EDA软件设计-绑定

2024-2025年春季学期

主讲人：孙静翎

有约束的调度

Constrained Scheduling

- 约束调度
  - 一般情况下是NP完全问题
  - 在面积或资源的约束下最小化延迟（ML-RCS）
  - 使受到延迟约束的资源最小化（MR-LCS）
- 确切解决方法
  - ILP：整数线性规划（Integer linear program）
  - Hu算法：适用于只有一种资源类型的问题
- 启发式算法
  - 列表调度（List scheduling）
  - 力导向调度（Force-directed scheduling）

列表调度算法：ML-RCS

List scheduling algorithm for minimum latency

```
1 LIST_L( G(V, E), a) {
2        t0 = 0;  l = 1;
3        重复执行以下步骤 {
4                对于每种资源类型 k = 1, 2, …, n_res {
5                   确定就绪的操作集 U_l,k;（①还未安排开始周期，②前置结点已在当前周期完成）
6                   确定当前正在进行的操作集 T_l,k;
7                   选择一个子集S_k ⊆ U_l,k ,使得 |S_k| + |T_l,k| ≤ a_k;
8                   在步骤 l 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;
9                }
10               l = l + 1;
11       }
12       直到 vn 被调度 ;
13 }
```

列表调度算法：MR-LCS

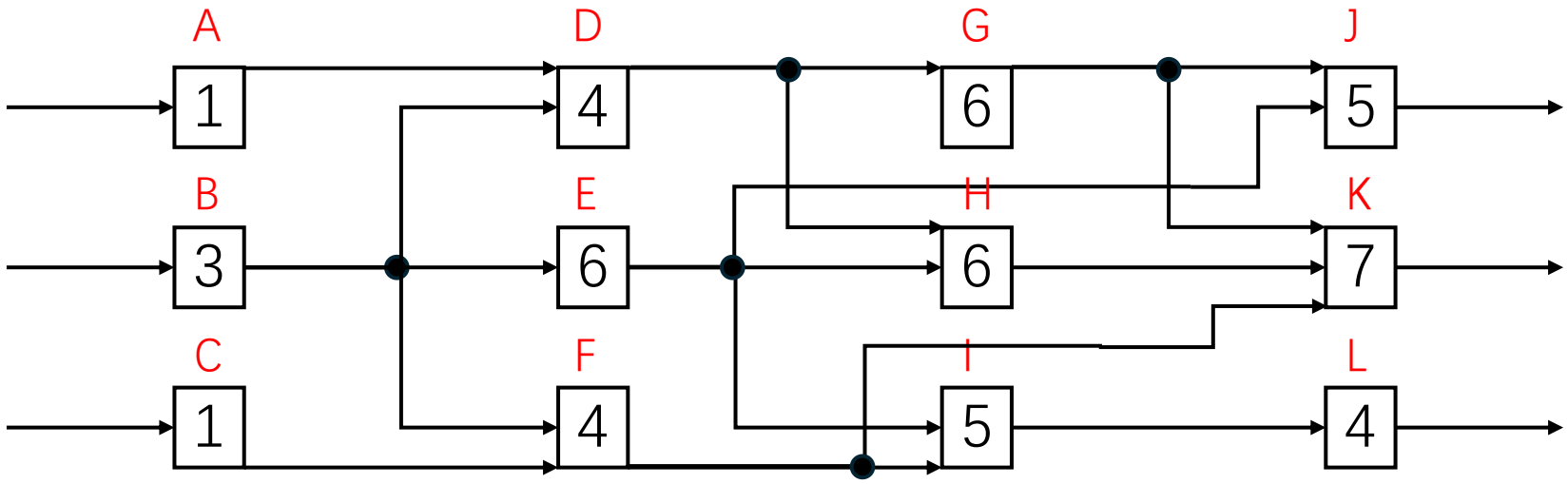
List scheduling algorithm for minimum resource usage

```
1 LIST_R( G(V, E), λ ) {
2         a = 1;（a代表资源数）
3         通过 ALAP ( G(V, E), λ) 计算所有操作的最晚开始时间 t^L ;
4         if (t0 < 0)
5            return (Ø);
6          t0 = 0;    l = 1;
7         重复执行以下步骤{
8                   对于每种资源类型 k = 1, 2, …, nres {
9                      确定就绪的操作集 Ul,k;
10                     对Ul,k中的每个顶点，计算其slacks si = t^L_i – l ;
11                     调度所有松弛度为零的候选操作，并更新 a;
12                     调度不需要额外资源的候选操作;
13                     }
14                  l = l + 1;
15        }
16        直到 vn 被调度完成;
17        return (t, a);
18 }
```

随堂作业

in-class assignment

请写出以下调度图的关键结点，以及代表每个结点关键性的值

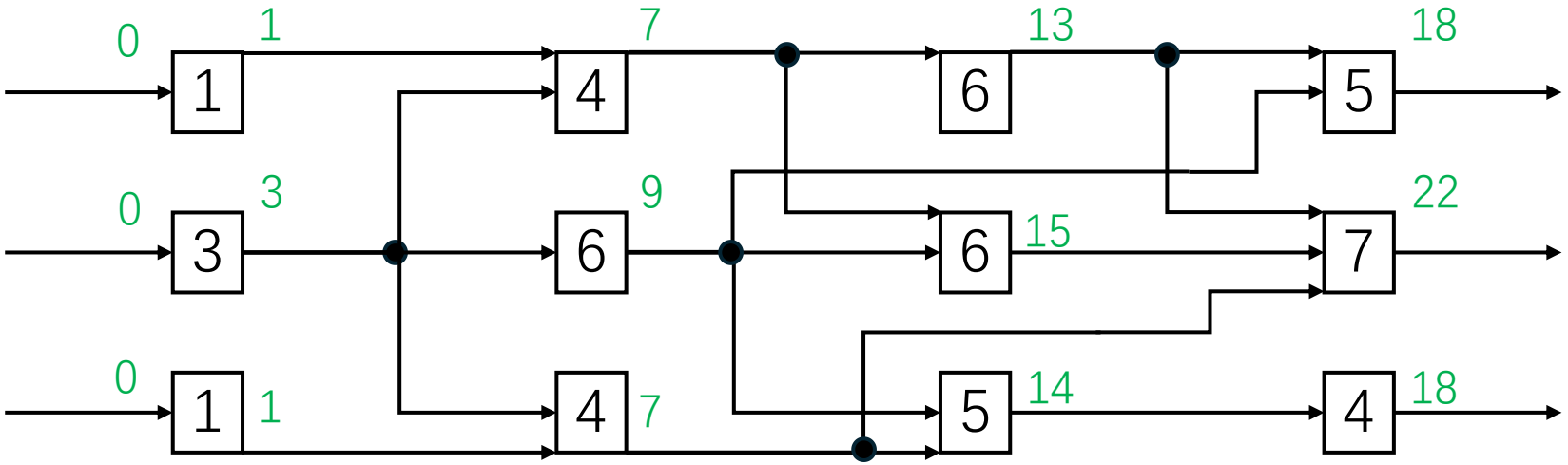


随堂作业

in-class assignment

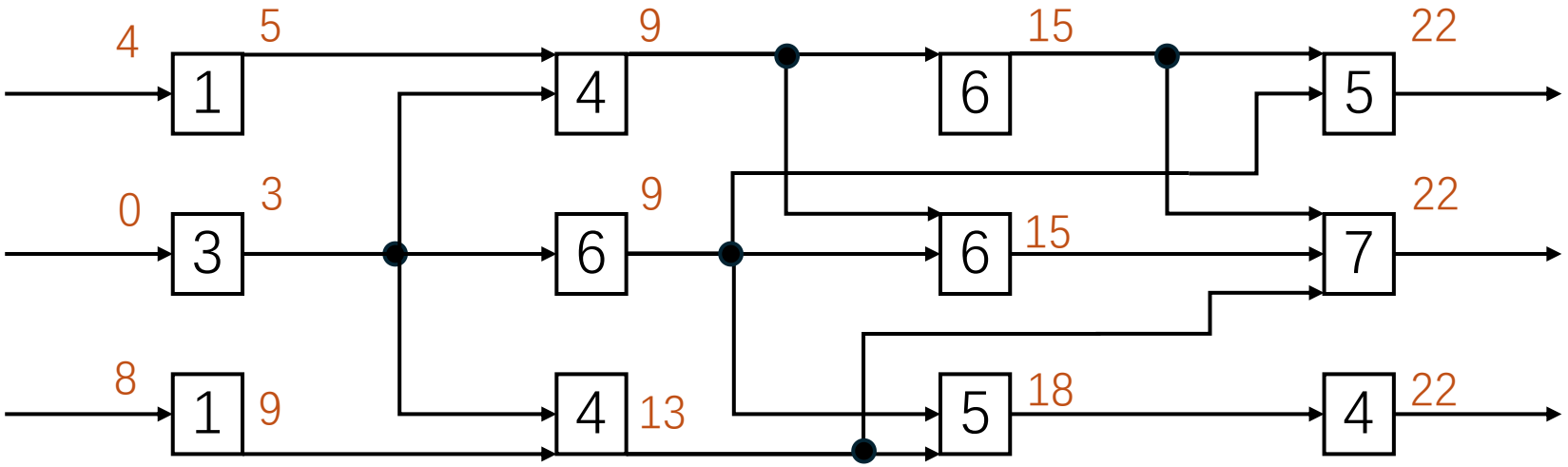
到达时间

(Arrival Time)



要求时间

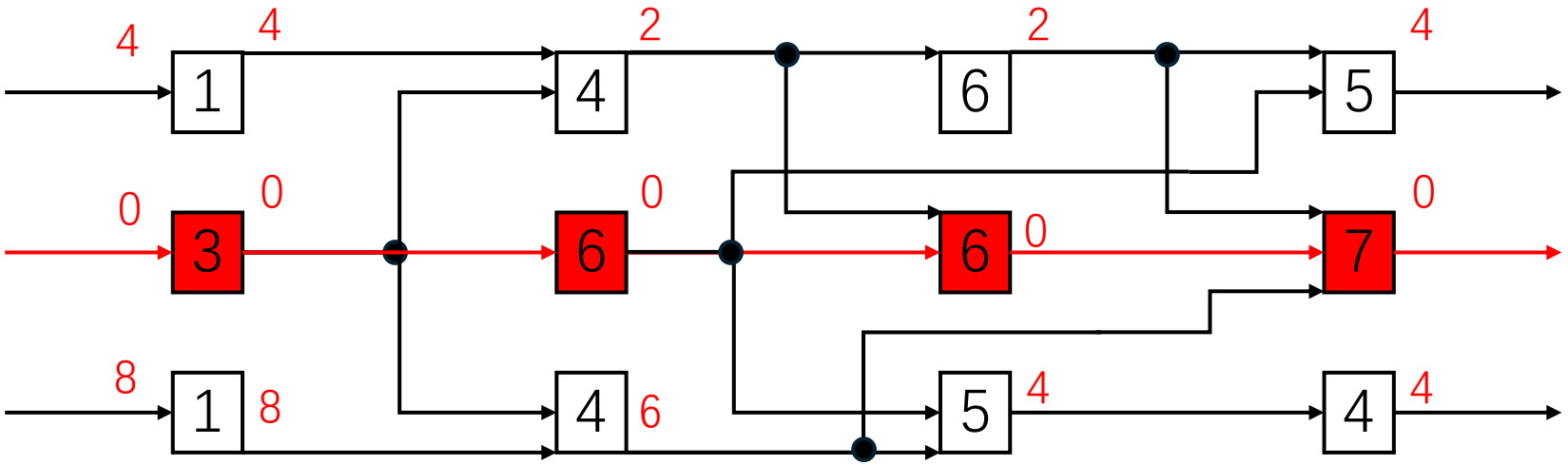
(Required Time)



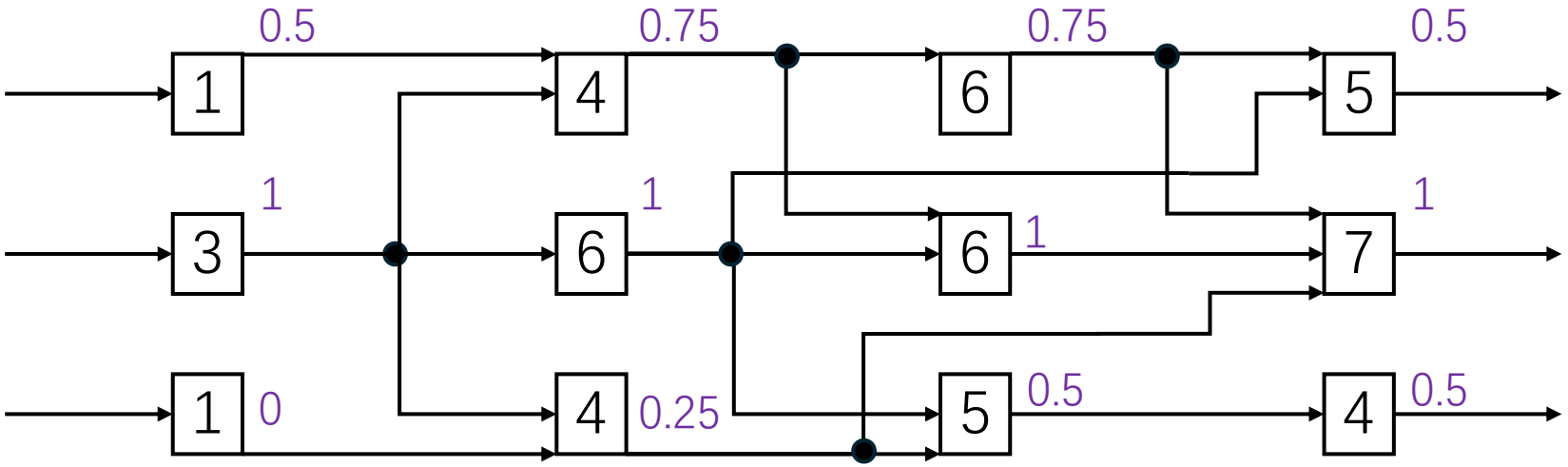
随堂作业

in-class assignment

裕量（Slack）



关键性（Criticality）



正式进入绑定

相关术语

Related Terminologies

- Allocation（资源分配）： 可用资源的数量
- Binding（绑定）： 操作与资源之间的对应关系
- Sharing（共享）： 多个操作用同一个资源
- Optimum binding/sharing（最优绑定/共享）： 最小化资源使用

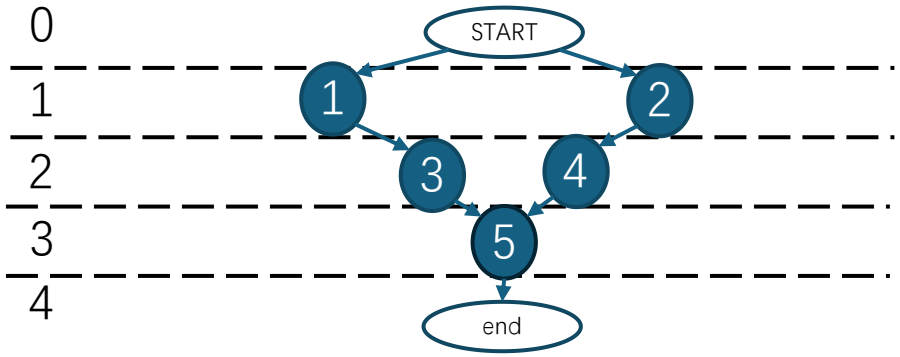
特殊情况

- Dedicated resources（专用资源）
  - 每个操作分配一个资源
  - 不存在共享
- One multi-task resource（可进行多种任务的资源）
  - eg：ALU（算术逻辑单元，一种组合逻辑电路）
- One resource per type（每一种资源只有一个）

最优共享问题

Optimum Sharing Problem

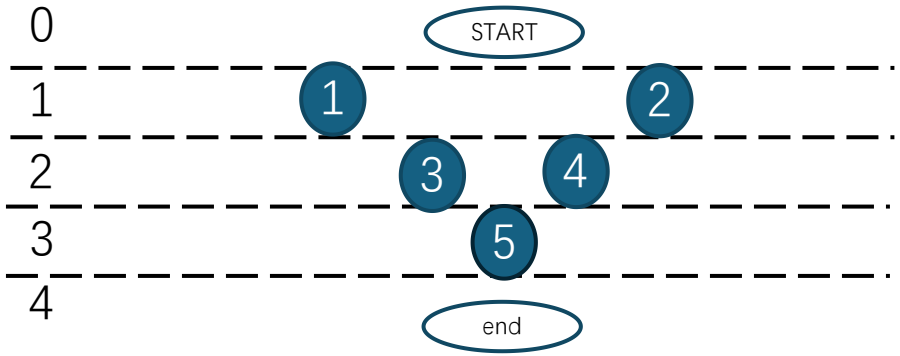
- 输入：已调度的顺序图
  - 已经明确操作的并发性
- 独立考虑操作类型
  - – 问题分解
  - – 针对每种资源类型进行分析

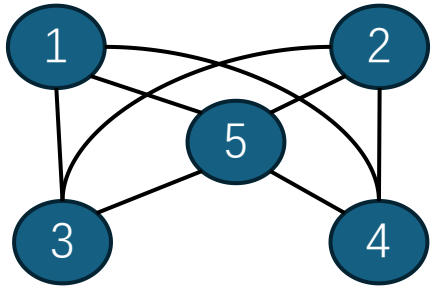


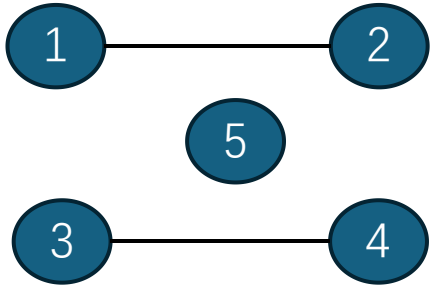
兼容与冲突

Compatibility and Conflicts

- 能兼容的操作:
  - – 相同类型
  - – 非并发
- 兼容图（Compatibility graph）：
  - – 顶点：操作
  - – 边：兼容关系
- 冲突图（Conflict graph）：
  - – 兼容图的补图



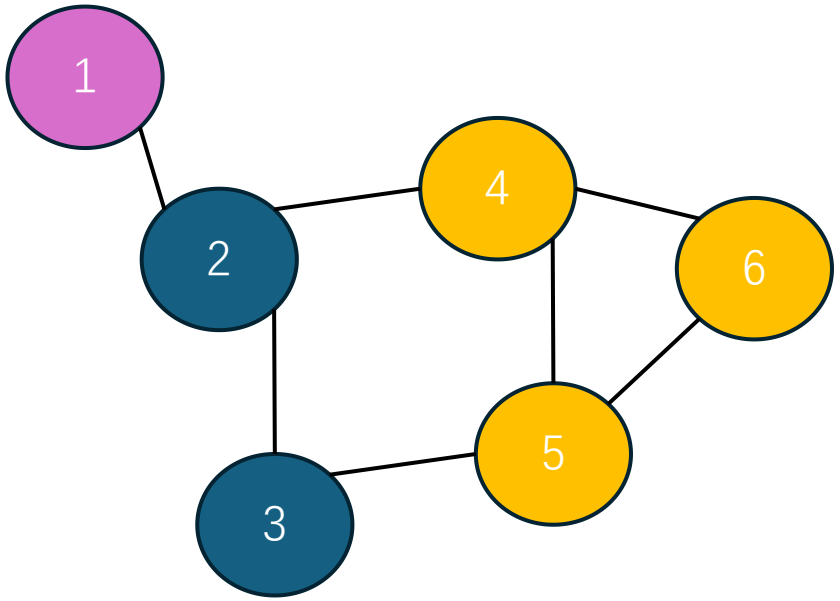




团

Clique

团（Clique） 是指一个无向图中的一个**顶点子集**，其中**任意两个顶点之间都有一条边相连**，即该子图是一个**完全子图（complete subgraph）**。



图术语

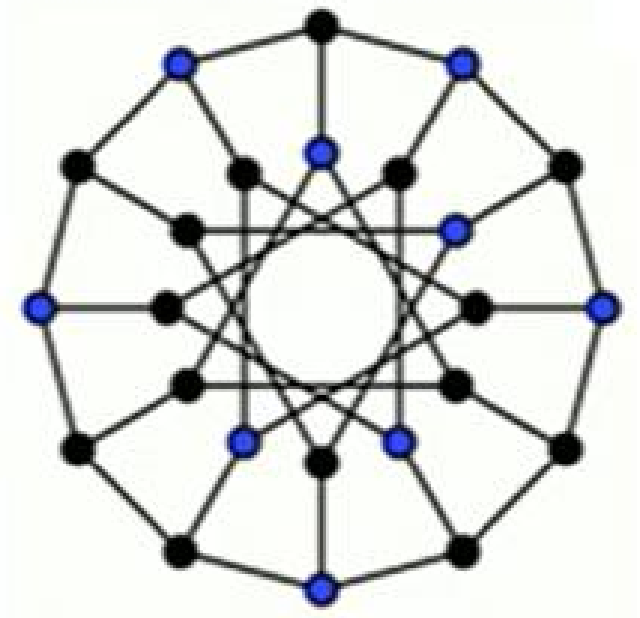
Graph Terminologies

- 团覆盖数（Clique cover number） κ(G)：最小团覆盖的基数（即把图分成团，使得团数最少）

独立集

independent set

独立集（Independent Set），也称为稳定集（Stable Set），是图中一个顶点子集，满足其中任意两个顶点之间都没有边相连。



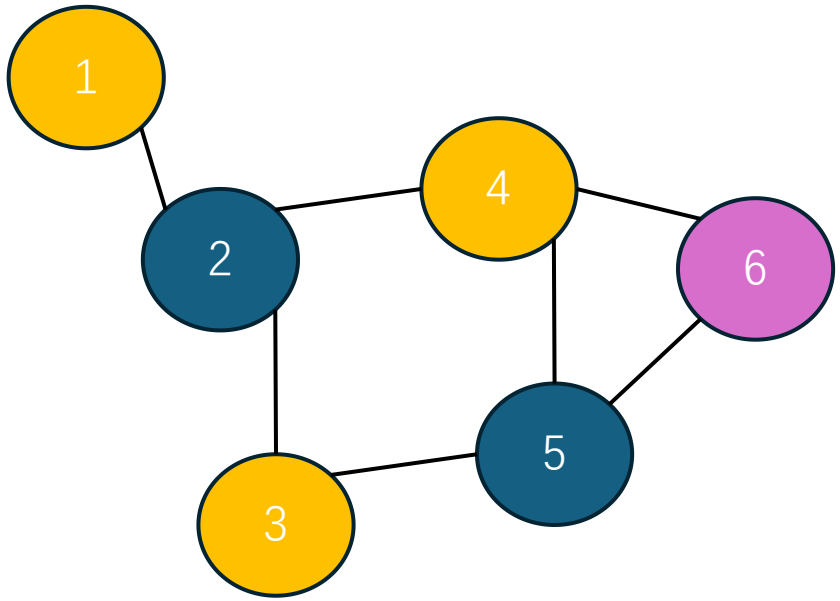
图术语

Graph Terminologies

- 染色数（Chromatic number） χ(G)：最小稳定集划分数（即最少用多少种颜色对图进行着色，使相邻顶点颜色不同）

图术语

Graph Terminologies



图术语

Graph Terminologies

- 团覆盖数（Clique cover number） κ(G)：最小团覆盖的基数（即把图分成团，使得团数最少）

- 染色数（Chromatic number） χ(G)：最小稳定集划分数（即最少用多少种颜色对图进行着色，使相邻顶点颜色不同）

兼容与冲突

Compatibility and Conflicts

- 兼容图（Compatibility graph）：
  - 将图划分为最少数量的团（clique）
  - 计算团覆盖数 $\kappa(G)$
- 冲突图（Conflict graph）：
  - 用尽可能少的颜色对顶点进行着色
  - 计算染色数 $\chi(G)$
- NP 完全问题（NP-complete problems）

兼容图 / 冲突图具有特殊性质

The compatibility/conflict graphs have special properties

- Compatibility graph 兼容图
  - Comparability graph比较图
- Conflict graph冲突图
  - Interval graph区间图

- 多项式时间算法:
  - Golumbic's algorithm
  - Left-edge algorithm

比较图

Comparability graph

- 给定一个偏序集（例如元素之间的“≤”关系），比较图的顶点对应集合中的元素，两个顶点之间有一条边当且仅当这两个元素满足定义的偏序关系。

兼容图->比较图

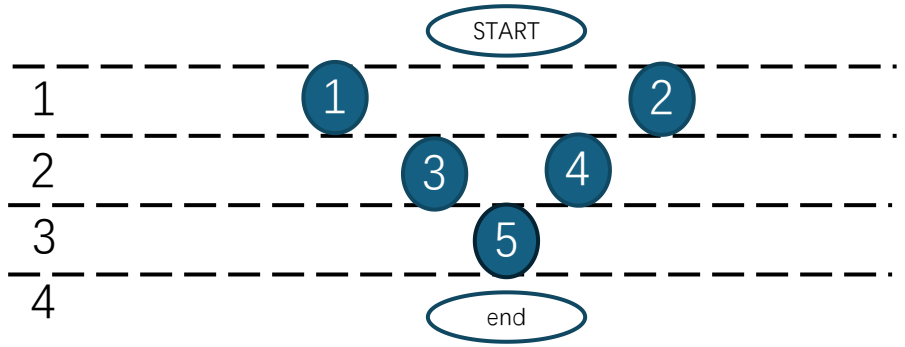
Compatibility graph -> Comparability graph

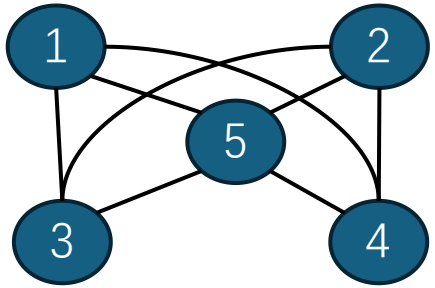
- 在之前的课程中，我们接触到的调度图的兼容图都是比较图

偏序关系：若任务 A 的开始时间早于任务 B 的开始时间，则定义 A≤B

对右图有：1≤3，3≤5，1≤4，

2≤3，2≤4，2≤5，3≤5，4≤5





区间图

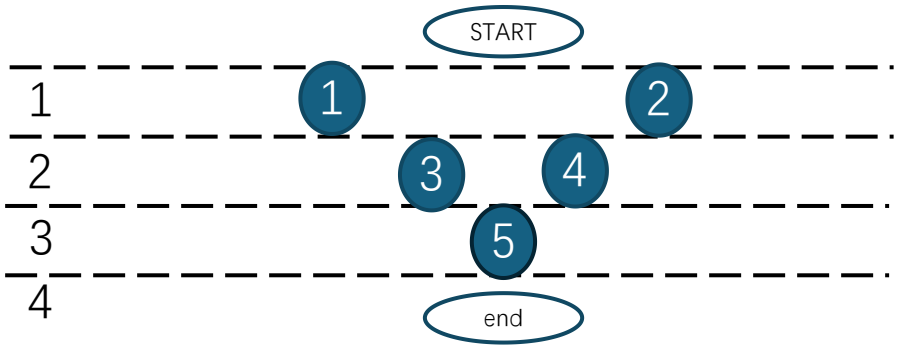
Interval graph

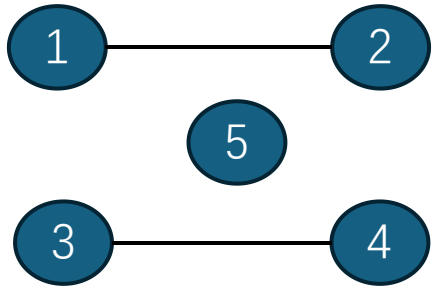
- 给定一些区间，用每个顶点表示一个区间，若两个区间的交集非空，则代表两区间的点间存在一条边，若干区间的相交图为一个区间图

冲突图->区间图

Conflict graph -> Interval graph

- 在之前的课程中，我们接触到的调度图的冲突图都是区间图
- 给定一些区间，用每个顶点表示一个区间，若两个区间的交集非空，则代表两区间的点间存在一条边，若干区间的相交图为一个区间图





兼容图 / 冲突图具有特殊性质

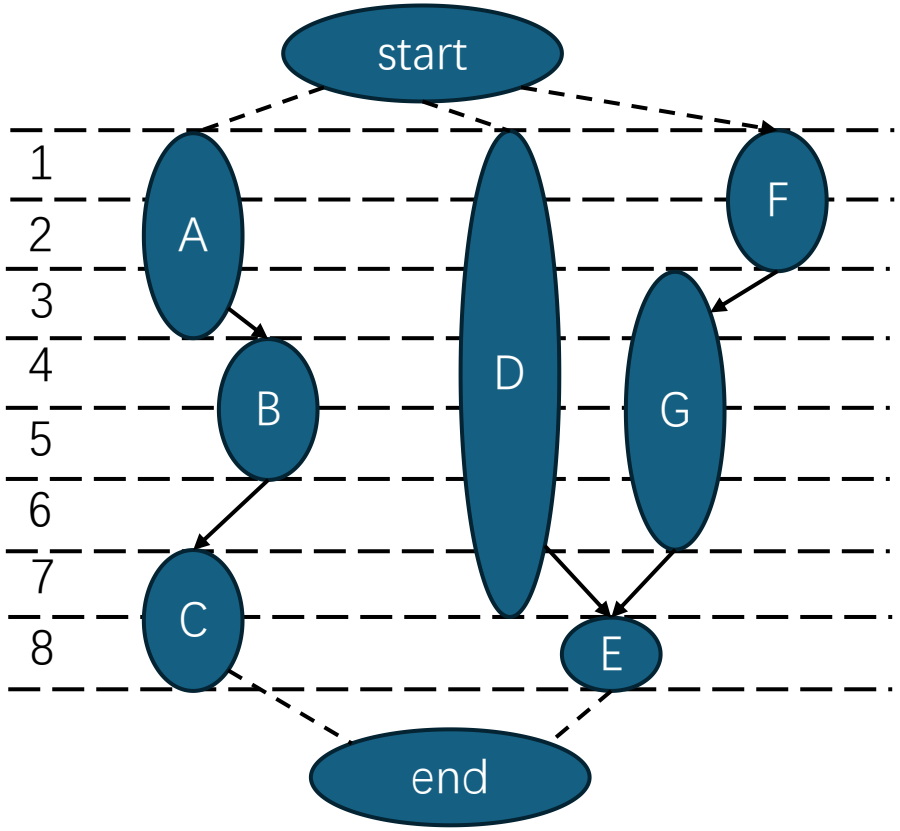
The compatibility/conflict graphs have special properties

- Compatibility graph 兼容图
  - Comparability graph可比较图
- Conflict graph冲突图
  - Interval graph区间图
- 多项式时间算法:
  - Golumbic's algorithm
  - Left-edge algorithm

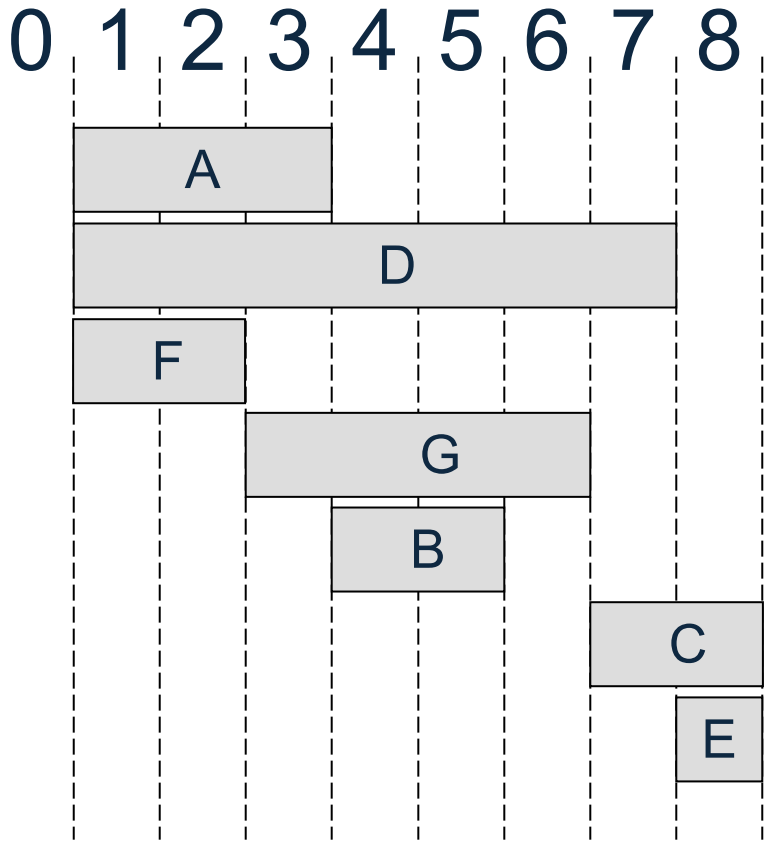
左边缘算法

Left-edge algorithm

- 输入：
  - 一组具有左边界和右边界的区间（即开始时间和结束时间）
  - 一组颜色（初始时只有一种颜色）
- 步骤：
  - 按照左边界对区间进行排序
  - 从排序后的列表中，使用第一种颜色分配互不重叠的区间
  - 当所有可能分配的区间都用完后，增加颜色计数器并重复上述过程



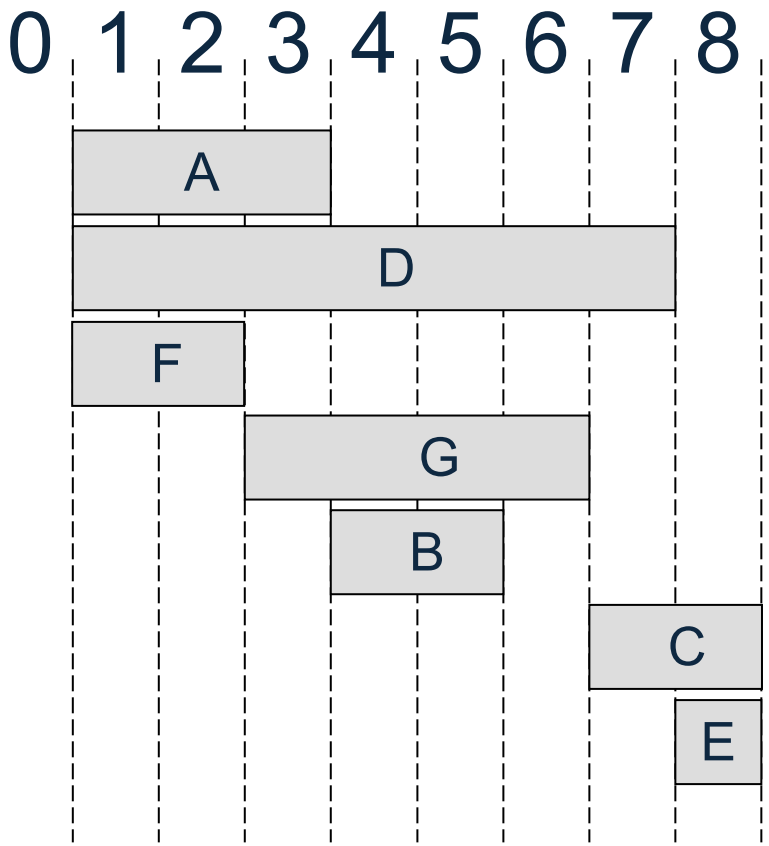
A: 1-3
B: 4-5
C: 7-8
D: 1-7
E: 8-8
F: 1-2
G: 3-6

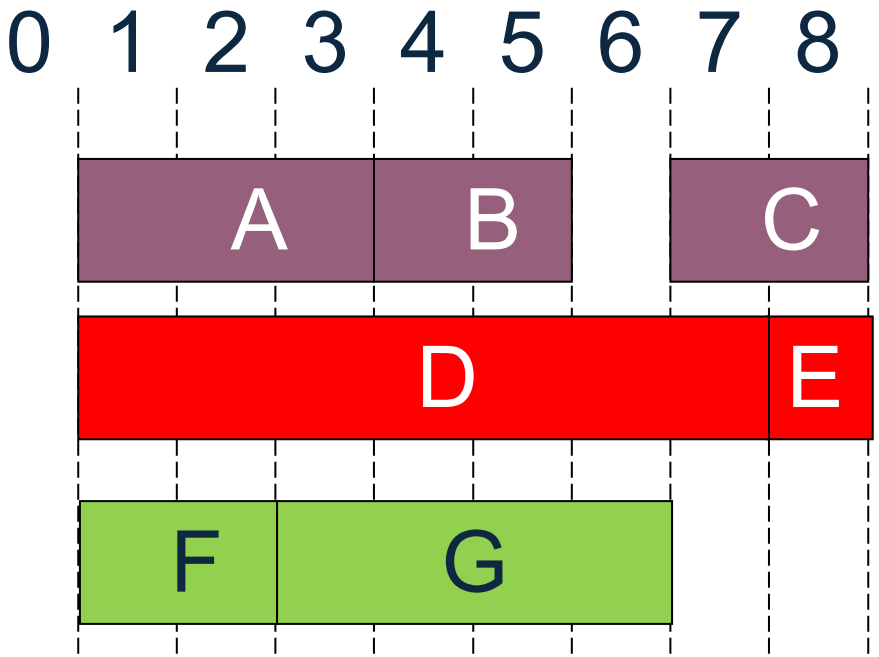


左边缘算法

Left-edge algorithm

- 输入：
  - 一组具有左边界和右边界的区间（即开始时间和结束时间）
  - 一组颜色（初始时只有一种颜色）
- 步骤：
  - 按照左边界对区间进行排序
  - 从排序后的列表中，使用第一种颜色分配互不重叠的区间
  - 当所有可能分配的区间都用完后，增加颜色计数器并重复上述过程


0 1 2 3 4 5 6 7 8
A
D
F
G
B
C
E


0 1 2 3 4 5 6 7 8
A
B
C
D
E
F
G

左边缘算法

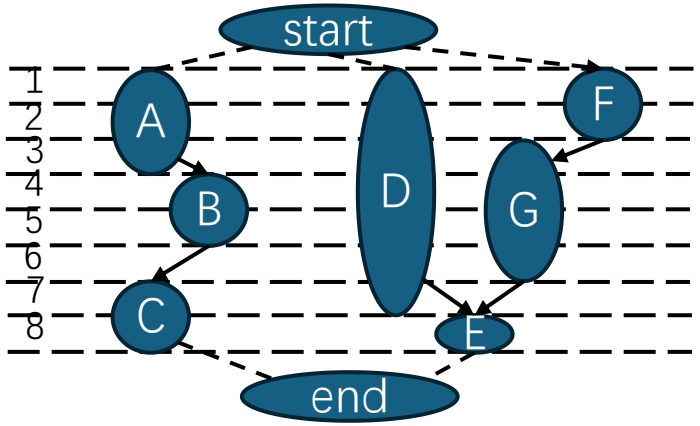
Left-edge algorithm

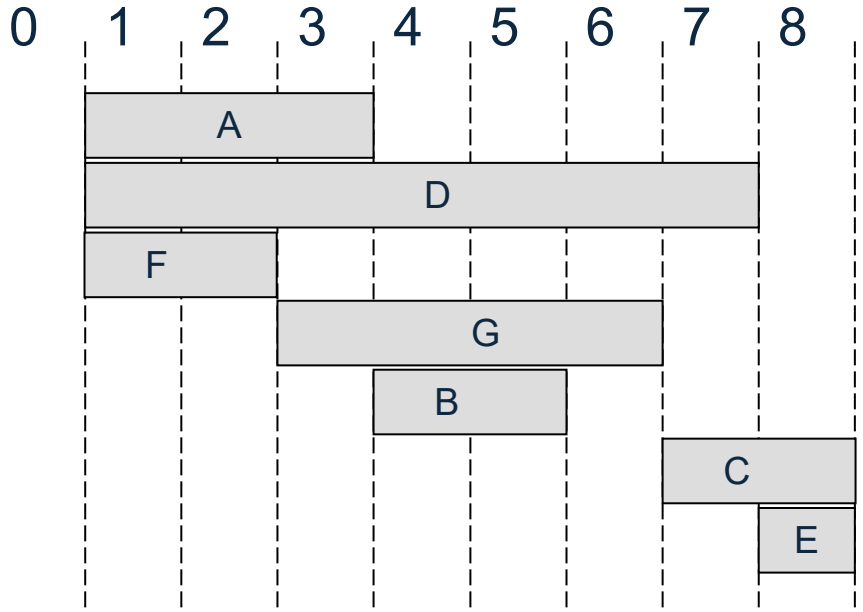
```
LEFT_EDGE(I) {
    Sort elements of I in a list L in ascending order of l_i;
    c = 0;
    while (some interval has not been colored) do {
            S = Ø;
            r = 0;
            while (  exists s ϵ L such that l_s > r ) do {
                    s = First element in the list L with l_s > r;
                    S = S U {s};
                    r = r_s;
                    Delete s from L;
            }
            c = c + 1;
            Label elements of S with color c;
    }
}
```

左边缘算法

Left-edge algorithm

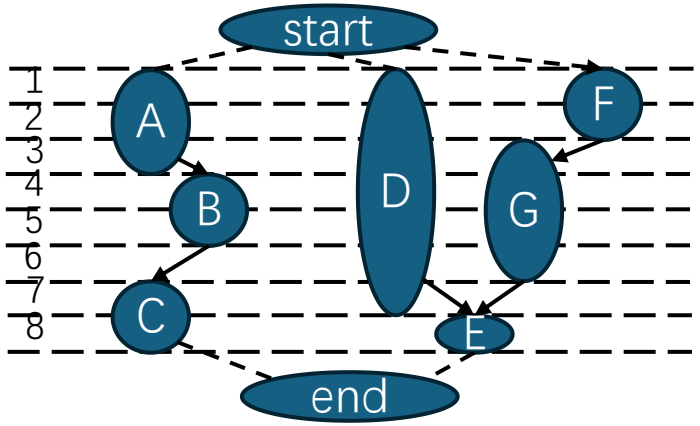
```
1 LEFT_EDGE(I) {
2    按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3    c = 0;
4    while（列表L为空）do {
5       S = 空集;
6       r = 0;
7       while（存在 s 属于 L，且 ls > r）do {
8          s = L 中第一个满足 l_s > r 的元素;
9          将 s 加入集合 S;
10         r = r_s;
11         从列表 L 中删除 s;
12      }
13      c = c + 1;
14      将集合 S 中的所有区间标记为颜色 c;
15   }
16}
```

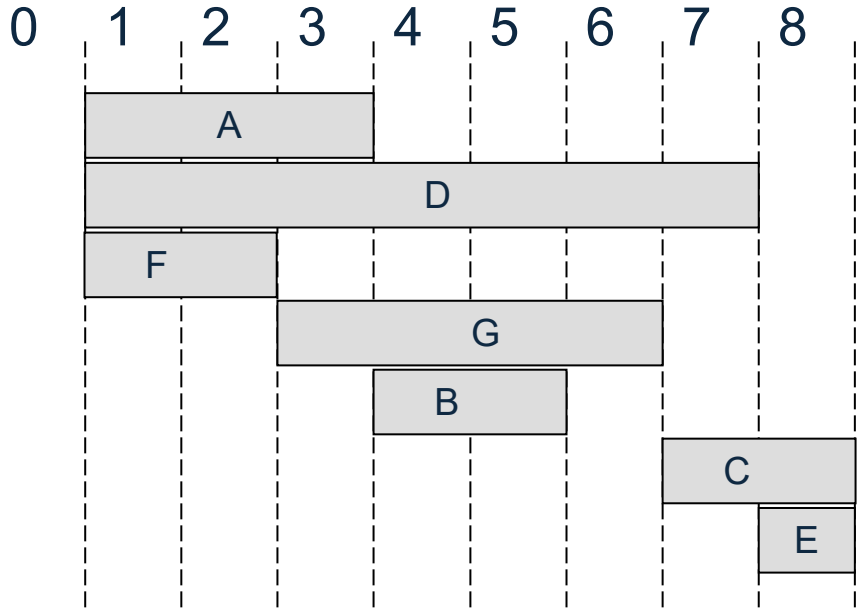





```
1 LEFT_EDGE(I) {
2    按区间的左端点 I_i 升序对集合 I 中的元素排序，得到列表 L;
3    c = 0;
4    while（列表L为空）do {
5       S = 空集;
6       r = 0;
7       while（存在 s 属于 L，且 Is > r）do {
8          s = L 中第一个满足 I_s > r 的元素;
9          将 s 加入集合 S;
10         r = r_s;
11         从列表 L 中删除 s;
12      }
13      c = c + 1;
14      将集合 S 中的所有区间标记为颜色 c;
15   }
16}
```

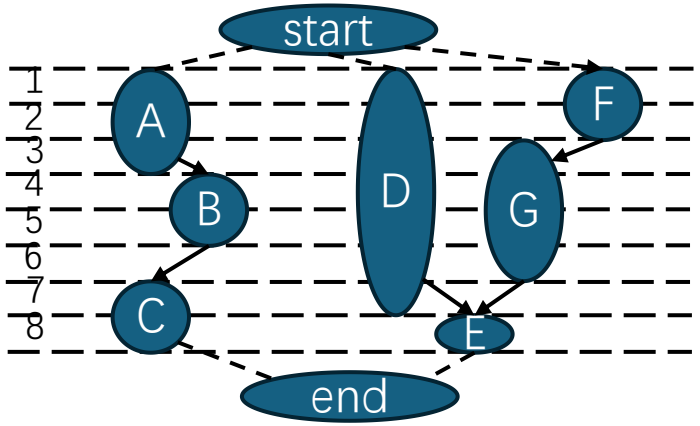
L	{A,D,F,G,B,C,E}
c	0
S	
s	
r	

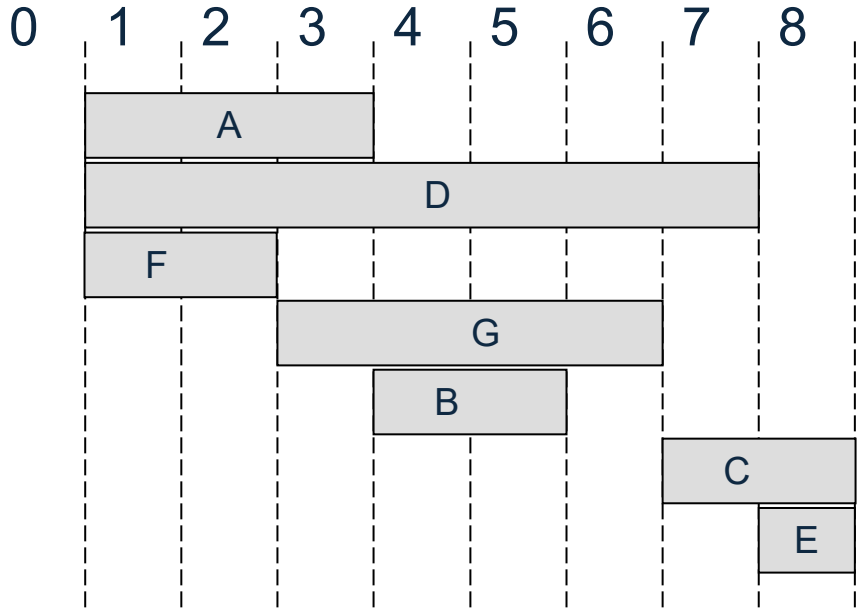





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

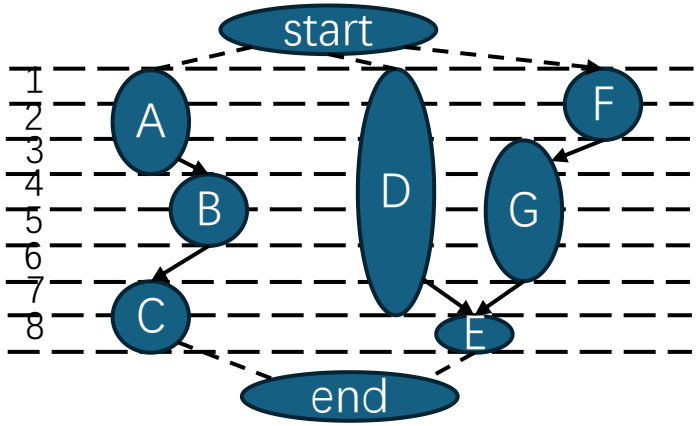
L	{A,D,F,G,B,C,E}
c	0
S	{}
s	
r	0

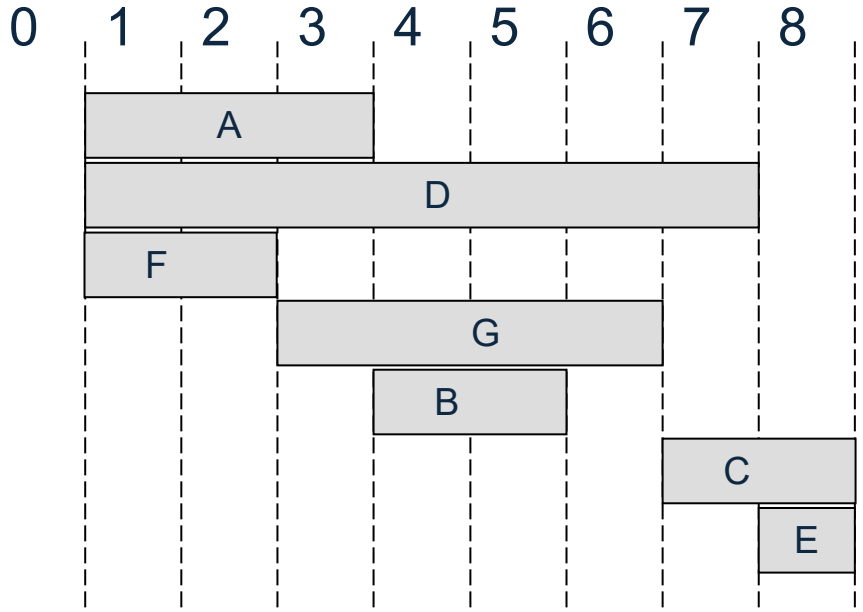





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

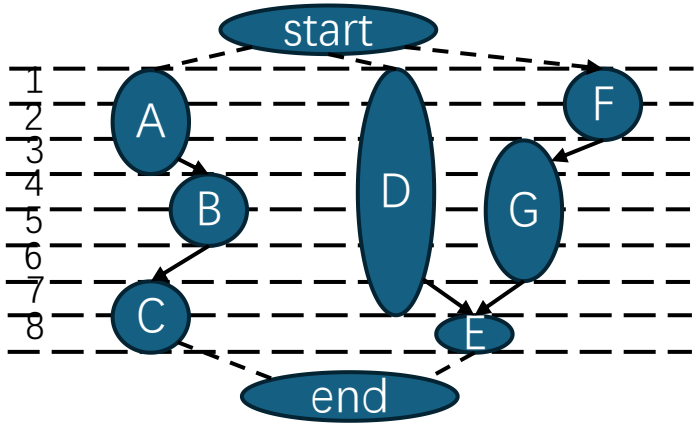
L	{A,D,F,G,B,C,E}
c	0
S	{}
s	A
r	0

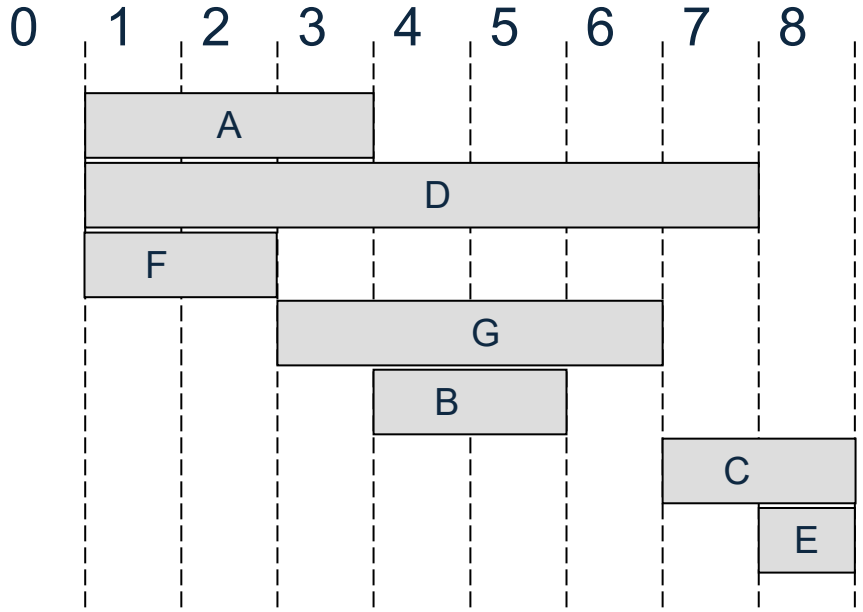





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

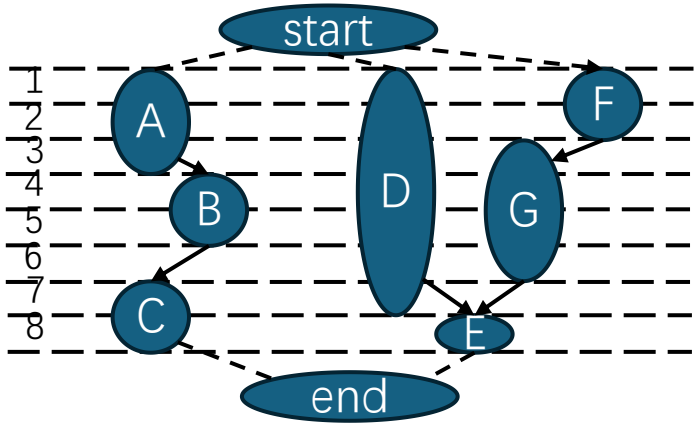
L	{A,D,F,G,B,C,E}
c	0
S	{A}
s	A
r	0

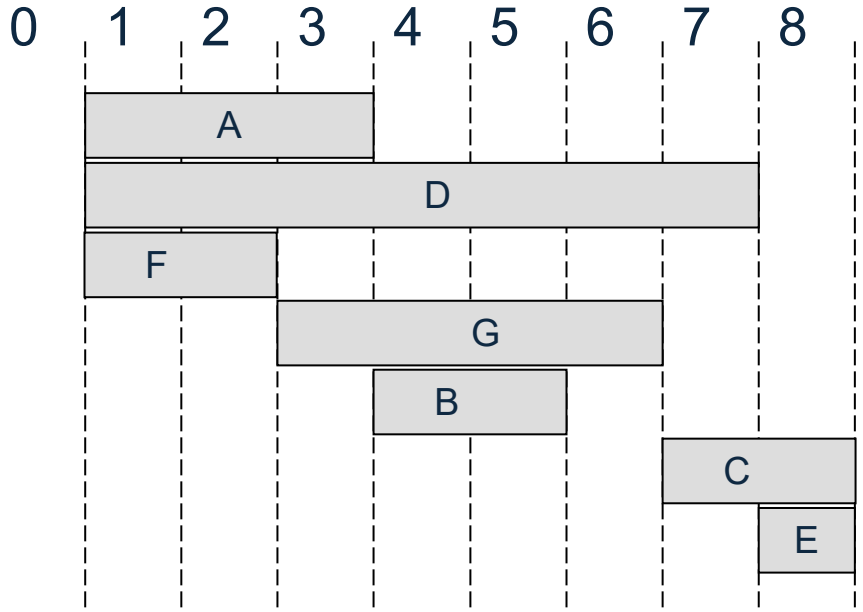





```
1 LEFT_EDGE(I) {
2    按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3    c = 0;
4    while（列表L为空）do {
5       S = 空集;
6       r = 0;
7       while（存在 s 属于 L，且 ls > r）do {
8          s = L 中第一个满足 l_s > r 的元素;
9          将 s 加入集合 S;
10         r = r_s;
11         从列表 L 中删除 s;
12      }
13      c = c + 1;
14      将集合 S 中的所有区间标记为颜色 c;
15   }
16}
```

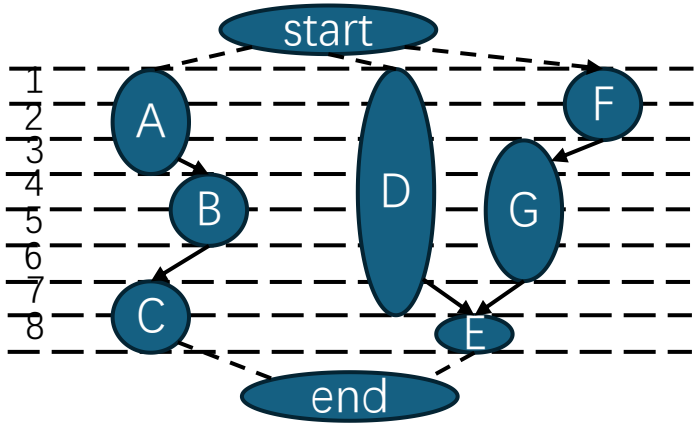
L	{A,D,F,G,B,C,E}
c	0
S	{A}
s	A
r	3

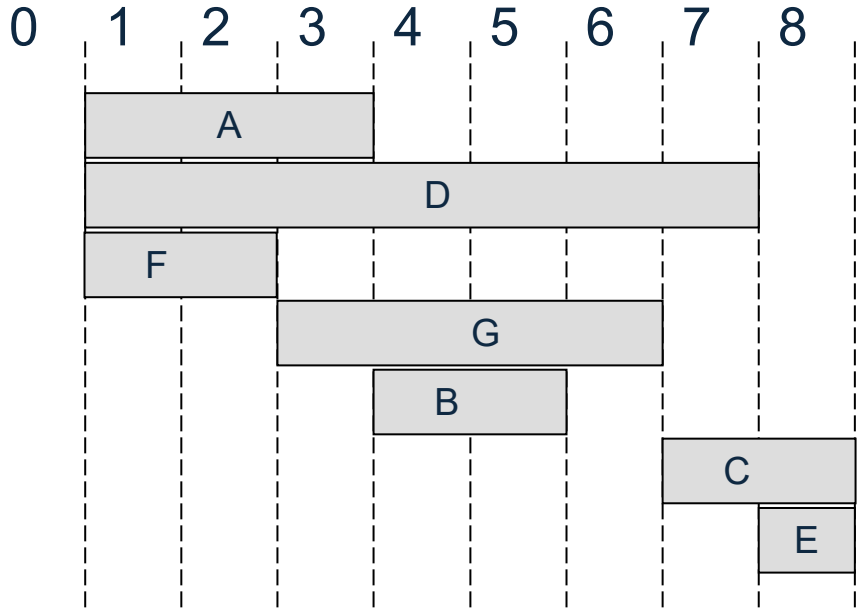





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

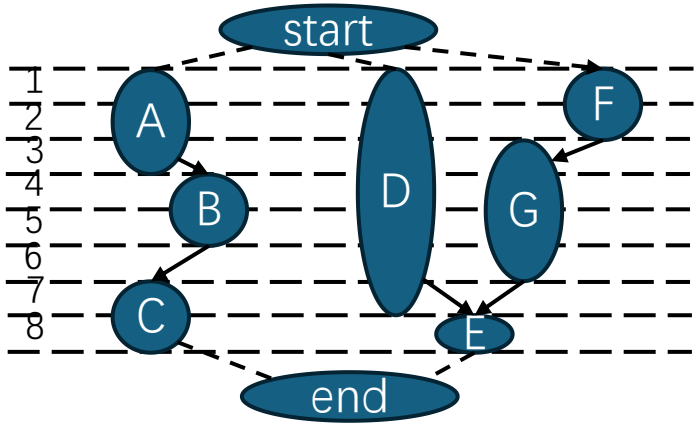
L	{D,F,G,B,C,E}
c	0
S	{A}
s	A
r	3

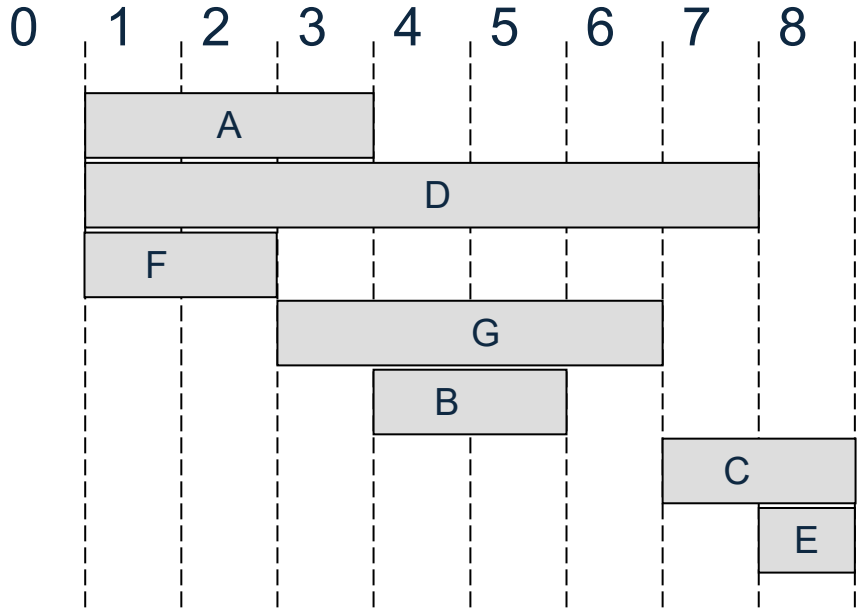





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

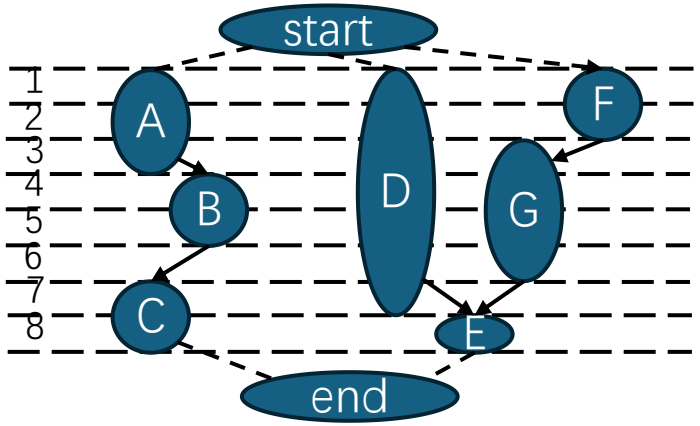
L	{D,F,G,B,C,E}
c	0
S	{A}
s	B
r	3

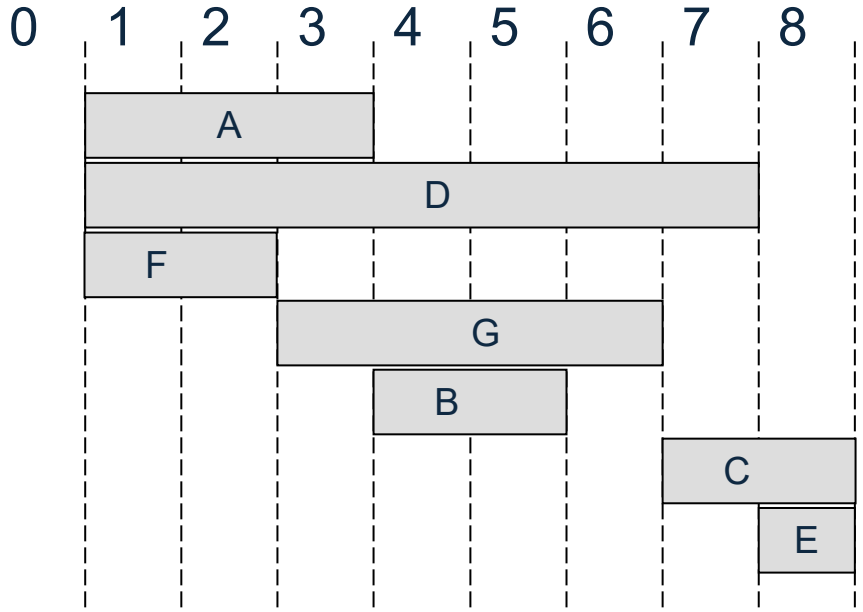





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

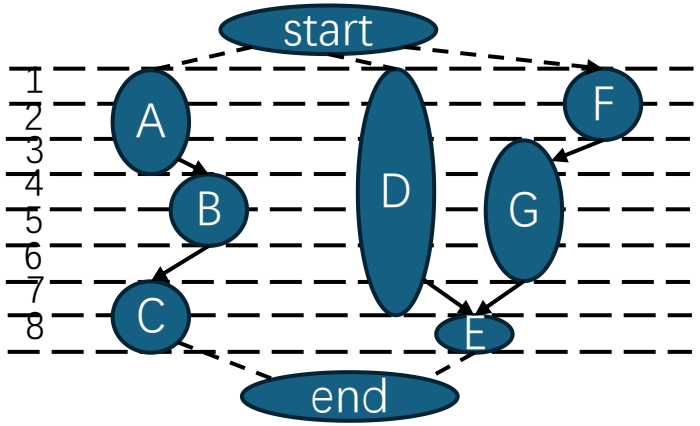
L	{D,F,G,B,C,E}
c	0
S	{A,B}
s	B
r	3

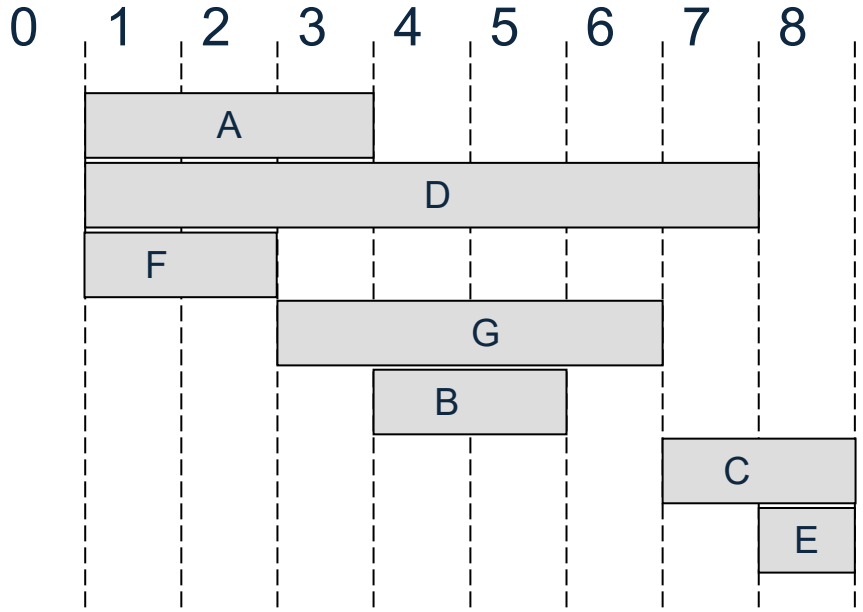





```
1 LEFT_EDGE(I) {
2    按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3    c = 0;
4    while（列表L为空）do {
5       S = 空集;
6       r = 0;
7       while（存在 s 属于 L，且 ls > r）do {
8          s = L 中第一个满足 l_s > r 的元素;
9          将 s 加入集合 S;
10         r = r_s;
11         从列表 L 中删除 s;
12      }
13      c = c + 1;
14      将集合 S 中的所有区间标记为颜色 c;
15   }
16}
```

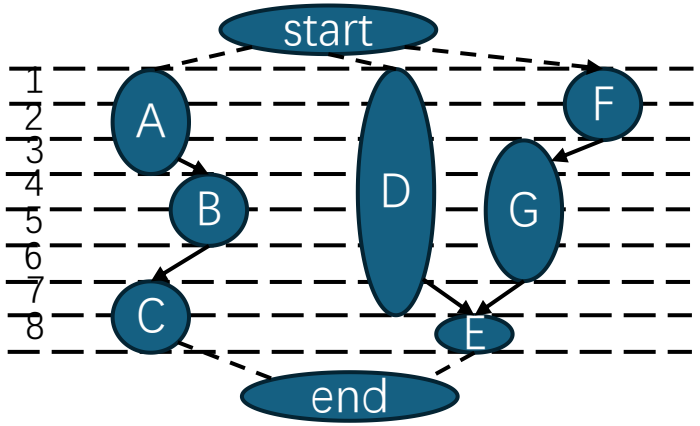
L	{D,F,G,B,C,E}
c	0
S	{A,B}
s	B
r	5

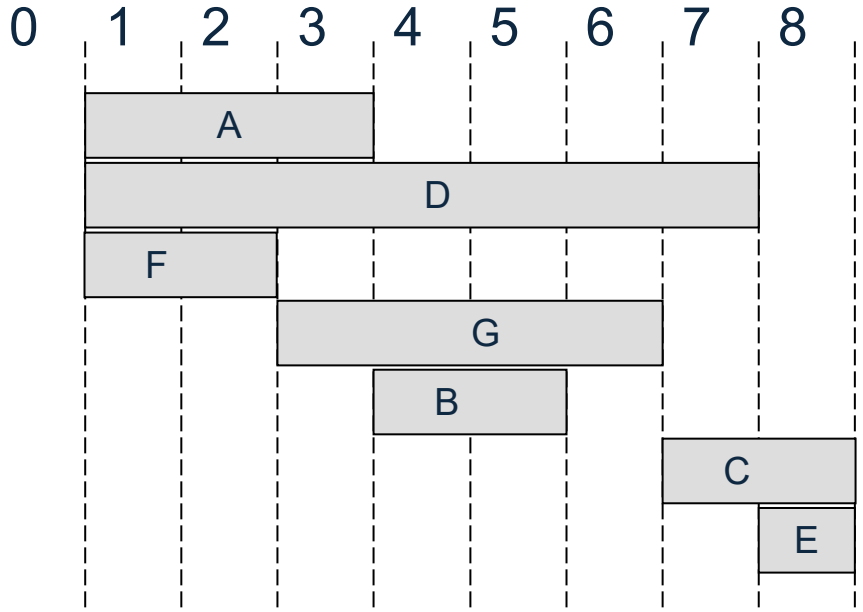





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

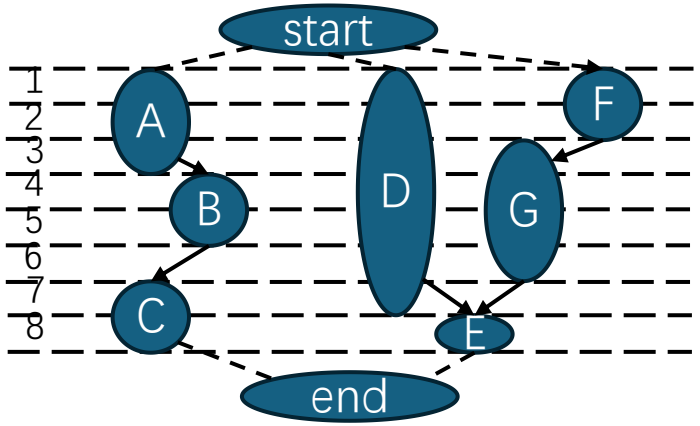
L	{D,F,G,C,E}
c	0
S	{A,B}
s	B
r	5

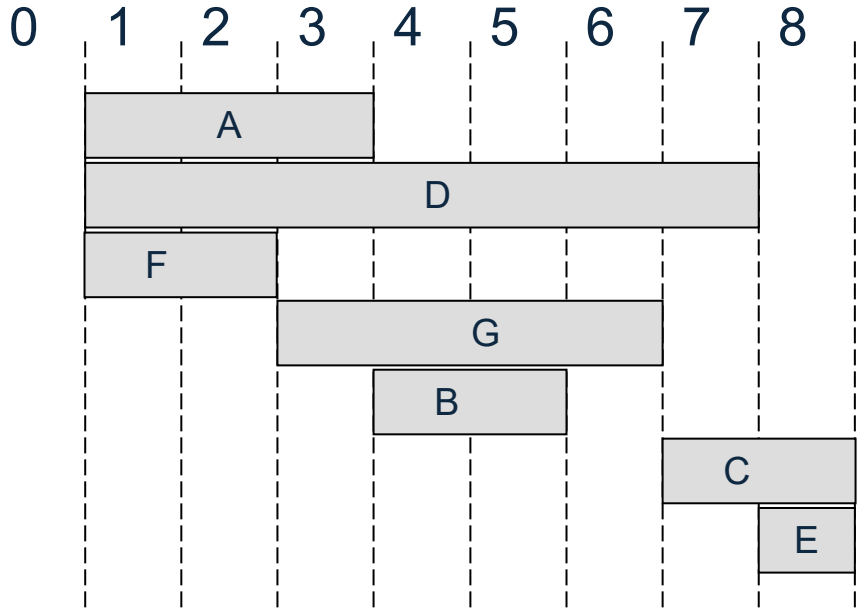





```
1 LEFT_EDGE(I) {
2     按区间的左端点 I_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 Is > r）do {
8             s = L 中第一个满足 I_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

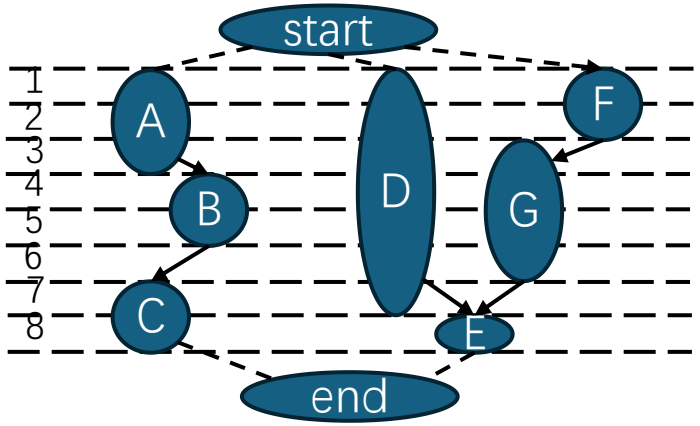
L	{D,F,G,C,E}
c	0
S	{A,B,C}
s	C
r	8

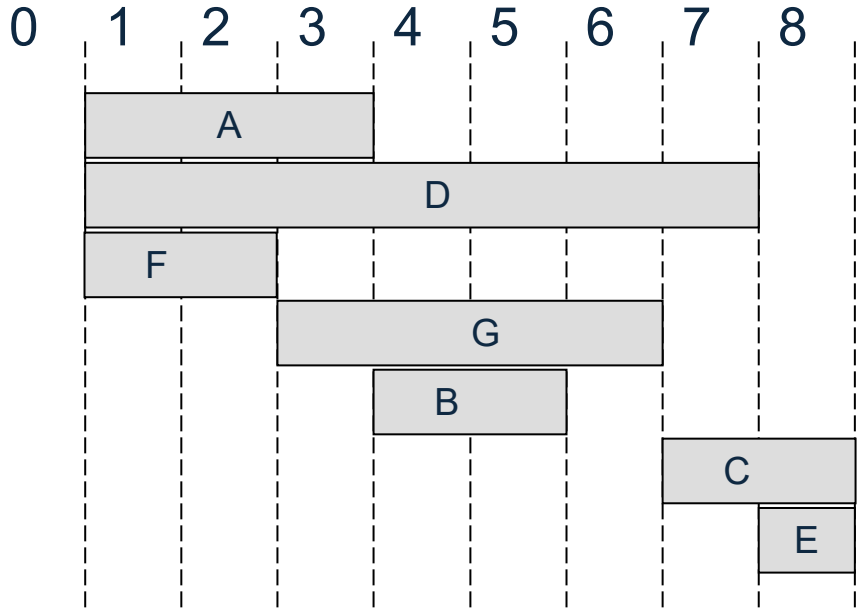





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

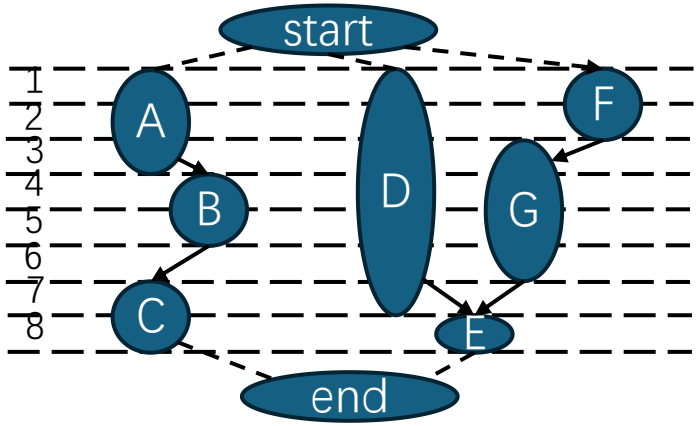
L	{D,F,G,E}
c	0
S	{A,B,C}
s	C
r	8

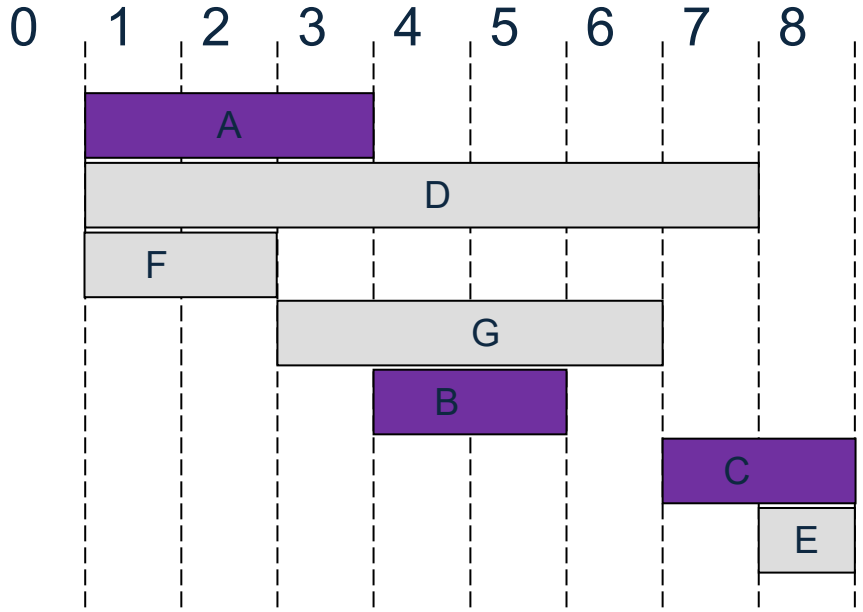





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

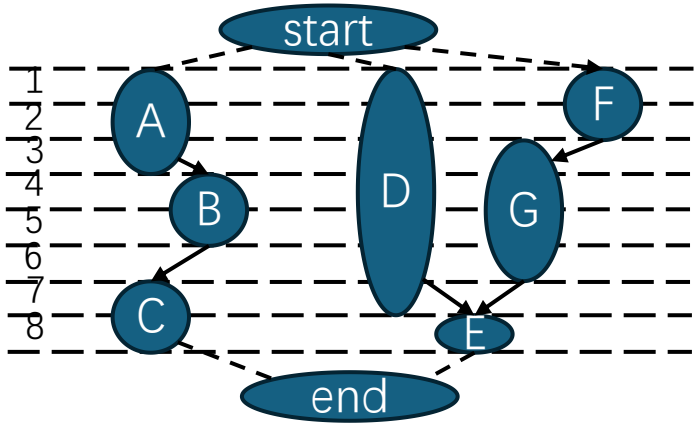
L	{D,F,G,E}
c	0
S	{A,B,C}
s	C
r	8

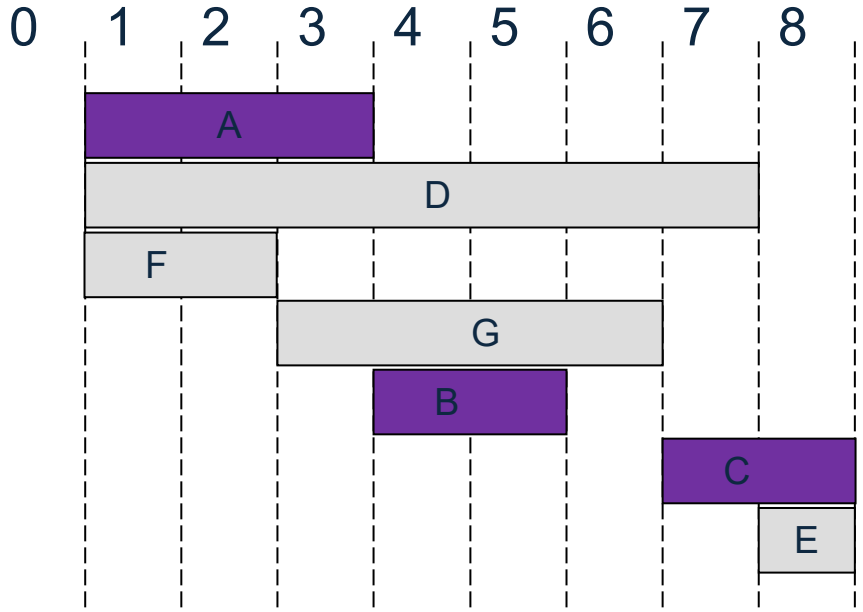





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

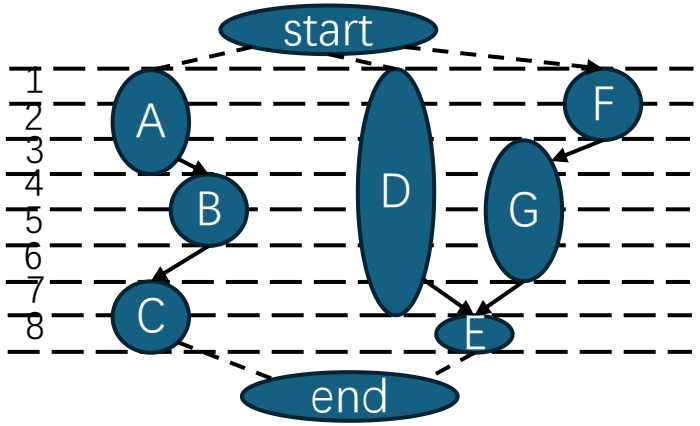
L	{D,F,G,E}
c	1（假设是紫色）
S	{A,B,C}
s	C
r	8

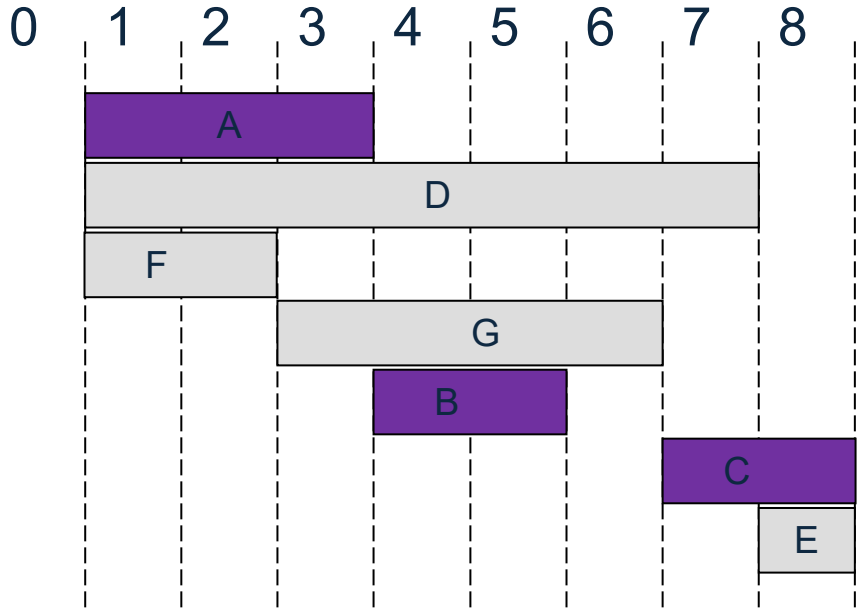





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

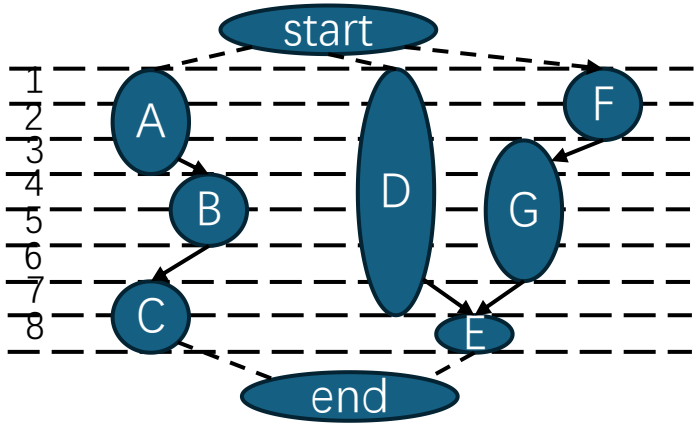
L	{D,F,G,E}
c	1（假设是紫色）
S	{}
s	C
r	0

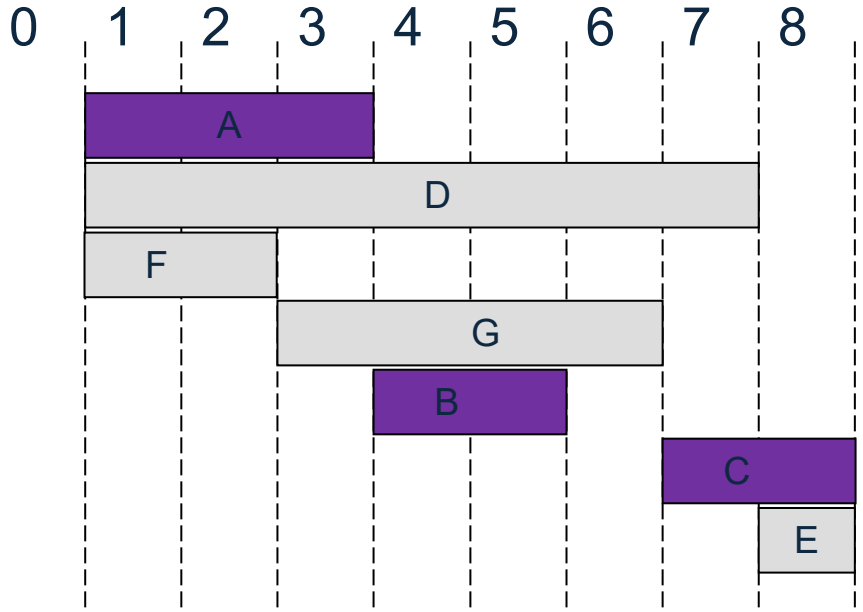





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

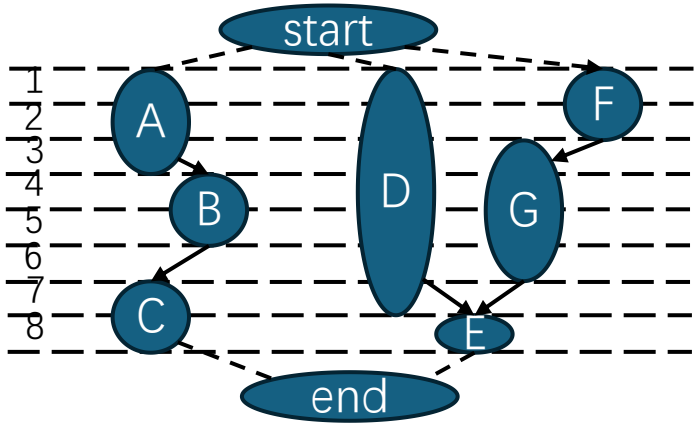
L	{D,F,G,E}
c	1（假设是紫色）
S	{D}
s	D
r	7

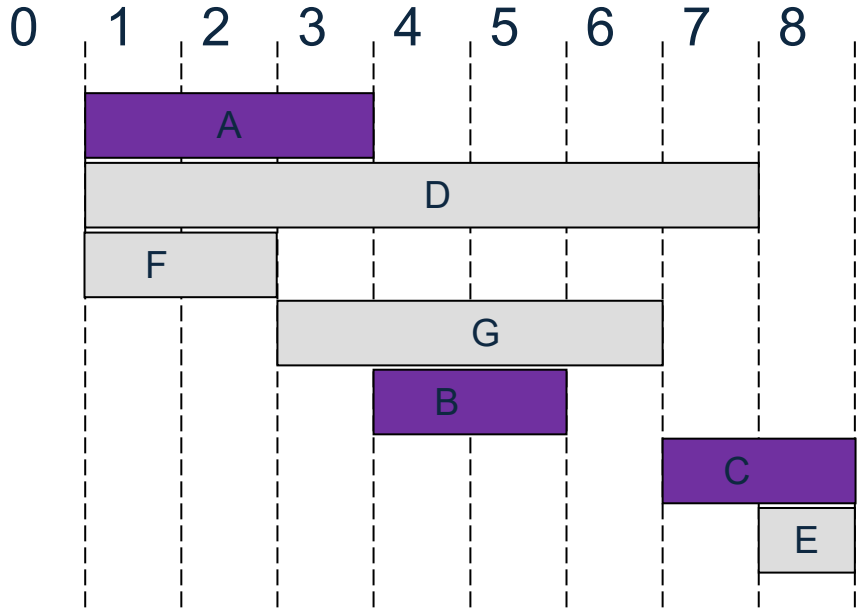





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

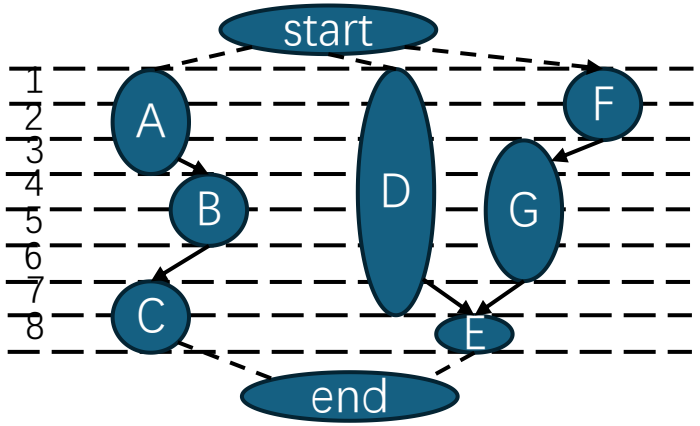
L	{F,G,E}
c	1（假设是紫色）
S	{D}
s	D
r	7

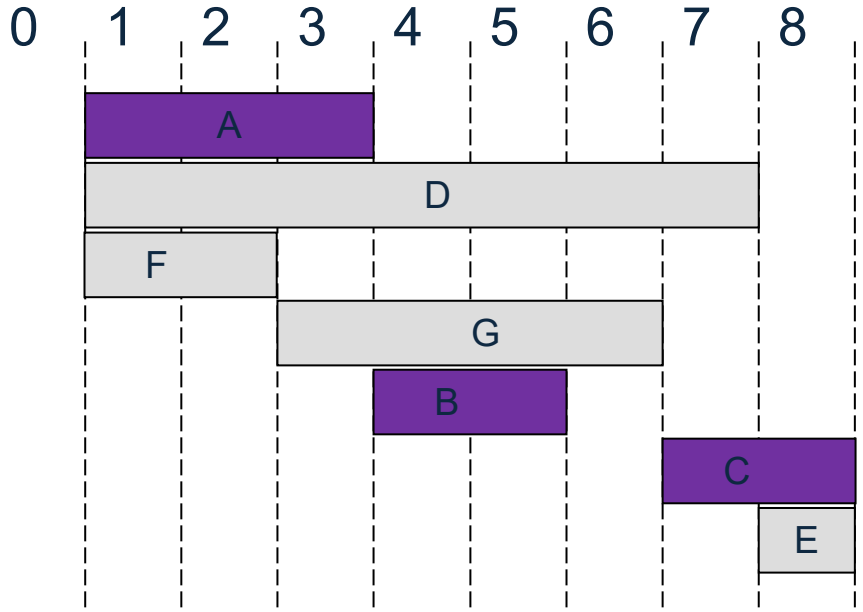





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

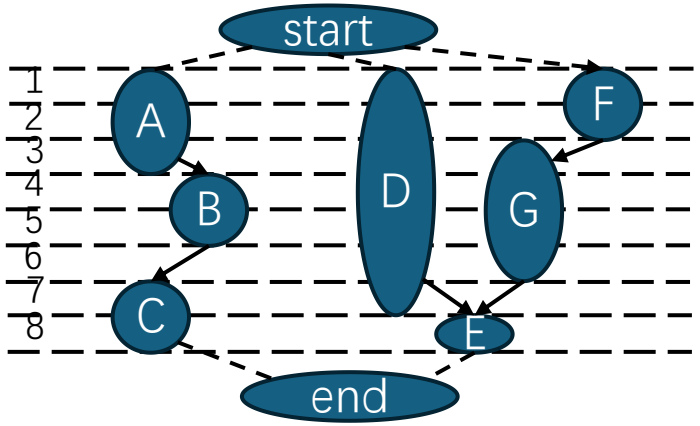
L	{F,G,E}
c	1（假设是紫色）
S	{D,E}
s	E
r	8

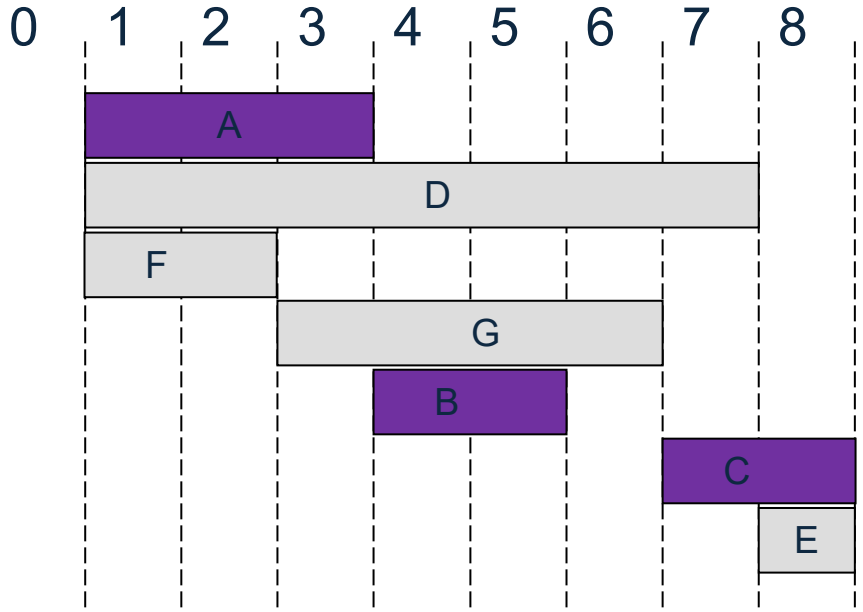





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

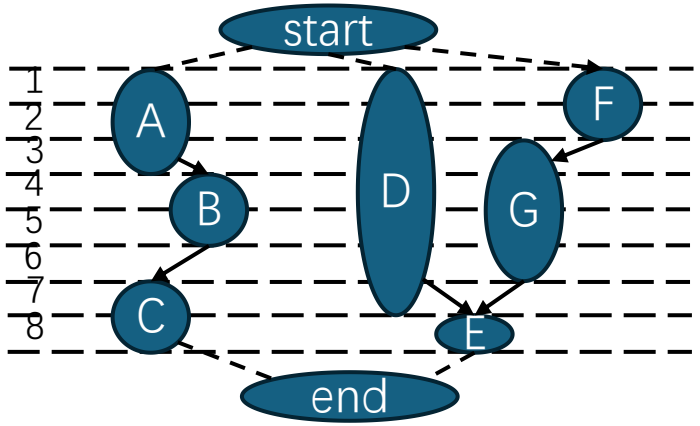
L	{F,G}
c	1（假设是紫色）
S	{D,E}
s	E
r	8

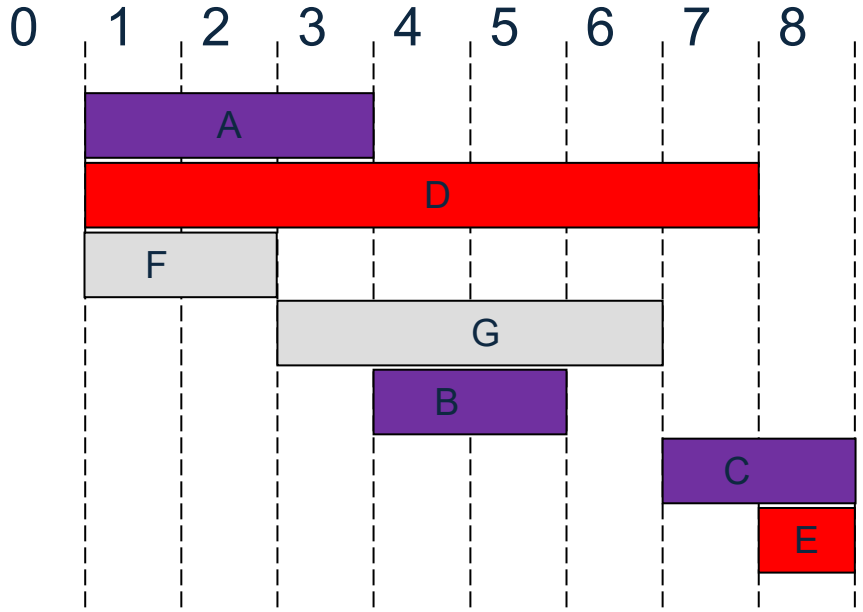





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

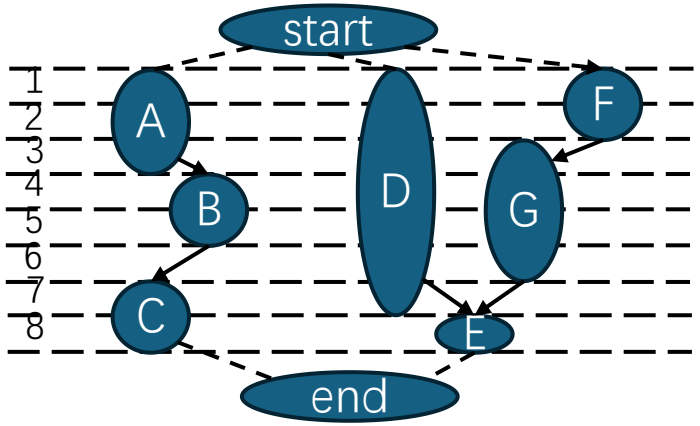
L	{F,G}
c	1（假设是紫色）
S	{D,E}
s	E
r	8

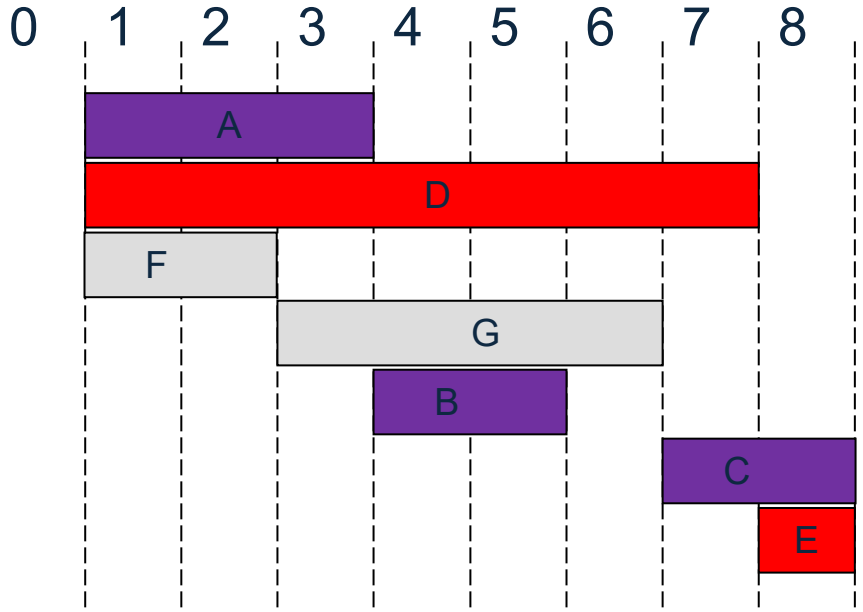





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

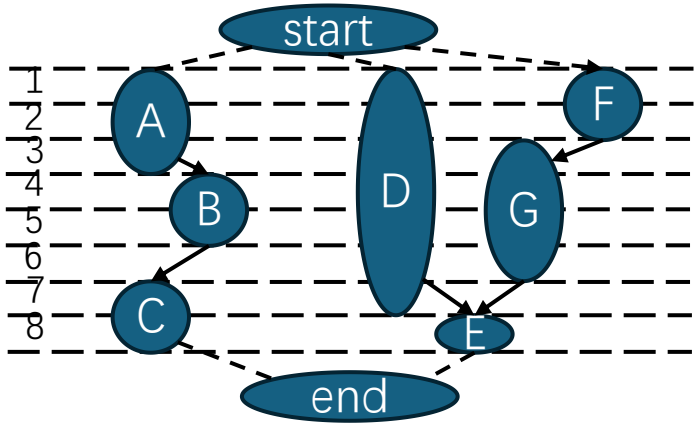
L	{F,G}
c	2（假设是红色）
S	{D,E}
s	E
r	8

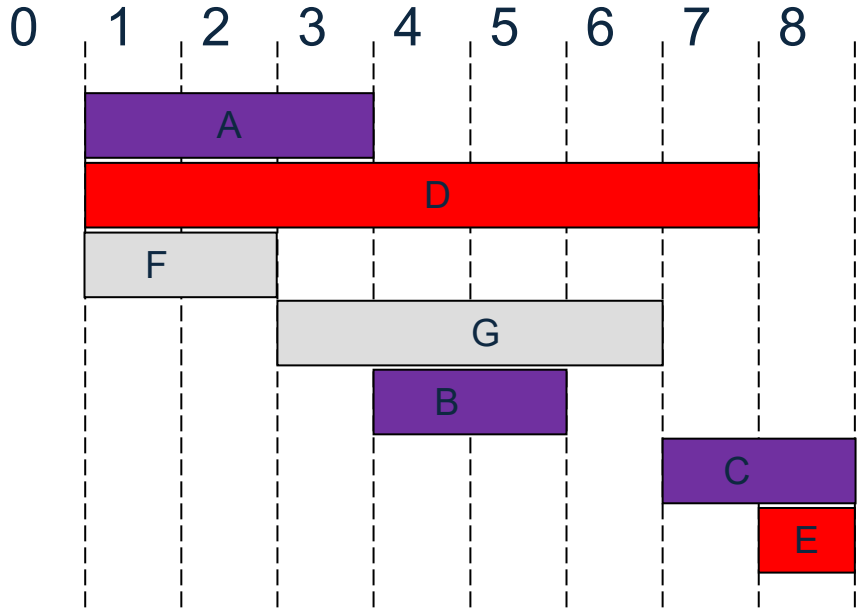





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

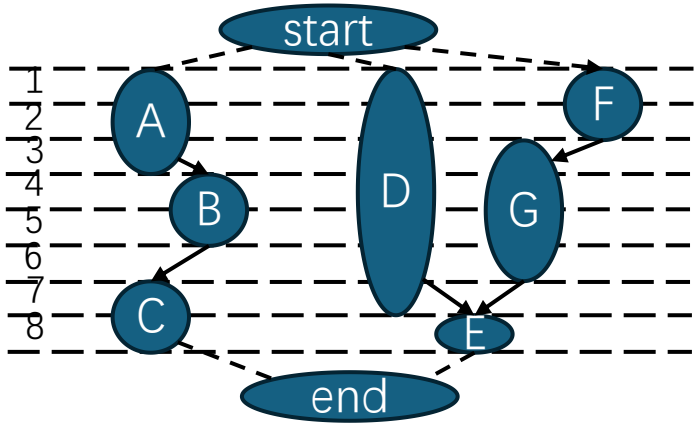
L	{F,G}
c	2（假设是红色）
S	{}
s	E
r	0

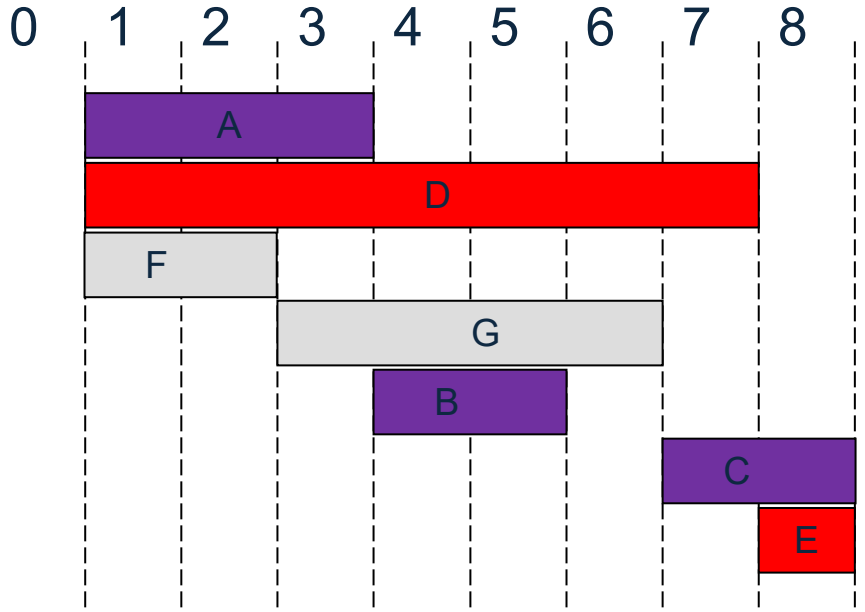





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

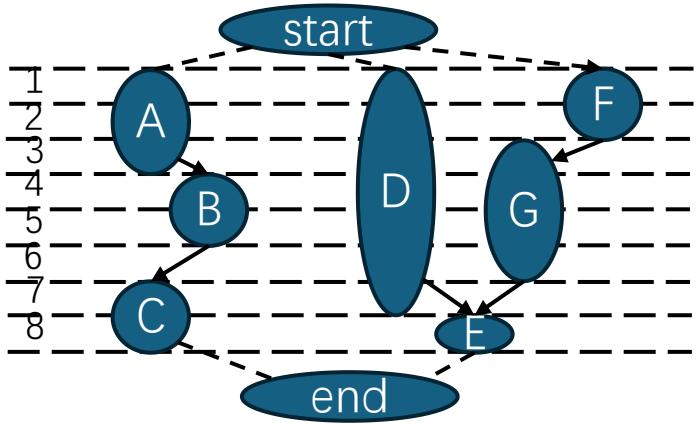
L	{F,G}
c	2（假设是红色）
S	{F}
s	F
r	2

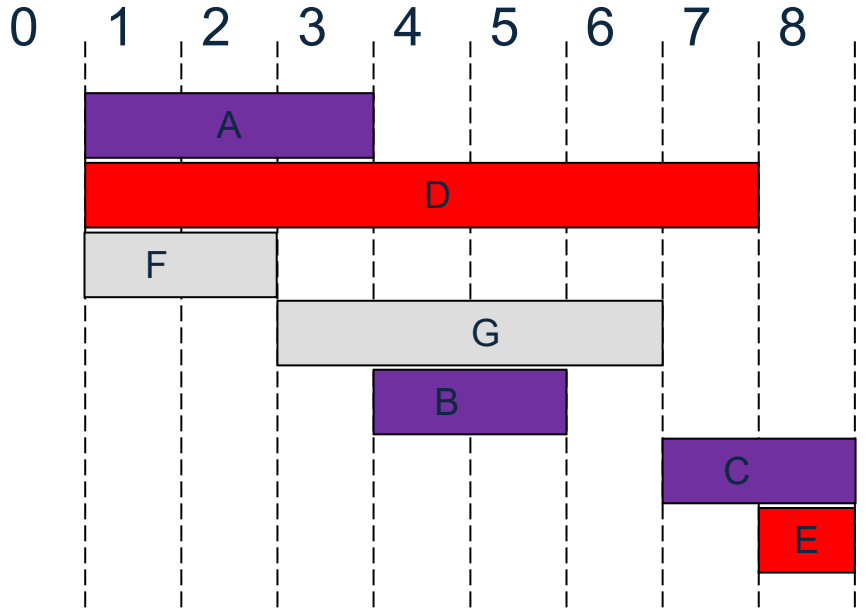





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

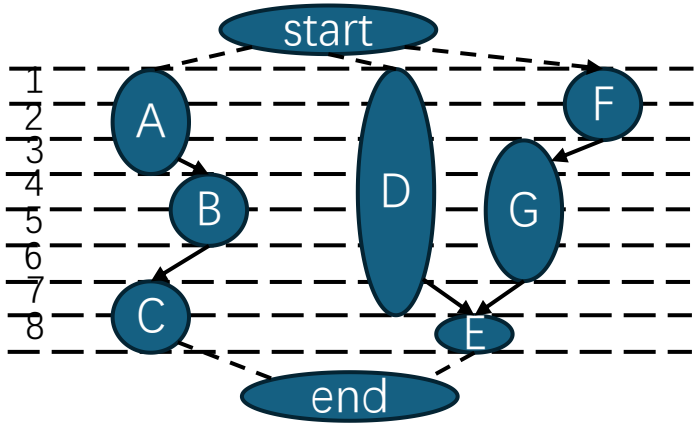
L	{G}
c	2（假设是红色）
S	{F}
s	F
r	2

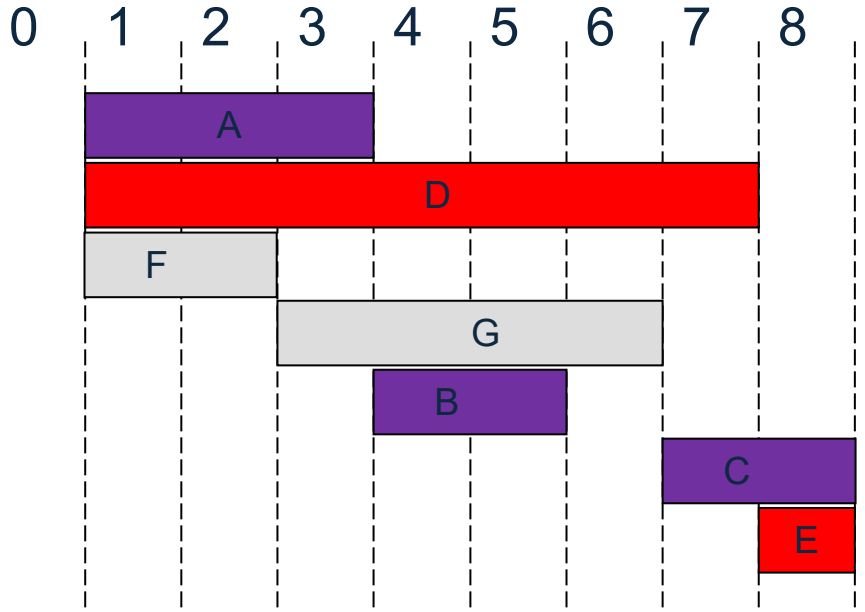





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

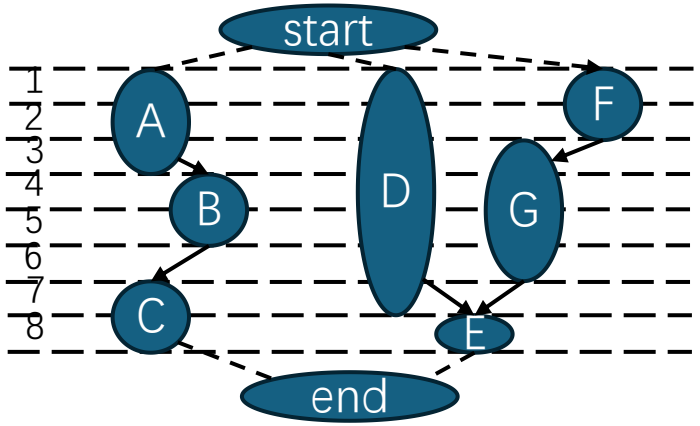
L	{G}
c	2（假设是红色）
S	{F,G}
s	G
r	6

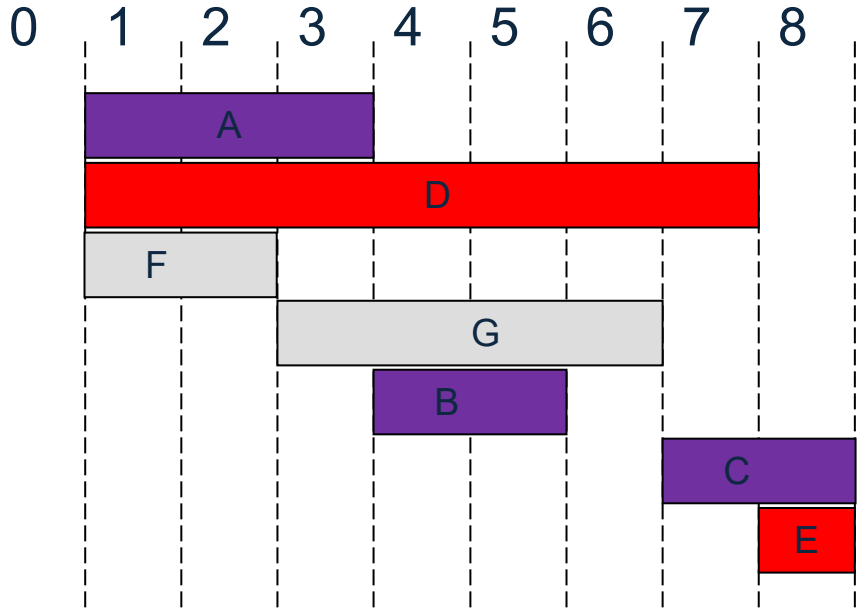





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

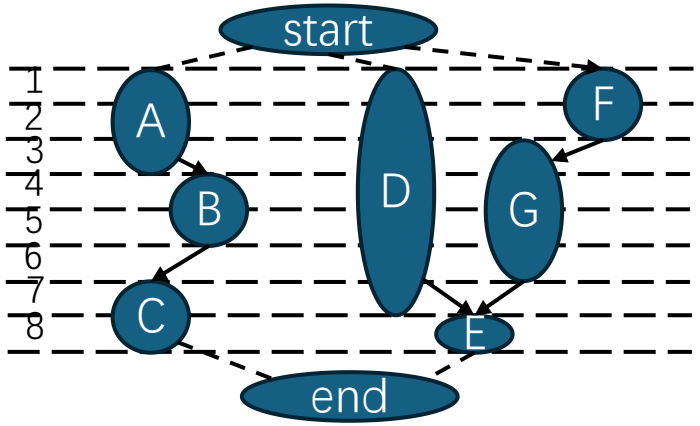
L	{}
c	2（假设是红色）
S	{F,G}
s	G
r	6

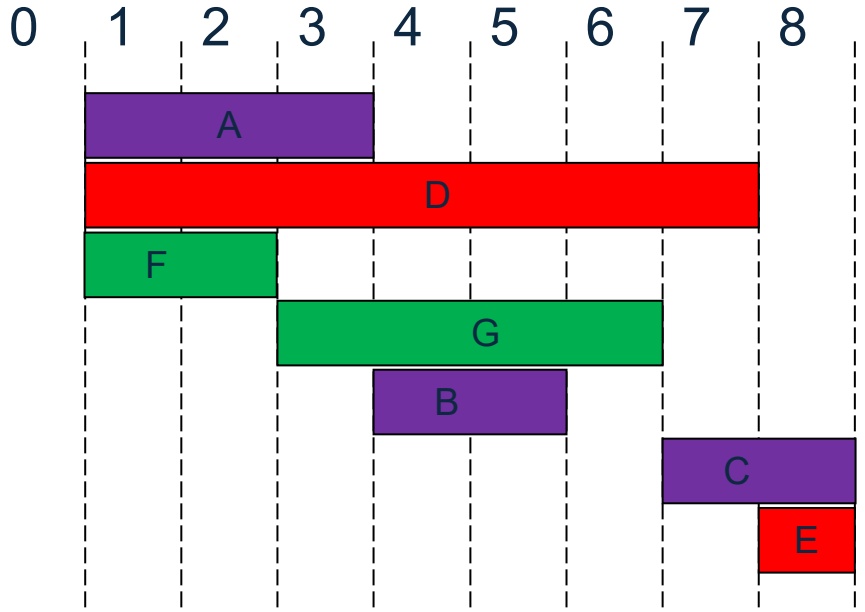





```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

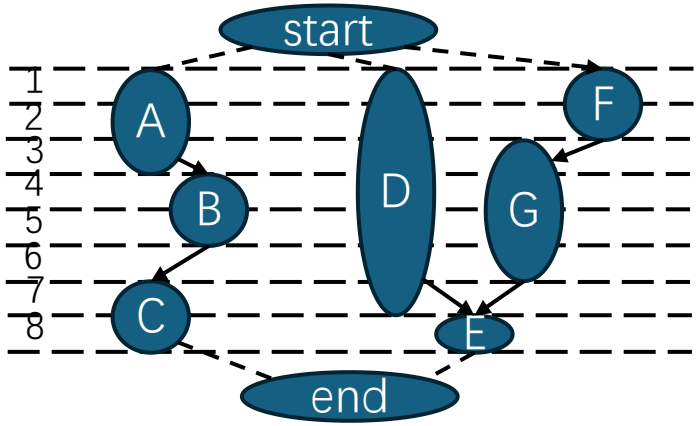
L	{}
c	2（假设是红色）
S	{F,G}
s	G
r	6

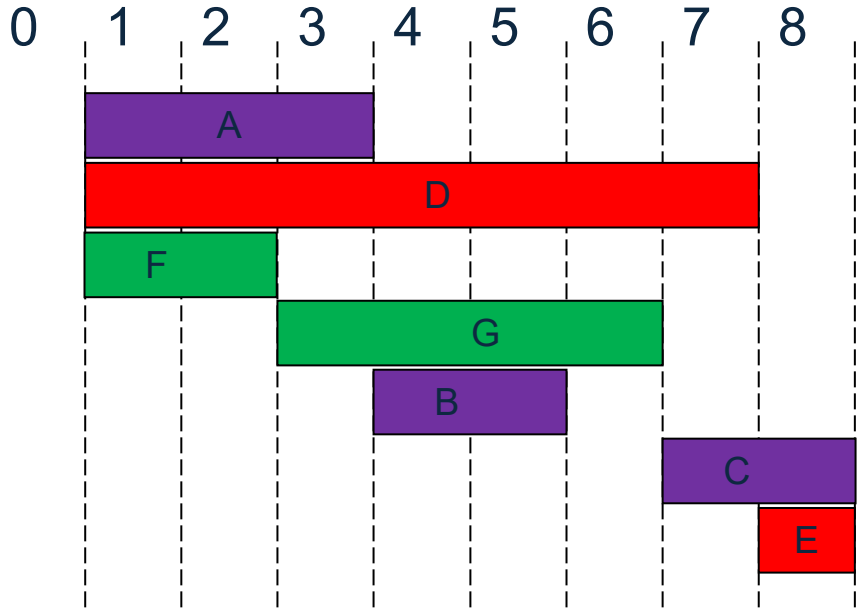





```
1 LEFT_EDGE(I) {
2     按区间的左端点 I_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 Is > r）do {
8             s = L 中第一个满足 I_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

L	{}
c	3（假设是绿色）
S	{F,G}
s	G
r	6






```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

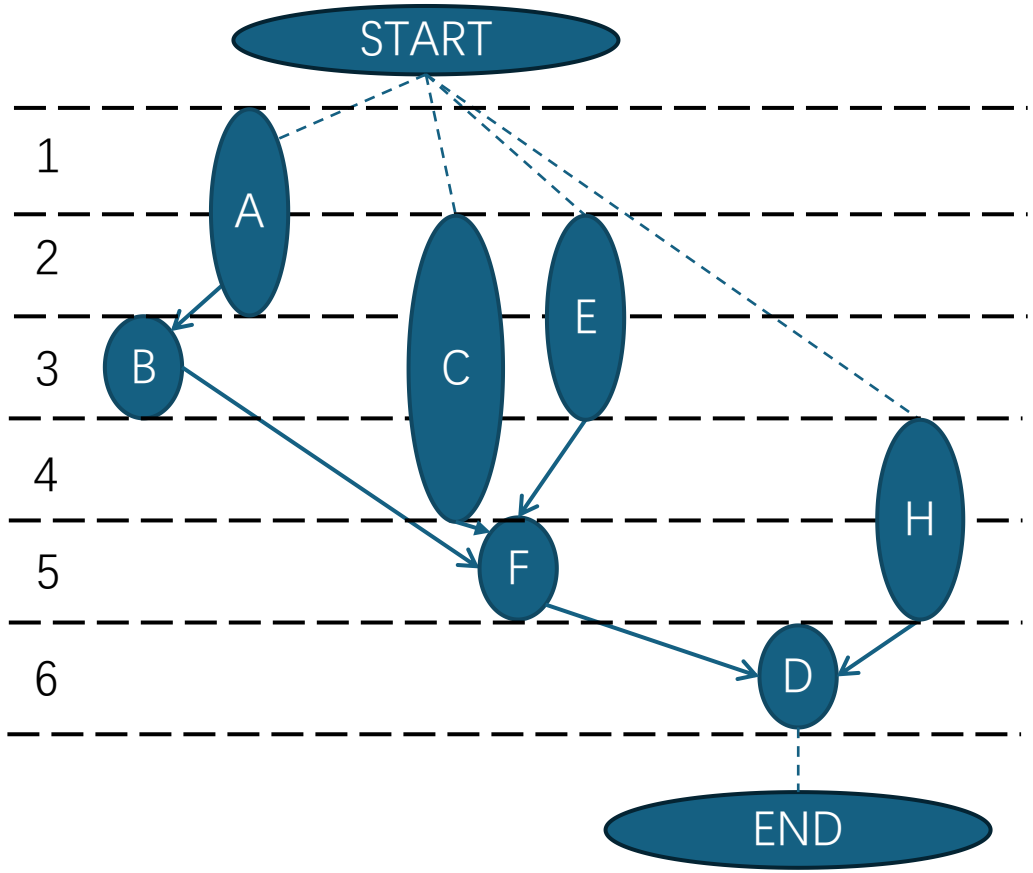
L	{}
c	3（假设是绿色）
S	{F,G}
s	G
r	6

随堂作业

in-class assignment

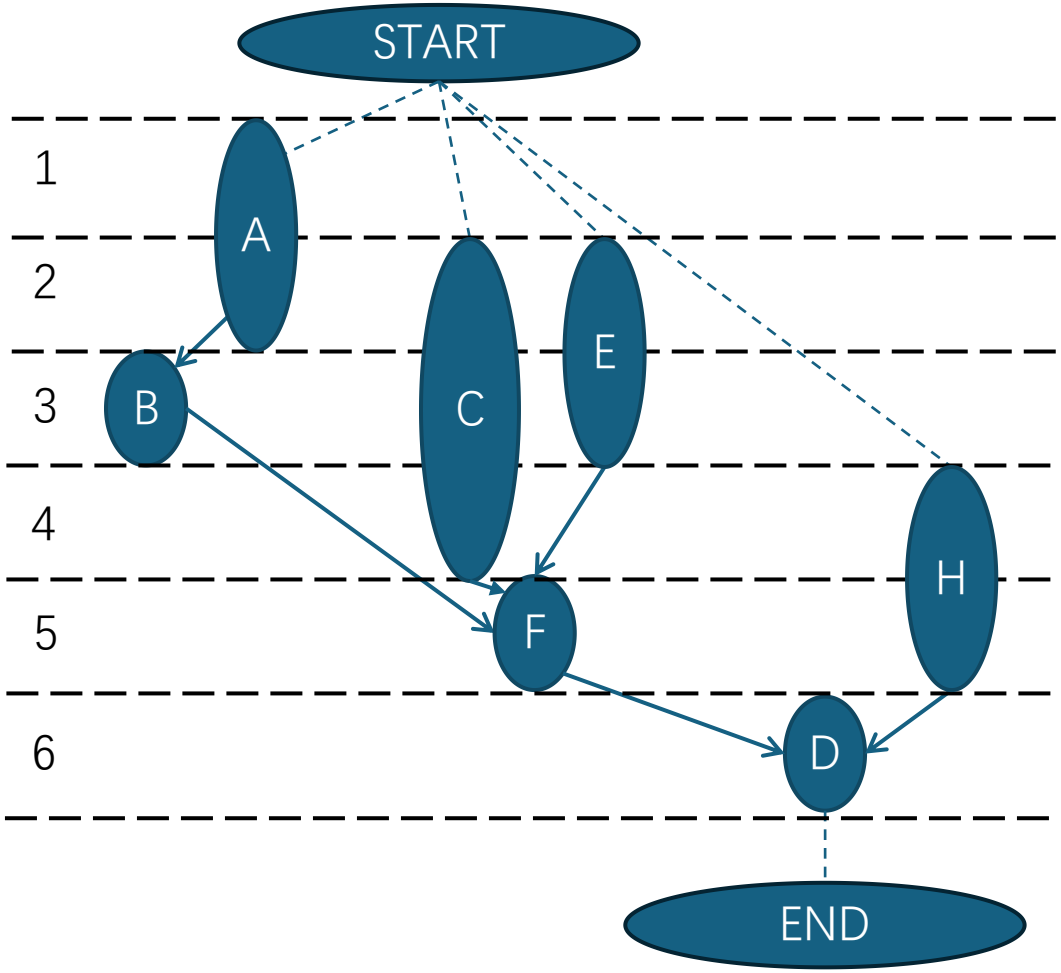
```
1 LEFT_EDGE(I) {
2     按区间的左端点 l_i 升序对集合 I 中的元素排序，得到列表 L;
3     c = 0;
4     while（列表L为空）do {
5         S = 空集;
6         r = 0;
7         while（存在 s 属于 L，且 ls > r）do {
8             s = L 中第一个满足 l_s > r 的元素;
9             将 s 加入集合 S;
10            r = r_s;
11            从列表 L 中删除 s;
12        }
13        c = c + 1;
14        将集合 S 中的所有区间标记为颜色 c;
15    }
16}
```

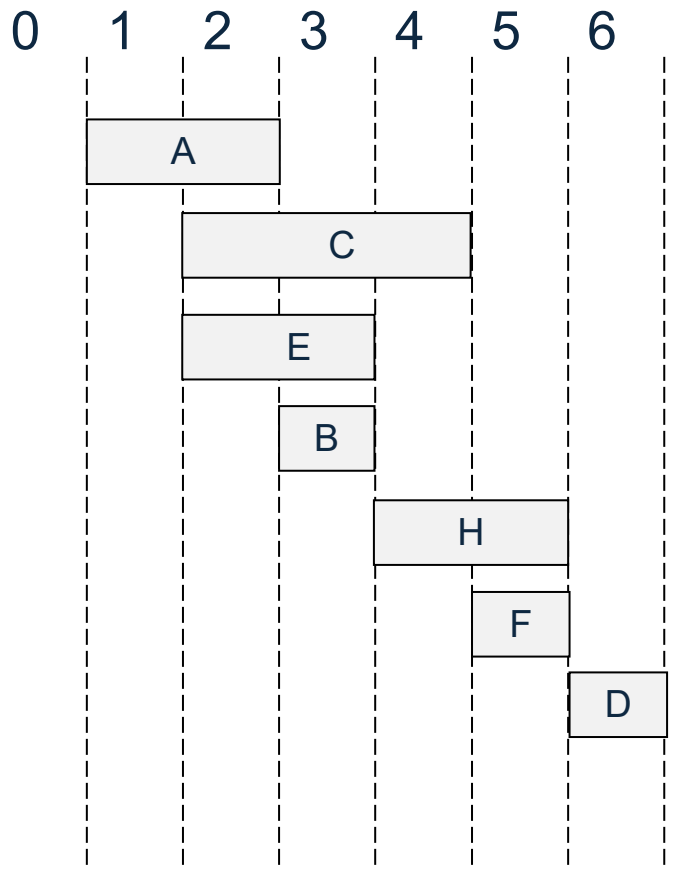
请写出用left-edge演算法得到的染色数是多少，每个独立集包含哪几个元素



随堂作业

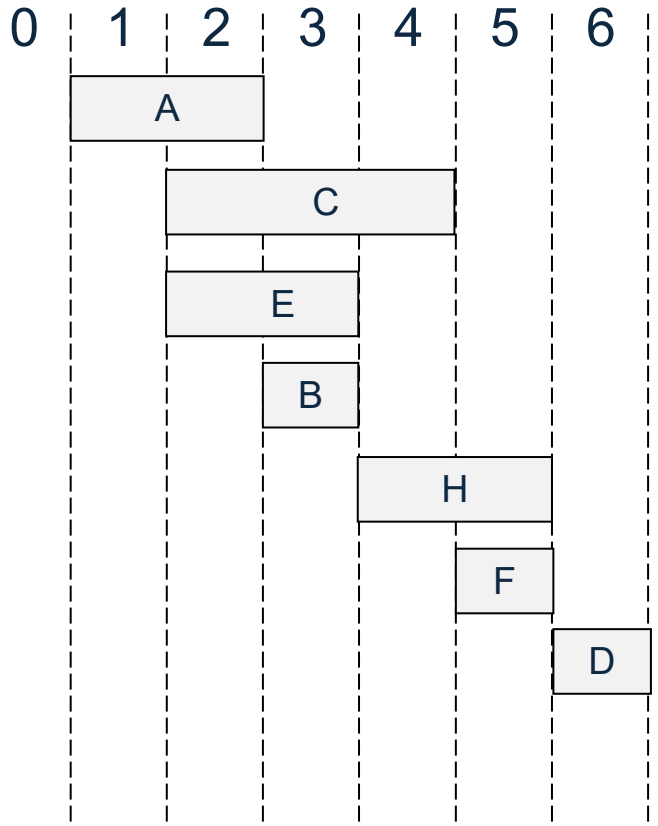
in-class assignment

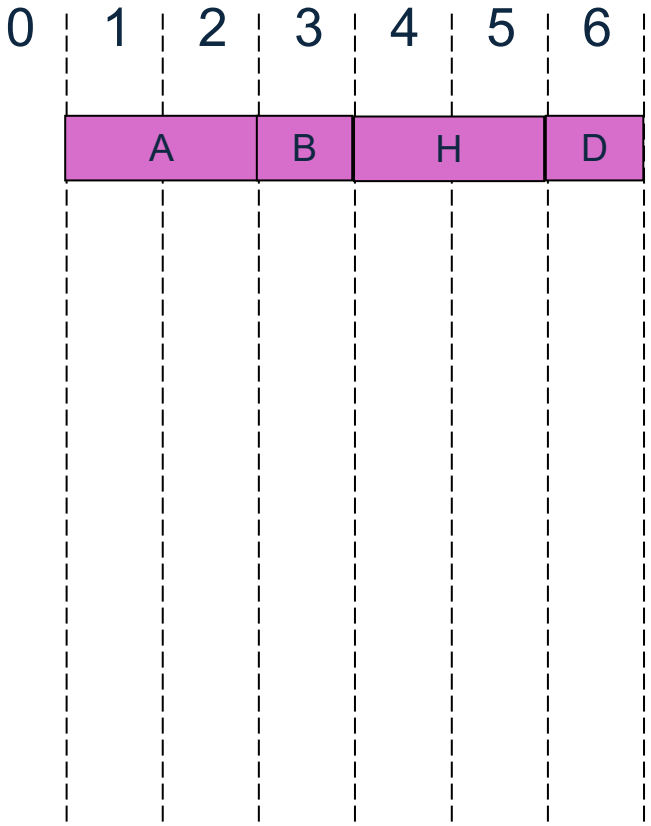




随堂作业

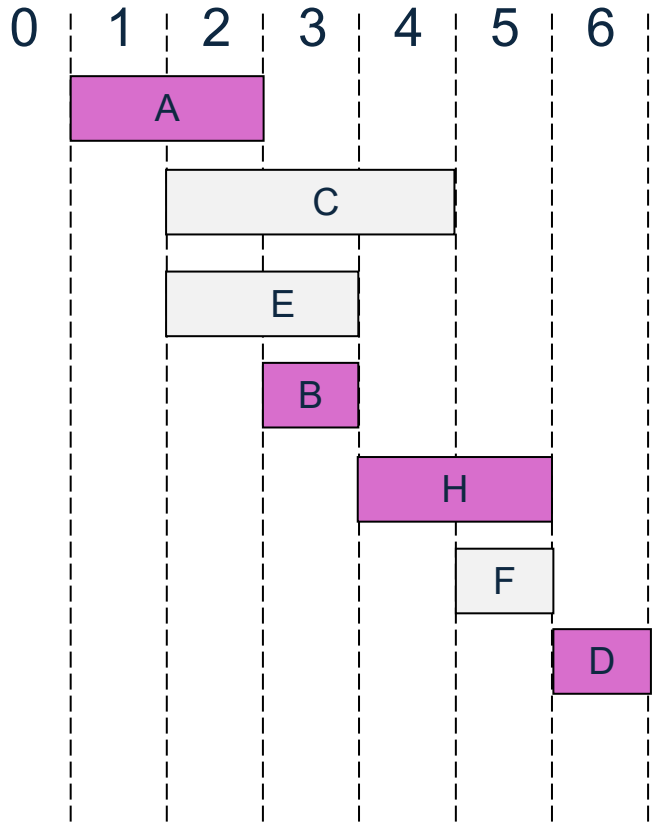
in-class assignment

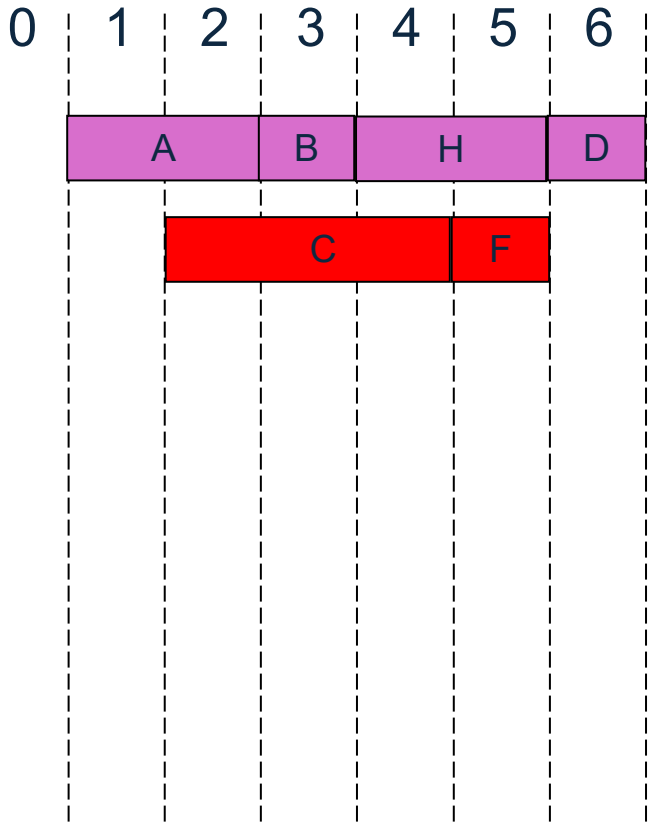




随堂作业

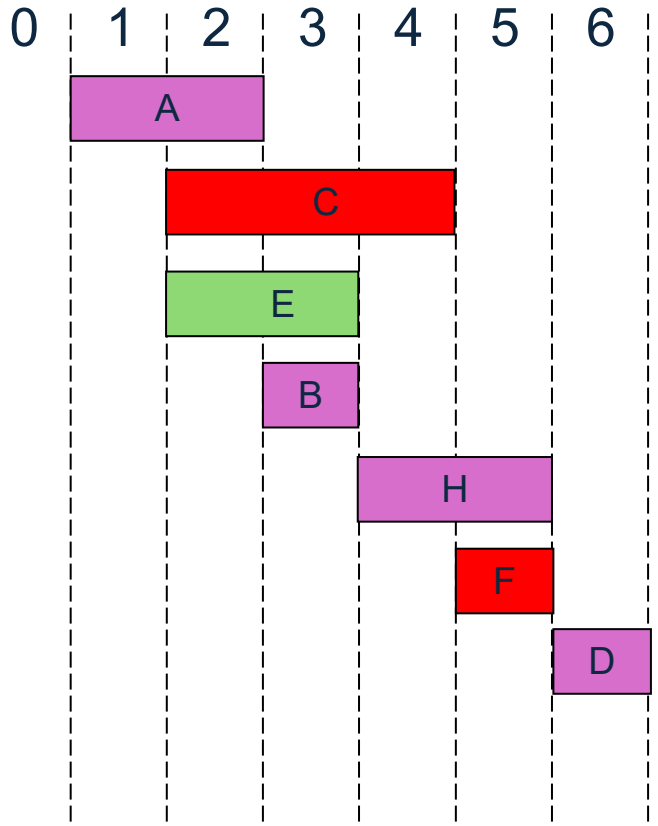
in-class assignment

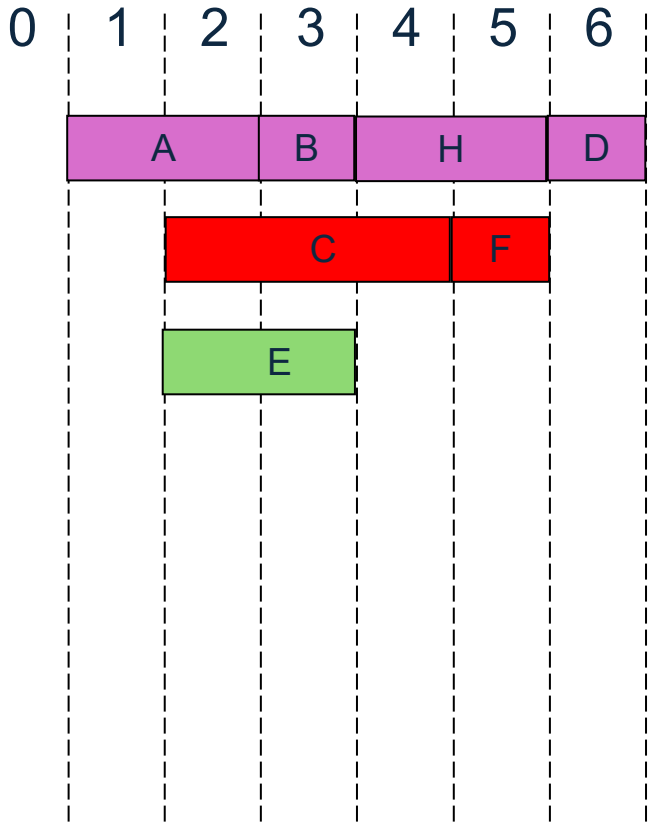




随堂作业

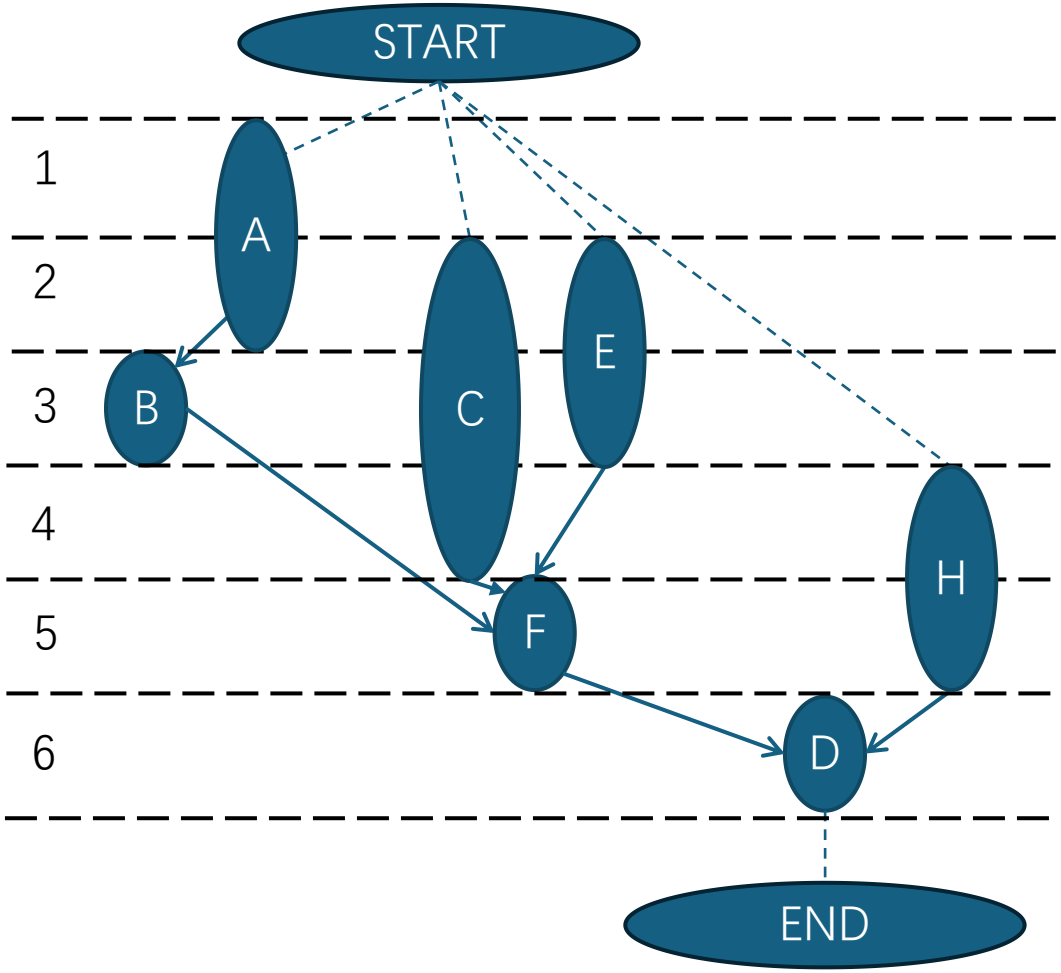
in-class assignment

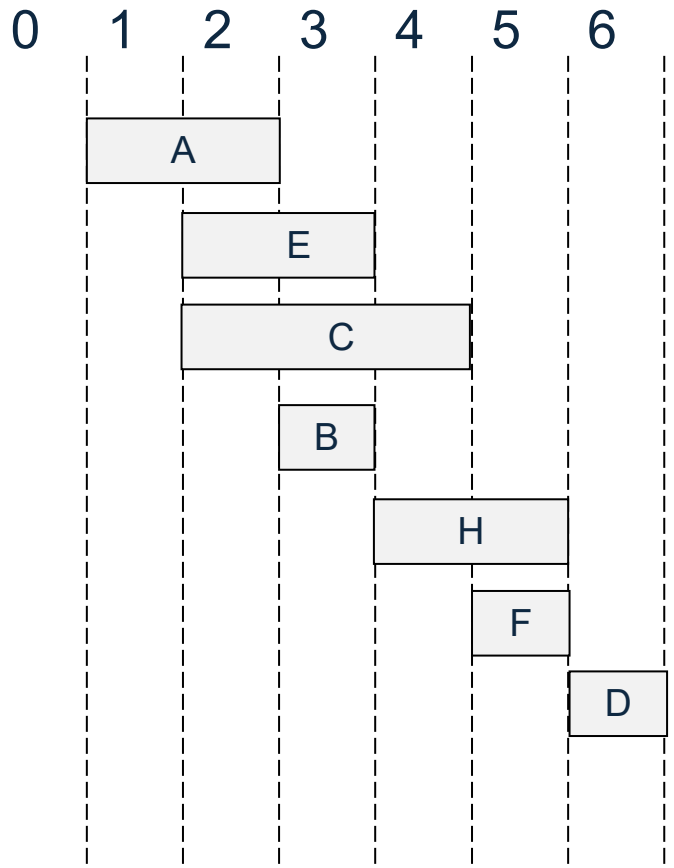




随堂作业

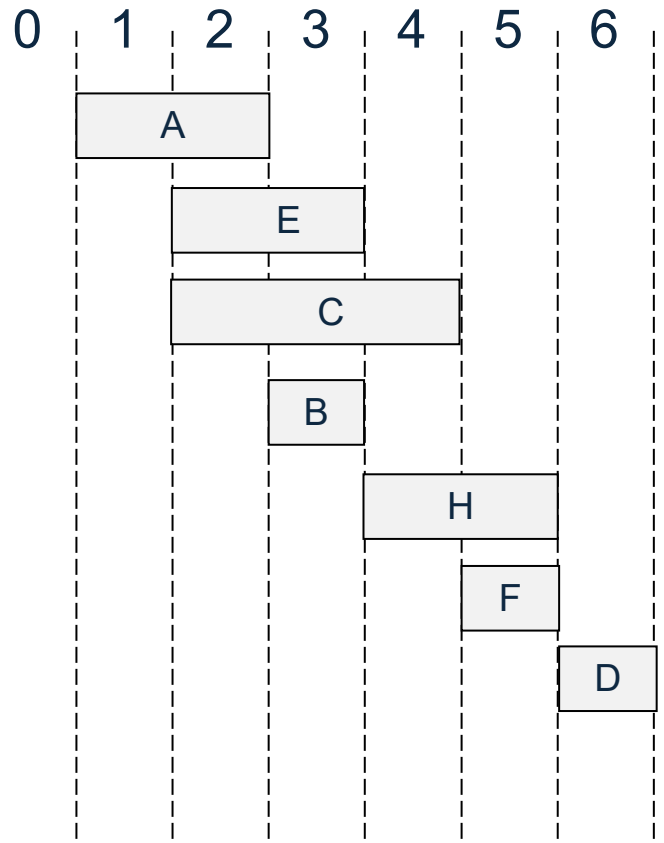
in-class assignment

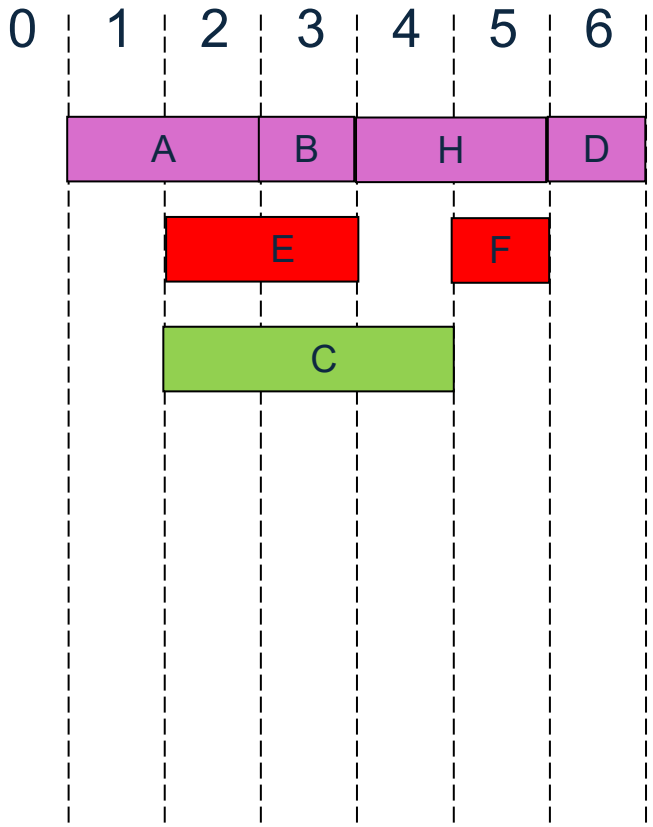




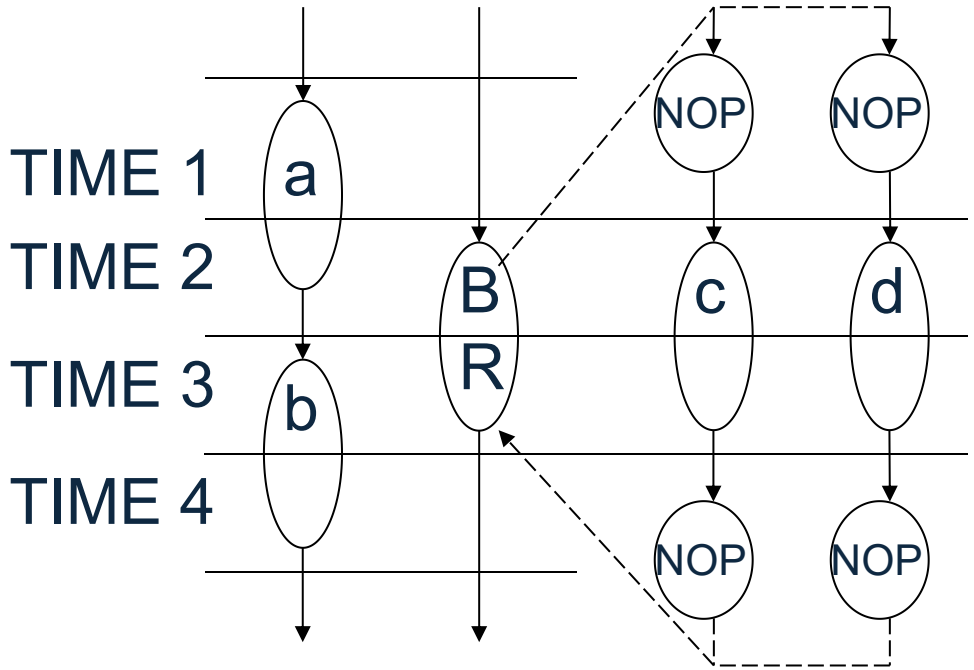
随堂作业

in-class assignment

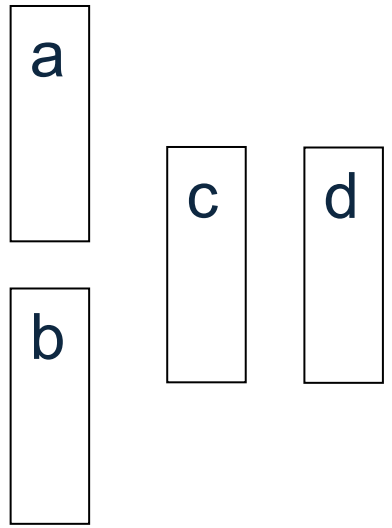




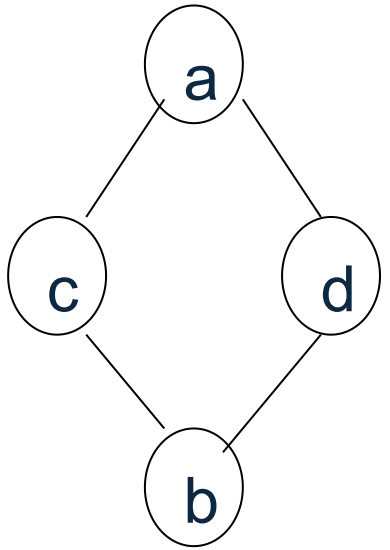
一些特殊情况



(a)



(b)



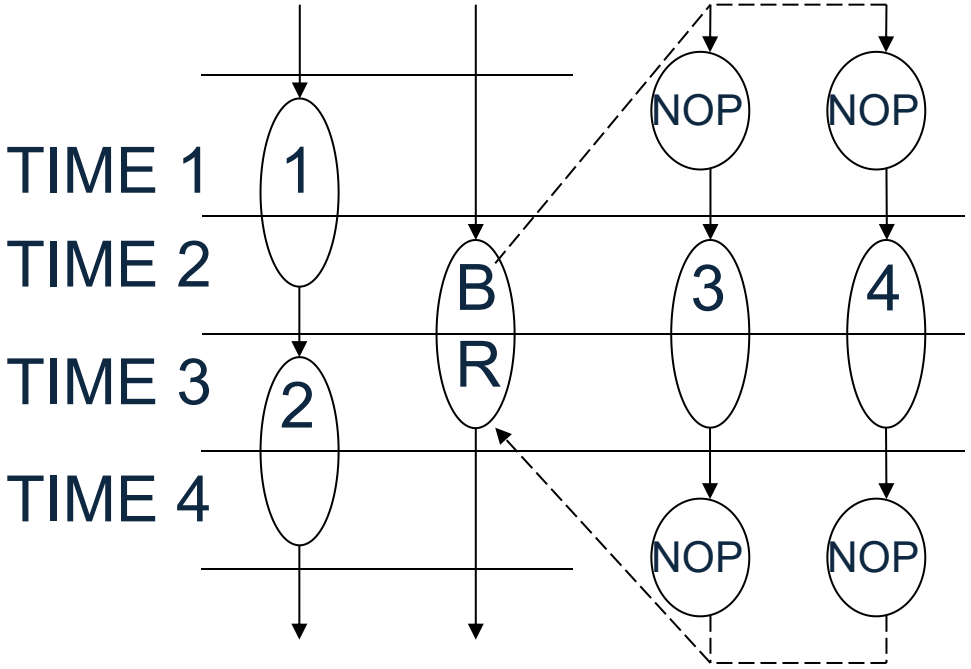
(c)

用ILP形式化绑定问题

ILP formulation of binding

- 布尔变量 bir
  - 表示操作 i 是否和资源 r绑定
- 最小化: $\sum_{r=1}^{a} \cup_{i=1}^{n} b_{i,r}$
- 约束:
- 1. 对每个操作i有: $\sum_{r=1}^{a} b_{i,r} = 1$
- 2. 对每个资源r，在每个周期都有: $\sum_{i=1}^{n} b_{ir} \sum_{m=l-(d_i-1)}^{l} x_{i,m} \leq 1$

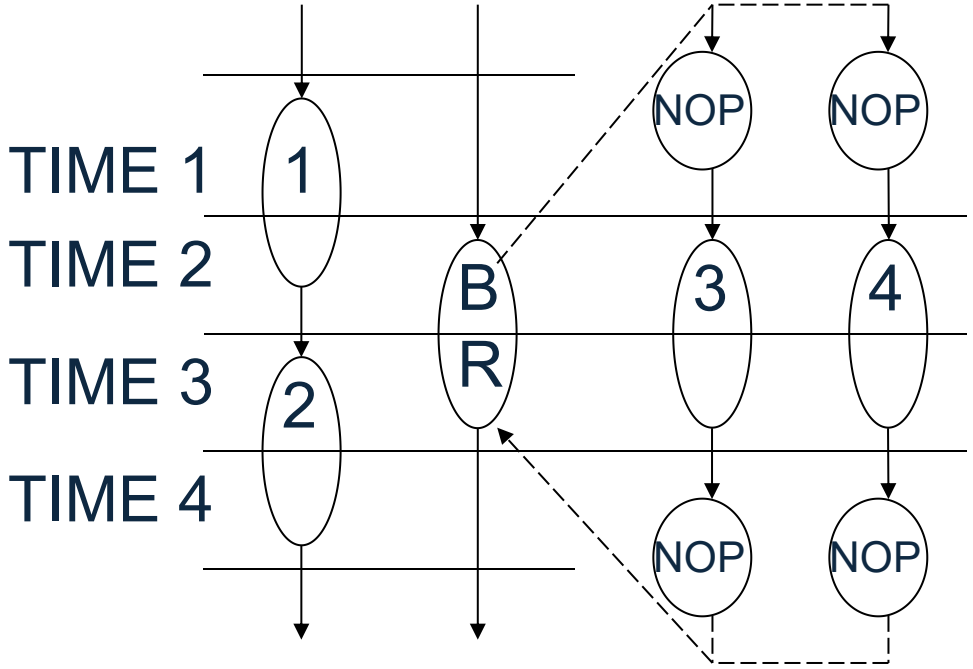
不适用于特殊情况



b11 b21 b31 b41
b12 b22 b32 b42
b13 b23 b33 b43
b14 b24 b34 b44

$$\sum_{r=1}^{a} \bigcup_{i=1}^{n} b_{i,r}$$

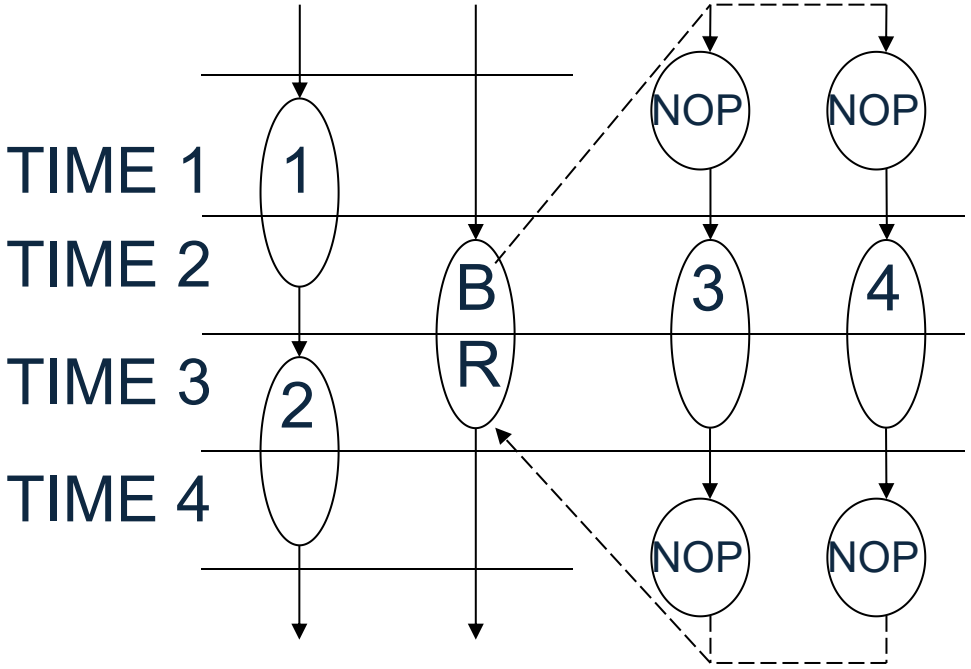
Min: (b11∪b21∪b31∪b41)+(b12∪b22∪b32∪b42)+(b13∪b23∪b33∪b43)+(b14∪b24∪b34∪b44)



b11 b21 b31 b41
b12 b22 b32 b42
b13 b23 b33 b43
b14 b24 b34 b44

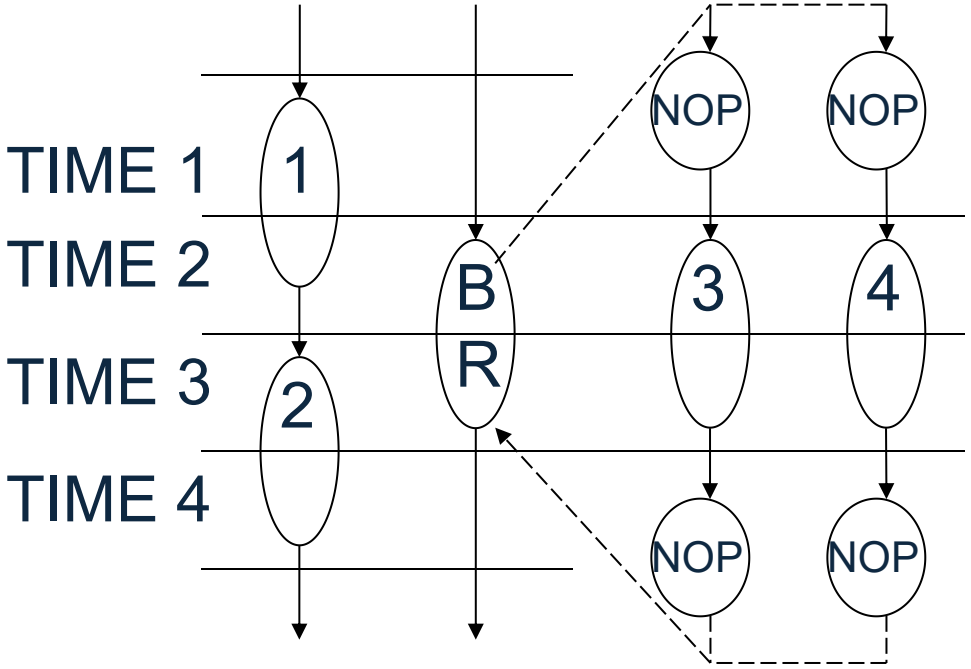
$$\sum_{r=1}^{a} b_{i,r} = 1$$

b11+b12+b13+b14=1
b21+b22+b23+b24=1
b31+b32+b33+b34=1
b41+b42+b43+b44=1



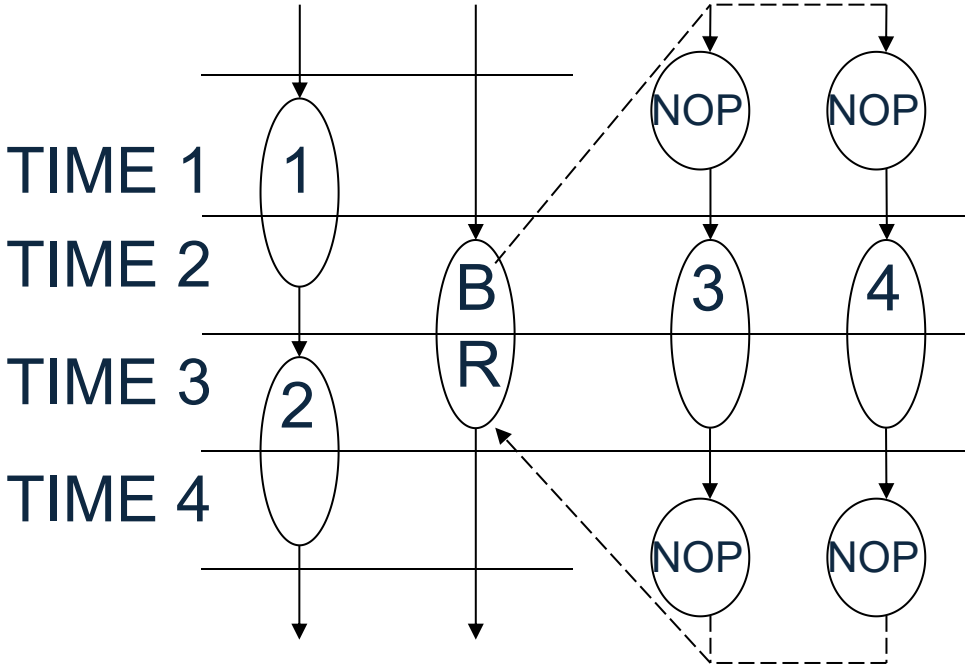
$$\sum_{i=1}^{n} b_{ir} \sum_{m=l-(d_i-1)}^{l} x_{i,m} \leq 1$$

b11<=1
b11+b31+b41<=1
b21+b31+b41 <=1
b21<=1
b12<=1
b12+b32+b42<=1
b22+b32+b42 <=1
b22<=1
b13<=1
b13+b33+b43<=1
b23+b33+b43<=1
b23<=1
b14<=1
b14+b34+b44<=1
b24+b34+b44<=1
b24<=1



$$\sum_{i=1}^{n} b_{ir} \sum_{m=l-(d_i-1)}^{l} x_{i,m} \leq 1$$

b11+b31+b41<=1
b21+b31+b41<=1
b12+b32+b42<=1
b22+b32+b42<=1
b13+b33+b43<=1
b23+b33+b43<=1
b14+b34+b44<=1
b24+b34+b44<=1



Subject to
b11+b12+b13+b14=1
b21+b22+b23+b24=1
b31+b32+b33+b34=1
b41+b42+b43+b44=1
b11+b31+b41<=1
b21+b31+b41<=1
b12+b32+b42<=1
b22+b32+b42<=1
b13+b33+b43<=1
b23+b33+b43<=1
b14+b34+b44<=1
b24+b34+b44<=1
Min
(b11∪b21∪b31∪b41)+(b12∪b22∪b32∪b42)+(b13∪b23∪b33∪b43)+(b14∪b24∪b34∪b44)

因为是特殊顺序图，所以无法求得最优解

用ILP形式化绑定问题

ILP formulation of binding

- 布尔变量 bir
  - 表示操作 I 是否和资源 r绑定
- 最小化：

$$\sum_{r=1}^{a} \cup_{i=1}^{n} b_{i,r}$$

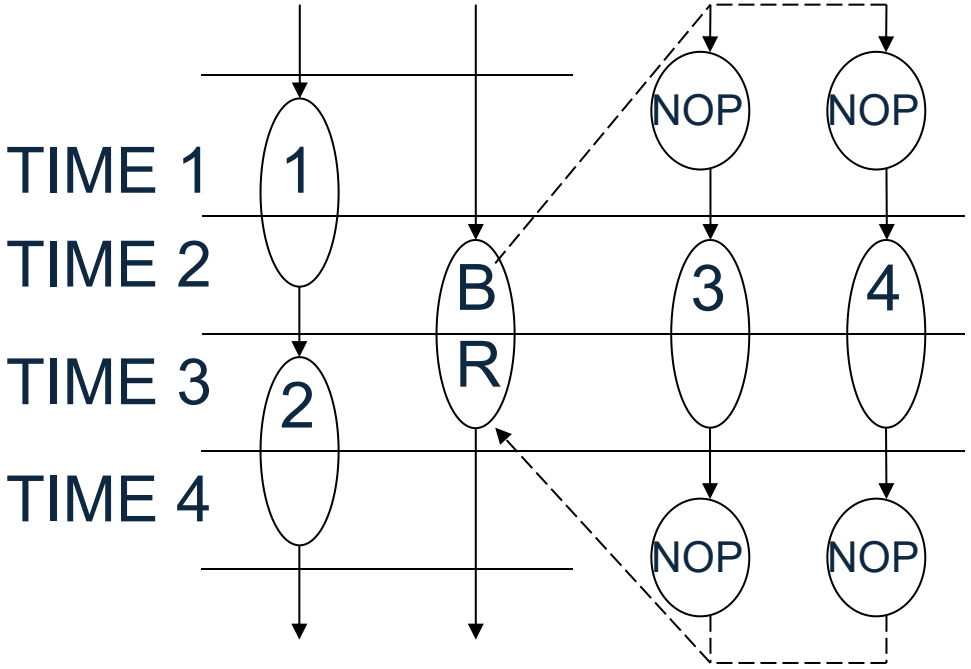
- 约束

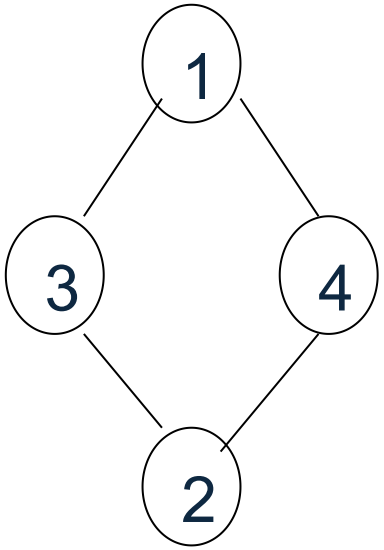
1. 对每个操作i有：

$$\sum_{r=1}^{a} b_{i,r} = 1$$

2. 对每个资源r，对每个冲突关系<vi,vj>有：

$$b_{ir}+b_{jr}<=1$$

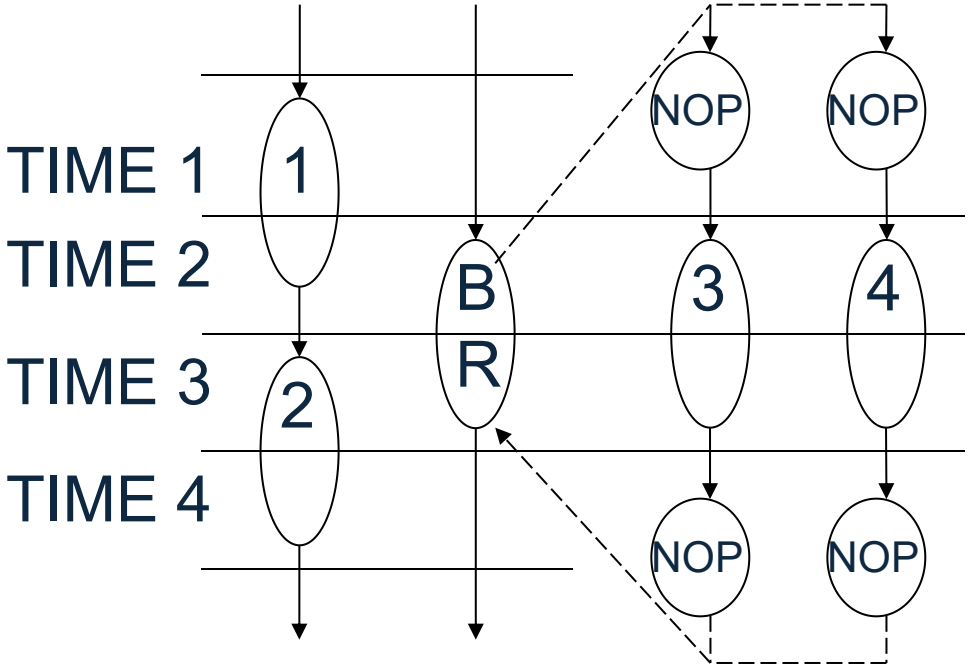




对每个r，对每个冲突关系<vi,vj>有：

bir+bjr<=1

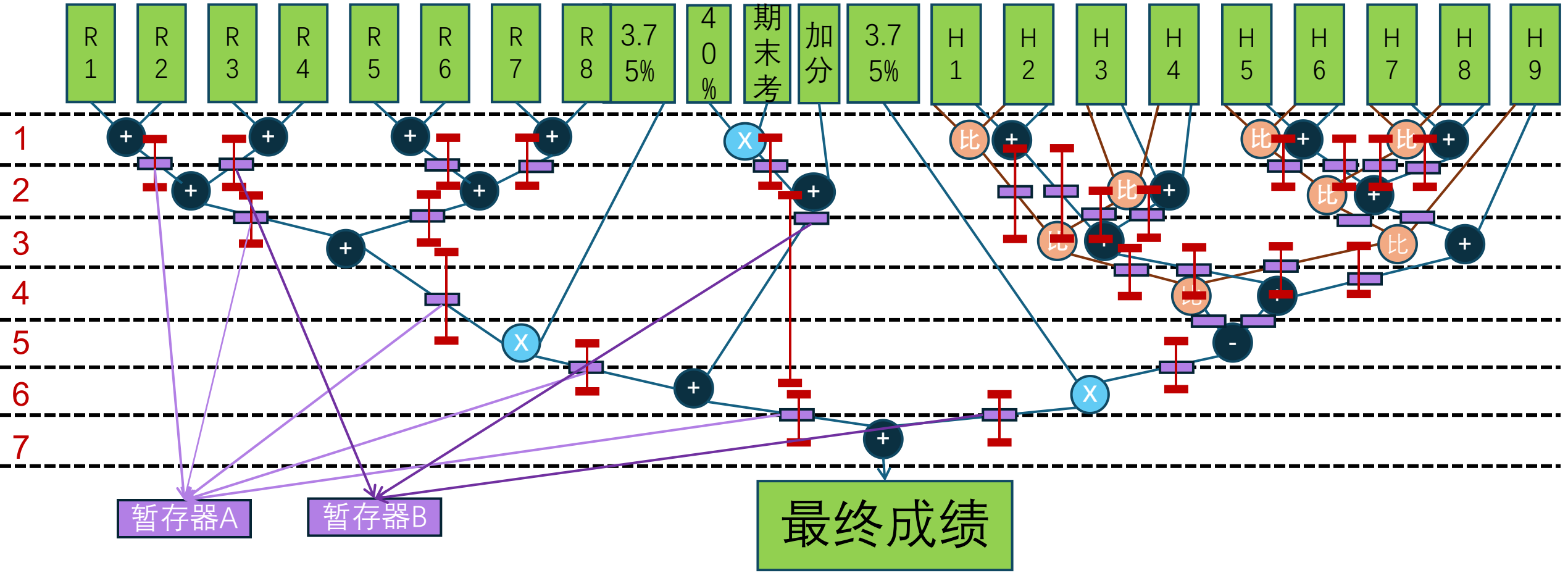
b11+b31<=1
b11+b41<=1
b21+b31<=1
b21+b41<=1
b12+b32<=1
b12+b42<=1
b22+b32<=1
b22+b42<=1
b13+b33<=1
b13+b43<=1
b23+b33<=1
b23+b43<=1
b14+b34<=1
b14+b44<=1
b24+b34<=1
b24+b44<=1



Subject to
b11+b12+b13+b14=1
b21+b22+b23+b24=1
b31+b32+b33+b34=1
b41+b42+b43+b44=1
b11+b31<=1
b11+b41<=1
b21+b31<=1
b21+b41<=1
b12+b32<=1
b12+b42<=1
b22+b32<=1
b22+b42<=1
b13+b33<=1
b13+b43<=1
b23+b33<=1
b23+b43<=1
b14+b34<=1
b14+b44<=1
b24+b34<=1
b24+b44<=1
Min
(b11∪b21∪b31∪b41)+(b12∪b22∪b32∪b42)+
(b13∪b23∪b33∪b43)+(b14∪b24∪b34∪b44)

```
b11 = 1.0
b12 = 0.0
b13 = 0.0
b14 = 0.0
b21 = 1.0
b22 = 0.0
b23 = 0.0
b24 = 0.0
b31 = 0.0
b32 = 1.0
b33 = 0.0
b34 = 0.0
b41 = 0.0
b42 = 1.0
b43 = 0.0
b44 = 0.0
```

顺序图

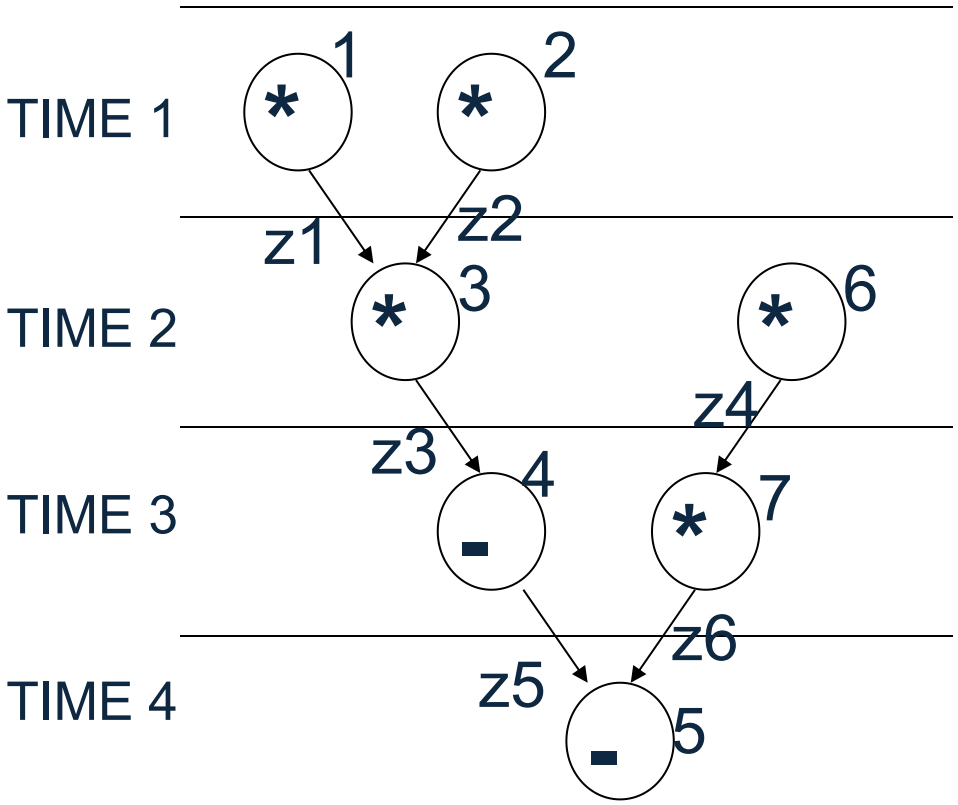

R1
R2
R3
R4
R5
R6
R7
R8
3.75%
40%
期末考
加分
3.75%
H1
H2
H3
H4
H5
H6
H7
H8
H9
1
2
3
4
5
6
7
比
暂存器A
暂存器B
最终成绩

兼容图 / 冲突图具有特殊性质

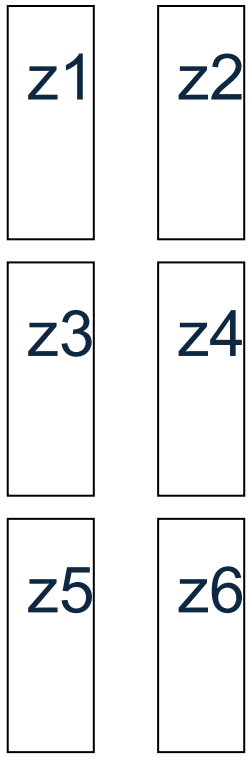
Register binding problem

- Compatibility graph 兼容图
  - Comparability graph可比较图
- Conflict graph冲突图
  - Interval graph区间图
- 多项式时间算法:
  - Golumbic’s algorithm
  - Left-edge algorithm

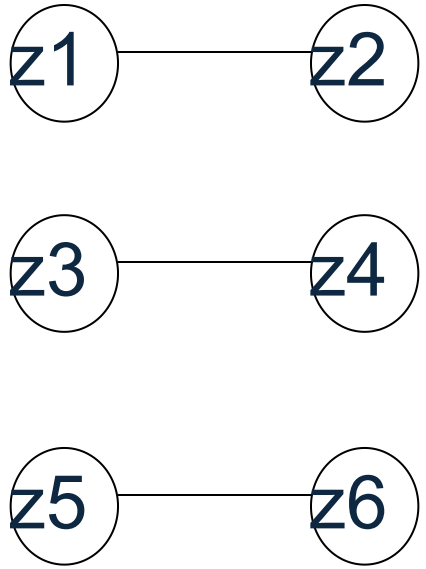
例子


TIME 1
TIME 2
TIME 3
TIME 4
1
2
3
4
5
6
7
z1
z2
z3
z4
z5
z6

(a)

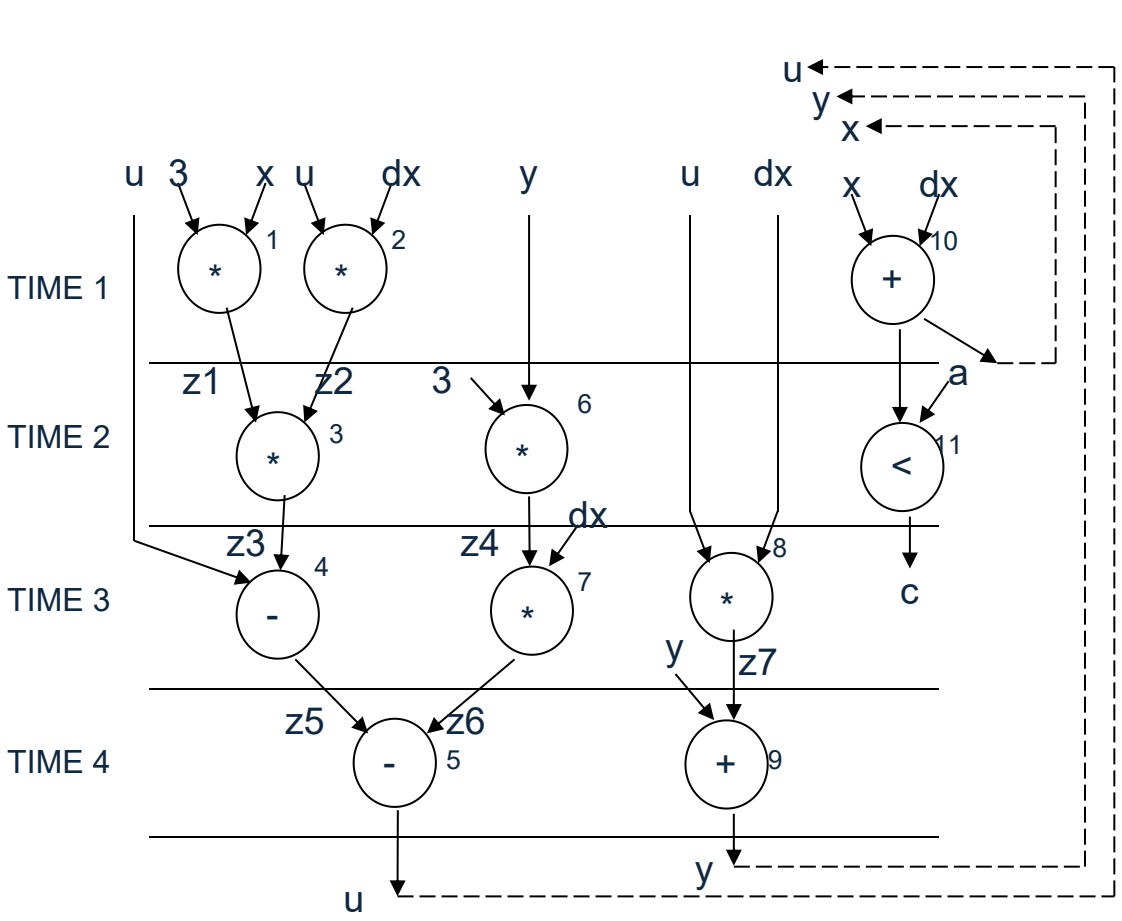

z1
z2
z3
z4
z5
z6

(b)

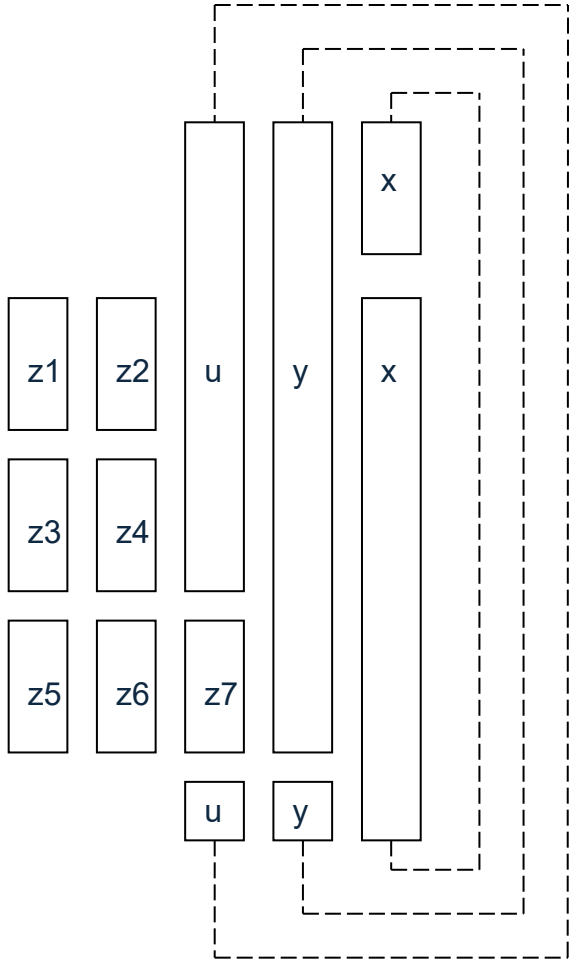

z1
z2
z3
z4
z5
z6

(c)

例子

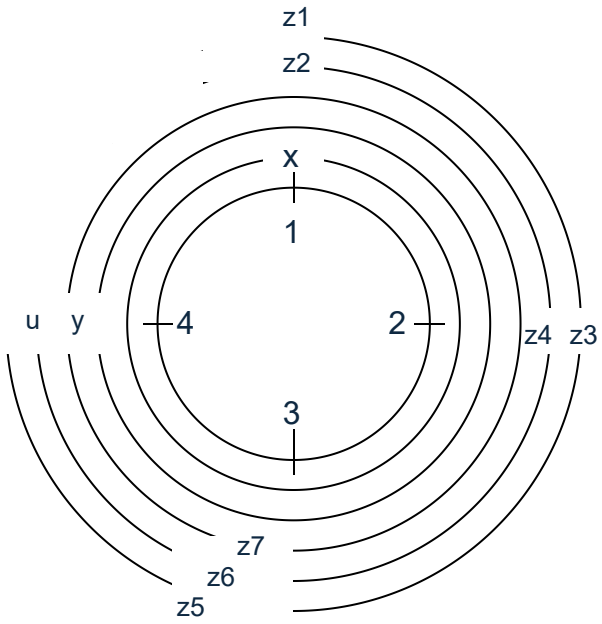

u
y
x
u 3 x u dx y u dx x dx
1
2
10
TIME 1
z1 z2 3 a
3
6
11
TIME 2
dx
z3 z4
4
7
8
TIME 3
c
y z7
z5 z6
5
9
TIME 4
y
u

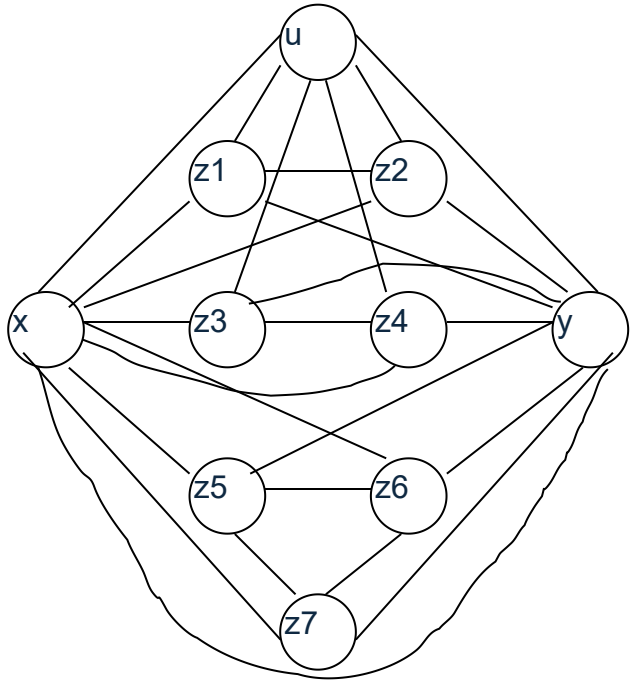
(a)


x
z1 z2 u y x
z3 z4
z5 z6 z7
u y

(b)

例子





模块选择问题

Module Selection Problem

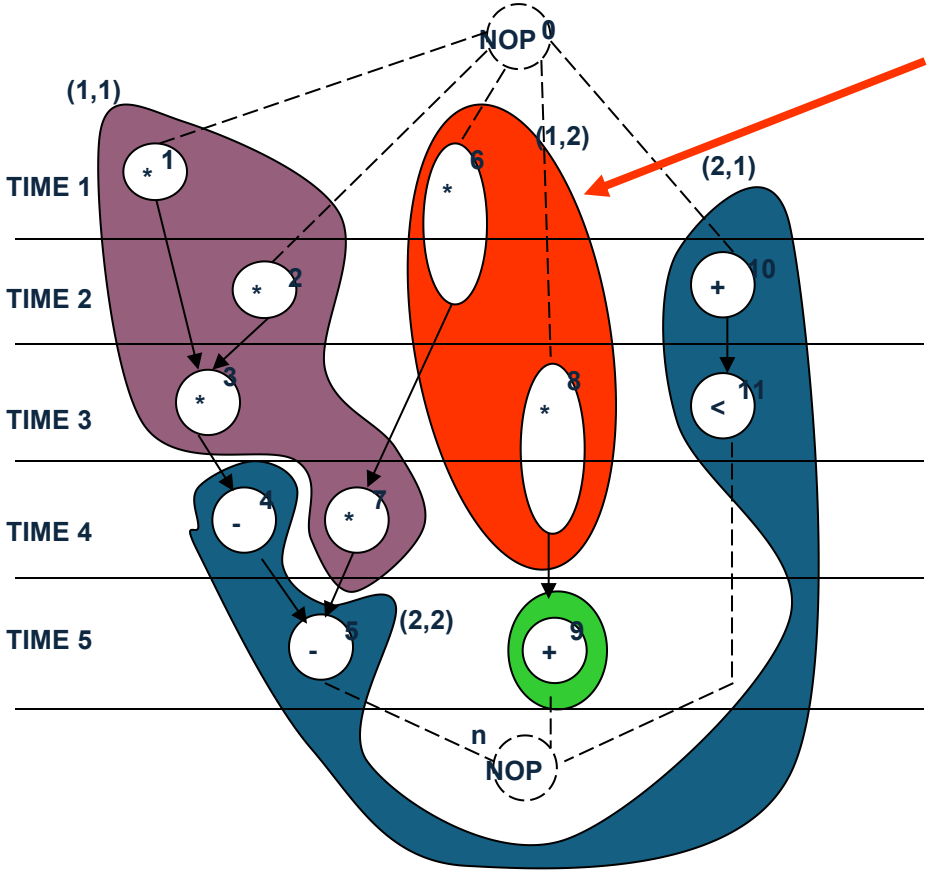
- 资源共享的扩展
  - 每种类型可以有多个资源
- 示例:
  - Ripple-carry adder串行进位加法器
  - Carry-look-ahead adder前瞻进位加法器
- 资源建模:
  - 拥有以下参数的资源子类型:
  - （面积、延迟）等参数

模块选择问题

Module Selection Problem

- ILP
- 启发式算法
  - 使用最快的资源子类型保证关键路径
  - 在非关键路径上使用较慢的资源来节省面积

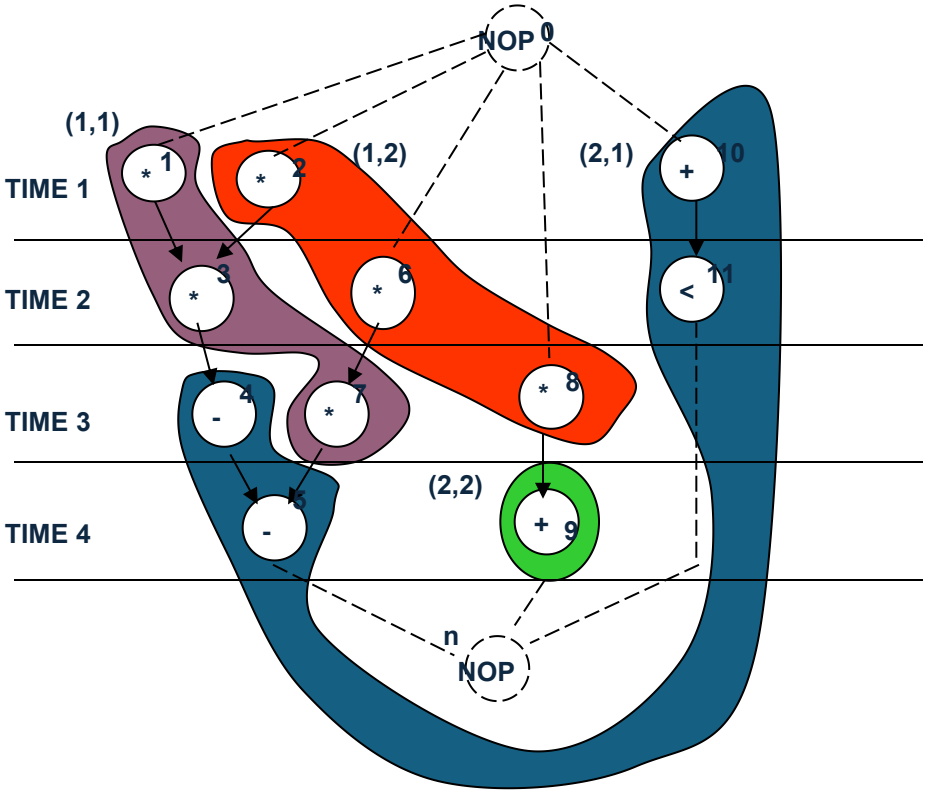
例子



通过用比较慢的资源节省时间

- 乘法器类型：（面积，延迟）=（5,1）和（2,2）
- 周期限制为5

例子



- 乘法器类型：（面积，延迟）=（5,1）和（2,2）
- 周期限制为5