余重量j
12             if(j<w[i])    // 物品i太重, 无法放入
13                 f[i][j]=f[i-1][j];
14             else    // 比较两种决策: 物品i可以放, 或者不放
15                 f[i][j]=max(f[i-1][j],f[i-1][j-w[i]]+v[i]);
16     cout<<f[n][C]<<endl;
17     return 0;
18 }
```

# 作业02 - 采药

```
1  #include<iostream>
2  #define MAXT 1005
3  #define MAXM 105
4  using namespace std;
5  int t[MAXM],v[MAXM],f[MAXM][MAXT],i,j,T,M;
6  int main(){
7      cin>>T>>M;
8      for(i=1;i<=M;i++) cin>>t[i]>>v[i];
9      for(i=1;i<=M;i++)
10         for(j=1;j<=T;j++) {
11             f[i][j]=f[i-1][j];
12             if(t[i]<=j)
13                 f[i][j]=max(f[i][j],f[i-1][j-t[i]]+v[i]);
14         }
15     cout<<f[M][T]<<endl;
16     return 0;
17 }
```

# 作业03

---

# 背包问题 代码优化

承重  
 $j-w_i$

承重  
 $j$

第 $i-1$ 种财物

...  $f[i-1][j-w_i]$  ...  $f[i-1][j]$

第 $i$ 种财物

$f[i][j]$

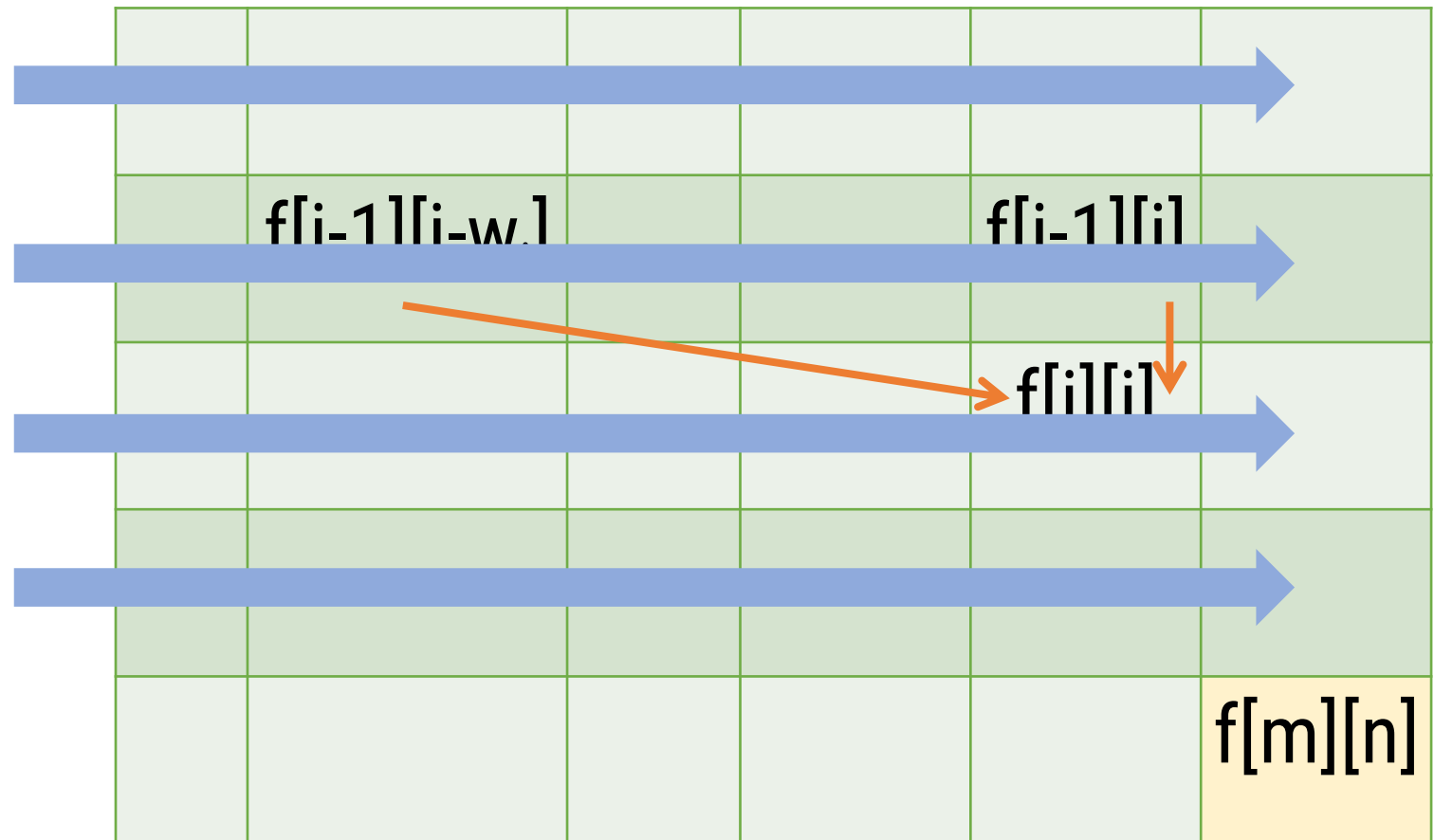
$f[m][n]$



# 01背包 - M\*N表格

从上往下填每一行

每一行内：从左往右填每一列



# 01背包 - M\*N表格

从上往下填每一行

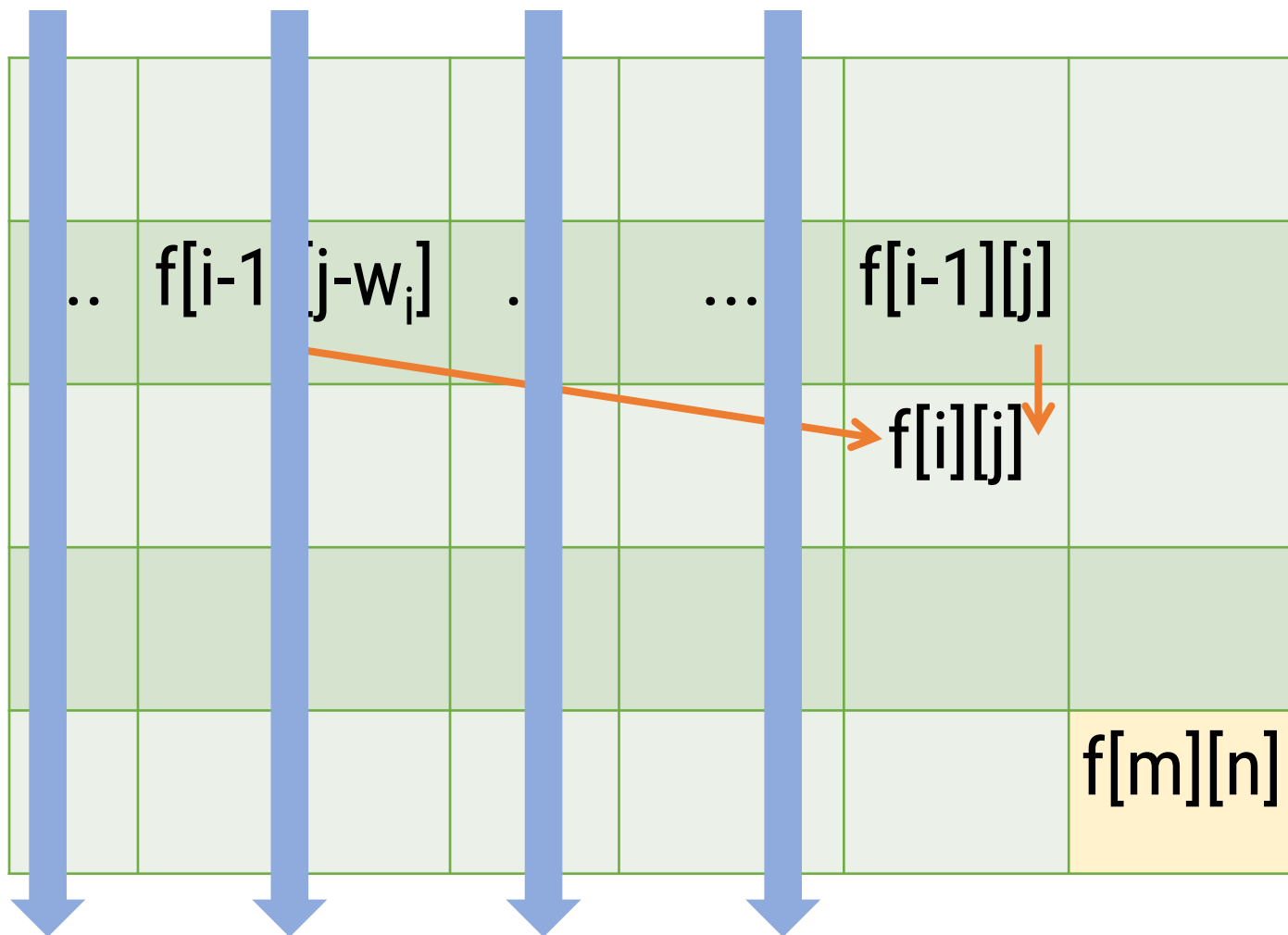
每一行内：从左往右填每一列

```
9  for(int i=1;i<=m;i++)
10  for(int j=0;j<=n;j++) {
11      if(j<w[i])
12          f[i][j]=f[i-1][j];
13      else
14          f[i][j]=max(f[i-1][j],f[i-1][j-w[i]]+v[i]);
15  }
```

# 01背包 - $M \times N$ 表格

从左往右填每一行

每一列内：从上往下填每一行



# 01背包 - M\*N表格

从左往右填每一行

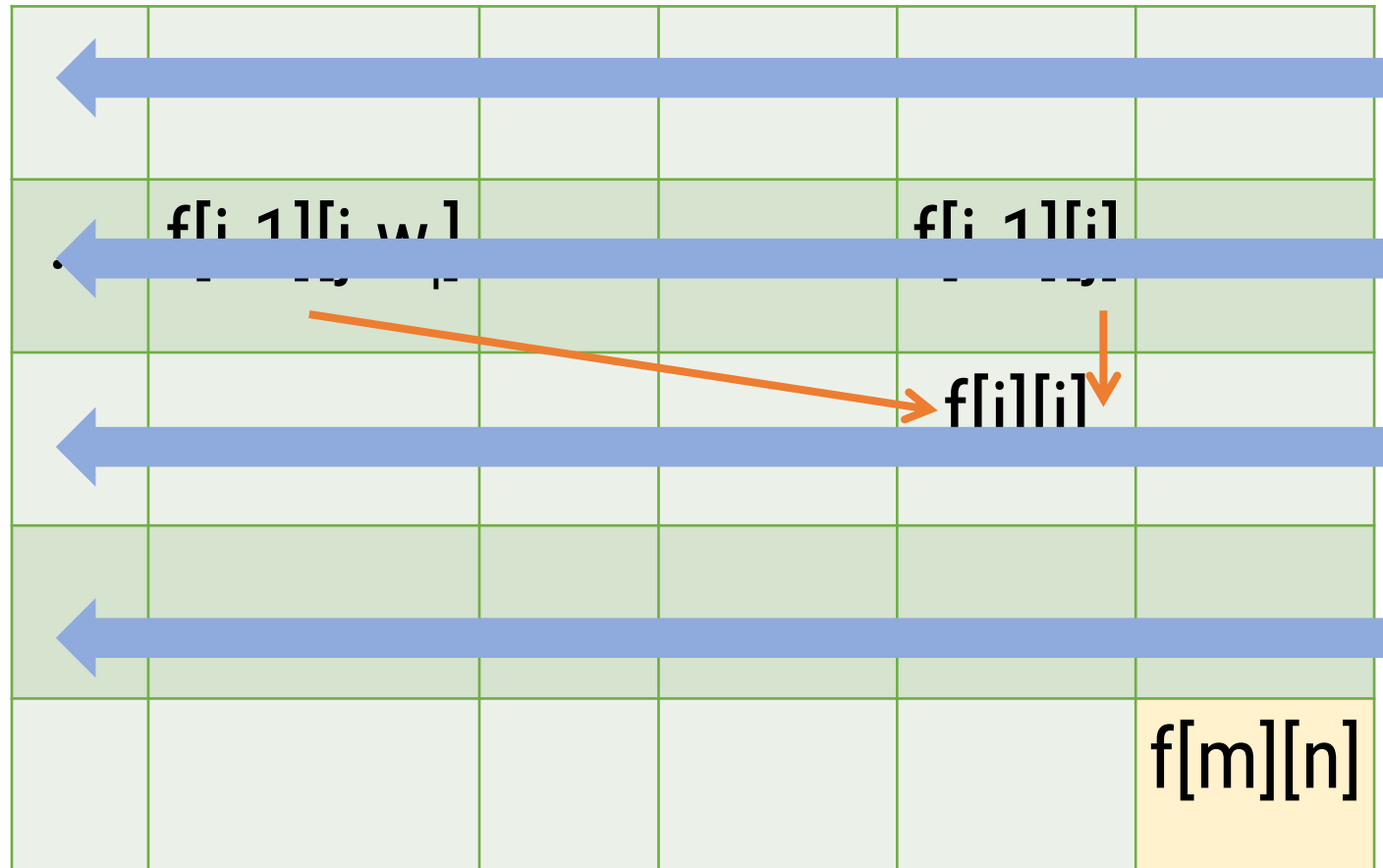
每一列内：从上往下填每一行

```
9  for(int j=0;j<=n;j++) ←
10  for(int i=1;i<=m;i++){ ←
11      if(j<w[i])
12          f[i][j]=f[i-1][j];
13      else
14          f[i][j]=max(f[i-1][j],f[i-1][j-w[i]]+v[i]);
15  }
```

# 01背包 - M\*N表格

从上往下填每一行

每一行内：从右往左填每一列



# 01背包 - M\*N表格

从上往下填每一行

每一行内：从右往左填每一列


```
9  for(int i=1;i<=m;i++)
10  for(int j=n;j>=0;j--) { ←
11      if(j<w[i])
12          f[i][j]=f[i-1][j];
13      else
14          f[i][j]=max(f[i-1][j],f[i-1][j-w[i]]+v[i]);
15  }
```

# 01背包 - 两行表格

只需要两行：当前这一行，和上一行。交替递推

节省储存空间：只需要 $2 \times N$ 的表格

...	$f[i-1][j-w_i]$	...	...	$f[i-1][j]$	
				$f[i][j]$	
					$f[m][n]$



# 01背包 - 两行表格

只需要两行：当前这一行，和上一行。交替递推

节省储存空间：只需要 $2*N$ 的表格

```
5 int n,m,w[M],v[M],f[2][N]; ←
```

```
9 bool r=0;
10 for(int i=1;i<=m;i++)
11     for(int j=n;j>=0;j--) {
12         r=1-r; ←
13         if(j<w[i])
14             f[r][j]=f[1-r][j]; ←
15         else
16             f[r][j]=max(f[1-r][j],f[1-r][j-w[i]]+v[i]);
17     }
```

表格的行数改成 $r$   
不再使用原本行数 $i$



# 01背包 - 两行表格

只需要两行：当前这一行，和上一行。交替递推

节省储存空间：只需要 $2*N$ 的表格

```
5 int n,m,w[M],v[M],f[2][N];

9 bool r=0;
10 for(int i=1;i<=m;i++)
11     for(int j=n;j>=0;j--) {
12         r=1-r;
13         if(j<w[i]) ←
14             f[r][j]=f[1-r][j];
15         else
16             f[r][j]=max(f[1-r][j],f[1-r][j-w[i]]+v[i]);
17     }
```

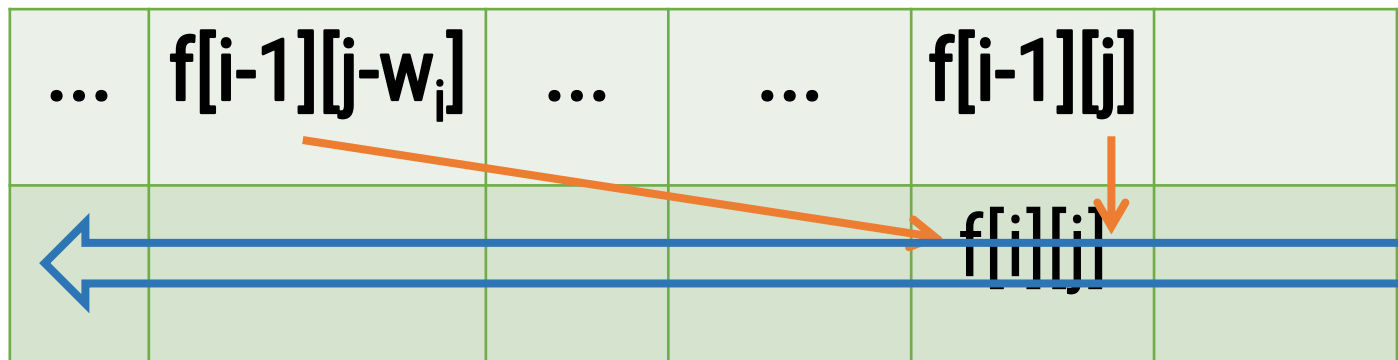
并不是所有i都改成r

# 01背包 - 一行表格

思考：如何只用**1\*N**的表格？

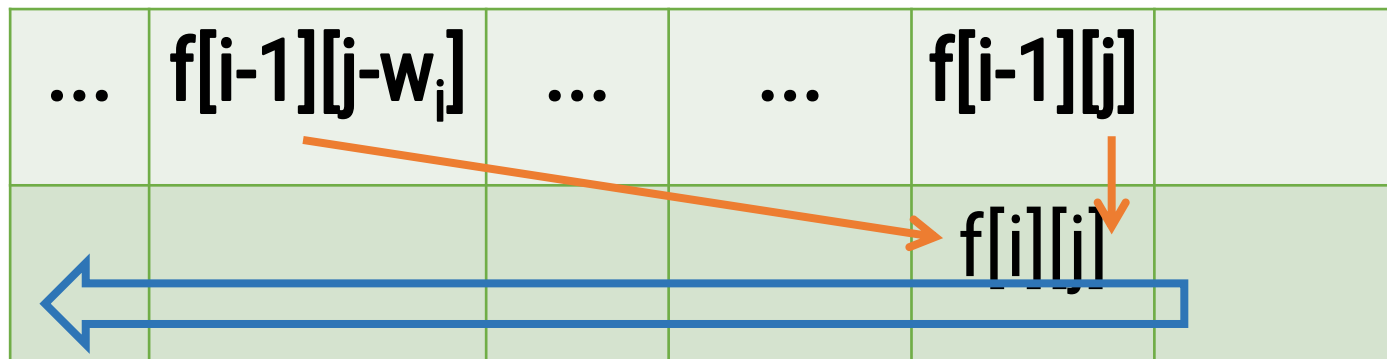
# 01背包 - 一行表格

思考：如何只用**1\*N**的表格？

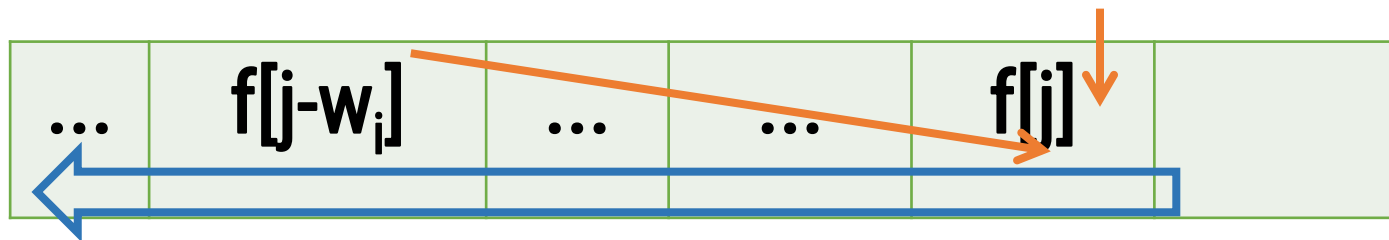


# 01背包 - 一行表格

思考：如何只用**1\*N**的表格？



$f[i][j]$ 的数值可以直接覆盖 $f[i-1][j]$



只用一行，每列只需 **$f[j]$** 存储数值

# 01背包 - 一行表格

思考：如何只用1\*N的表格？

```
1  #include <iostream>
2  #define M 505
3  #define N 2005
4  using namespace std;
5  int n,m,w[M],v[M],f[N];
6  int main(){
7      cin>>n>>m;
8      for(int i=1;i<=m;i++) cin>>w[i]>>v[i];
9      for(int i=1;i<=m;i++)
10         for(int j=n;j>=0;j--) ←
11             if(j<w[i]) ←
12                 f[j]=f[j]; ←
13             else
14                 f[j]=max(f[j],f[j-w[i]]+v[i]);
15     cout<<f[n]<<endl;
16     return 0;
17 }
```

# 01背包 - 一行表格

思考：如何只用1\*N的表格？

```
1 #include <iostream>
2 #define M 505
3 #define N 2005
4 using namespace std;
5 int n,m,w[M],v[M],f[N];
6 int main(){
7     cin>>n>>m;
8     for(int i=1;i<=m;i++) cin>>w[i]>>v[i];
9     for(int i=1;i<=m;i++)
10         for(int j=n;j>=w[i];j--)
11             f[j]=max(f[j],f[j-w[i]]+v[i]);
12     cout<<f[n]<<endl;
13     return 0;
14 }
```

最简版！

# 回顾：01背包问题的变种

给定总的容量/时间/预算限制

有若干种备选物品/草药，每种只能用0次或1次

每次装包/购买/使用，都会消耗/费用

最大化总价值/收益/满足度

# 凑数问题

思考：凑合问题和背包问题的相似点



# 多数凑和 - 总和尽量大

给定一个目标数 $a$ ，和 $n$ 个正整数 $x_1, x_2, \dots, x_n$ ，判断能否找到若干个数总和尽量大但不超过 $a$ 。每个数只能用一次。

输入样例：

10 3

3 4 8

输入样例：

5 2

8 9

输出样例：

8

输出样例：

0

# 多数凑和- 总和尽量大

给定一个目标数 $a$ ，和 $n$ 个正整数 $x_1, x_2, \dots, x_n$ ，判断能否找到若干个数总和尽量大但不超过 $a$ 。每个数只能用一次。

$f[i][j]$ 表示只用前 $i$ 个数中的数求和，  
总和不大于 $j$ 时最大是多少

当 $i$ 为0时 $f[0][j] = 0$

当 $j$ 为0时 $f[i][0] = 0$

$$f[i][j] = \max \{ f[i-1][j], f[i-1][j-x_i] + x_i \}$$

不用第 $i$ 个数

用第 $i$ 个数

# 多数凑和- 判断可行性

给定一个目标数 $a$ ，和 $n$ 个正整数 $x_1, x_2, \dots, x_n$ ，判断能否找到若干个数总和恰巧为 $a$ 。每个数只能用一次。

输入样例：

10 3

3 4 8

输入样例：

10 5

6 5 4 3 2

输出样例：

No

输出样例：

Yes

# 多数凑和- 判断可行性

给定一个目标数 $a$ ，和 $n$ 个正整数 $x_1, x_2, \dots, x_n$ ，判断能否找到若干个数总和恰巧为 $a$ 。每个数只能用一次。

$f[i][j]$ 表示只用前 $i$ 个数中的数求和，  
总和恰巧为 $j$ 是否可以  
可以用1表示，不可以用0表示

当 $i=0, j>0$ 时  $f[0][j] = 0$

当 $j=0$ 时  $f[i][0] = 1$

$$f[i][j] = f[i-1][j] \text{ or } f[i-1][j-x_i]$$

↑  
不用第 $i$ 个数

↑  
用第 $i$ 个数

# 思考题

如何利用“可行性问题”的解答，来回答“总和尽量大”的问题

# 多数凑和- 计数

给定一个目标数 $a$ ，和 $n$ 个正整数，请统计其中若干个数的总和恰巧为 $a$ 有几种可能的组合。**每个数只能用一次。**

输入样例：

10 3  
3 4 8

输入样例：

10 5  
6 5 4 3 2

输出样例：

0

输出样例：

2

# 多数凑和- 计数

给定一个目标数 $a$ ，和 $n$ 个正整数，请统计其中若干个数总和恰巧为 $a$ 有几种可能的组合。**每个数只能用一次。**

$f[i][j]$ 表示只用前 $i$ 个数中的数求和，  
总和恰巧为 $j$ 有几种可能的组合

当 $i=0, j>0$ 时  $f[0][j] = 0$

当 $j=0$ 时  $f[i][0] = 1$

$$f[i][j] = f[i-1][j] + f[i-1][j-x_i]$$

不用第 $i$ 个数

用第 $i$ 个数

# 多数凑和- 总和最接近目标

给定一个目标数 $a$ ，和 $n$ 个正整数，判断能否找到若干个  
数总和尽量接近目标 $a$ 。每个数只能用一次。

输入样例：

10 3

3 4 8

输入样例：

5 2

8 9

输出样例：

11

输出样例：

8

思考题：如何利用“可行性问题”的解答，来回答“总和最接近目标”的问题



# 动态规划常见问题分类

可行性判定问题

计数问题：共几种可能性

最优性问题：求最大/最小

# 棋盘路径

可行性  
问题

在 $n*m$ 的棋盘格上，有一些障碍物用#表示，如果每一步只可以向右走或者向下走一格，请问能否从第1行第1列的位置（左上角）到第 $n$ 行第 $m$ 列的位置（右下角）。

计数性  
问题

在 $n*m$ 的棋盘格上，有一些障碍物用#表示，如果每一步只可以向右走或者向下走一格，请问从第1行第1列的位置（左上角）到第 $n$ 行第 $m$ 列的位置（右下角）一共有几条路径。

最优性  
问题

在 $n*m$ 的棋盘格上，有一些金块，每个用\$表示，如果每一步只可以向右走或者向下走一格，请问从第1行第1列的位置（左上角）到第 $n$ 行第 $m$ 列的位置（右下角）最多捡到多少个金块。