清華大學
Tsinghua University

程序设计基础

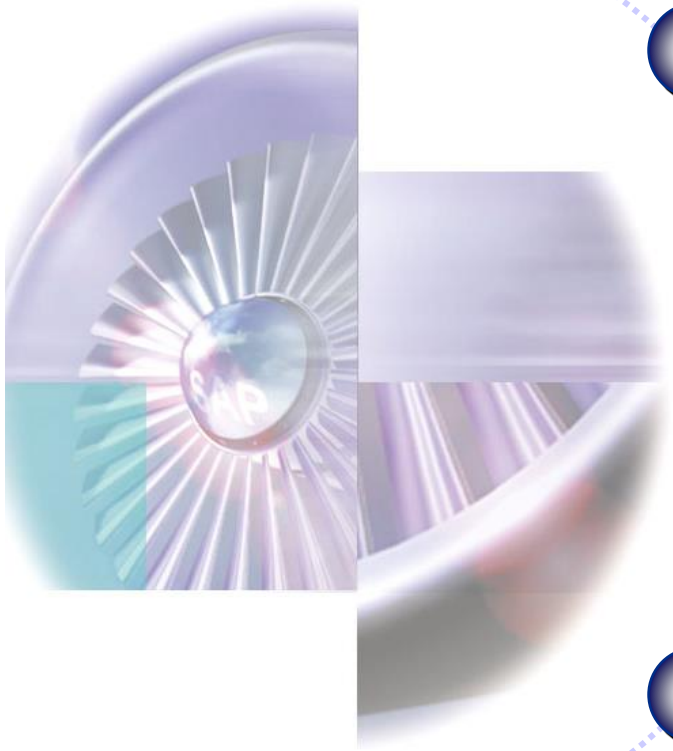
Fundamental of Programming

清华大学软件学院

刘玉身

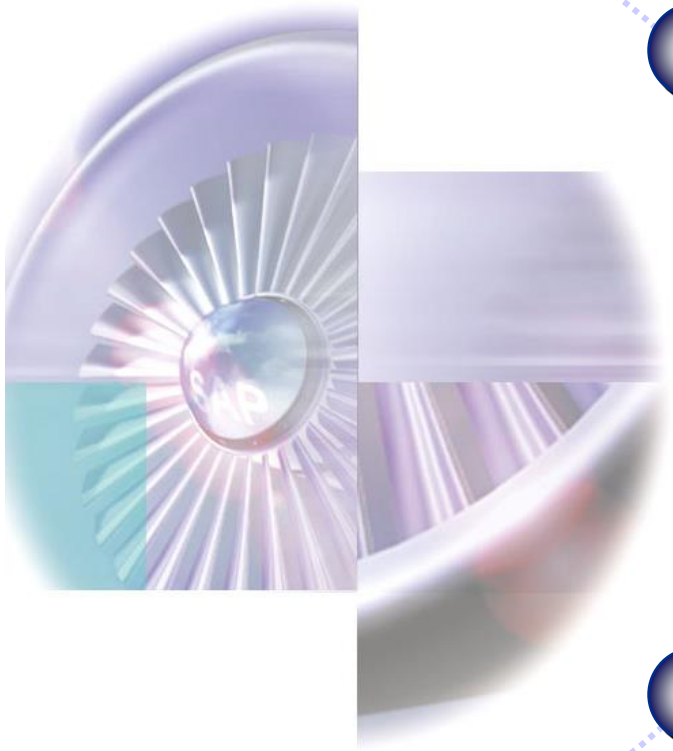
liuyushen@tsinghua.edu.cn

Lecture 10: 动态规划



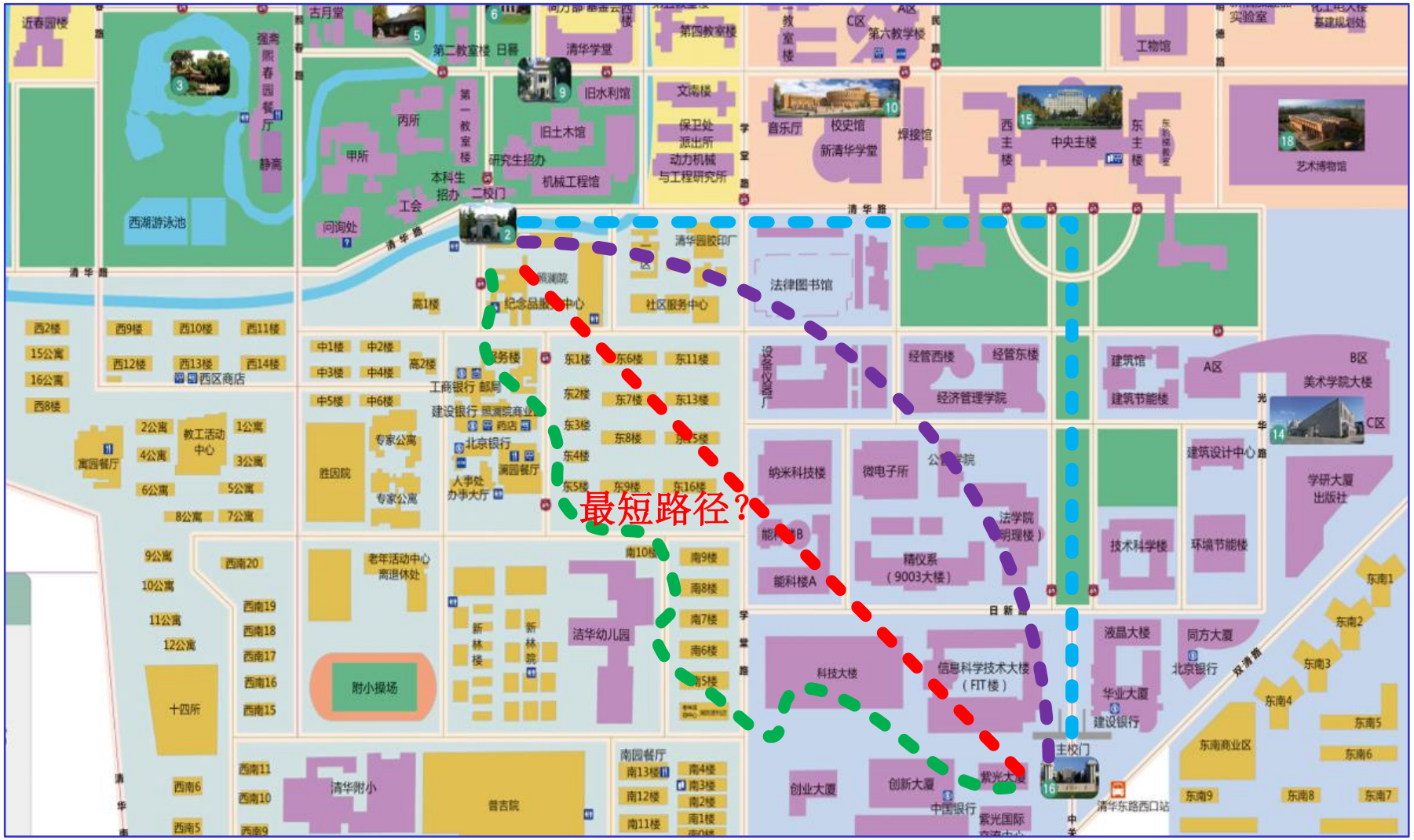
1. 实际问题
2. 动态规划的基本概念
3. 算法举例

Lecture 10: 动态规划



1. 实际问题
2. 动态规划的基本概念
3. 算法举例

实际问题



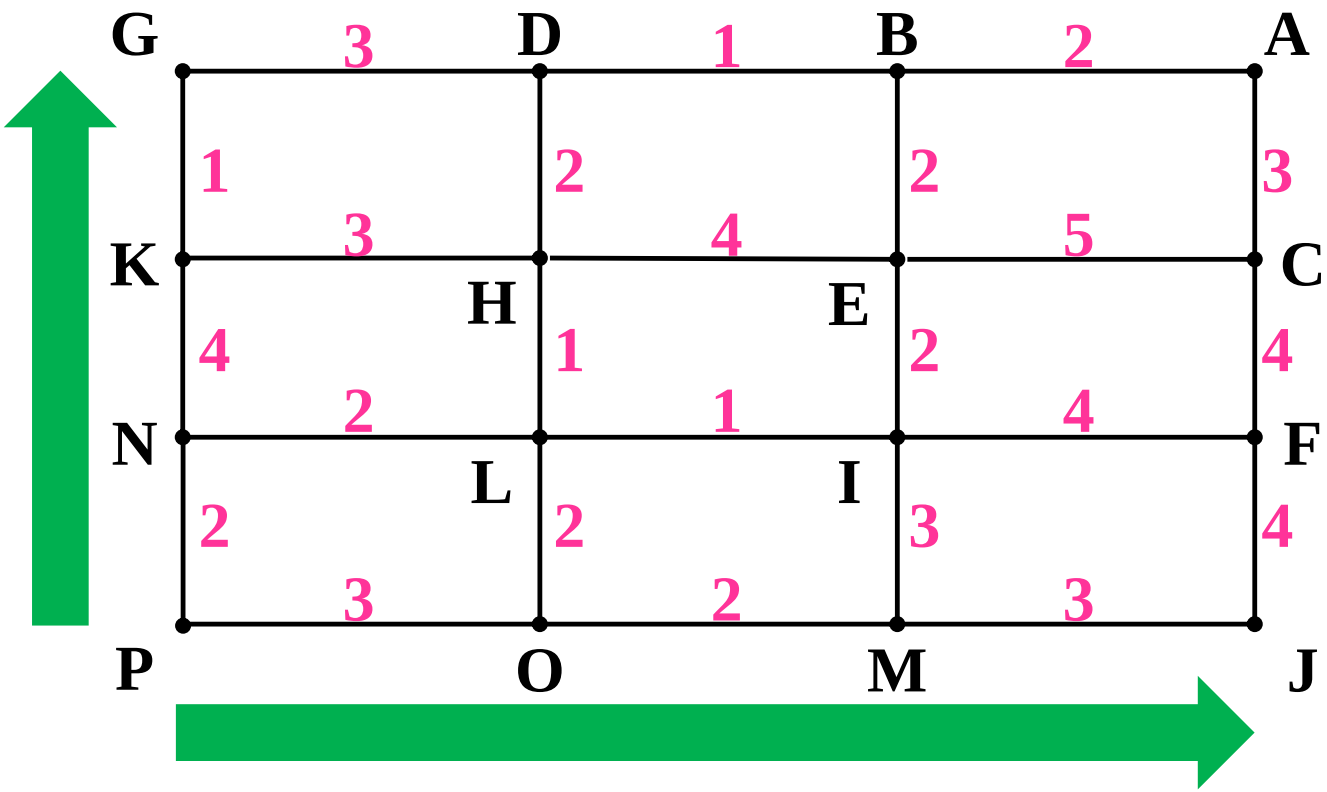
清华大学校园地图 **CAMPUS MAP**

http://www.tsinghua.edu.cn/publish/newthu/newthu_cnt/intothu/picture/map.png

2017年4月版

建立模型

任务：P是出发点，从P到A，求最短路径 (图1)



枚举？

分治？

回溯？

……

要求：单行道，只能向上或向右前进

思路

因为只能向上向右，故可分成5个阶段

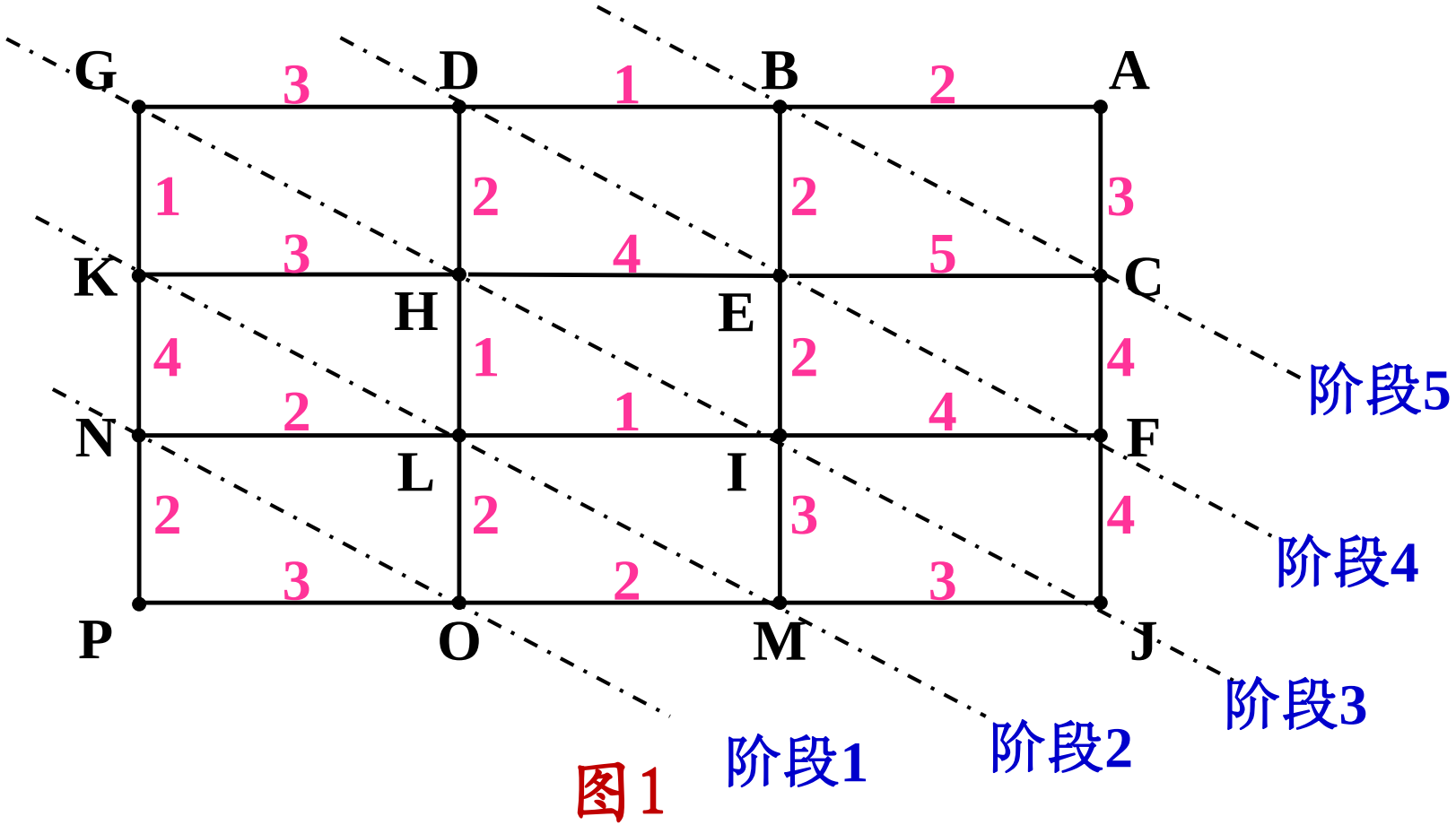
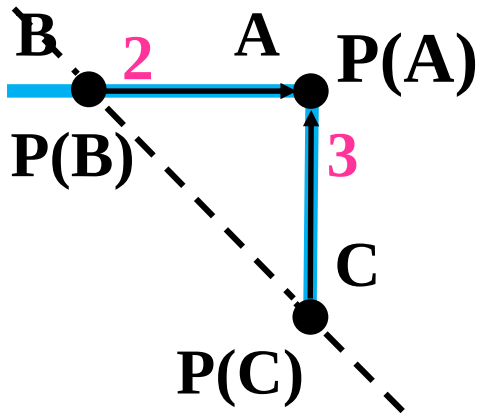


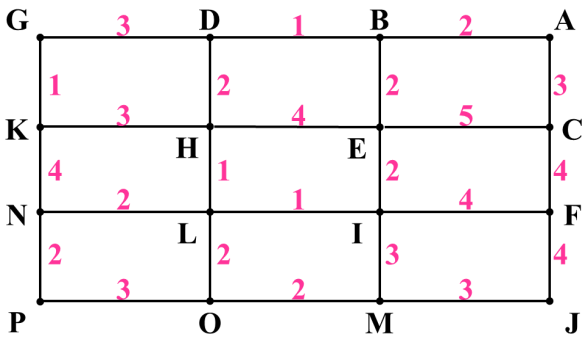
图1

思路

1. 先看第5阶段，到达A点有两条路
   - B → A，需要2km
   - C → A，需要3km
2. 令
   - 从P → A的最短路径为P(A);
   - 从P → B的最短路径为P(B);
   - 从P → C的最短路径为P(C) ……
   - P(A) = min{P(B)+2, P(C)+3};
   - P(B) = min{P(D)+1, P(E)+2};
   - P(C) = min{P(E)+5, P(F)+4};



阶段5

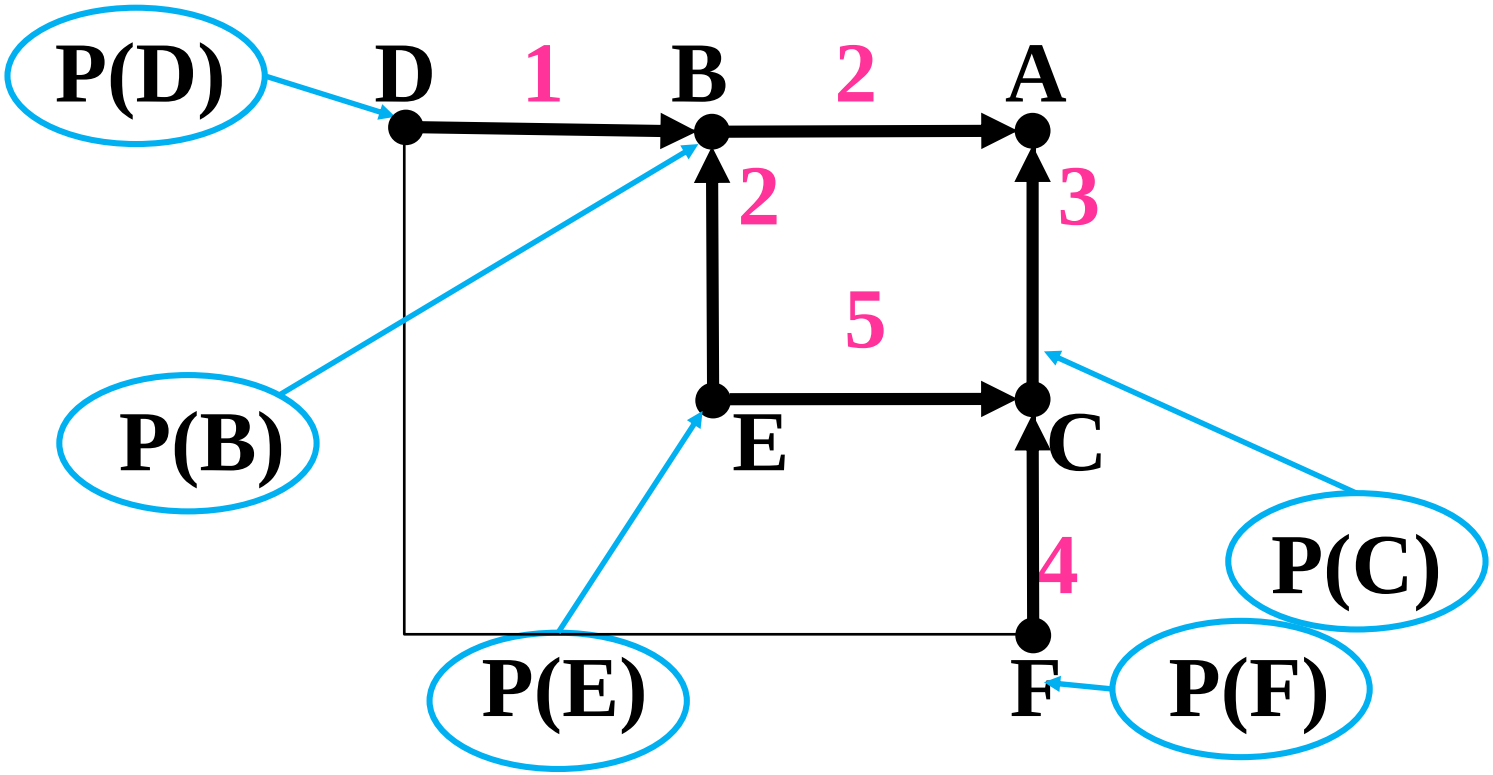


思路

P(A) = min{P(B)+2, P(C)+3};

P(B) = min{P(D)+1, P(E)+2};

P(C) = min{P(E)+5, P(F)+4};



思路

P(A) = min{P(B)+2, P(C)+3};

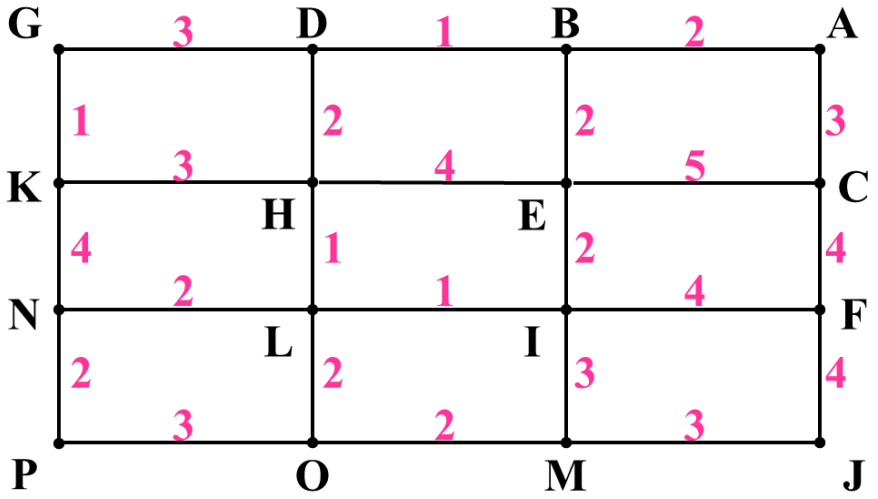
P(B) = min{P(D)+1, P(E)+2};

P(C) = min{P(E)+5, P(F)+4};

⋮

P(N) = 2;

P(O) = 3;



思路

上述递推公式告诉我们，要求P(A)需要先求出阶段5中的P(B)和P(C)；要求 P(B)(或者P(C))，又要先求出阶段4中的 P(D) 和 P(E) (或P(F)和P(E))……

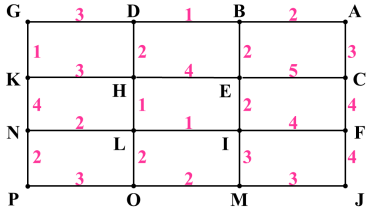
显然，若想照上述递推过程求解，需要倒过来：从P(P)出发，先求出第一阶段的P(O)和P(N)，再求第二阶段的P(K)，P(L)，P(M)，……，最后得到P(A)。

算法思想

3. 选择数据结构，将每条路经的长度存在数组中。

东西方向上的道路长度存在两维数组h[4][3]中规定数组的第一维为行号，第二维为列号。

3	3	1	2
2	3	4	5
1	2	1	4
0	3	2	3
	0	1	2

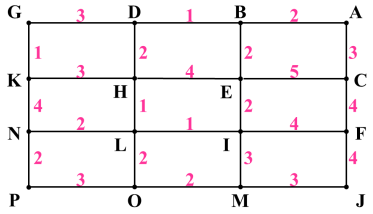


h[4][3] = { {3,2,3}, {2,1,4}, {3,4,5}, {3,1,2} };

算法思想

南北方向上道路长度存至数组v[3][4]中，也规定第一维为行号，第二维为列号。

2	1	2	2	3
1	4	1	2	4
0	2	2	3	4
	0	1	2	3



v[3][4] = {{2, 2, 3, 4},{4, 1, 2, 4},{1, 2, 2, 3}};

算法思想

4. 为了计算方便，使地图更好地与二维数组对应，下面将图1改为图2

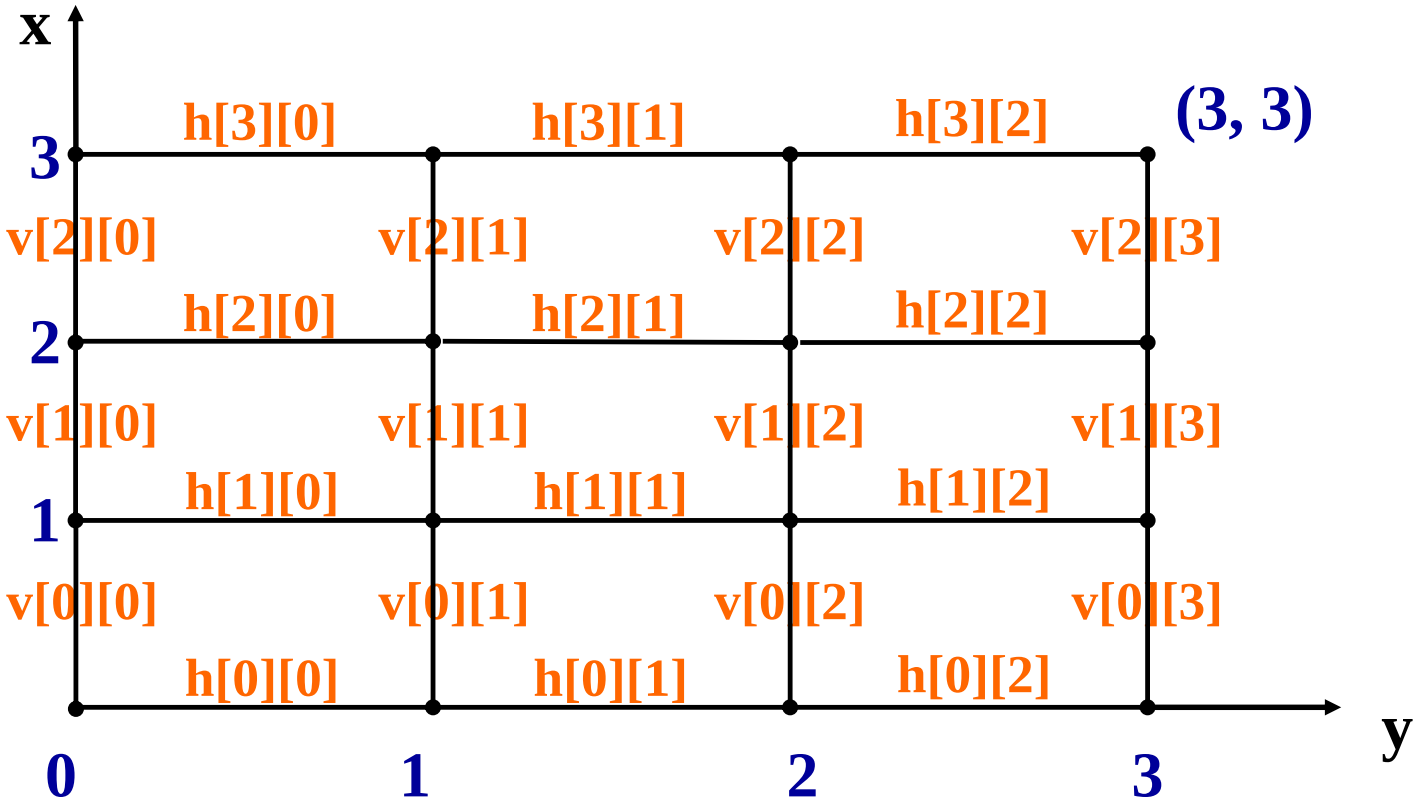


图2

算法思想

5. 求解过程为从(0, 0)到(3, 3)求最短路径问题

定义二维数组，

P[4][4]={{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}},

第一维为行，第二维为列。这时：

P(O)为P[0][1]；P(N)为P[1][0]；…P(A)为P[3][3]，以下为分阶段递推求解过程。

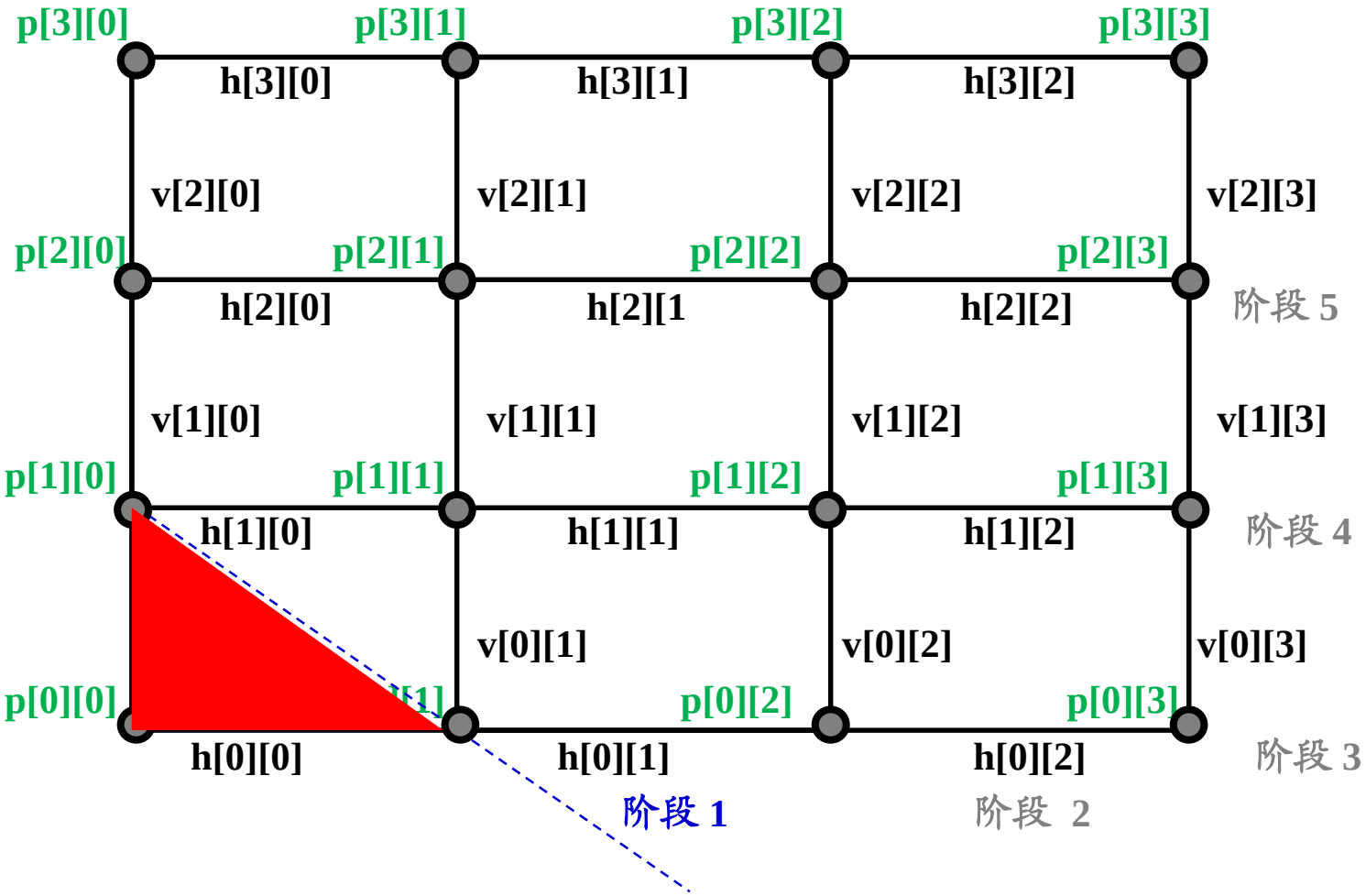
P[0][0] = 0;

对于阶段**1**:

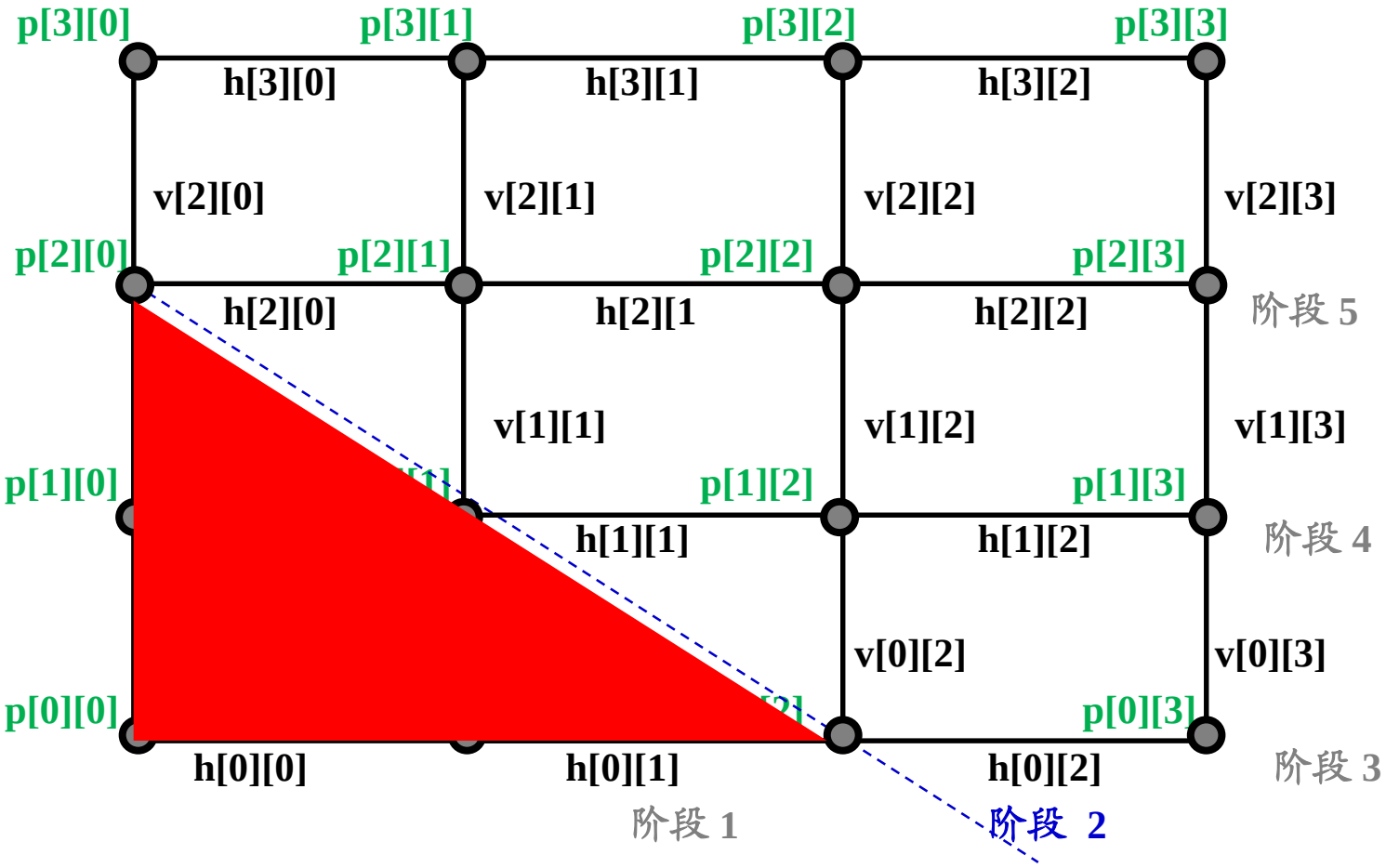
P[0][1] = P[0][0]+h[0][0] = 0+3 = 3;

P[1][0] = P[0][0]+v[0][0] = 0+2 = 2;

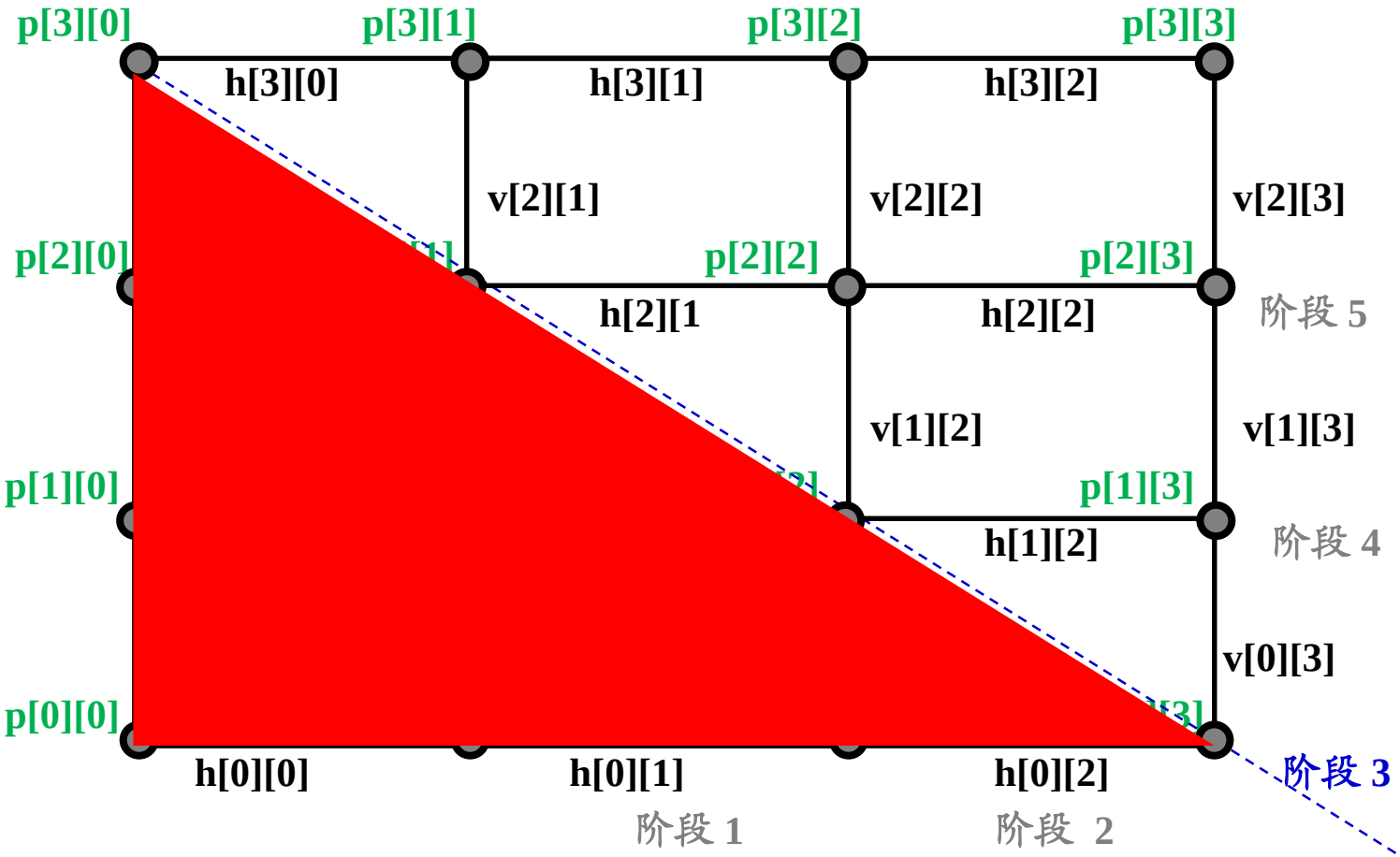
算法流程


p[3][0]
p[3][1]
p[3][2]
p[3][3]
h[3][0]
h[3][1]
h[3][2]
v[2][0]
v[2][1]
v[2][2]
v[2][3]
p[2][0]
p[2][1]
p[2][2]
p[2][3]
h[2][0]
h[2][1
h[2][2]
阶段 5
v[1][0]
v[1][1]
v[1][2]
v[1][3]
p[1][0]
p[1][1]
p[1][2]
p[1][3]
h[1][0]
h[1][1]
h[1][2]
阶段 4
v[0][1]
v[0][2]
v[0][3]
p[0][0]
p[0][2]
p[0][3]
h[0][0]
h[0][1]
h[0][2]
阶段 3
阶段 1
阶段 2

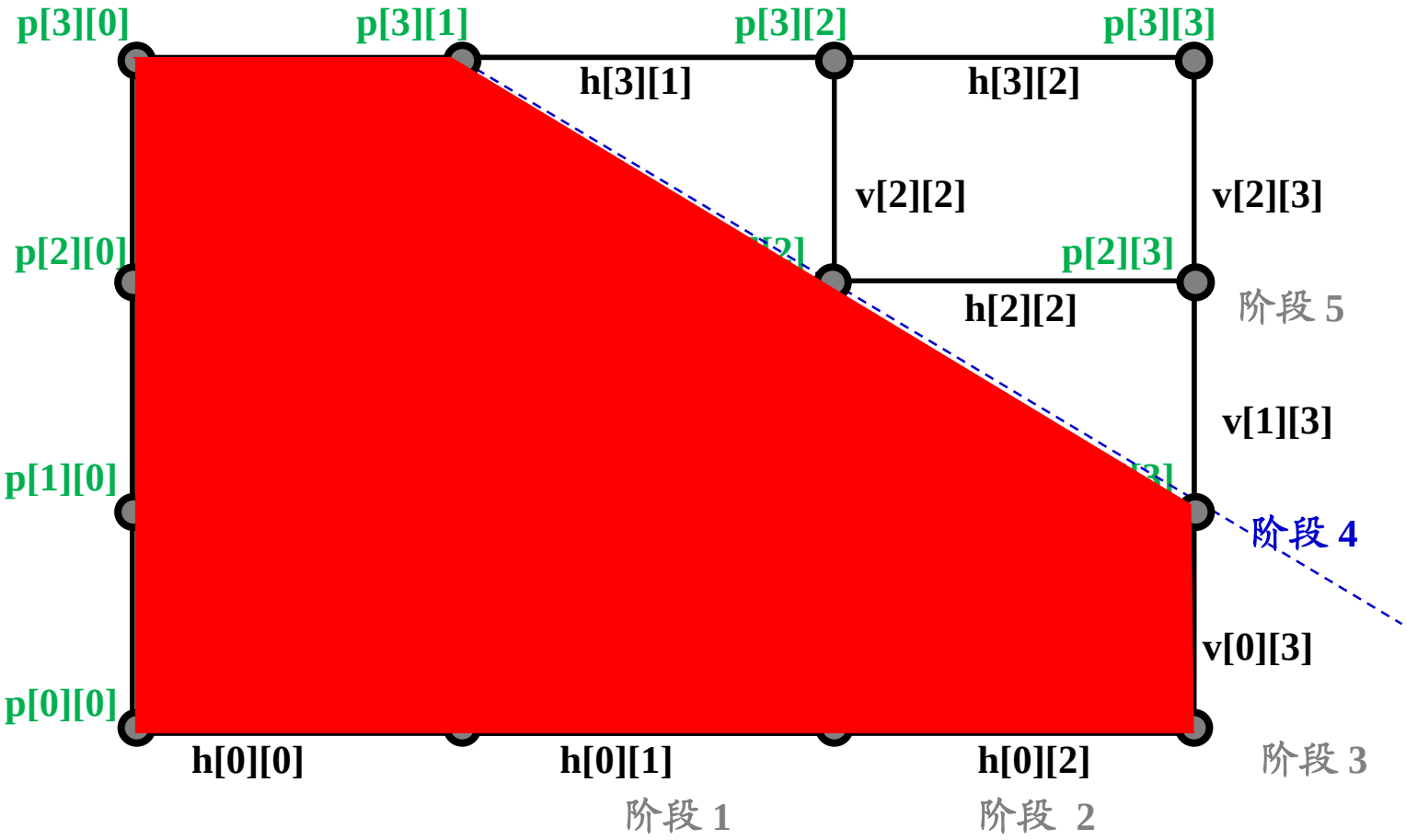
算法流程


p[3][0]
p[3][1]
p[3][2]
p[3][3]
h[3][0]
h[3][1]
h[3][2]
v[2][0]
v[2][1]
v[2][2]
v[2][3]
p[2][0]
p[2][1]
p[2][2]
p[2][3]
h[2][0]
h[2][1
h[2][2]
阶段 5
v[1][1]
v[1][2]
v[1][3]
p[1][0]
p[1][2]
p[1][3]
h[1][1]
h[1][2]
阶段 4
v[0][2]
v[0][3]
p[0][0]
p[0][3]
h[0][0]
h[0][1]
h[0][2]
阶段 3
阶段 1
阶段 2

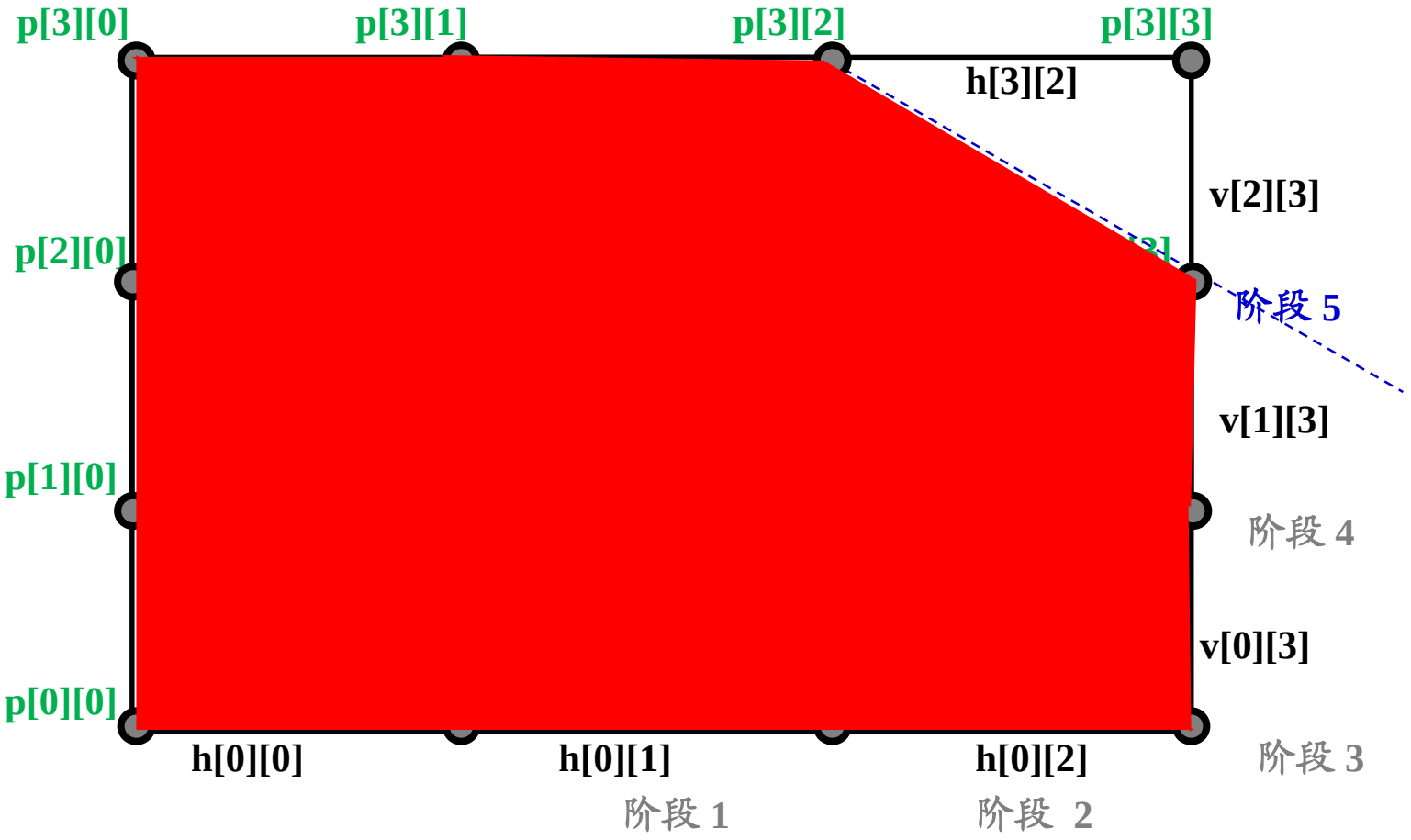
算法流程


p[3][0]
p[3][1]
p[3][2]
p[3][3]
h[3][0]
h[3][1]
h[3][2]
v[2][1]
v[2][2]
v[2][3]
p[2][0]
p[2][2]
p[2][3]
h[2][1
h[2][2]
阶段 5
v[1][2]
v[1][3]
p[1][0]
p[1][3]
h[1][2]
阶段 4
v[0][3]
p[0][0]
h[0][0]
h[0][1]
h[0][2]
阶段 3
阶段 1
阶段 2

算法流程


p[3][0]
p[3][1]
p[3][2]
p[3][3]
h[3][1]
h[3][2]
v[2][2]
v[2][3]
p[2][0]
p[2][3]
h[2][2]
阶段 5
v[1][3]
p[1][0]
阶段 4
v[0][3]
p[0][0]
h[0][0]
h[0][1]
h[0][2]
阶段 3
阶段 1
阶段 2

算法流程


p[3][0]
p[3][1]
p[3][2]
p[3][3]
h[3][2]
v[2][3]
p[2][0]
阶段 5
v[1][3]
p[1][0]
阶段 4
v[0][3]
p[0][0]
h[0][0]
h[0][1]
h[0][2]
阶段 3
阶段 1
阶段 2

算法流程

对于阶段2

P[1][1] = min{ P[0][1]+v[0][1], P[1][0]+h[1][0]}

= min{3+2, 2+2} = 4

P[0][2] = P[0][1]+h[0][1] = 3+2 = 5

P[2][0] = P[1][0]+v[1][0] = 2+4 = 6

对于阶段3

P[1][2] = min{ P[0][2]+v[0][2], P[1][1]+h[1][1]}

= min{5+3, 4+1} = 5

P[0][3] = P[0][2]+h[0][2] = 5+3 = 8

P[2][1] = min{ P[1][1]+v[1][1], P[2][0]+h[2][0]}

= min{4+1, 6+3} = 5

P[3][0] = P[2][0]+v[2][0] = 6+1 = 7

算法流程

对于阶段4

P[1][3] = min{ P[0][3]+v[0][3], P[1][2]+h[1][2]}

= min{8+4, 5+4} = 9

P[2][2] = min{ P[1][2]+v[1][2], P[2][1]+h[2][1]}

= min{5+2, 5+4} = 7

P[3][1] = min{ P[2][1]+v[2][1], P[3][0]+h[3][0]}

= min{5+2, 7+3} = 7

对于阶段5

P[2][3] = min{ P[1][3]+v[1][3], P[2][2]+h[2][2]}

= min{9+4, 7+5} = 12

P[3][2] = min{ P[2][2]+v[2][2], P[3][1]+h[3][1]}

= min{7+2, 7+1} = 8

算法流程

最后

P[3][3] = min{ P[2][3]+v[2][3], P[3][2]+h[3][2]}

= min{12+3, 8+2} = 10

综上，数组P的通项表示为

P[i][j]= min((p[i-1][j]+v[i-1][j]),

(p[i][j-1]+h[i][j-1])) (i, j>0)

P[0][j]=P[0][j-1]+h[0][j-1] (i=0, j>0)

P[i][0]=P[i-1][0]+v[i-1][0] (i>0, j=0)

下面给出P数组中的数据

3	7	7	8	10
2	6	5	7	12
1	2	4	5	9
0	0	3	5	8
	0	1	2	3

算法核心

数组P的通项表示为

$$P[i][j]= \min((p[i-1][j]+v[i-1][j]), (p[i][j-1]+h[i][j-1])) \quad (i, j>0)$$

$$P[0][j]=P[0][j-1]+h[0][j-1] \quad (i=0, j>0)$$

$$P[i][0]=P[i-1][0]+v[i-1][0] \quad (i>0, j=0)$$

算法效果

- 画出用动态规划思想求出的各个路口对P点的最小距离。图中圆圈里就是这个距离。箭头表示所寻得的最佳行走路径。（图 3）

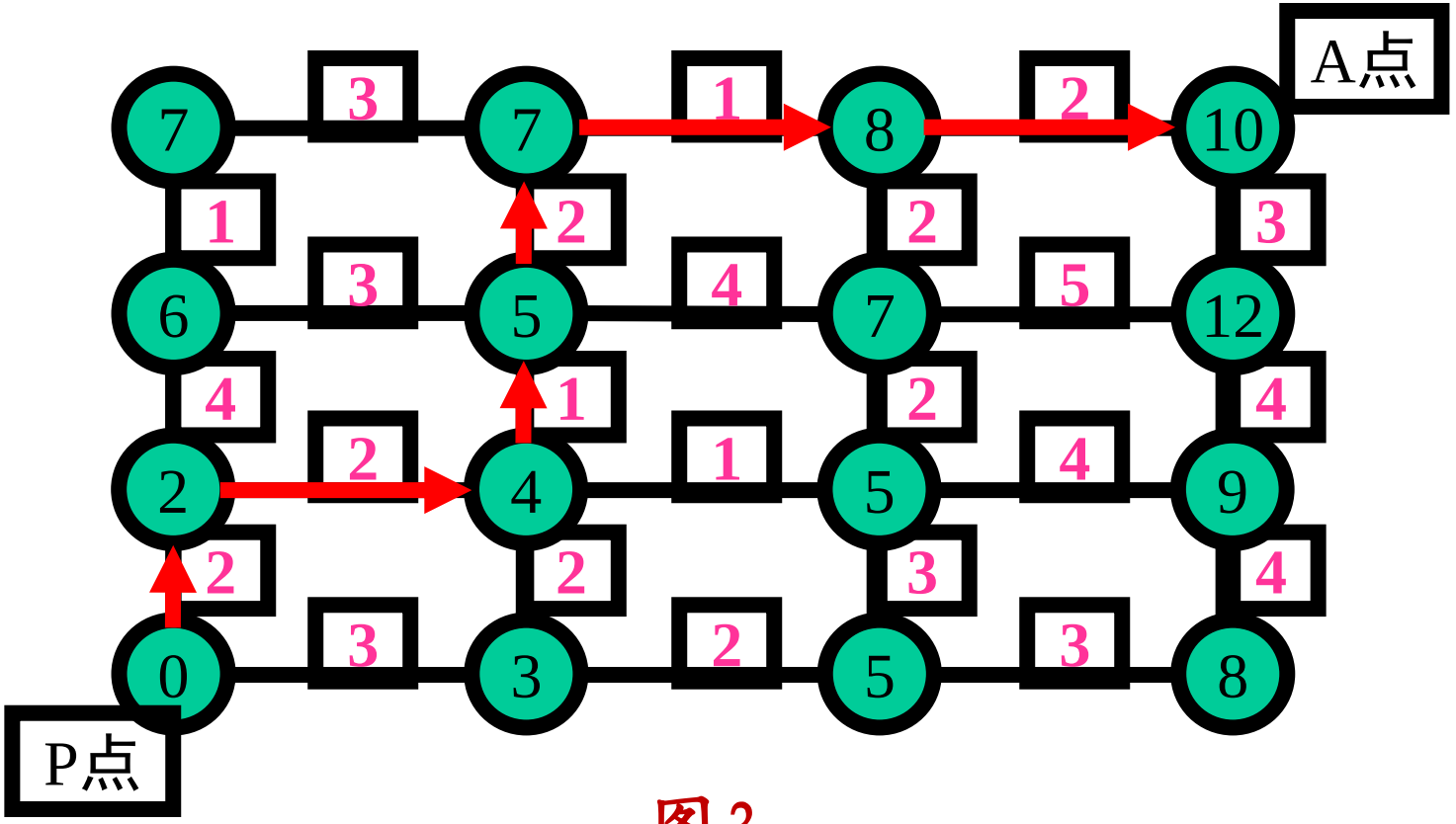


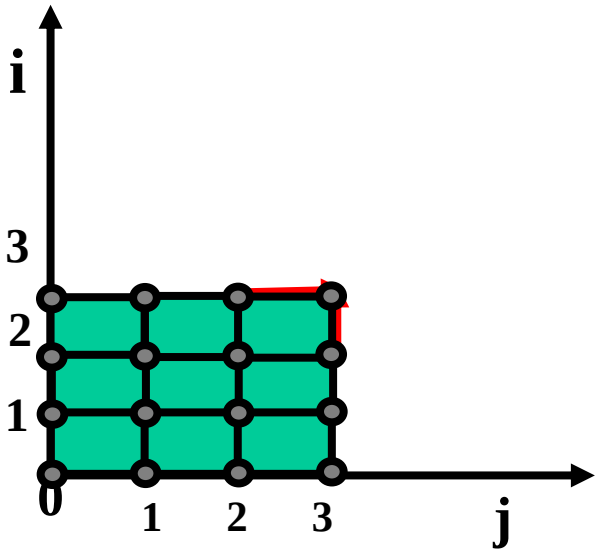
图 3

参考程序

```
#include <iostream>
using namespace std;
int min(int, int);      // 声明有子函数 min( )
int  main()             // 主函数
{
   int h[4][3]={ {3,2,3},{2,1,4},{3,4,5},{3,1,2} }; //东西路段
   int v[3][4]={ {2,2,3,4},{4,1,2,4},{1,2,2,3} };    //南北路段
   int p[4][4]={ {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
   p[0][0]=0;                    // int p[4][4] = {{0}}; 可以吗?
   for(int  j=1;j<4;j++)                              //x轴上的点
      p[0][j]=p[0][j-1]+h[0][j-1];
   for(int  i=1;i<4;i++)                              //y轴上的点
      p[i][0]=p[i-1][0]+v[i-1][0];
```

参考程序

```
    for (int  i=1; i<4; i++)  //内部的点
        for (int  j=1; j<4; j++)
            p[i][j]=min( ( p[i-1][j]+v[i-1][j]) ,
                         ( p[i][j-1]+h[i][j-1]));  // min(…) : next page
    cout<<“from P to A is ”<<p[3][3]<<endl;
    //输出每个路口对P点的最小距离
    for(int  i=3;i>=0;i--)
    {
       for (int  j=0; j<=3; j++)
       {  cout << p[i][j] << "   "; }
        cout<<endl;
     }
     return 0;
}
```



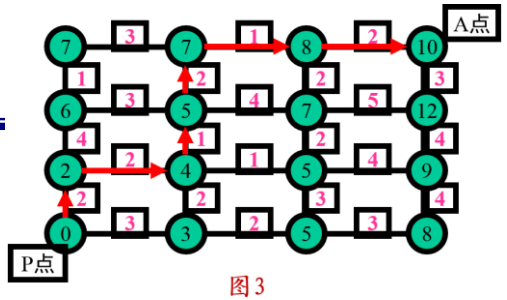


图3

QuickWatch

Expression: p

Reevaluate

Add Watch

Value:

Name	Value	Type
p	0x001ff6e4 {0x001ff6e4 {0, 0,	int[4][4]
[0]	0x001ff6e4 {0, 0, 0, 0}	int[4]
[1]	0x001ff6f4 {0, 0, 0, 0}	int[4]
[2]	0x001ff704 {0, 0, 0, 0}	int[4]
[3]	0x001ff714 {0, 0, 0, 0}	int[4]

Close Help

QuickWatch

Expression: p

Reevaluate

Add Watch

Value:

Name	Value	Type
p	0x001ff6e4 {0x001ff6e4 {0, 3,	int[4][4]
[0]	0x001ff6e4 {0, 3, 5, 8}	int[4]
[1]	0x001ff6f4 {2, 0, 0, 0}	int[4]
[2]	0x001ff704 {6, 0, 0, 0}	int[4]
[3]	0x001ff714 {7, 0, 0, 0}	int[4]

Close Help

QuickWatch

Expression: p

Reevaluate

Add Watch

Value:

Name	Value	Type
p	0x001ff6e4 {0x001ff6e4 {0, 3,	int[4][4]
[0]	0x001ff6e4 {0, 3, 5, 8}	int[4]
[1]	0x001ff6f4 {2, 4, 0, 0}	int[4]
[2]	0x001ff704 {6, 0, 0, 0}	int[4]
[3]	0x001ff714 {7, 0, 0, 0}	int[4]

Close Help

QuickWatch

Expression: p

Reevaluate

Add Watch

Value:

Name	Value	Type
p	0x001ff6e4 {0x001ff6e4 {0, 3,	int[4][4]
[0]	0x001ff6e4 {0, 3, 5, 8}	int[4]
[1]	0x001ff6f4 {2, 4, 5, 9}	int[4]
[2]	0x001ff704 {6, 0, 0, 0}	int[4]
[3]	0x001ff714 {7, 0, 0, 0}	int[4]

Close Help

QuickWatch

Expression: p

Reevaluate

Add Watch

Value:

Name	Value	Type
p	0x001ff6e4 {0x001ff6e4 {0, 3,	int[4][4]
[0]	0x001ff6e4 {0, 3, 5, 8}	int[4]
[1]	0x001ff6f4 {2, 4, 5, 9}	int[4]
[2]	0x001ff704 {6, 5, 7, 12}	int[4]
[3]	0x001ff714 {7, 0, 0, 0}	int[4]

Close Help

QuickWatch

Expression: p

Reevaluate

Add Watch

Value:

Name	Value	Type
p	0x001ff6e4 {0x001ff6e4 {0, 3,	int[4][4]
[0]	0x001ff6e4 {0, 3, 5, 8}	int[4]
[1]	0x001ff6f4 {2, 4, 5, 9}	int[4]
[2]	0x001ff704 {6, 5, 7, 12}	int[4]
[3]	0x001ff714 {7, 7, 8, 10}	int[4]

Close Help

参考程序

```
// 两整数比大小，返回小者
int min(int a, int b)
{
    if (a<=b )  return a;
    else  return b;
}
```

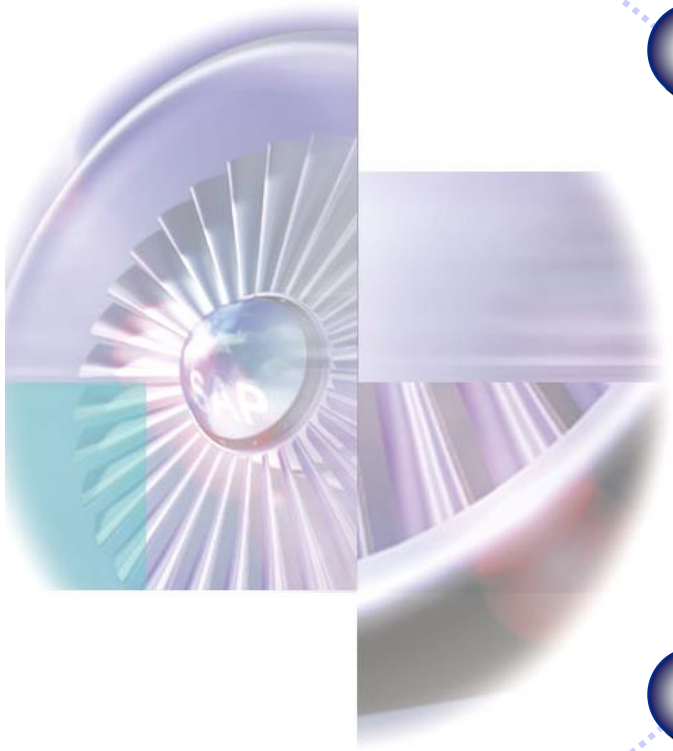
是否有多个最短路径？

所有边长置为 1

```
from P to A is 10
7  7  8  10
6  5  7  12
2  4  5  9
0  3  5  8
```

```
from P to A is 6
3  4  5  6
2  3  4  5
1  2  3  4
0  1  2  3
```

Lecture 10: 动态规划



1. 实际问题
2. 动态规划的基本概念
3. 算法举例

动态规划的基本概念

阶段：据空间顺序或时间顺序对问题的求解划分阶段。 5个阶段

状态：描述事物的性质，不同事物有不同的性质，因而用不同的状态来刻画。对问题的求解状态的描述是分阶段的。 P[i][j]

决策：根据题意要求，对每个阶段所做出的某种选择性操作。 min

状态转移方程：用数学公式描述与阶段相关的状态间的演变规律。

P[i][j]= min((p[i-1][j]+v[i-1][j]), (p[i][j-1]+h[i][j-1]))

概念——动态规划

动态规划是运筹学的一个重要分支，是解决多阶段决策过程最优化的一种方法。

所谓多阶段决策过程，是将所研究的过程划分为若干个相互联系的阶段，在求解时，对每一个阶段都要做出决策，前一个决策确定以后，常常会影响下一个阶段的决策。

动态规划——依据

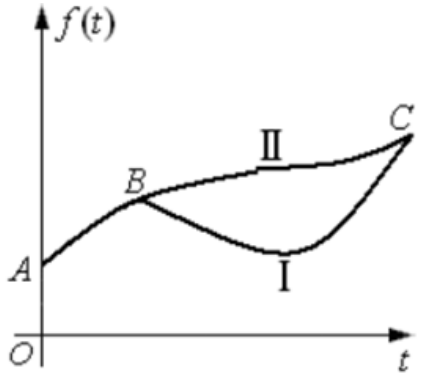
动态规划所依据的是“最优性原理”。

“最优性原理”可陈述为：不论初始状态和第一步决策是什么，余下的决策相对于前一次决策所产生的新状态，构成一个最优决策序列。

最优决策序列的子序列，一定是局部最优决策子序列。

包含有非局部最优的决策子序列，一定不是最优决策序列。

if (AB + BⅡC == A到C的最优路线)
BⅡC一定是从B到C的最优路线;

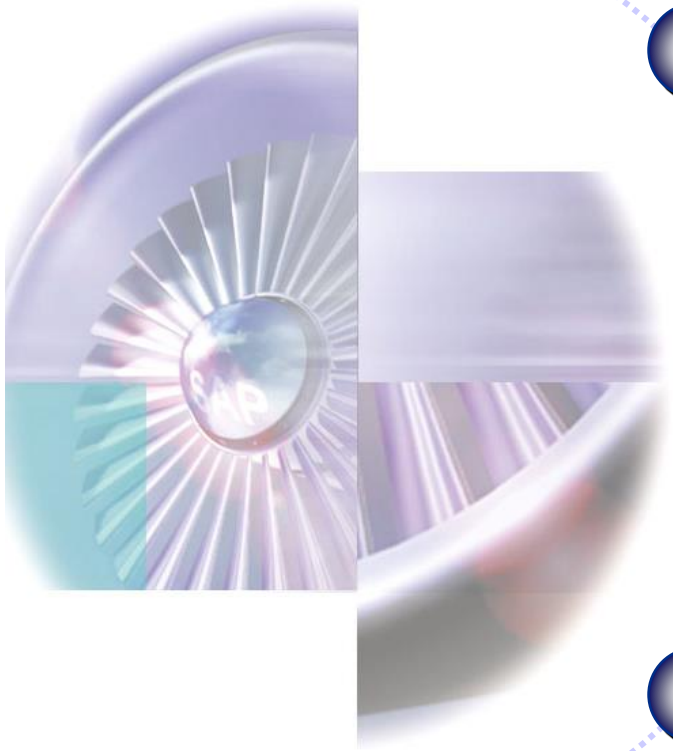


指导思想

动态规划编程的指导思想：在做每一步决策时，列出各种可能的局部解，依据某种判定条件，舍弃那些肯定不能得到最优解的局部解。

这样，在每一步都进行筛选，以每一步都是最优的来保证全局是最优的。筛选相当于最大限度地有效剪枝(从搜索角度看)，效率会十分高。

Lecture 10: 动态规划



1. 实际问题
2. 动态规划的基本概念
3. 算法举例

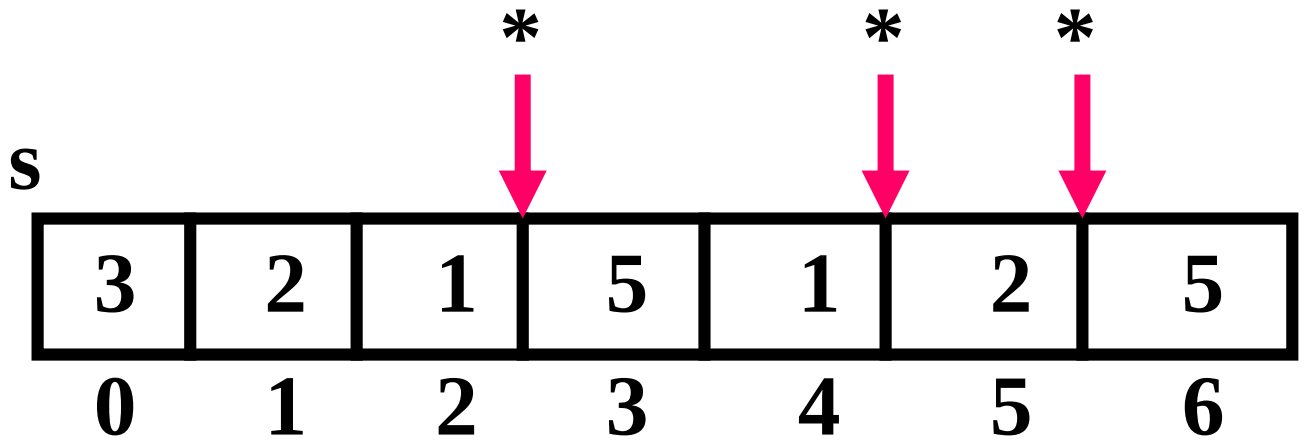
举例

动态规划举例

任务：编写一个程序，在一个给定的数字串中插入 k 个乘号，使总的乘积最大。

如何寻找解题思路？

没思路时的办法——用实例进行分析与归纳



请插入3个乘号使乘积最大。

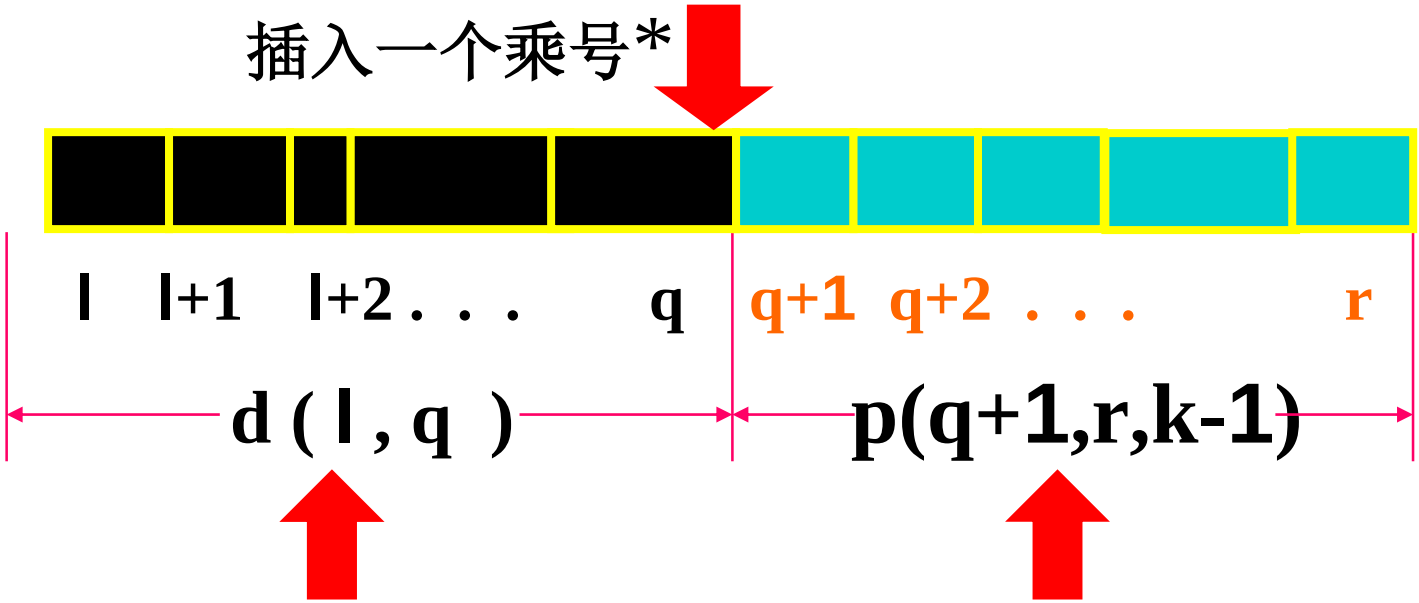
32*15*12*5= 28800　　第1种方案

3*215*12*5= 38700　　第2种方案

321*51*2*5=163710　　第3种方案

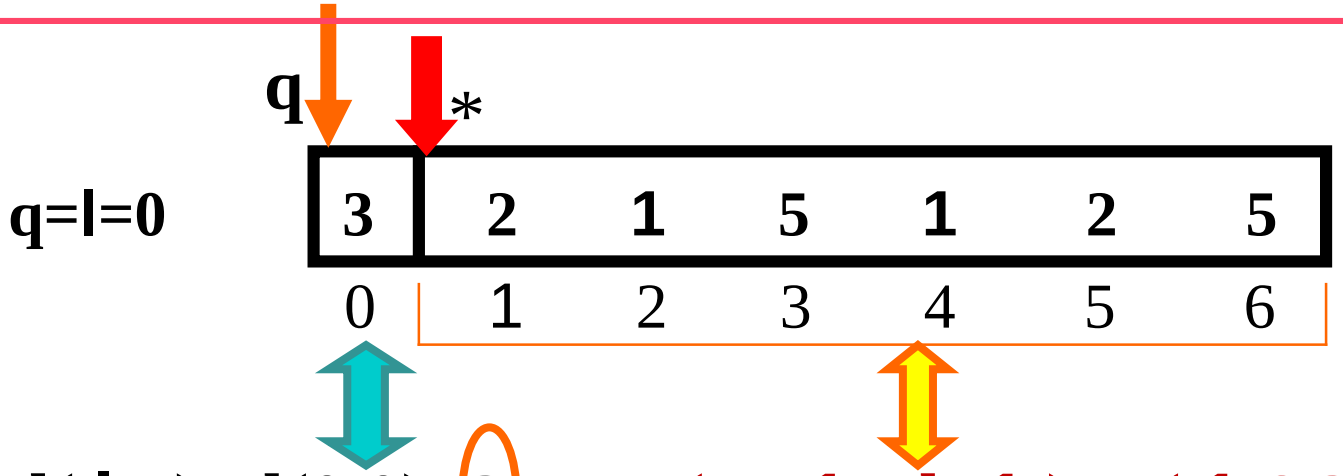
定义 ： p(l , r , k)为从 l 到 r 加入 k 个乘号的最大乘积值，d(l, q) = s[l]s[l+1]…s[q]组成的数字值。

令 p(l, r, k) = d(l , q) * p(q+1, r, k-1)



$$p(l, r, k) = \max_{q} \{ d(l, q) * p(q+1, r, k-1) \}$$

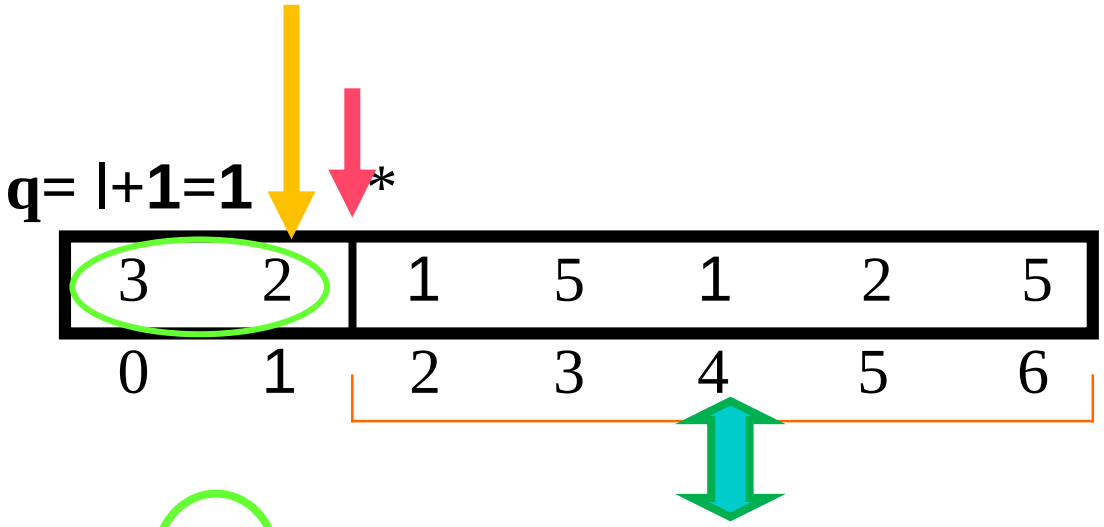
$q = l, l+1, \ldots, r-k$



$d(l,q) = d(0,0) = 3$ $\quad p(q+1, r, k-1) = p(1, 6, 2)$

$(p(0,6,3) | q=0) = 3 * p(1,6,2)$

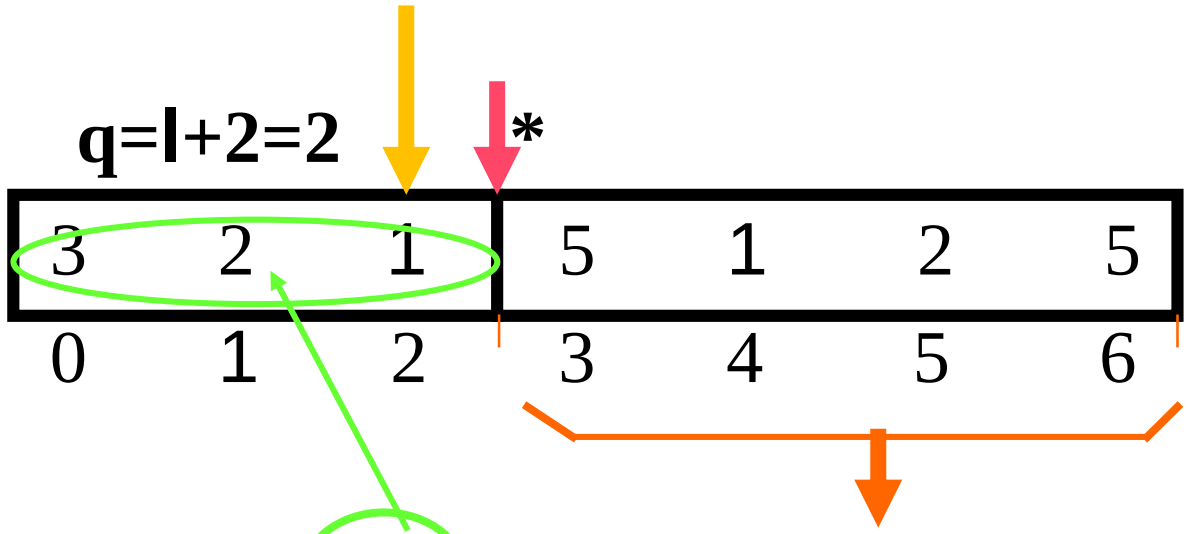
$3 * 215125$



d(l,q)=d(0,1)= 32

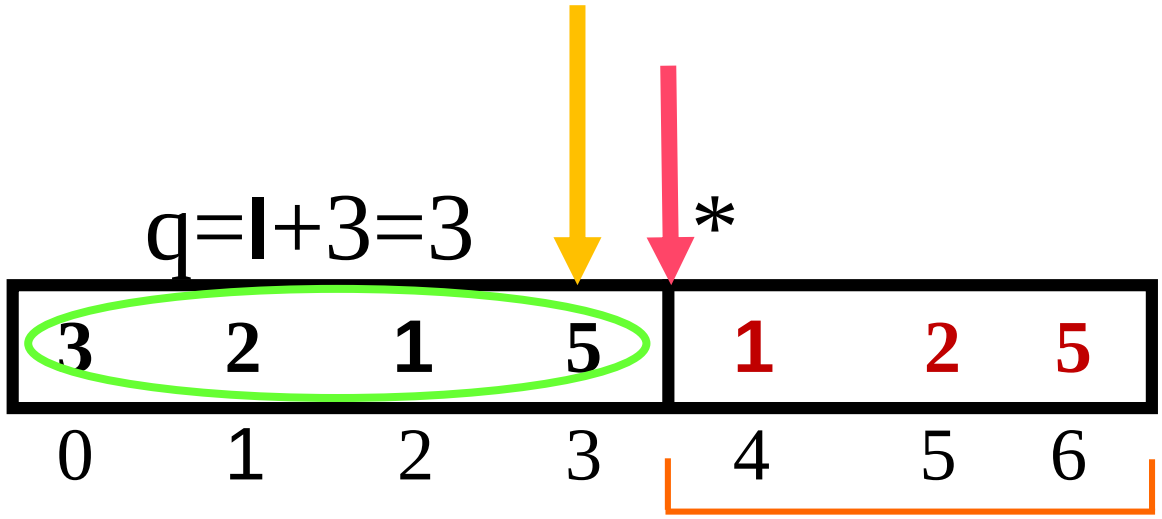
p(q+1, r, k-1) = p(2, 6, 2)

(p(0,6,3)|q=1) = 32 * p(2, 6, 2)



d(l,q)=d(0,2)=321 **p(q+1,r,k-1)=p(3,6,2)**

(p(0,6,3)|q=2) = 321 * p(3,6,2)



d(l,q)=d(0,3)=3215

p(q+1,r,k-1)=p(4,6,2)

(p(0,6,3)|q=3) = 3215 * p(4,6,2)

$$p(0, 6, 3) = \max\{\ 3 * p(1,6,2),\ 32 * p(2,6,2),\ 321 * p(3,6,2),\ 3215 * p(4,6,2)\ \}$$

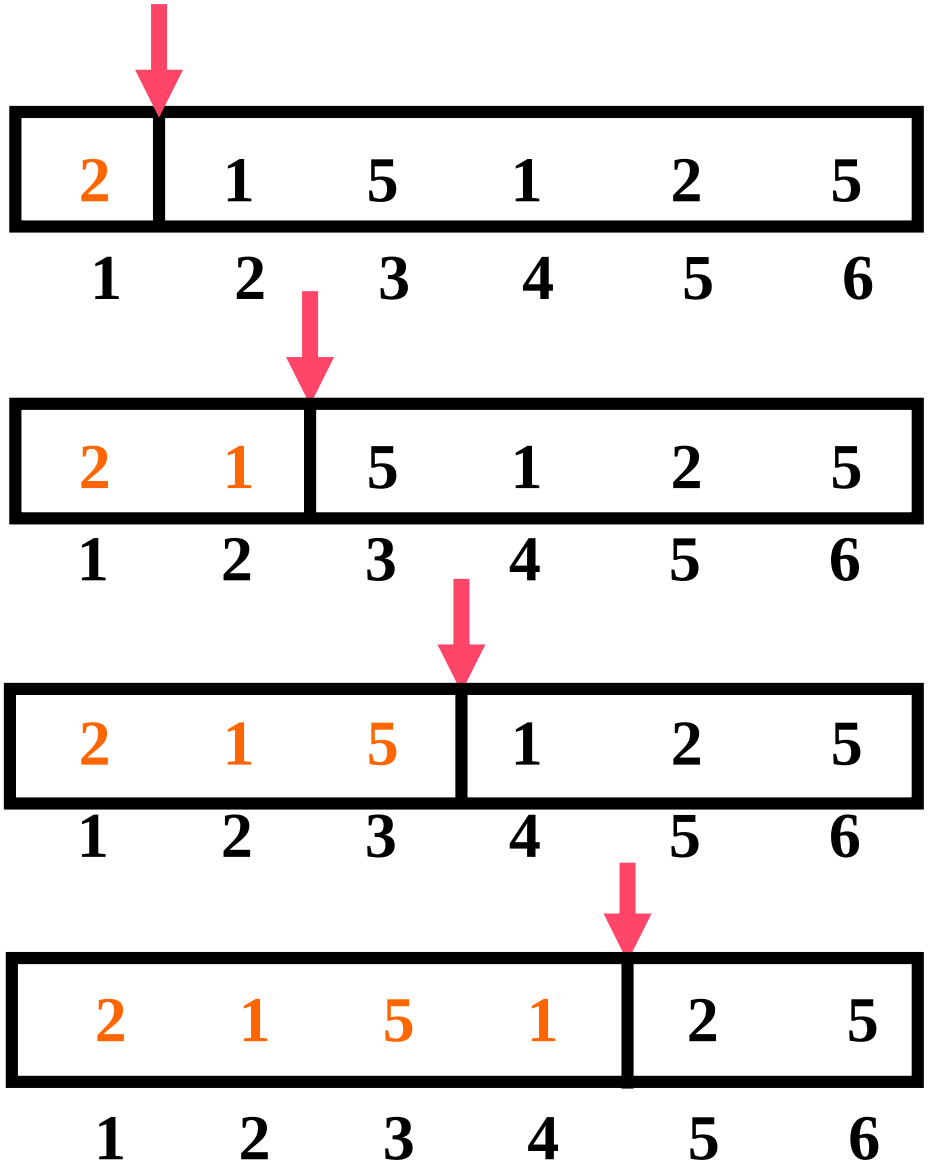
// q=0

// q=1

// q=2

// q=3

p(1,6,2)


2 1 5 1 2 5
1 2 3 4 5 6
2 * p(2,6,1)
2 1 5 1 2 5
1 2 3 4 5 6
21 * p(3,6,1)
2 1 5 1 2 5
1 2 3 4 5 6
215 * p(4,6,1)
2 1 5 1 2 5
1 2 3 4 5 6
2151 * p(5,6,1)

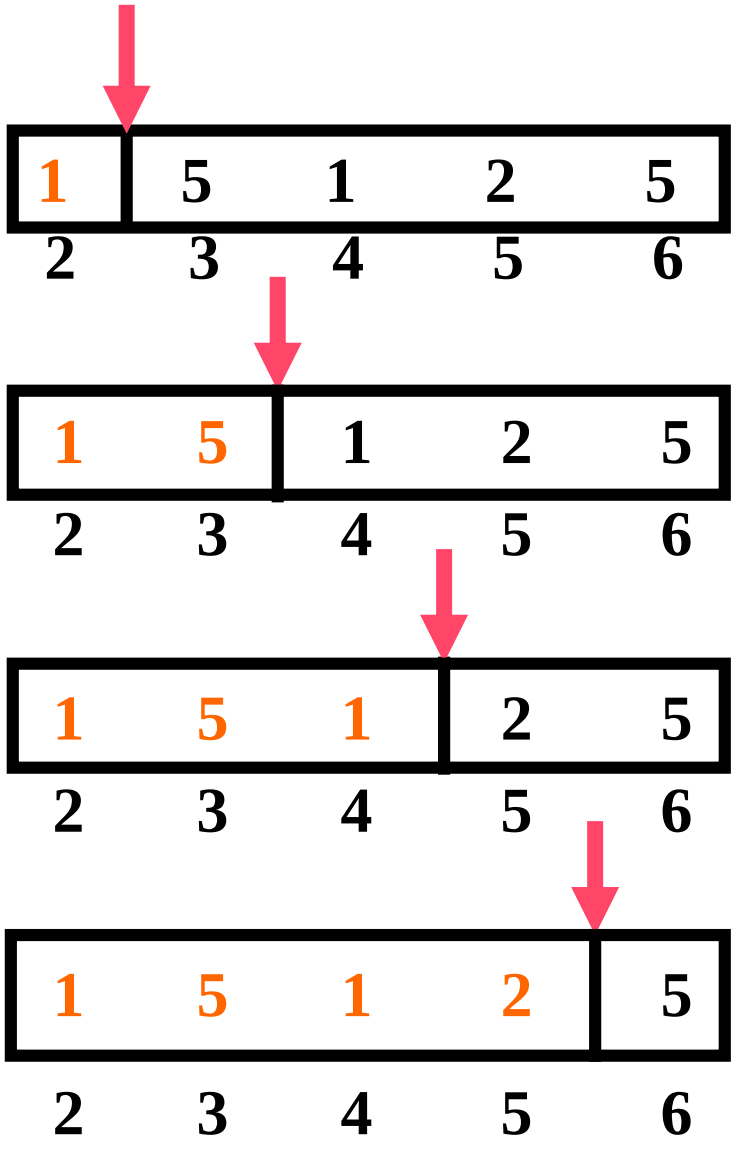
p(2,6,1)

1* 5125

15 * 125

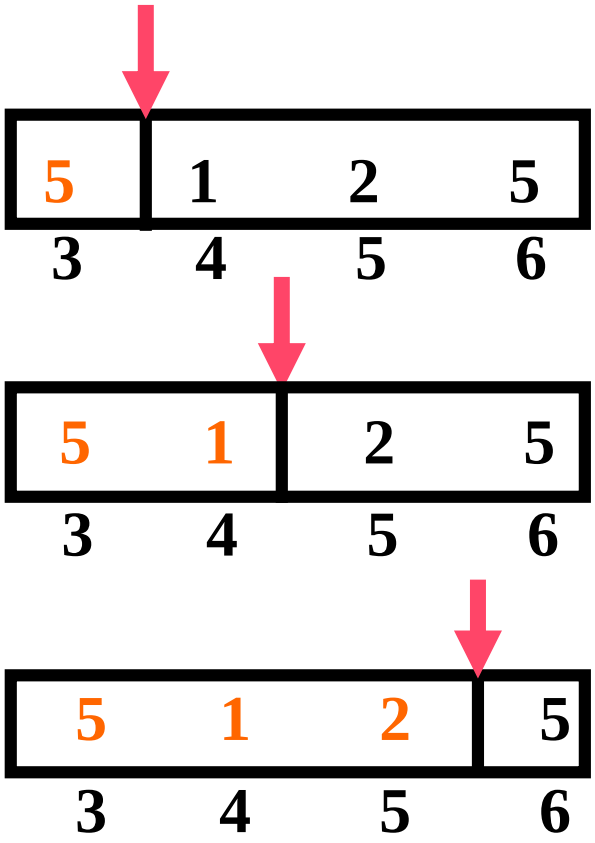
151* 25

1512 *5



$$
\begin{aligned}
p(2,6,1) = \max \{ & \\
& 1 * 5125 , \\
& 15 * 125 , \\
& 151 * 25 , \\
& 1512 * 5 \\
& \} \\
= 7560 &
\end{aligned}
$$

p(3,6,1)



5 * 125

51 * 25

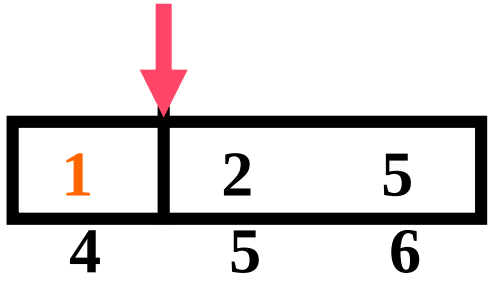
5 12* 5

p(3,6,1)=max{5*125, 51*25, 512*5 }

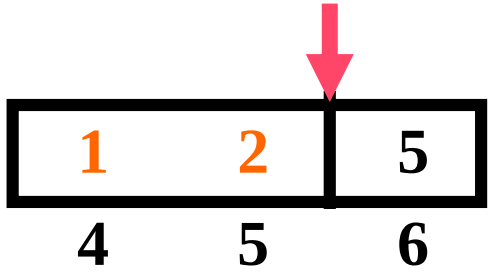
= 2560

p(4,6,1)

1 * 25



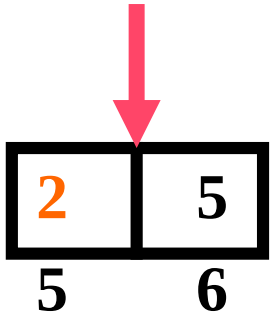
12 * 5



p(4,6,1) = max { 1 * 25, 12 * 5 }

= 60

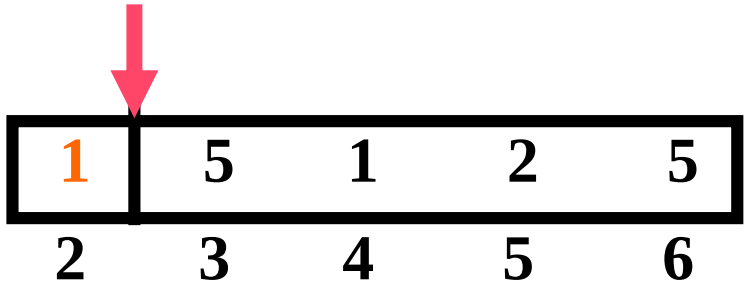
p(5,6,1)



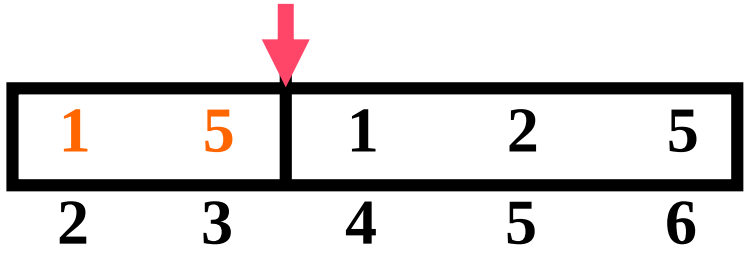
2 * 5

p(5,6,1) = 10

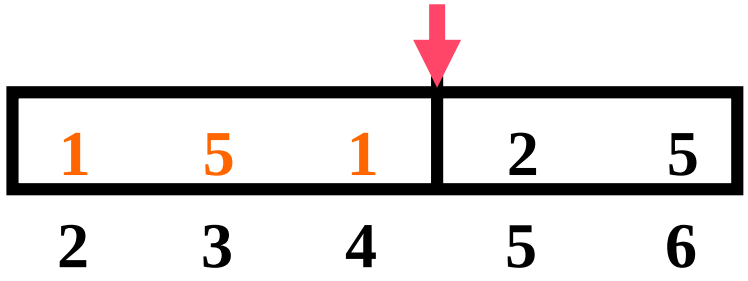
p(2,6,2)



1* p(3,6,1)



15 * p(4,6,1)

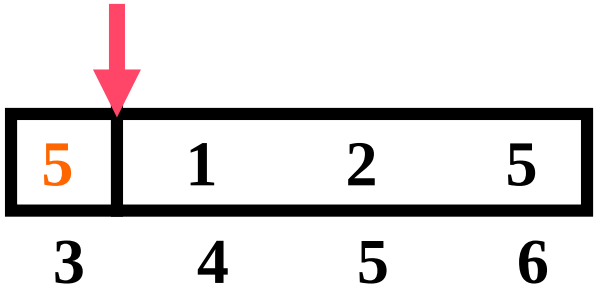


151 * p(5,6,1)

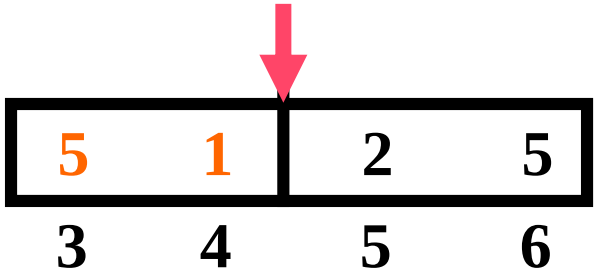
p(2,6,2)= { 1 * 2560, 15 * 60, 151 * 10 }

= 2560

p(3,6,2)



5 * p(4,6,1)

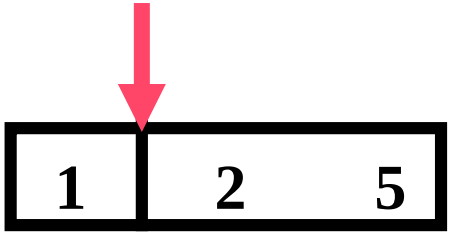


51 * p(5,6,1)

p(3,6,2) = { 5 * 60 , 51 * 10 }

= 510

p(4,6,2)



＝ 1* p(5,6,1)

因为 **p(5,6,1) = 2 * 5 = 10**

所以 **p(4,6,2) = 1 * p(5,6,1) = 10**

$$p(0,6,3) = \max\{\ 3 * p(1,6,2)\ ,\quad //\ q=0$$
$$32 * p(2,6,2)\ ,\quad //\ q=1$$
$$321 * p(3,6,2)\ ,\quad //\ q=2$$
$$3215 * p(4,6,2)\ \}\quad //\ q=3$$

$p(1,6,2) = 53760$

$p(2,6,2) = 2560$

$p(3,6,2) = 510$

$p(4,6,2) = 60$

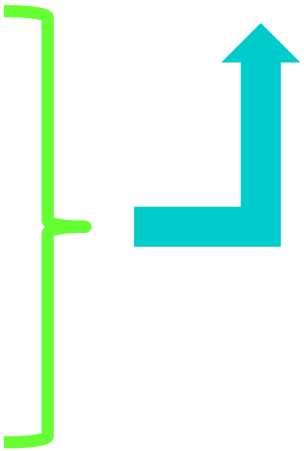
$$p(0,6,3) = \max\{\ 3 * 53760,\ 32 * 2560,\ 321 * 510,\ 3215 * 60\ \} = 163710$$

$p(1,6,2) = 53760$

$p(2,6,2) = 2560$

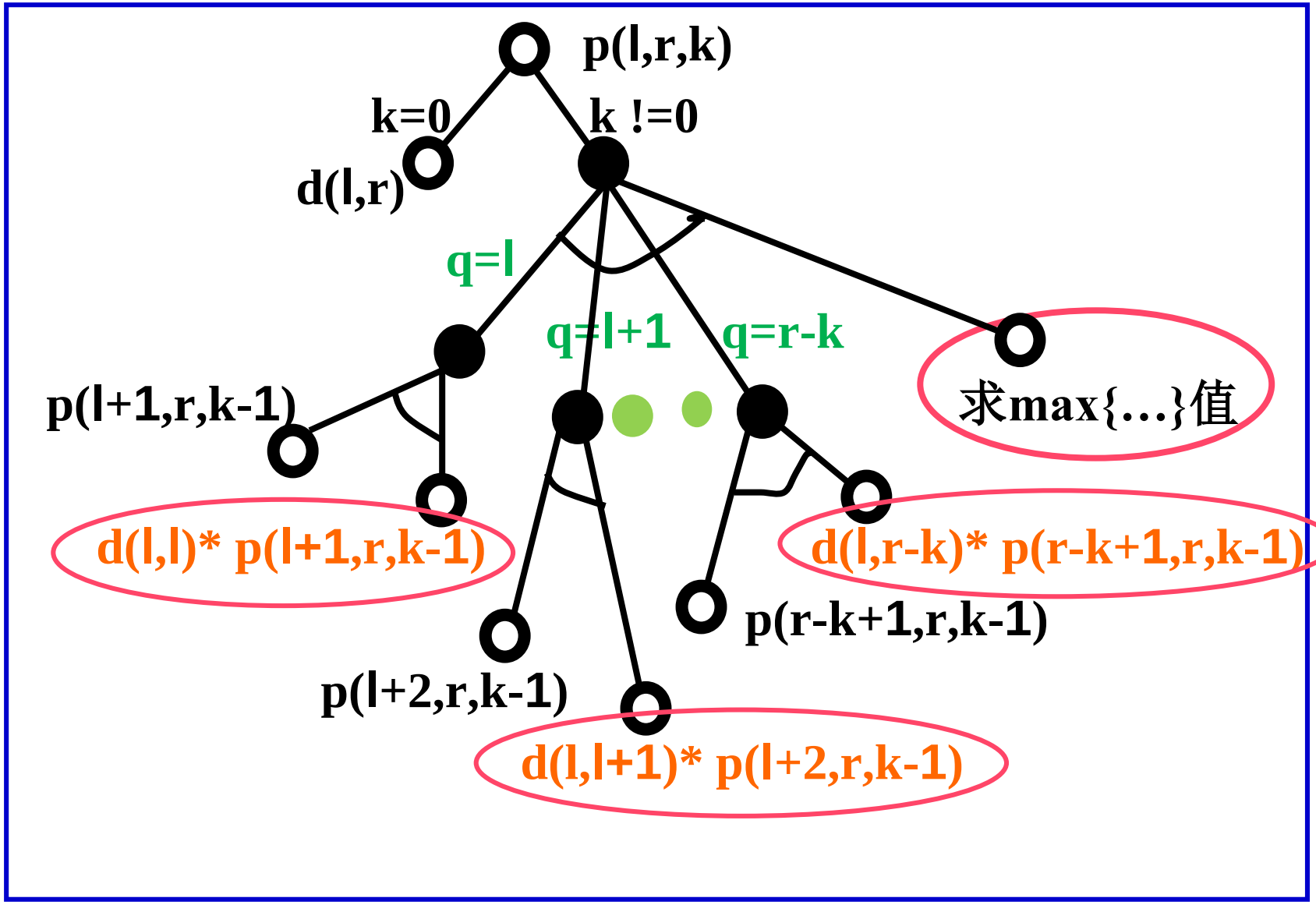
$p(3,6,2) = 510$

$p(4,6,2) = 60$



$$p(l, r, k) = \max_{q}\{ d(l, q) * p(q+1, r, k-1) \}$$

$q = l, l+1, \ldots, r-k$



d(l, q) = s[l] s[l+1]…s[q]组成的数字值

d[i][j] i \ j	0	1	2	3	4	5	6
0	3	32	321	3215	32151	321512	3215125
1		2	21	215	2151	21512	215125
2			1	15	151	1512	15125
3				5	51	512	5125
4					1	12	125
5						2	25
6							5

3215125

思考：如何自动地高效生成上表？

怎样计算这张表 ？

d[i][6], i = 0, 1, 2, 3, 4, 5, 6

d[0][6] = s = 3215125

d[1][6] = 215125

= 3215125 % 1000000

= s % 1000000

= s % s1

s1 = 1000000

s1 = s1 / 10

d[2][6] = d[1][6] % s1

```
s1=1000000;  d[0][6]=s;
for(int  i=1; i<= 6; i++ )
{
        d[i][6] = d[i-1][6] % s1;
        s1 = s1/10;
}
```

d[i][j] i \ j	0	1	2	3	4	5	6
0	3	32	321	3215	32151	321512	3215125
1		2	21	215	2151	21512	215125
2			1	15	151	1512	15125
3				5	51	512	5125
4					1	12	125
5						2	25
6							5

d[i][j] i \ j	0	1	2	3	4	5	6
0	3	32	321	3215	32151	321512	3215125
1		2	21	215	2151	21512	215125
2			1	15	151	1512	15125
3				5	51	512	5125
4					1	12	125
5						2	25
6							5

怎样求 d[i][5], d[i][4],…d[i][0] ？

i=0,1,2,3,4,5,6

```
for  (int  j=5; j>=0; j-- )
       for(int  i=0; i<= j;  i ++ )
       {
              d[i][j] = d[i][j+1] / 10;
        }
```

参考程序

```
#include<iostream>   // 预编译命令
#include<cstring>    // 预编译命令
using namespace std; // 使用名字空间

 const int S=3215125; //定义常整数
 int d[7][7] = {0};       //定义二维数组
```

```
int P( int l, int r, int k ) //计算P函数值
{
    if ( k==0)    return d[l][r];
    int x, ans=0;
    for( int q=l; q<=r-k; q++ )
    {
        x=d[l][q]*P( q+1,r,k-1);
        if( x>ans ) ans=x;
    }
    return ans;
}
```

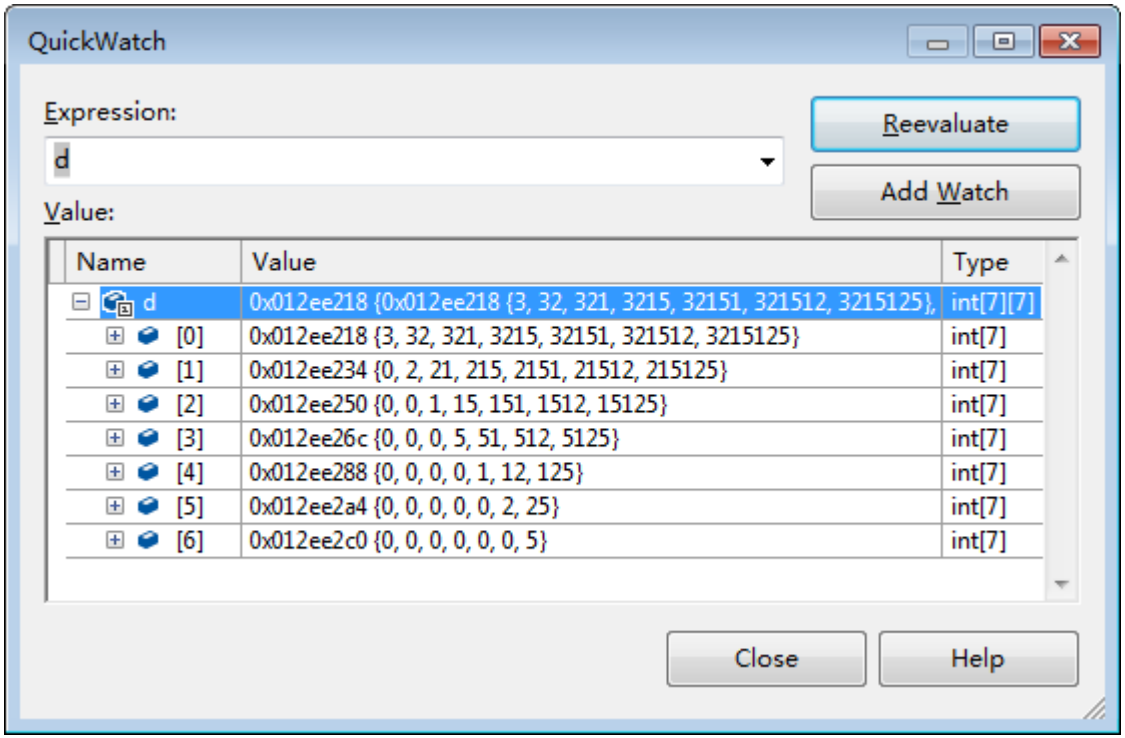
$$p(l, r, k) = \max_{q}\{ d(l, q) * p(q+1, r, k-1) \}$$

$q = l, l+1, \ldots, r-k$

```
int main()
{
    int s1=1000000;  d[0][6]=S;
    for( int  i=1; i<= 6; i++ )
     {
        d[i][6] = d[i-1][6] % s1;
        s1 = s1/10;
     }
```

```
    for( int  j=5; j>=0; j-- )
        for( int  i=0; i<= j;  i ++ )
          {
              d[i][j]=d[i][j+1]/10;
           }
    cout<<P( 0,6,3 )<<endl;
    return 0;
}
```

d[i][j] i \ j	0	1	2	3	4	5	6
0	3	32	321	3215	32151	321512	3215125
1		2	21	215	2151	21512	215125
2			1	15	151	1512	15125
3				5	51	512	5125
4					1	12	125
5						2	25
6							5



163710

动态规划的基本概念

阶段：据空间顺序或时间顺序对问题的求解划分阶段。 k个阶段

状态：描述事物的性质，不同事物有不同的性质，因而用不同的状态来刻画。对问题的求解状态的描述是分阶段的。 p(l, r, k)

决策：根据题意要求，对每个阶段所做出的某种选择性操作。 max

状态转移方程：用数学公式描述与阶段相关的状态间的演变规律。

$$p(l, r, k) = \max_{q}\{ d(l, q) * p(q+1, r, k-1) \}$$

$$q = l, l+1, \ldots, r-k$$

动态规划总结

- 动态规划算法通常基于一个递推公式及一个或多个初始状态
- 当前子问题的解将由上一次子问题的解推出
- 使用动态规划来解题往往比回溯法、暴力法等要快许多

动态规划总结

- 基本思想与分治法类似
  - 也是将待求解的问题分解为若干个子问题（阶段），按顺序求解子阶段，前一子问题的解，为后一子问题的求解提供了有用的信息。在求解任一子问题时，列出各种可能的局部解，通过决策保留那些有可能达到最优的局部解，丢弃其他局部解。依次解决各子问题，最后一个子问题就是初始问题的解
- 与分治法最大的差别是：
  - 适合于用动态规划法求解的问题，经分解后得到的子问题往往不是互相独立的
  - 即下一个子阶段的求解是建立在上一个子阶段的解的基础上，进行进一步的求解