贪心策略篇：活动选择问题

童咏昕

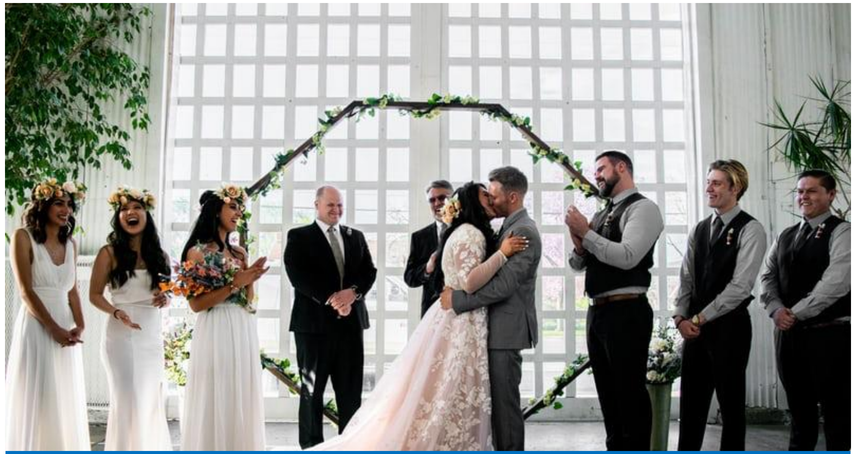
北京航空航天大学
计算机学院

中国大学MOOC北航《算法设计与分析》

问题背景

- 会场出租

公司年会：10:00～19:00



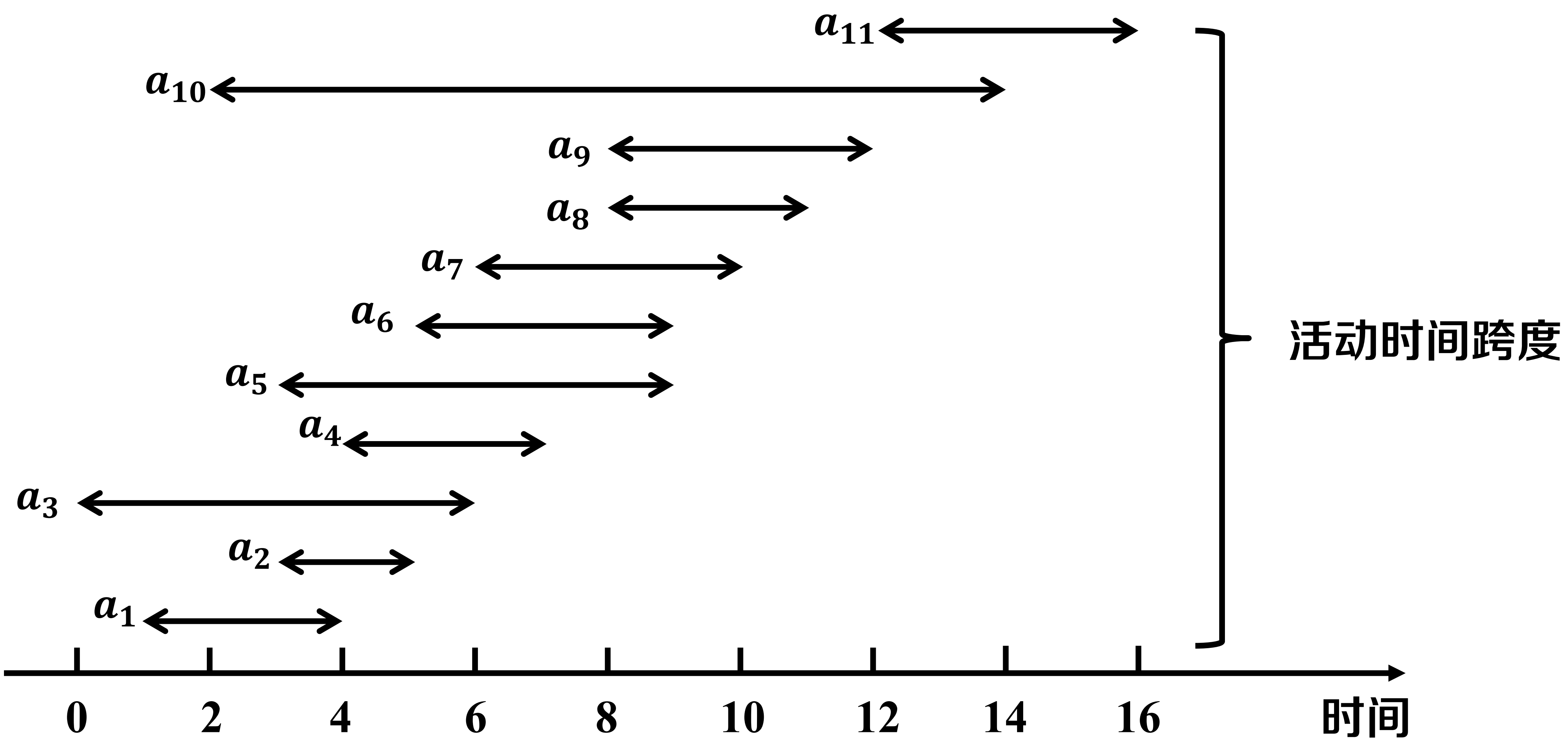
婚礼宴请：11:00～14:00

生日聚会：12:00～17:00

学术研讨：14:00～16:00

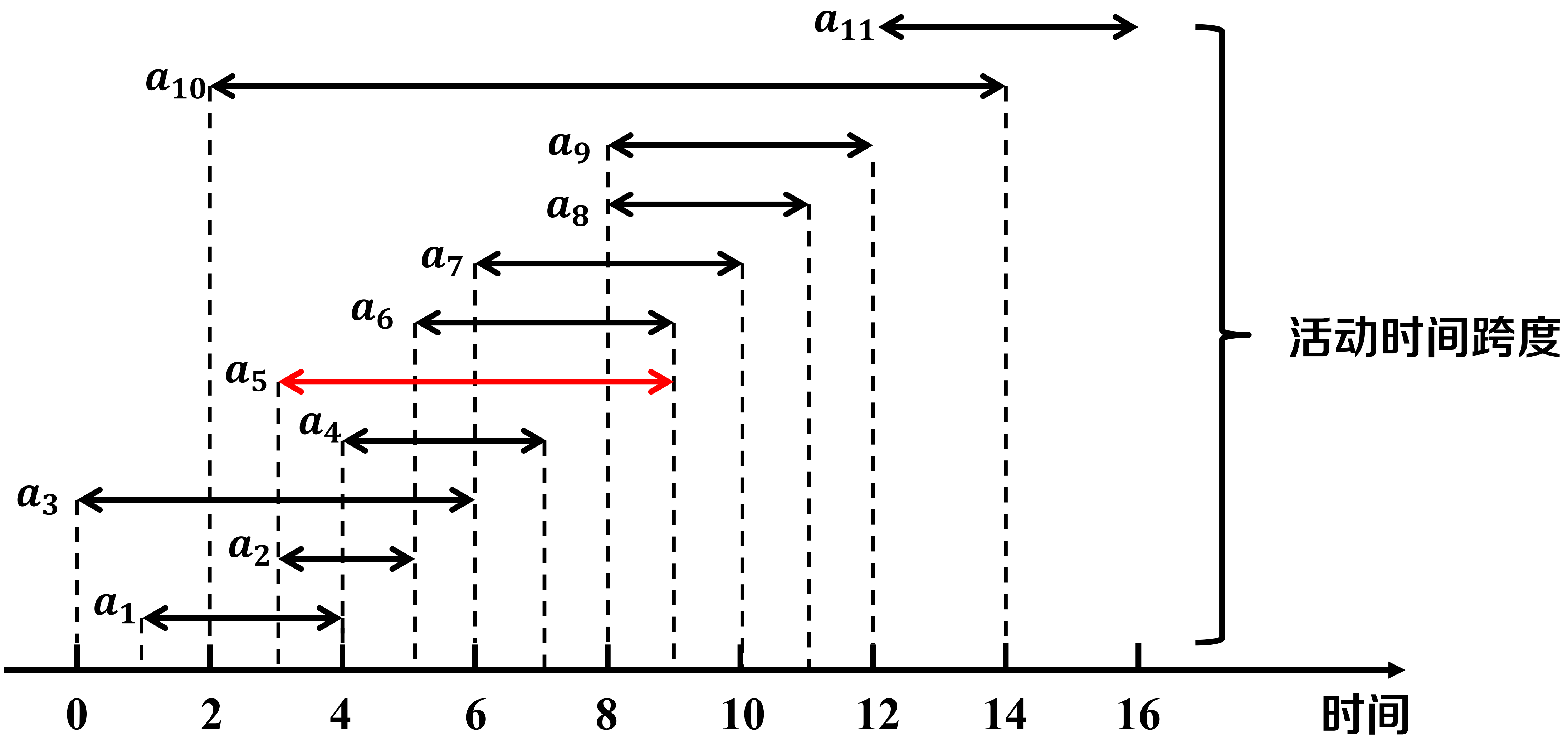
问题背景

- 会场出租



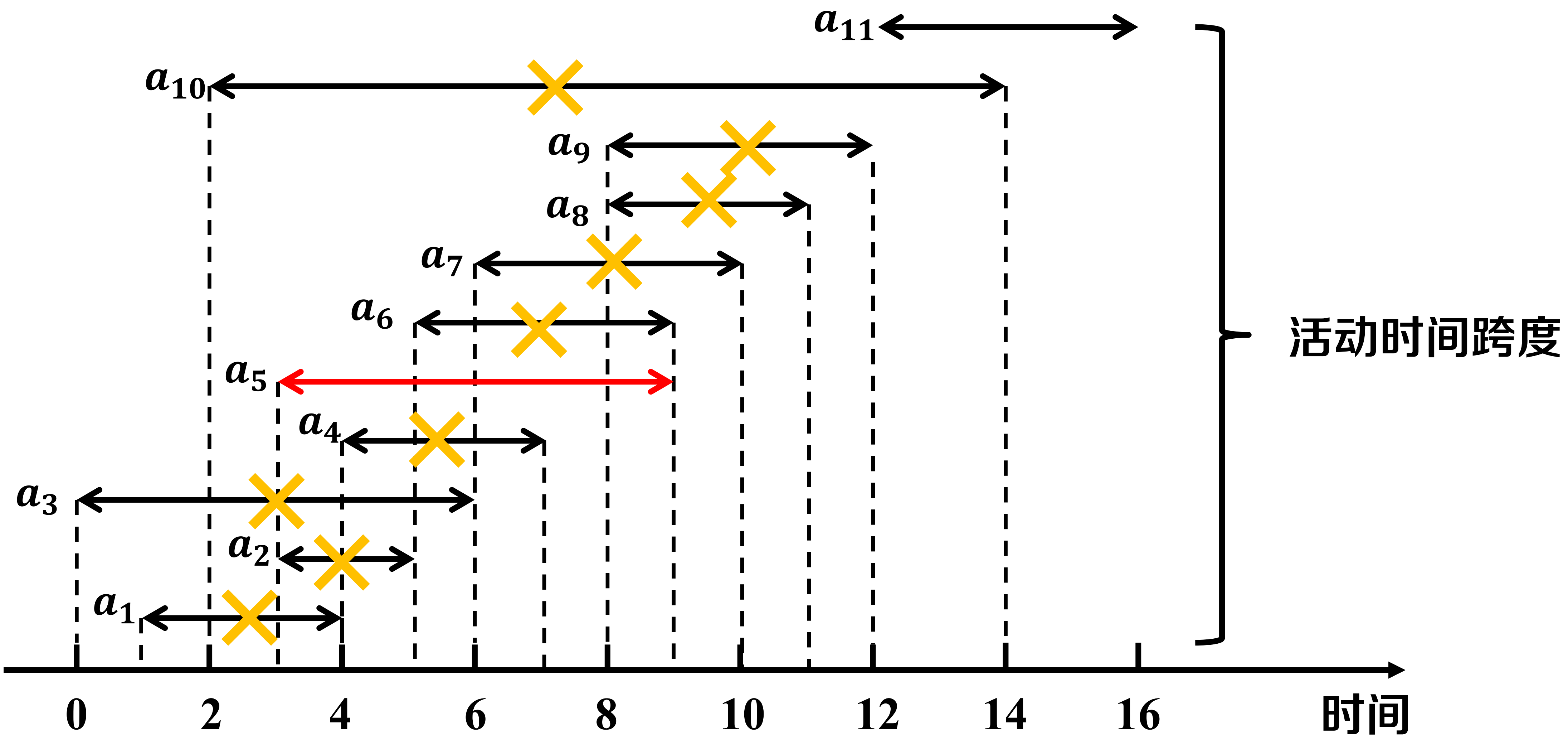
问题背景

- 会场出租
  - 选择出租的活动时间不能冲突



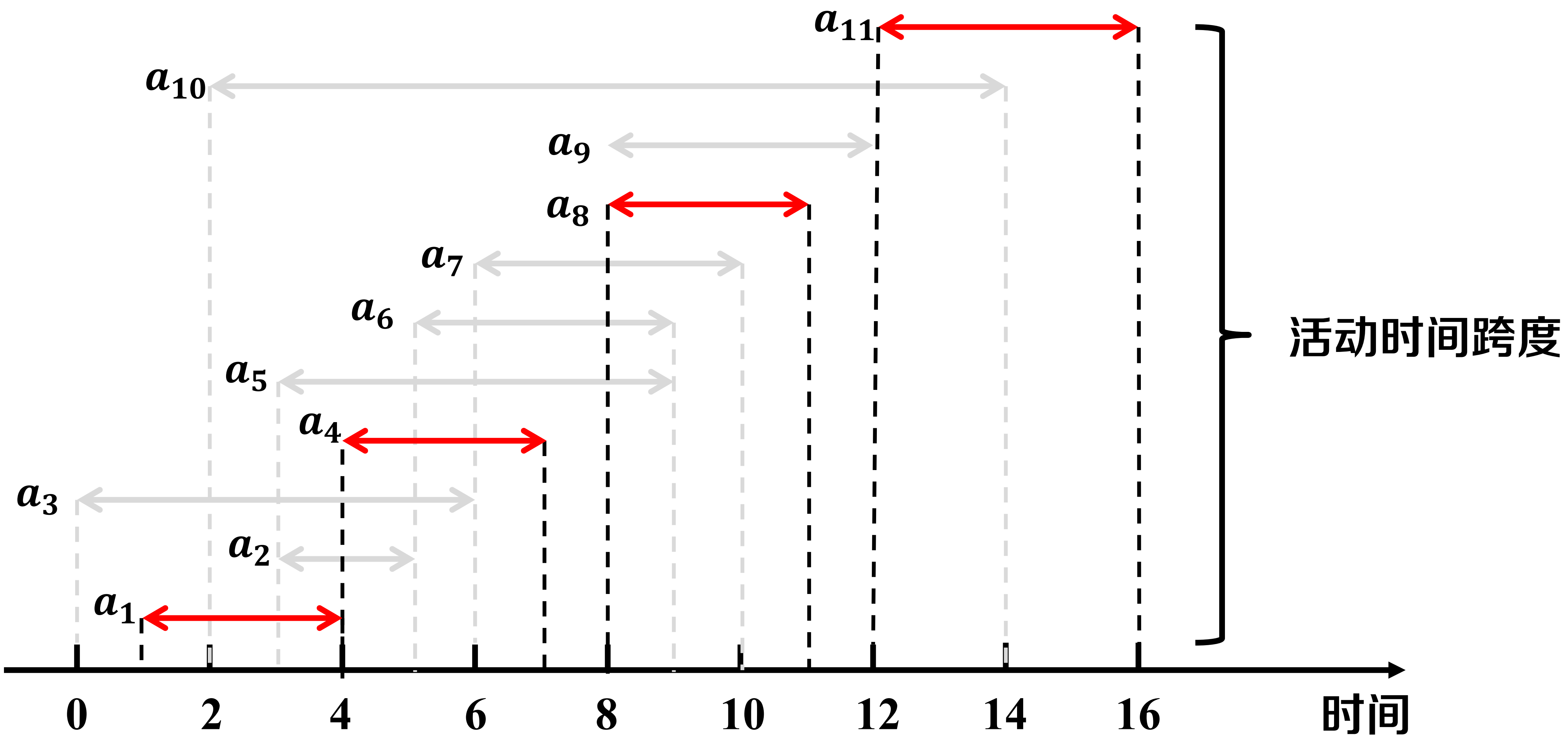
问题背景

- 会场出租
  - 选择出租的活动时间不能冲突



问题背景

- 会场出租
  - 选择出租的活动时间不能冲突，怎样选择才能选更多的活动?



问题定义

活动选择问题

Activity Selection Problem

输入

- n个活动组成的集合$S = \{a_1, a_2, \ldots, a_n\}$
- 每个活动a_i的开始时间s_i和结束时间f_i

输出

- 找出活动集合S的子集S'，令

$$\max|S'|$$

$$s.t. \forall a_i, a_j \in S', s_i \geq f_j 或 s_j \geq f_i$$

优化目标：最大化选择活动个数

问题定义

活动选择问题

Activity Selection Problem

输入

- n个活动组成的集合$S=\{a_1,a_2,\ldots,a_n\}$
- 每个活动a_i的开始时间s_i和结束时间f_i

输出

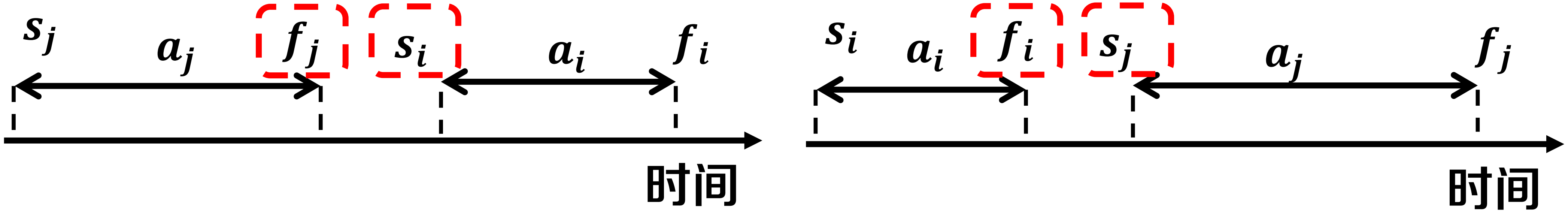
- 找出活动集合S的子集S'，令

优化目标：最大化选择活动个数

$$\max|S'|$$

约束条件

$$s.t.\ \forall a_i,a_j\in S', s_i\ge f_j\text{或}s_j\ge f_i$$



贪心策略：一般步骤

提出贪心策略

观察问题特征，构造贪心选择

证明策略正确

假设最优方案，通过替换证明

贪心策略

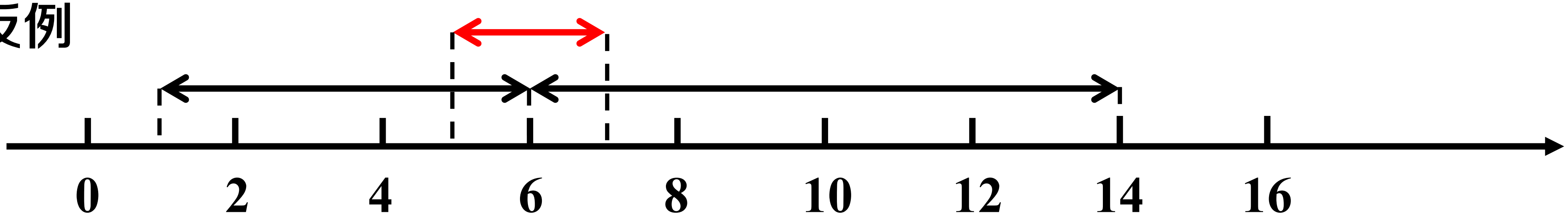
- 策略1：最短活动优先

- 策略2：最早开始活动优先

- 策略3：最早结束活动优先

贪心策略

- 策略1：最短活动优先
  - 反例

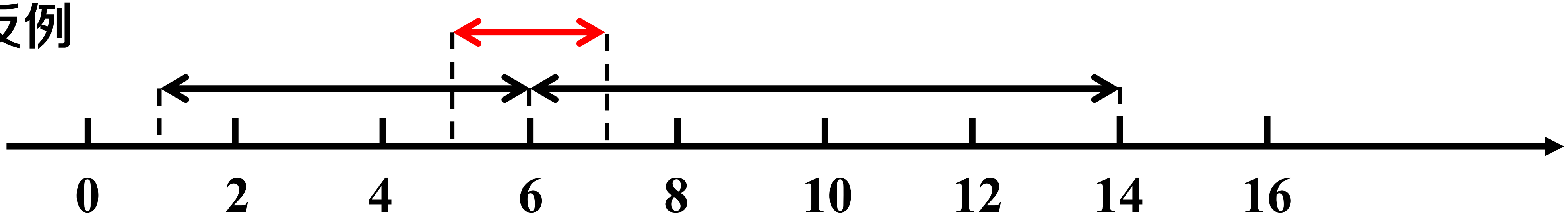


- 策略2：最早开始活动优先

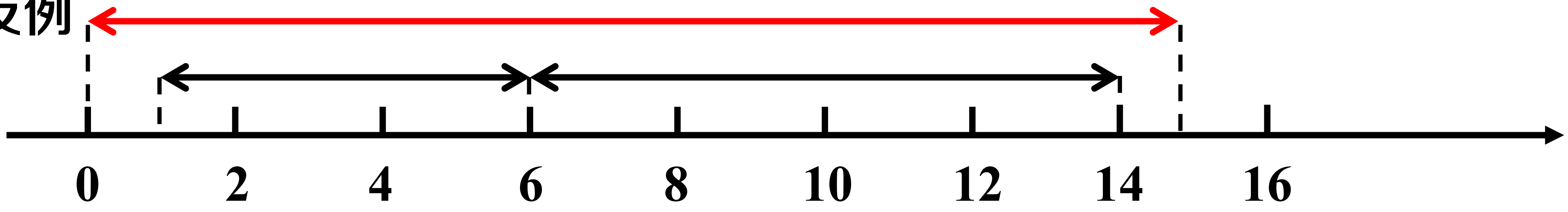
- 策略3：最早结束活动优先

贪心策略

- 策略1：最短活动优先
  - 反例



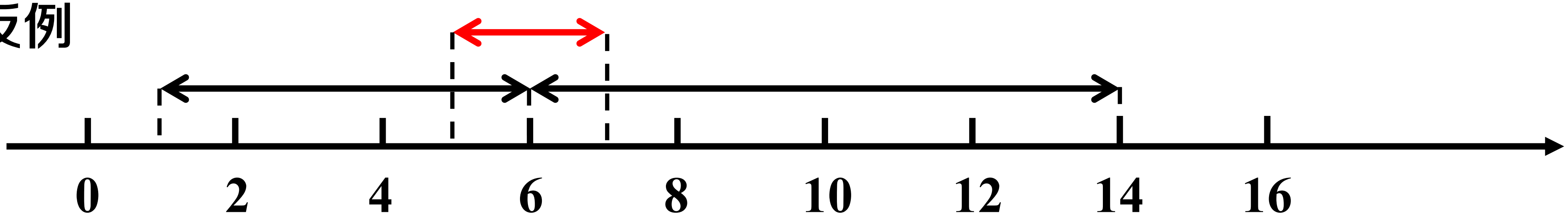
- 策略2：最早开始活动优先
  - 反例



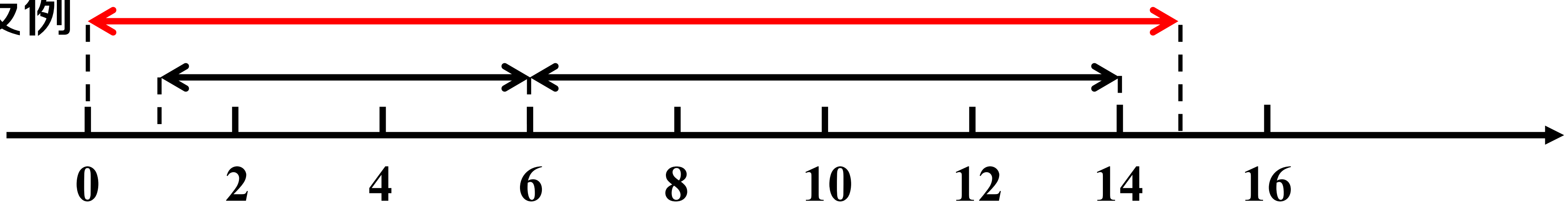
- 策略3：最早结束活动优先

贪心策略

- 策略1：最短活动优先
  - 反例



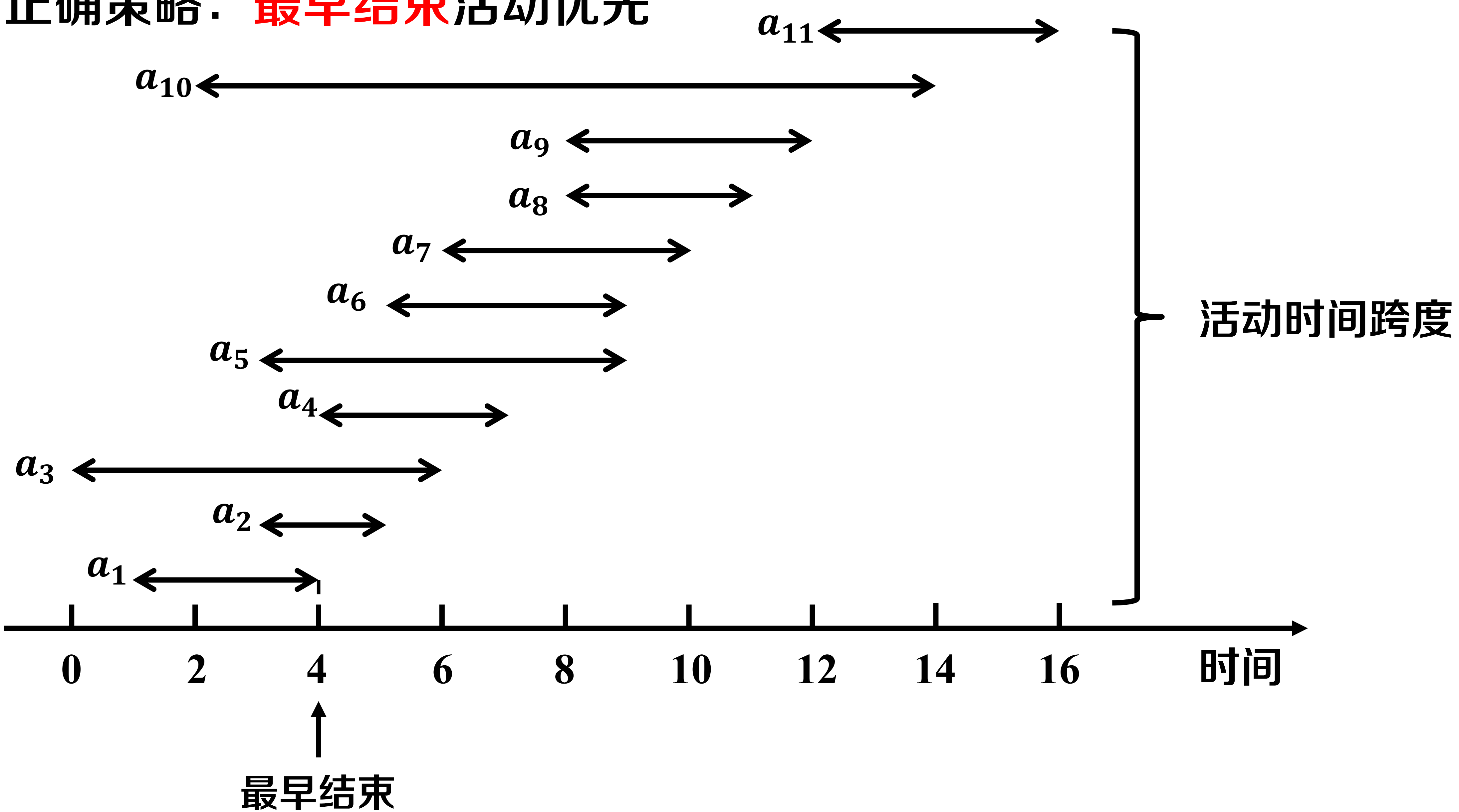
- 策略2：最早开始活动优先
  - 反例



- 策略3：最早结束活动优先
  - 选择最早结束的活动，可以给后面的活动留更大的选择空间

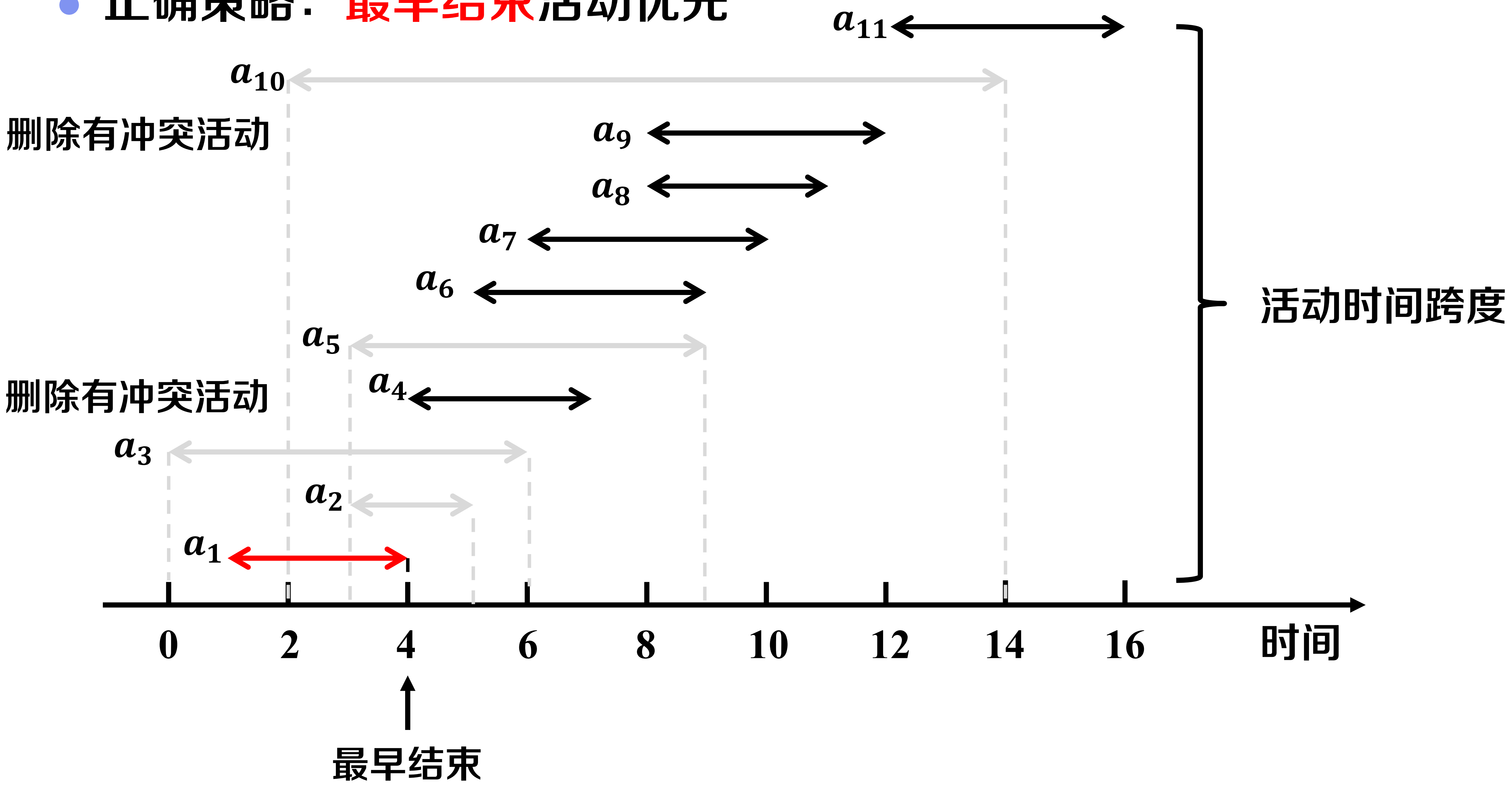
算法实例

- 正确策略：**最早结束**活动优先



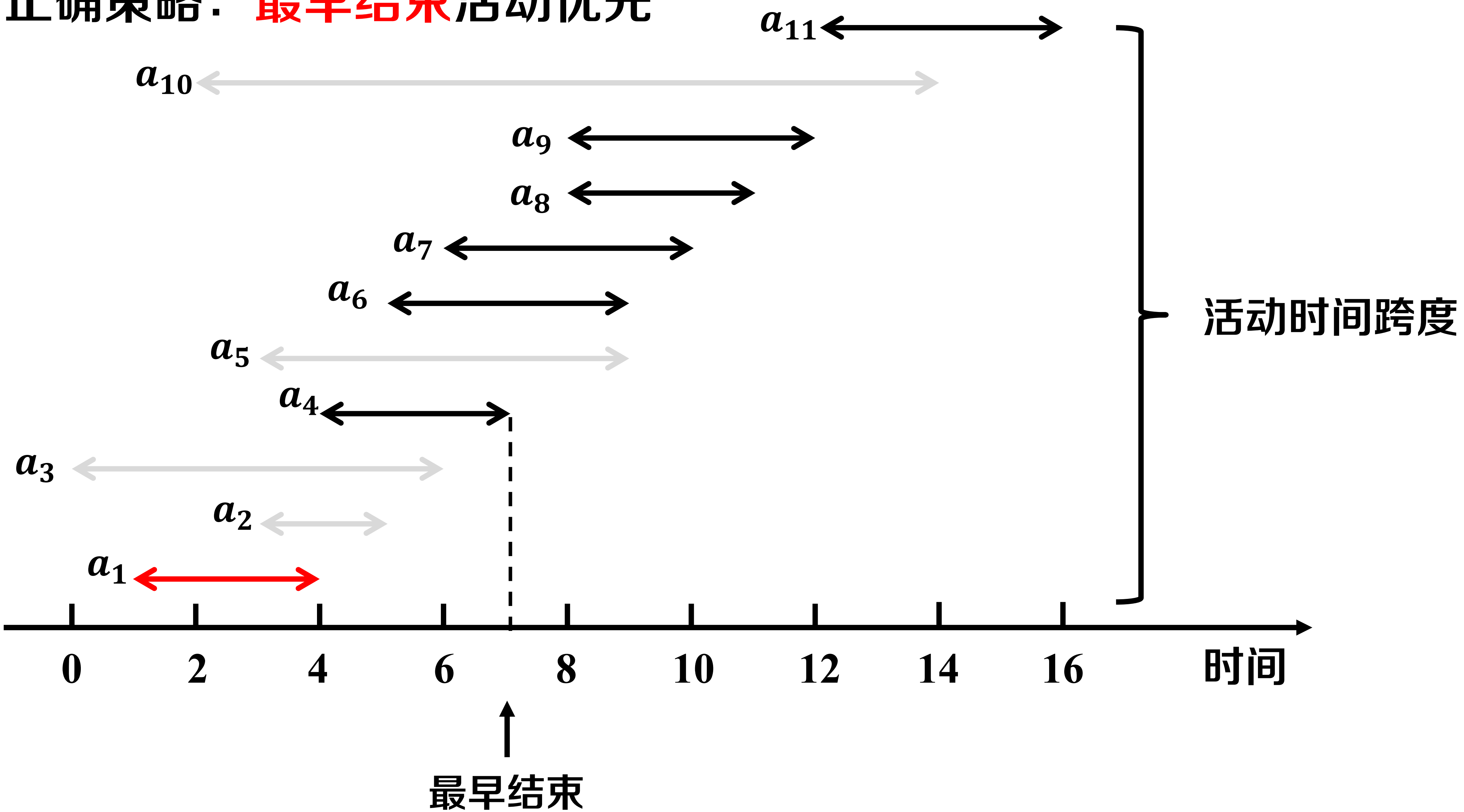
算法实例

- 正确策略：**最早结束**活动优先



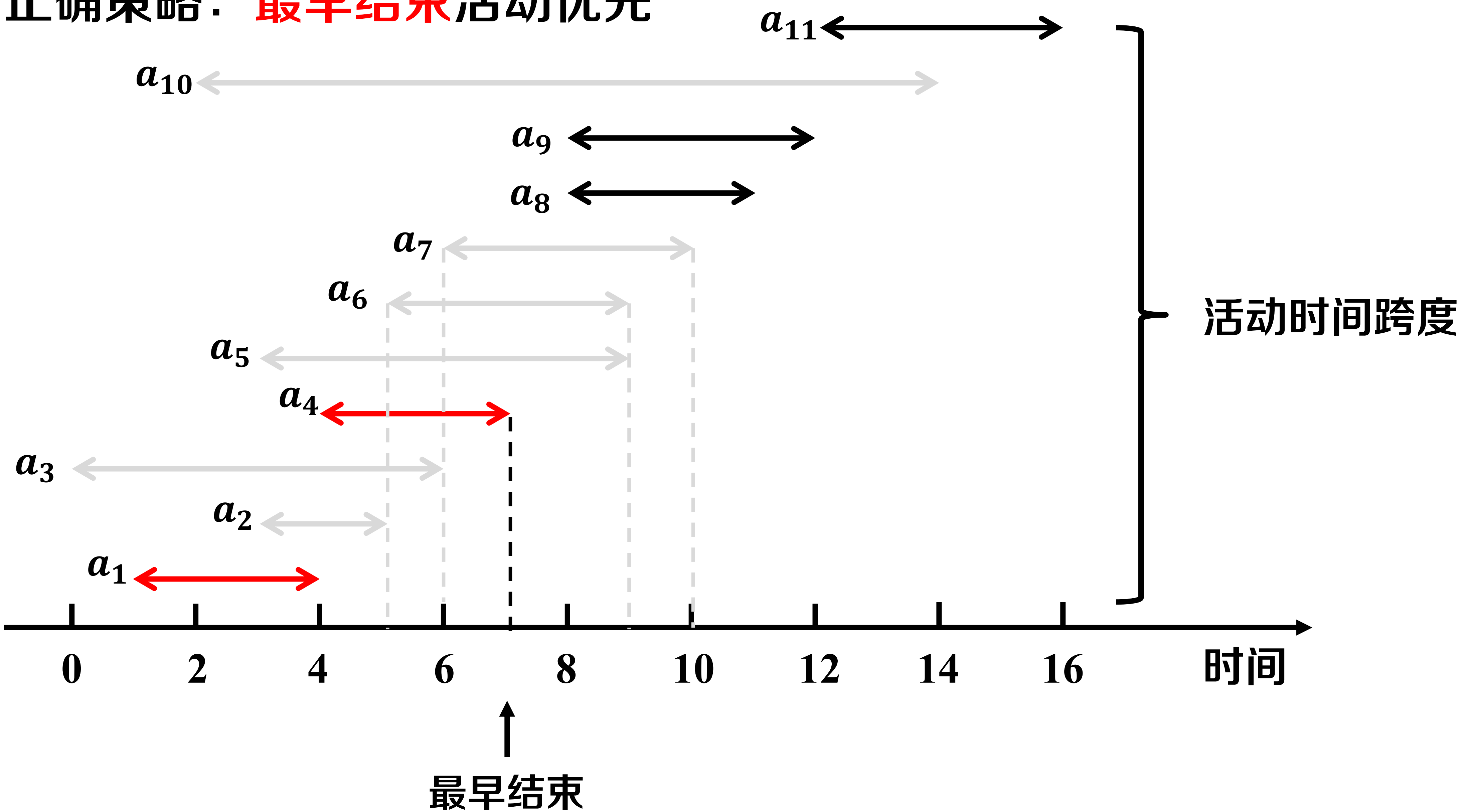
算法实例

- 正确策略：**最早结束**活动优先



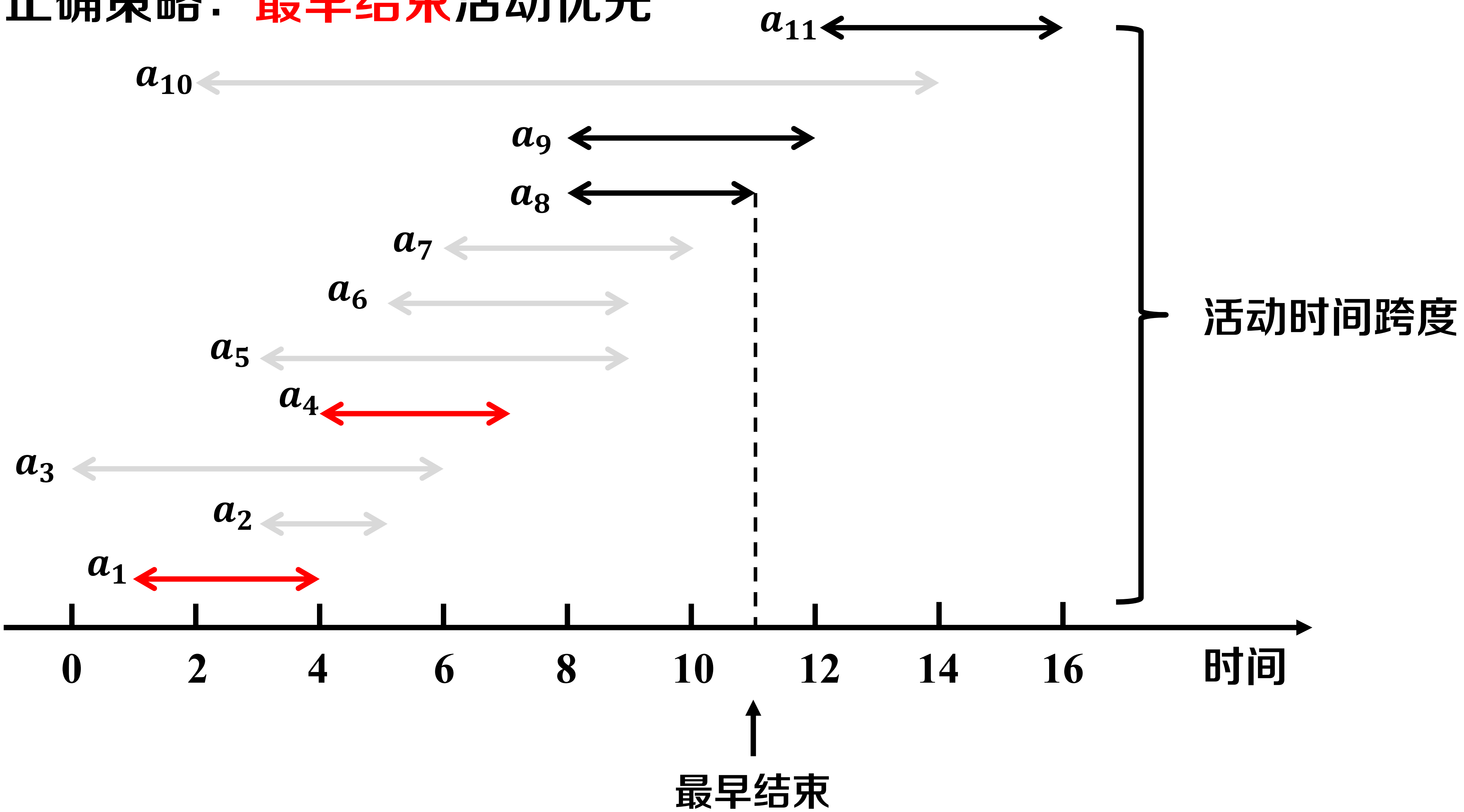
算法实例

- 正确策略：**最早结束**活动优先



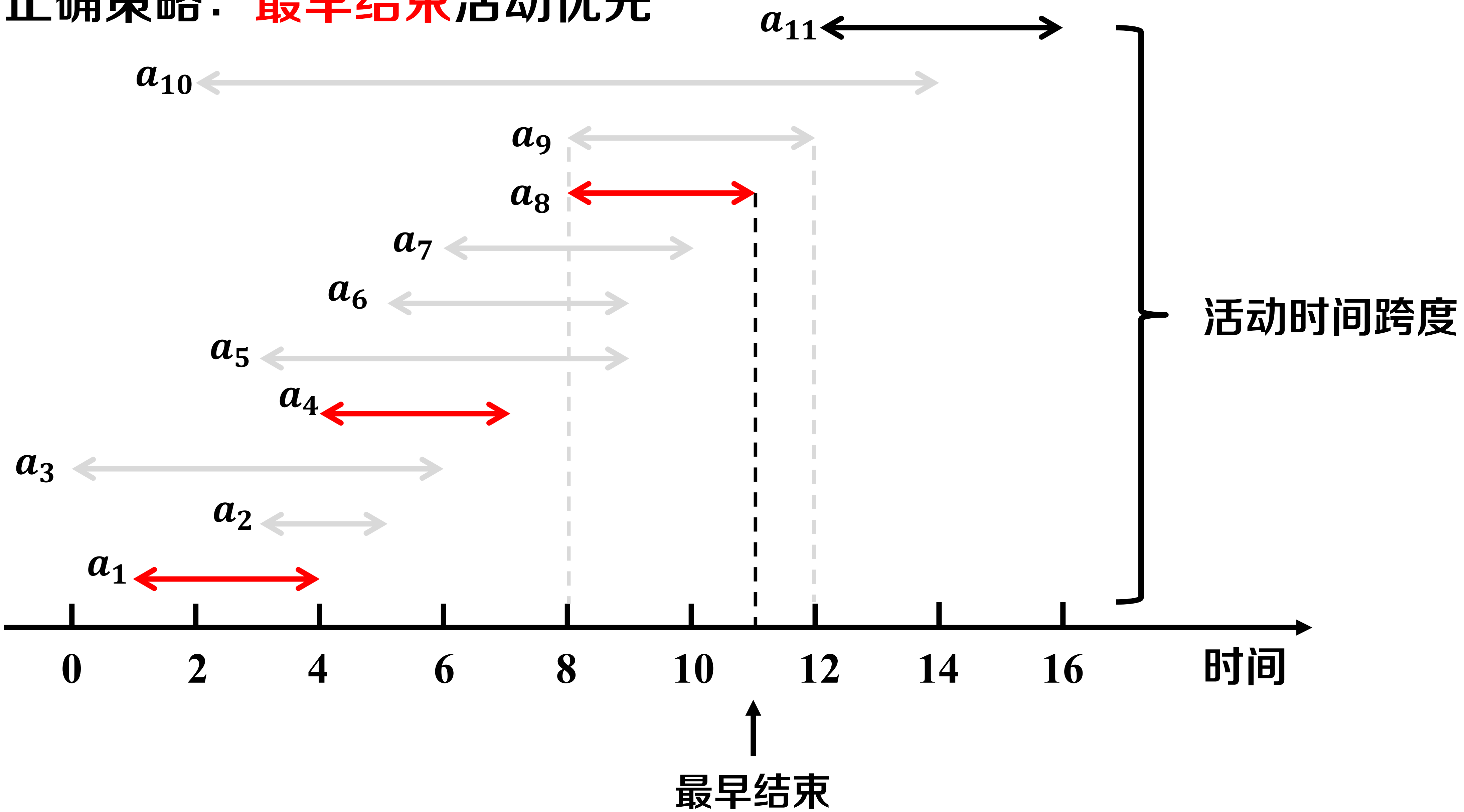
算法实例

- 正确策略：最早结束活动优先



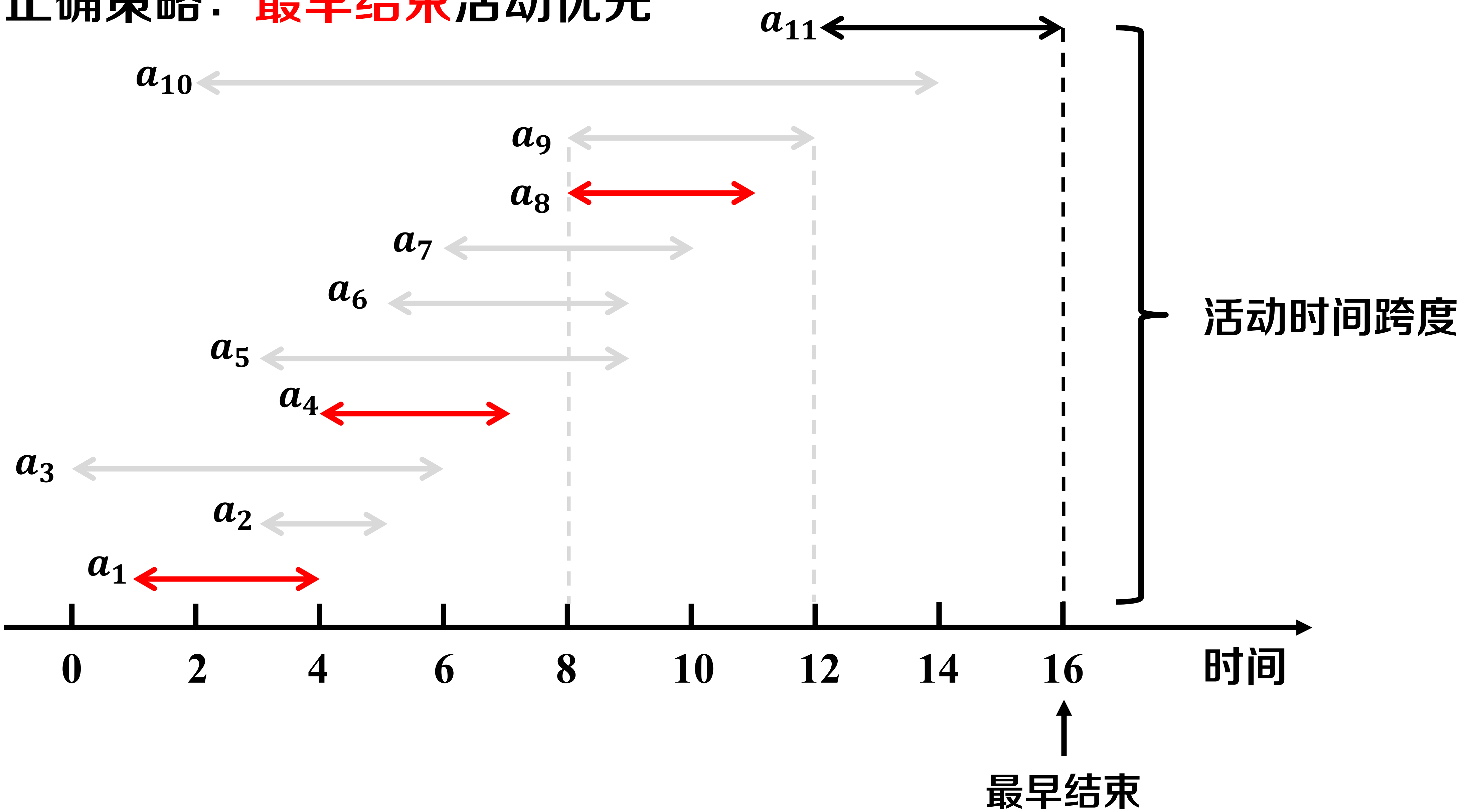
算法实例

- 正确策略：**最早结束**活动优先



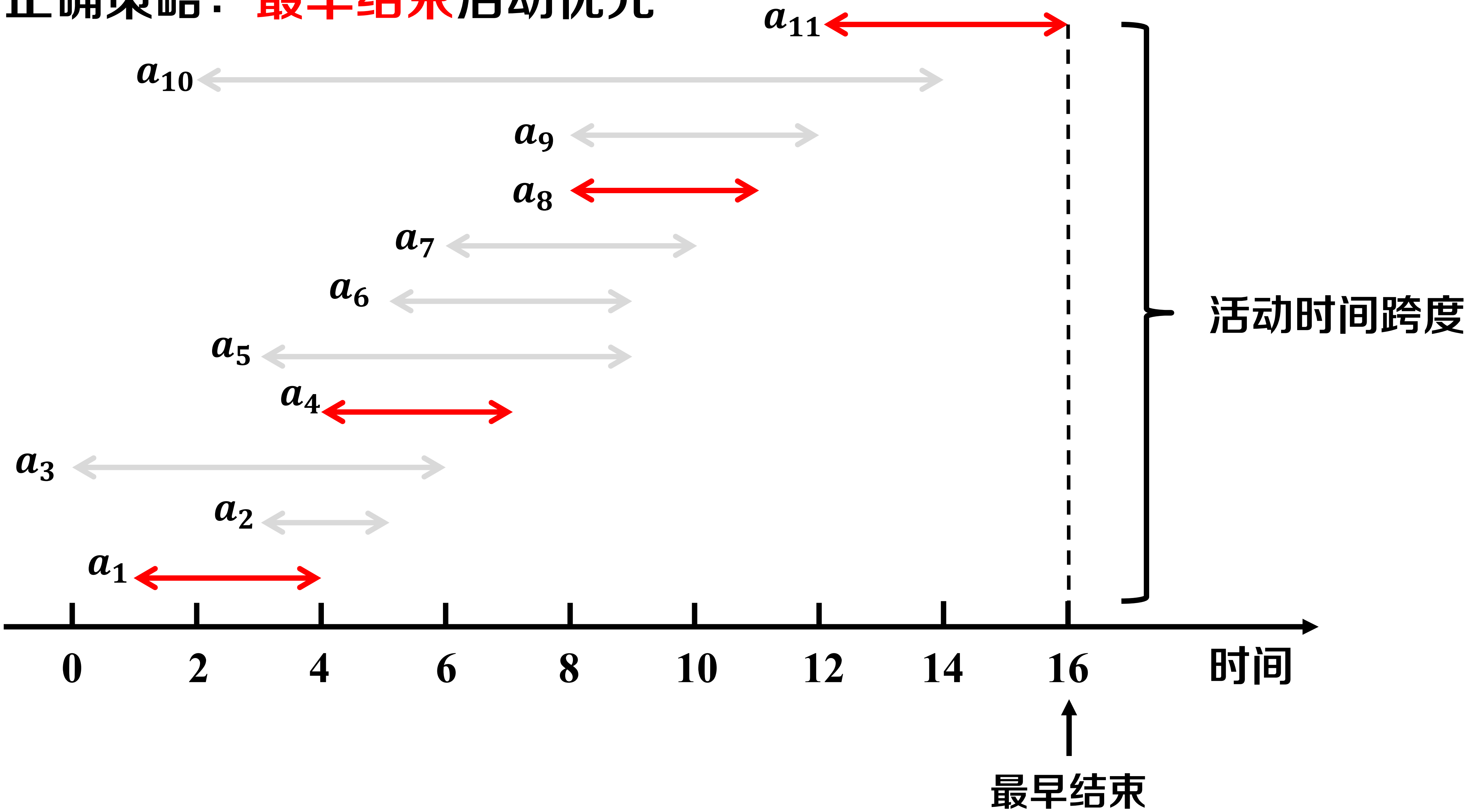
算法实例

- 正确策略：**最早结束**活动优先



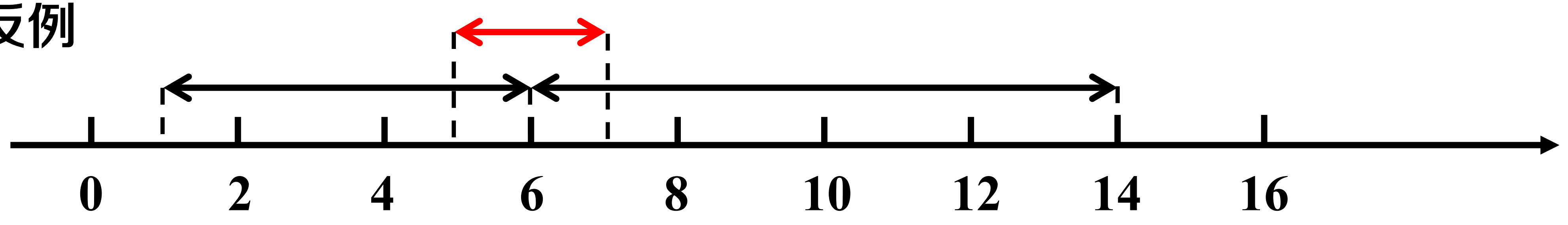
算法实例

- 正确策略：**最早结束**活动优先

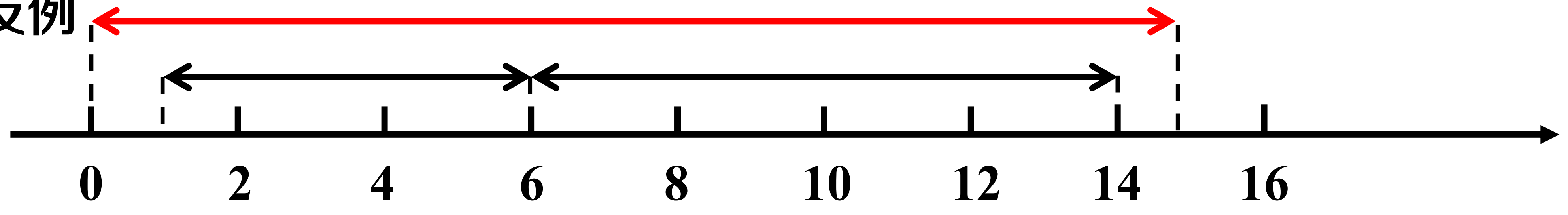


贪心策略

- 策略1：**最短**活动优先
  - 反例



- 策略2：**最早开始**活动优先
  - 反例



- 策略3：**最早结束**活动优先
  - 选择最早结束的活动，可以给后面的活动留更大的选择空间

问题：策略3是否可以保证最优解？

贪心策略：一般步骤

提出贪心策略

观察问题特征，构造贪心选择

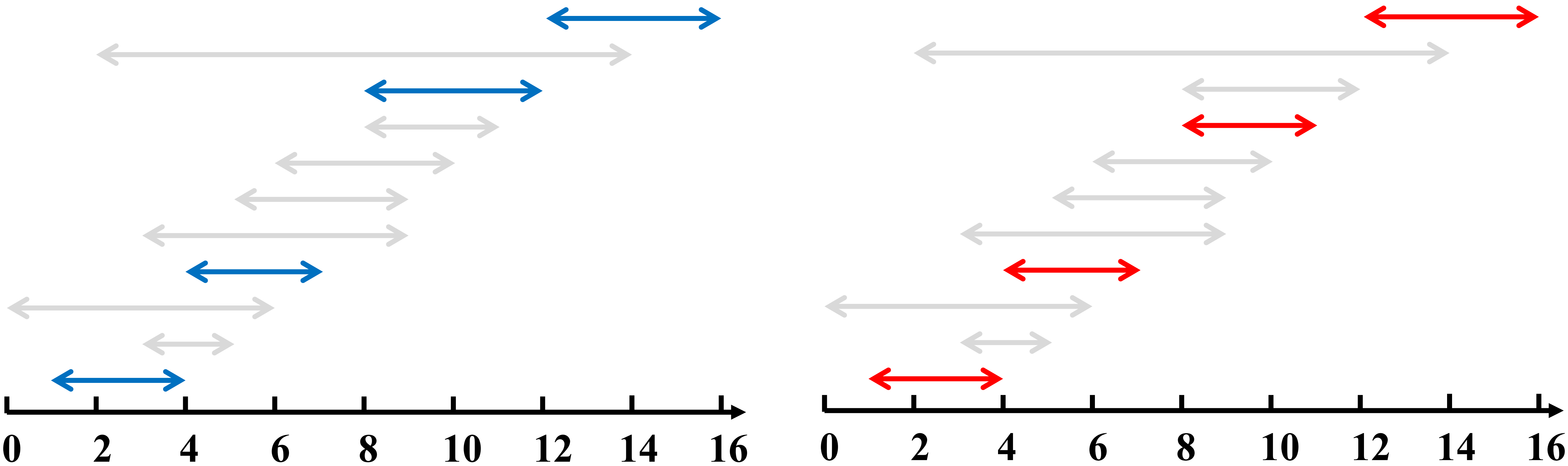
证明策略正确

假设最优方案，通过替换证明

正确性证明

- 正确策略：**最早结束**活动优先
- 证明：贪心解不劣于最优解

任意最优活动集合 —依次检查并替换→ 贪心所得活动集合

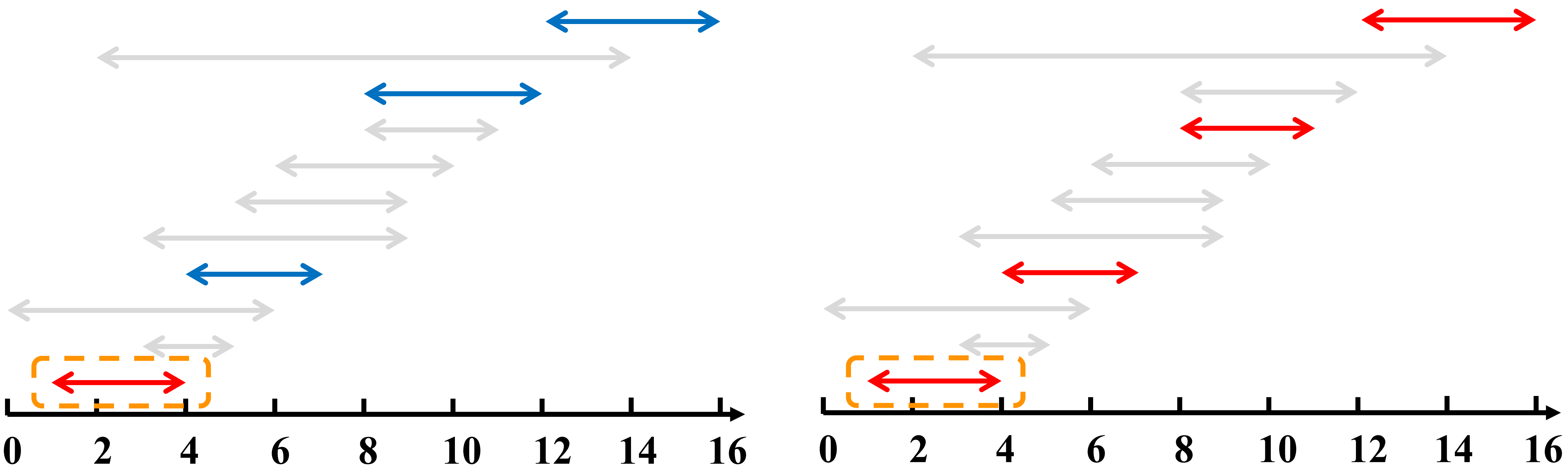


正确性证明

- 正确策略：**最早结束**活动优先
- 证明：贪心解不劣于最优解

任意最优活动集合 —依次检查并替换→ 贪心所得活动集合

活动相同，无需替换

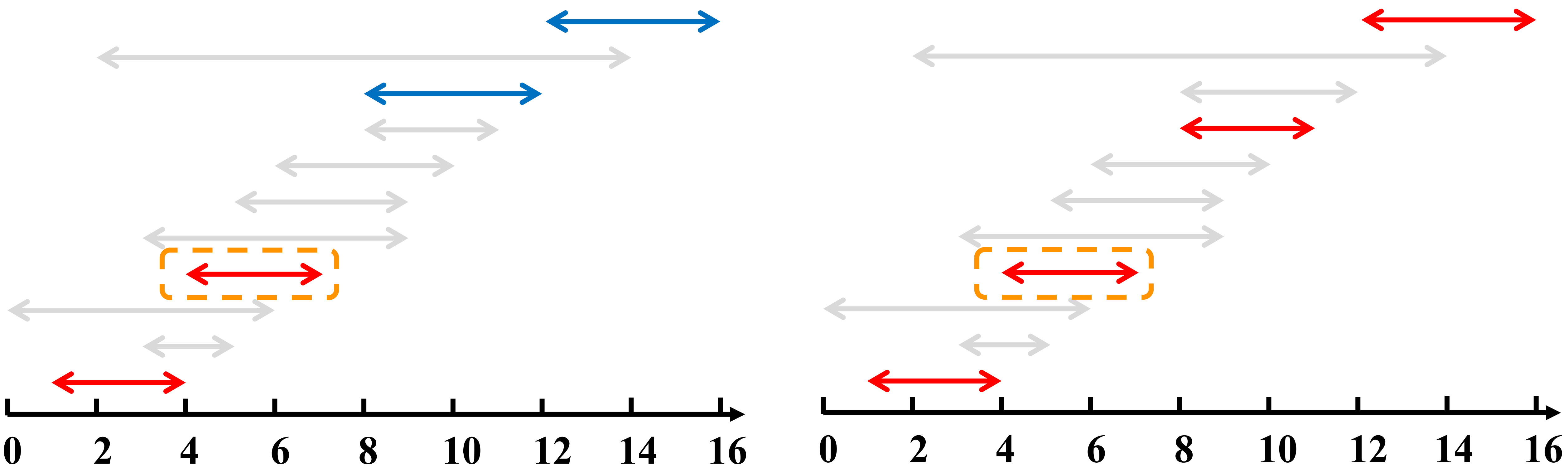


正确性证明

- 正确策略：**最早结束**活动优先
- 证明：贪心解不劣于最优解

任意最优活动集合 —依次检查并替换→ 贪心所得活动集合

活动相同，无需替换

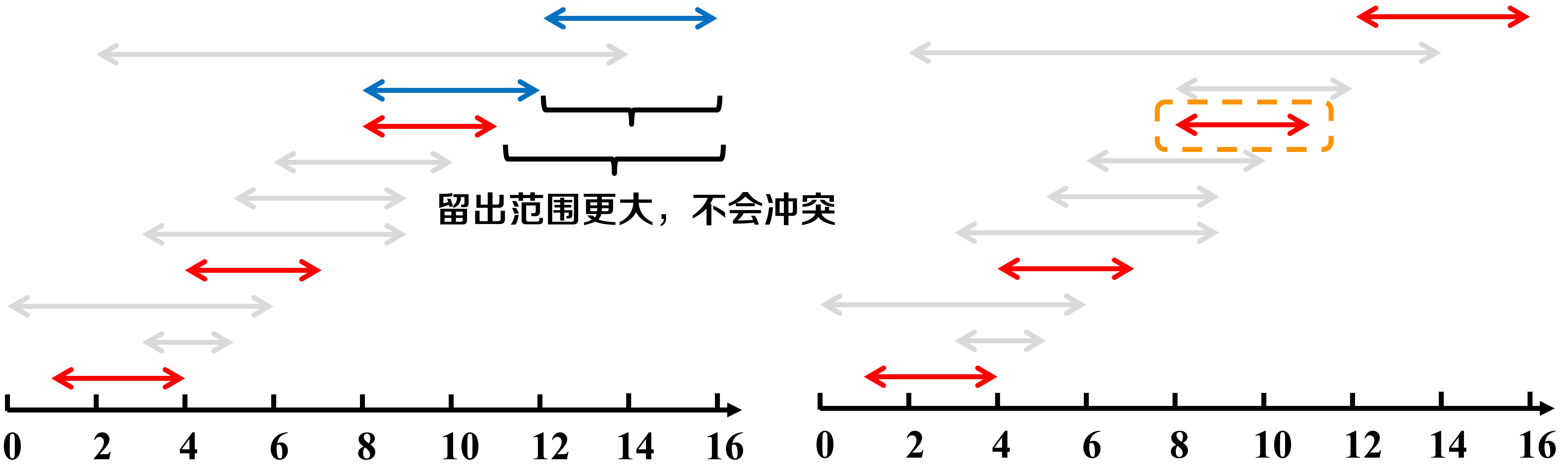


正确性证明

- 正确策略：**最早结束**活动优先
- 证明：贪心解不劣于最优解

任意最优活动集合 → 依次检查并替换 → 贪心所得活动集合

活动相同，无需替换
活动不同，必能替换

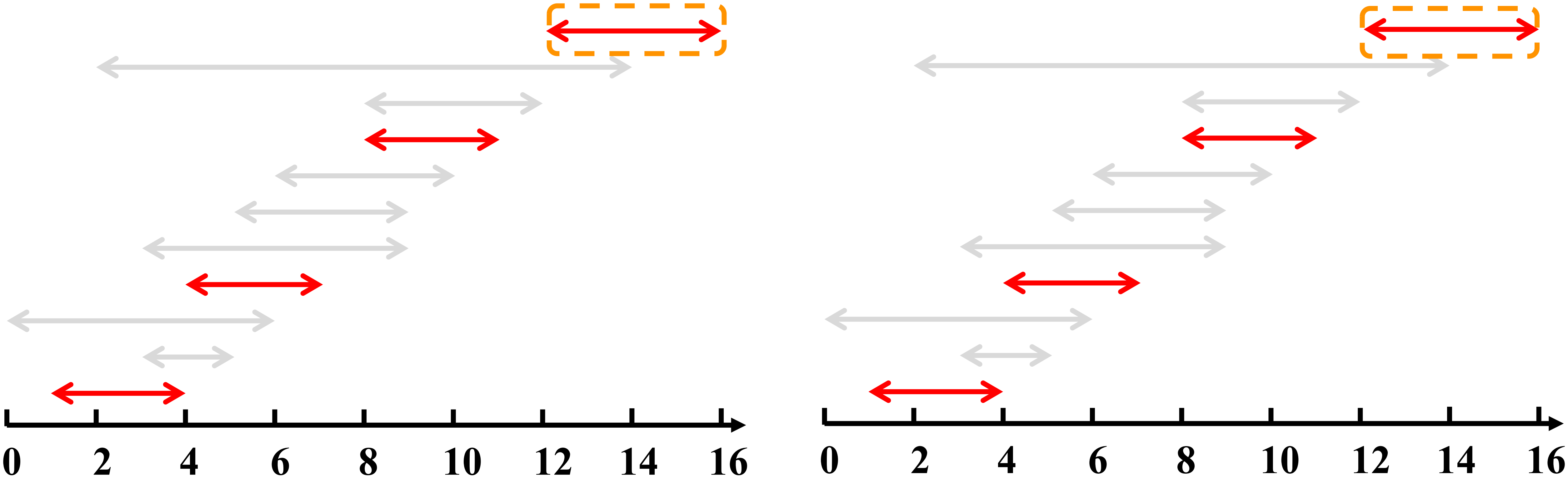


正确性证明

- 正确策略：**最早结束**活动优先
- 证明：贪心解不劣于最优解

任意最优活动集合 → 依次检查并替换 → 贪心所得活动集合

活动相同，无需替换
活动不同，必能替换



贪心算法：伪代码

输入: 活动集合$S = \{a_1, a_2, ..., a_n\}$, 每个活动$a_i$的起止时间$s_i, f_i$
输出: 不冲突活动的最大子集S'

把活动按照结束时间升序排序

为使用贪心策略作准备

```
S' ← {a_1}
k ← 1
for i ← 2 to n do
    if s_i ≥ f_k then
        S' ← S' ∪ {a_i}
        k ← i
    end
end
return S'
```

贪心算法：伪代码

输入: 活动集合$S=\{a_1,a_2,\ldots,a_n\}$, 每个活动a_i的起止时间s_i, f_i
输出: 不冲突活动的最大子集S'
把活动按照结束时间升序排序

```
S' ← {a_1}
k ← 1
for i ← 2 to n do
    if s_i ≥ f_k then
        S' ← S' ∪ {a_i}
        k ← i
    end
end
return S'
```

把最早结束活动加入到集合

贪心算法：伪代码

输入: 活动集合$S = \{a_1, a_2, ..., a_n\}$, 每个活动$a_i$的起止时间$s_i$, f_i
输出: 不冲突活动的最大子集S'
把活动按照结束时间升序排序

$S' \leftarrow \{a_1\}$
$k \leftarrow 1$
for $i \leftarrow 2$ *to* n **do**
  if $s_i \geq f_k$ **then**
    $S' \leftarrow S' \cup \{a_i\}$
    $k \leftarrow i$
  end
end
return S'

记录当前选择的活动

贪心算法：伪代码

输入: 活动集合$S=\{a_1,a_2,\dots,a_n\}$, 每个活动a_i的起止时间s_i, f_i
输出: 不冲突活动的最大子集S'
把活动按照结束时间升序排序

```
S' ← {a_1}
k ← 1
for i ← 2 to n do
    if s_i ≥ f_k then
        S' ← S' ∪ {a_i}
        k ← i
    end
end
return S'
```

检查每个活动

贪心算法：伪代码

输入: 活动集合$S = \{a_1, a_2, ..., a_n\}$, 每个活动$a_i$的起止时间$s_i, f_i$
输出: 不冲突活动的最大子集S'
把活动按照结束时间升序排序

```
S' ← {a_1}
k ← 1
for i ← 2 to n do
    if s_i ≥ f_k then
        S' ← S' ∪ {a_i}
        k ← i
    end
end
return S'
```

没有冲突，则加入子集

贪心算法：伪代码

输入: 活动集合$S = \{a_1, a_2, ..., a_n\}$, 每个活动$a_i$的起止时间$s_i$, f_i
输出: 不冲突活动的最大子集S'
把活动按照结束时间升序排序

```
S' ← {a_1}
k ← 1
for i ← 2 to n do
    if s_i ≥ f_k then
        S' ← S' ∪ {a_i}
        k ← i
    end
end
return S'
```

更新当前选择的活动

贪心算法：复杂度分析

输入: 活动集合$S = \{a_1, a_2, ..., a_n\}$, 每个活动$a_i$的起止时间$s_i, f_i$
输出: 不冲突活动的最大子集S'

把活动按照结束时间升序排序 ------------ $O(n \log n)$

$S' \leftarrow \{a_1\}$
$k \leftarrow 1$
for $i \leftarrow 2$ *to* n **do**
　　if $s_i \geq f_k$ **then**
　　　　$S' \leftarrow S' \cup \{a_i\}$
　　　　$k \leftarrow i$
　　end
end
return S'

$O(n)$

时间复杂度：$O(n \log n)$

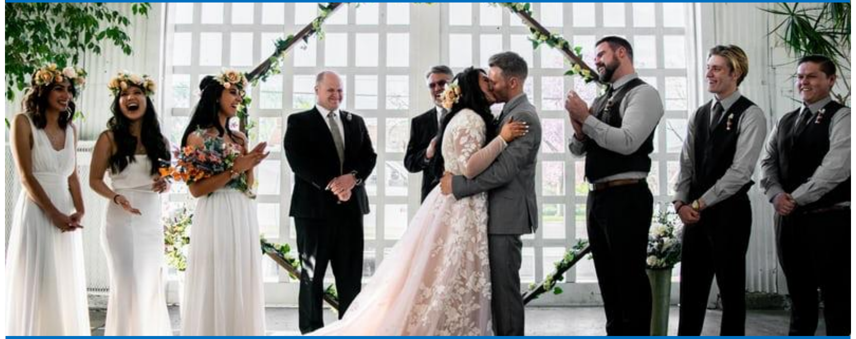
问题拓展

- 会场出租

收益很大

公司年会：10:00～19:00

收益良好



婚礼宴请：11:00～14:00

收益较多

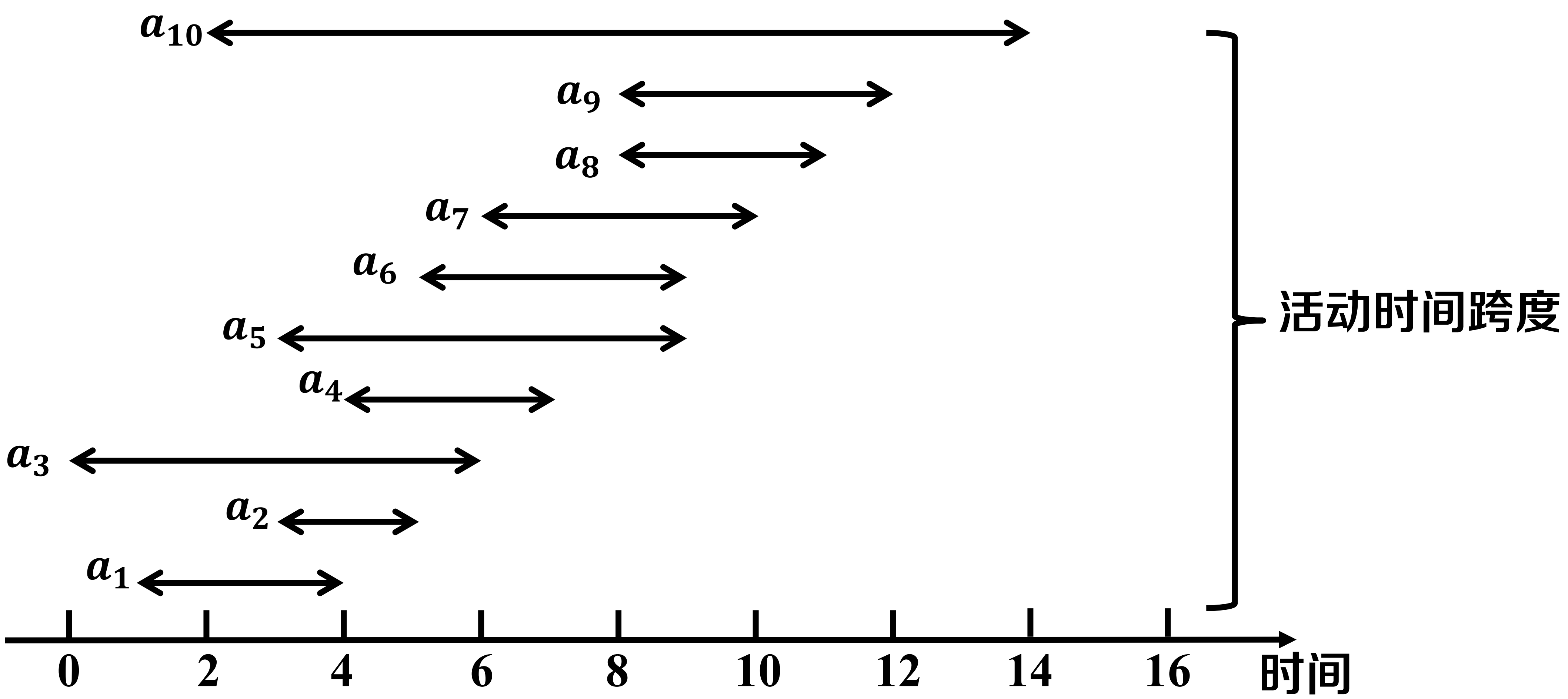
生日聚会：12:00～17:00

收益较少

学术研讨：14:00～16:00

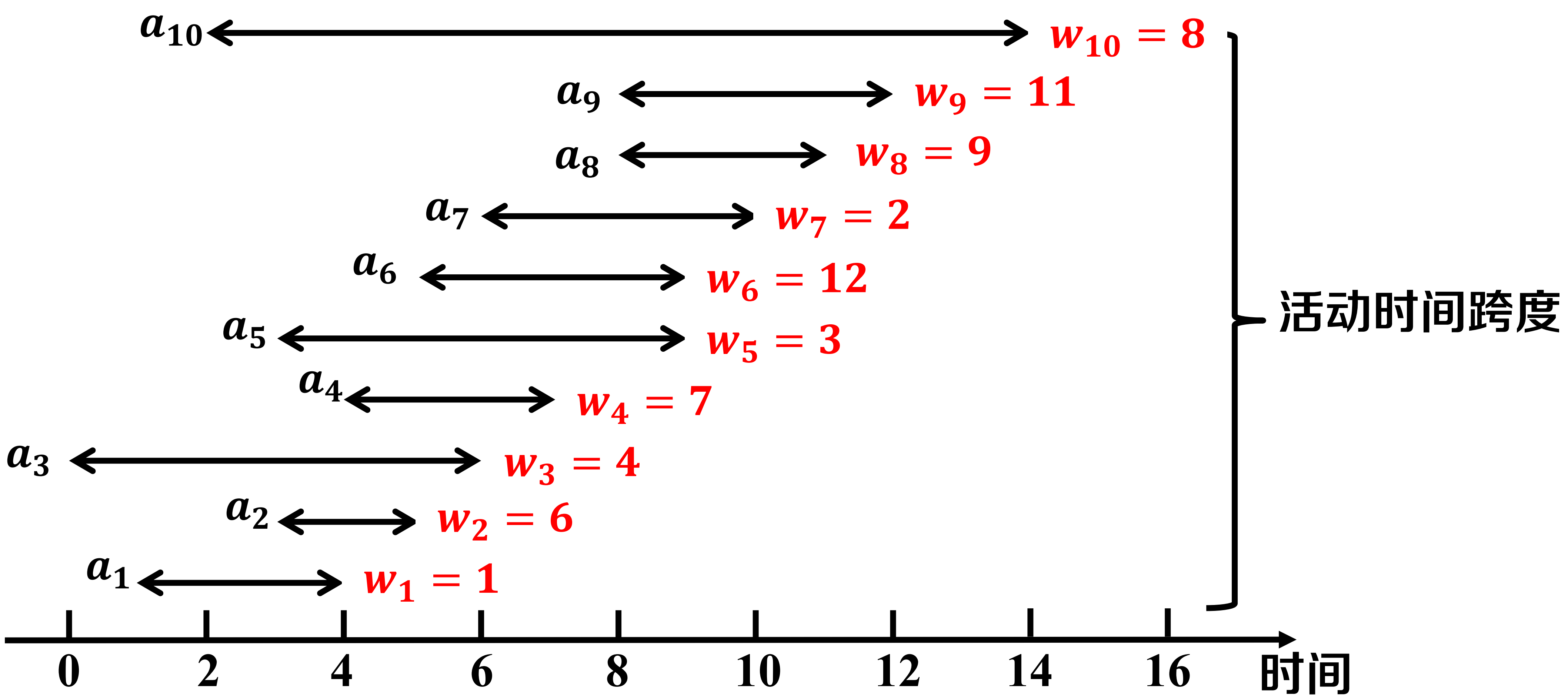
问题拓展

- 会场出租
  - 选择出租的活动时间不能冲突



问题拓展

- 会场出租
  - 选择出租的活动时间不能冲突，活动出租收益各不相同
  - 怎样选让收益总和最大?



问题定义

带权活动选择问题

Weighted Activity Selection Problem

输入

- n个活动组成的集合$S = \{a_1, a_2, \dots, a_n\}$
- 每个活动a_i的开始时间s_i，结束时间f_i和权重w_i

输出

- 找出活动集合S的子集S'，令

优化目标：最大化权重之和

$$\max \sum_{a_i \in S'} w_i$$

约束条件

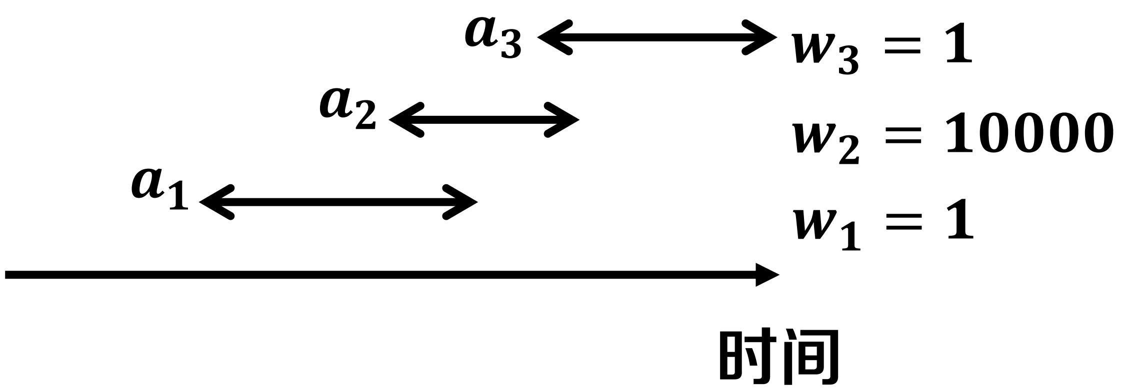
$$s.t. \forall a_i, a_j \in S', s_i \geq f_j \text{或} s_j \geq f_i$$

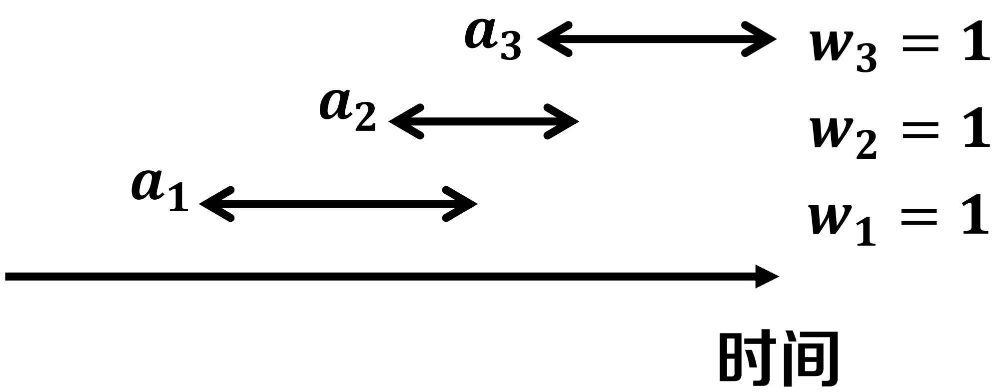
问题比较

带权活动选择问题	权重均为1 →	活动选择问题
$\max \sum_{a_i \in S'} w_i$		$\max \lvert S' \rvert$

问题比较

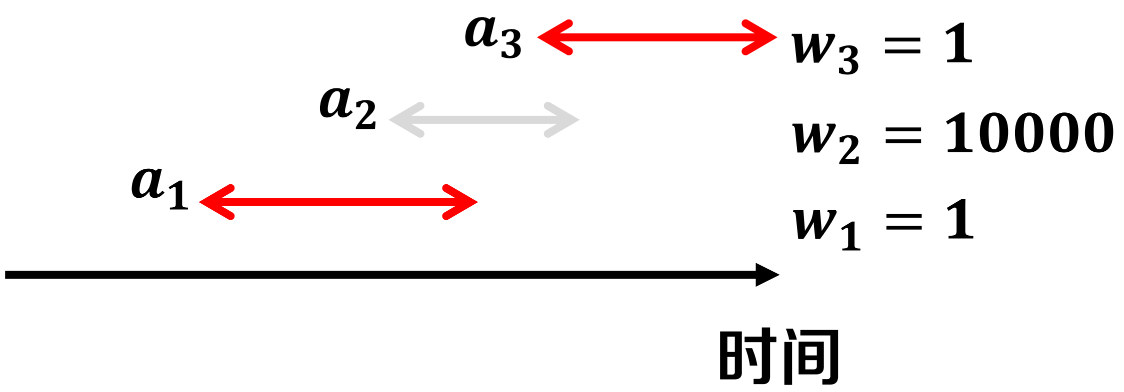
带权活动选择问题	权重均为1 →	活动选择问题
$\max \sum_{a_i \in S'} w_i$		$\max\lvert S'\rvert$

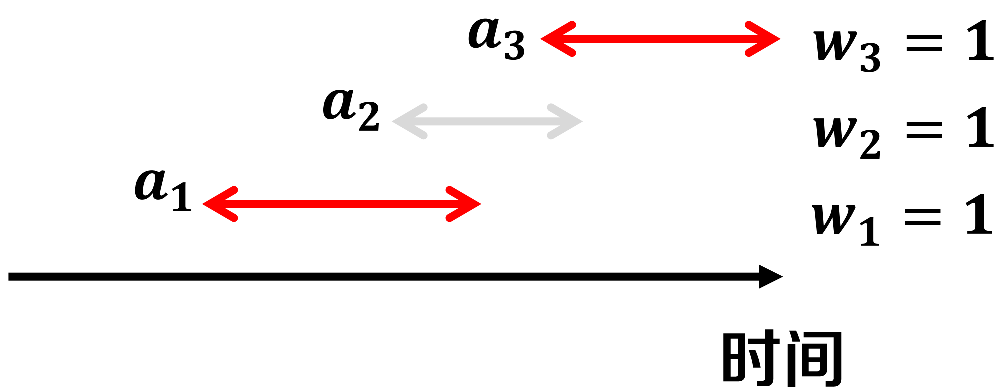




问题比较

带权活动选择问题	权重均为1 →	活动选择问题
$\max \sum_{a_i \in S'} w_i$		$\max \lvert S' \rvert$





问题比较

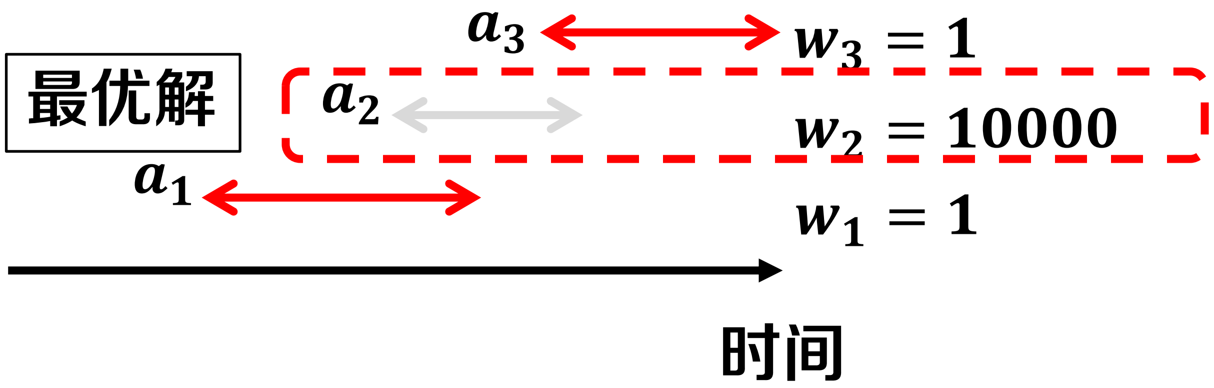
带权活动选择问题

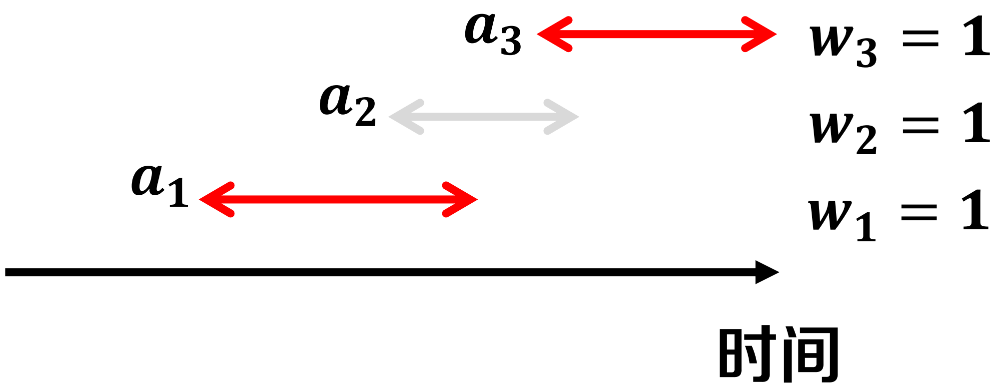
$$\max \sum_{a_i \in S'} w_i$$

权重均为1 →

活动选择问题

$$\max |S'|$$

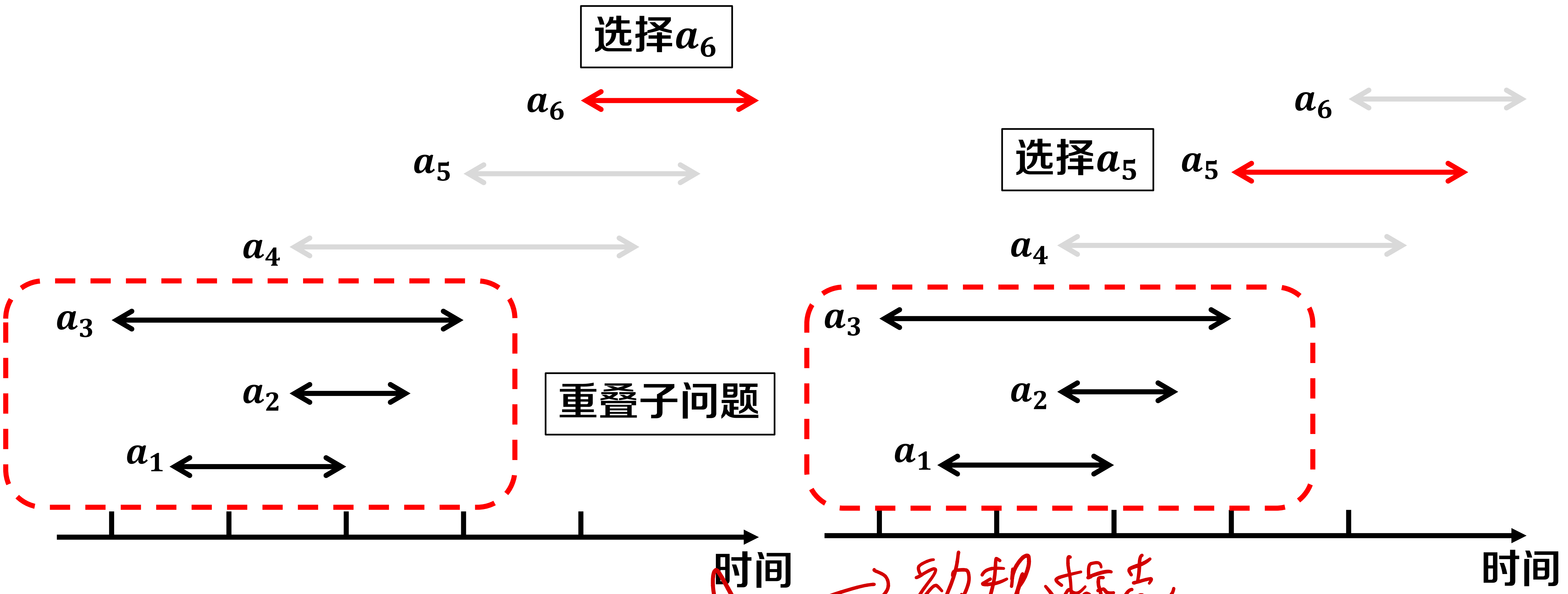




贪心策略不正确

贪心策略正确

从贪心策略到动态规划

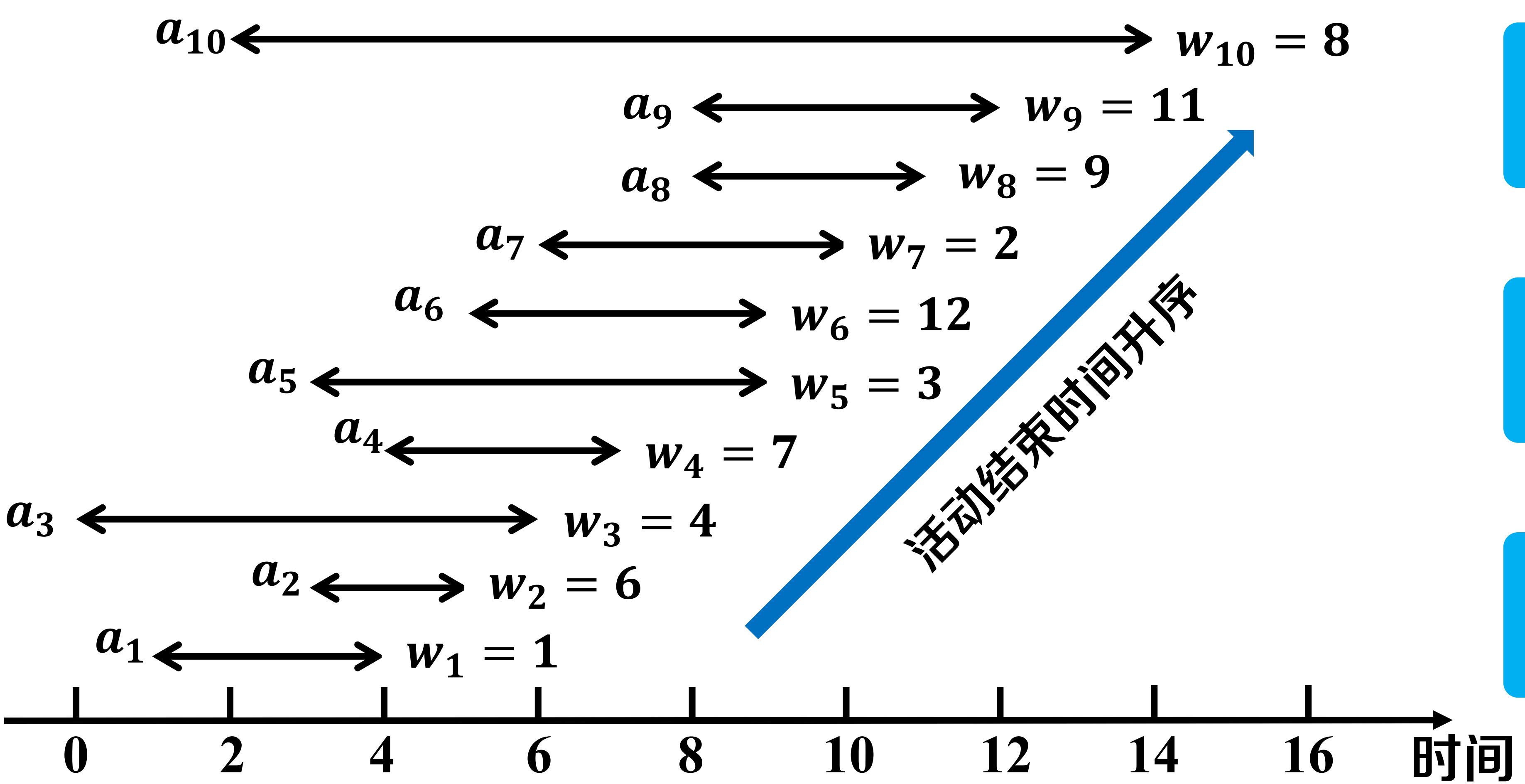


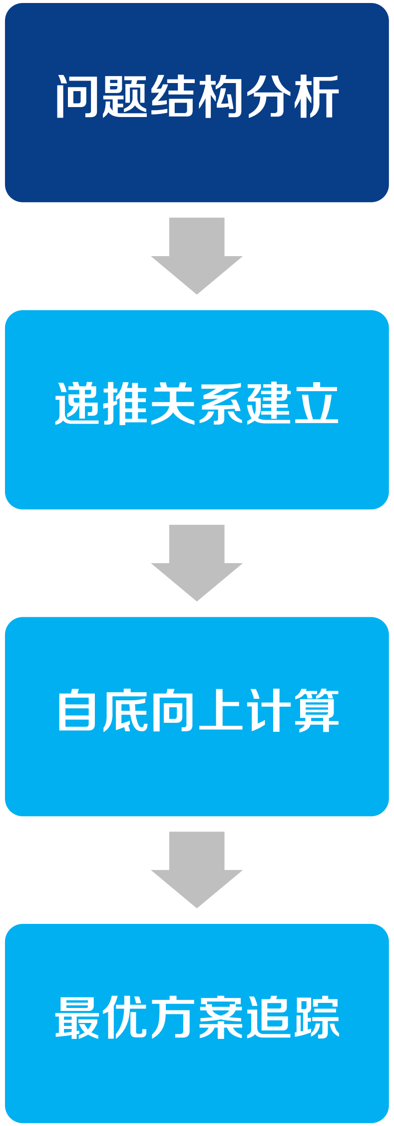
动规标志，

存在重叠子问题，使用动态规划求解

问题结构分析

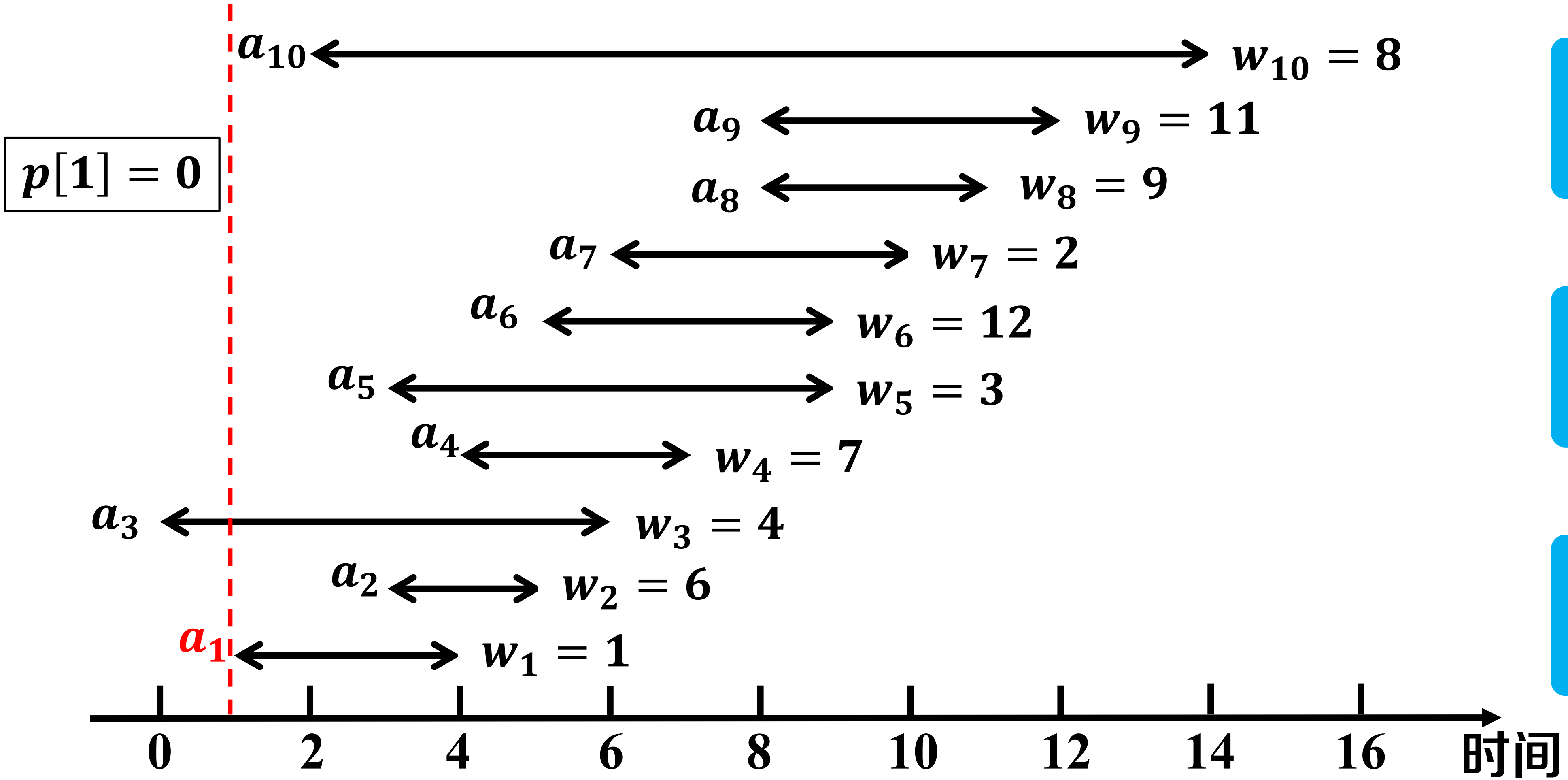
- 预处理
  - 排序：按活动结束时间升序

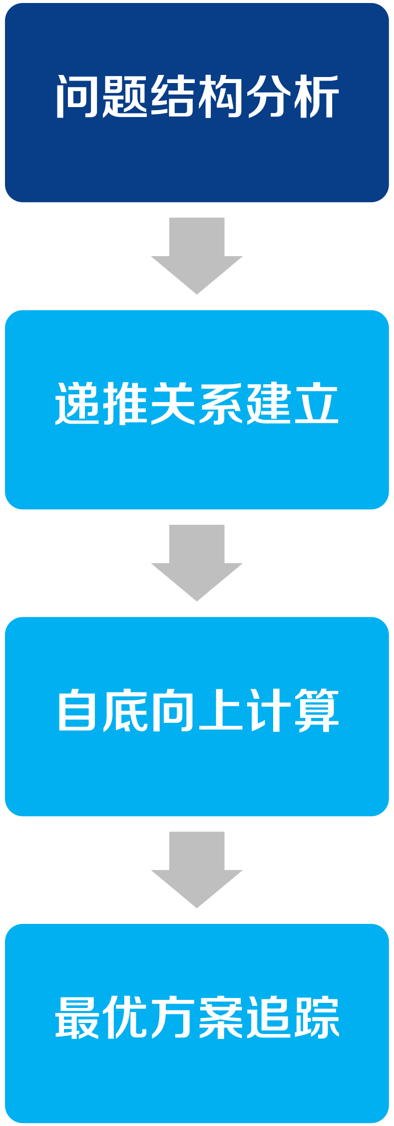




问题结构分析

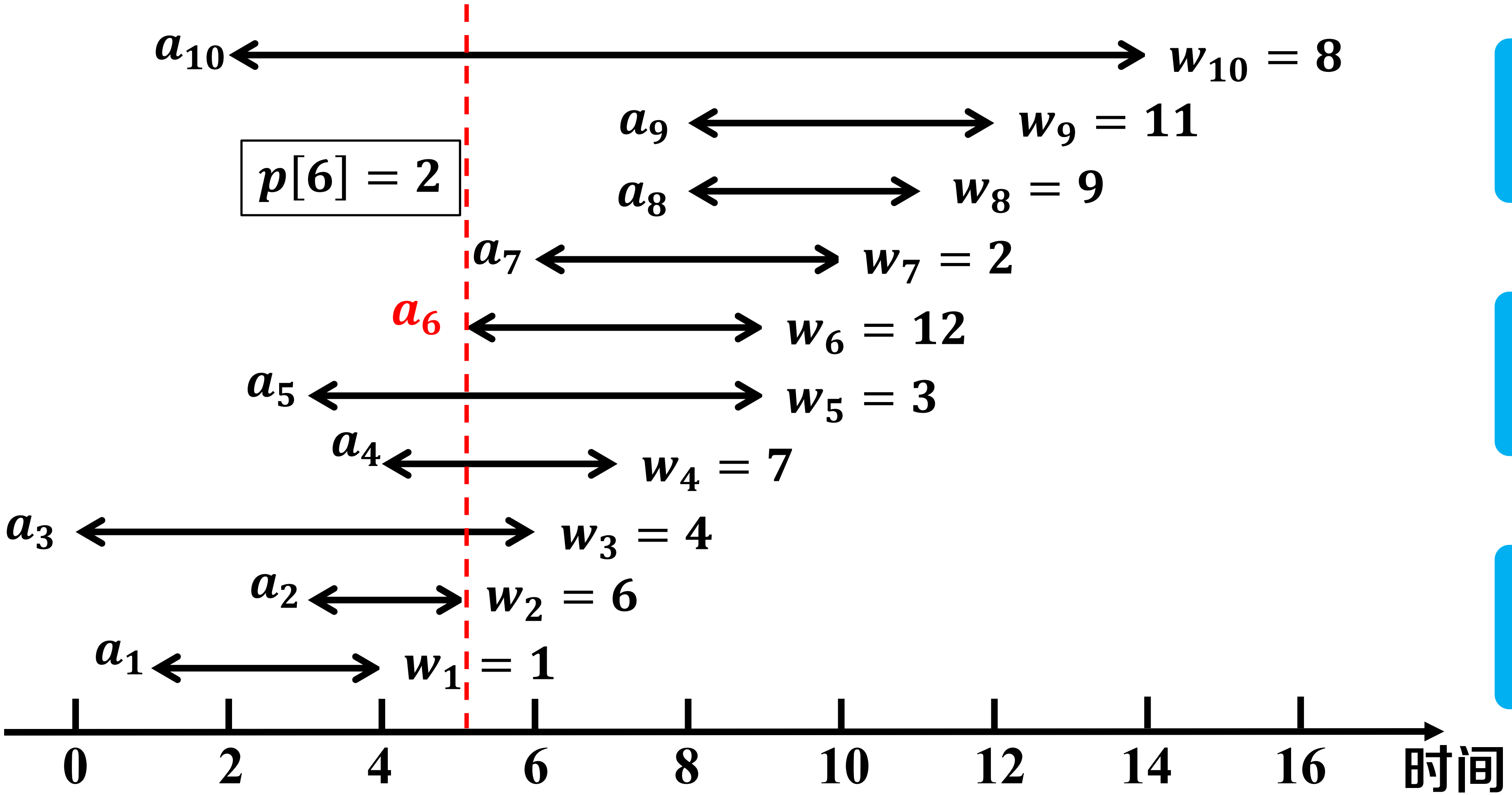
- 预处理
  - 排序：按活动结束时间升序
  - 求$p[i]$：在a_i开始前最后结束的活动

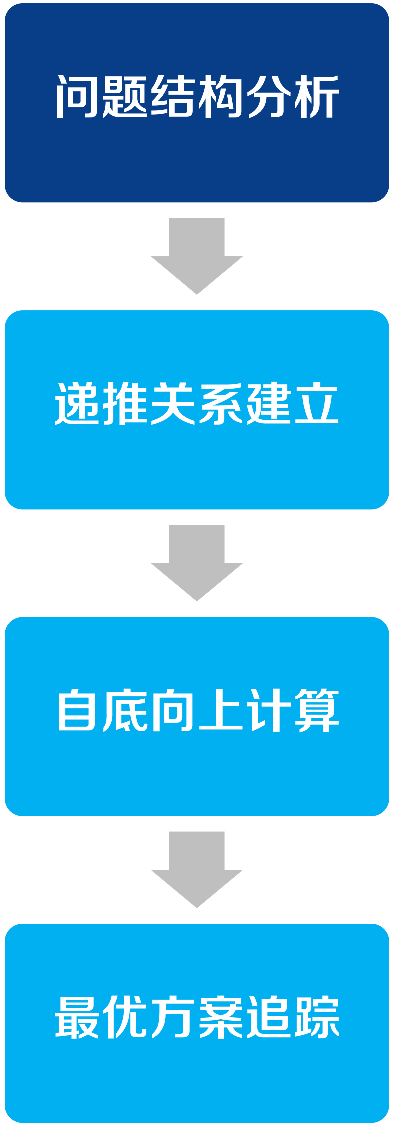




问题结构分析

- 预处理
  - 排序：按活动结束时间升序
  - 求$p[i]$：在a_i开始前最后结束的活动





问题结构分析

- 预处理
  - 排序：按活动结束时间升序
  - 求$p[i]$：在a_i开始前最后结束的活动
    - 如何求解$p[i]$?
    - 排序后使用二分查找 $\log n$

问题结构分析 → 递推关系建立 → 自底向上计算 → 最优方案追踪

问题结构分析

- 预处理
  - 排序：按活动结束时间升序
  - 求$p[i]$：在a_i开始前最后结束的活动

加速递归查找过程.

- 给出问题表示
  - $D[i]$：集合$\{a_1, a_2, a_3, \dots, a_i\}$中不冲突活动最大权重和
- 明确原始问题
  - $D[n]$：集合$\{a_1, a_2, a_3, \dots, a_n\}$中不冲突活动最大权重和

问题结构分析

递推关系建立

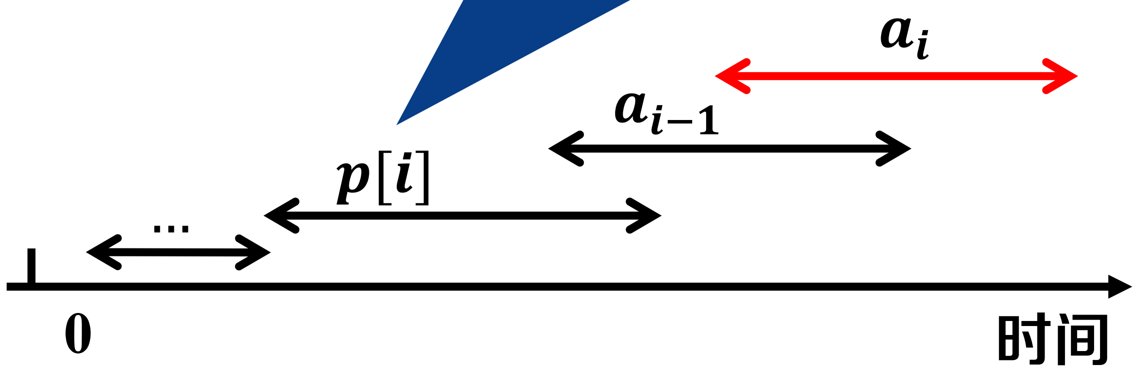
自底向上计算

最优方案追踪

递推关系建立：分析最优（子）结构

- 考察活动a_i
  - 选择a_i

在a_i开始前最后结束的活动



问题结构分析

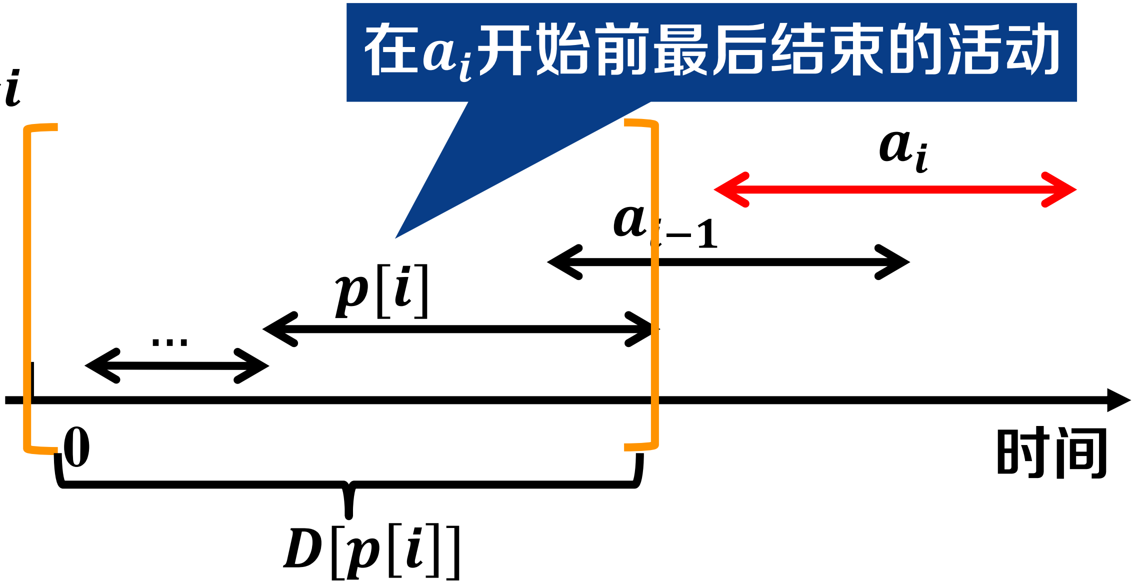
递推关系建立

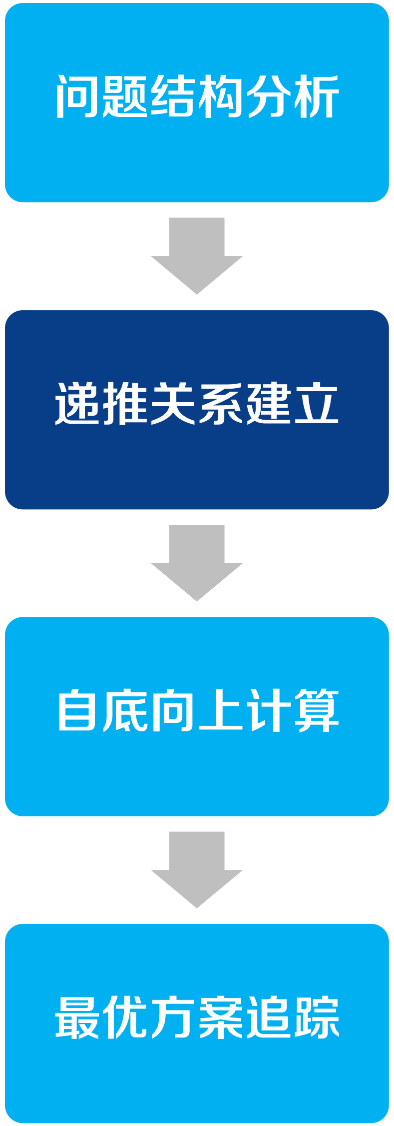
自底向上计算

最优方案追踪

递推关系建立：分析最优（子）结构

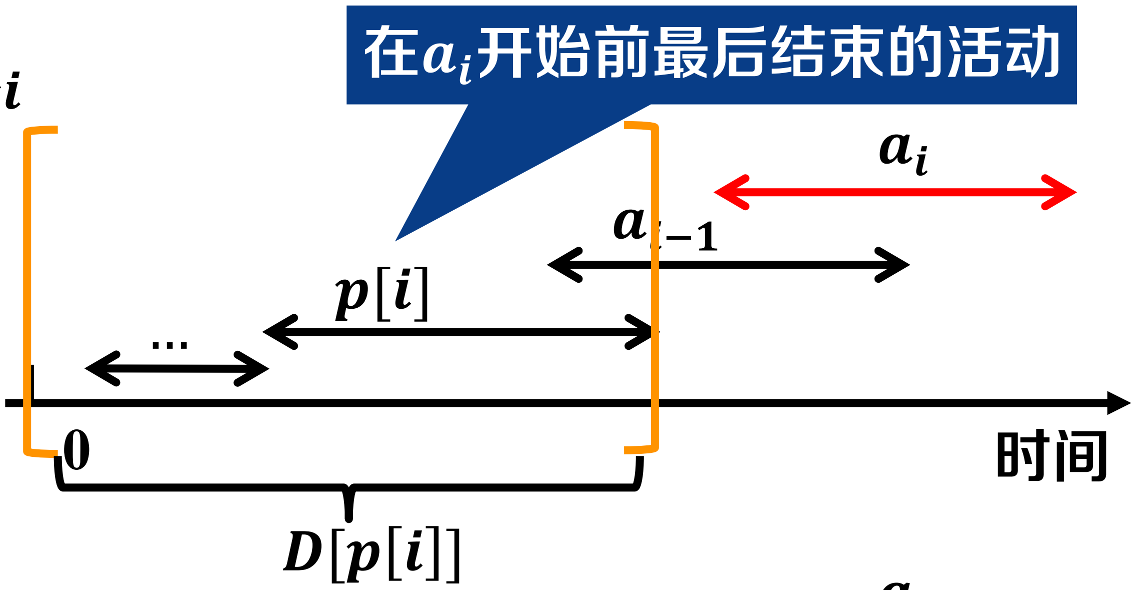
- 考察活动a_i
  - 选择a_i



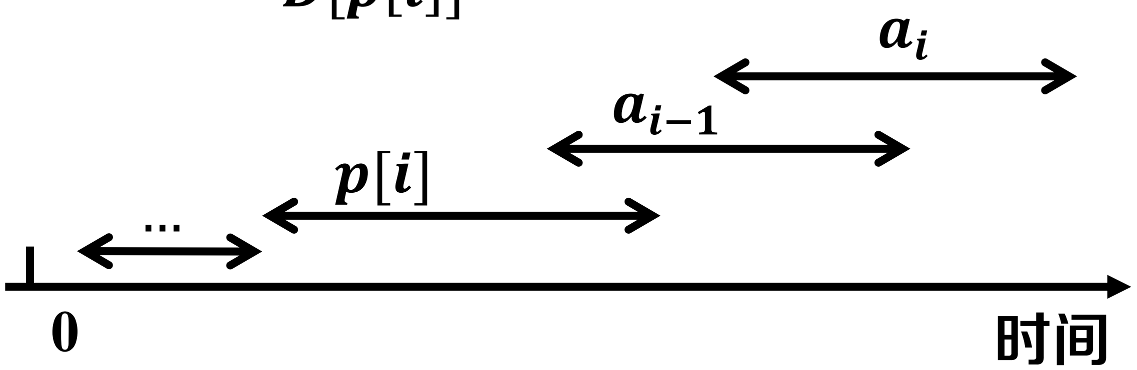


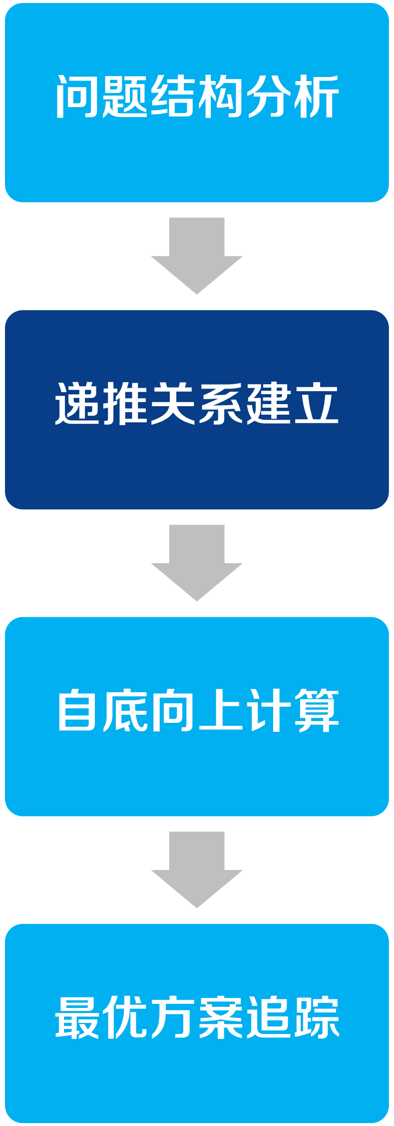
递推关系建立：分析最优（子）结构

- 考察活动a_i
  - 选择a_i



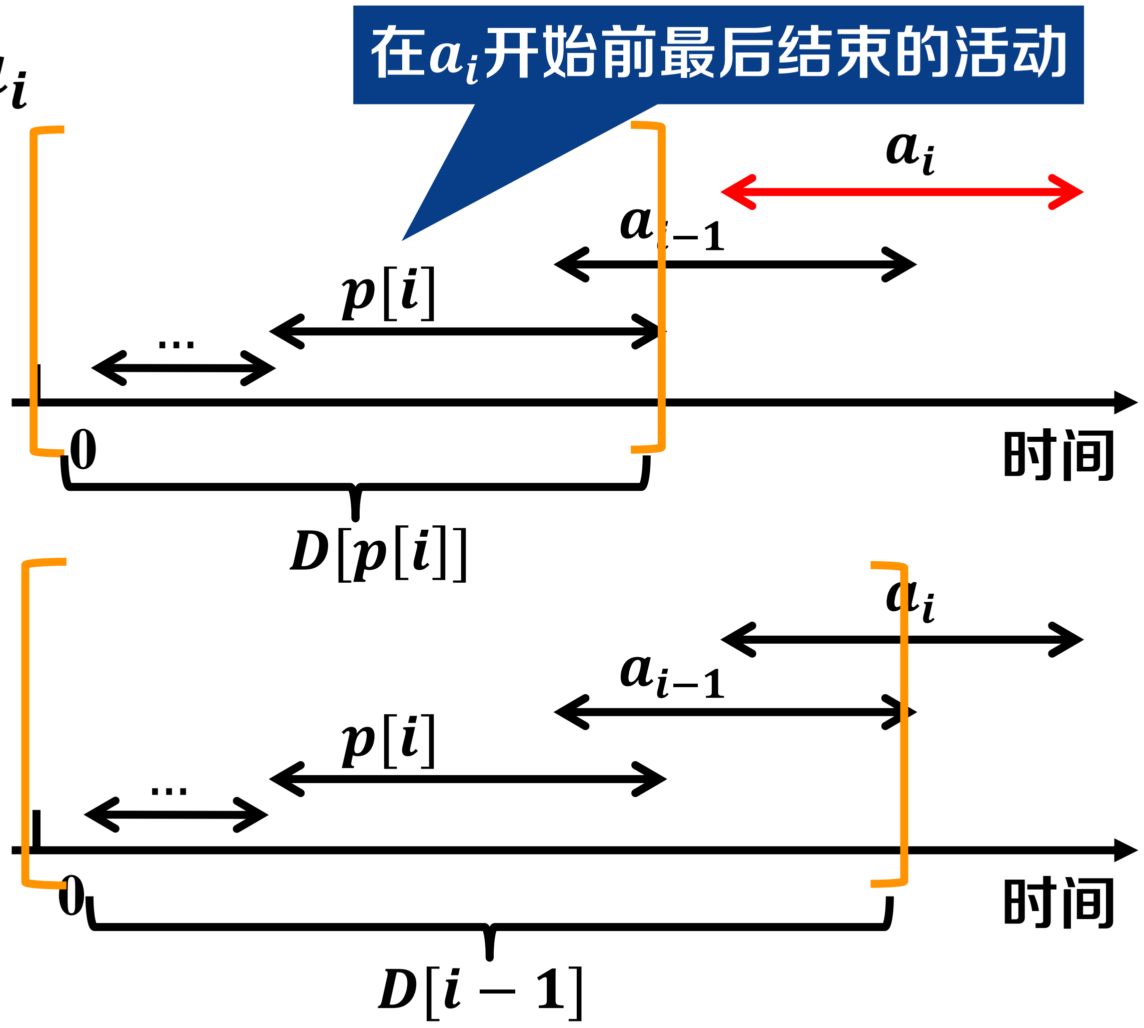
  - 不选a_i

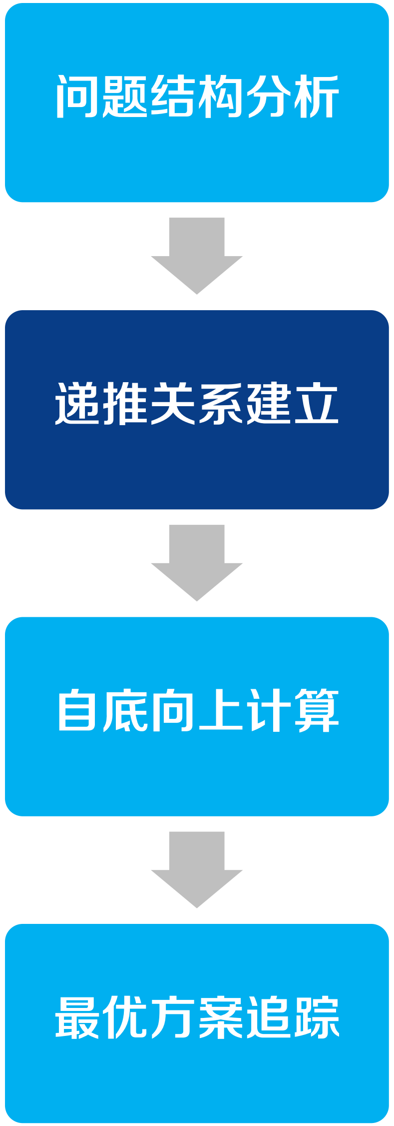




递推关系建立：分析最优（子）结构

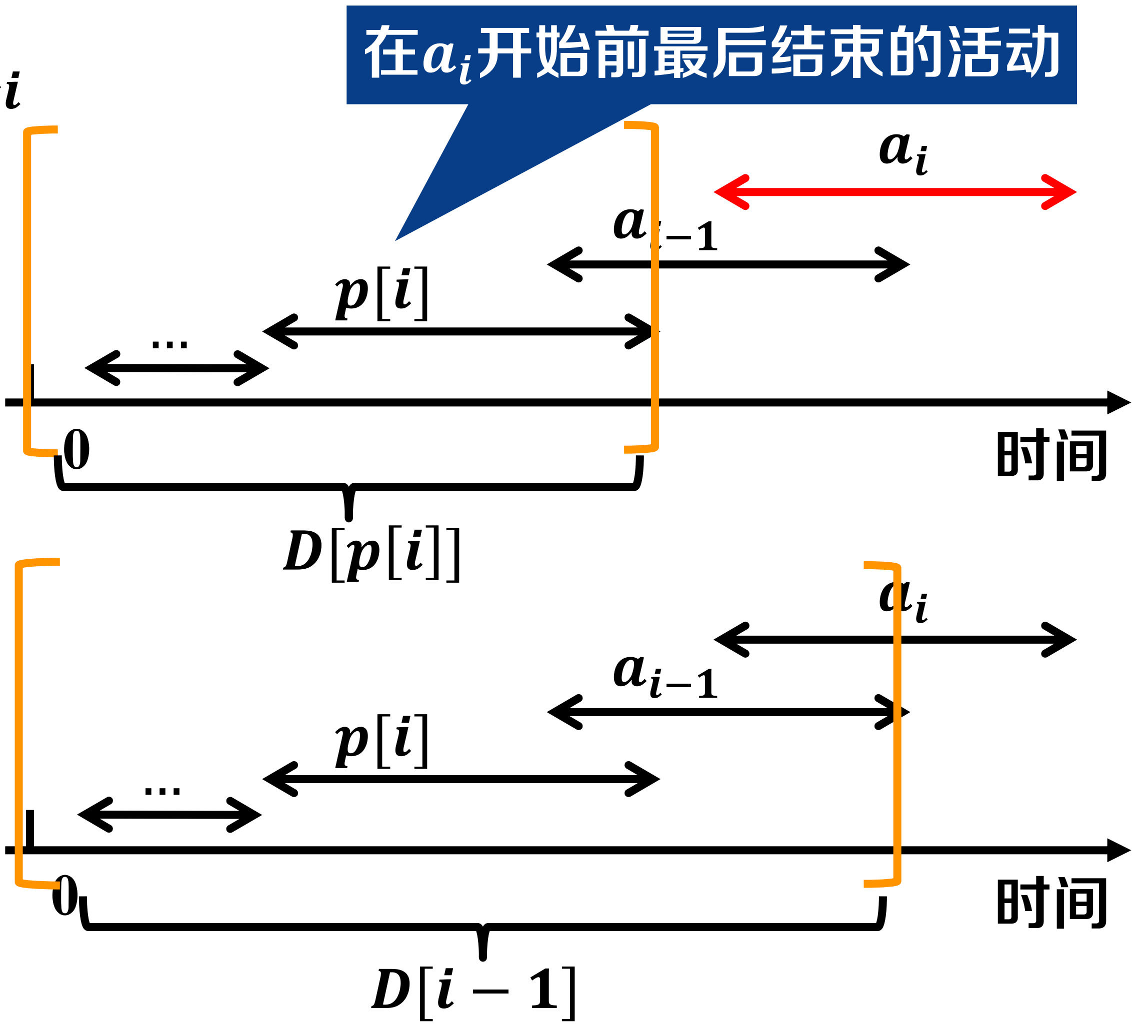
- 考察活动a_i
  - 选择a_i
  - 不选a_i



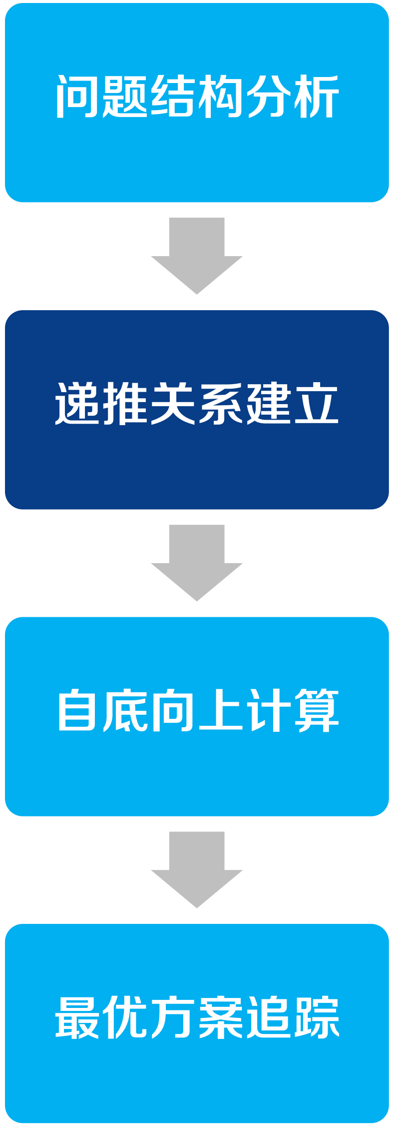


递推关系建立：构造递推公式

- 考察活动a_i
  - 选择a_i
  - 不选a_i

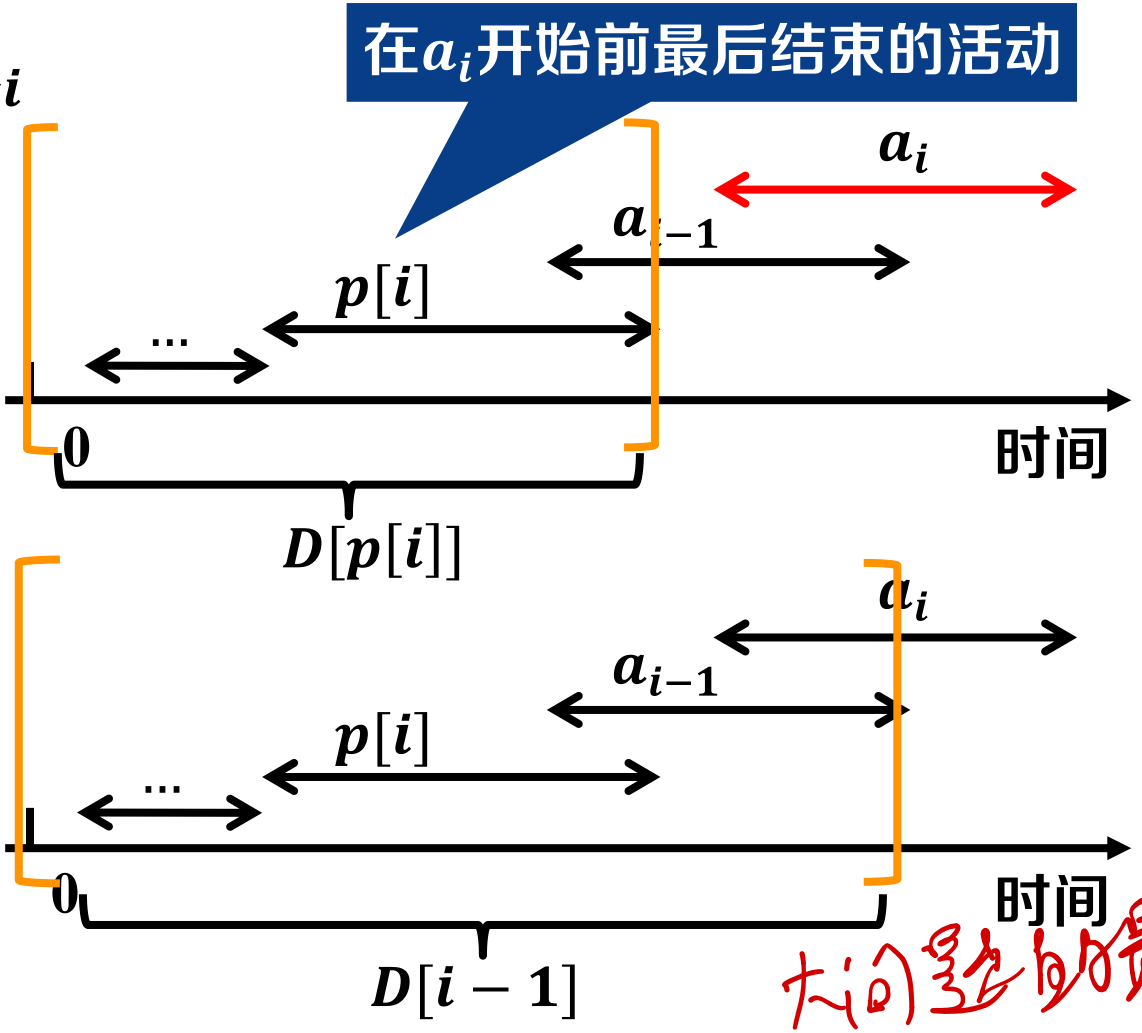


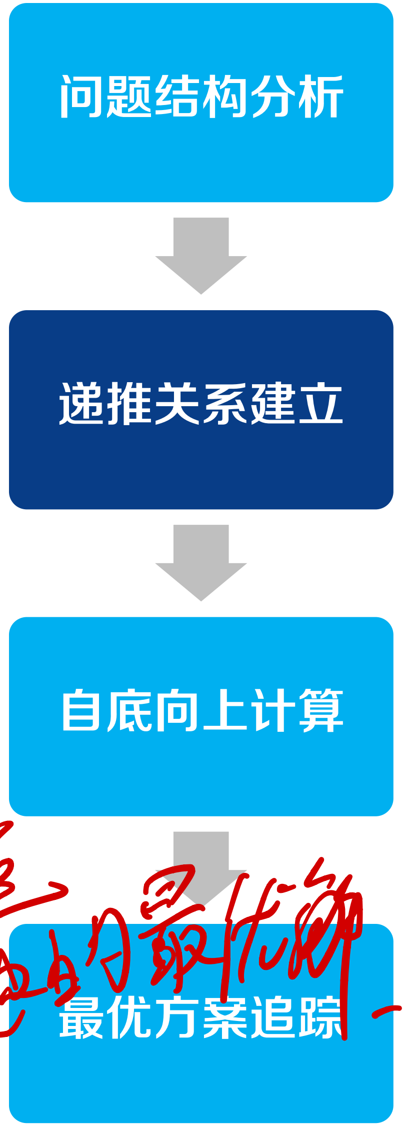
- 递推公式
  - $D[i] = \max\{D\big[p[i]\big] + w_i, D[i-1]\}$



递推关系建立：构造递推公式

- 考察活动a_i
  - 选择a_i
  - 不选a_i





- 递推公式
  - $D[i] = \max\{\boxed{D[p[i]]} + w_i, \boxed{D[i-1]}\}$

最优子结构

大问题的最优值
依赖于小问题的最优解

自底向上计算：确定计算顺序

- 初始化
  - $D[0]=0$：空活动集最大权重和为0

自底向上计算：确定计算顺序

- 初始化
  - $D[0] = 0$：空活动集最大权重和为0
- 递推公式
  - $D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$

已知

	1	2	…		$i-1$	i	…	n
w						w_i		

$p[i]$

	1	2	…		$i-1$	i	…	n
p						$p[i]$		

$D[i]$

	0	…	$p[i]$	…	$i-1$	i	…	n
D	0		$D[p[i]]$		$D[i-1]$			

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

自底向上计算：确定计算顺序

- 初始化
  - $D[0]=0$：空活动集最大权重和为0
- 递推公式
  - $D[i]=\max\{D\left[p[i]\right]+w_i, D[i-1]\}$

已知

	1	2	…		$i-1$	i	…	n
w						w_i		

$p[i]$

	1	2	…		$i-1$	i	…	n
p						$p[i]$		

$D[i]$

	0	…	$p[i]$	…	$i-1$	i	…	n
D	0		$D[p[i]]$		$D[i-1]$	$D[i]$		

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

自底向上计算：依次求解问题

- 初始化
  - $D[0]=0$：空活动集最大权重和为0
- 递推公式
  - $D[i]=\max\{D[p[i]]+w_i, D[i-1]\}$

已知

	1	2	…		$i-1$	i	…	n
w						w_i		

$p[i]$

	1	2	…		$i-1$	i	…	n
p						$p[i]$		

自底向上计算

$D[i]$

	0	…	$p[i]$	…	$i-1$	i	…	n
D	0							★

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

最优方案追踪

- **记录决策过程**
  - $$Rec[i] = \begin{cases} 1, & \text{选择活动} a_i \\ 0, & \text{不选活动} a_i \end{cases}$$

当前层决策活动i
是否选择.

最优方案追踪

- 记录决策过程
  - $Rec[i] = \begin{cases} 1, & \text{选择活动}a_i \\ 0, & \text{不选活动}a_i \end{cases}$
- 输出最优方案
  - $Rec[i] = 1$时，选择活动a_i，考察子问题$D[p[i]]$
  - $Rec[i] = 0$时，不选活动a_i，考察子问题$D[i-1]$

已求

	1	2	…	$i-1$	i	…	$n-1$	n
p					$p[i]$		i	

	1	…	$p[i]$	…	i	…	$n-1$	n
Rec	1	0	0	0	1	0	1	0

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

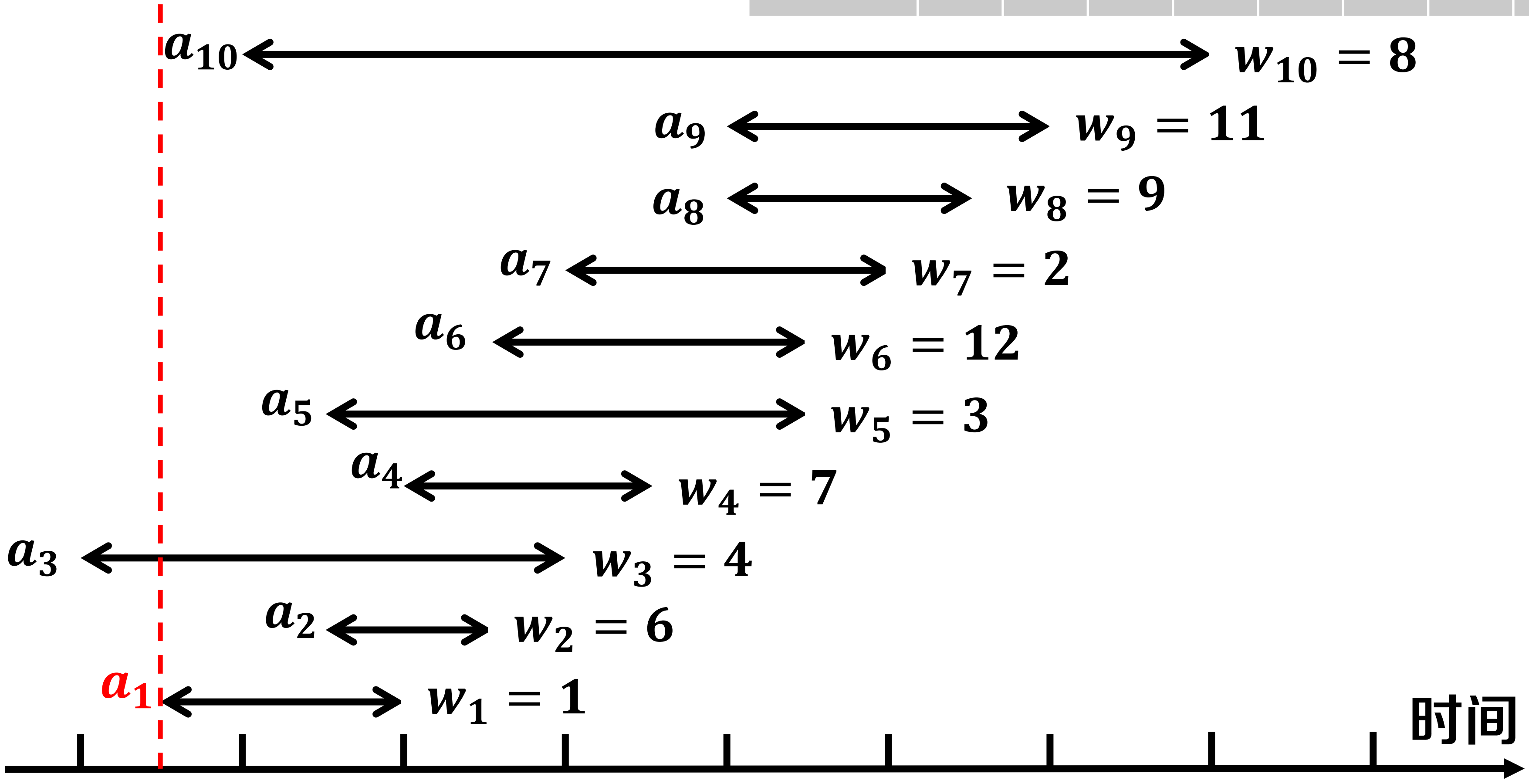
算法实例

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

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

$p[i]$：在a_i开始前最后结束的活动

	1	2	3	4	5	6	7	8	9	10
Rec										



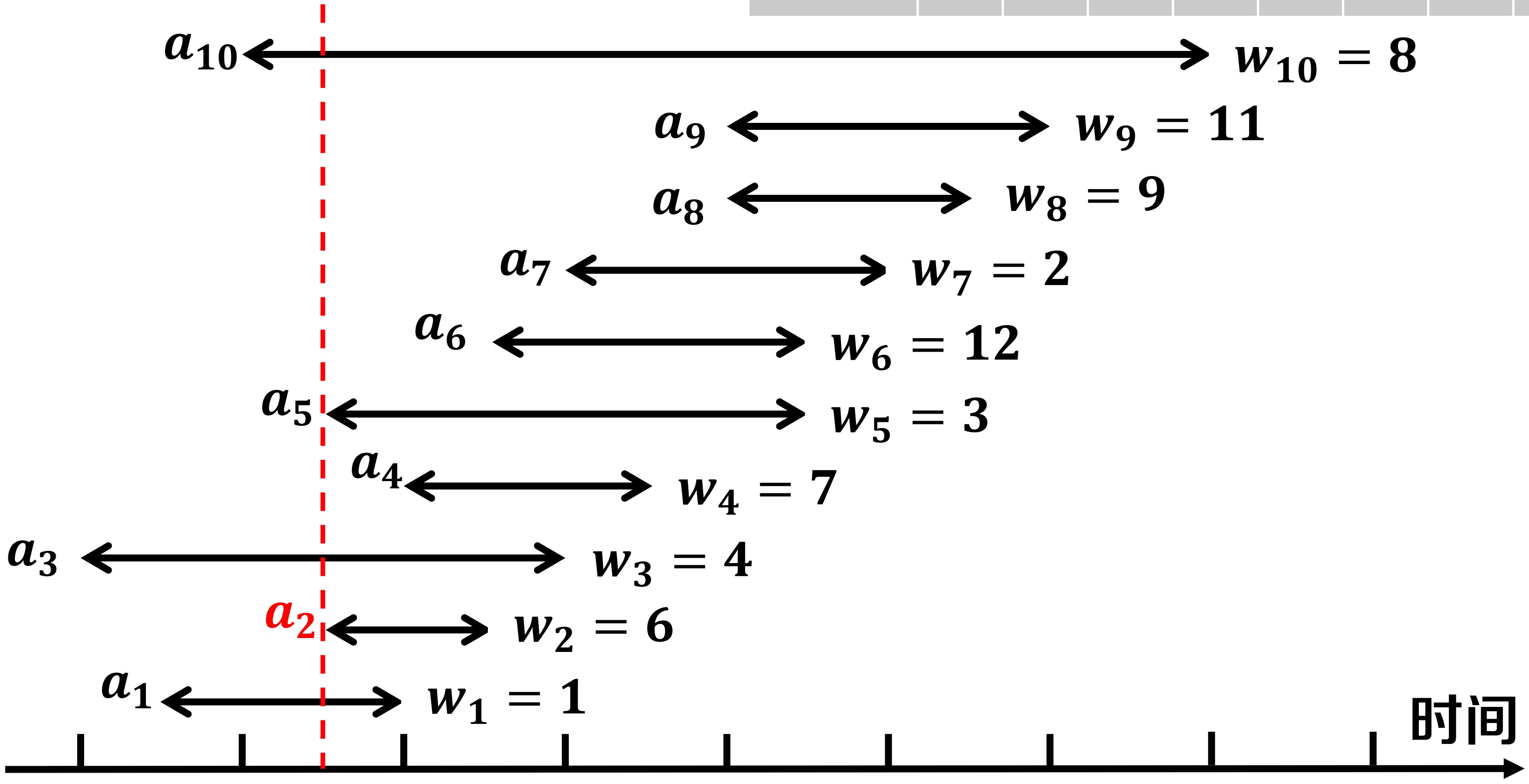
算法实例

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

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动



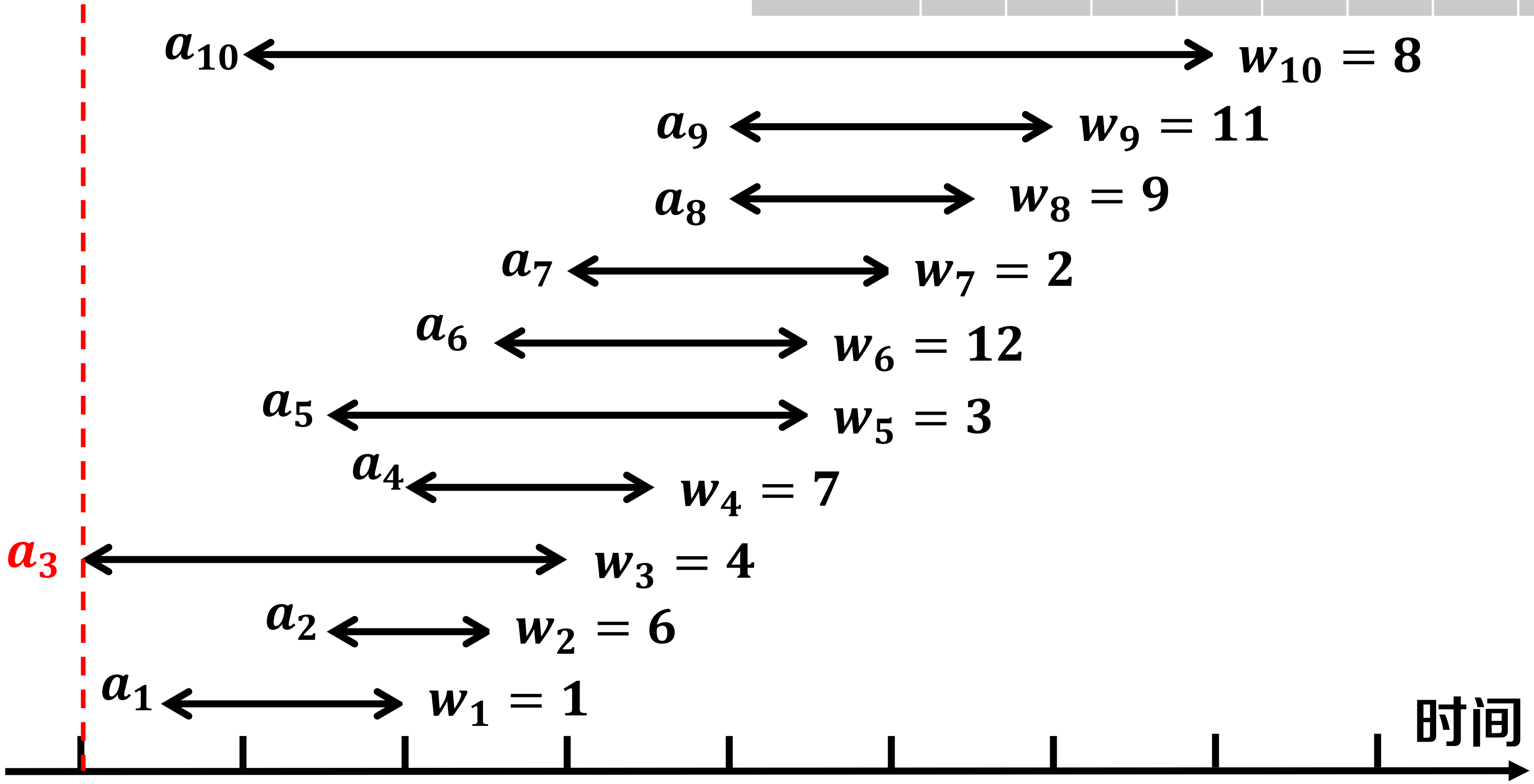
算法实例

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

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动



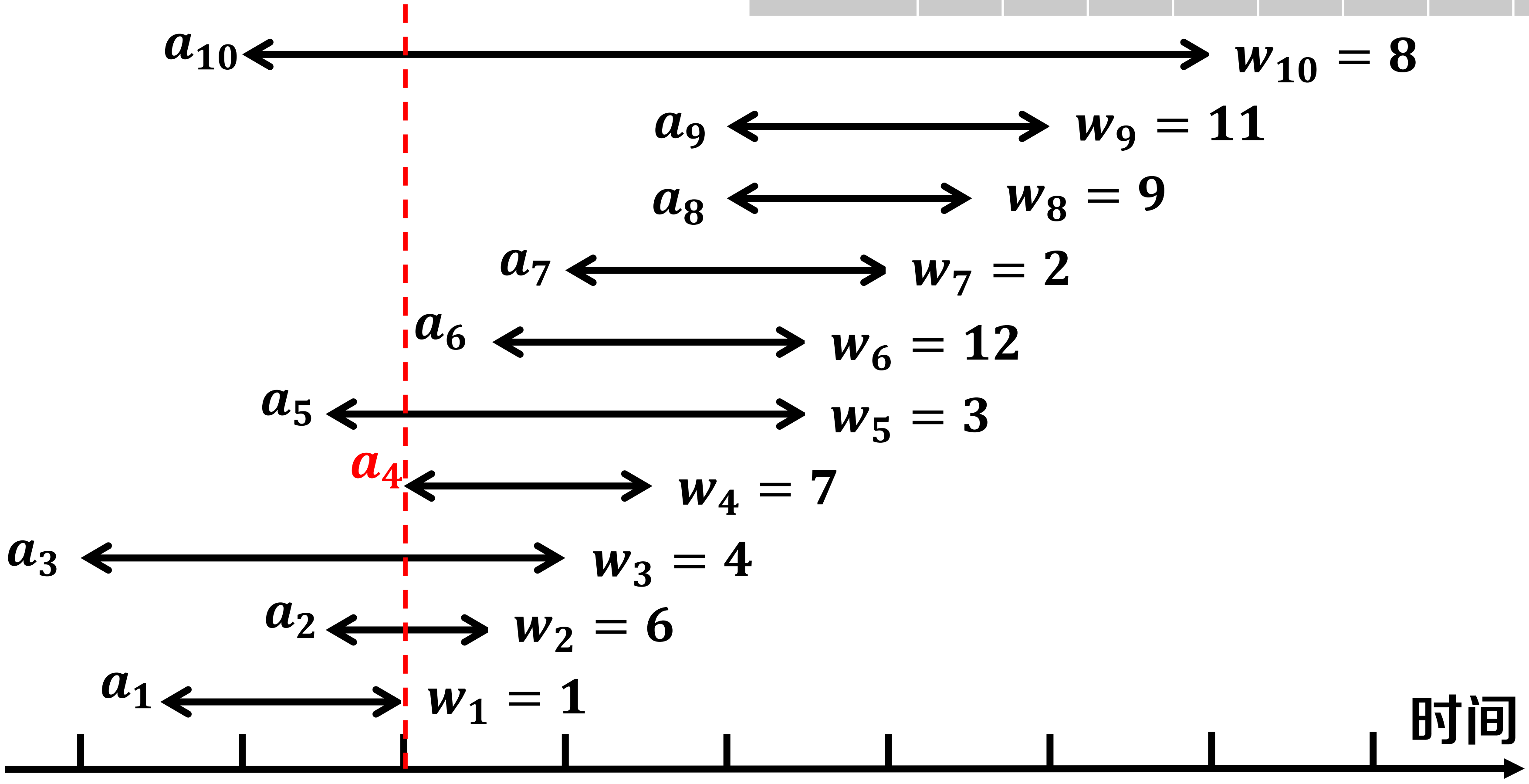
算法实例

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

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动



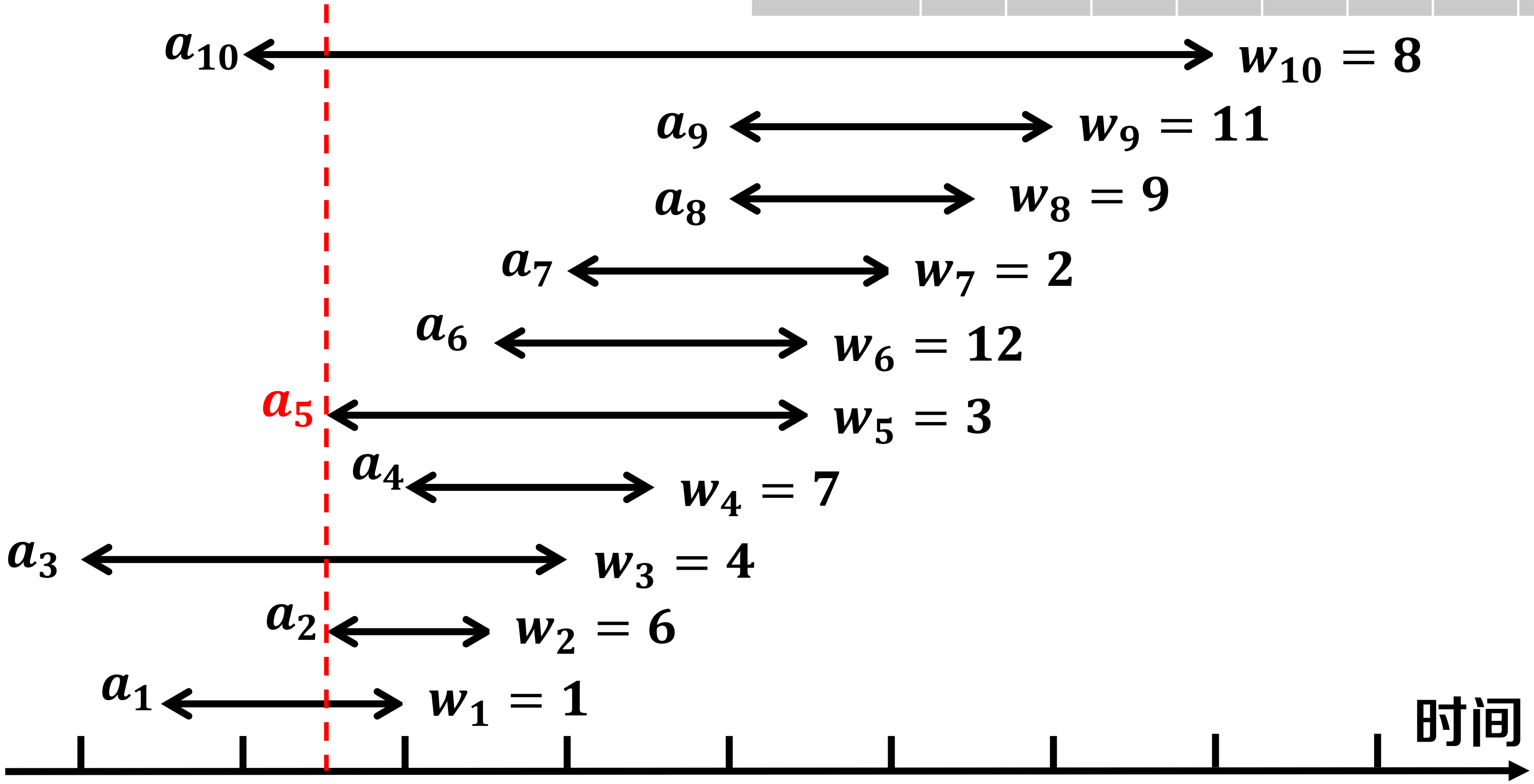
算法实例

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

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动



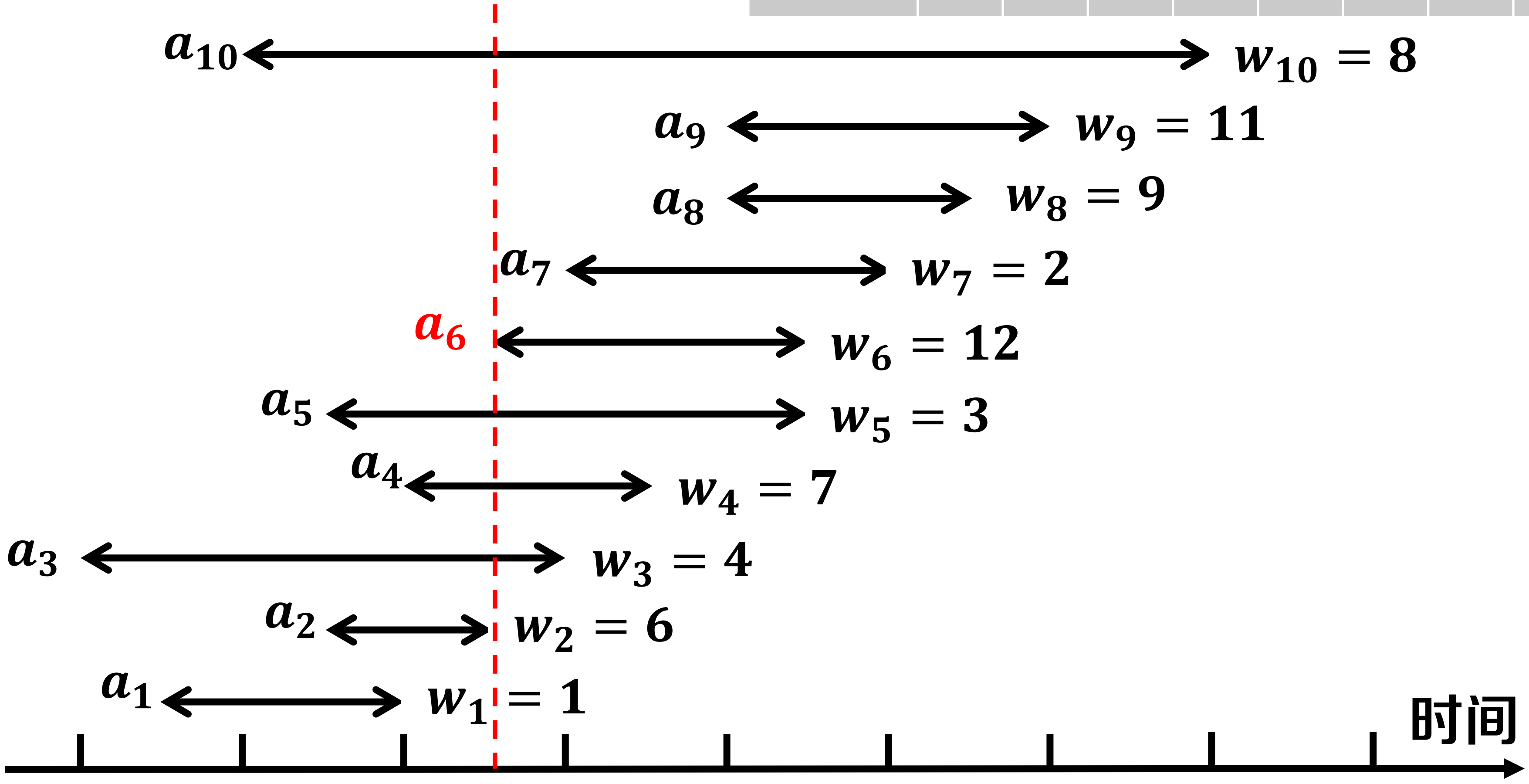
算法实例

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

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动



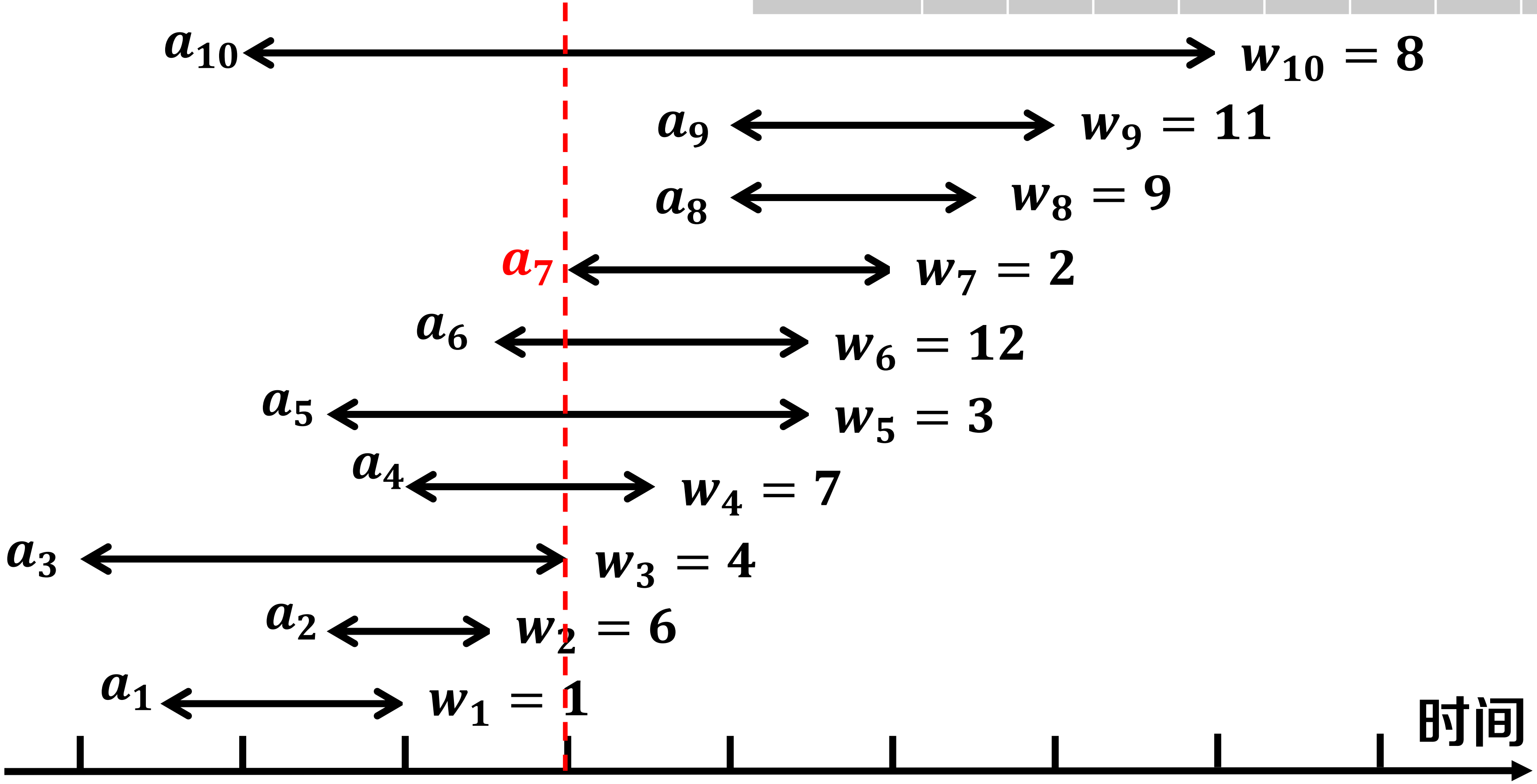
算法实例

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

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动



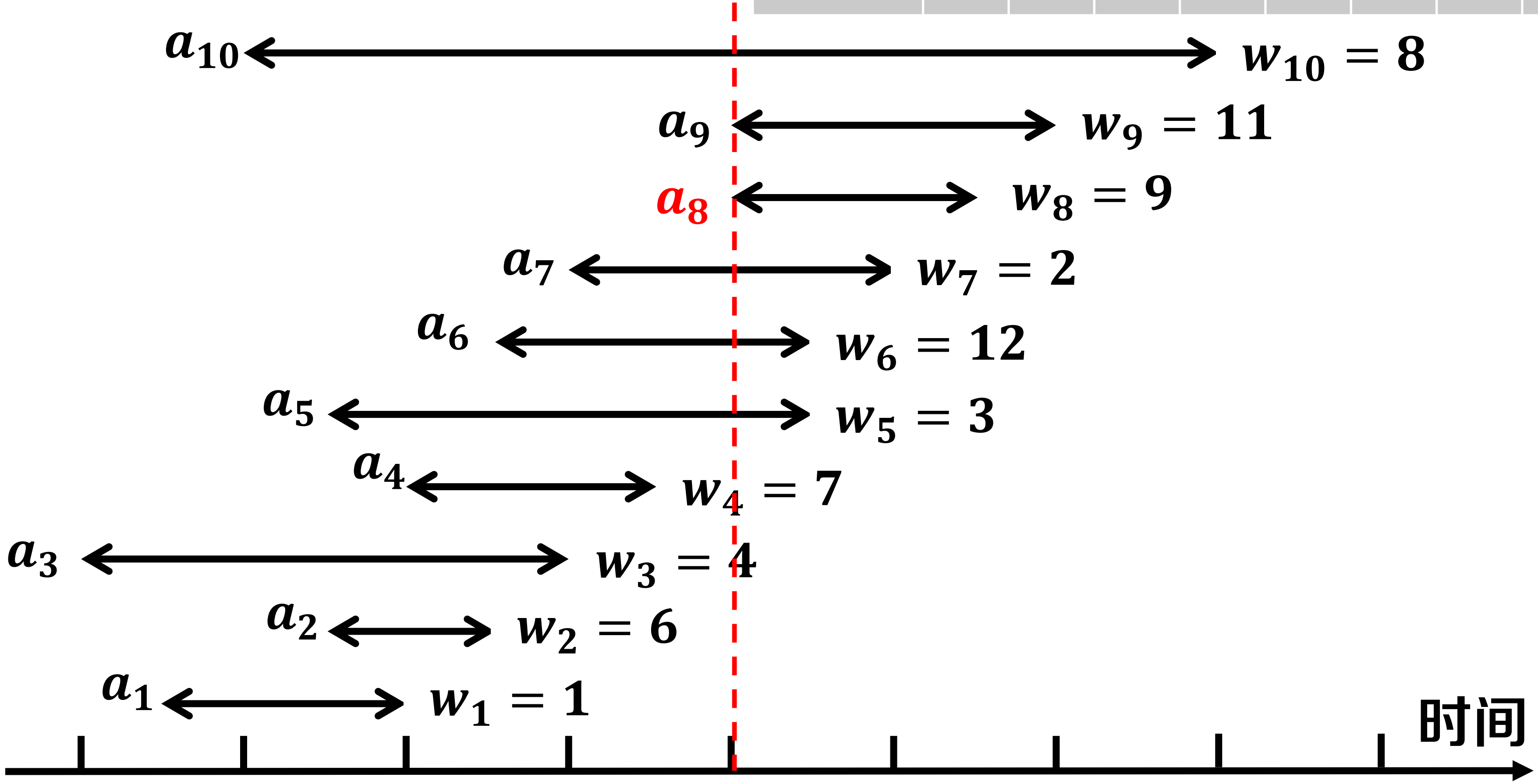
算法实例

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

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

$p[i]$：在a_i开始前最后结束的活动

	1	2	3	4	5	6	7	8	9	10
Rec										



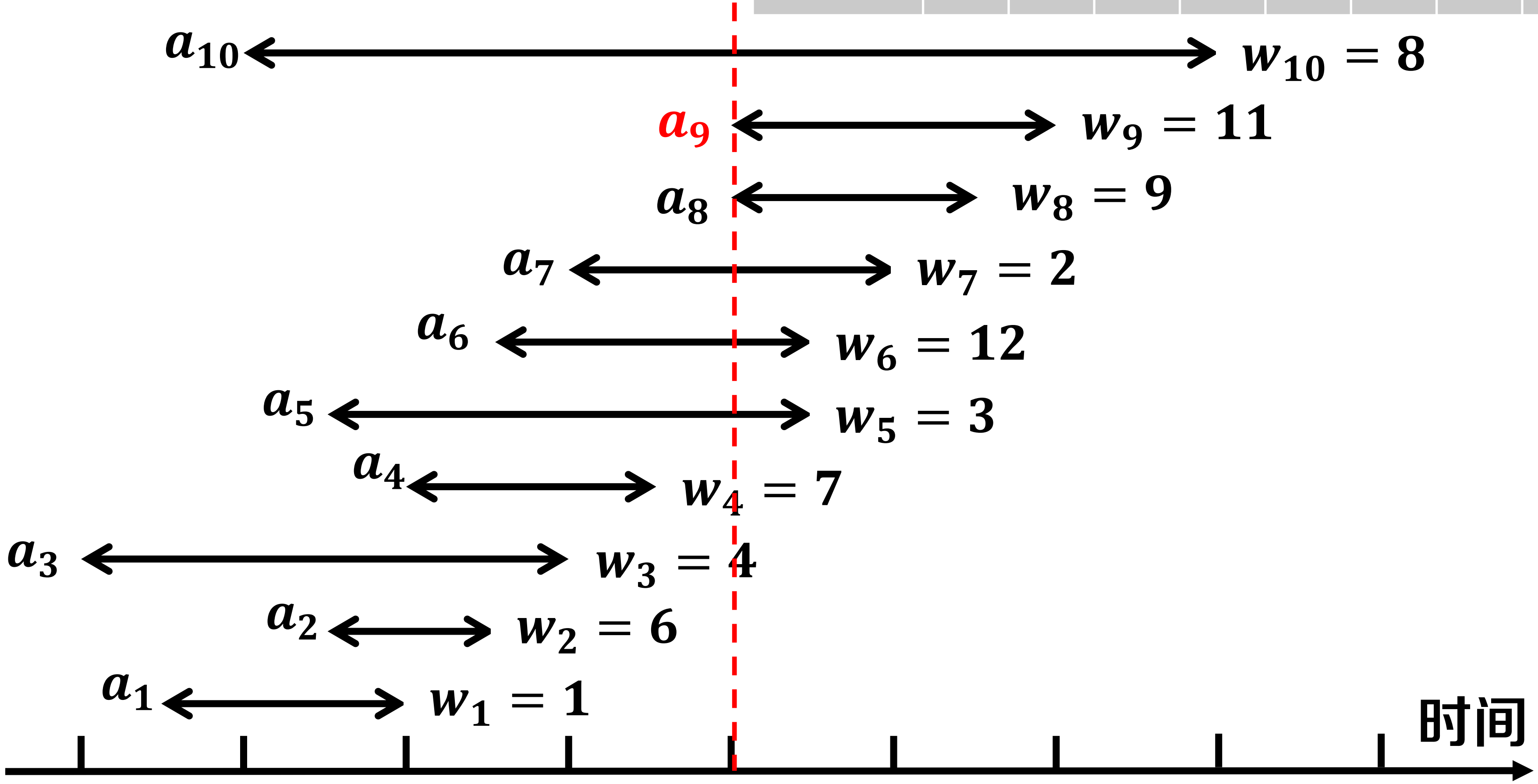
算法实例

	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	$\boxed{4}$	

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

$p[i]$：在a_i开始前最后结束的活动

	1	2	3	4	5	6	7	8	9	10
Rec										



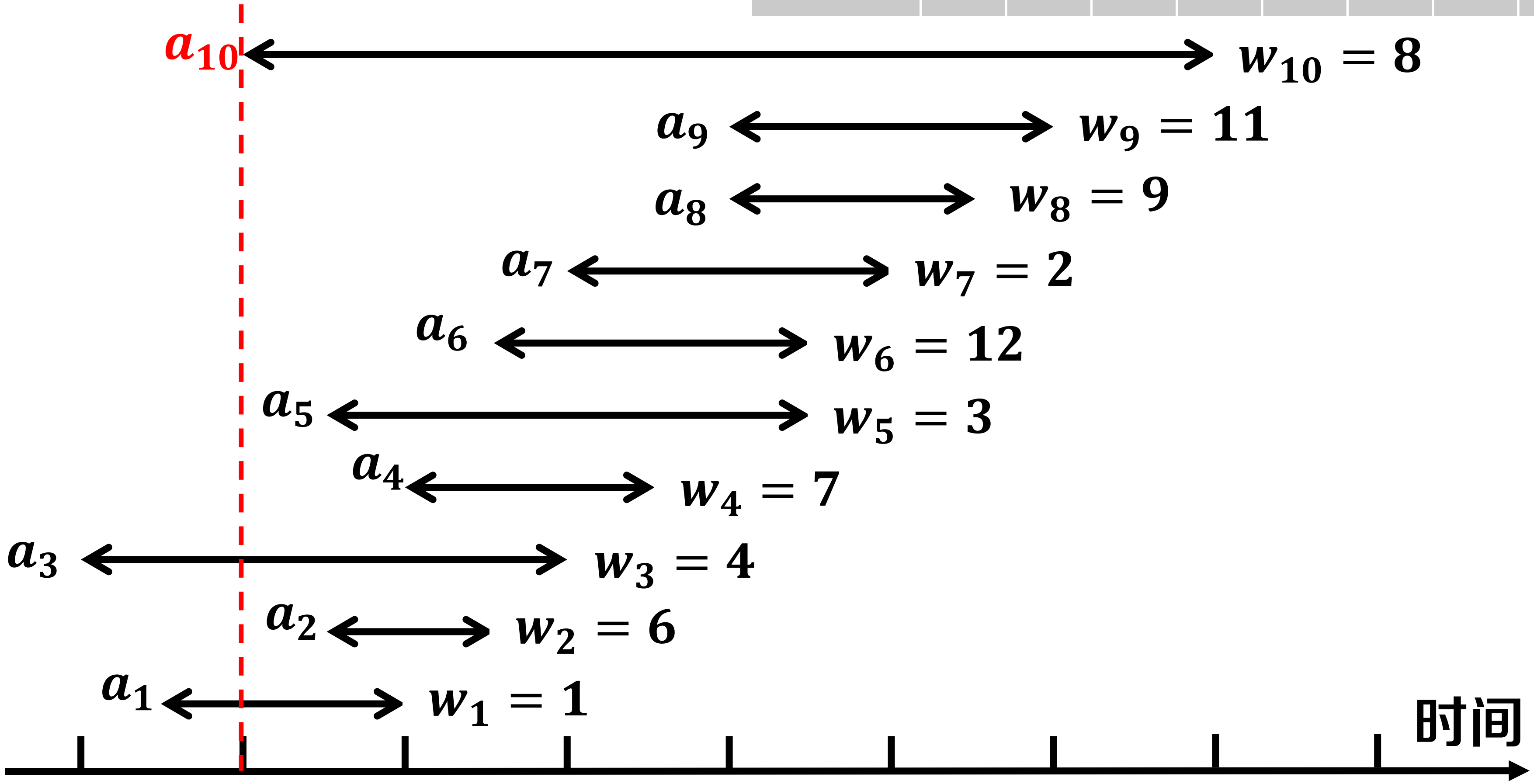
算法实例

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

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动



算法实例

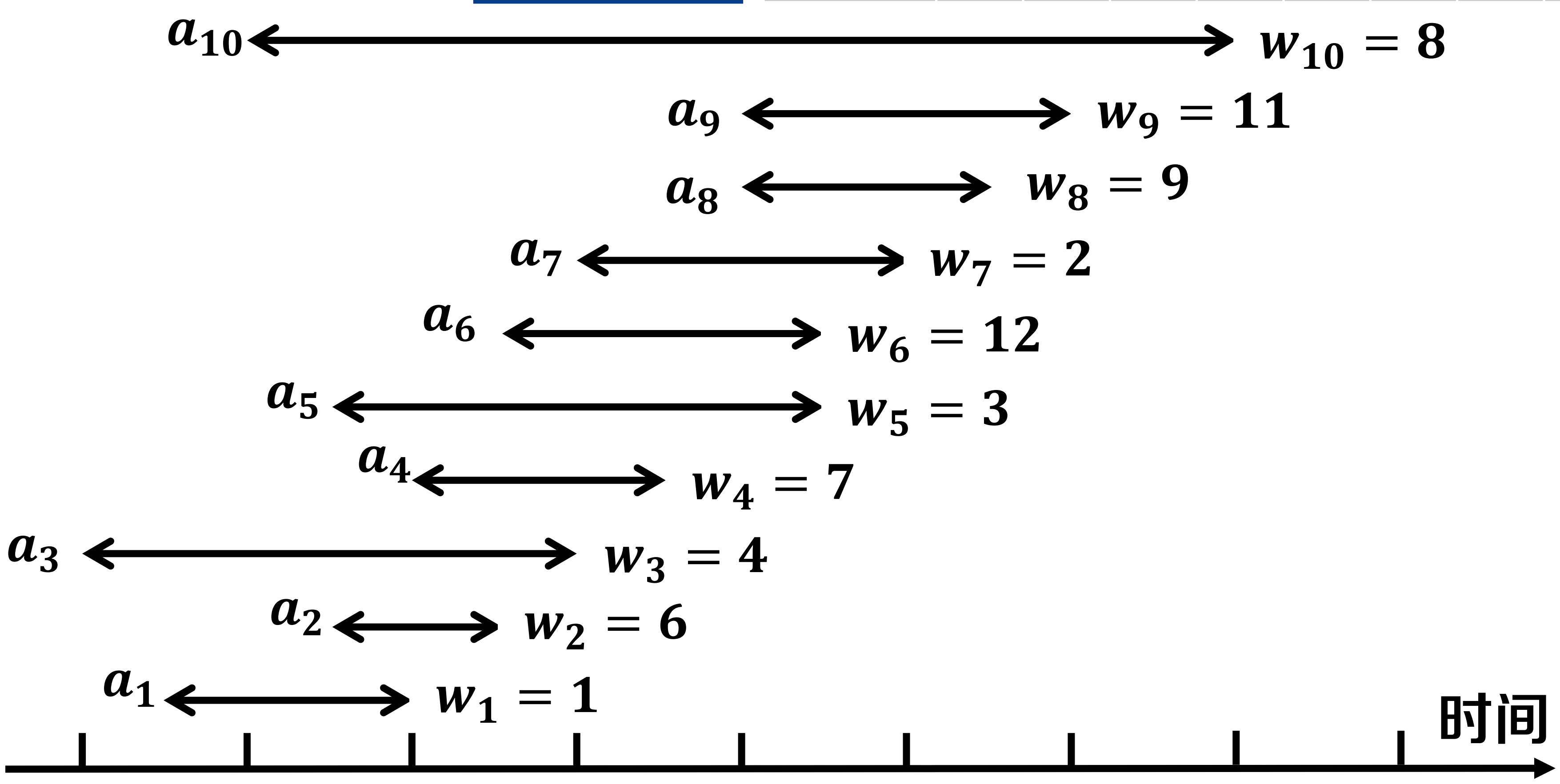
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

初始化

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动



算法实例

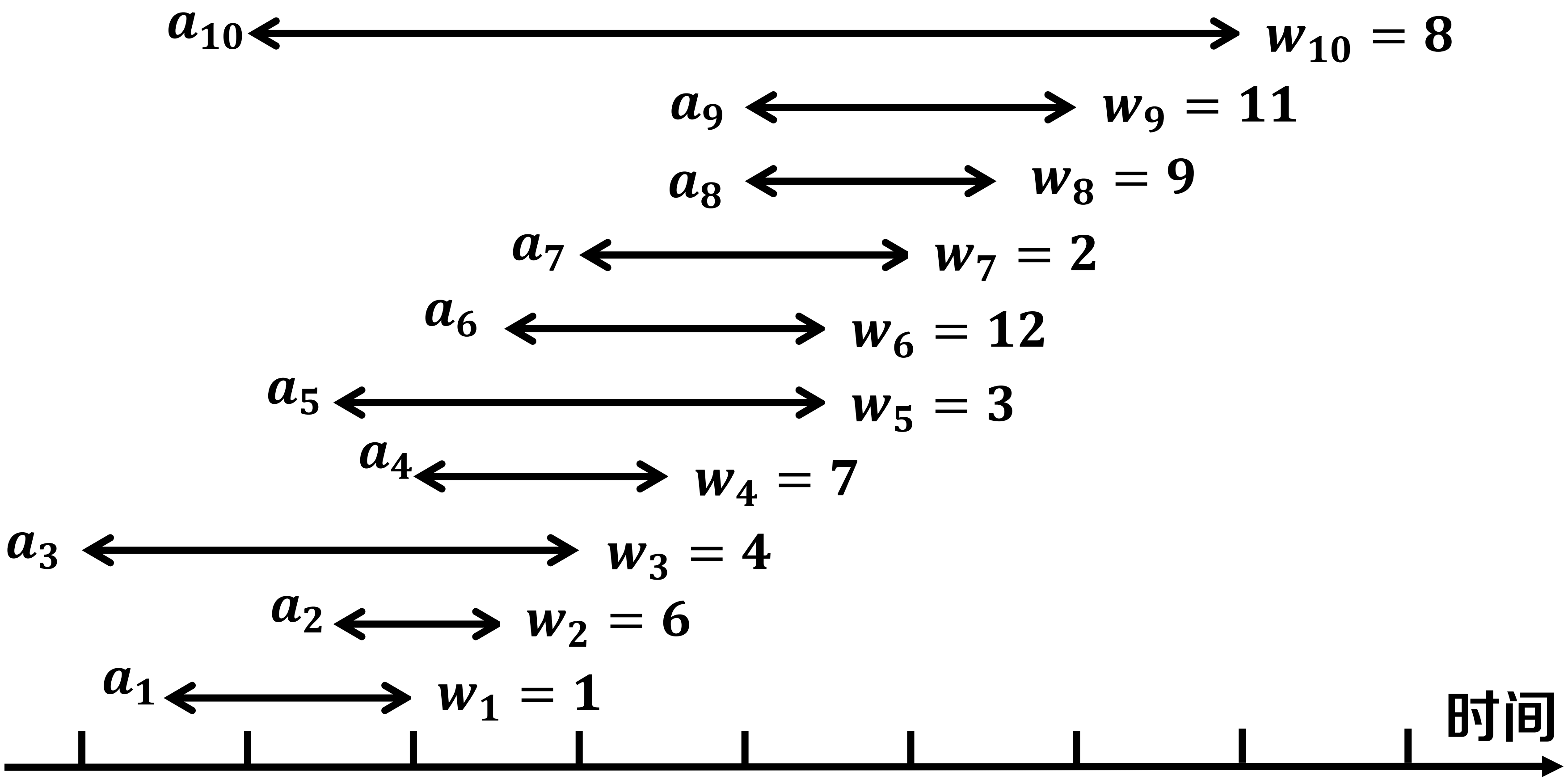
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

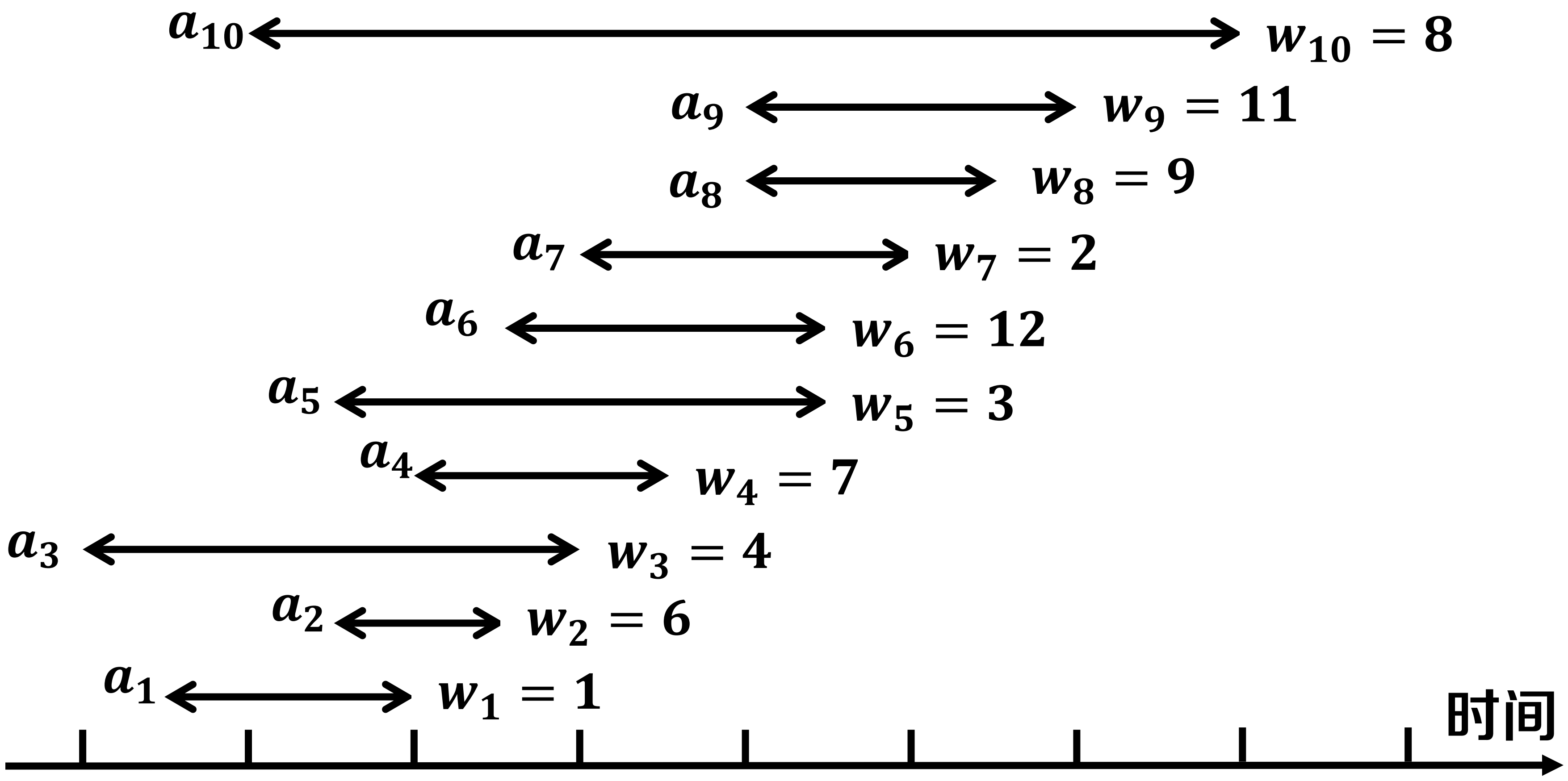
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

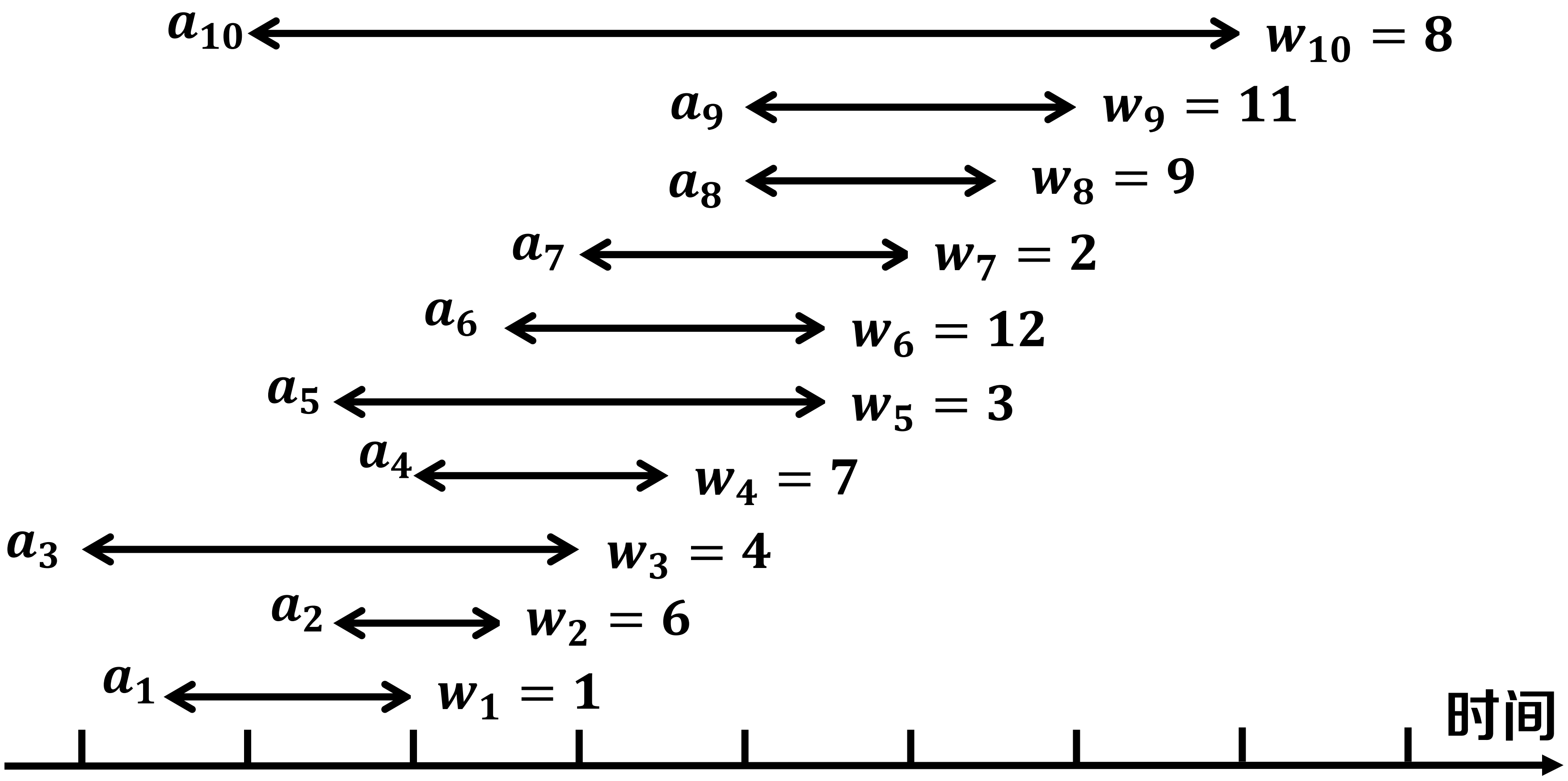
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

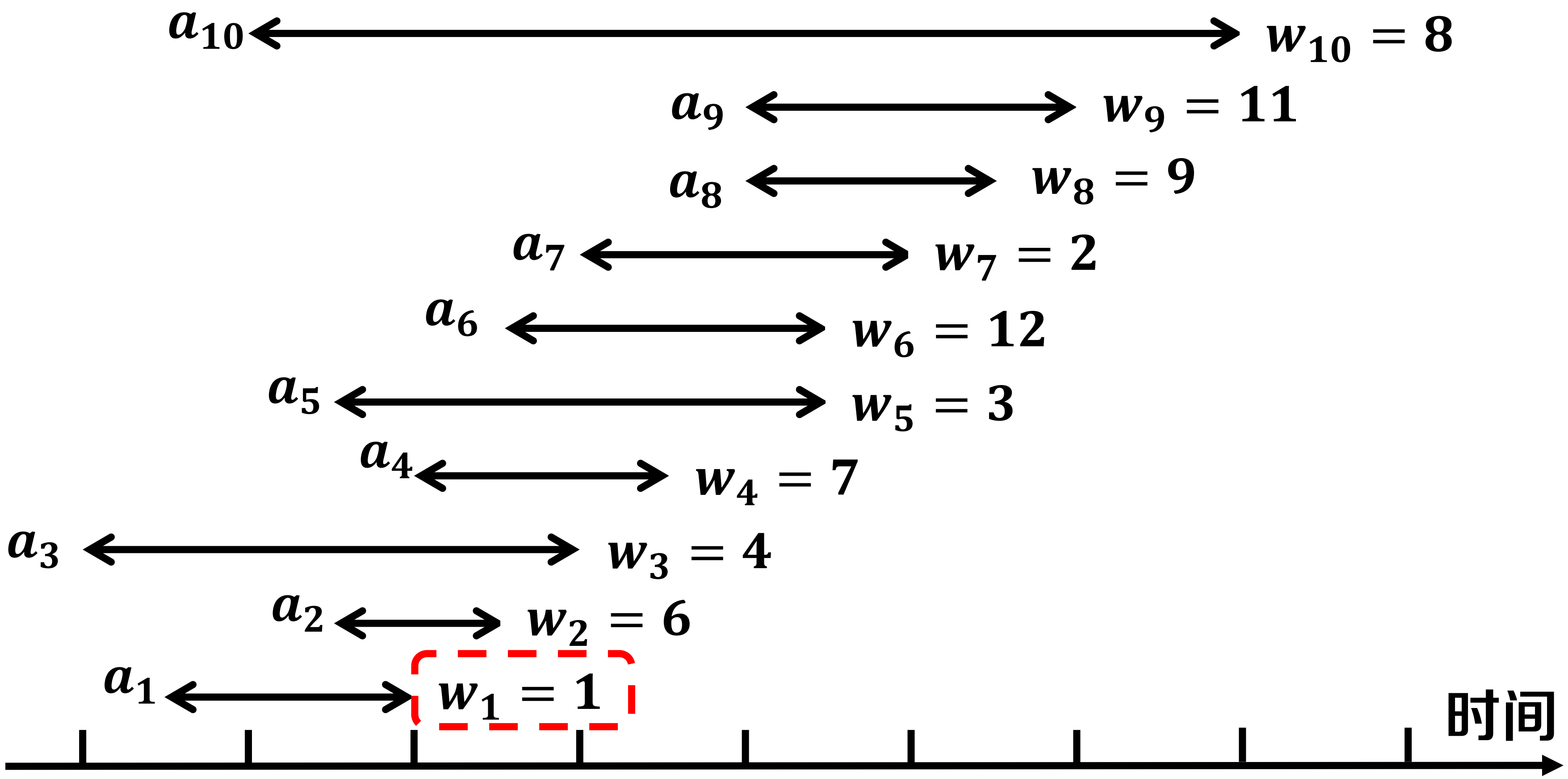
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

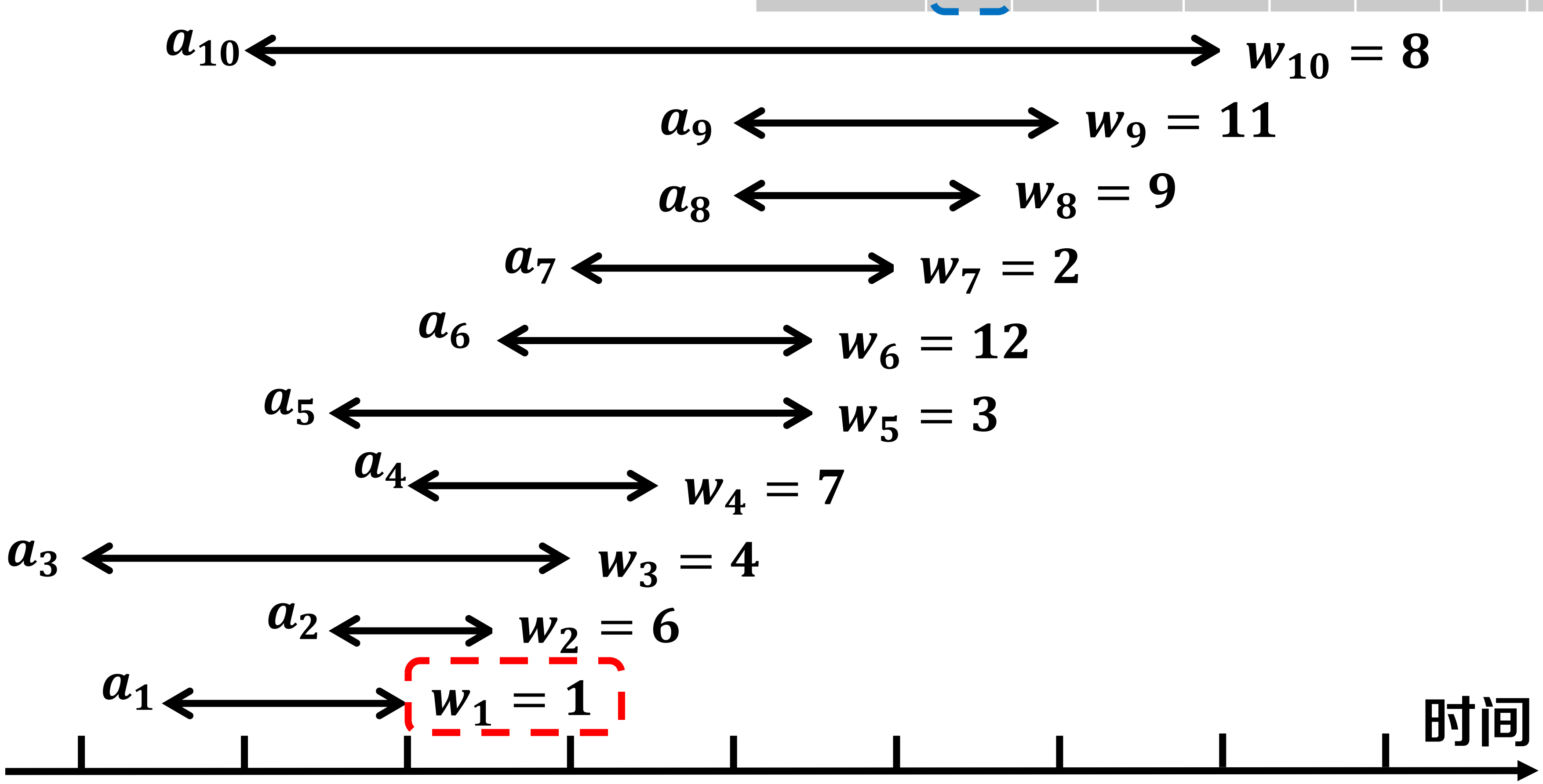
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

	1	2	3	4	5	6	7	8	9	10
Rec										

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

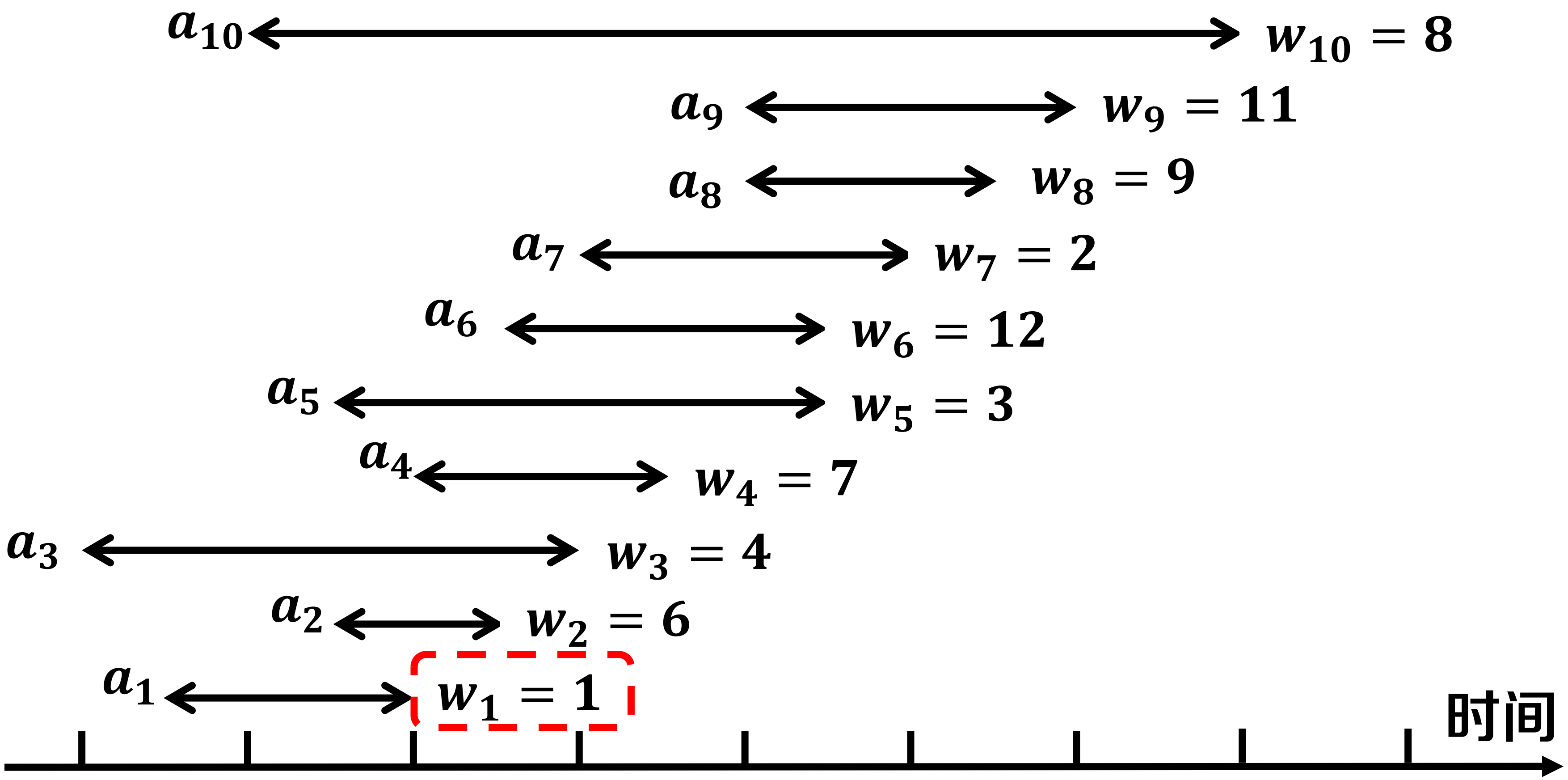
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

	1	2	3	4	5	6	7	8	9	10
Rec	1									

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

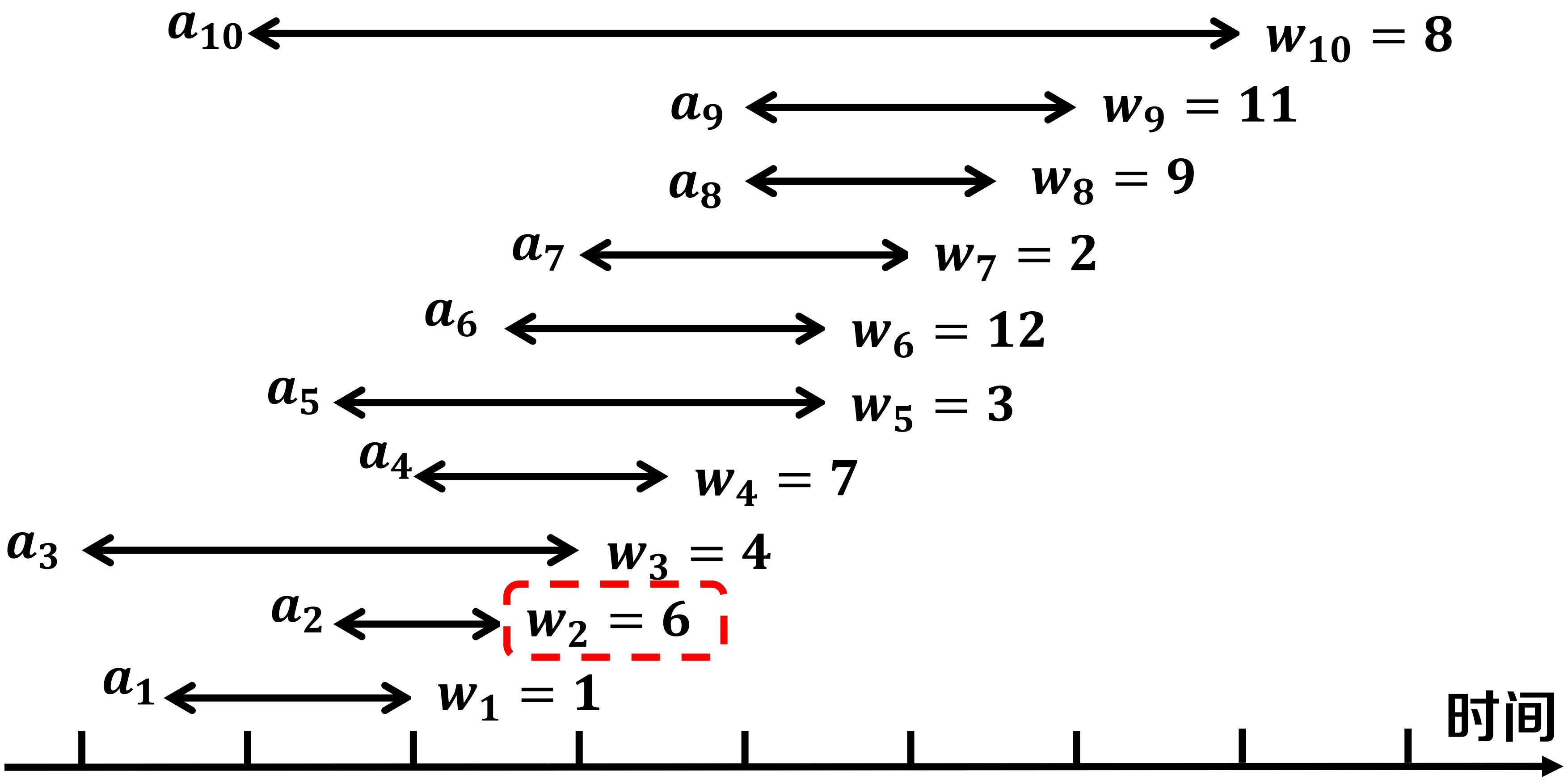
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

	1	2	3	4	5	6	7	8	9	10
Rec	1	1								

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

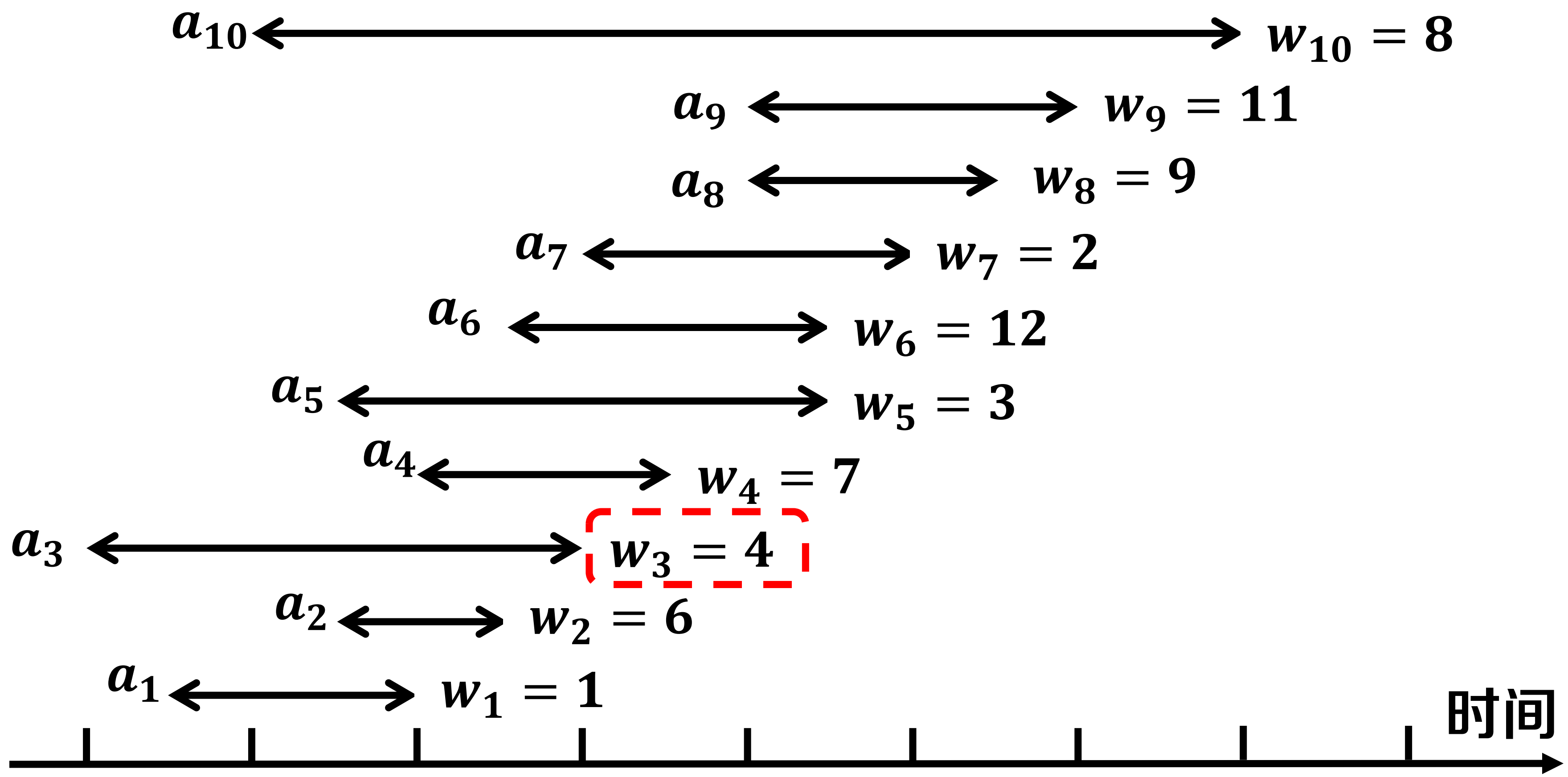
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

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

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

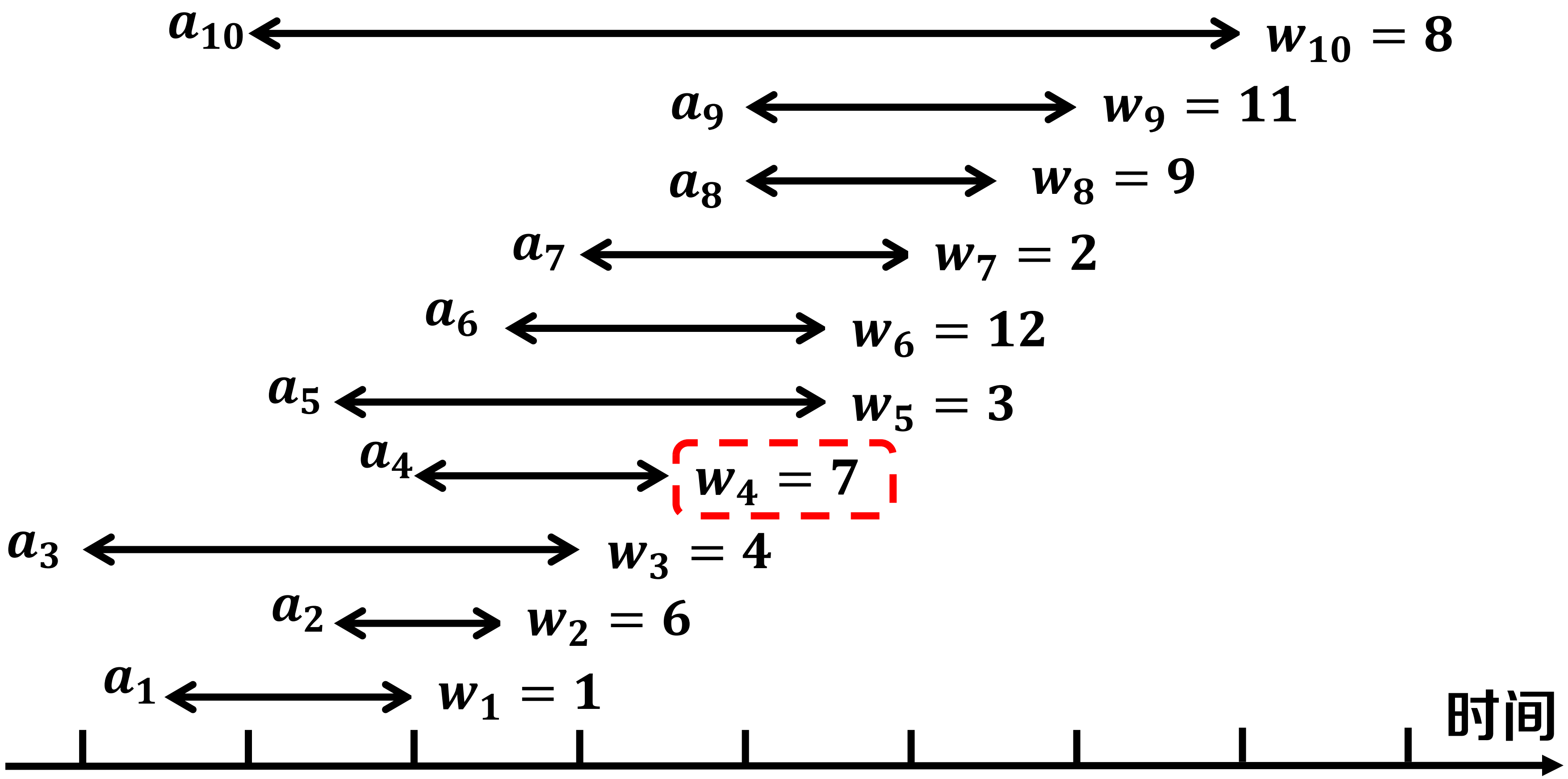
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

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

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

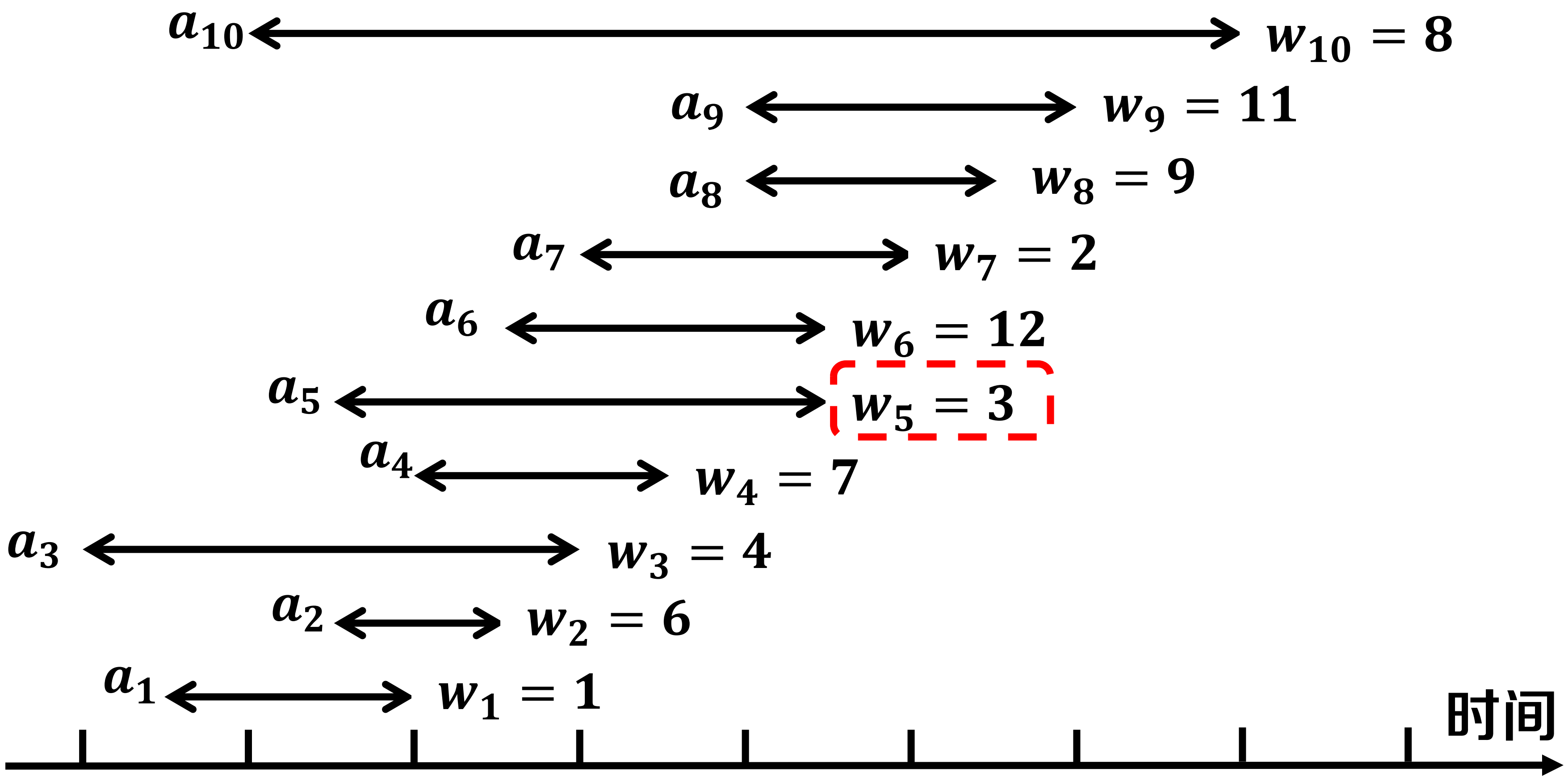
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

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

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

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

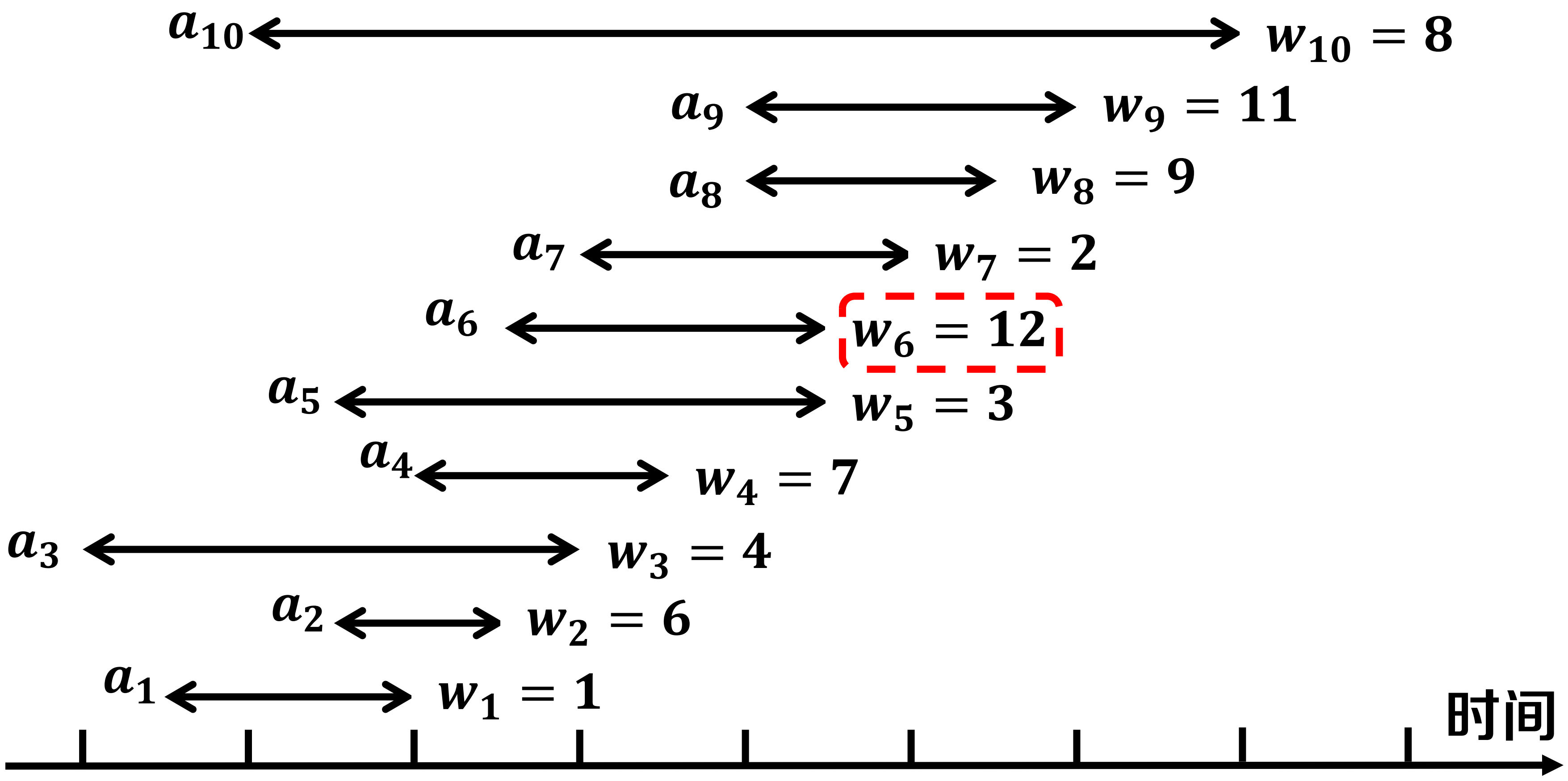
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18				

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

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

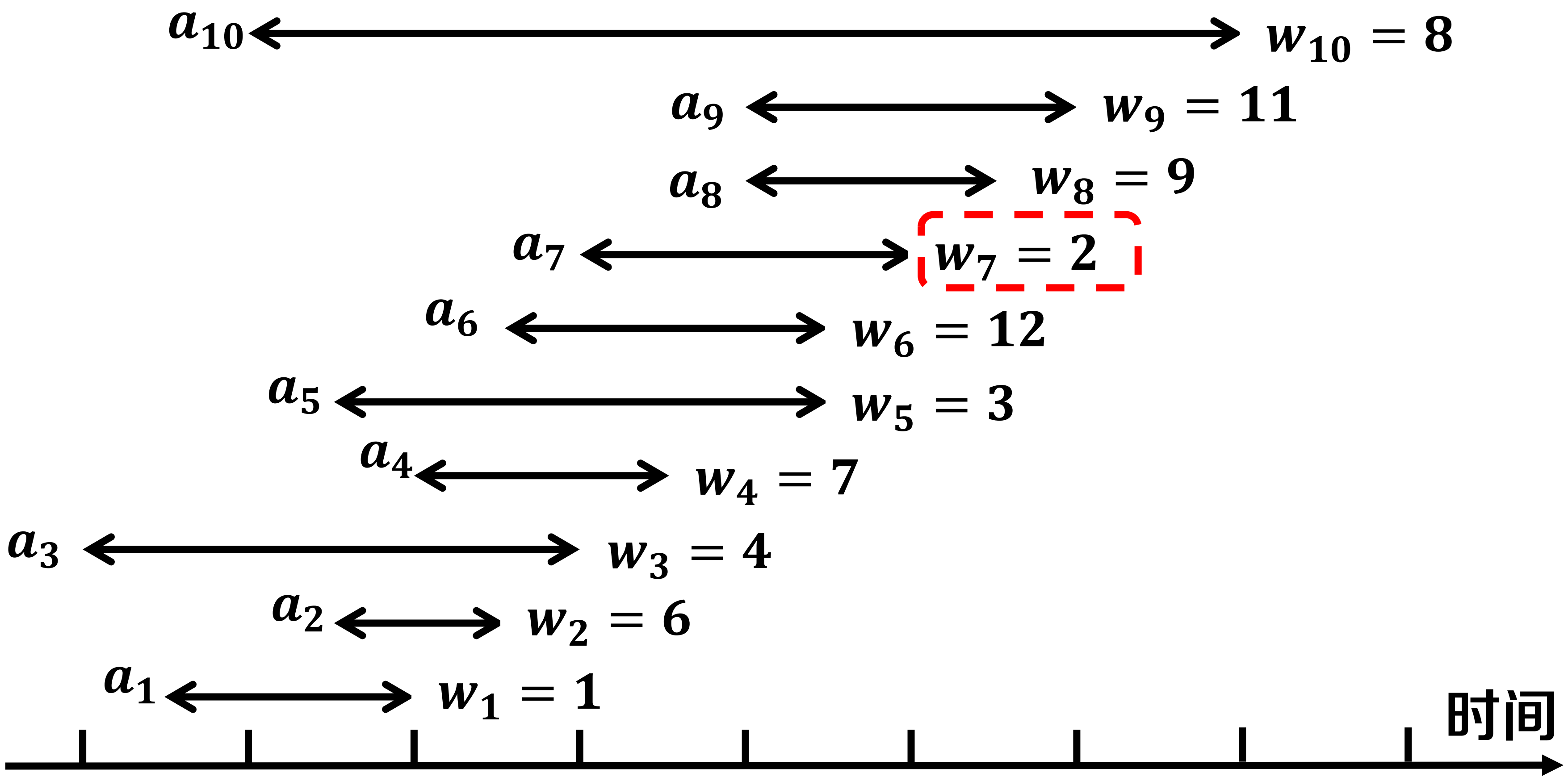
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18			

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$

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



算法实例

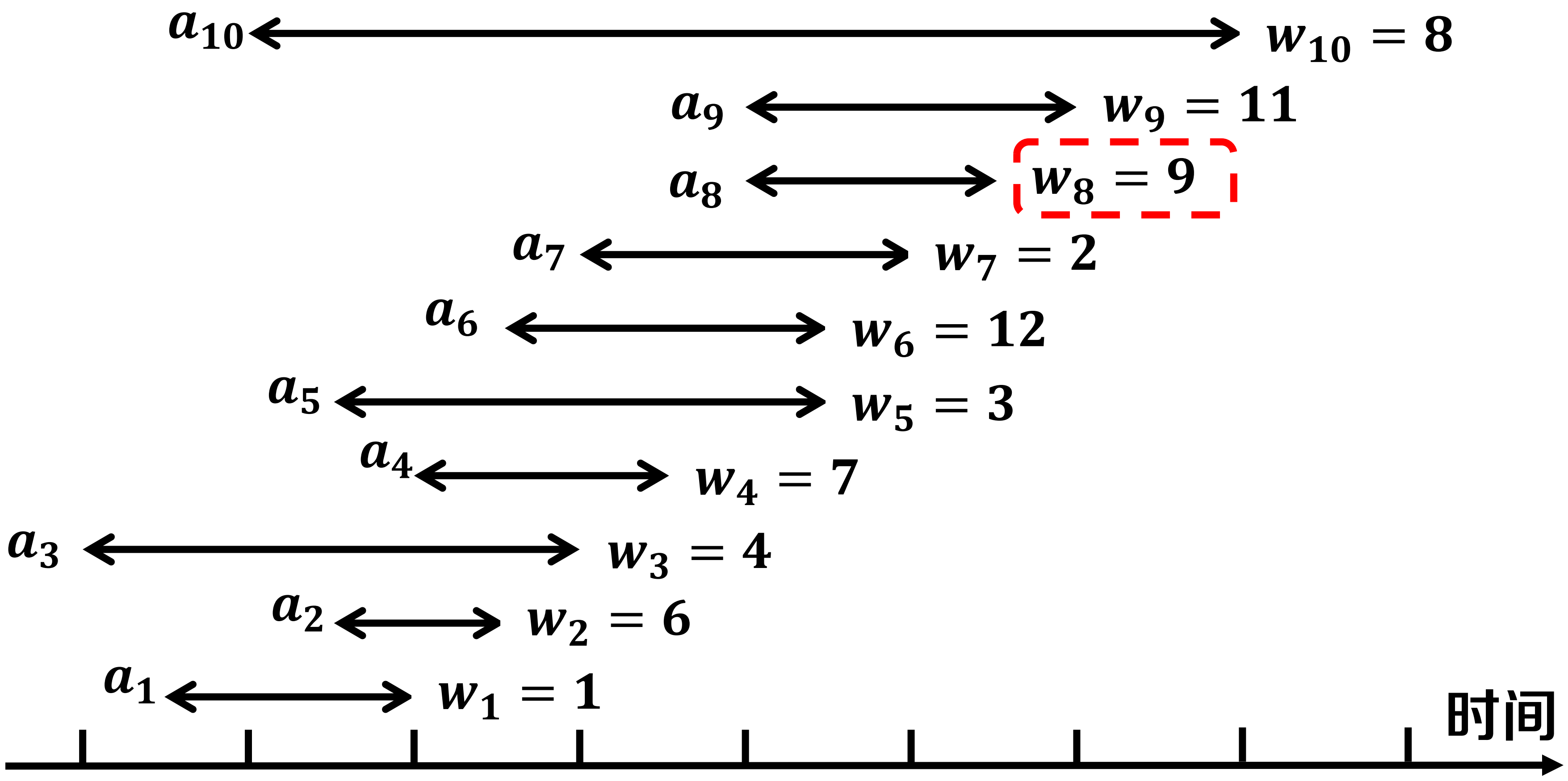
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18		

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

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$



算法实例

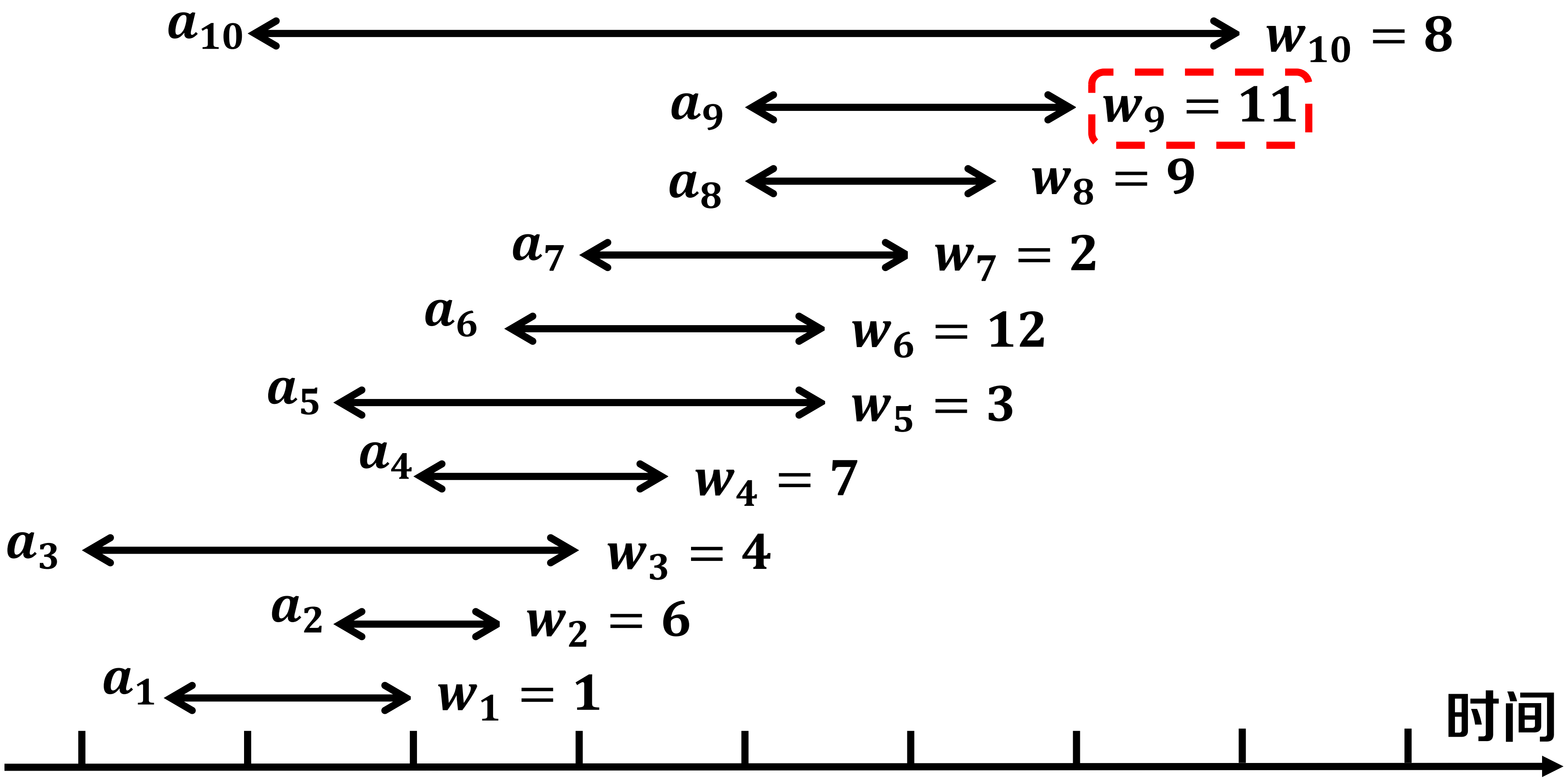
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$

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



算法实例

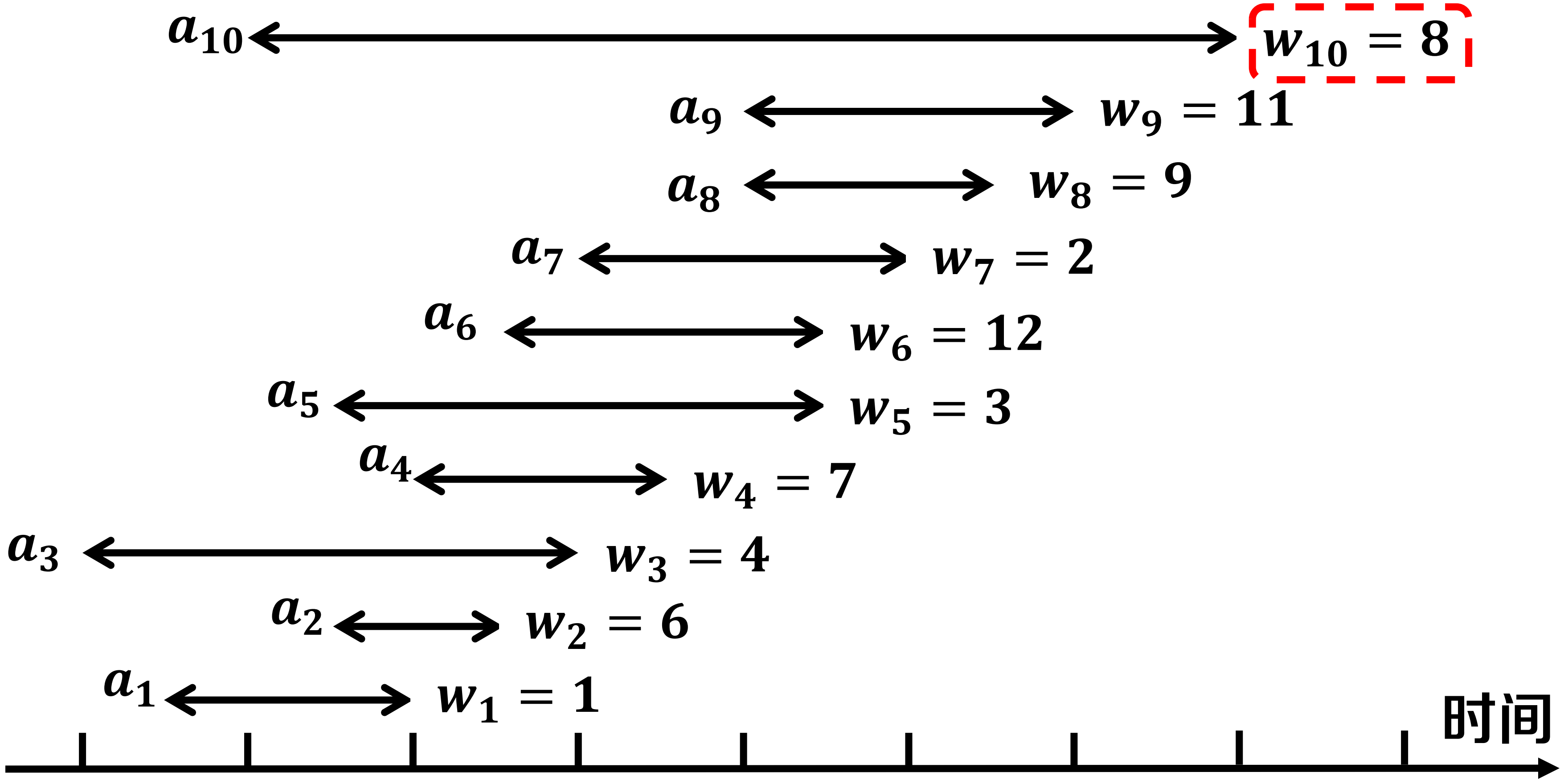
	1	2	3	4	5	6	7	8	9	10
p	0	0	0	1	0	2	3	4	4	0

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	19

$p[i]$：在a_i开始前最后结束的活动

$D[i] = \max\{D[p[i]] + w_i, D[i-1]\}$

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



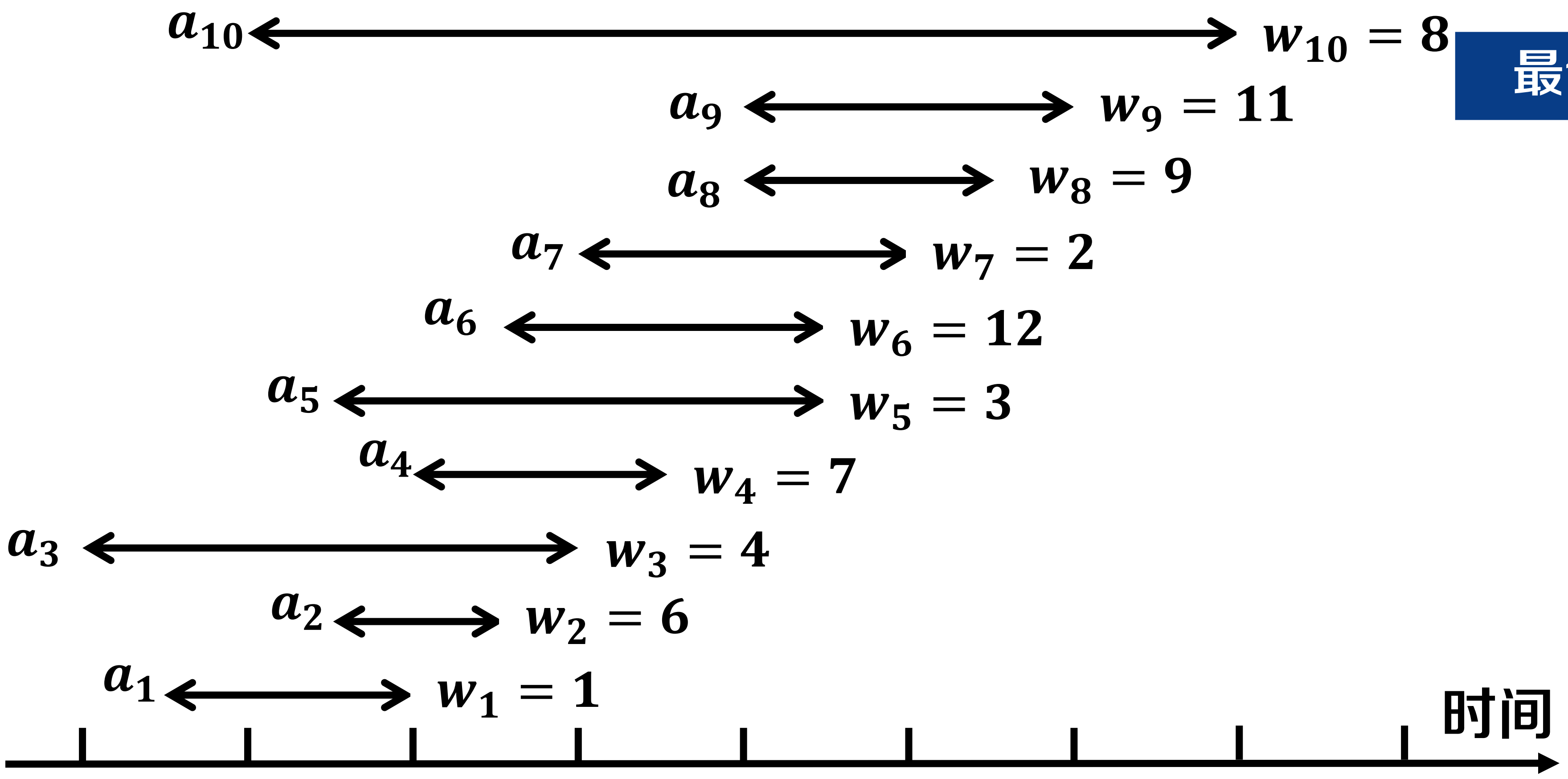
算法实例

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

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	19

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

最优解



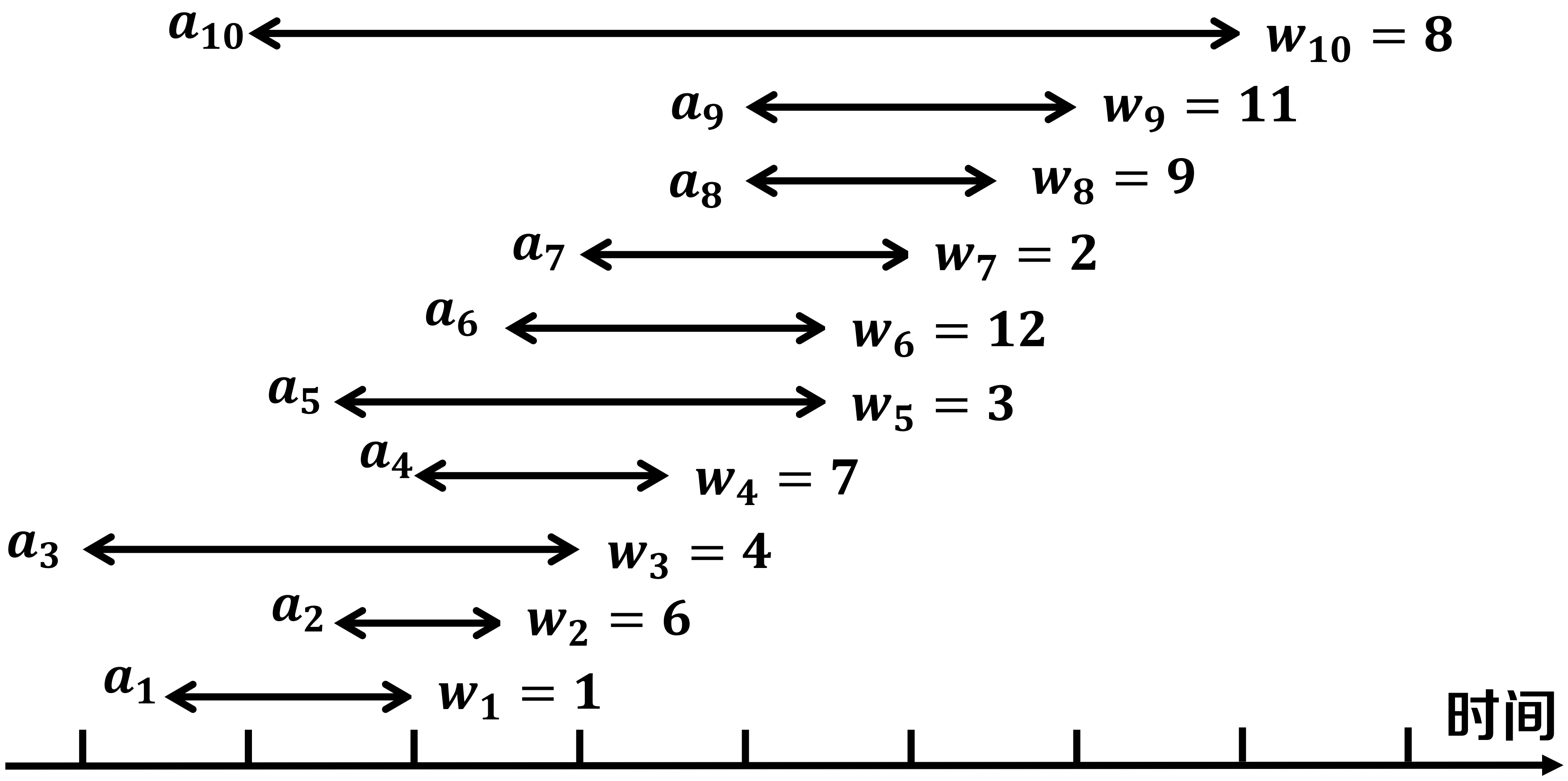
算法实例

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

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	19

活动集合$S' = \{\}$

	1	2	3	4	5	6	7	8	9	10
Rec	1	1	0	1	0	1	0	0	1	$\boxed{0}$



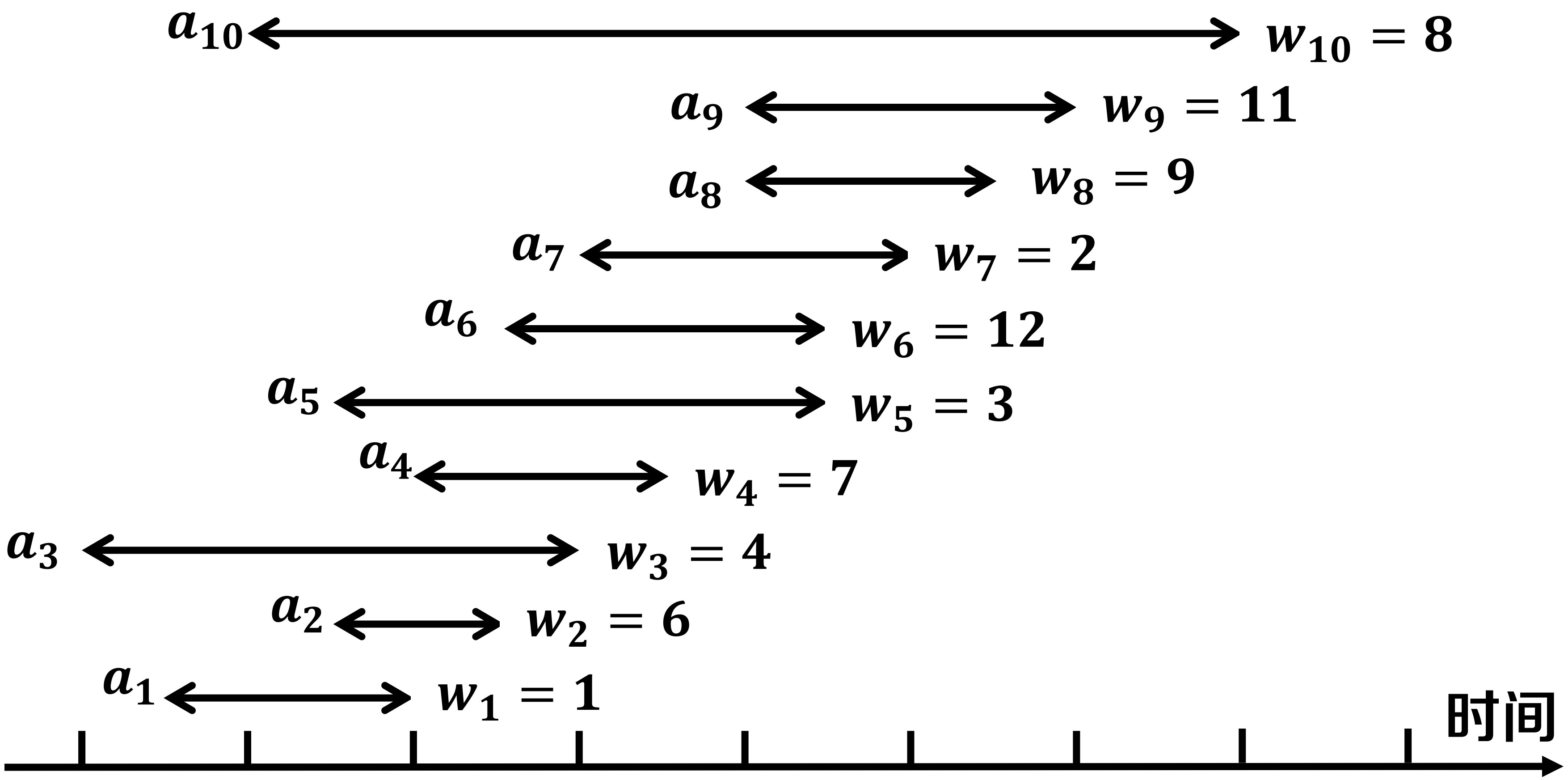
算法实例

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

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	19

活动集合$S' = \{a_9\}$

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



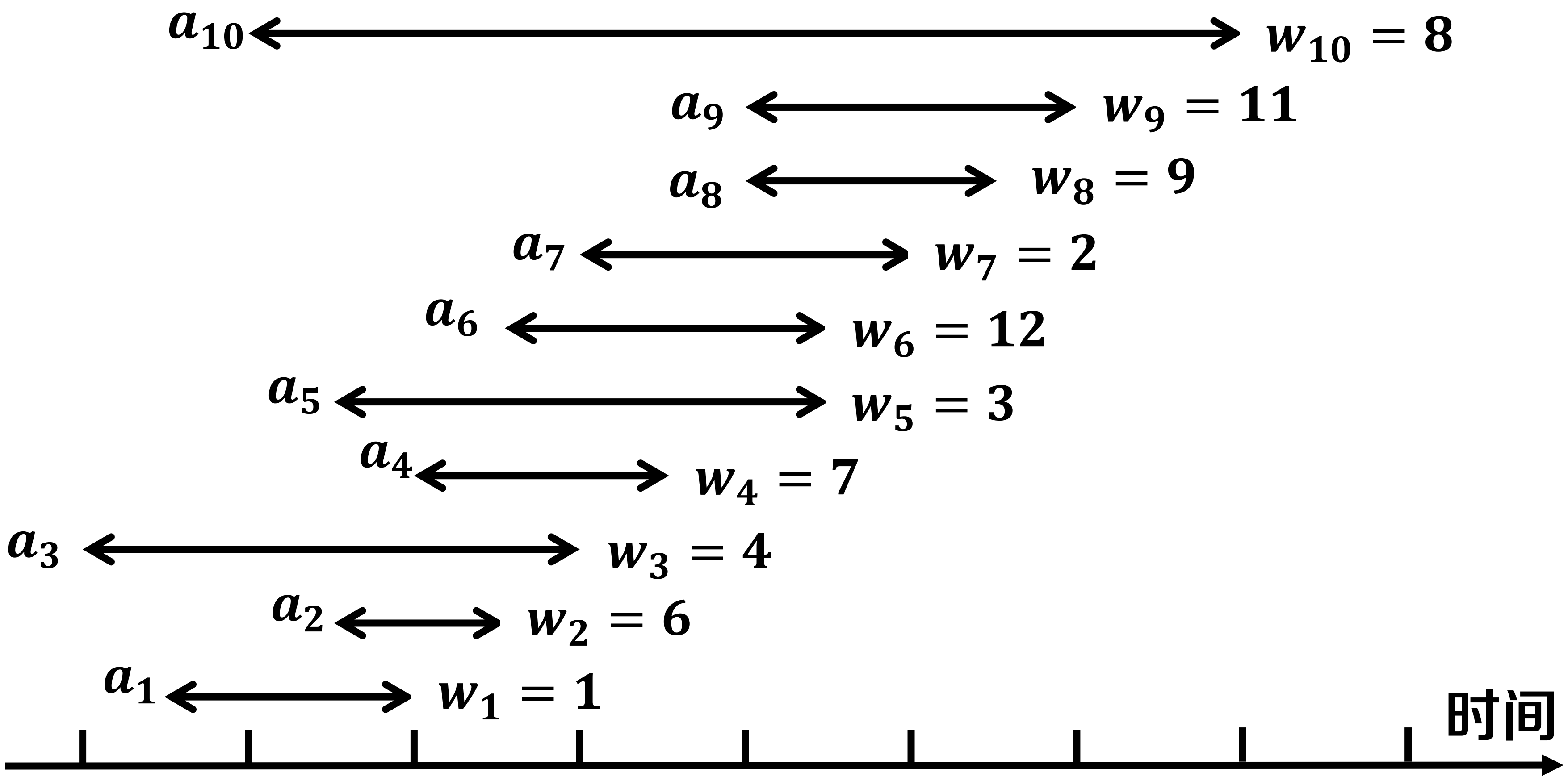
算法实例

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

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	19

活动集合$S' = \{a_9\}$

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



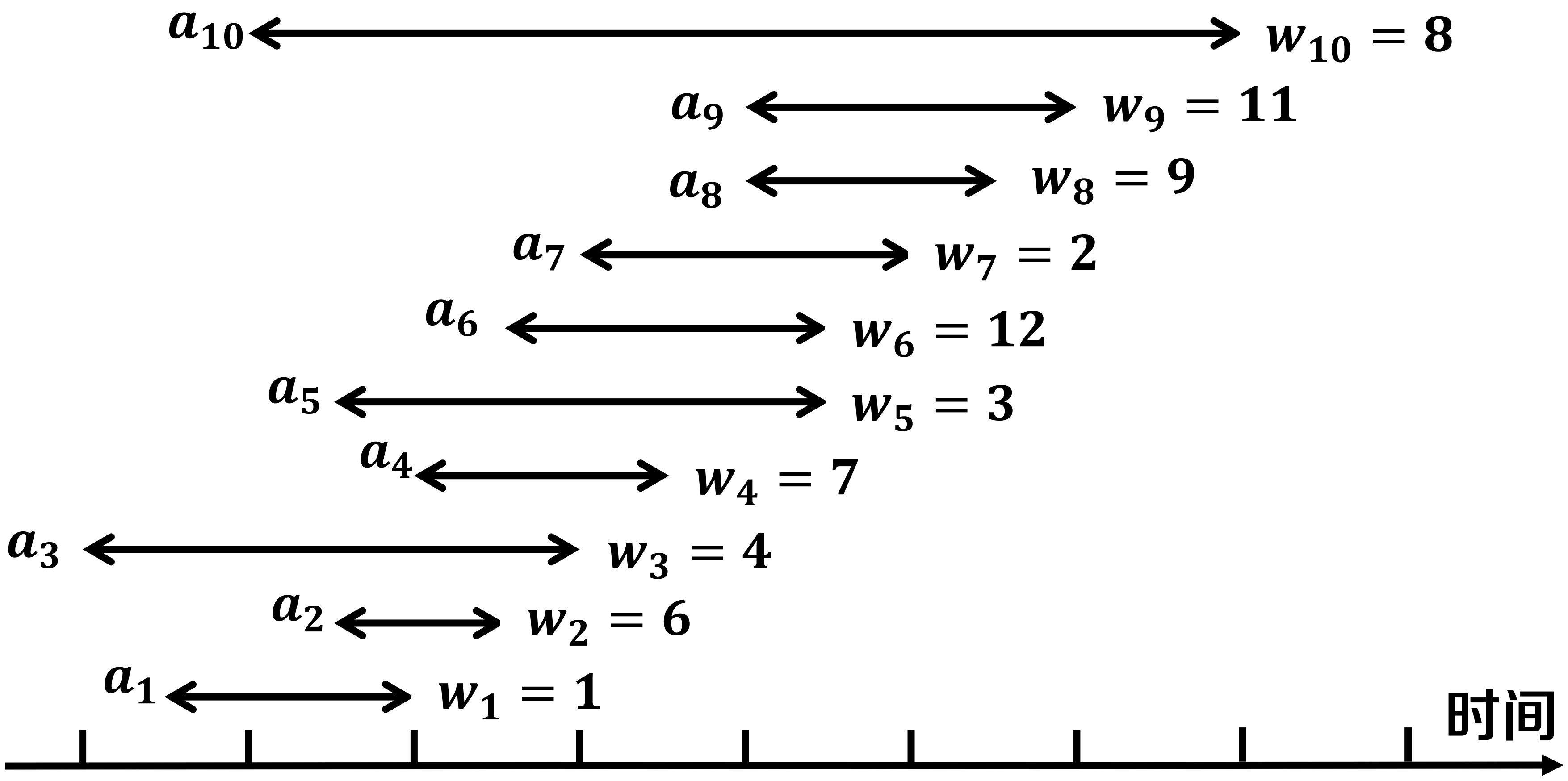
算法实例

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

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	19

活动集合$S' = \{a_4, a_9\}$

	1	2	3	4	5	6	7	8	9	10
Rec	1	1	0	$\boxed{1}$	0	1	0	0	1	0



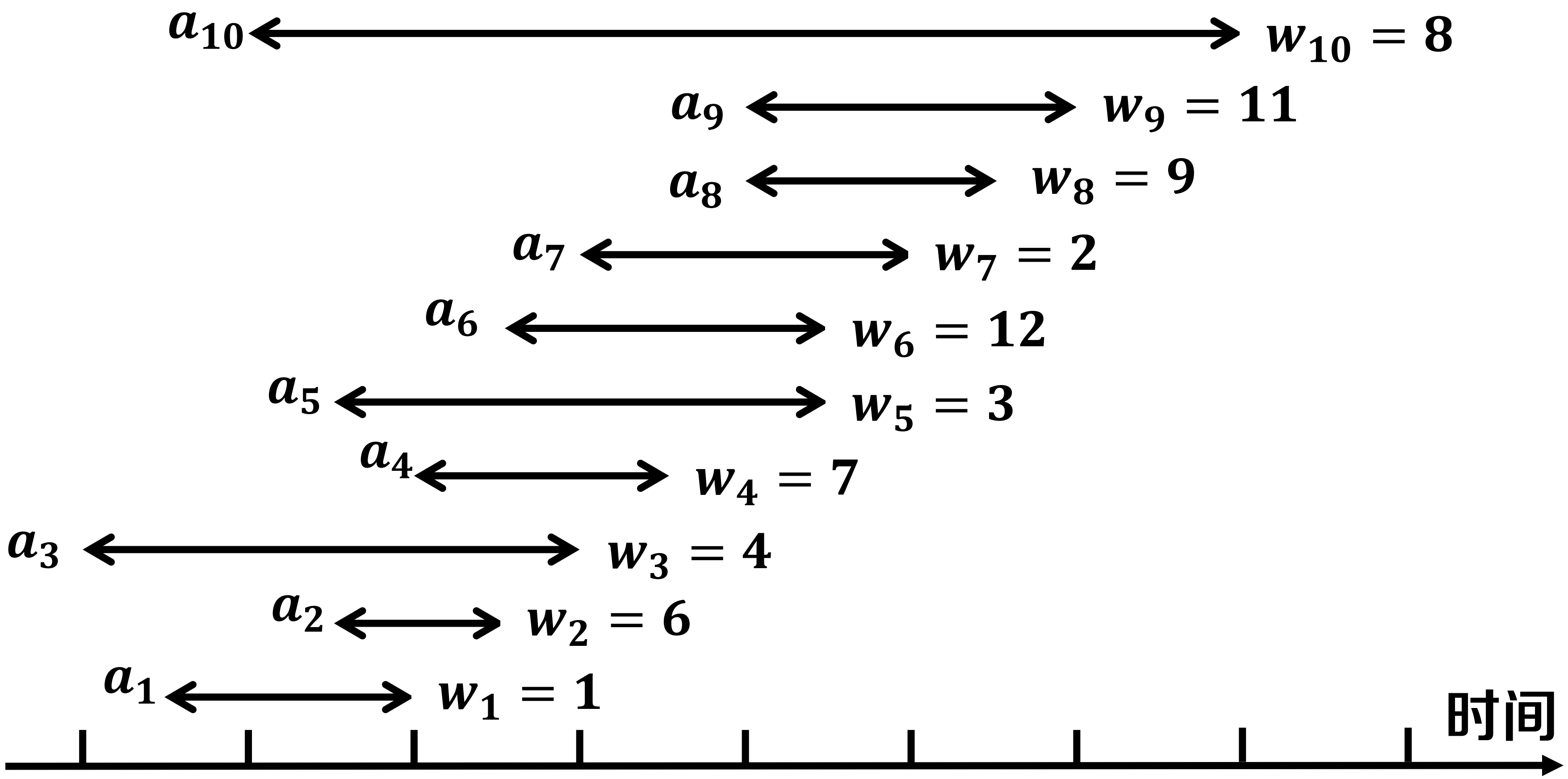
算法实例

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

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	19

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

活动集合$S' = \{a_4, a_9\}$



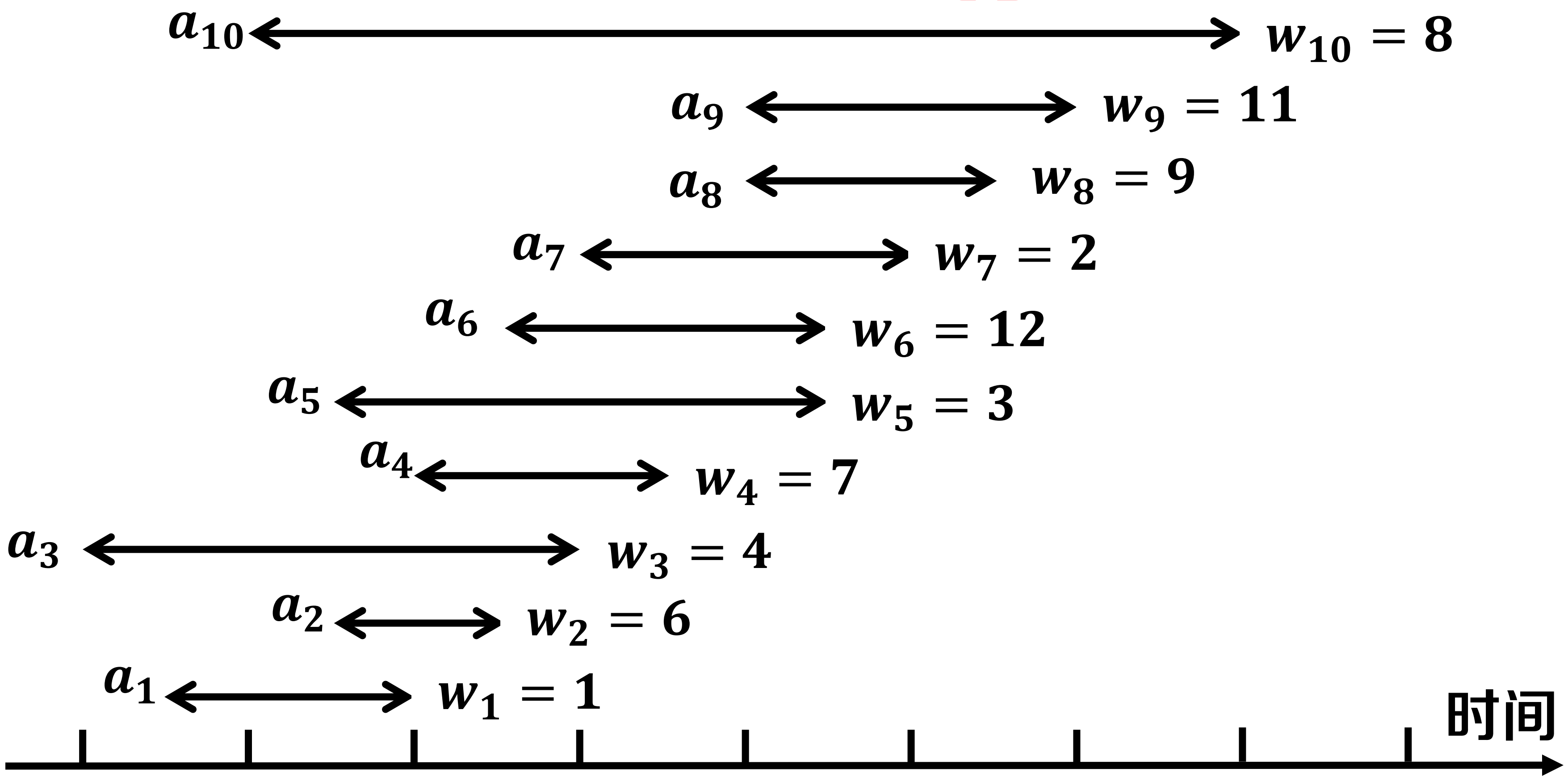
算法实例

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

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	19

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

活动集合$S' = \{a_1, a_4, a_9\}$



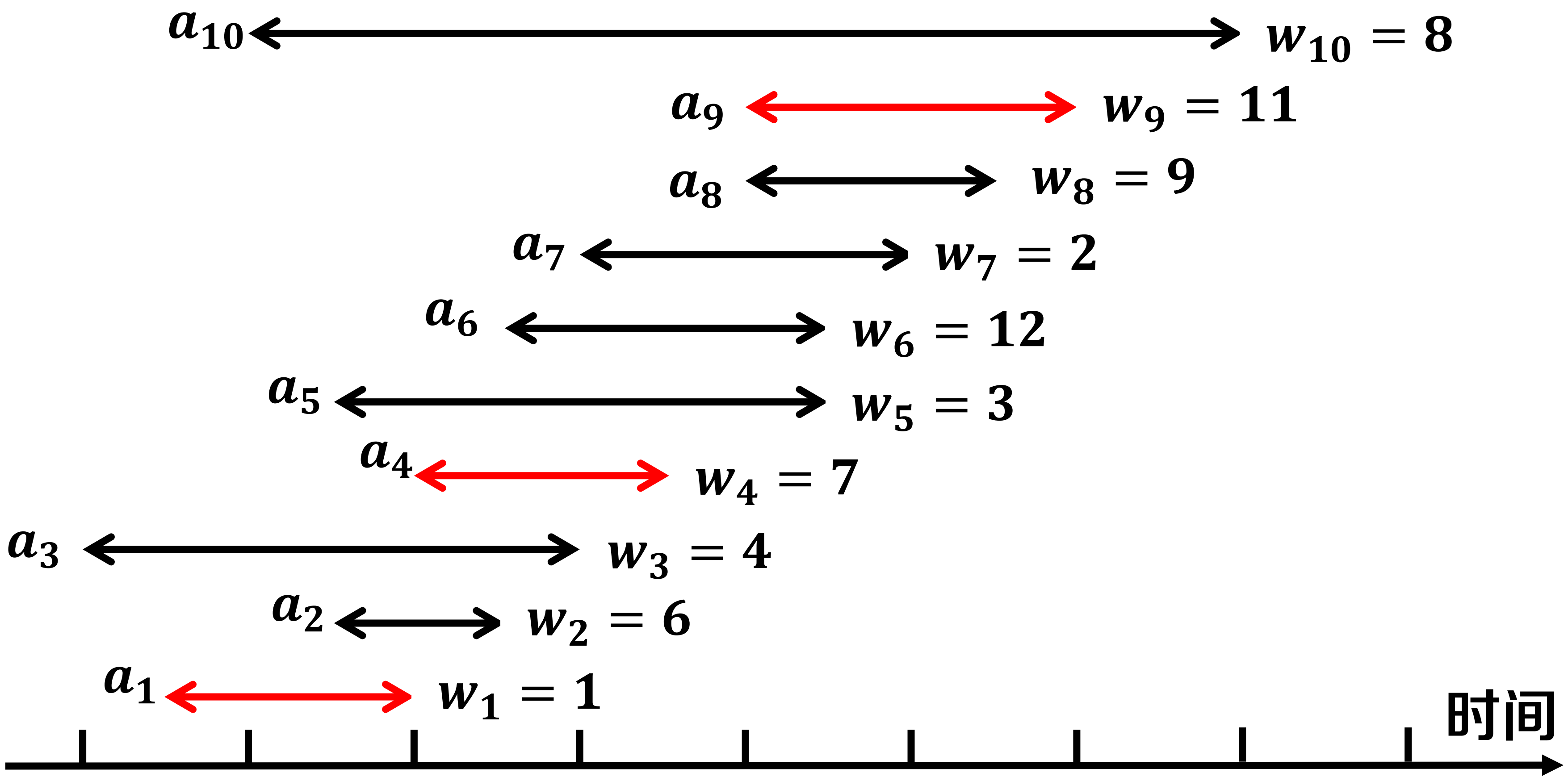
算法实例

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

	0	1	2	3	4	5	6	7	8	9	10
D	0	1	6	6	8	8	18	18	18	19	19

活动集合$S' = \{a_1, a_4, a_9\}$

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



动态规划：伪代码

输入: 活动集合$S=\{a_1,a_2,...,a_n\}$,
每个活动a_i的起止时间s_i, f_i和权重w_i
输出: 不冲突活动的最大子集S'
//预处理
把活动按照结束时间升序排序
for $i \leftarrow 1$ *to* n do
  二分查找求解$p[i]$
end
//初始化
新建数组$D[0..n]$, $Rec[1..n]$
$D[0] \leftarrow 0$

预处理

动态规划：伪代码

输入: 活动集合$S = \{a_1, a_2, ..., a_n\}$,

每个活动a_i的起止时间s_i, f_i和权重w_i

输出: 不冲突活动的最大子集S'

//预处理

把活动按照结束时间升序排序

for $i \leftarrow 1$ *to* n do

 二分查找求解$p[i]$

end

//初始化

新建数组$D[0..n]$, $Rec[1..n]$

$D[0] \leftarrow 0$

初始化

动态规划：伪代码

//动态规划

对每个子问题

$$
\begin{array}{l}
\textbf{for } j \leftarrow 1 \ to \ n \textbf{ do} \\
\quad \textbf{if } D[p[j]] + w_j > D[j-1] \textbf{ then} \\
\quad\quad D[j] \leftarrow D[p[j]] + w_j \\
\quad\quad Rec[j] \leftarrow 1 \\
\quad \textbf{end} \\
\quad \textbf{else} \\
\quad\quad D[j] \leftarrow D[j-1] \\
\quad\quad Rec[j] \leftarrow 0 \\
\quad \textbf{end} \\
\textbf{end}
\end{array}
$$

动态规划：伪代码

```
//动态规划
for j ← 1 to n do
    if D[p[j]] + w_j > D[j − 1] then
        D[j] ← D[p[j]] + w_j
        Rec[j] ← 1
    end
    else
        D[j] ← D[j − 1]
        Rec[j] ← 0
    end
end
```

选择活动a_j

动态规划：伪代码

```
//动态规划
for j ← 1 to n do
    if D[p[j]] + w_j > D[j − 1] then
        D[j] ← D[p[j]] + w_j
        Rec[j] ← 1
    end
    else
        D[j] ← D[j − 1]
        Rec[j] ← 0
    end
end
```

不选活动a_j

动态规划：伪代码

```
//输出方案
k ← n
while k >0 do
    if Rec[k] = 1 then
        print 选择a[k]
        k ← p[k]
    end
    else
        k ← k − 1
    end
end
return D[n]
```

选择活动a_k

动态规划：伪代码

```
//输出方案
k ← n
while k >0 do
    if Rec[k] = 1 then
        print 选择a[k]
        k ← p[k]
    end
    else
        k ← k − 1
    end
end
return D[n]
```

回溯子问题

动态规划：伪代码

//输出方案

$k \leftarrow n$

while $k > 0$ **do**

  if $Rec[k] = 1$ **then**

    print **选择**$\boldsymbol{a[k]}$

    $k \leftarrow p[k]$

  end

  else

    $k \leftarrow k - 1$

  end

end

return $D[n]$

不选活动a_k

动态规划：复杂度分析

```
输入: 活动集合S = {a_1, a_2, ..., a_n},
      每个活动a_i的起止时间s_i, f_i和权重w_i
输出: 不冲突活动的最大子集S'
//预处理和初始化
把活动按照结束时间升序排序                     O(n log n)
for i ← 1 to n do                               O(n log n)
    二分查找求解p[i]
end
新建数组D[0..n], Rec[1..n]
D[0] ← 0
//动态规划
for j ← 1 to n do                               O(n)
    if D[p[j]] + w_j > D[j − 1] then
        D[j] ← D[p[j]] + w_j
        Rec[j] ← 1
    end
    else
        D[j] ← D[j − 1]
        Rec[j] ← 0
    end
end
//输出方案
k ← n
while k >0 do                                   O(n)
    if Rec[k] = 1 then
        print 选择a[k]
        k ← p[k]
    end
    else
        k ← k − 1
    end
end
return D[n]
```

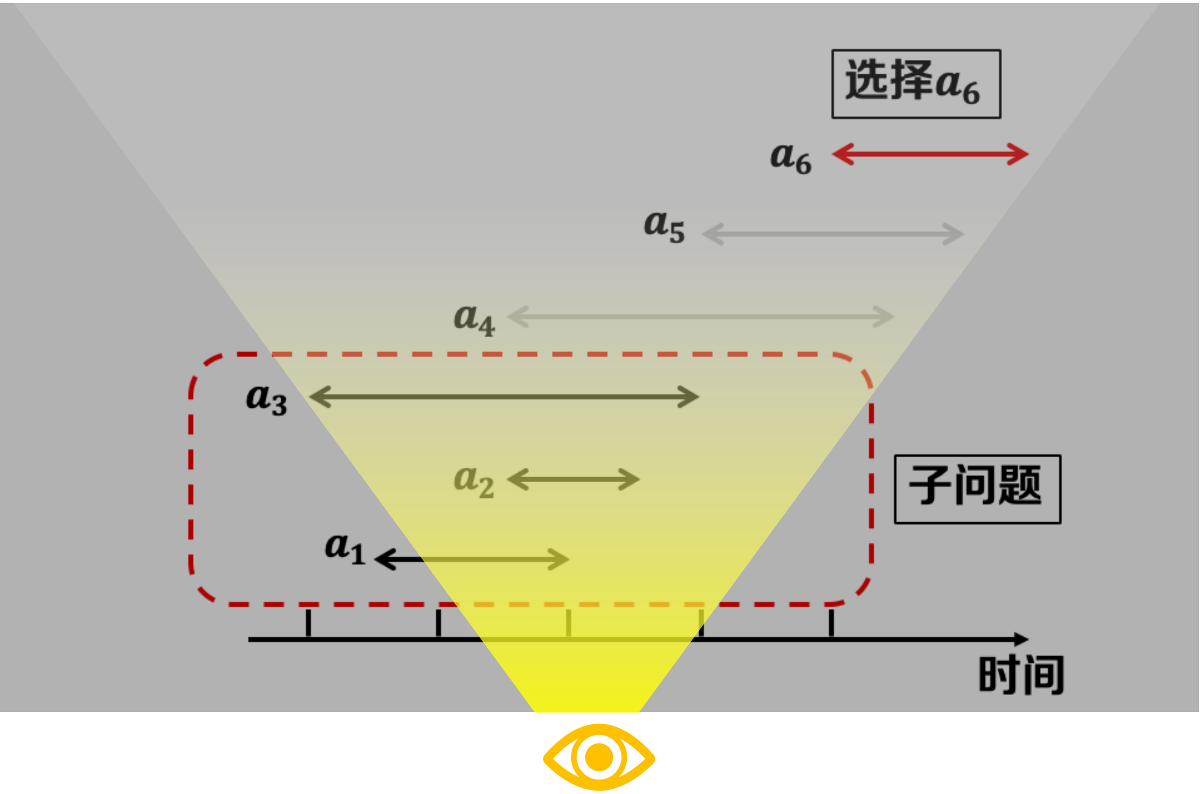
时间复杂度：$O(n\log n)$

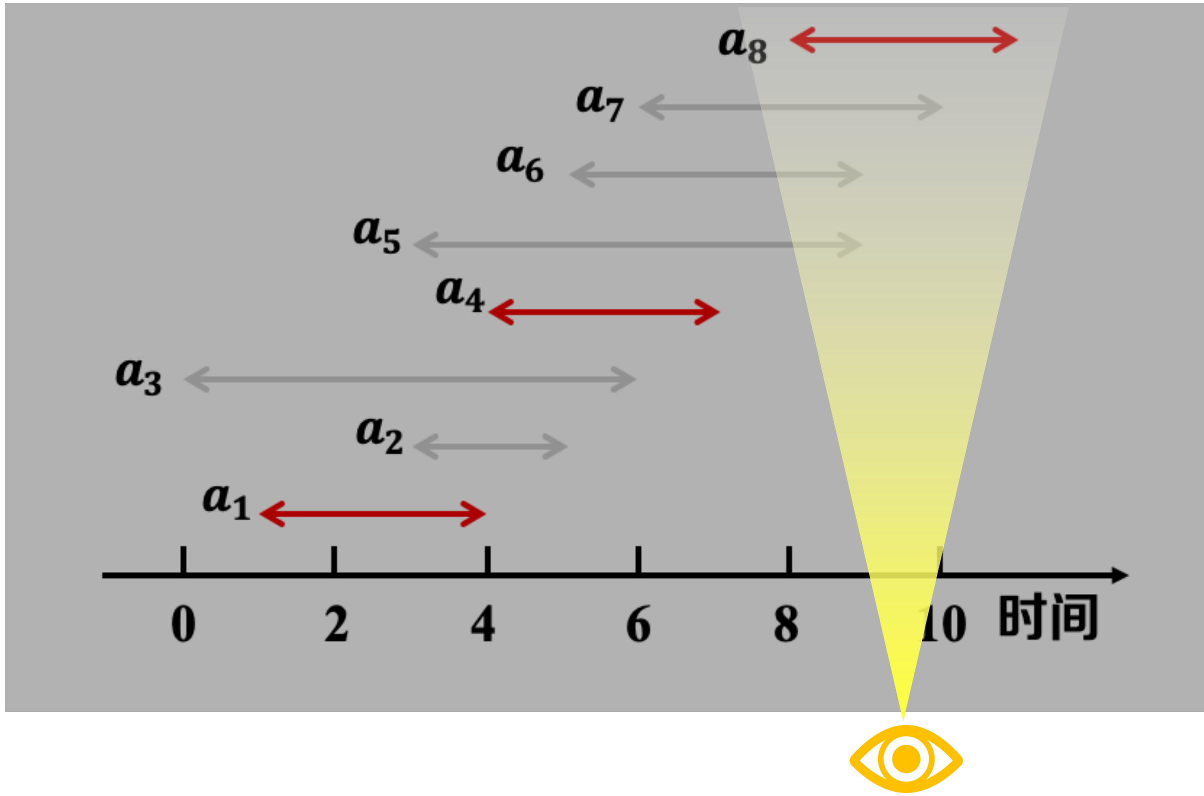
活动选择问题：动态规划 vs. 贪心策略

带权活动选择问题 → 活动选择问题

权重均为1

性质更好





求解子问题，组合最优解

动态规划：考察全局

直接做决策，构造最优解

贪心策略：考察局部

课程总结

算法设计与分析

分而治之篇
- 归并排序
- 最大子数组问题Ⅰ
- 逆序计数问题
- 快速排序
- 次序选择问题

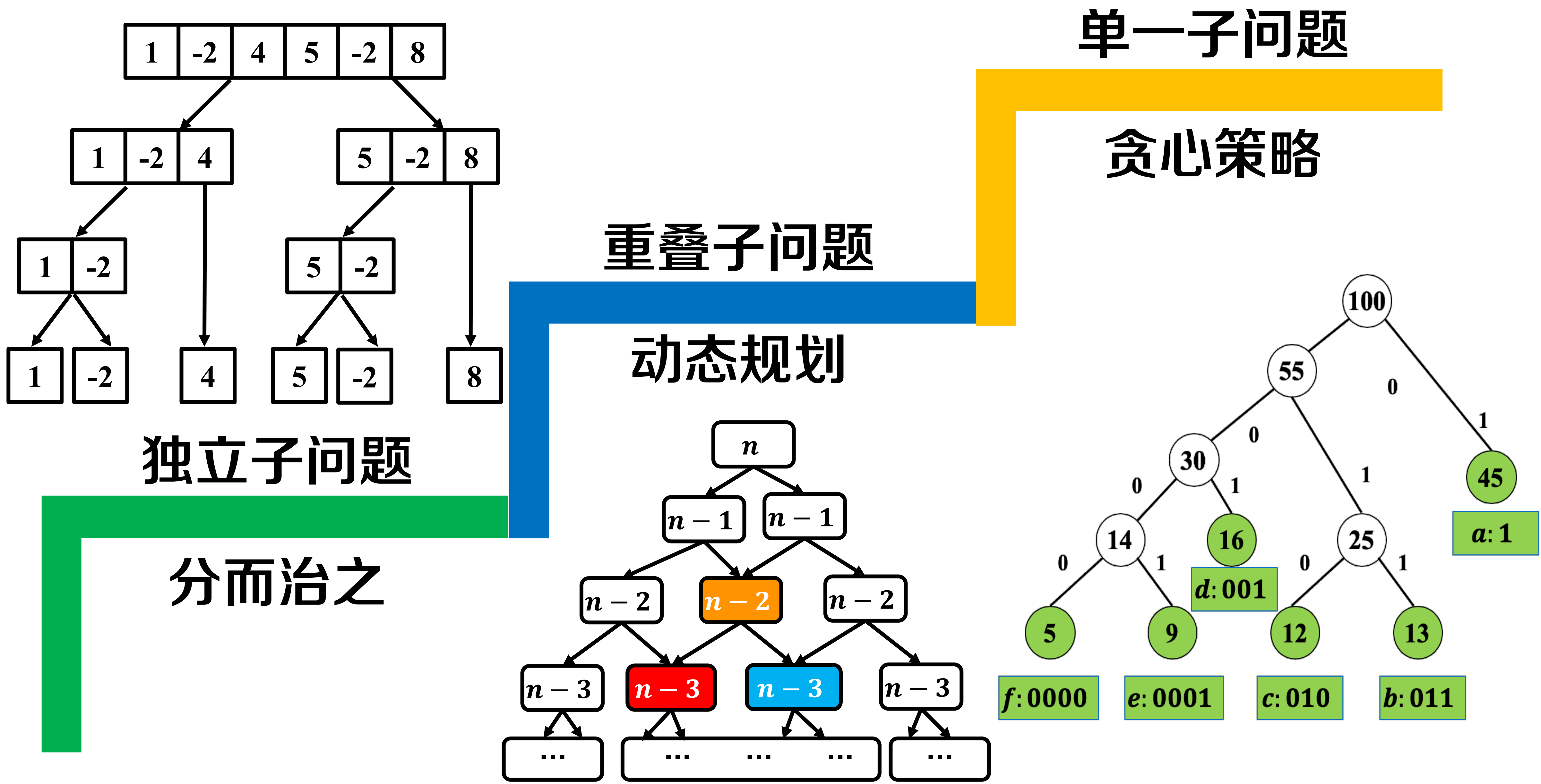
动态规划篇
- 0-1背包问题
- 最大子数组问题Ⅱ
- 最长公共子序列问题
- 最长公共子串问题
- 最小编辑距离问题
- 钢条切割问题
- 矩阵链乘法问题

贪心策略篇
- 部分背包问题
- 霍夫曼编码
- 活动选择问题

课程总结

分而治之	动态规划	贪心策略
分解原问题	问题结构分析	提出贪心策略
↓	↓	↓
解决子问题	递推关系建立	证明策略正确
↓	↓	
合并问题解	自底向上计算	
	↓	
	最优方案追踪	

课程总结



单一子问题
贪心策略
重叠子问题
动态规划
独立子问题
分而治之
1 -2 4 5 -2 8
1 -2 4
5 -2 8
n
$n-1$
$n-2$
$n-3$
…
100
55
30
14
16
25
45
5
9
12
13
a: 1
d: 001
f: 0000
e: 0001
c: 010
b: 011