EDA软件设计-调度算法

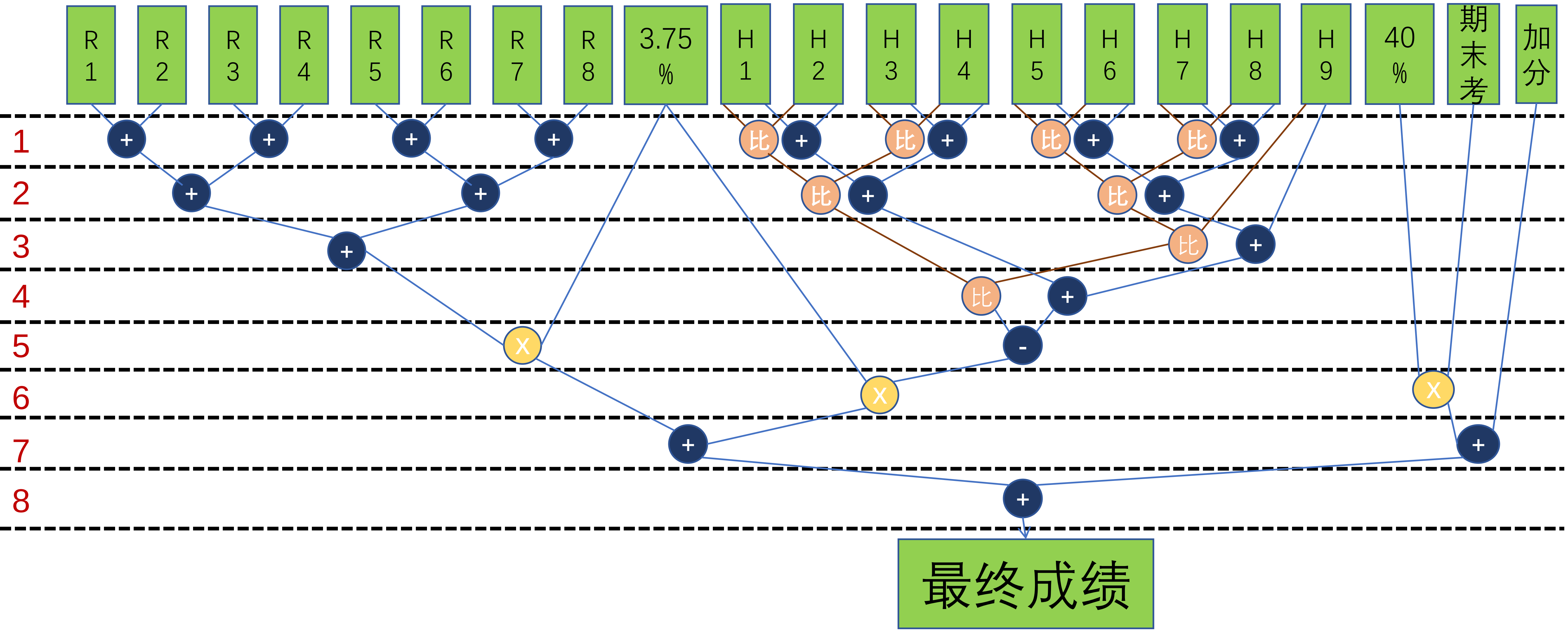
2024-2025年春季学期

在正式开始之前

软件开发者

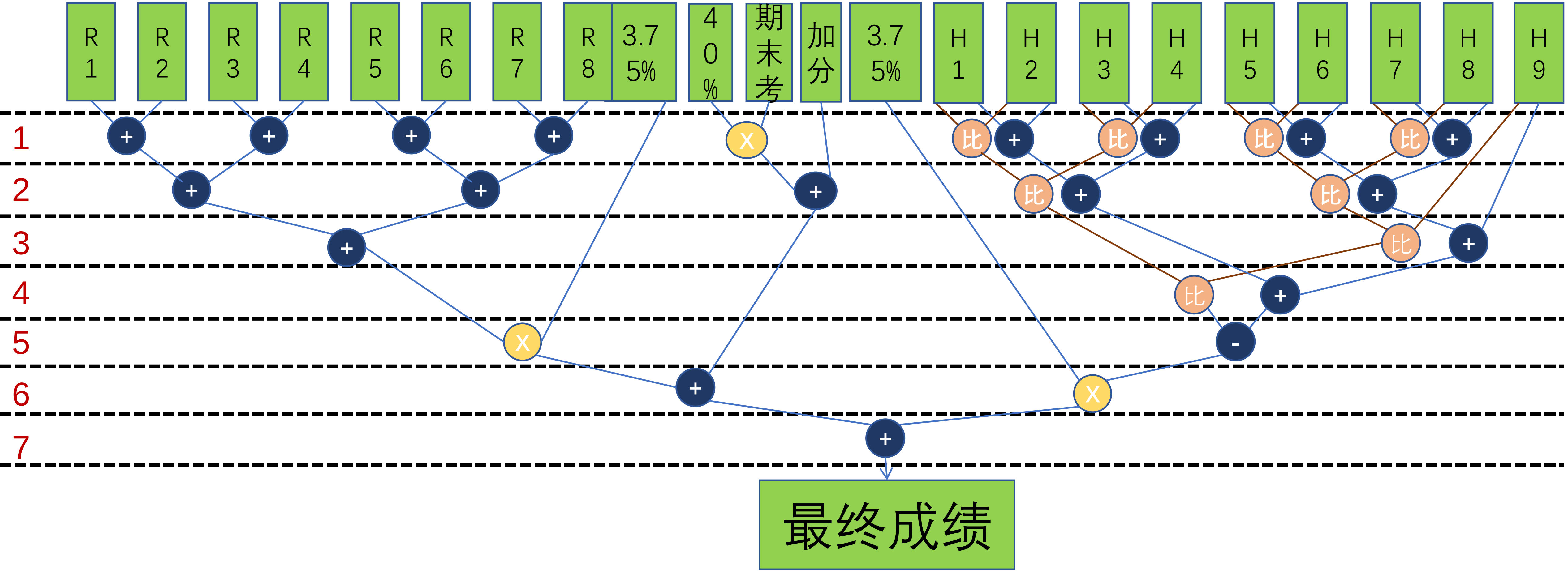
EDA软件开发者

顺序图



R 1
R 2
R 3
R 4
R 5
R 6
R 7
R 8
3.75 %
H 1
H 2
H 3
H 4
H 5
H 6
H 7
H 8
H 9
40 %
期末考
加分
1
2
3
4
5
6
7
8
+
比
X
-
最终成绩

顺序图



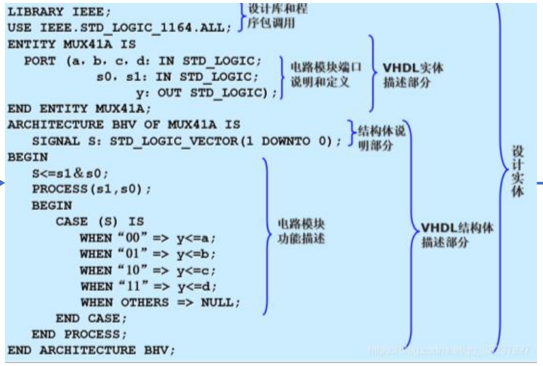
R 1
R 2
R 3
R 4
R 5
R 6
R 7
R 8
3.7 5%
4 0 %
期末考
加分
3.7 5%
H 1
H 2
H 3
H 4
H 5
H 6
H 7
H 8
H 9
1
2
3
4
5
6
7
最终成绩

软件开发者

EDA软件开发者

在这门课程中我们主要关注

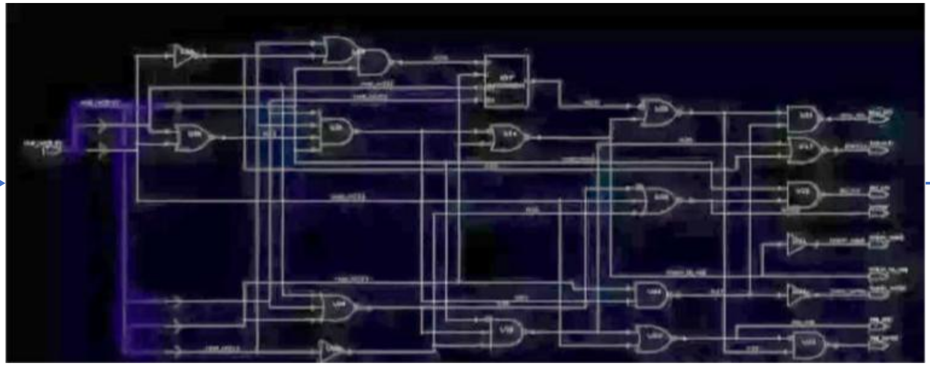
高级
综合



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MUX41A IS
  PORT (a, b, c, d: IN STD_LOGIC;
         s0, s1: IN STD_LOGIC;
              y: OUT STD_LOGIC);
END ENTITY MUX41A;
ARCHITECTURE BHV OF MUX41A IS
  SIGNAL S: STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
  S<=s1&s0;
  PROCESS(s1,s0);
  BEGIN
    CASE (S) IS
      WHEN "00" => y<=a;
      WHEN "01" => y<=b;
      WHEN "10" => y<=c;
      WHEN "11" => y<=d;
      WHEN OTHERS => NULL;
    END CASE;
  END PROCESS;
END ARCHITECTURE BHV;
```

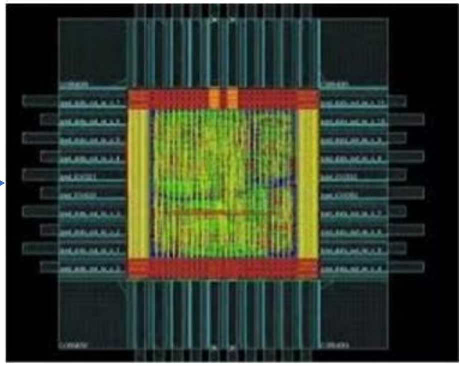

HDL代码

逻辑
综合



门级网表

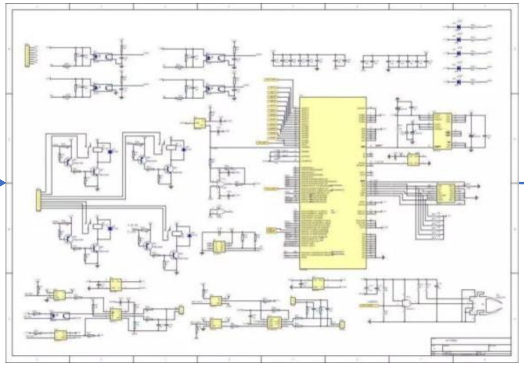
布局
布线



物理版图

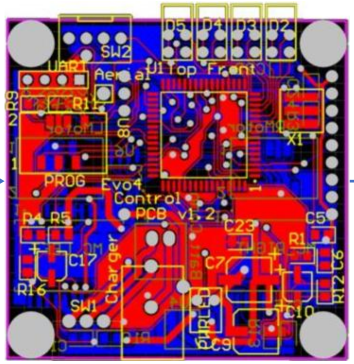
工厂

原理图
设计



电路原理图

PCB
设计



PCB设计文件

工厂

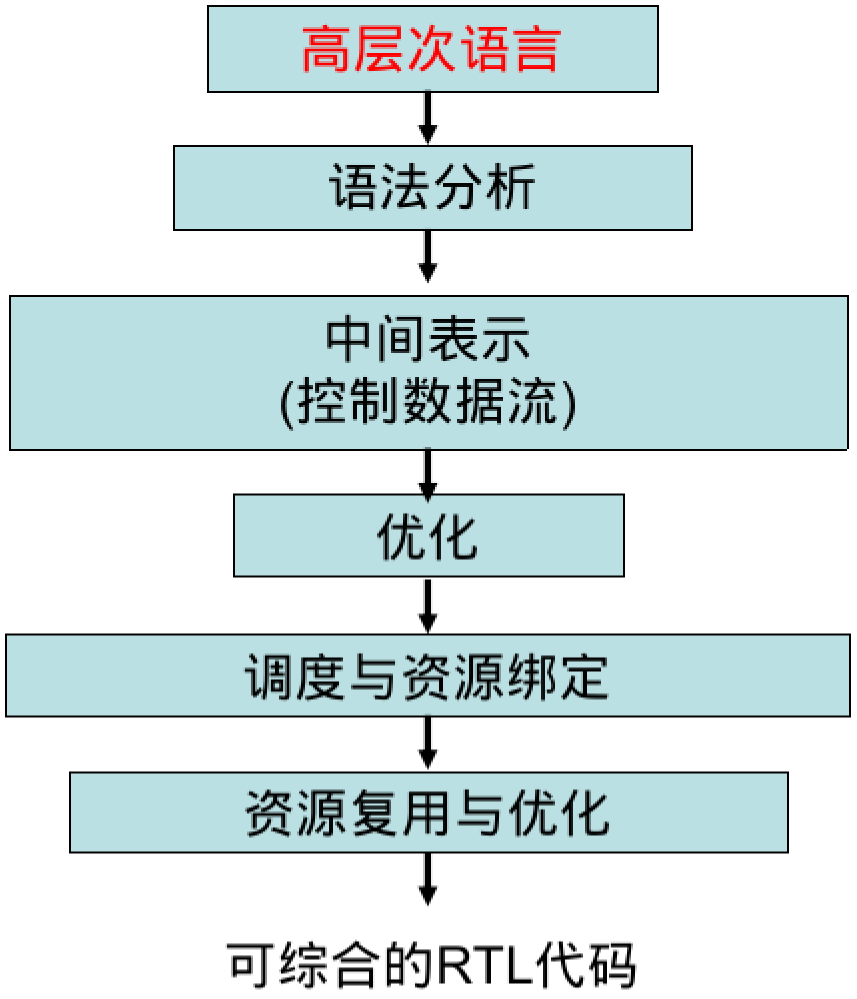
PCB实物板

工厂

芯片

高层次综合编译器的基本流程

The basic flow of a high-level synthesis compiler



前端/中端: 代码转化与优化

- 循环展开/流水线
- 强度弱化 / 运算表达式平衡
- 位宽分析与优化
- 访存分析, 数组切分, 数据复用

后端 : 行为综合与优化

- 调度
- 资源绑定

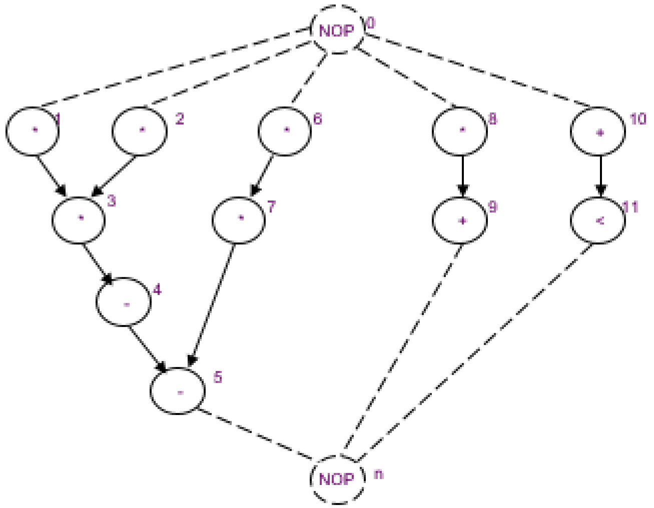
  例如, 功能模块绑定、寄存器绑定、RAM端口绑定
- 代码生成

  例如微架构生成 & RTL/约束生成

高层次综合

High-level synthesis

- 以顺序图（即依赖图或数据流图）为行为参考



- 操作时序的约束
- 硬件资源可用性的约束

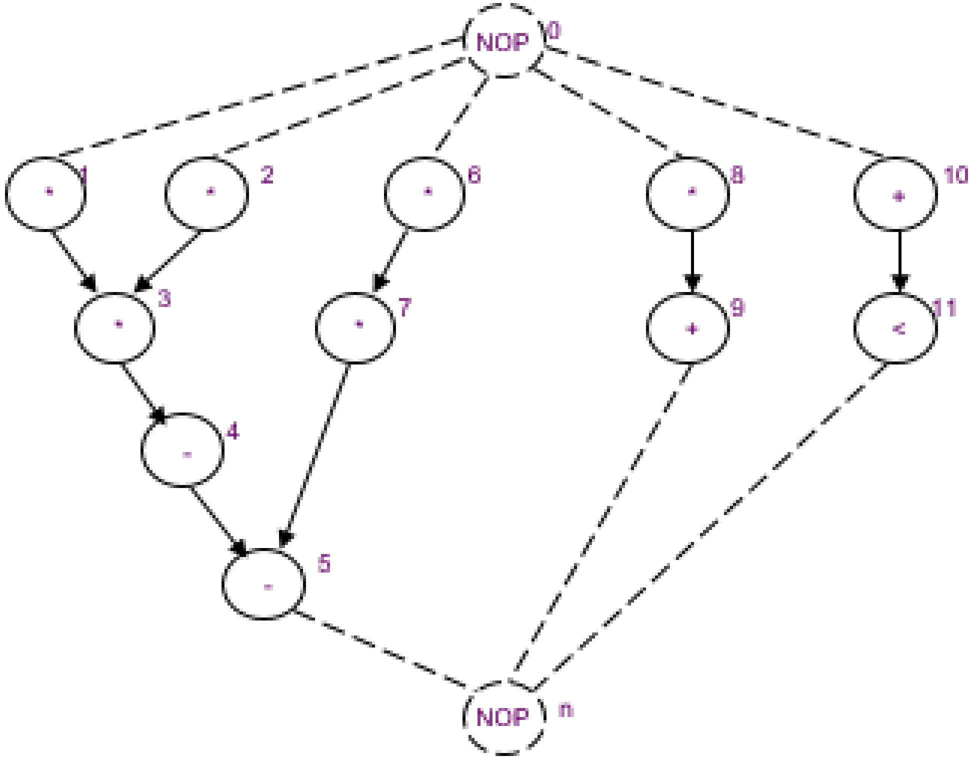
目标：

- 生成带有开始时间和资源注释的顺序图
- 可能有多个可行的解决方案（探索权衡）
- 满足约束，最小化目标：
  - 在面积约束的基础上最大化效率
  - 在效率约束的基础上最小化面积

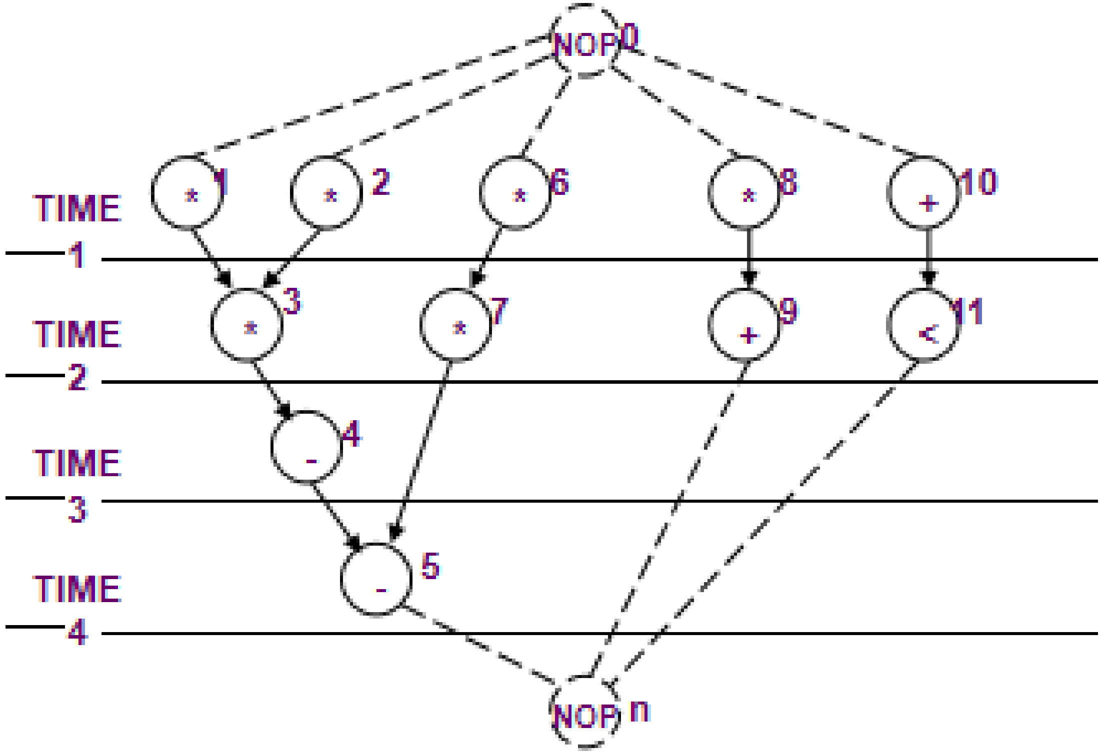
在时间域中的综合

Synthesis in the time domain

- 调度:
  - 为每个操作关联一个开始时间



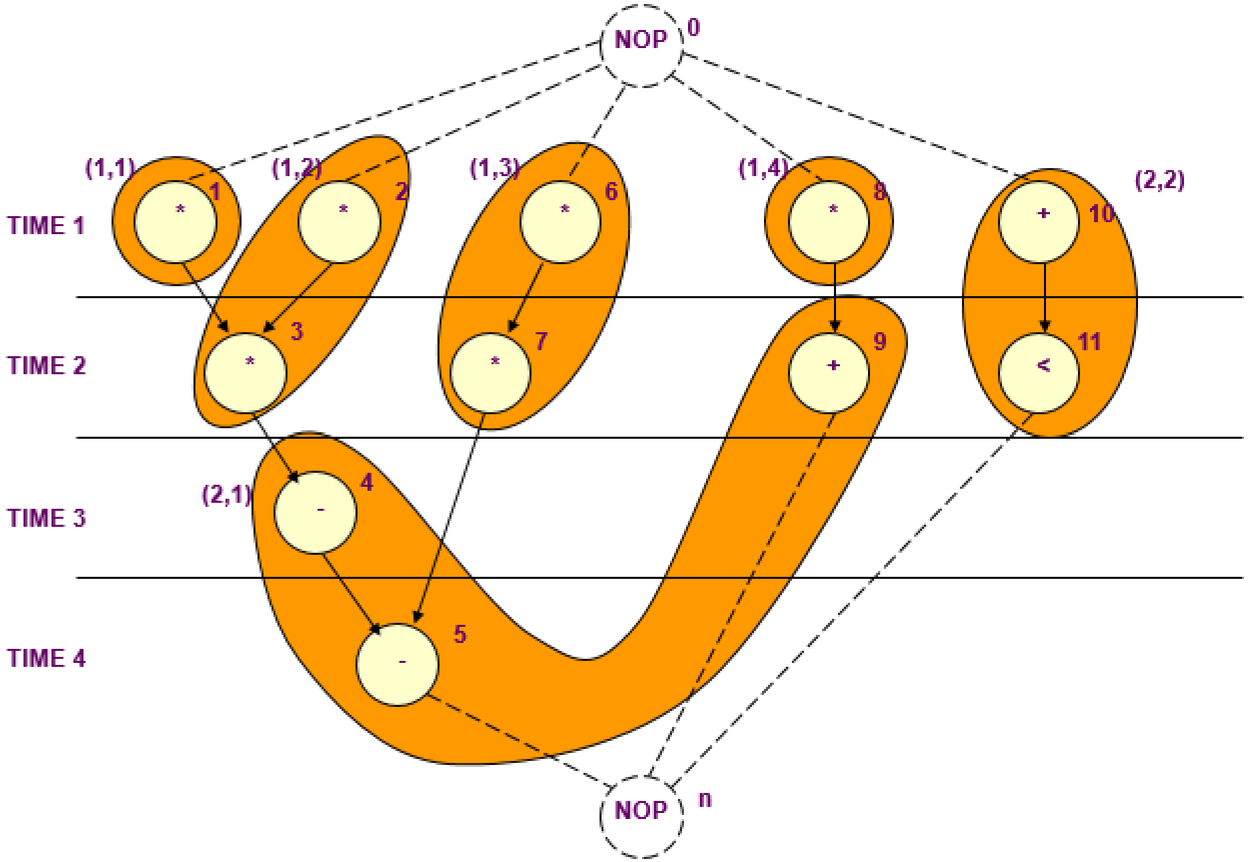
- 已调度的顺序图:
  - 带有开始时间注释的顺序图



在空间域中的综合

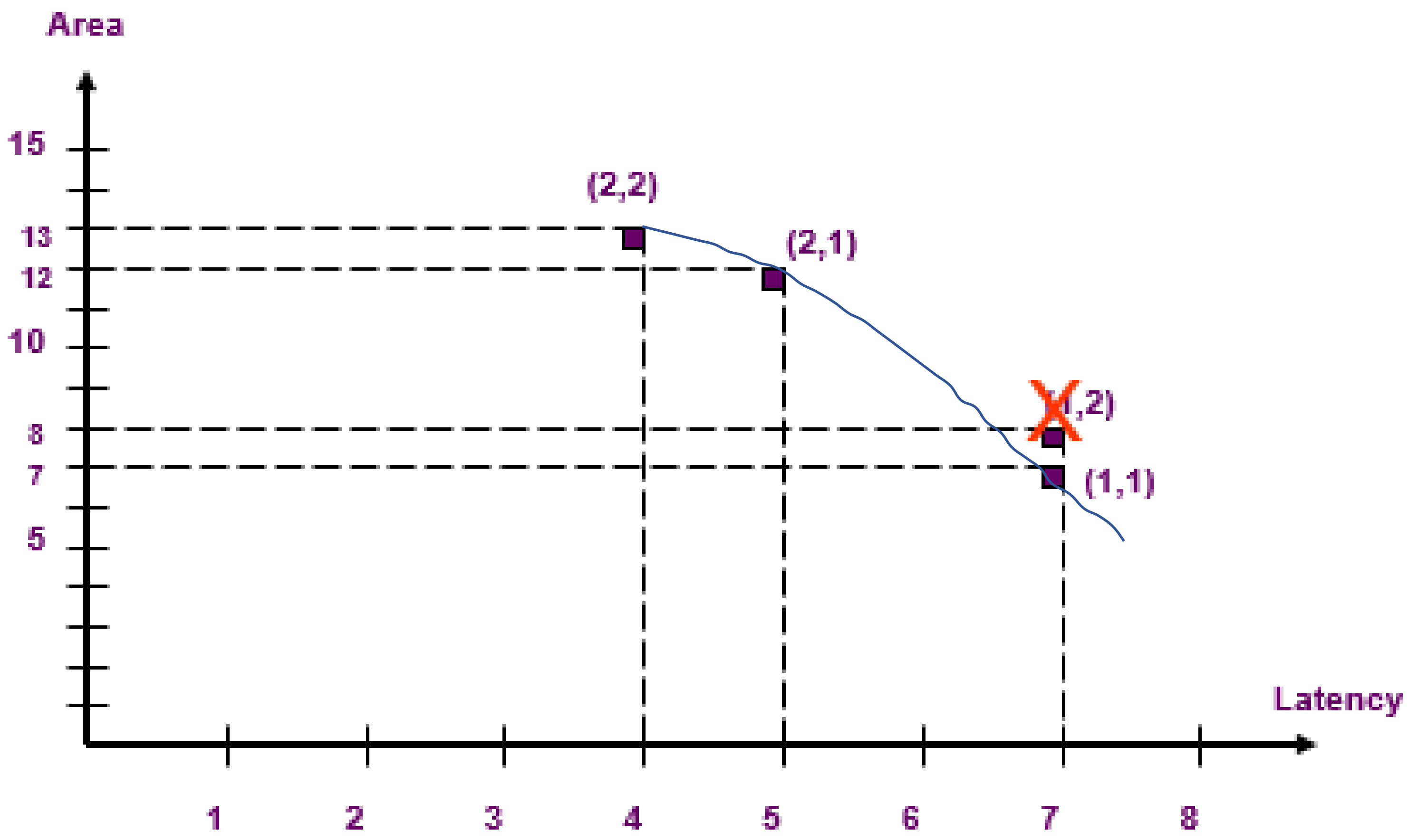
Synthesis in the spatial domain

- 绑定：
  - 将资源与具有相同类型的每个操作关联
  - 确定实现的面积
- 共享：
  - 将资源绑定到多个操作
  - 操作不能同时执行
- 绑定的顺序图：
  - 带有资源注释的顺序图



架构优化

Architecture optimization



调度算法

Scheduling Algorithms

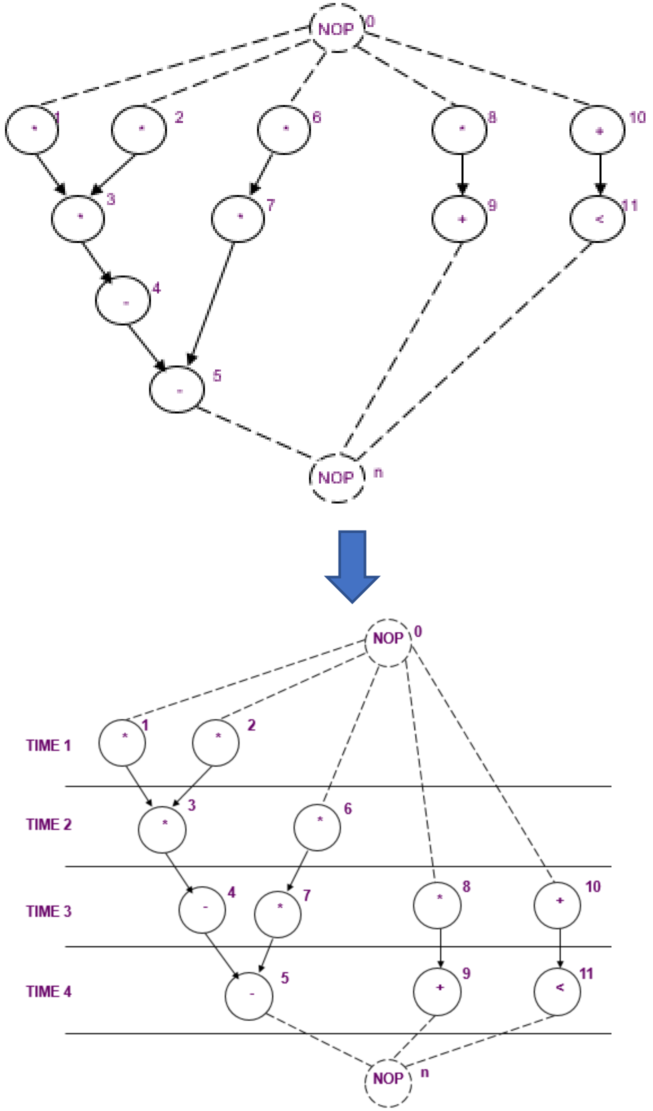
2024-3

主讲人：孙静翎

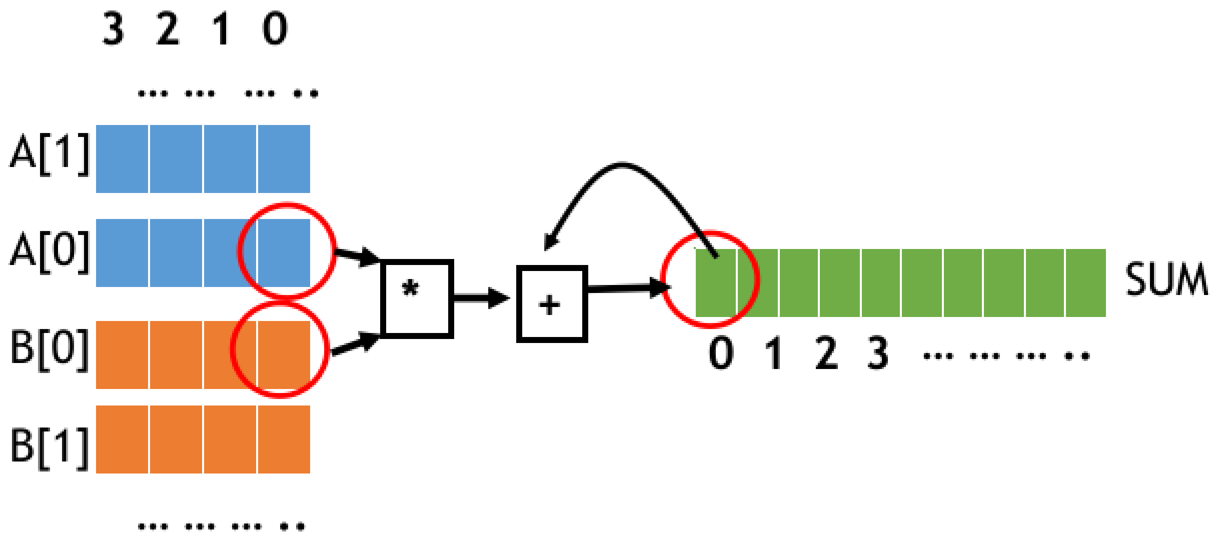
操作调度

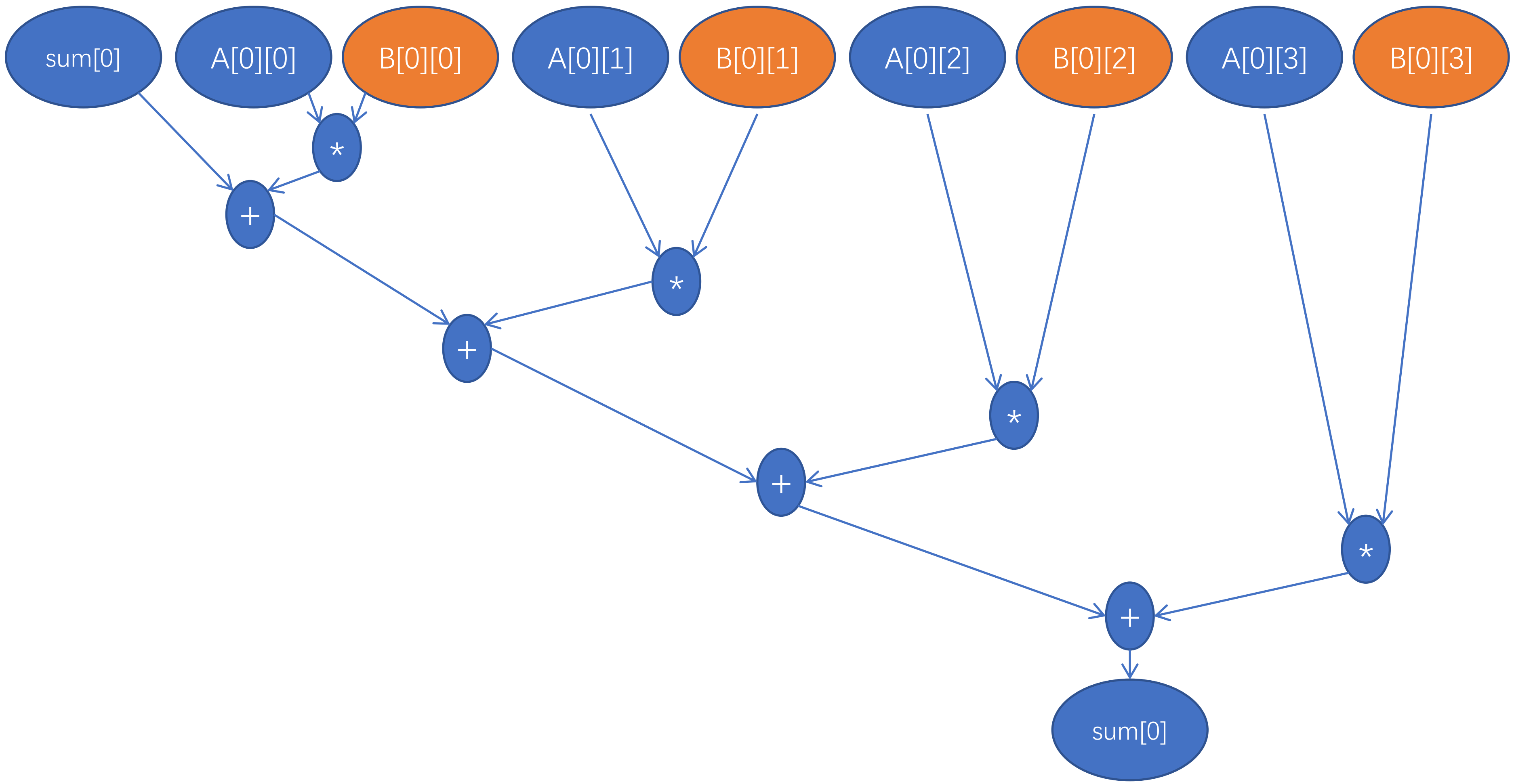
Operation Scheduling

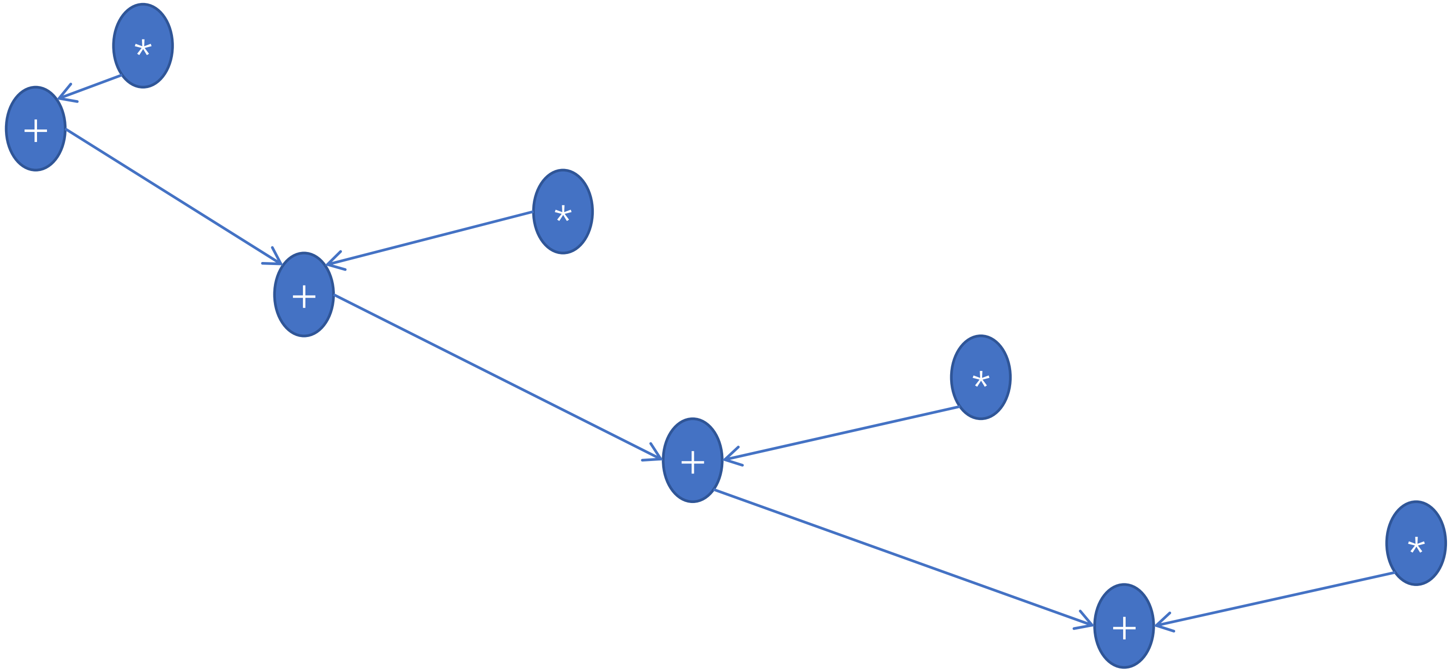
- 输入：
  - 顺序图 G(V, E)，含 n 个顶点
  - 操作延迟 $D = \{d_i: i=0..n\}$
- 输出：
  - 调度 φ ： 确定操作 vi 的开始时间 t_i
  - 延迟 $\lambda = t_n - t_0$
- 目标：权衡资源和时间

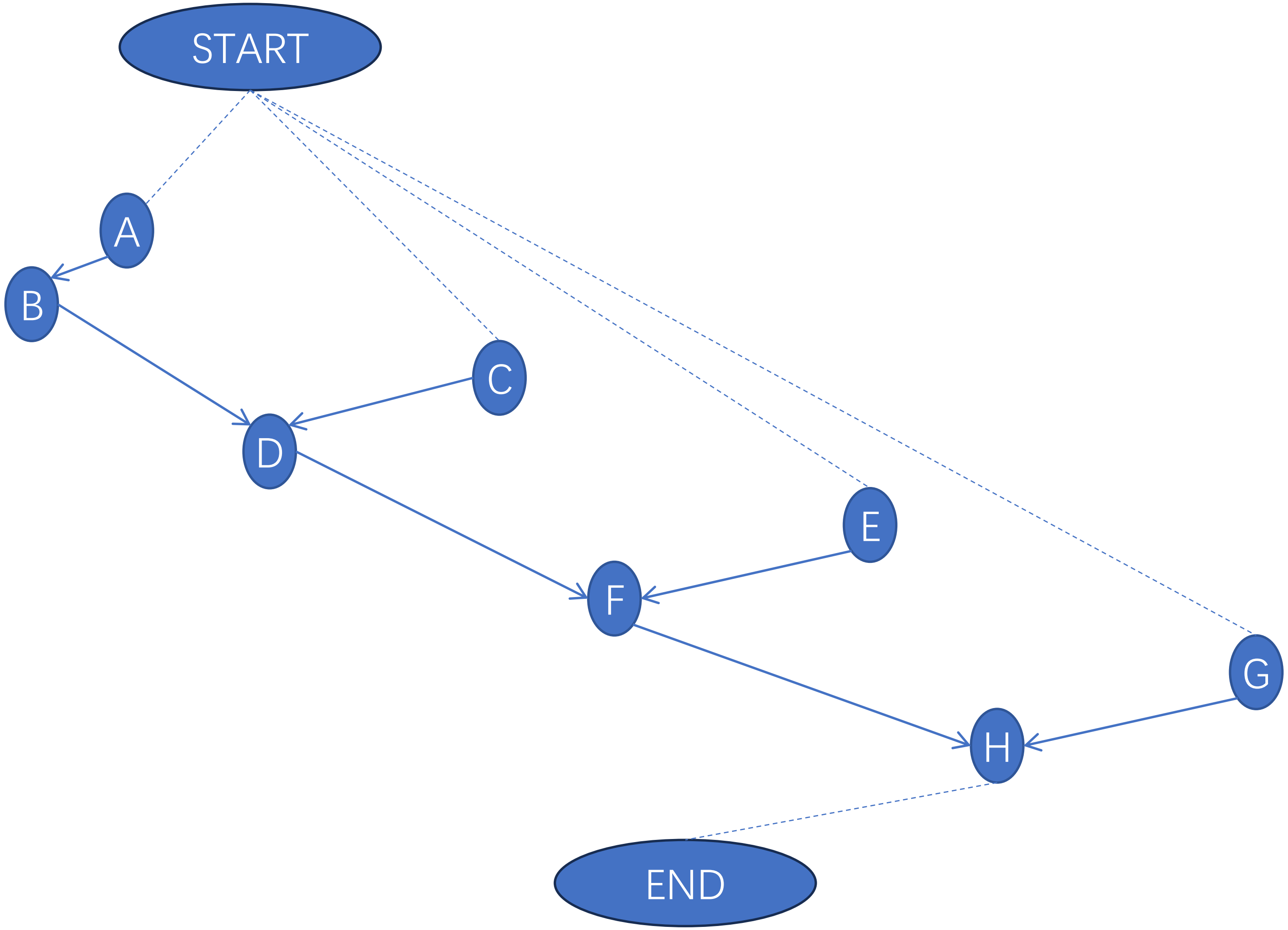


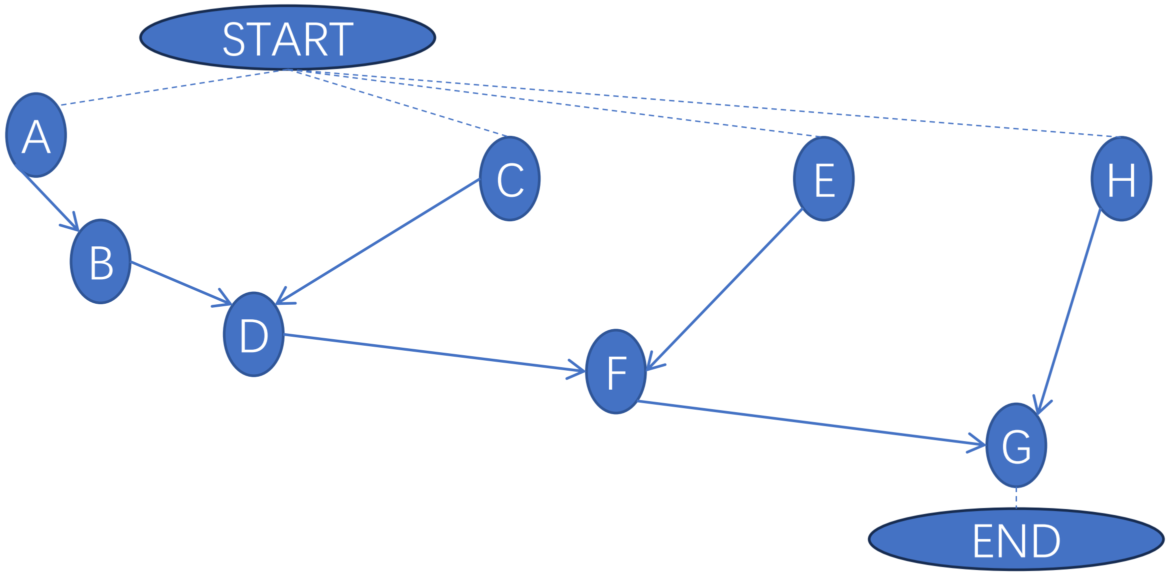
```
void vectorDot(float A[10][4], float B[10][4], float SUM[10])
{
    for (int i=0;i<10;i++)
    {
        SUM[i] = 0;
        for (int j=0;<4;j++)
        {
            SUM[i] += A[i][j] * B[i][j];
        }
    }
}
```




sum[0]
A[0][0]
B[0][0]
A[0][1]
B[0][1]
A[0][2]
B[0][2]
A[0][3]
B[0][3]
*
+
*
+
*
+
*
+
sum[0]




START
A
B
C
D
E
F
G
H
END



START
A
B
C
D
E
F
H
G
END

操作调度

Operation Scheduling

- 输入：
  - 顺序图 G(V, E)，含 n 个顶点
  - 操作延迟 $D = \{d_i: i=0..n\}$
- 输出：
  - 调度 φ ： 确定操作 vi 的开始时间 t_i
  - 延迟 $\lambda = t_n - t_0$
- 目标：权衡资源和时间

输入文件内容:
7
START 1
A 2
B 1
C 3
D 1
E 2
END 1
A B
C B
A E

输出文件内容:
START 0
A 1
C 1
B 4
E 3
D 1
END 5

最小延迟无约束调度问题

最小延迟无约束调度

Minimum-latency Unconstrained Scheduling

- 最简单的情况：没有约束，找到最小的延迟
- 输入：顶点集 V，延迟 D 和偏序关系集E，
- 输出：对每个操作进行标记$\varphi: V \rightarrow Z^+$，使得：
  - $t_i = \varphi(v_i)$
  - $t_i \geqslant t_j + d_j$ 对于所有 $(v_i, v_j) \in E$
  - $\lambda = t_n - t_0$ 是最小的
- 可在多项式时间内解决
- 使用的 ASAP 算法：拓扑顺序

ASAP调度算法

ASAP调度算法

ASAP scheduling algorithm

ASAP ($G_s(V,E)$) {

  Schedule v_0 by setting $t_0 = 0$;

  repeat {

    Select a vertex v_i whose predecessors are all scheduled;

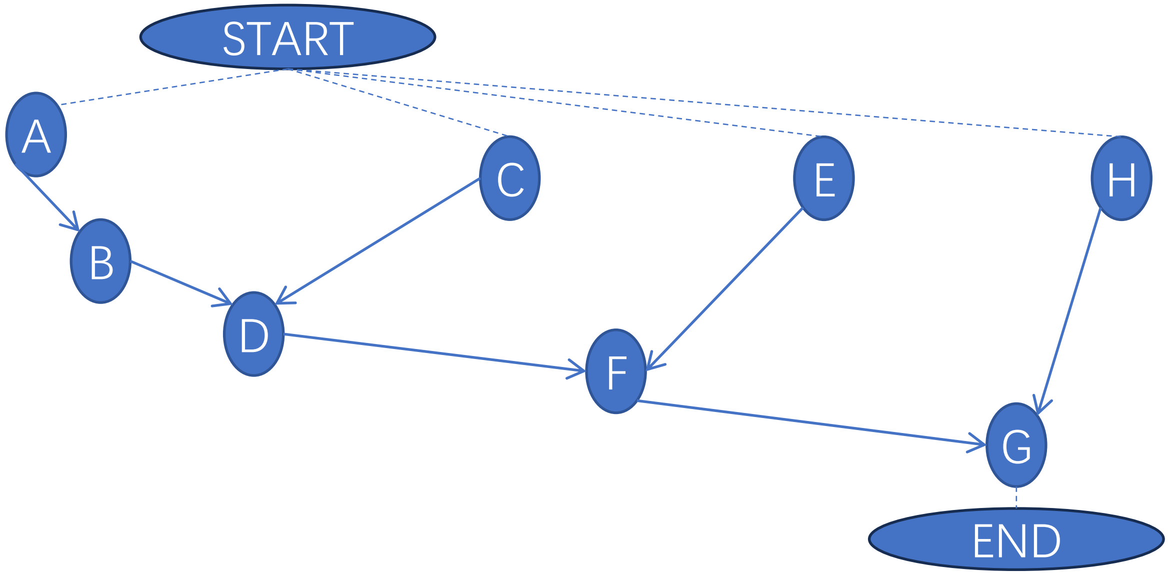
    Schedule v_i by setting $t_i = \max \; t_j + d_j$;

  }

  until (v_n is scheduled);

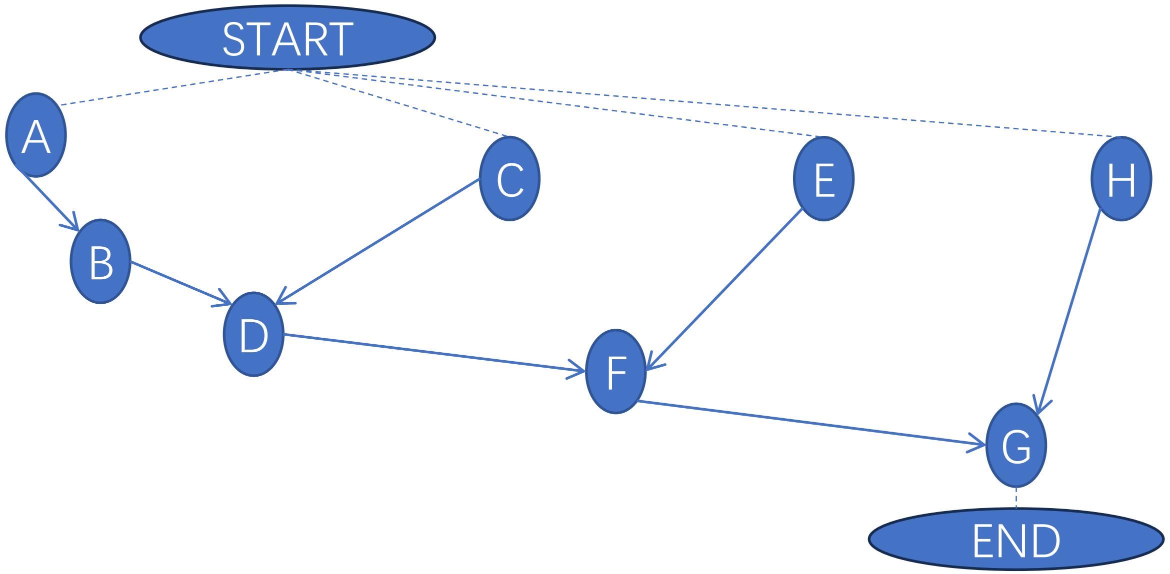
  return (G);

}



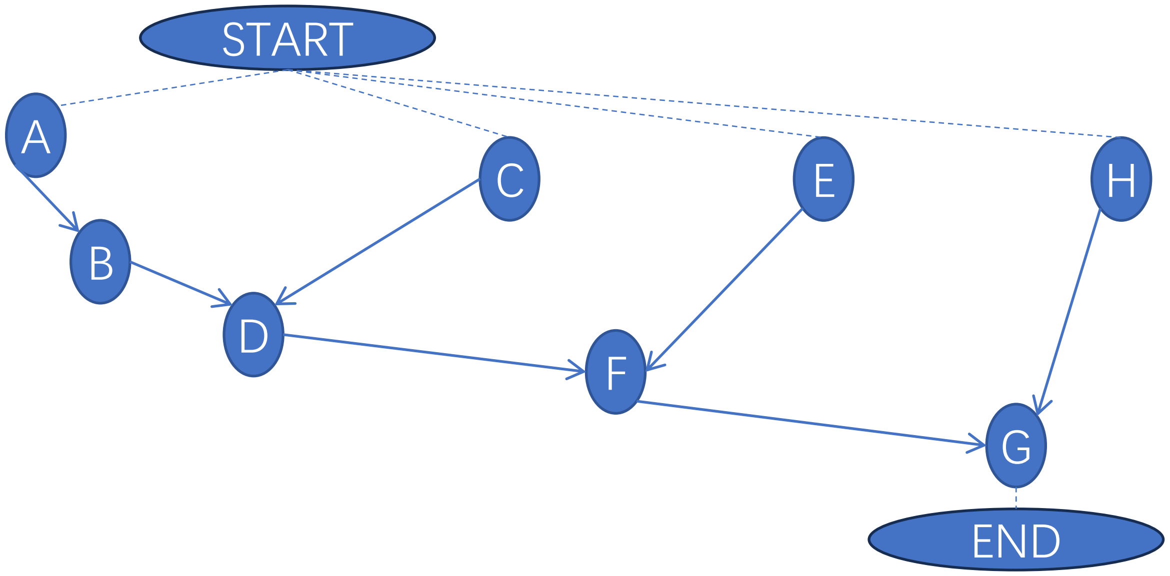
ASAP调度算法

ASAP scheduling algorithm



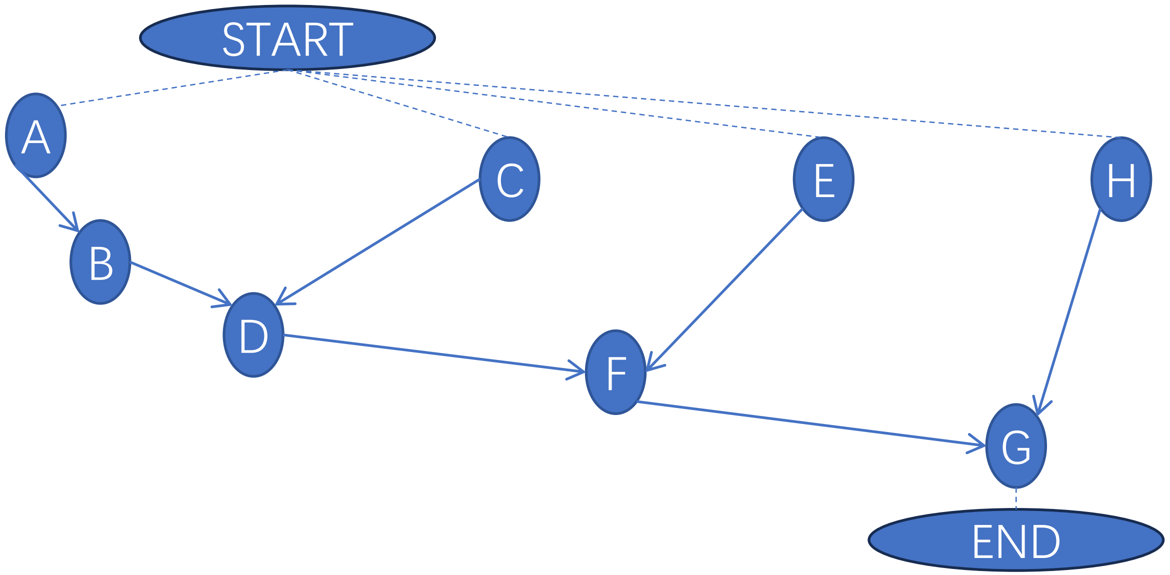
10
START 1
A 1
B 1
C 1
D 1
E 1
F 1
G 1
H 1
END 1

START A
START C
START E
START G
A B
C D
B D
E F
D F
H G
F G
G END



START 0

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
        until (v_n is scheduled);
        return ( G );}
```



START 0

ASAP ($G_s(V,E)$) {

Schedule v_0 by setting $t_0 = 0$;

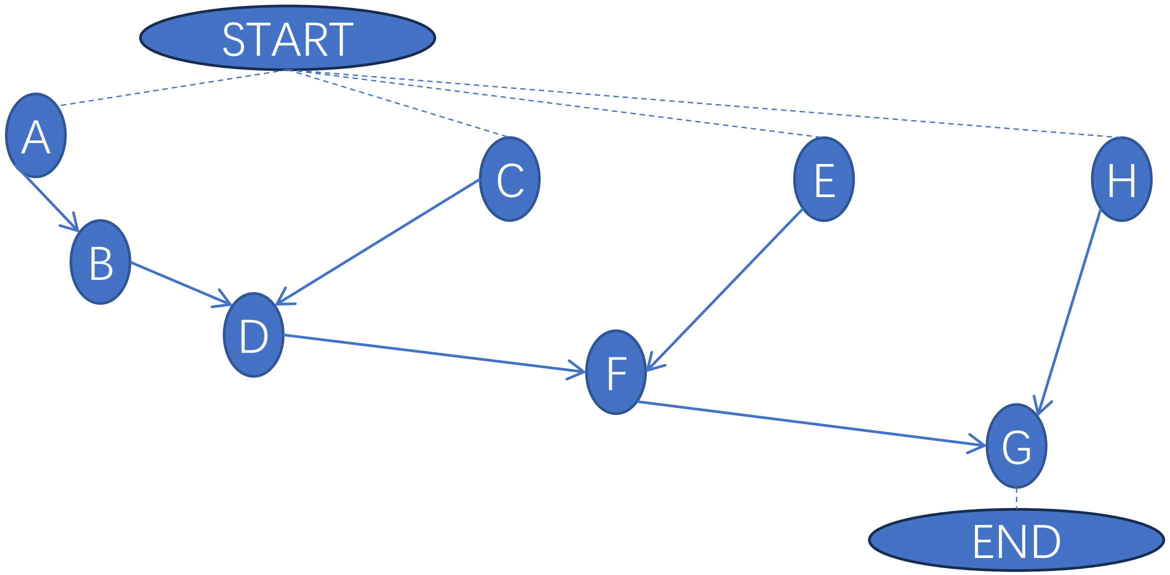
repeat {

Select a vertex v_i whose predecessors are all scheduled;

Schedule v_i by setting $t_i = \max \; t_j + d_j$;}

until (v_n is scheduled);

return (G);}



START 0

A

ASAP ($G_s(V,E)$) {

Schedule v_0 by setting $t_0 = 0$;

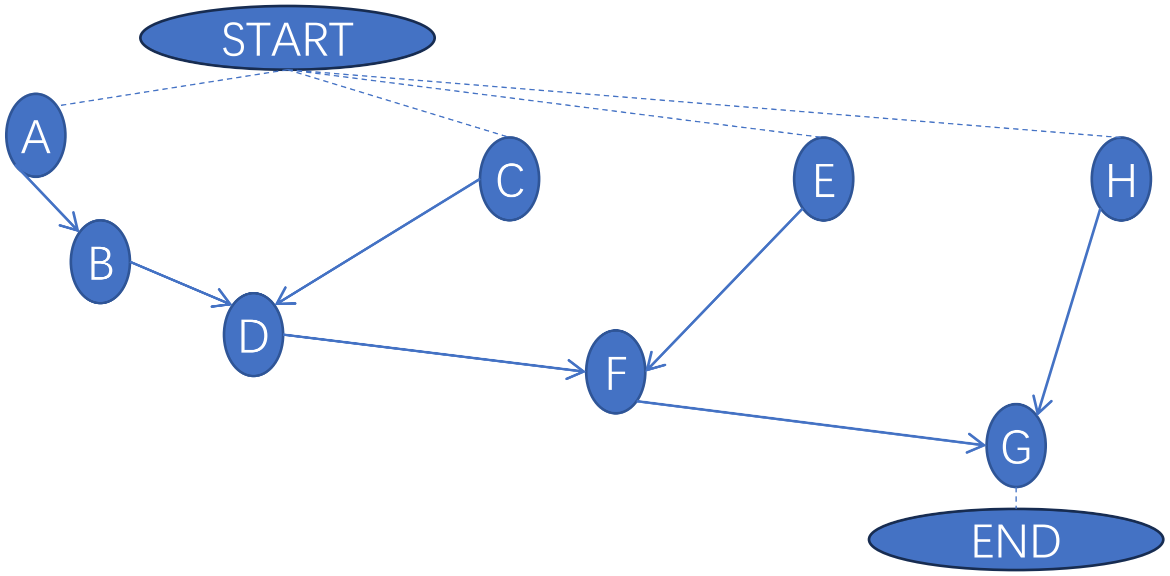
repeat {

Select a vertex v_i whose predecessors are all scheduled;

Schedule v_i by setting $t_i = \max \quad t_j + d_j$;}

until (v_n is scheduled);

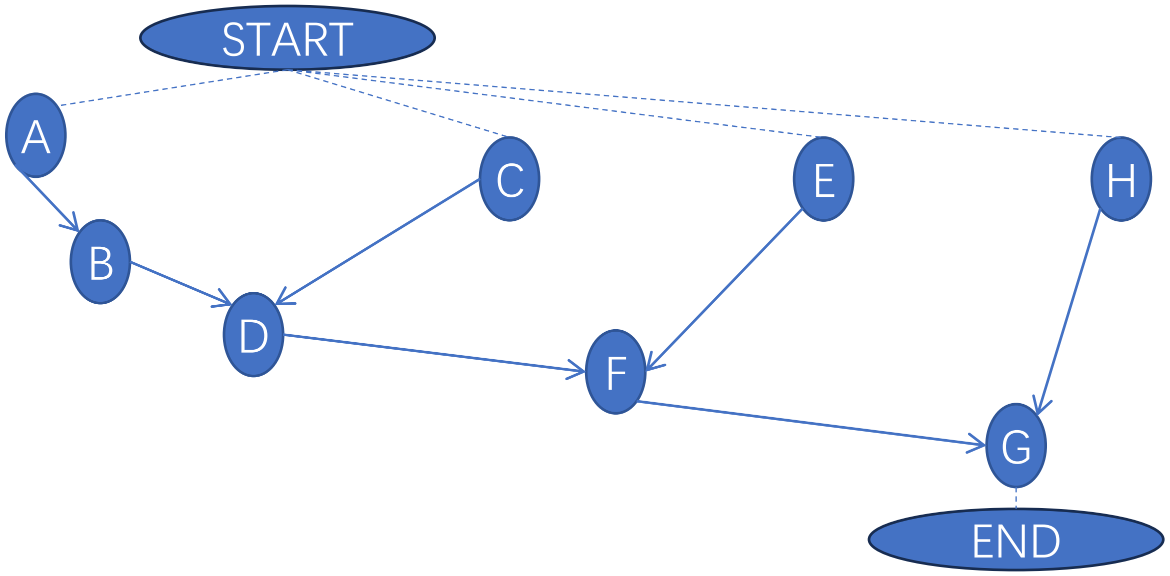
return (G);}



START 0

A 1

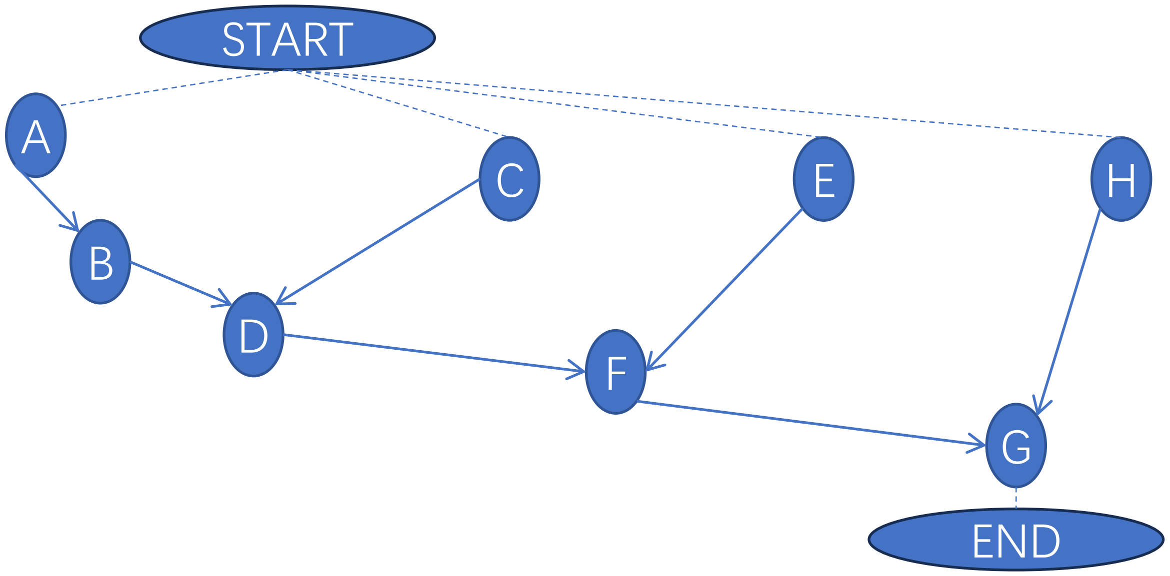
```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
        until (v_n is scheduled);
        return ( G );}
```



START 0

A 1

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```

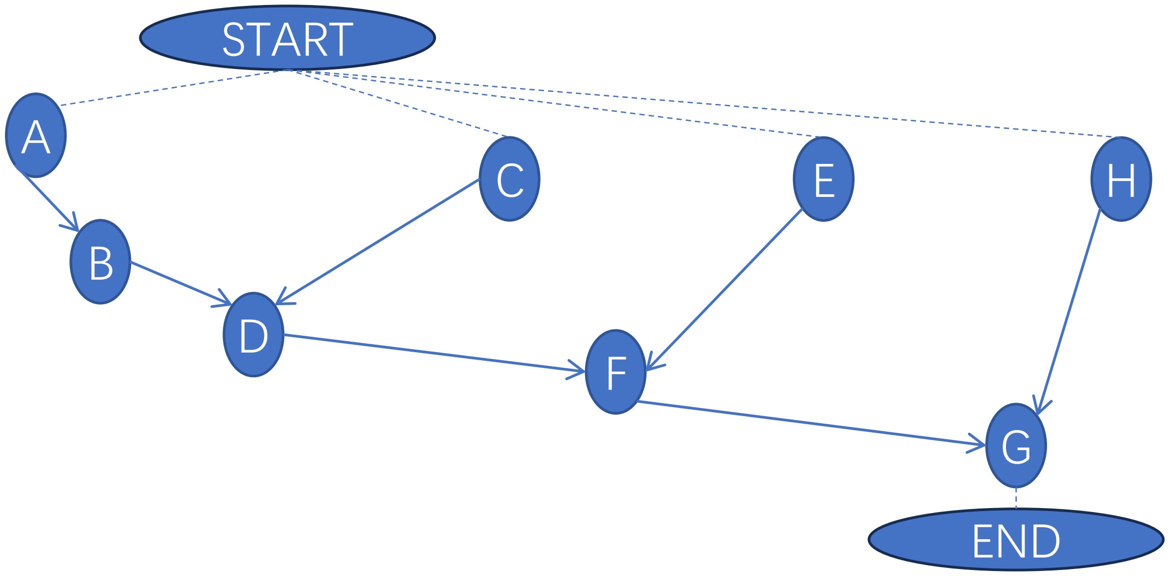


START 0

A 1

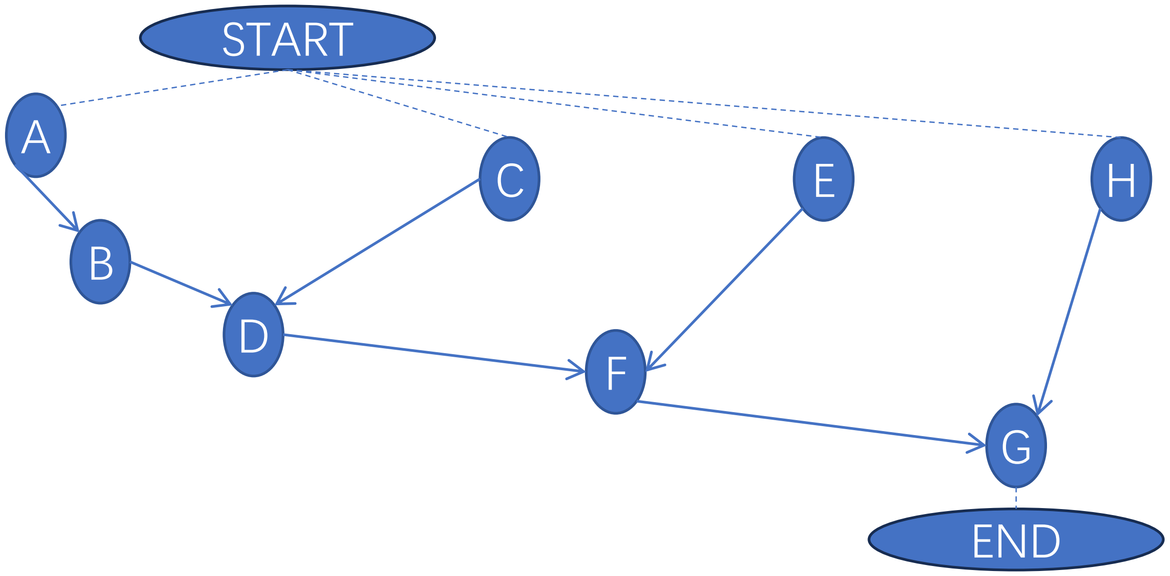
B

```
ASAP ( G_s(V,E) ) {
      Schedule v_0 by setting t_0 = 0;
      repeat {
            Select a vertex v_i whose predecessors are all scheduled;
            Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
      until (v_n is scheduled);
      return ( G );}
```



START 0
A 1
B 2

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```

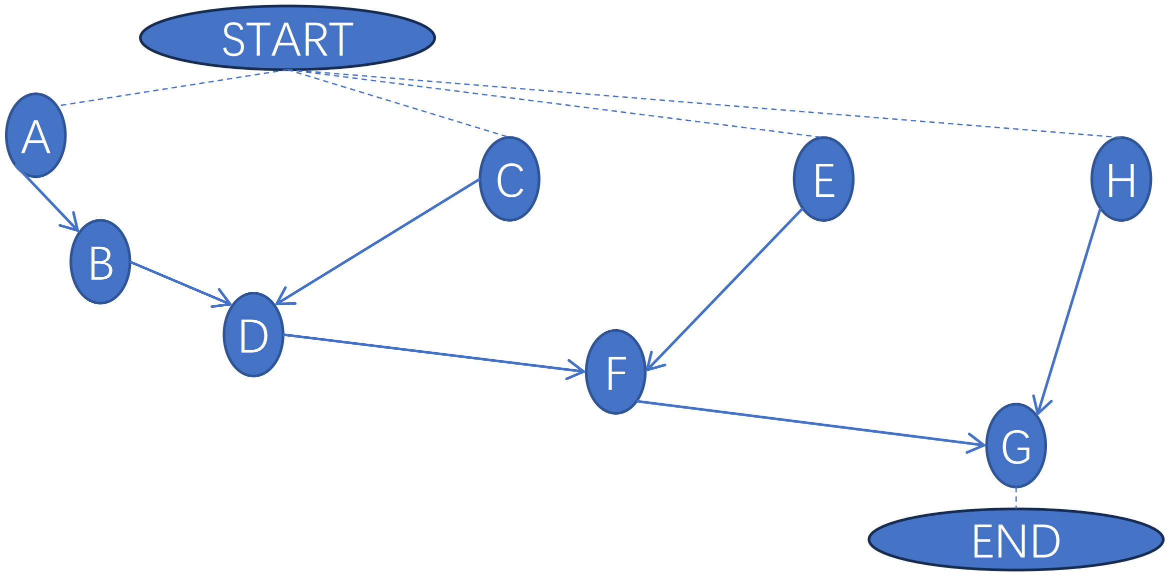


START 0

A 1

B 2

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting t_i = max t_j + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```



START 0
A 1
B 2
C

ASAP ($G_s(V,E)$) {

Schedule v_0 by setting $t_0 = 0$;

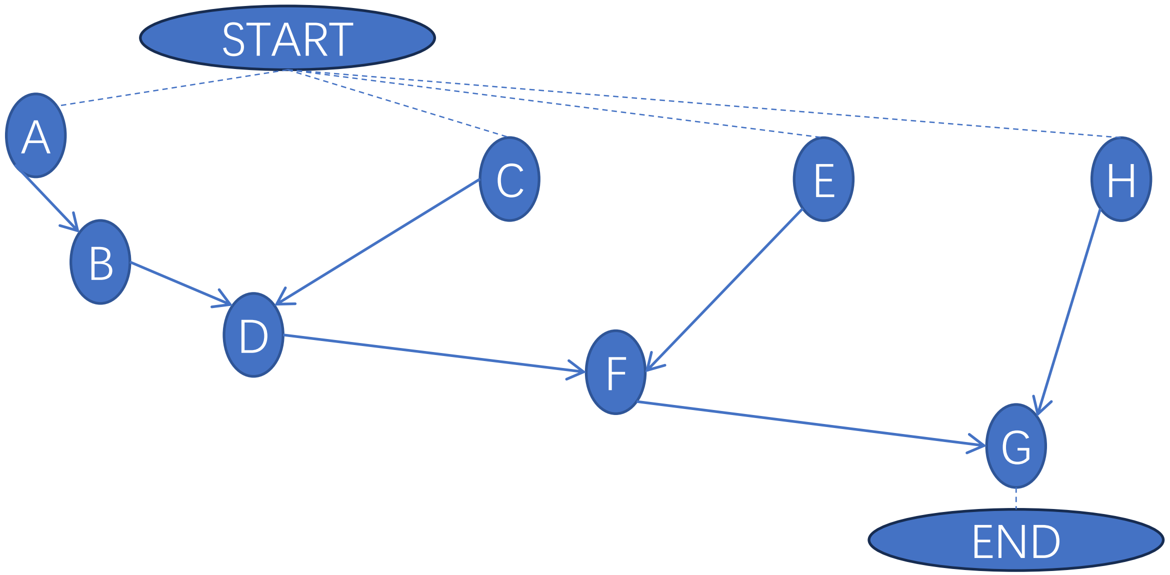
repeat {

Select a vertex v_i whose predecessors are all scheduled;

Schedule v_i by setting $t_i = \max \; t_j + d_j$;}

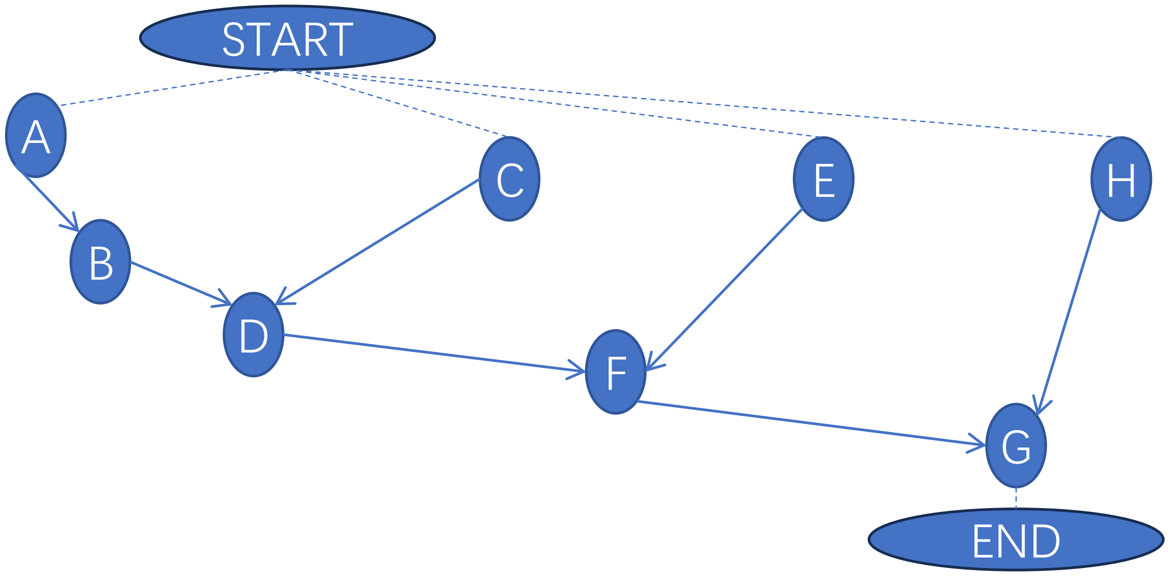
until (v_n is scheduled);

return (G);}



START 0
A 1
B 2
C 1

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```



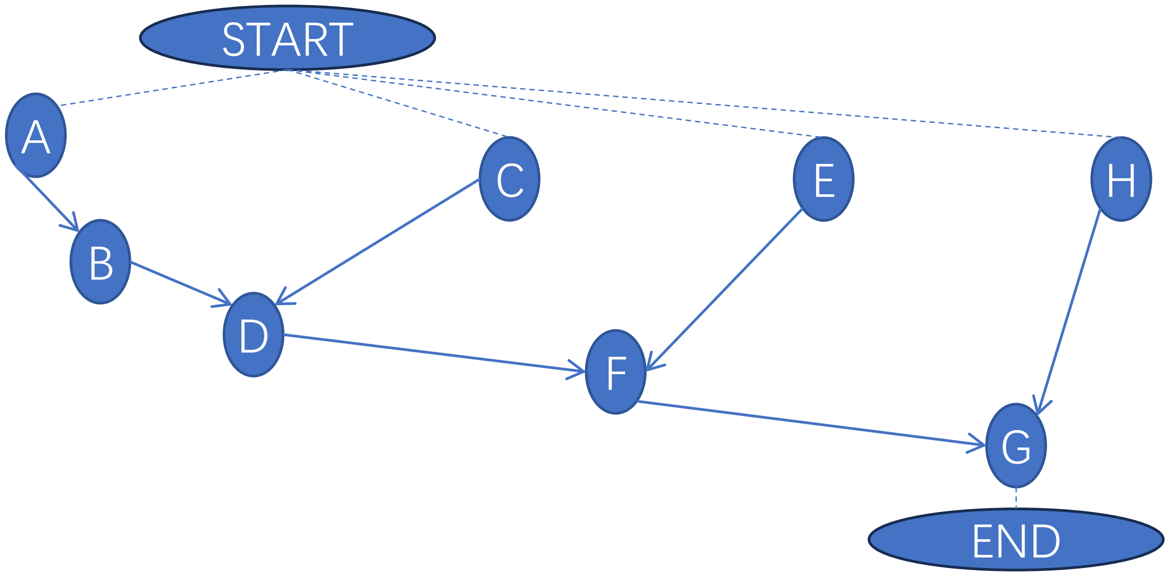
START 0

A 1

B 2

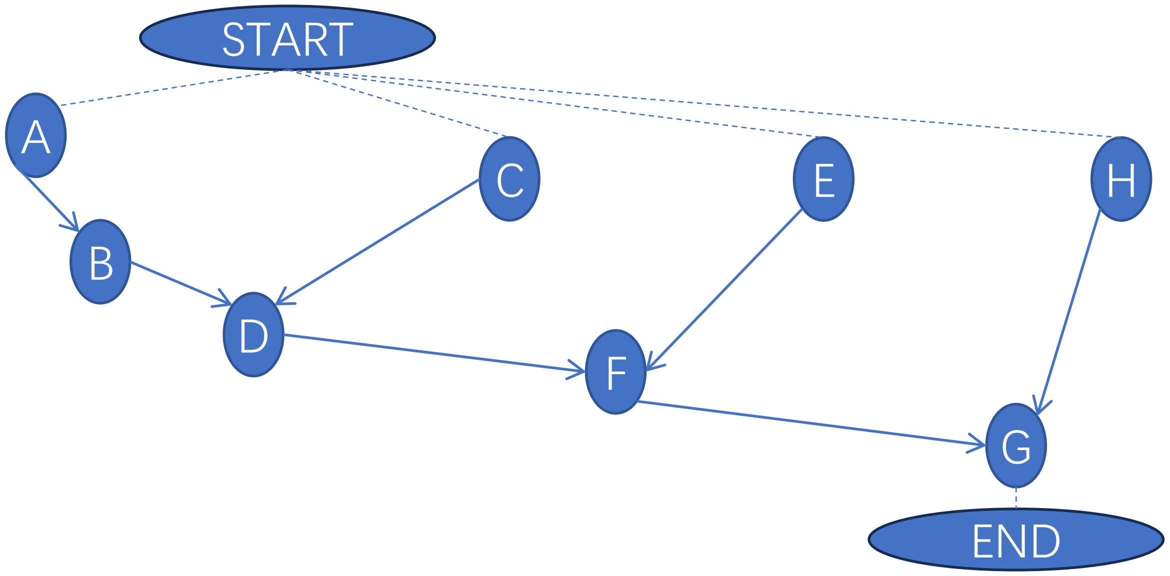
C 1

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```



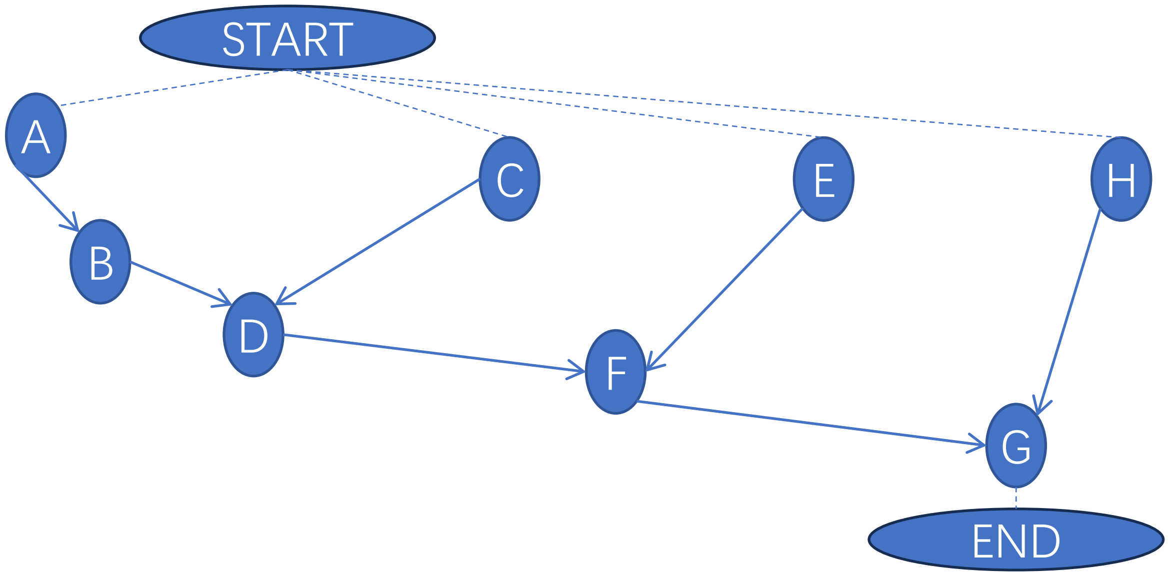
START 0
A 1
B 2
C 1
D

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
        until (v_n is scheduled);
        return ( G );}
```



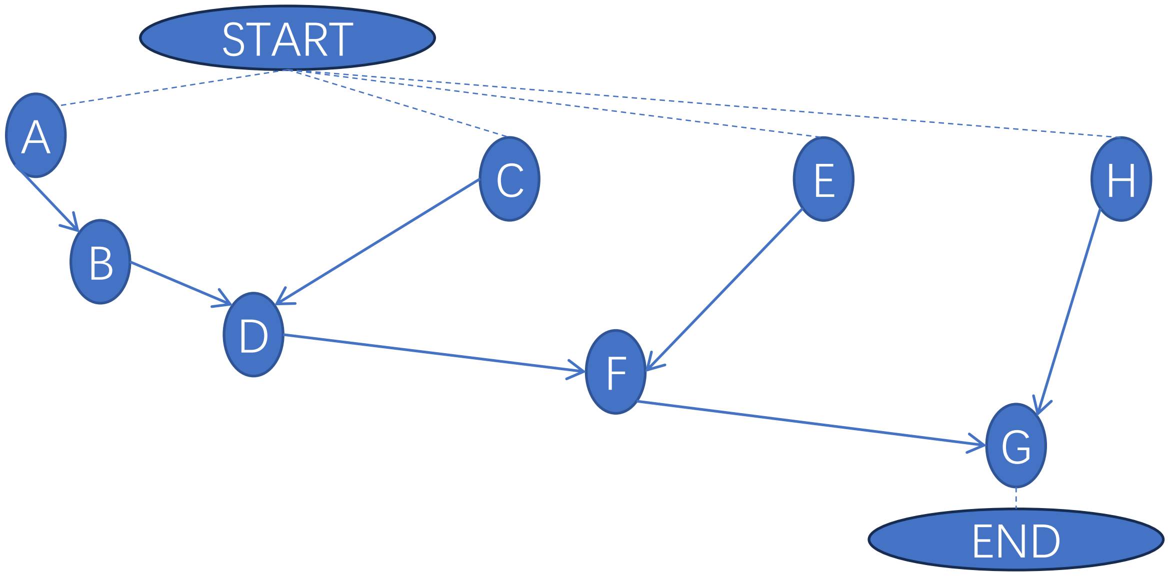
START 0
A 1
B 2
C 1
D 3

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
        until (v_n is scheduled);
        return ( G );}
```



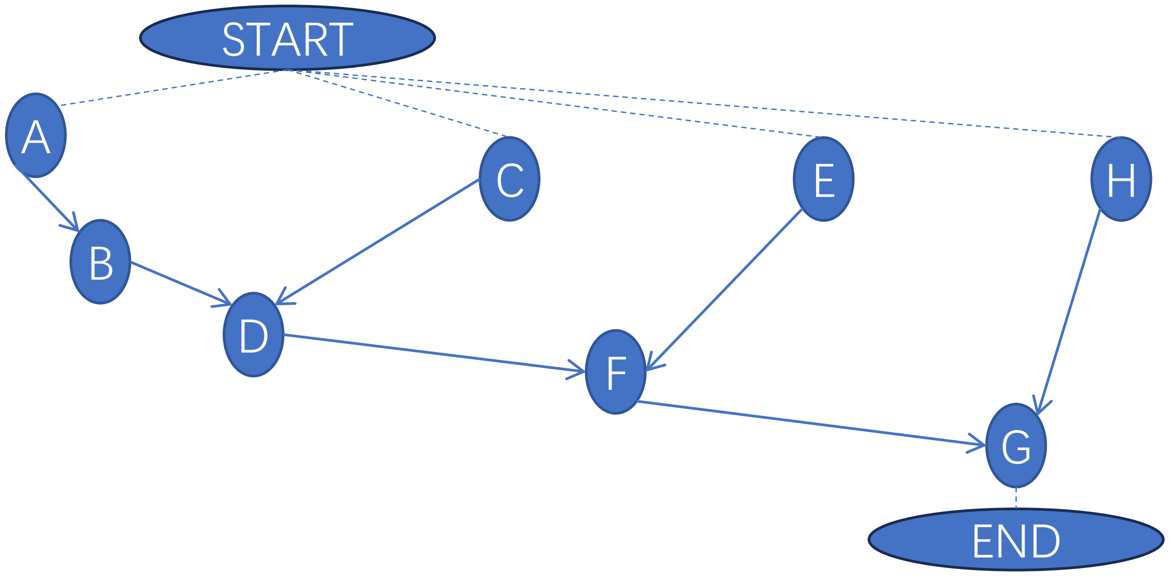
START 0
A 1
B 2
C 1
D 3

ASAP ($G_s(V,E)$) {
    Schedule v_0 by setting $t_0 = 0$;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting $t_i = \max \ t_j + d_j$;}
    until (v_n is scheduled);
    return (G);}



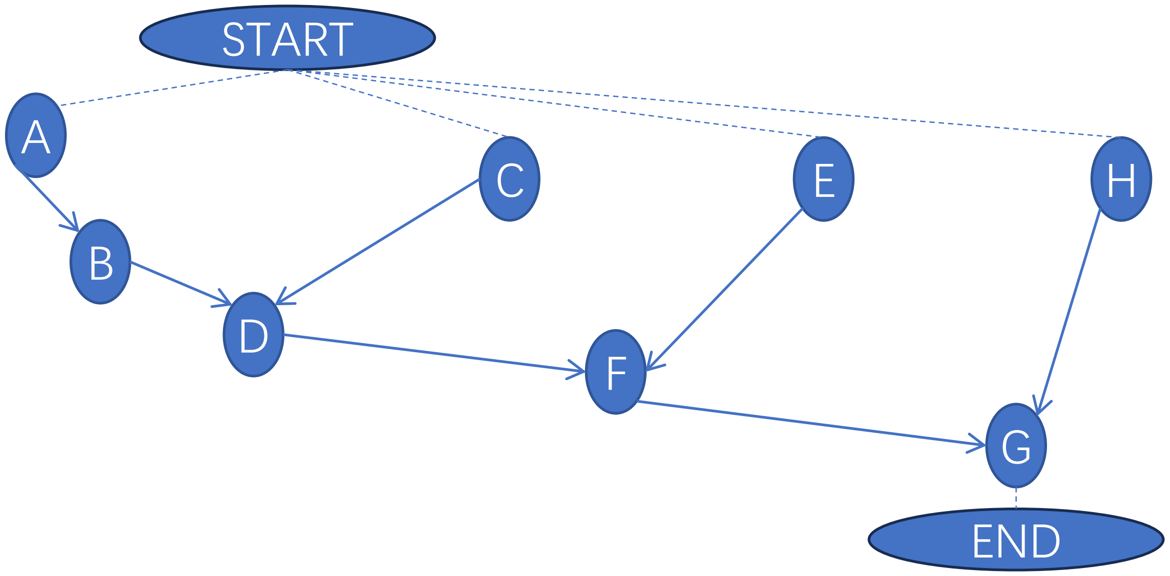
START 0
A 1
B 2
C 1
D 3
E

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```



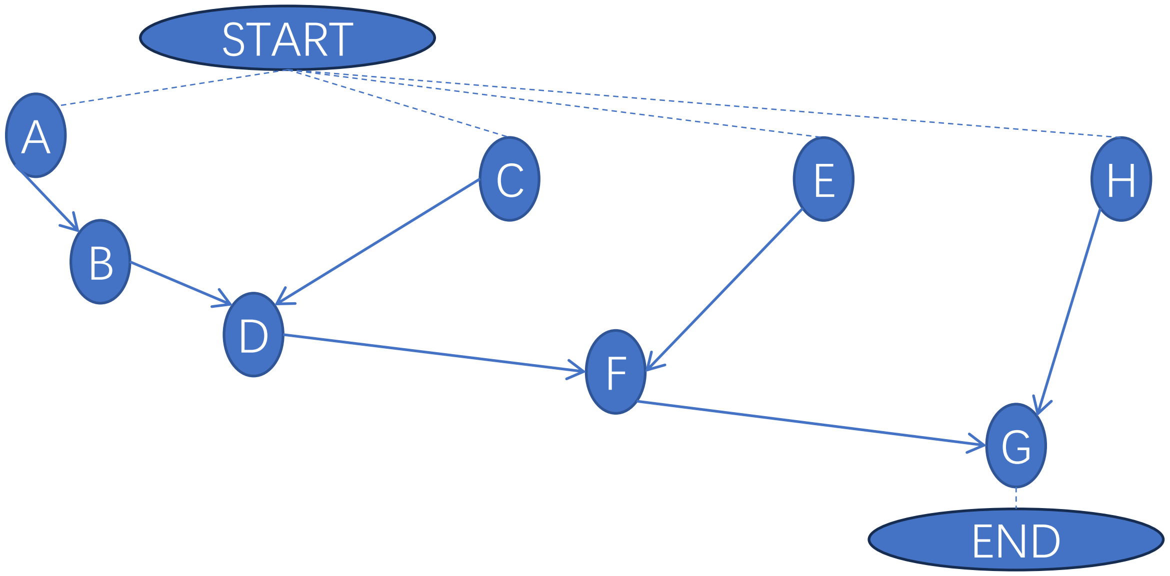
START 0
A 1
B 2
C 1
D 3
E 1

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
        until (v_n is scheduled);
        return ( G );}
```



START 0
A 1
B 2
C 1
D 3
E 1

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```



START 0
A 1
B 2
C 1
D 3
E 1
F

1 ASAP ($G_s(V,E)$) {

2 Schedule v_0 by setting $t_0 = 0$;

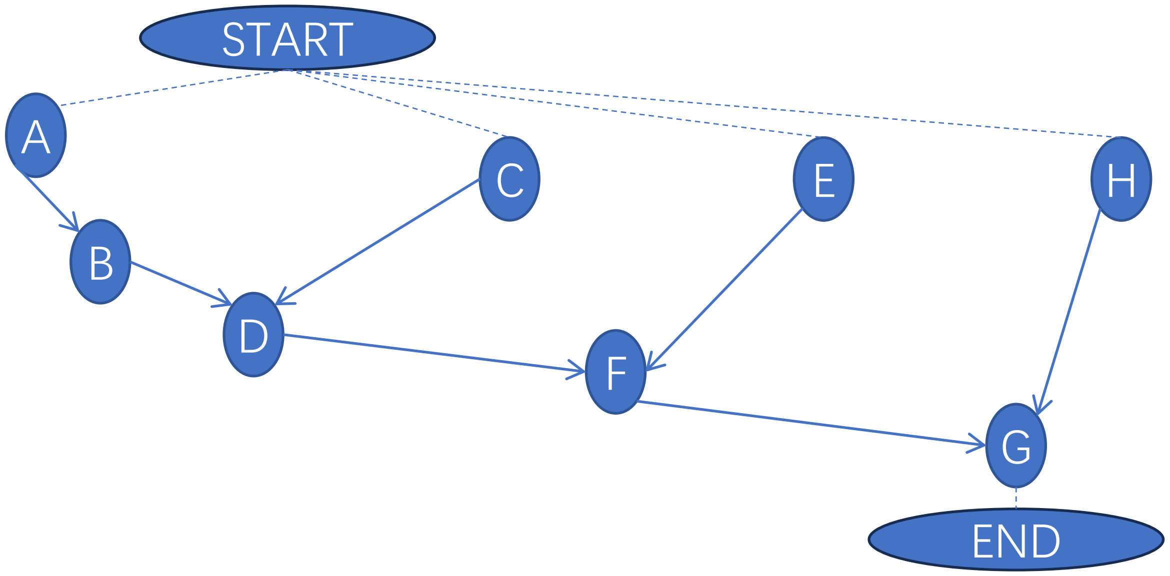
3 repeat {

4 Select a vertex v_i whose predecessors are all scheduled;

5 Schedule v_i by setting $t_i = \max \; t_j + d_j$;}

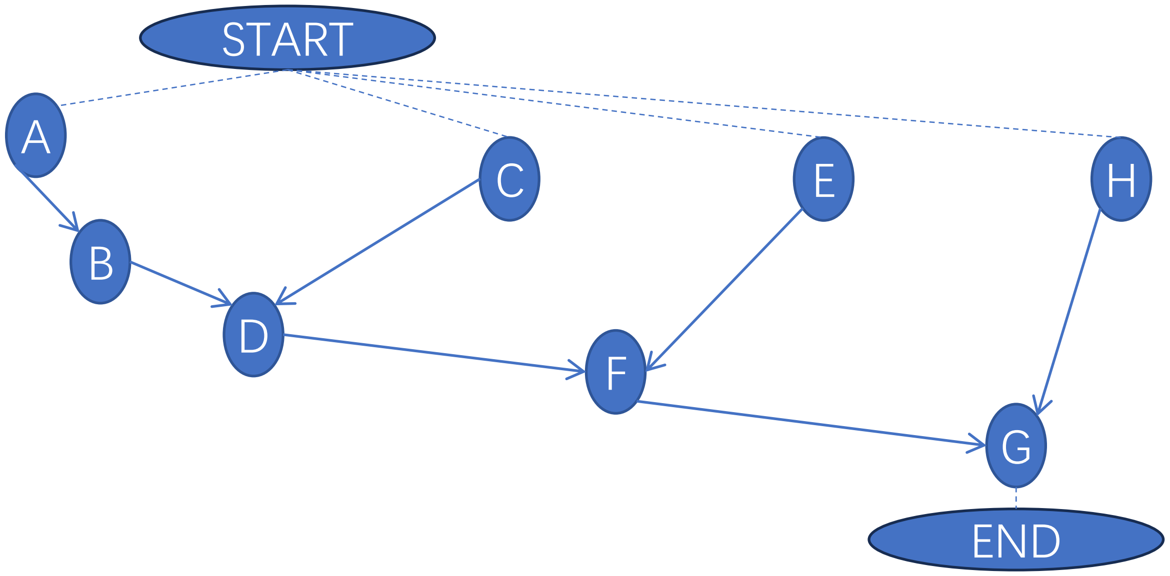
7 until (v_n is scheduled);

8 return (G);}



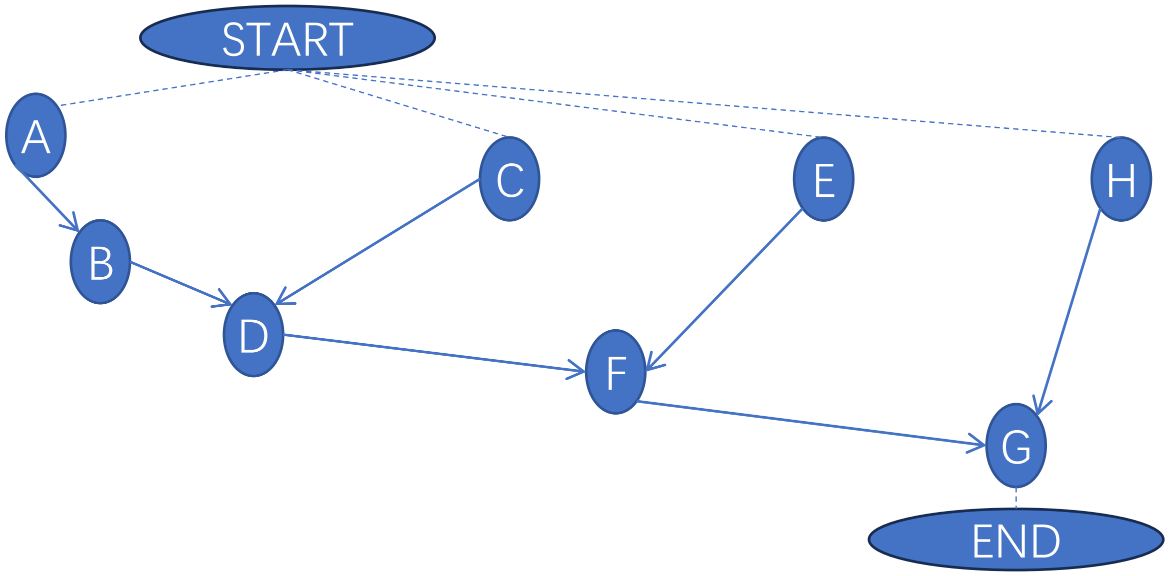
START 0
A 1
B 2
C 1
D 3
E 1
F 4

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
        until (v_n is scheduled);
        return ( G );}
```



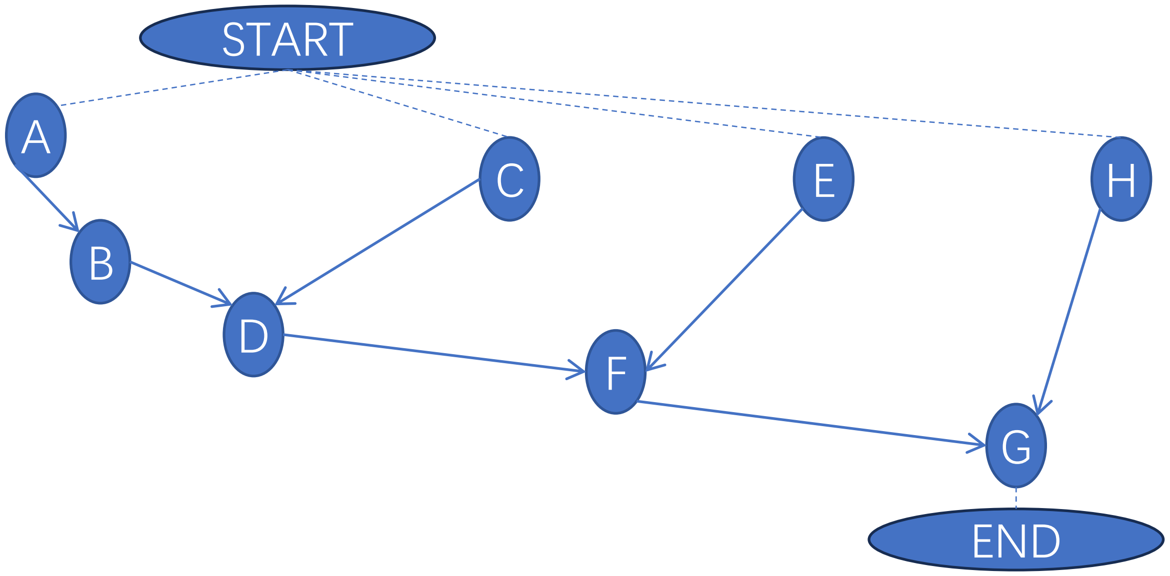
START 0
A 1
B 2
C 1
D 3
E 1
F 4

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```



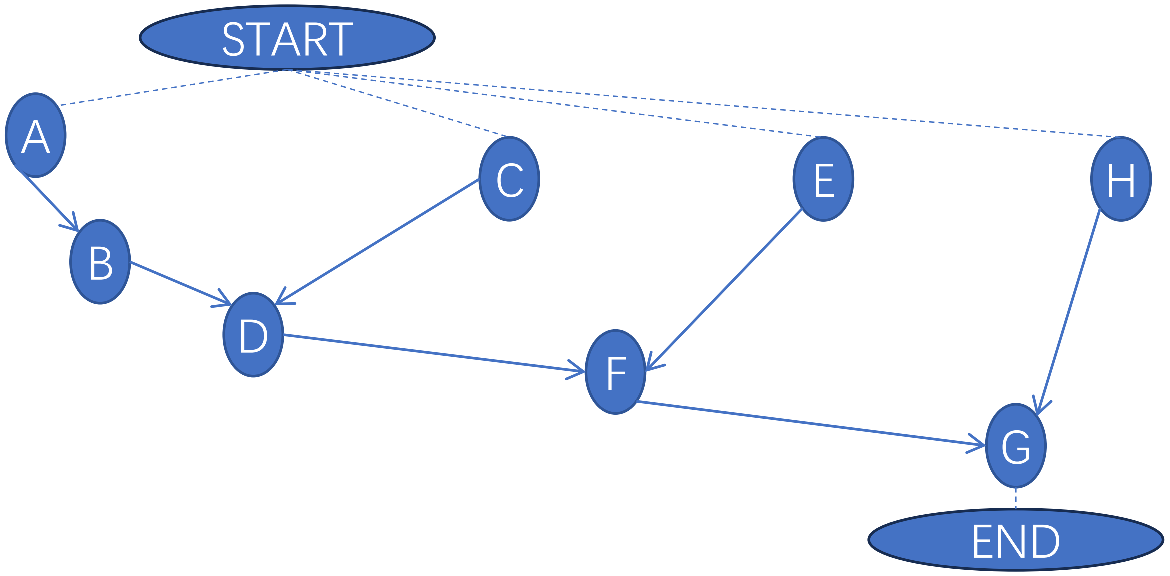
START 0
A 1
B 2
C 1
D 3
E 1
F 4
H

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
        until (v_n is scheduled);
        return ( G );}
```



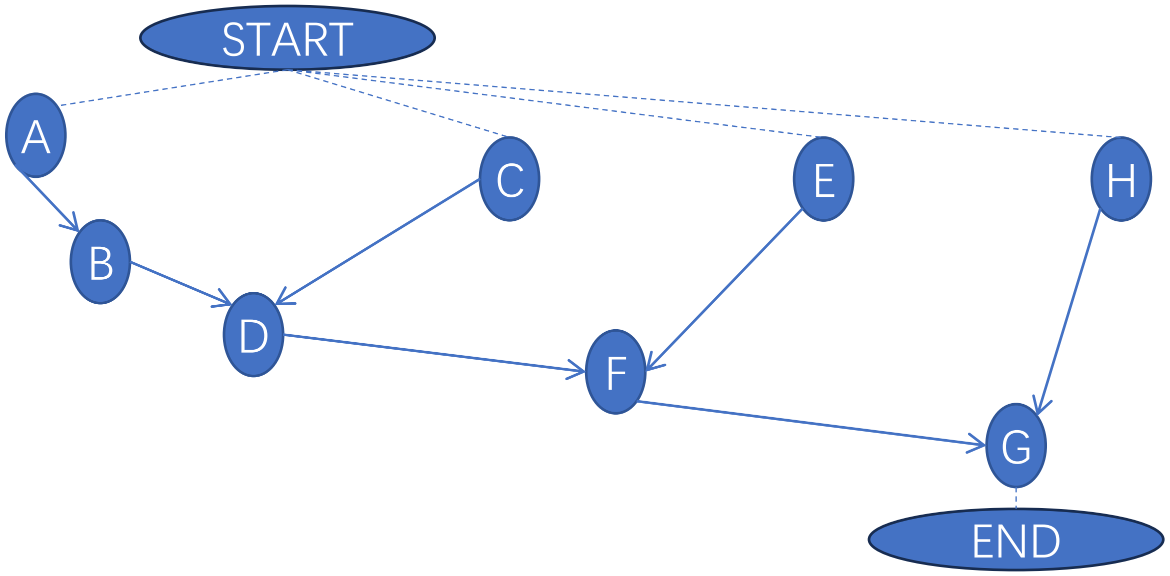
START 0
A 1
B 2
C 1
D 3
E 1
F 4
H 1

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
        until (v_n is scheduled);
        return ( G );}
```



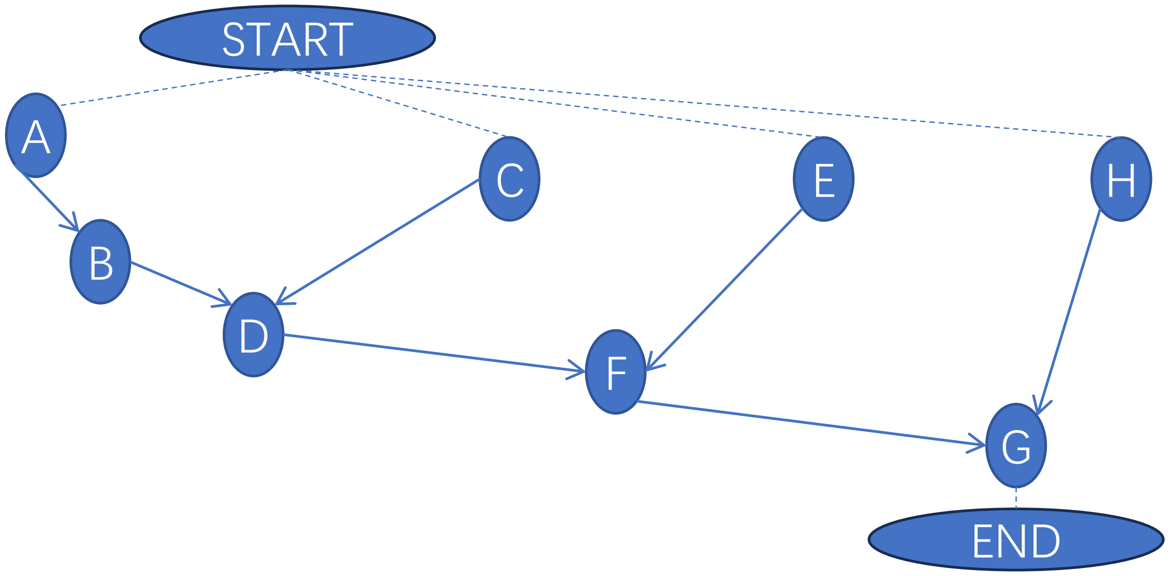
START 0
A 1
B 2
C 1
D 3
E 1
F 4
H 1

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```



START 0
A 1
B 2
C 1
D 3
E 1
F 4
H 1
G

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;}
        until (v_n is scheduled);
        return ( G );}
```

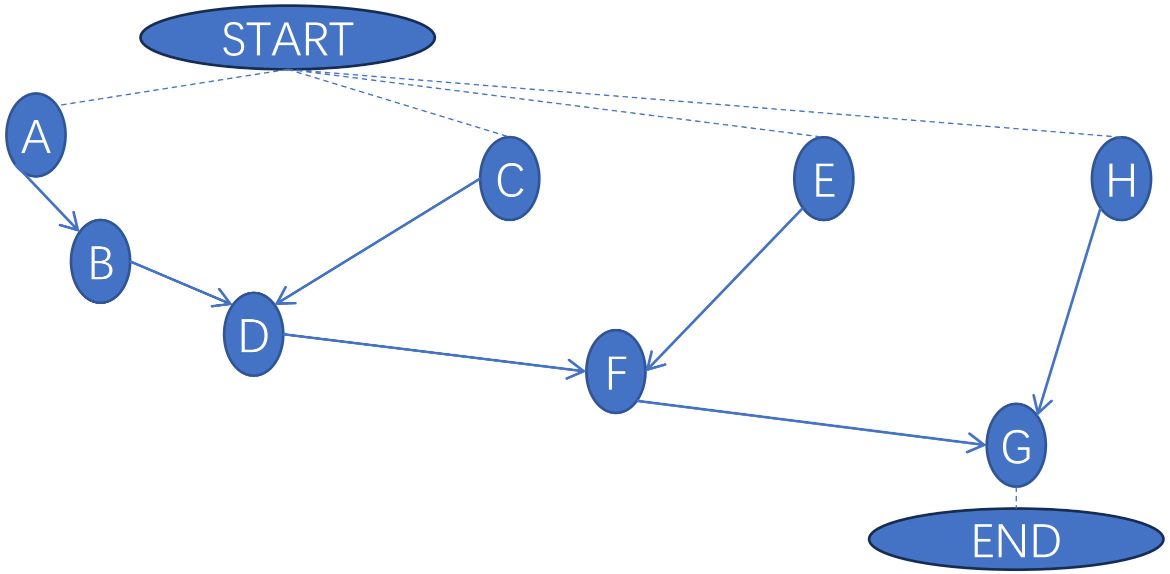


START 0
A 1
B 2
C 1
D 3
E 1
F 4
H 1
G 5
END 6

```
ASAP ( G_s(V,E) ) {
    Schedule v_0 by setting t_0 = 0;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting t_i = max t_j + d_j ;}
    until (v_n is scheduled);
    return ( G );}
```

ASAP调度算法

ASAP scheduling algorithm



10	START A
START 1	START C
A 1	START E
B 1	START G
C 1	A B
D 1	C D
E 1	B D
F 1	E F
G 1	D F
H 1	H G
END 1	F G
	G END

ASAP调度算法

ASAP scheduling algorithm

```
class Node:
    def __init__(self, name, duration):
        self.name = name
        self.duration = duration
        self.is_scheduled = False
        self.start_time = -1
        self.predecessors = []
```

```
node=Node(START, 1)
```

name	START
duration	1
Is_scheduled	False
start_time	-1
predecessors	[]

node

10
START 1
A 1
B 1
C 1
D 1
E 1
F 1
G 1
H 1
END 1

START A
START C
START E
START G
A B
C D
B D
E F
D F
H G
F G
G END

ASAP调度算法

ASAP scheduling algorithm

```python
def read_graph_from_file(input_file):
    with open(input_file, "r", encoding="utf-8") as f:
        N = int(f.readline().strip())  # 读取节点数
        nodes = []
        # 读取节点信息
        for _ in range(N):
            parts = f.readline().strip().split()
            node_name = parts[0]
            duration = int(parts[1])
            nodes.append(Node(node_name, duration))
```

```
10
START 1
A 1
B 1
C 1
D 1
E 1
F 1
G 1
H 1
END 1
```

```
START A
START C
START E
START G
A B
C D
B D
E F
D F
H G
F G
G END
```

ASAP调度算法

ASAP scheduling algorithm

```python
def read_graph_from_file(input_file):
    with open(input_file, "r", encoding="utf-8") as f:
        N = int(f.readline().strip())  # 读取节点数
        nodes = []
        # 读取节点信息
        for _ in range(N):
            parts = f.readline().strip().split()
            node_name = parts[0]
            duration = int(parts[1])
            nodes.append(Node(node_name, duration))
```

0	name	START
	duration	1
	Is_scheduled	False
	start_time	-1
	predecessors	[]
1	name	A
	duration	1
	Is_scheduled	False
	start_time	-1
	predecessors	[]
2	…	
…		

nodes

ASAP调度算法

```python
def read_graph_from_file(input_file):
    with open(input_file, "r", encoding="utf-8") as f:
        N = int(f.readline().strip())  # 读取节点数
        nodes = []
        edges = []
        # 读取节点信息 ...
        # 读取边信息并建立前驱关系
        for line in f:
            src, dst = line.strip().split()
            src_node = None
            dst_node = None
            for node in nodes:
                if node.name == src:
                    src_node = node
                if node.name == dst:
                    dst_node = node
            if dst_node and src_node:
                dst_node.predecessors.append(src_node)
```

```
10
START 1
A 1
B 1
C 1
D 1
E 1
F 1
G 1
H 1
END 1
```

```
START A
START C
START E
START G
A B
C D
B D
E F
D F
H G
F G
G END
```

ASAP调度算法

```python
def read_graph_from_file(input_file):
    with open(input_file, "r", encoding="utf-8") as f:
        N = int(f.readline().strip())  # 读取节点数
        nodes = []
        # 读取节点信息 ...
        # 读取边信息并建立前驱关系
        for line in f:
            src, dst = line.strip().split()
            src_node = None
            dst_node = None
            for node in nodes:
                if node.name == src:
                    src_node = node
                if node.name == dst:
                    dst_node = node
            if dst_node and src_node:
                dst_node.predecessors.append(src_node)
```

0	name	START
	duration	1
	Is_scheduled	False
	start_time	-1
	predecessors	[]
1	name	A
	duration	1
	Is_scheduled	False
	start_time	-1
	predecessors	[nodes[0]]
2	…	
…		

nodes

ASAP调度算法

ASAP scheduling algorithm

ASAP ($G_s(V,E)$) {

    Schedule v_0 by setting $t_0 = 0$;

    repeat {

        Select a vertex v_i whose predecessors are all scheduled;

        Schedule v_i by setting $t_i = \max \; t_j + d_j$;

    }

    until (v_n is scheduled);

    return (G);

}

ASAP调度算法

ASAP scheduling algorithm

```python
def asap_algorithm(nodes):
    # 1. 令 `START` 先调度
    start_node = next(node for node in nodes if node.name == "START")
    start_node.is_scheduled = True
    start_node.start_time = 0
```

ASAP调度算法

ASAP scheduling algorithm

ASAP ($G_s(V,E)$) {

  Schedule v_0 by setting $t_0 = 0$;

  repeat {

    Select a vertex v_i whose predecessors are all scheduled;

    Schedule v_i by setting $t_i = \max \; t_j + d_j$;

  }

  until (v_n is scheduled);

  return (G);

}

```python
def asap_algorithm(nodes):
    # 1. 令 `START` 先调度
    start_node = next(node for node in nodes if node.name == "START")
    start_node.is_scheduled = True
    start_node.start_time = 0
    # 2. 直到 `END` 也被调度
    end_node = next(node for node in nodes if node.name == "END")
    while not end_node.is_scheduled:
        # 找到一个满足条件的节点
        node_to_schedule = find_suitable_node(nodes)
        # 决定该节点的开始时间
        schedule_node(node_to_schedule)
```

ASAP调度算法

ASAP scheduling algorithm

ASAP ($G_s(V,E)$) {
    Schedule v_0 by setting $t_0 = 0$;
    repeat {
        Select a vertex v_i whose predecessors are all scheduled;
        Schedule v_i by setting $t_i = \max \; t_j + d_j$;
    }
    until (v_n is scheduled);
    return (G);
}

```python
def find_suitable_node(nodes):
    for node in nodes:
        # 只对当前还没被调度的结点进行检查
        if not node.is_scheduled:
            # 检查所有前置结点是否已调度
            all_predecessors_scheduled = True
            for p in node.predecessors:
                if not p.is_scheduled:
                    all_predecessors_scheduled = False
                    break
            if all_predecessors_scheduled:
                return node
```

ASAP调度算法

ASAP scheduling algorithm

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;
        }
        until (v_n is scheduled);
        return ( G );
}
```

```python
def schedule_node(node):
    # 统计所有前置结点的结束时间
    pred_finish_times = []
    for pred in node.predecessors:
        finish_time = pred.start_time + pred.duration
        pred_finish_times.append(finish_time)
    # 设定当前节点的开始时间
    node.start_time = max(pred_finish_times)
    node.is_scheduled = True
```

```
ASAP ( G_s(V,E) ) {
        Schedule v_0 by setting t_0 = 0;
        repeat {
                Select a vertex v_i whose predecessors are all scheduled;
                Schedule v_i by setting  t_i =   max   t_j  + d_j ;
        }
        until (v_n is scheduled);
        return ( G );
}
```

```python
def asap_algorithm(nodes):
    start_node = next(node for node in nodes if node.name == "START")
    start_node.is_scheduled = True
    start_node.start_time = 0
    end_node = next(node for node in nodes if node.name == "END")
    while not end_node.is_scheduled:
        node_to_schedule = find_suitable_node(nodes)
        schedule_node(node_to_schedule, nodes)
    return nodes
```

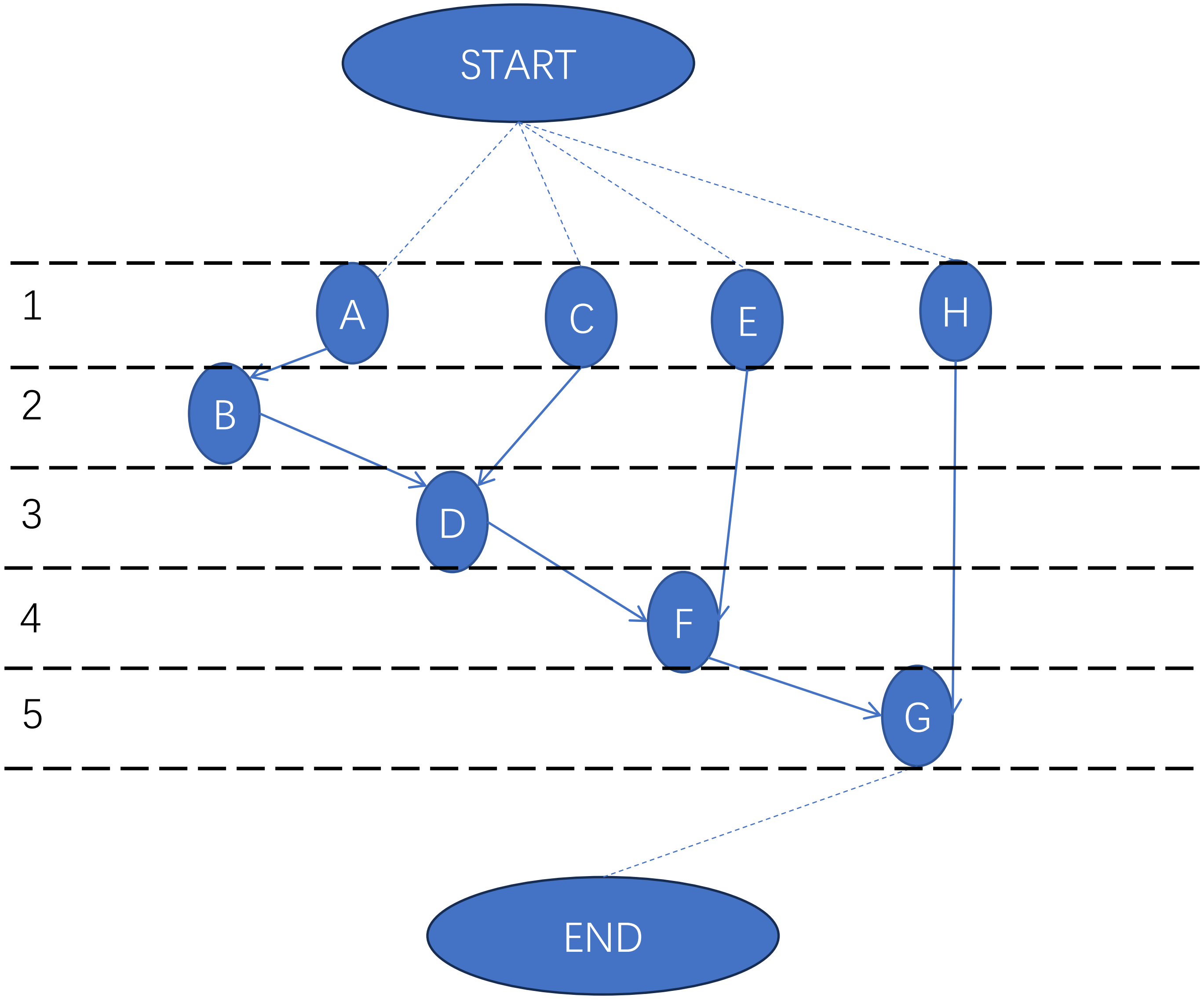
```python
def write_schedule_to_file(nodes, output_file):
    with open(output_file, 'w', encoding="utf-8") as f:
        for node in nodes:
            f.write(f"{node.name} {node.start_time}\n")
```

```python
input_file = "input.txt"
output_file = "output.txt"
nodes = read_graph_from_file(input_file)
nodes = asap_algorithm(nodes)
write_schedule_to_file(nodes, output_file)
```

START 0
A 1
B 2
C 1
D 3
E 1
F 4
H 1
G 5
END 6

ASAP的结果

输入文件内容:
10
START 1
A 1
B 1
C 1
D 1
E 1
F 1
G 1
H 1
END 0
START B
START A
START C
START E
START G
A B
C D
B D
E F
D F
H G
F G
B END
D END
F END
G END



为顶点决定开始周期的顺序:

START->

A->B->C->D->E->F->H->G

->END

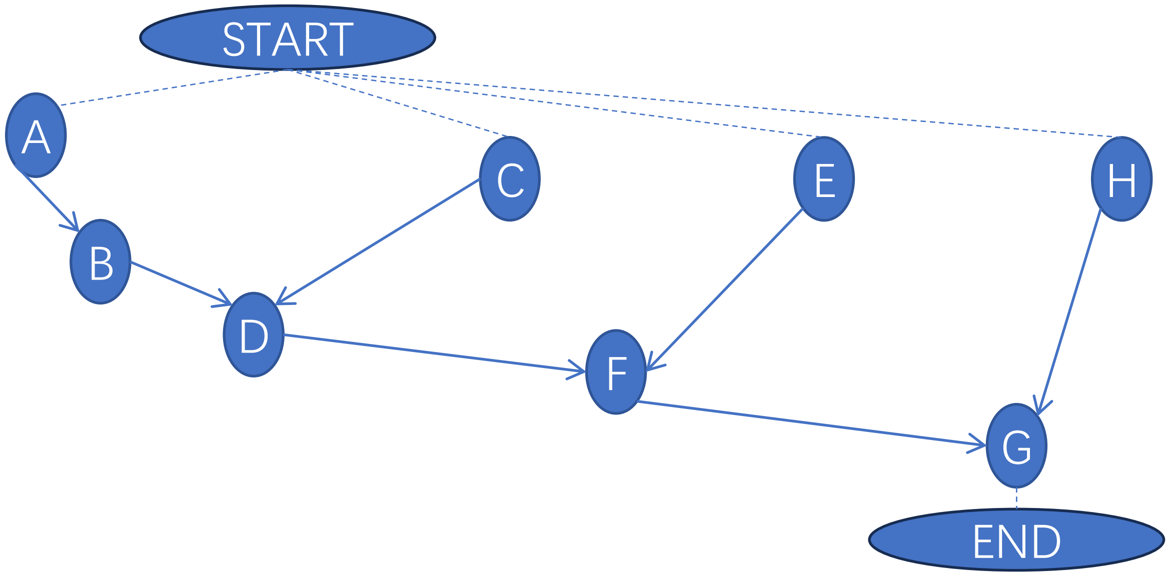
输出文件内容:
START 0
A 1
B 2
C 1
D 3
E 1
F 4
H 1
G 5
END 6

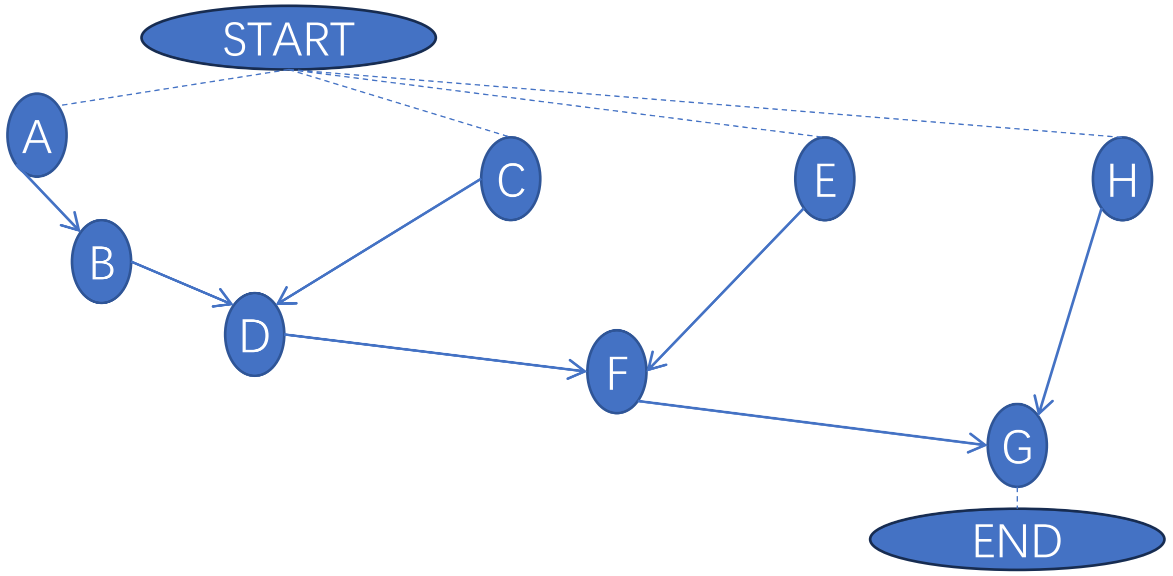
ALAP调度算法

ALAP调度算法（最晚时间约束）

ALAP scheduling algorithm（Latency Constrained）

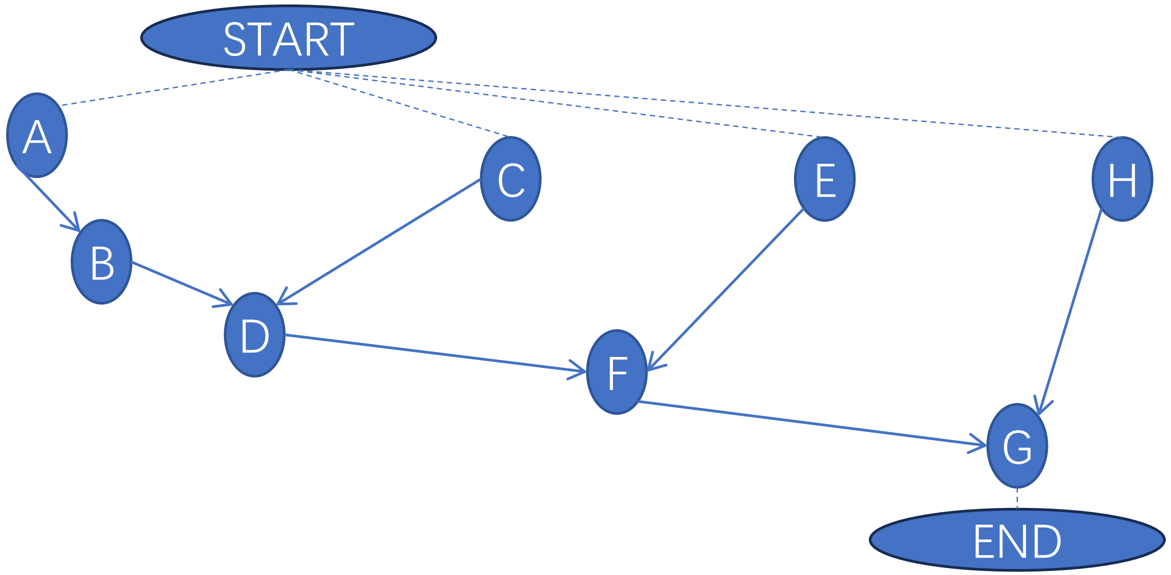
```
ALAP ( G_s(V,E), x ) {
        Schedule v_n by setting t_n = x+1 ;
        repeat {
                Select a vertex v_i whose successors are all scheduled;
                Schedule v_i by setting  t_i =   min   t_j  - d_i;
        }
        until (v_0 is scheduled);
        return (t);
}
```





END 6

```
ALAP ( G_s(V,E), x ) {
        Schedule v_n by setting t_n = x + 1;
        repeat {
                Select a vertex v_i whose successors are all scheduled;
                Schedule v_i by setting  t_i =   min   t_j  - d_i;}
        until (v_0 is scheduled);
        return (t);}
```



END 6

ALAP ($G_s(V,E)$, x) {

Schedule v_n by setting $t_n = x + 1$;

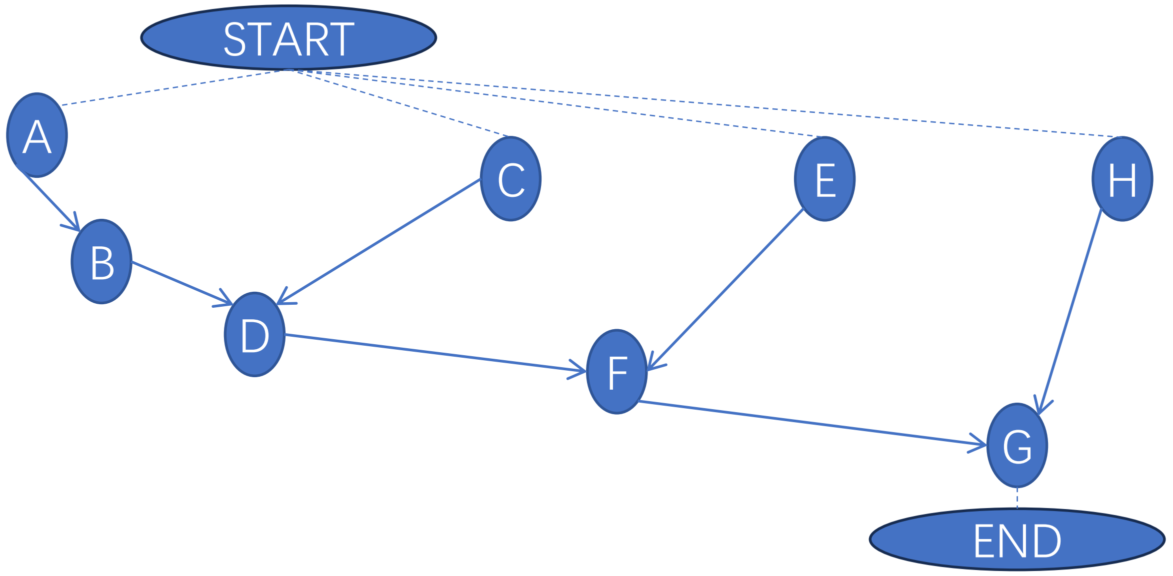
repeat {

Select a vertex v_i whose successors are all scheduled;

Schedule v_i by setting $t_i = \min \; t_j - d_i$;}

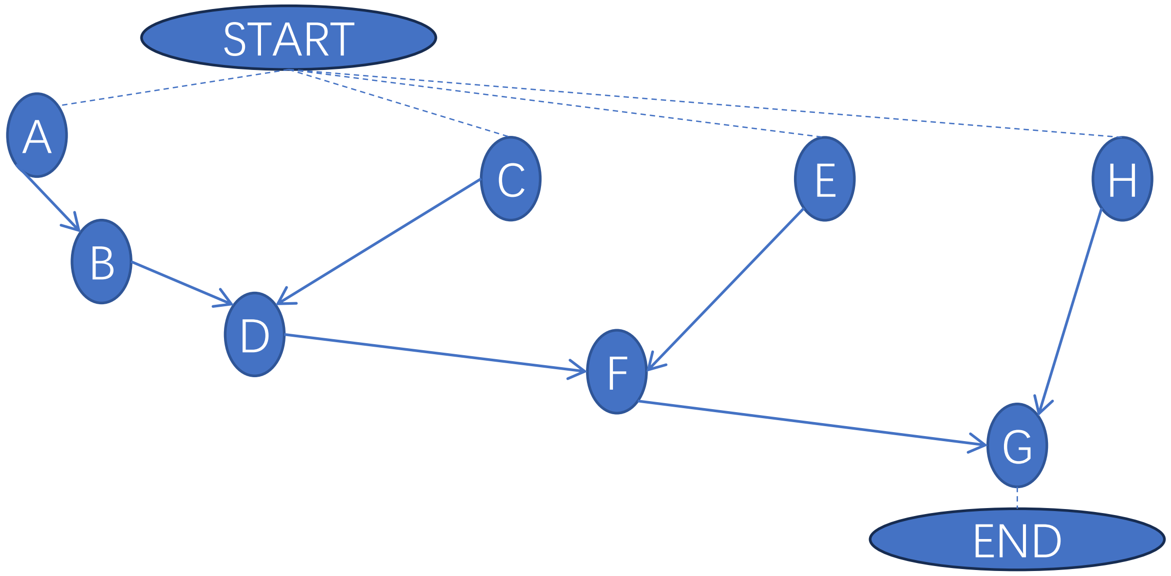
until (v_0 is scheduled);

return (t);}



END 6
G

```
ALAP ( G_s(V,E), x ) {
    Schedule v_n by setting t_n = x + 1;
    repeat {
        Select a vertex v_i whose successors are all scheduled;
        Schedule v_i by setting  t_i =   min   t_j  - d_i;}
    until (v_0 is scheduled);
    return (t);}
```



END 6

G 5

ALAP ($G_s(V,E)$, x) {

Schedule v_n by setting $t_n = x + 1$;

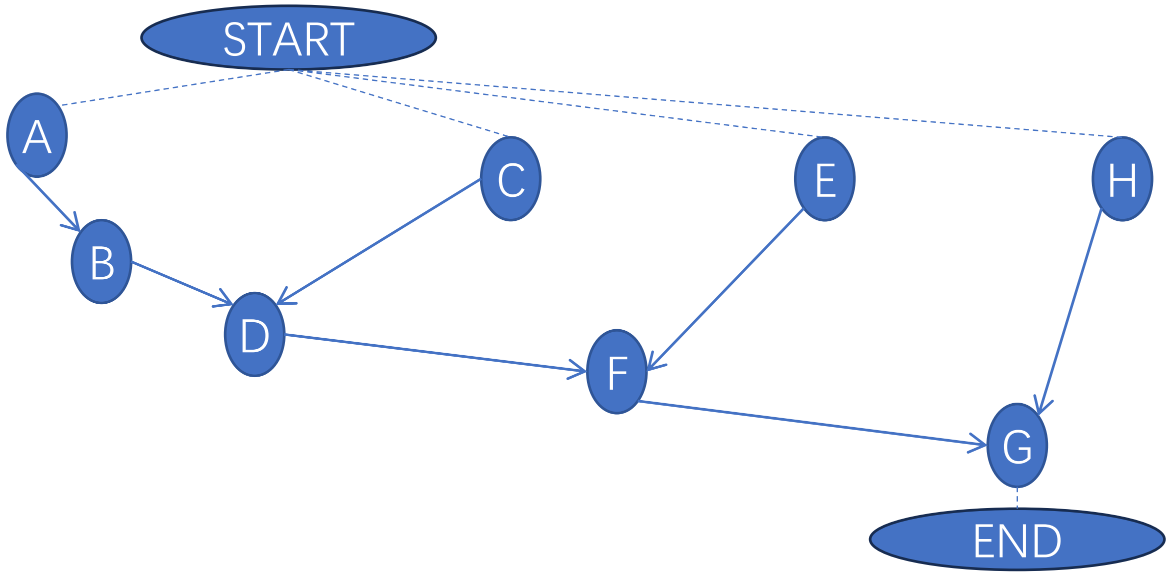
repeat {

Select a vertex v_i whose successors are all scheduled;

Schedule v_i by setting $t_i = \min \; t_j - d_i$;}

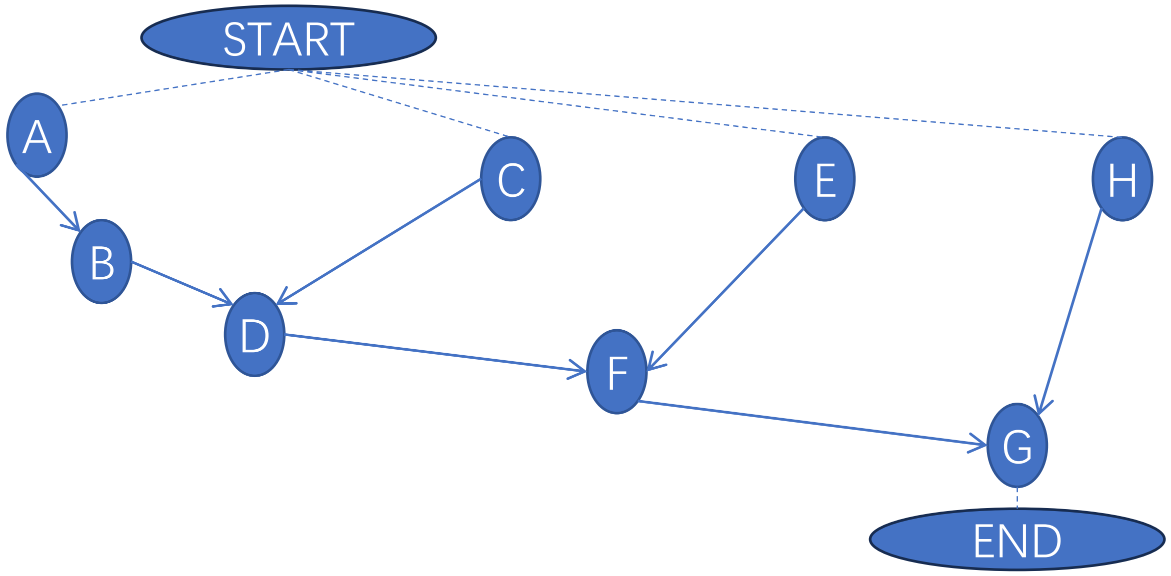
until (v_0 is scheduled);

return (t);}



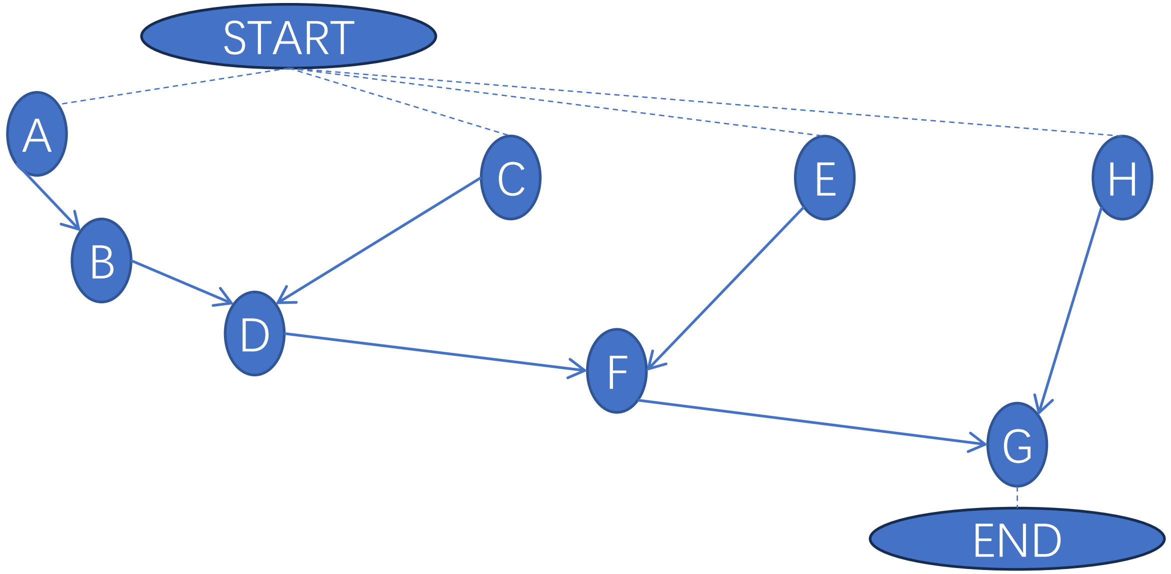
END 6
G 5
F

```
ALAP ( G_s(V,E), x ) {
      Schedule v_n by setting t_n = x + 1;
      repeat {
            Select a vertex v_i whose successors are all scheduled;
            Schedule v_i by setting  t_i =  min  t_j - d_i;}
      until (v_0 is scheduled);
      return (t);}
```



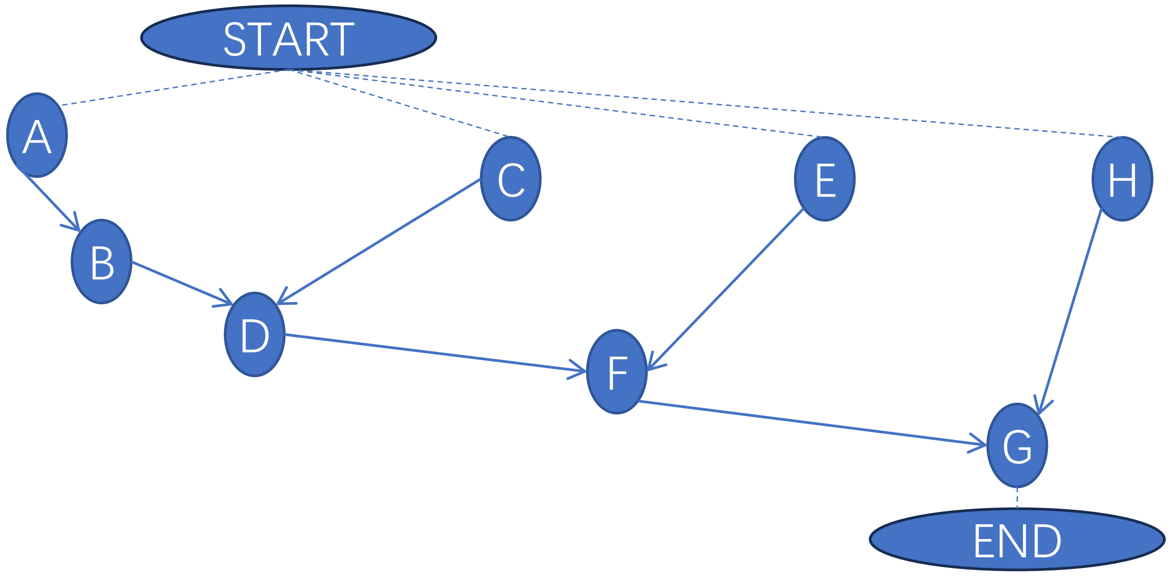
END 6
G 5
F 4

```
ALAP ( G_s(V,E), x ) {
    Schedule v_n by setting t_n = x + 1;
    repeat {
        Select a vertex v_i whose successors are all scheduled;
        Schedule v_i by setting  t_i =   min   t_j  - d_i;}
    until (v_0 is scheduled);
    return (t);}
```



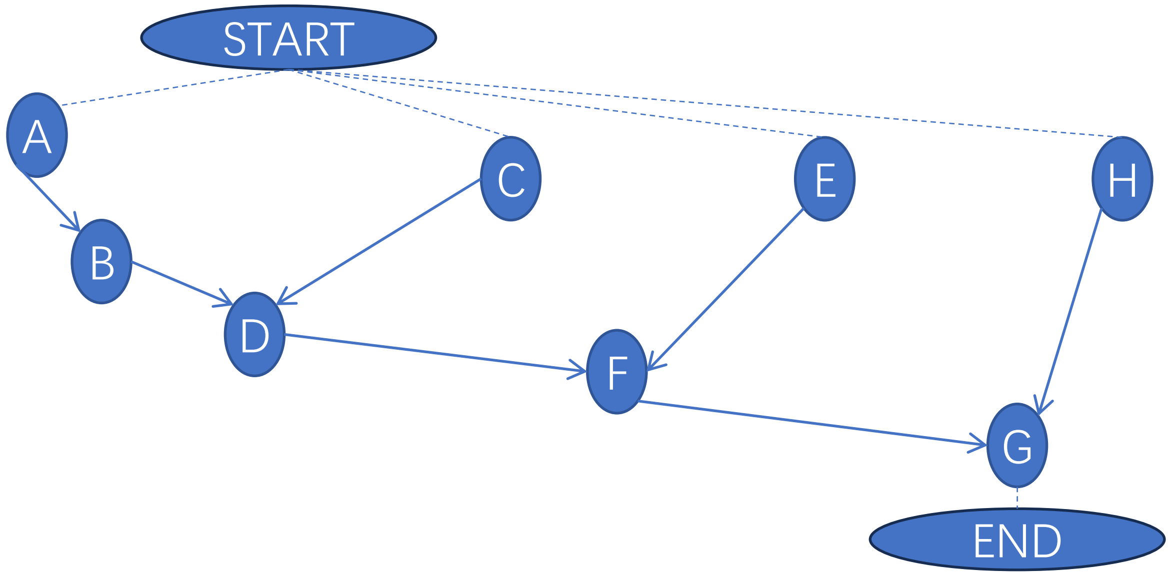
END 6
G 5
F 4
D

```
ALAP ( G_s(V,E), x ) {
      Schedule v_n by setting t_n = x + 1;
      repeat {
            Select a vertex v_i whose successors are all scheduled;
            Schedule v_i by setting  t_i =   min   t_j  - d_i;}
      until (v_0 is scheduled);
      return (t);}
```



END 6
G 5
F 4
D 3

```
ALAP ( G_s(V,E), x ) {
        Schedule v_n by setting t_n = x + 1;
        repeat {
                Select a vertex v_i whose successors are all scheduled;
                Schedule v_i by setting  t_i =   min   t_j  - d_i;}
        until (v_0 is scheduled);
        return (t);}
```



END 6
G 5
F 4
D 3
B

ALAP ($G_s(V,E)$, x) {

Schedule v_n by setting $t_n = x + 1$;

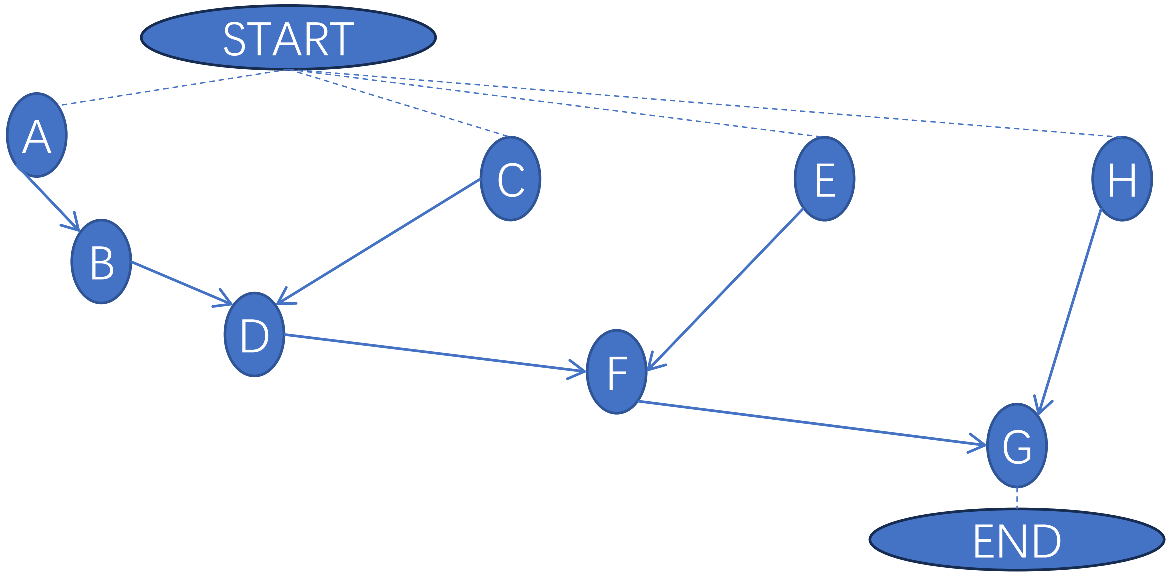
repeat {

Select a vertex v_i whose successors are all scheduled;

Schedule v_i by setting $t_i = \min \ t_j - d_i$;}

until (v_0 is scheduled);

return (t);}



END 6

G 5

F 4

D 3

B 2

ALAP ($G_s(V,E)$, x) {

Schedule v_n by setting $t_n = x + 1$;

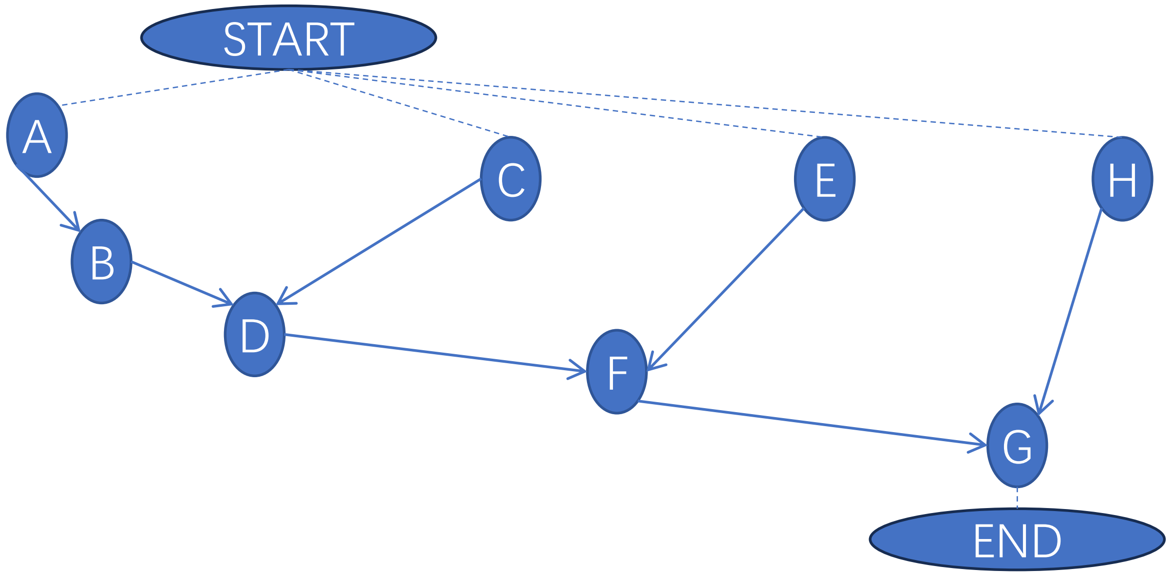
repeat {

Select a vertex v_i whose successors are all scheduled;

Schedule v_i by setting $t_i = \min \; t_j - d_i$;}

until (v_0 is scheduled);

return (t);}



END 6
G 5
F 4
D 3
B 2
A

ALAP ($G_s(V,E)$, x) {

Schedule v_n by setting $t_n = x + 1$;

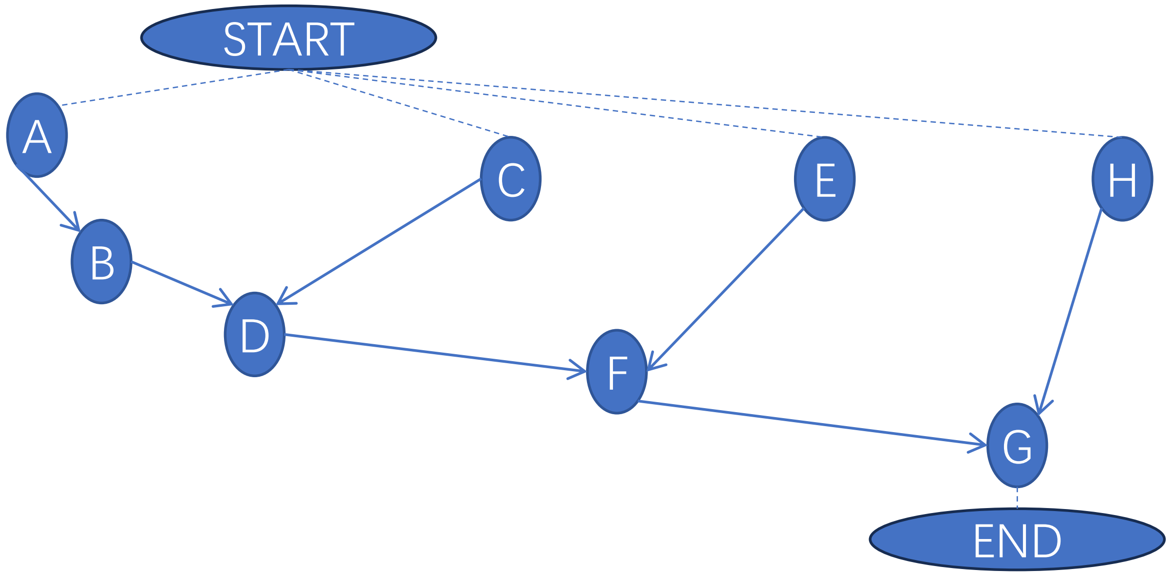
repeat {

Select a vertex v_i whose successors are all scheduled;

Schedule v_i by setting $t_i = \min \; t_j - d_i$;}

until (v_0 is scheduled);

return (t);}



END 6
G 5
F 4
D 3
B 2
A 1

ALAP ($G_s(V,E)$, x) {

Schedule v_n by setting $t_n = x + 1$;

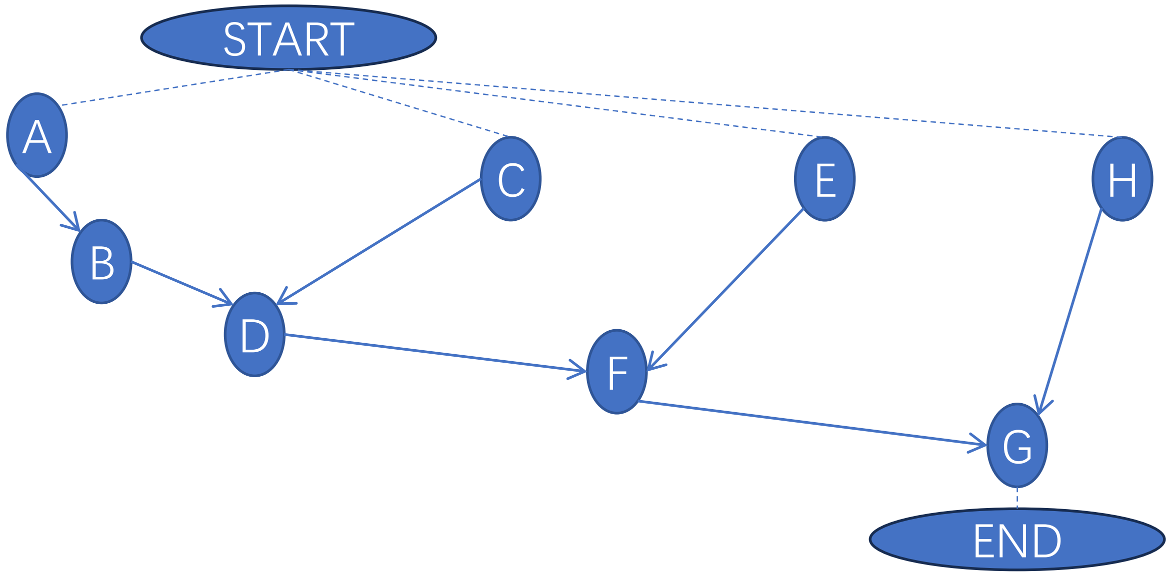
repeat {

Select a vertex v_i whose successors are all scheduled;

Schedule v_i by setting $t_i = \min \ t_j - d_i$;}

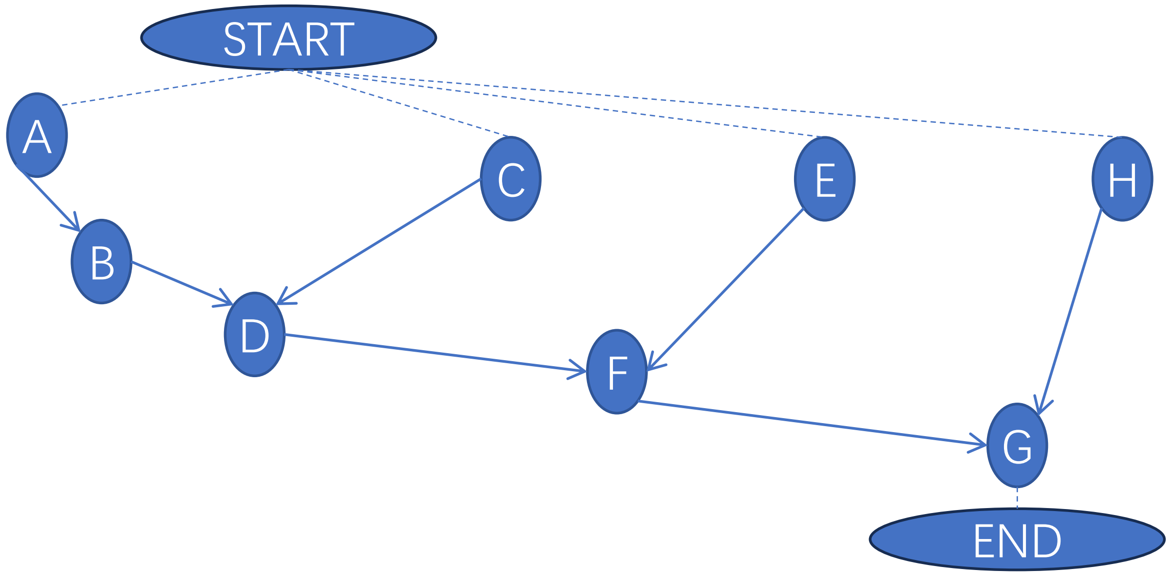
until (v_0 is scheduled);

return (t);}



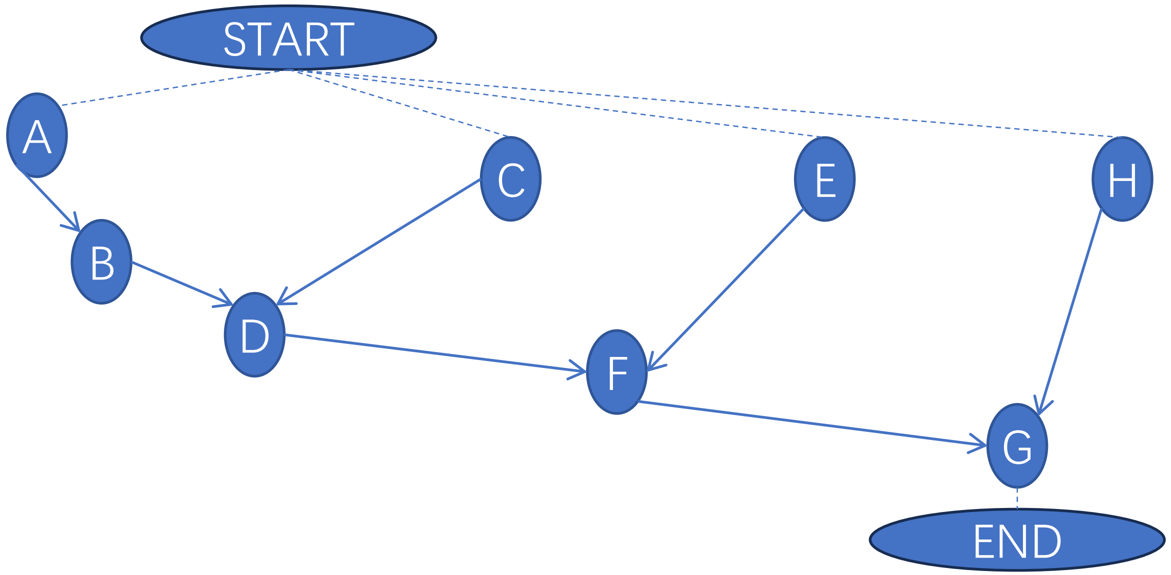
END 6
G 5
F 4
D 3
B 2
A 1
C

```
ALAP ( G_s(V,E), x ) {
        Schedule v_n by setting t_n = x + 1;
        repeat {
                Select a vertex v_i whose successors are all scheduled;
                Schedule v_i by setting  t_i =   min   t_j  - d_i;}
        until (v_0 is scheduled);
        return (t);}
```



END 6
G 5
F 4
D 3
B 2
A 1
C 2

```
ALAP ( G_s(V,E), x ) {
    Schedule v_n by setting t_n = x + 1;
    repeat {
        Select a vertex v_i whose successors are all scheduled;
        Schedule v_i by setting  t_i =   min   t_j  - d_i;}
    until (v_0 is scheduled);
    return (t);}
```



END 6
G 5
F 4
D 3
B 2
A 1
C 2
E

ALAP ($G_s(V,E)$, x) {

Schedule v_n by setting $t_n = x + 1$;

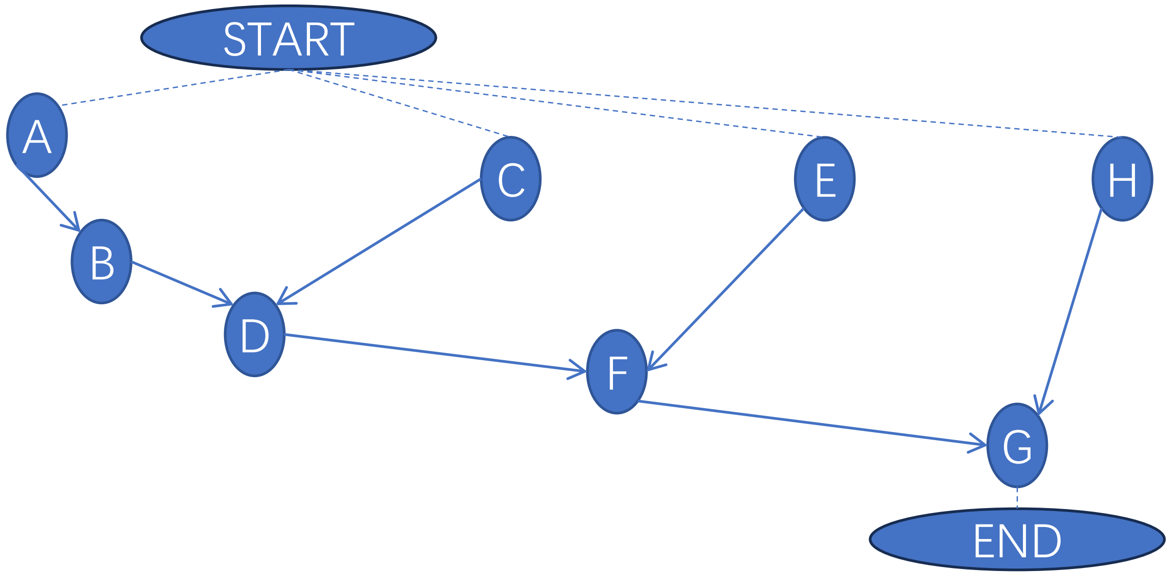
repeat {

Select a vertex v_i whose successors are all scheduled;

Schedule v_i by setting $t_i = \min \; t_j - d_i$;}

until (v_0 is scheduled);

return (t);}



END 6
G 5
F 4
D 3
B 2
A 1
C 2
E 3

ALAP ($G_s(V,E)$, x) {

Schedule v_n by setting $t_n = x + 1$;

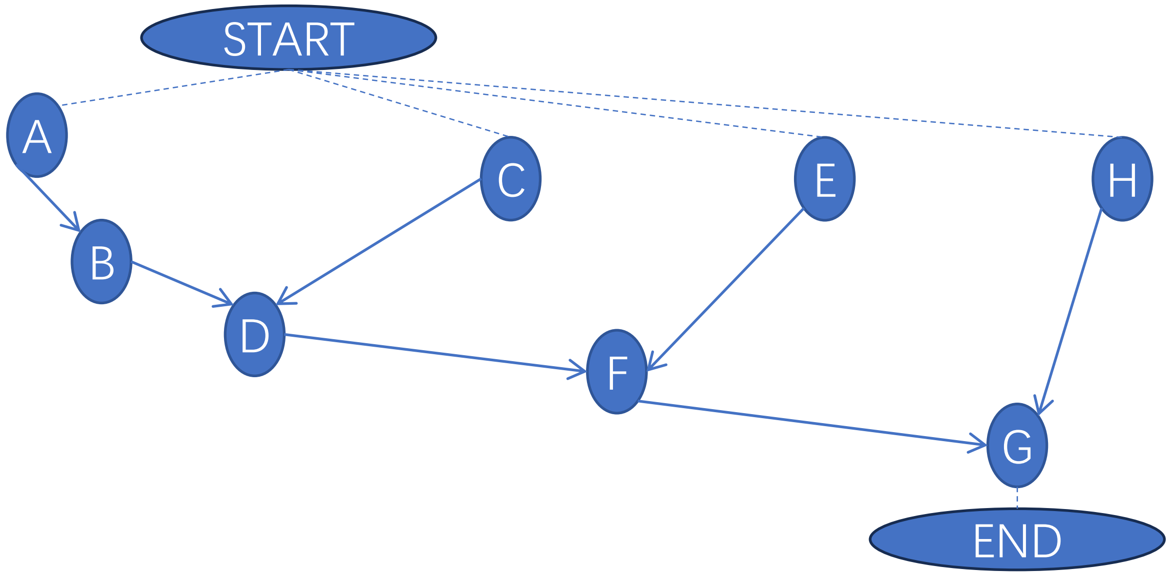
repeat {

Select a vertex v_i whose successors are all scheduled;

Schedule v_i by setting $t_i = \min \ t_j - d_i$;}

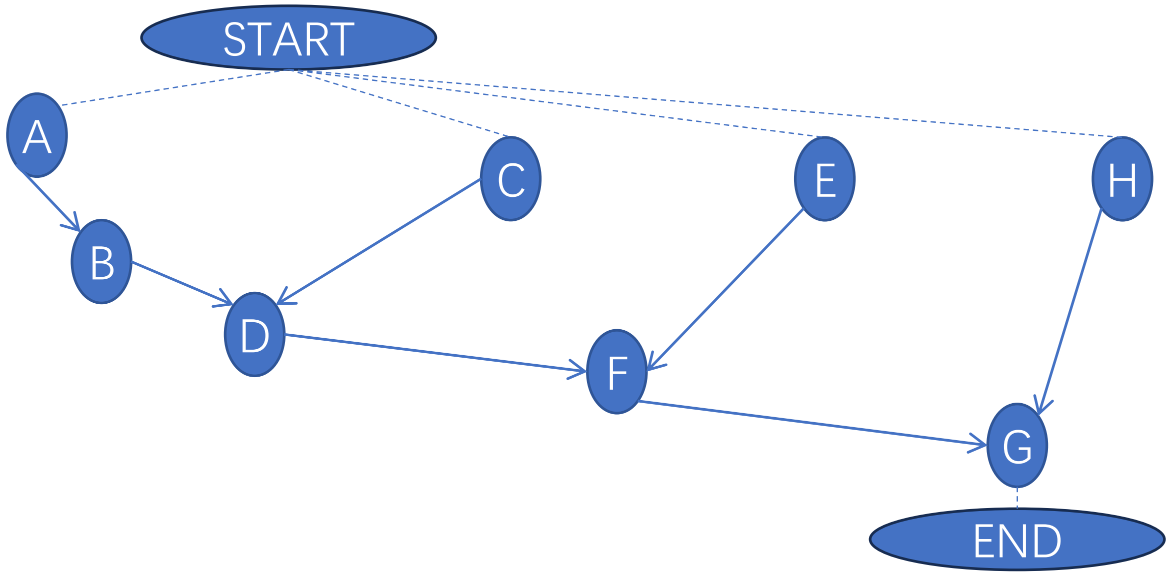
until (v_0 is scheduled);

return (t);}

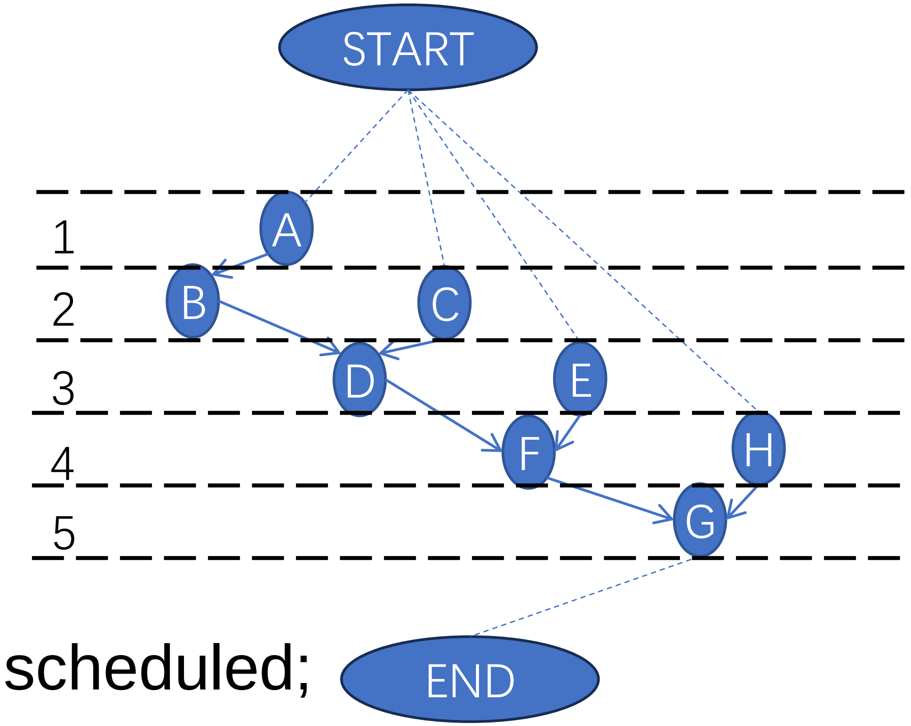


END 6
G 5
F 4
D 3
B 2
A 1
C 2
E 3
H

```
ALAP ( G_s(V,E), x ) {
    Schedule v_n by setting t_n = x + 1;
    repeat {
        Select a vertex v_i whose successors are all scheduled;
        Schedule v_i by setting  t_i =  min  t_j - d_i;}
    until (v_0 is scheduled);
    return (t);}
```



END 6
G 5
F 4
D 3
B 2
A 1
C 2
E 3
H 4




```
ALAP ( G_s(V,E), x ) {
        Schedule v_n by setting t_n = x + 1;
        repeat {
                Select a vertex v_i whose successors are all scheduled;
                Schedule v_i by setting  t_i =   min   t_j  - d_i;}
        until (v_0 is scheduled);
        return (t);}
```

休息一下

随堂作业

in-class assignment

已知输入文件的格式如下：

第1行：一个数字n，代表顶点个数

第2到2+n行：每一行由一个字符串和一个数字组成，以空格隔开，分别代表顶点的名字和所需的延迟时间

首尾两个顶点是虚拟顶点，代表起始点和结束点

第2+n+1行到结束：每一行由两个顶点名组成，以空格隔开，代表第一个顶点需要在第二个顶点之前完成

请分别写出以数组方式依次存储和遍历顶点时，ASAP和ALAP调度下的程序**为顶点决定开始周期的顺序**和**输出文件的内容**，输出文件格式如下：

每一行由一个字符串和一个数字组成，以空格隔开，分别代表顶点名和开始的调度周期

输入文件内容：

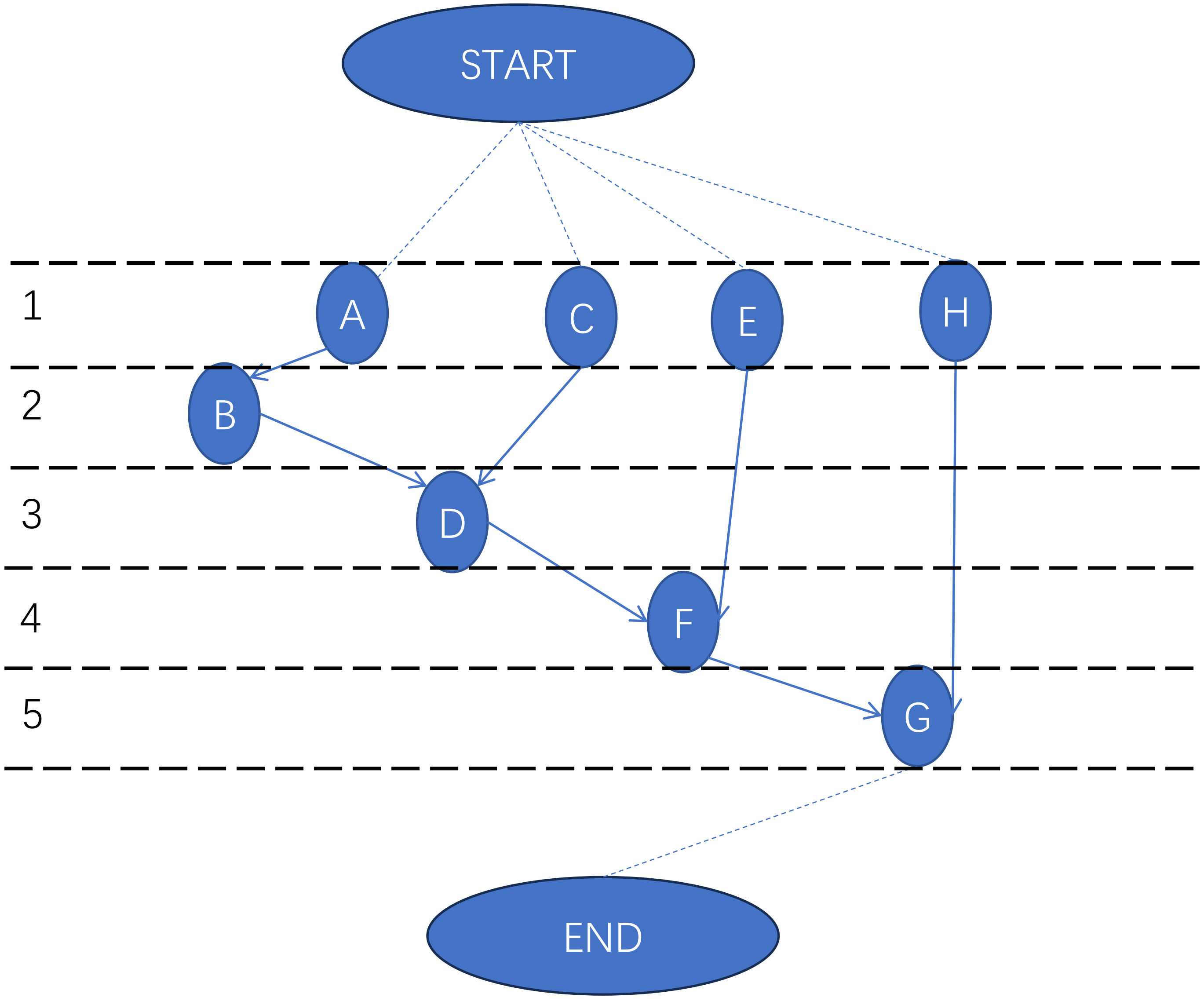
```
7
START 1
A 2
B 1
C 3
D 2
E 1
END 1
START A
START C
A B
C B
A E
B D
E END
D END
```

你觉得ASAP和ALAP算法？

A. 太难
B. 适中
C. 太简单

输入文件内容:
10
START 1
A 1
B 1
C 1
D 1
E 1
F 1
G 1
H 1
END 0
START B
START A
START C
START E
START G
A B
C D
B D
E F
D F
H G
F G
B END
D END
F END
G END

输入输出的例子



为顶点决定开始周期的顺序:

START->

A->B->C->D->E->F->H->G

->END

输出文件内容:
START 0
A 1
B 2
C 1
D 3
E 1
F 4
H 1
G 5
END 6

输入文件内容:

7

START 1

A 2

B 1

C 3

D 2

E 1

END 0

START A

START C

A B

C B

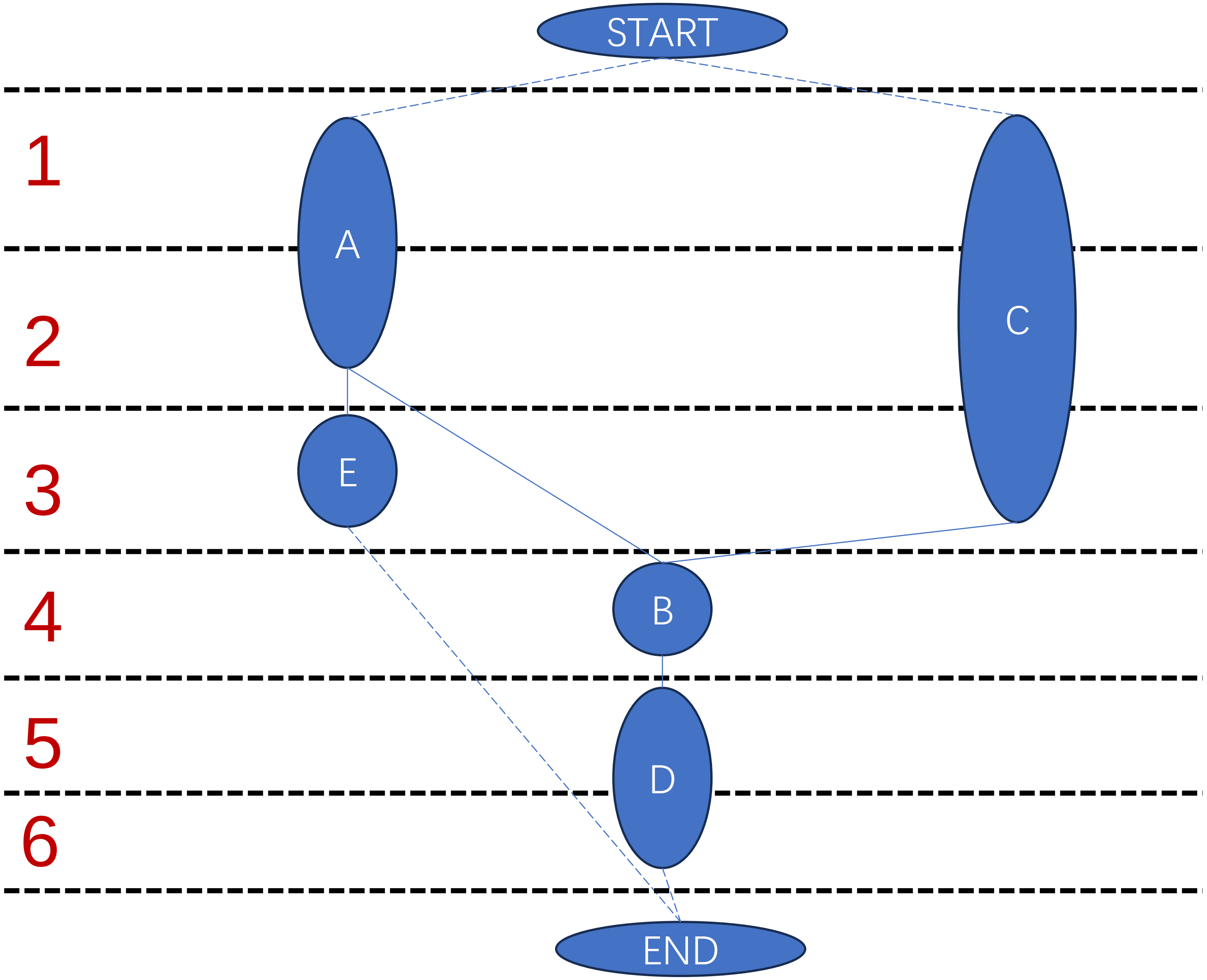
A E

B D

E END

D END

ASAP的结果



为顶点决定开始周期的顺序:

START->

A->C->B->D->E

->END

输出文件内容:

START 0

A 1

C 1

B 4

D 5

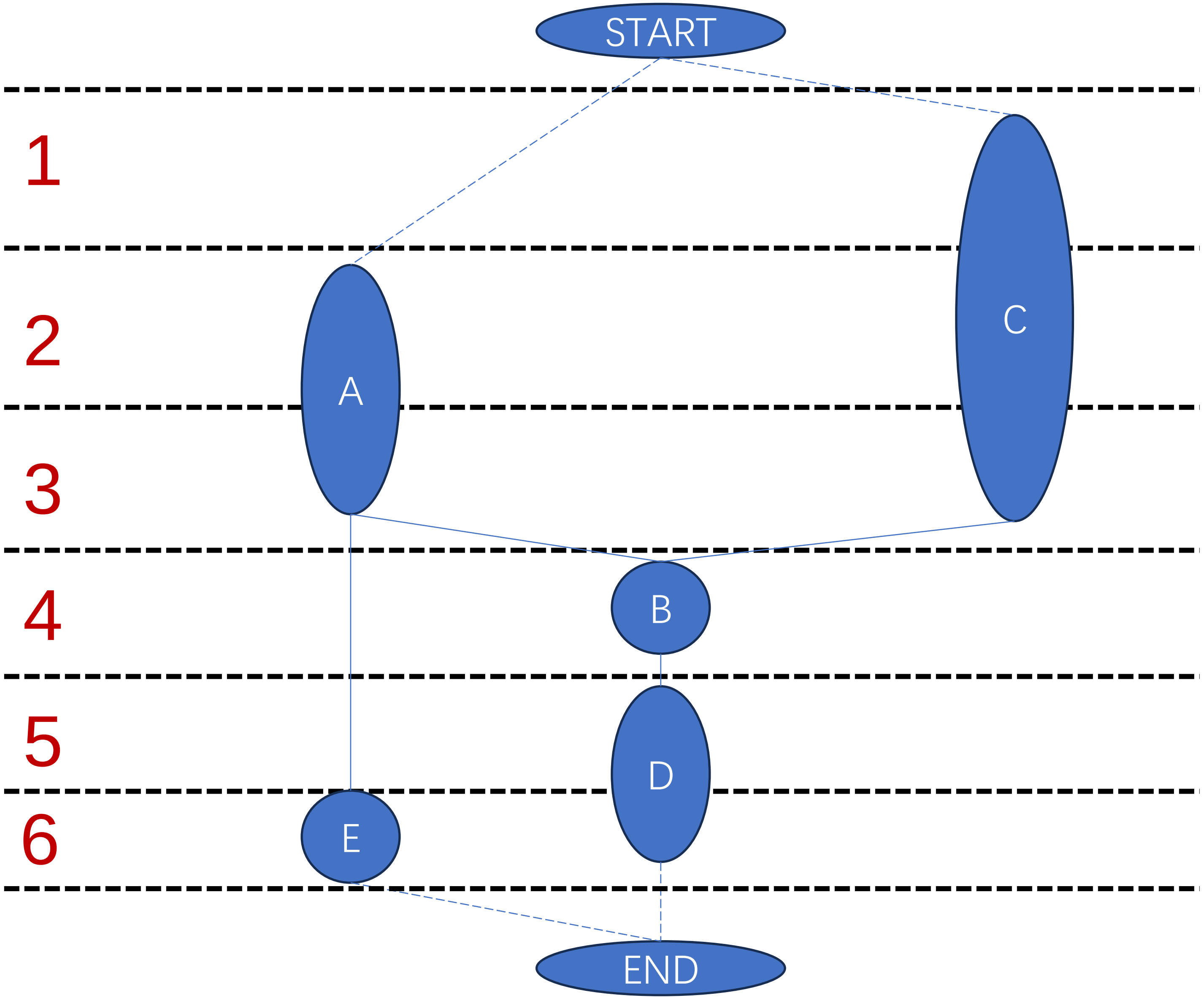
E 3

END 7

ALAP的结果

输入文件内容:

7
START 1
A 2
B 1
C 3
D 2
E 1
END 0
START A
START C
A B
C B
A E
B D
E END
D END



为顶点决定开始周期的顺序:

END->
D -> B -> C -> E -> A
->START

输出文件内容:

START 0
D 5
B 4
C 1
E 6
A 2
END 7

有约束的调度问题

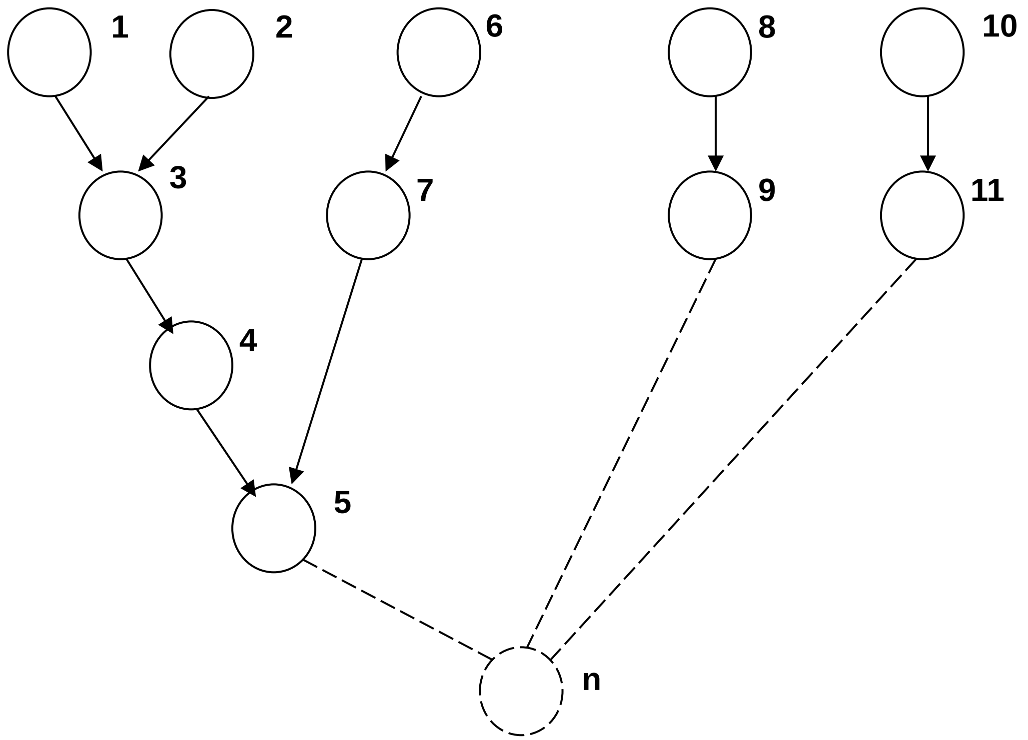
有约束的调度

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）

优先级约束的多处理器调度

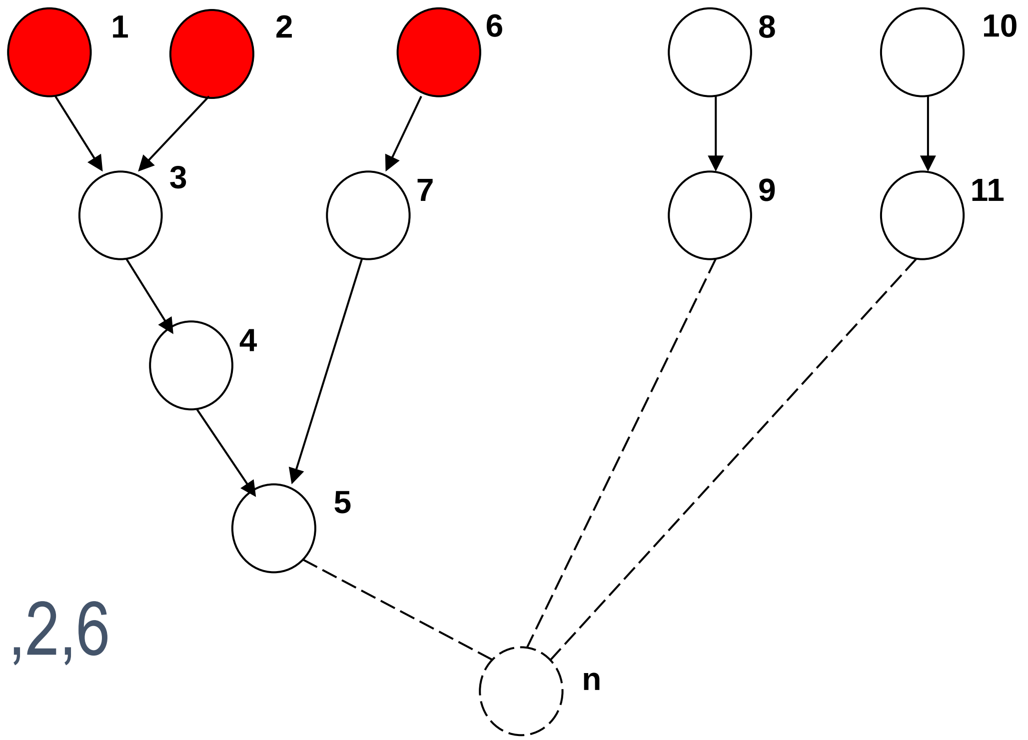
Precedence-constrained Multiprocessor Scheduling



优先级约束的多处理器调度

Precedence-constrained Multiprocessor Scheduling

$\bar{a} = 3$

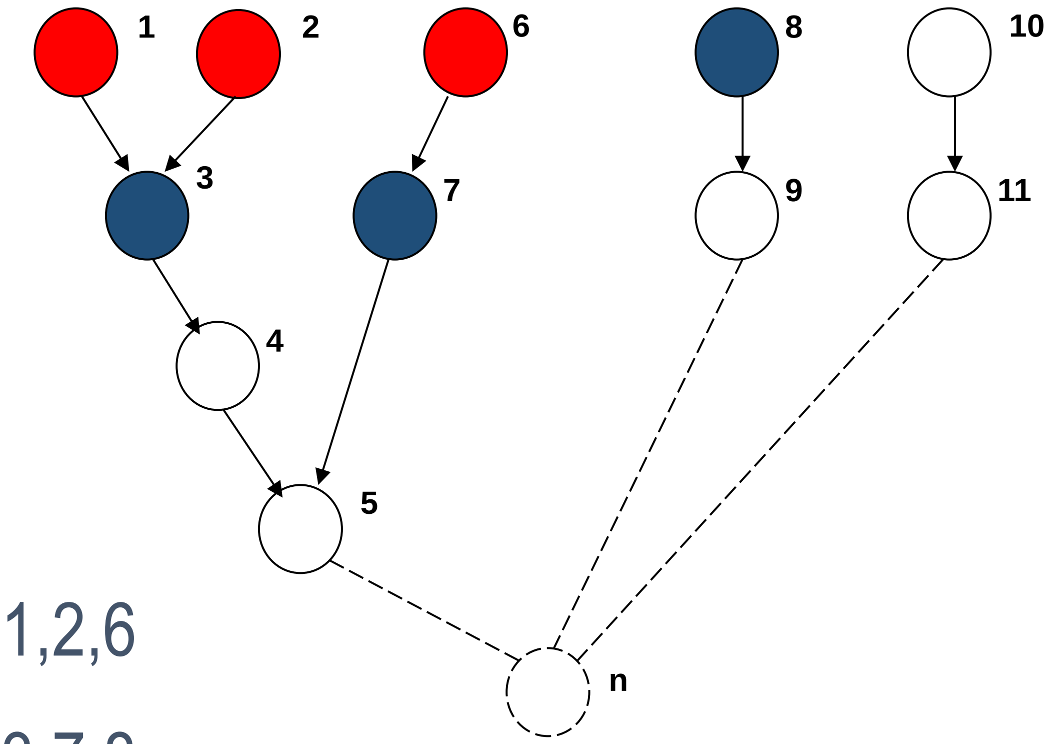


Step 1: Op 1,2,6

优先级约束的多处理器调度

Precedence-constrained Multiprocessor Scheduling

$\bar{a} = 3$



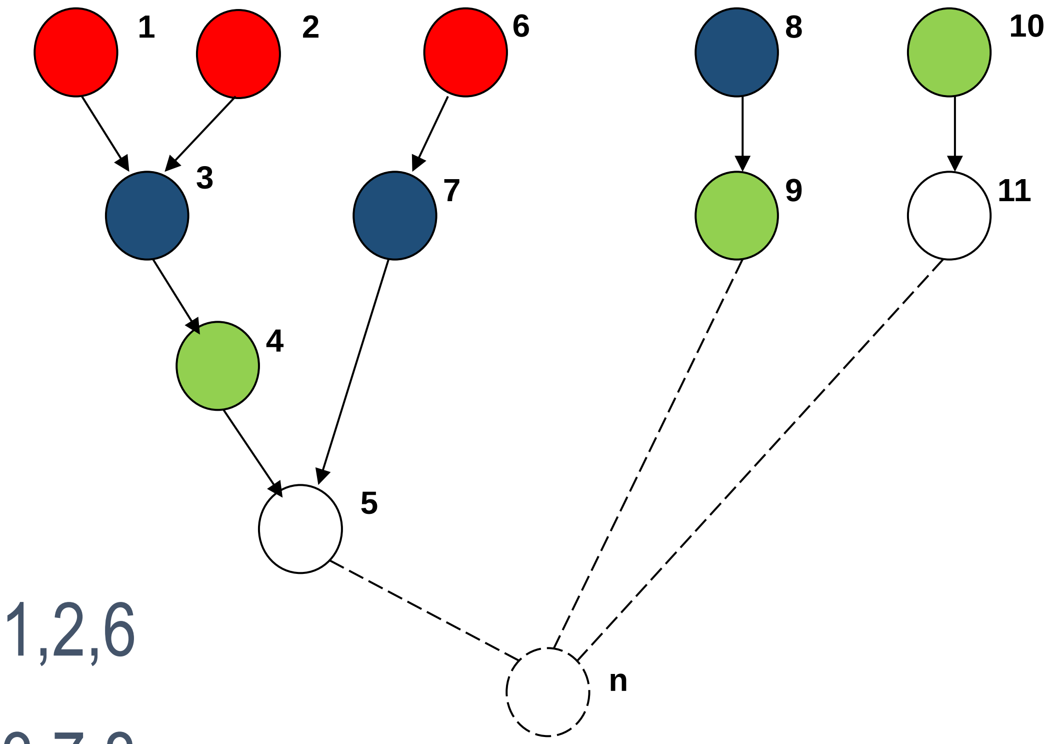
Step 1: Op 1,2,6

Step 2: Op 3,7,8

优先级约束的多处理器调度

Precedence-constrained Multiprocessor Scheduling

$\bar{a} = 3$



Step 1: Op 1,2,6

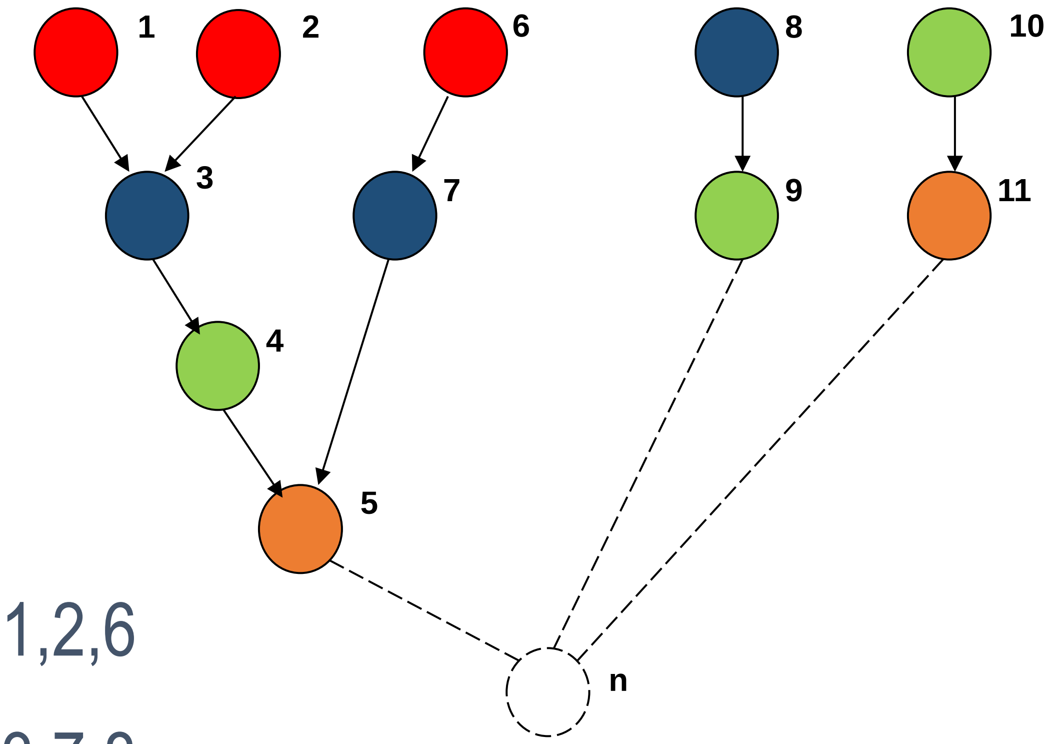
Step 2: Op 3,7,8

Step 3: Op 4,9,10

优先级约束的多处理器调度

Precedence-constrained Multiprocessor Scheduling

$\bar{a} = 3$



Step 1: Op 1,2,6

Step 2: Op 3,7,8

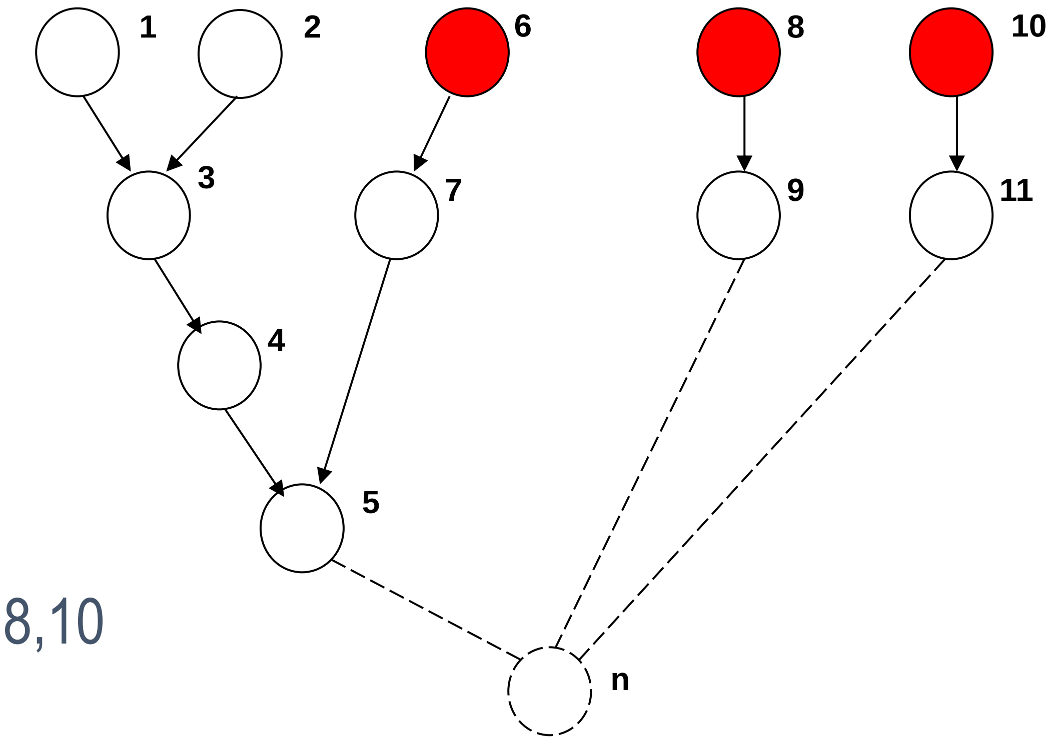
Step 3: Op 4,9,10

Step 4: Op 5,11

优先级约束的多处理器调度

Precedence-constrained Multiprocessor Scheduling

$\bar{a} = 3$

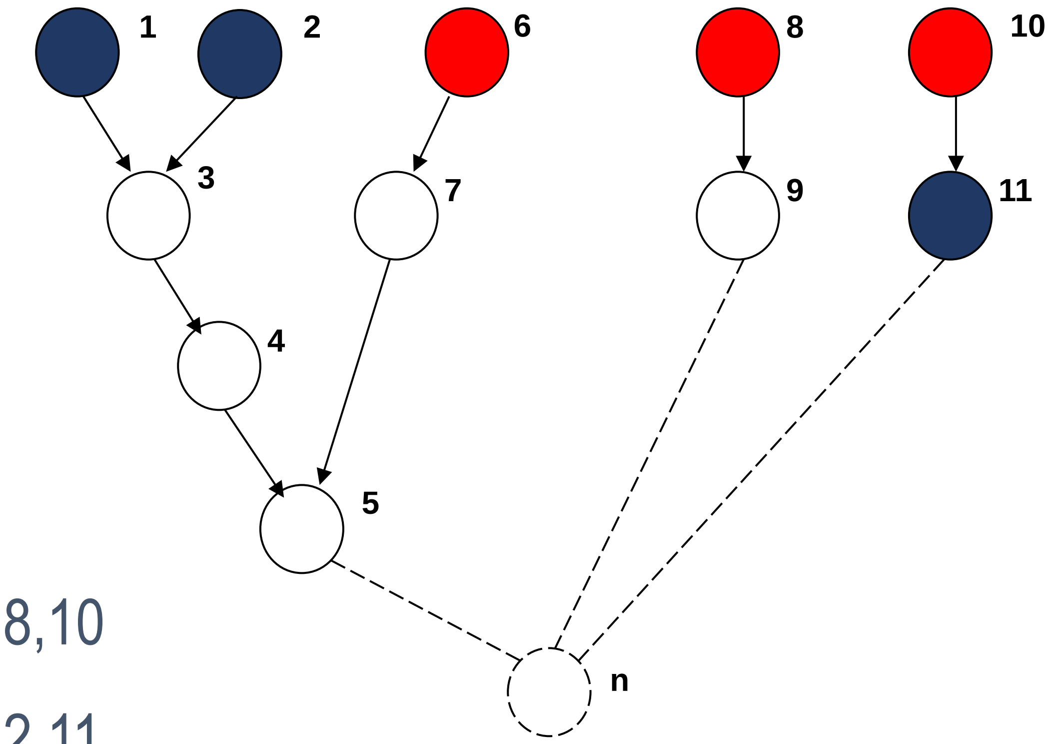


Step 1: Op 6,8,10

优先级约束的多处理器调度

Precedence-constrained Multiprocessor Scheduling

$\bar{a} = 3$



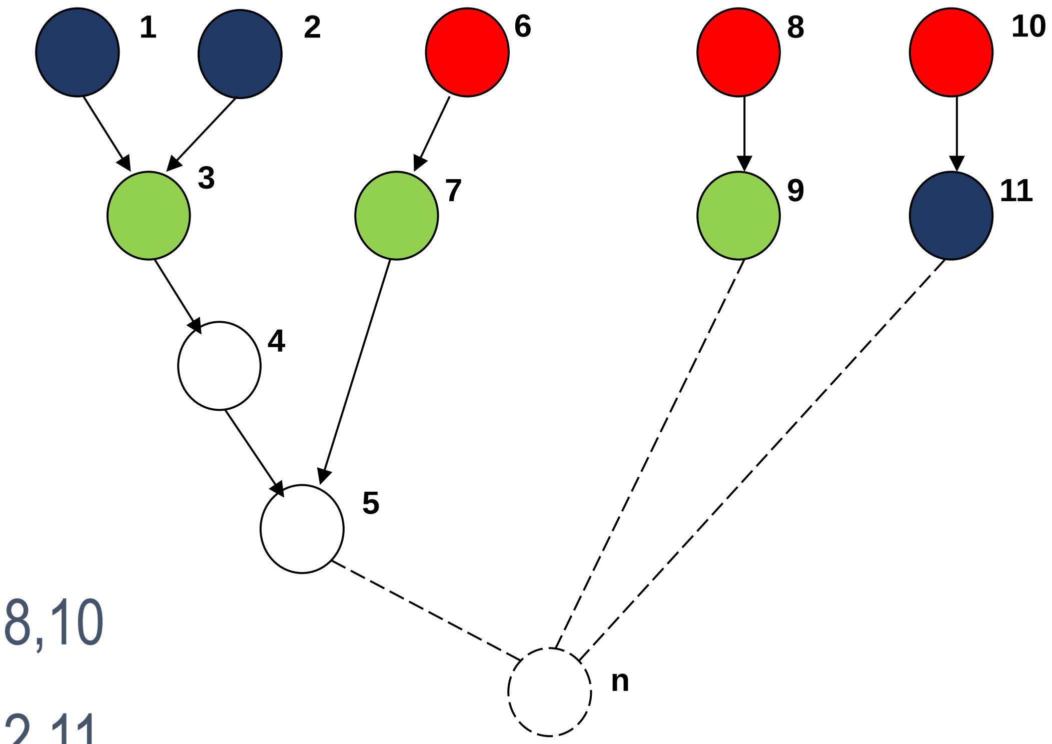
Step 1: Op 6,8,10

Step 2: Op 1,2,11

优先级约束的多处理器调度

Precedence-constrained Multiprocessor Scheduling

$\bar{a} = 3$



Step 1: Op 6,8,10

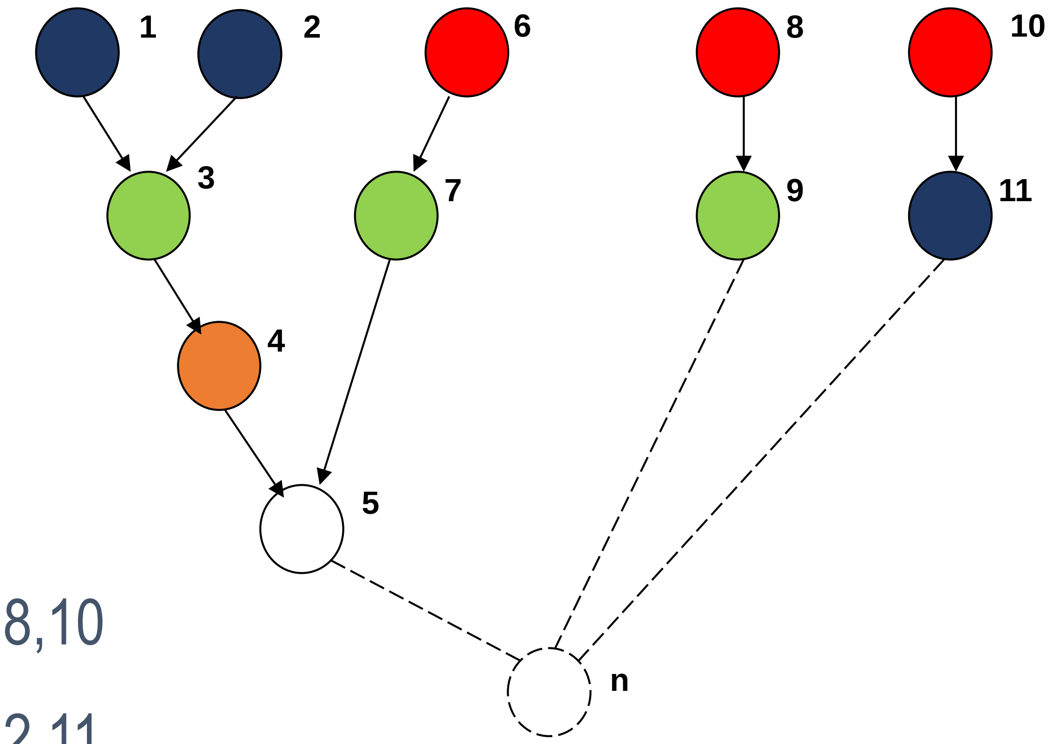
Step 2: Op 1,2,11

Step 3: Op 3,7,9

优先级约束的多处理器调度

Precedence-constrained Multiprocessor Scheduling

$\bar{a} = 3$



Step 1: Op 6,8,10

Step 2: Op 1,2,11

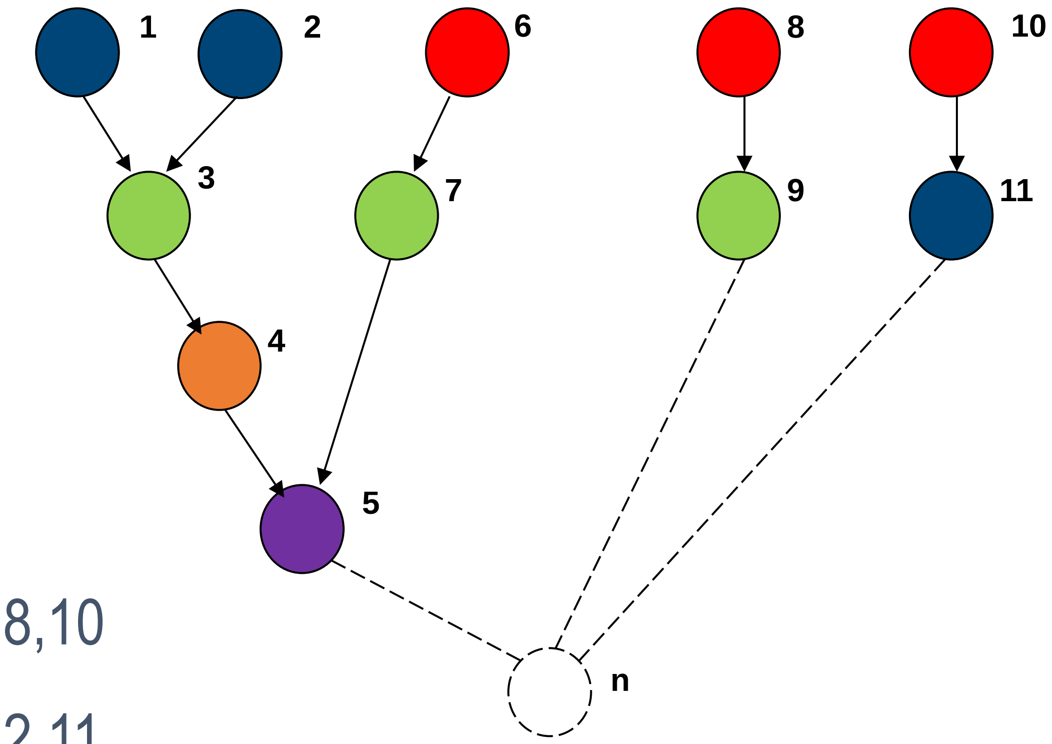
Step 3: Op 3,7,9

Step 4: Op 4

优先级约束的多处理器调度

Precedence-constrained Multiprocessor Scheduling

$\bar{a} = 3$



Step 1: Op 6,8,10

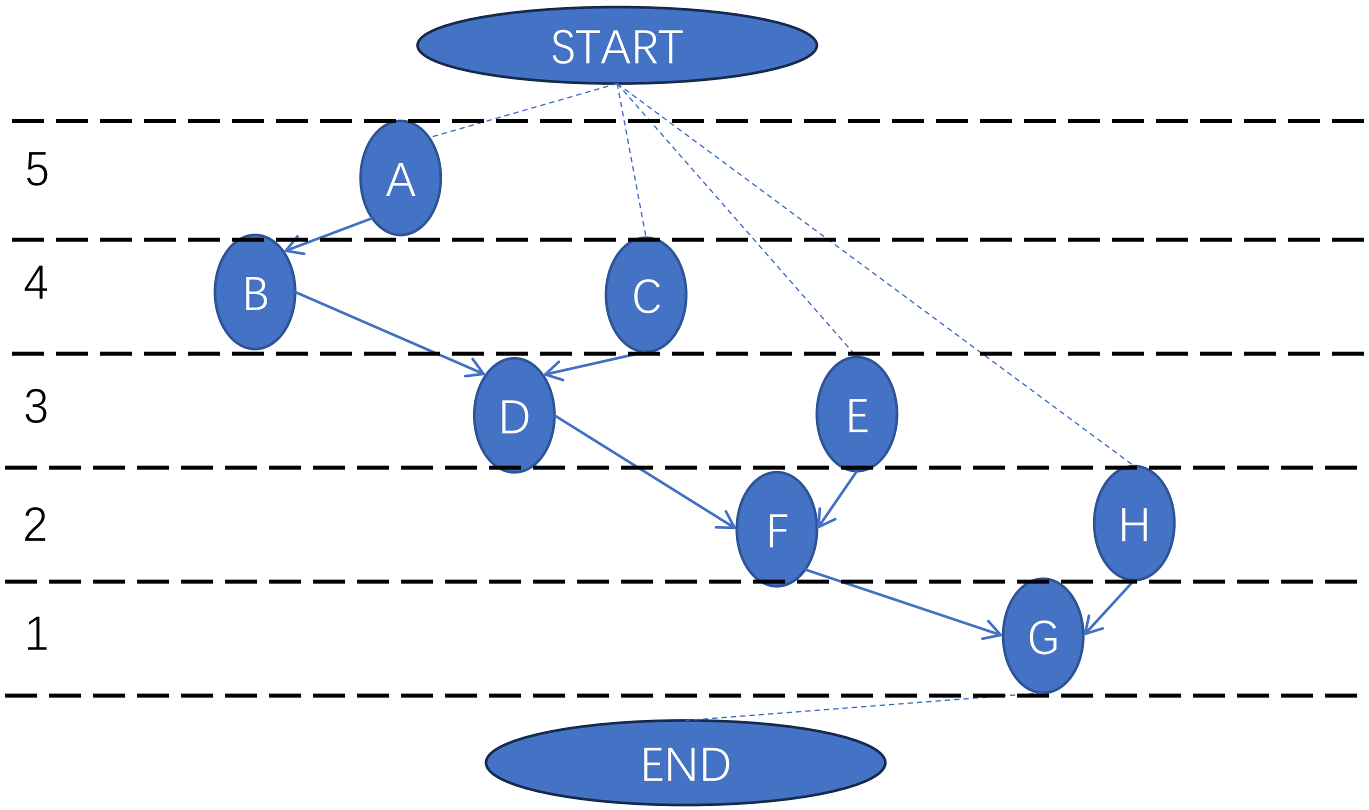
Step 2: Op 1,2,11

Step 3: Op 3,7,9

Step 4: Op 4

Step 5: Op 5

距离终点的距离



END 6
G 5
F 4
D 3
B 2
A 1
C 2
E 3
H 4

HU调度算法

HU scheduling algorithm

HU (G(V, E), a) {

  Label the vertices;

  t0 = 0; l = 1;

  repeat {

    U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled;

    Select $S \subseteq U$ vertices, such that $|S| \leq a$ and labels in S are maximal;

    Schedule the S operations at step t by setting $t_i = l \; \forall v_i \in S$;

    l = l + 1;

  }

  until (v_n is scheduled);

}

HU调度算法

HU scheduling algorithm

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

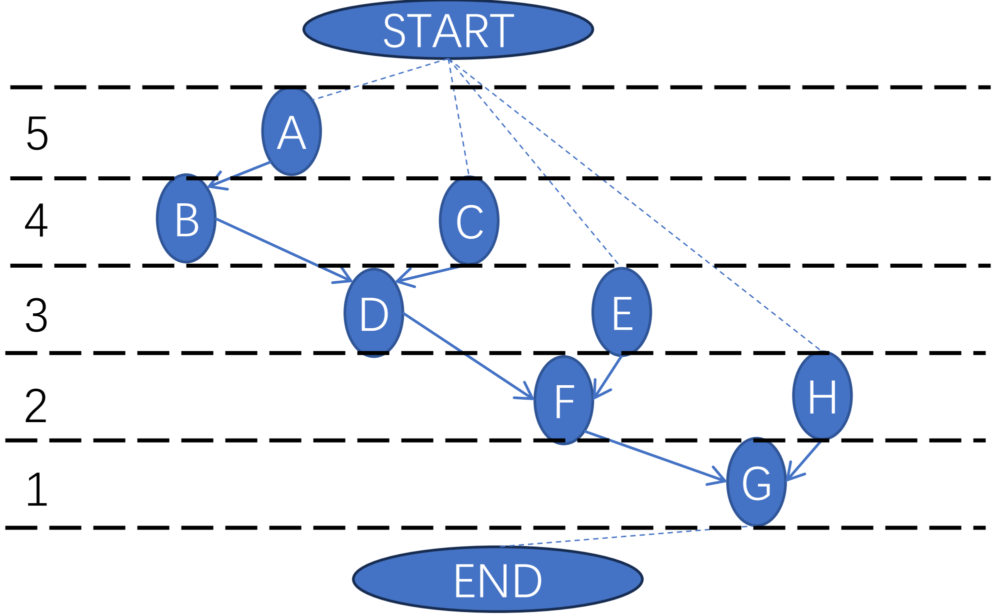
选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

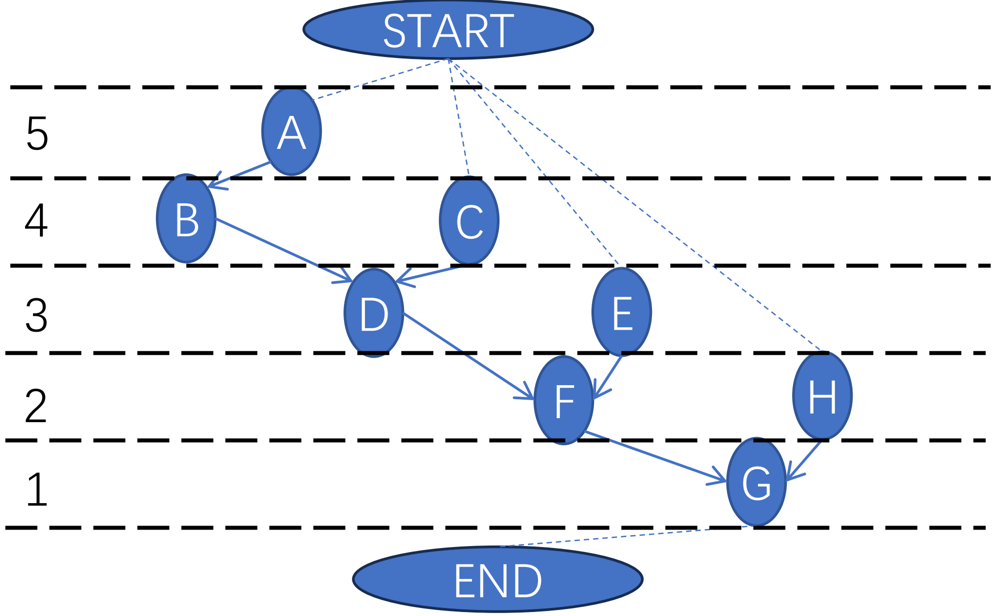
选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

在步骤 t 处调度 S 中的所有操作，即对所有 vi ∊ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

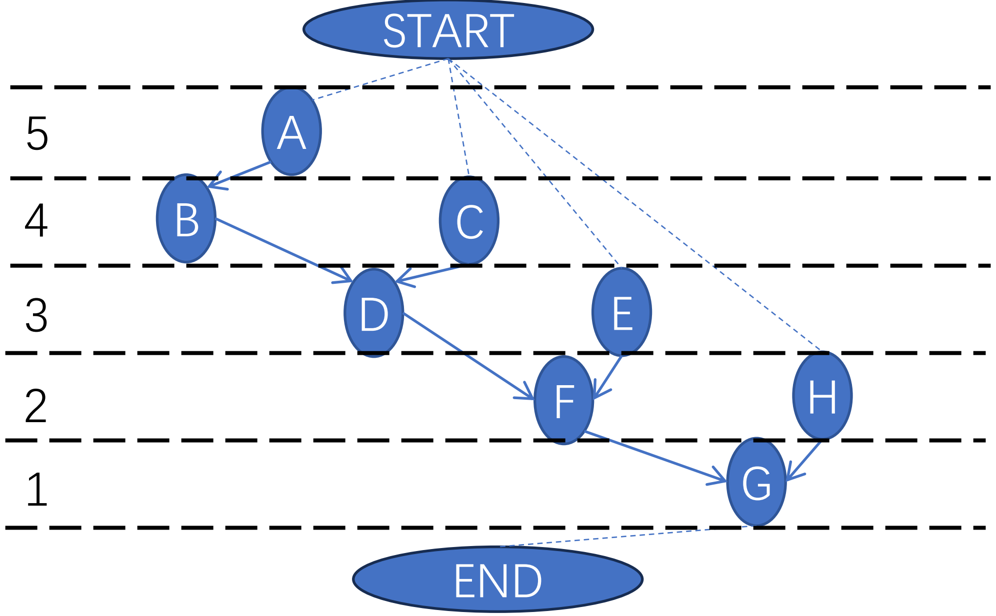
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	1
U	
S	



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

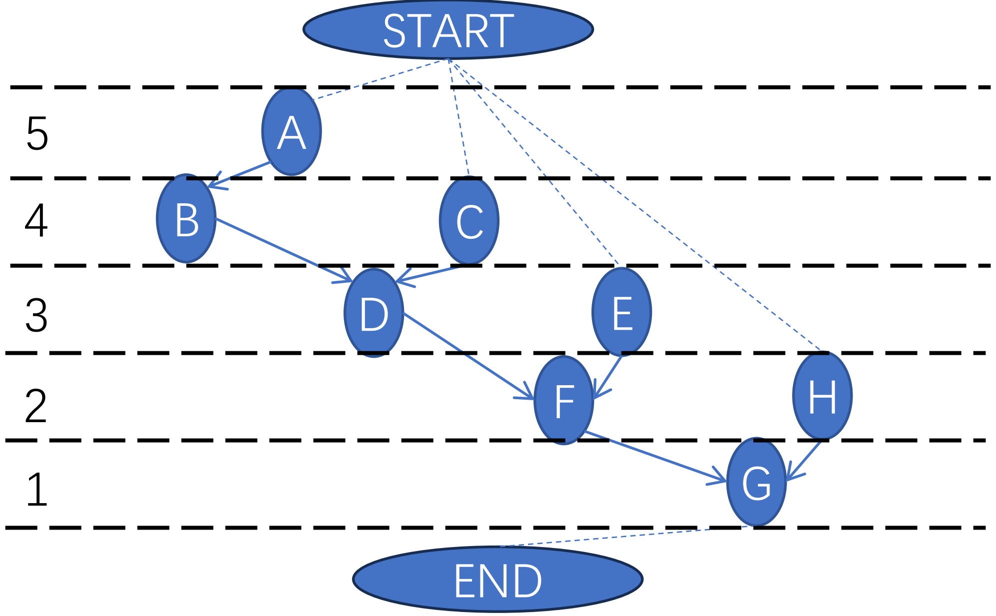
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	1
U	
S	



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

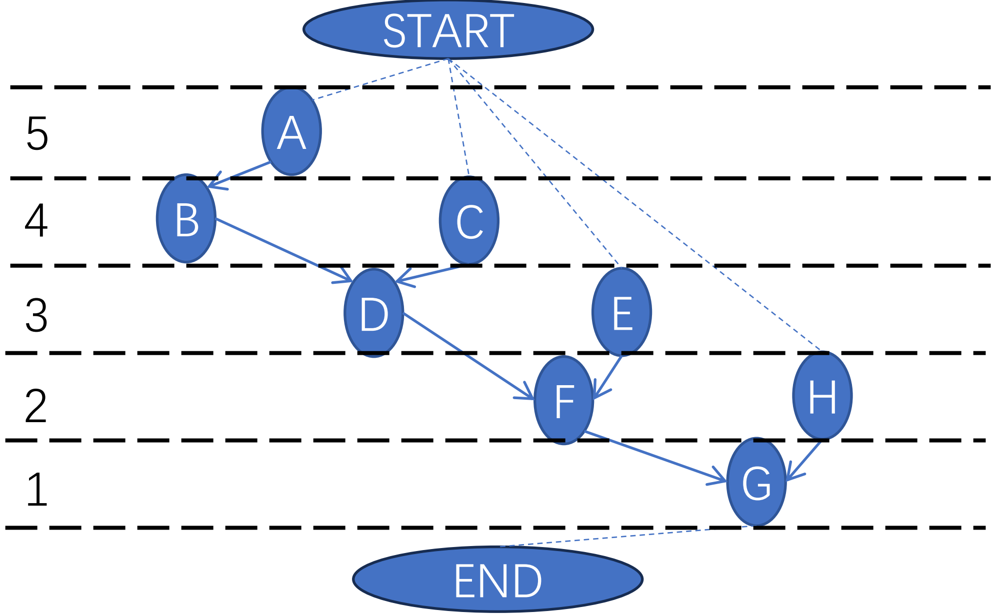
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	1
U	{A, C, E, H}
S	



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

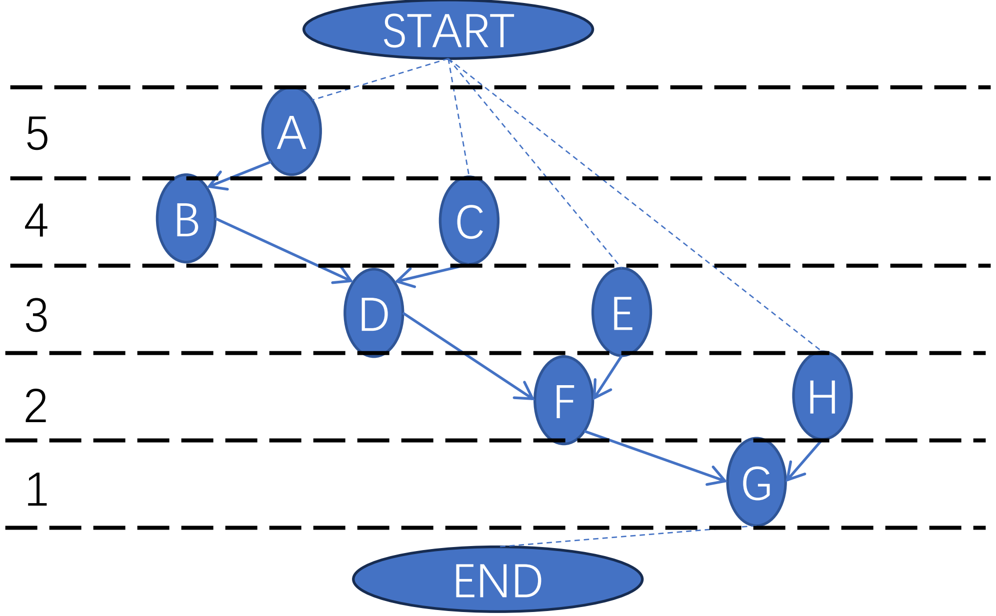
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	1
U	{A, C, E, H}
S	{A,C,E}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1

HU (G(V, E), a) {

  对顶点进行标号;

  t0 = 0; l = 1;

  重复执行以下步骤:

    U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

    选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

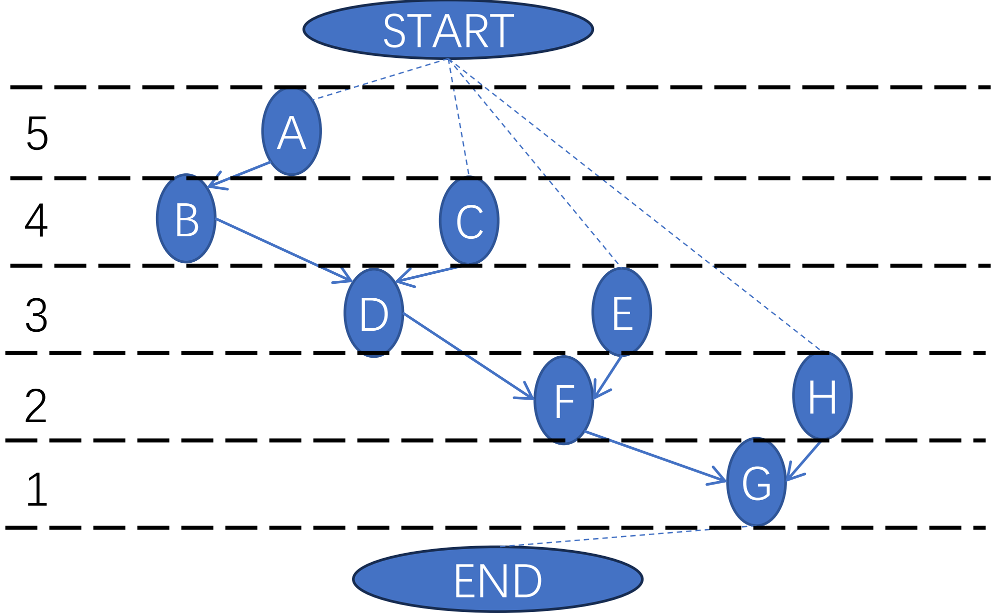
    在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

    l = l + 1;

  直到 vn 被调度;

}

变量	当前值
l	1
U	{A, C, E, H}
S	{A,C,E}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1

HU (G(V, E), a) {

  对顶点进行标号;

  t0 = 0; l = 1;

  重复执行以下步骤:

    U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

    选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

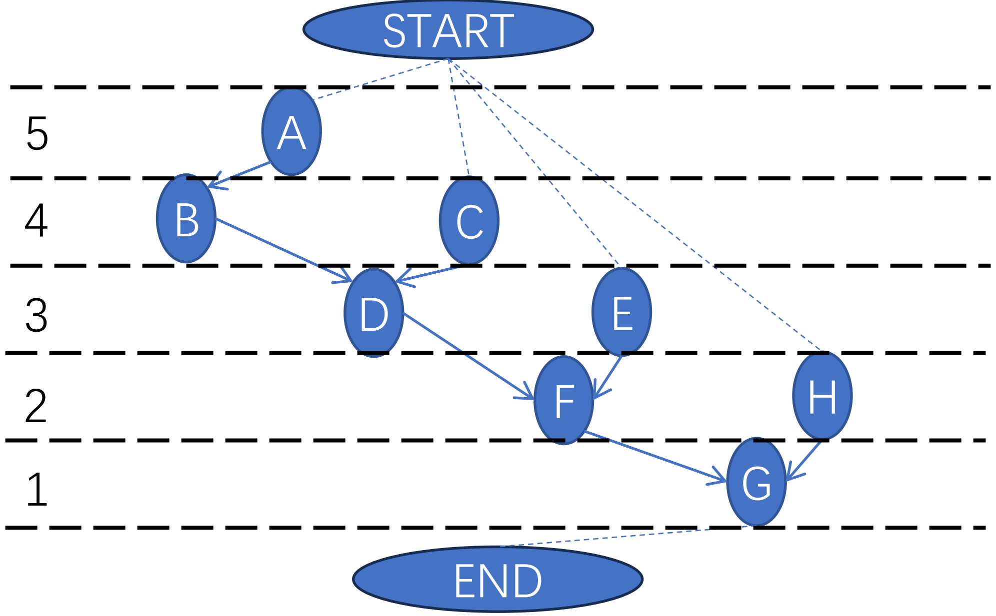
    在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

    l = l + 1;

  直到 vn 被调度;

}

变量	当前值
l	2
U	{A, C, E, H}
S	{A,C,E}

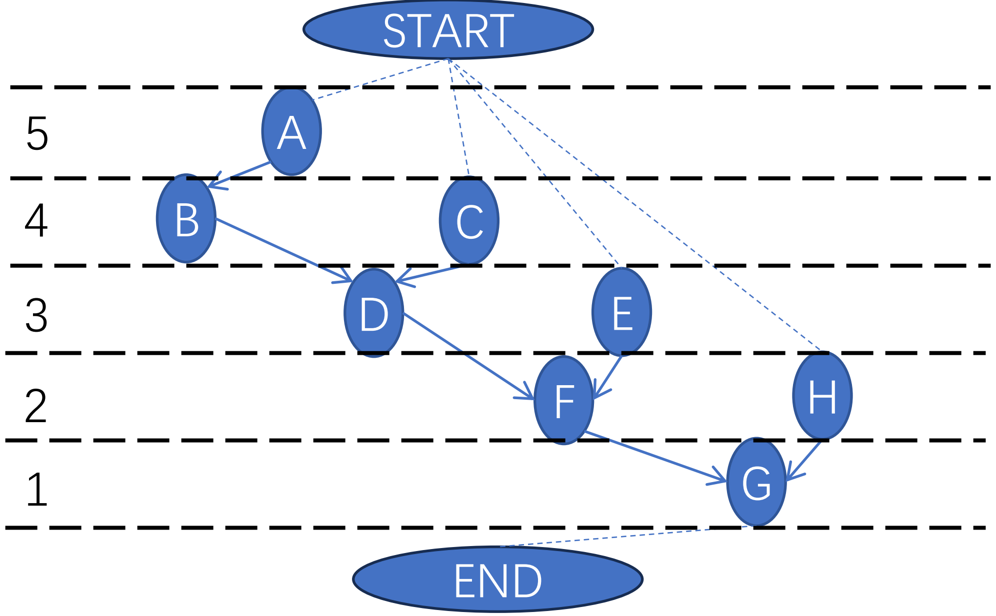


A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1

HU (G(V, E), a) {
    对顶点进行标号;
    t0 = 0; l = 1;
    重复执行以下步骤:
        U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;
        选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;
        在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;
        l = l + 1;
    直到 vn 被调度;
}

变量	当前值
l	2
U	{A, C, E, H}
S	{A,C,E}

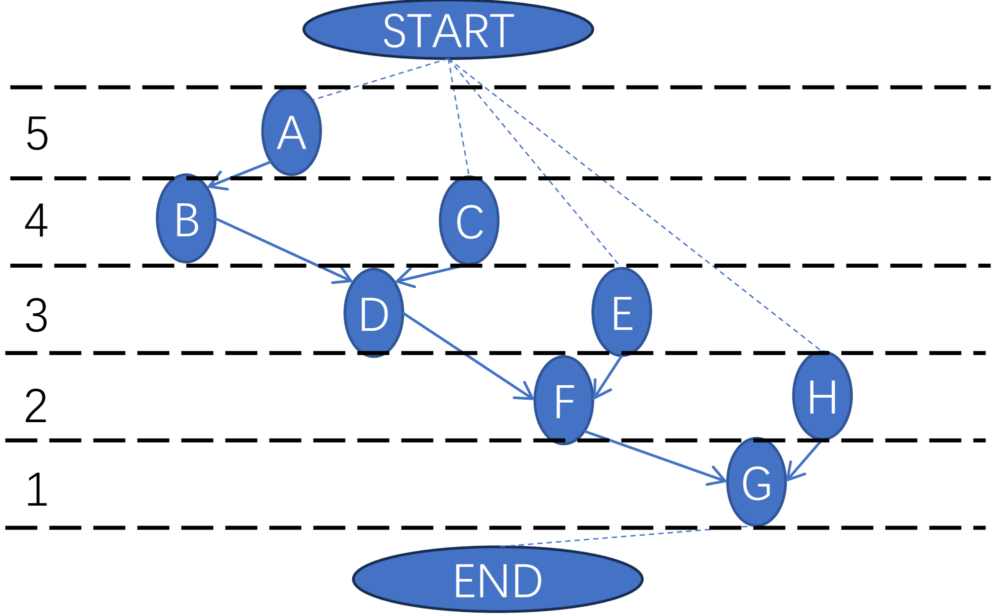


A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1

HU (G(V, E), a) {
　对顶点进行标号；
　t0 = 0； l = 1;
　重复执行以下步骤：
　　U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度；
　　选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大；
　　在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;
　　l = l + 1;
　直到 vn 被调度；
}

变量	当前值
l	2
U	{H, B}
S	{A,C,E}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1

HU (G(V, E), a) {

　对顶点进行标号;

　t0 = 0; l = 1;

　重复执行以下步骤:

　　U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

　　选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

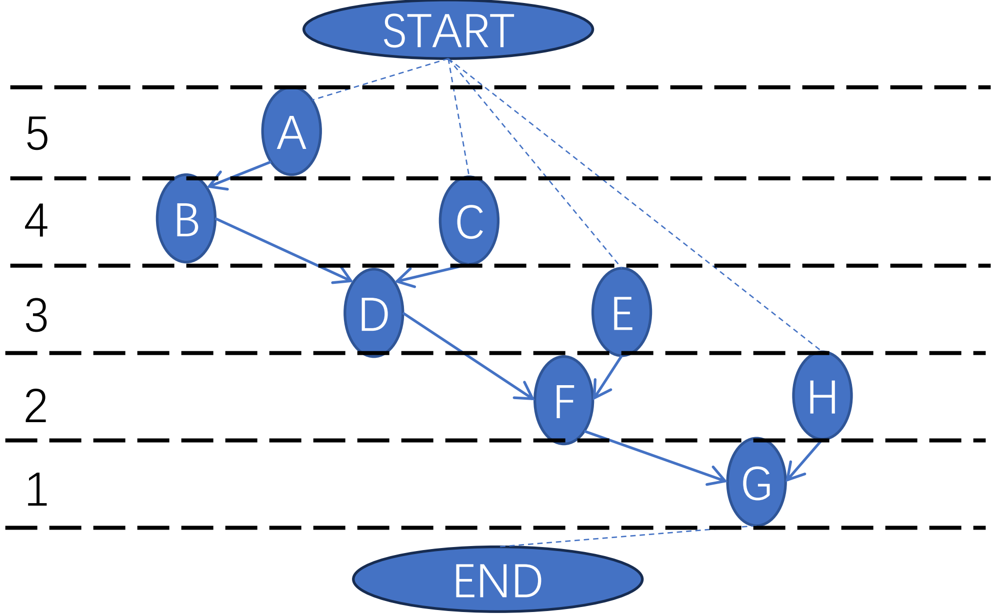
　　在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

　　l = l + 1;

　直到 vn 被调度;

}

变量	当前值
l	2
U	{H, B}
S	{H, B}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

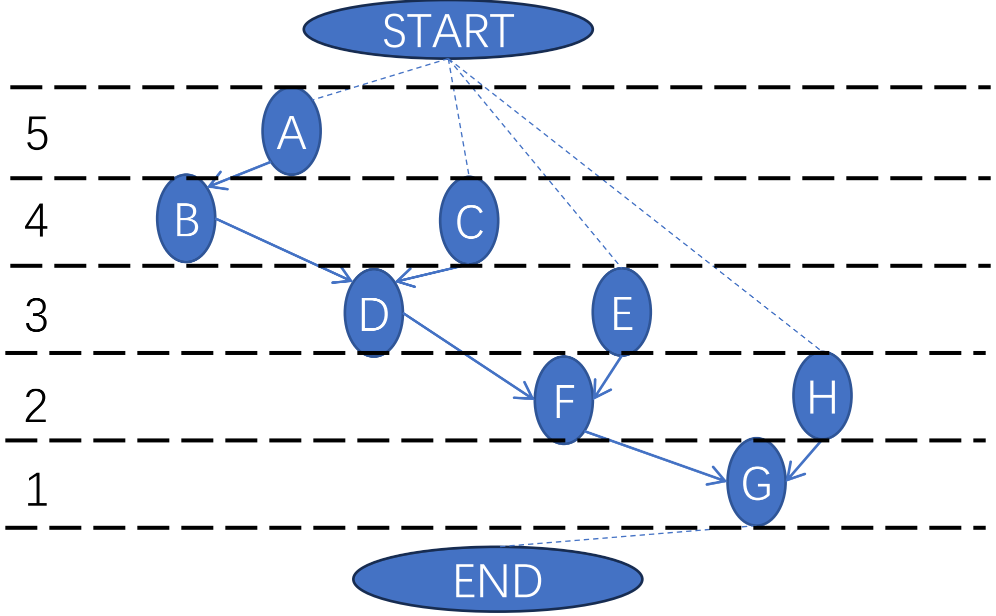
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	2
U	{H, B}
S	{H, B}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

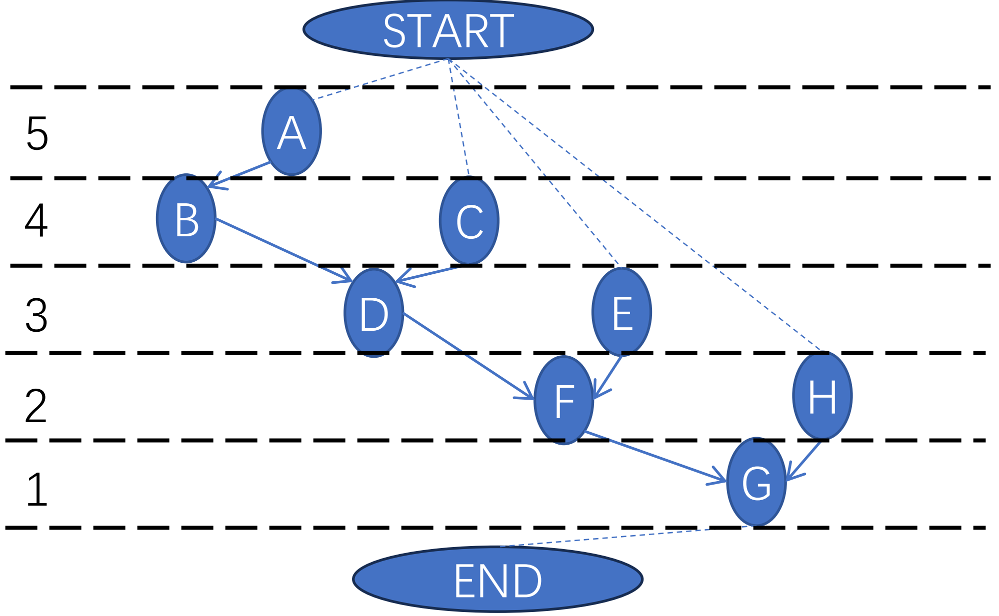
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	3
U	{H, B}
S	{H, B}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

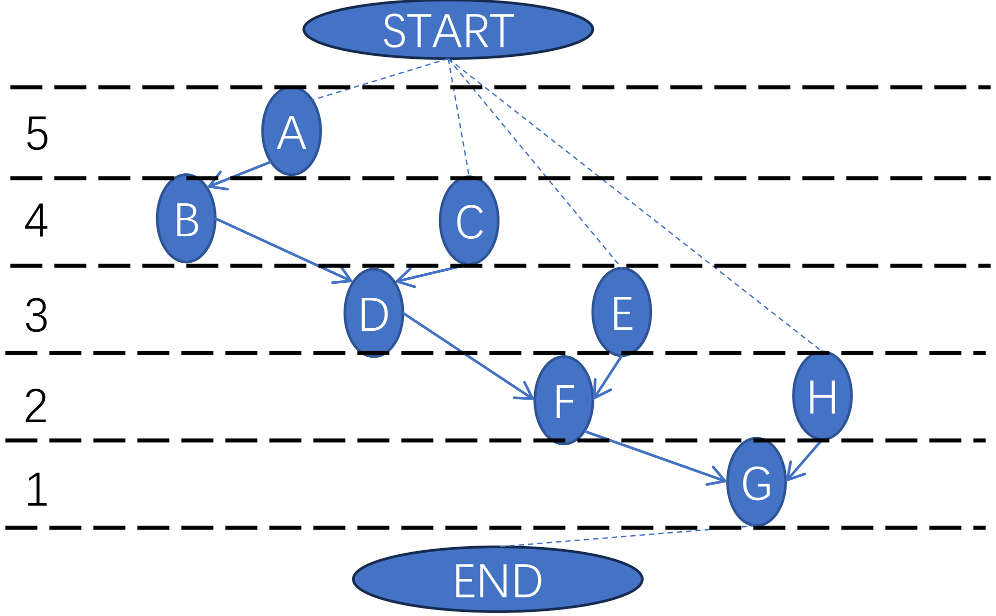
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	3
U	{D}
S	{H, B}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

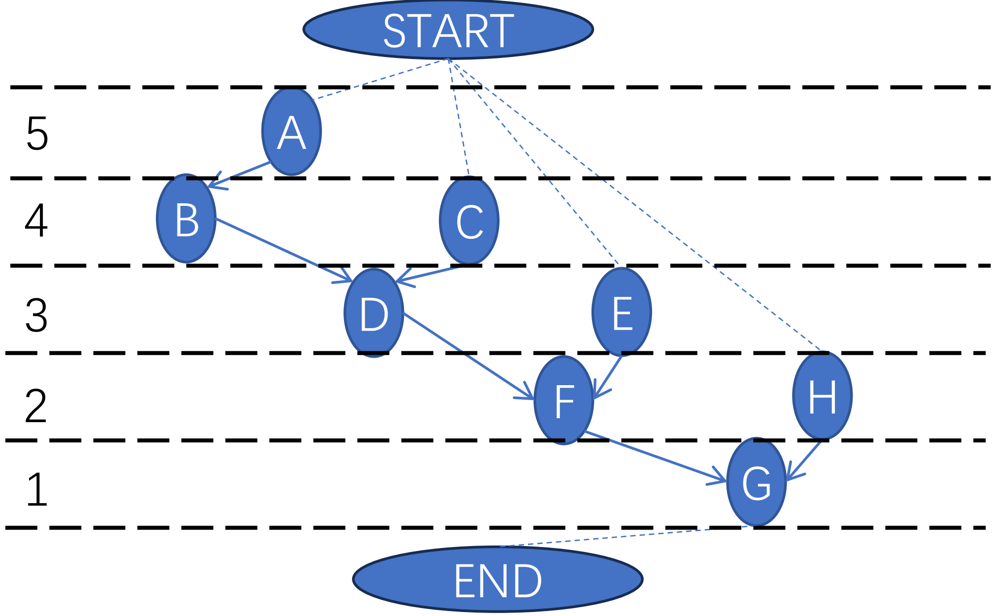
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	3
U	{D}
S	{D}

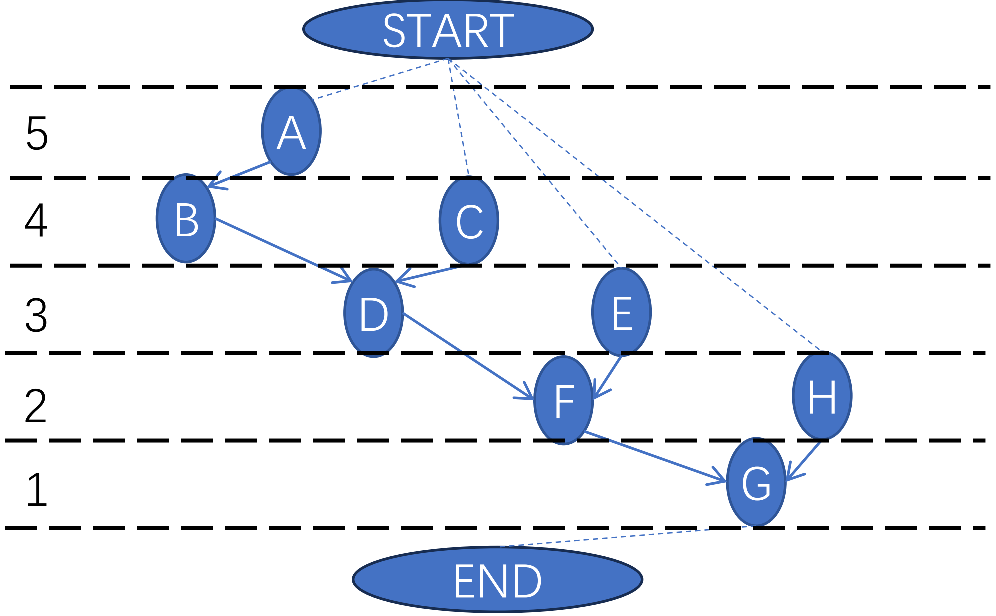


A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3

HU (G(V, E), a) {
  对顶点进行标号；
  t0 = 0；l = 1;
  重复执行以下步骤：
    U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度；
    选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大；
    在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;
    l = l + 1;
  直到 vn 被调度；
}

变量	当前值
l	4
U	{D}
S	{D}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3

HU (G(V, E), a) {

  对顶点进行标号;

  t0 = 0; l = 1;

  重复执行以下步骤:

    U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

    选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

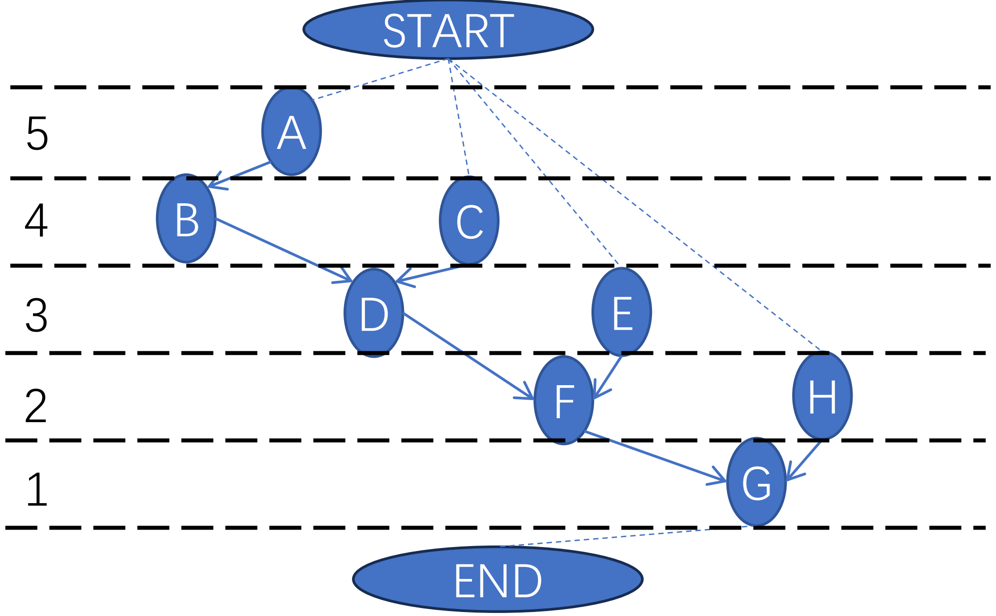
    在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

    l = l + 1;

  直到 vn 被调度;

}

变量	当前值
l	4
U	{F}
S	{D}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3
F 4

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

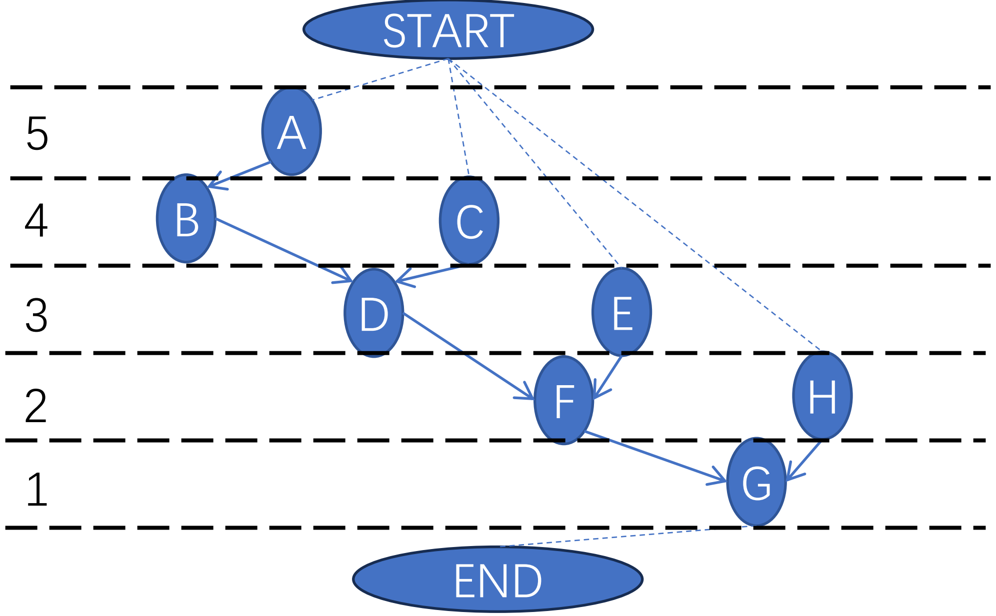
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	4
U	{F}
S	{F}



A 5	START 0
B 4	A 1
C 4	C 1
D 3	E 1
E 3	H 2
F 2	B 2
H 2	D 3
G 1	F 4

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

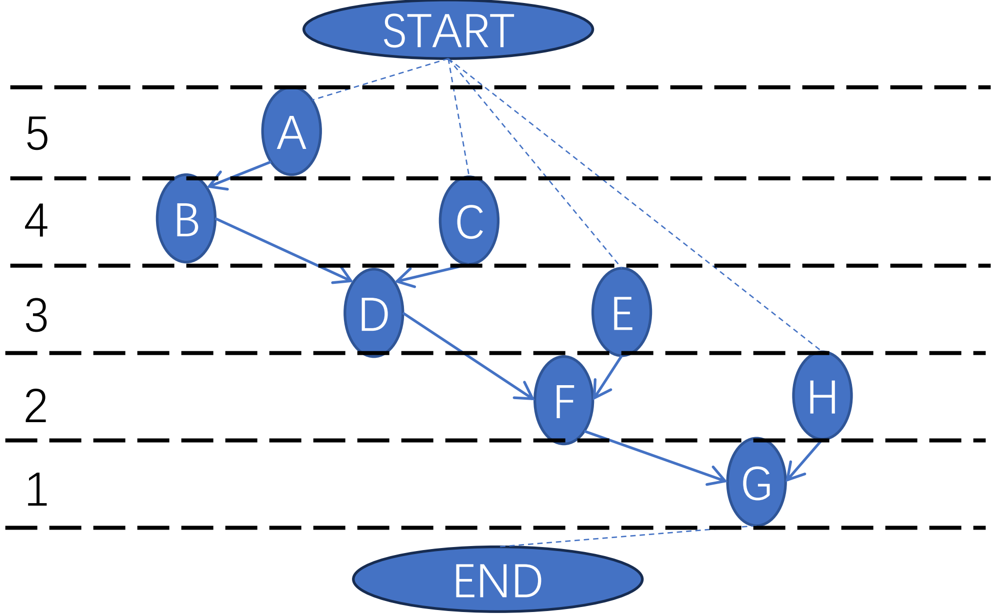
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	5
U	{F}
S	{F}



A 5	START 0
B 4	A 1
C 4	C 1
D 3	E 1
E 3	H 2
F 2	B 2
H 2	D 3
G 1	F 4

HU (G(V, E), a) {

  对顶点进行标号;

  t0 = 0; l = 1;

  重复执行以下步骤:

    U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

    选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

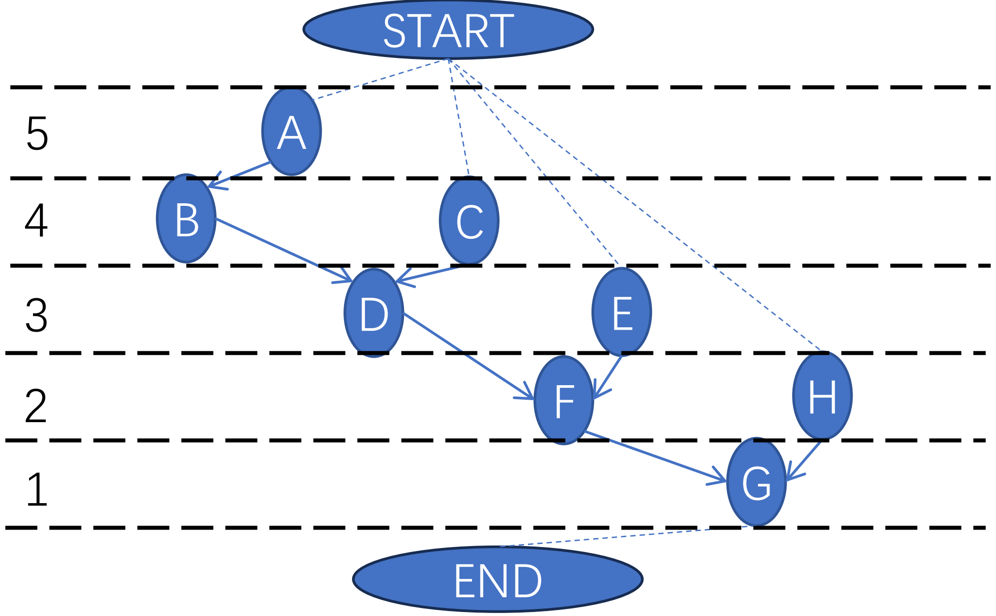
    在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

    l = l + 1;

  直到 vn 被调度;

}

变量	当前值
l	5
U	{G}
S	{F}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3
F 4
G 5

HU (G(V, E), a) {

  对顶点进行标号;

  t0 = 0; l = 1;

  重复执行以下步骤:

    U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

    选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

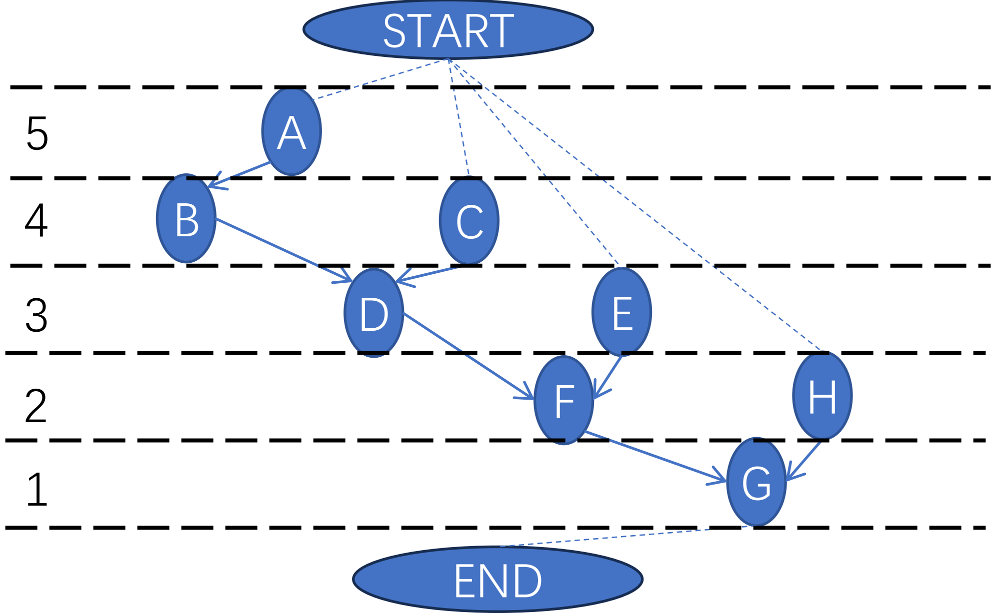
    在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

    l = l + 1;

  直到 vn 被调度;

}

变量	当前值
l	5
U	{G}
S	{G}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3
F 4
G 5

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

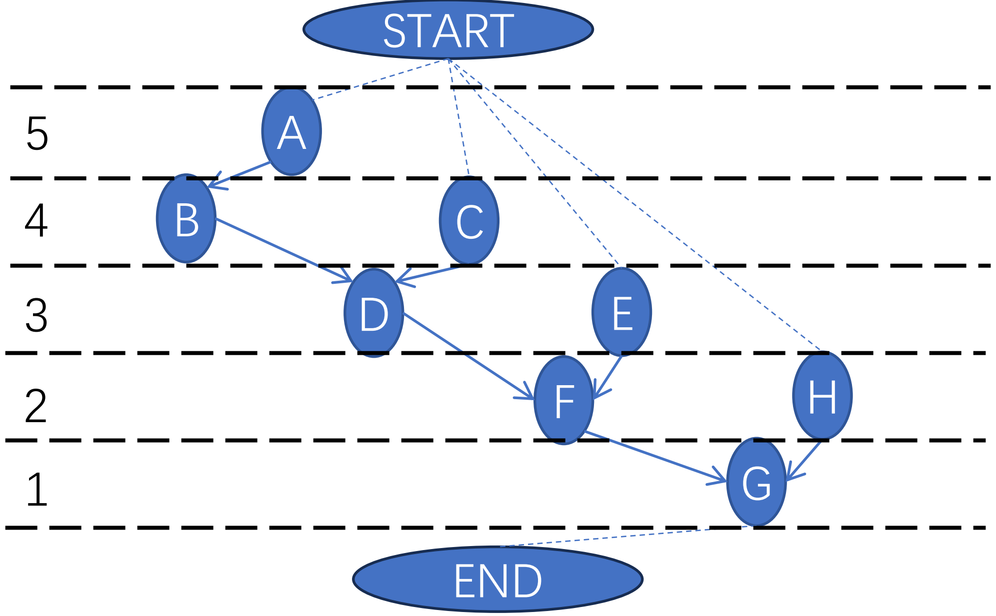
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	6
U	{G}
S	{G}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3
F 4
G 5

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

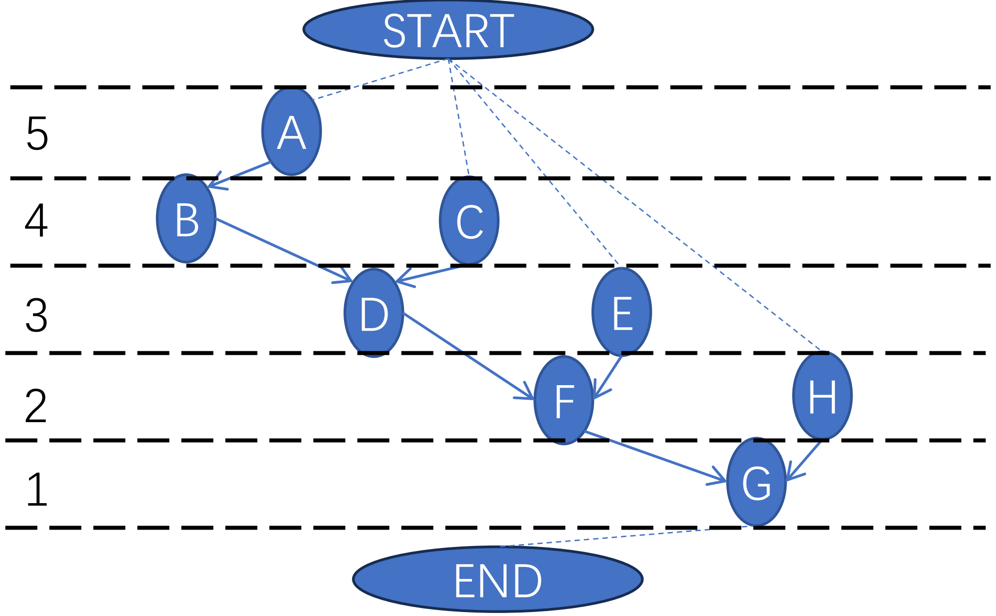
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	6
U	{END}
S	{G}

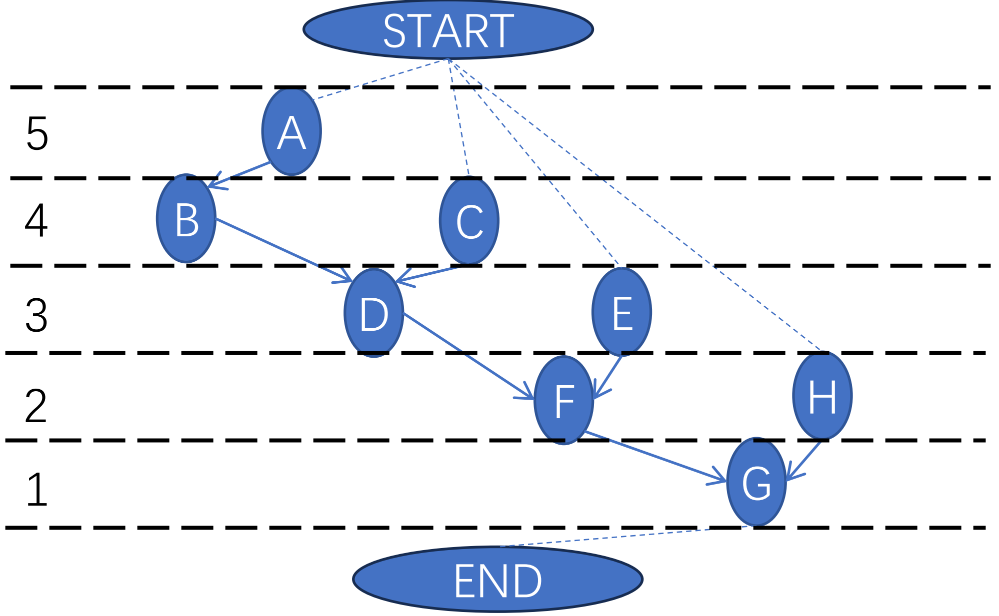


A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3
F 4
G 5
END 6

HU (G(V, E), a) {
  对顶点进行标号;
  t0 = 0; l = 1;
  重复执行以下步骤:
    U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;
    选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;
    在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;
    l = l + 1;
  直到 vn 被调度;
}

变量	当前值
l	6
U	{END}
S	{END}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3
F 4
G 5
END 6

HU (G(V, E), a) {

对顶点进行标号;

t0 = 0; l = 1;

重复执行以下步骤:

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度;

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大;

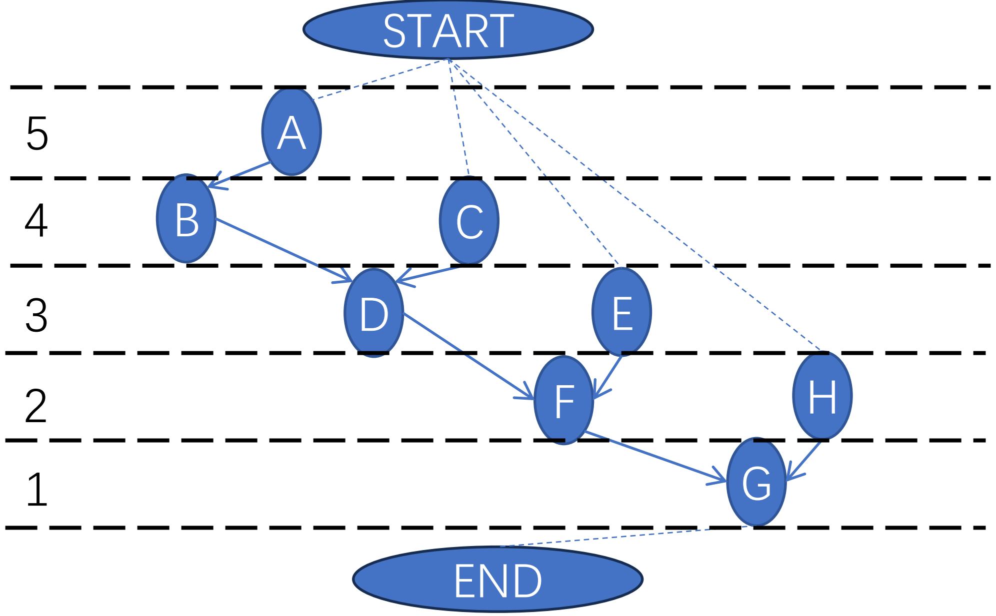
在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度;

}

变量	当前值
l	7
U	{END}
S	{END}



A 5
B 4
C 4
D 3
E 3
F 2
H 2
G 1

START 0
A 1
C 1
E 1
H 2
B 2
D 3
F 4
G 5
END 6

HU (G(V, E), a) {

对顶点进行标号；

t0 = 0； l = 1;

重复执行以下步骤：

U = 仍未调度的顶点集合，这些顶点要么没有前驱，要么其所有前驱都已被调度；

选择一个子集 S ⊆ U，使得 |S| ≤ a，且 S 中的顶点标签值最大；

在步骤 t 处调度 S 中的所有操作，即对所有 vi ∈ S 设定调度时间 ti = l;

l = l + 1;

直到 vn 被调度；

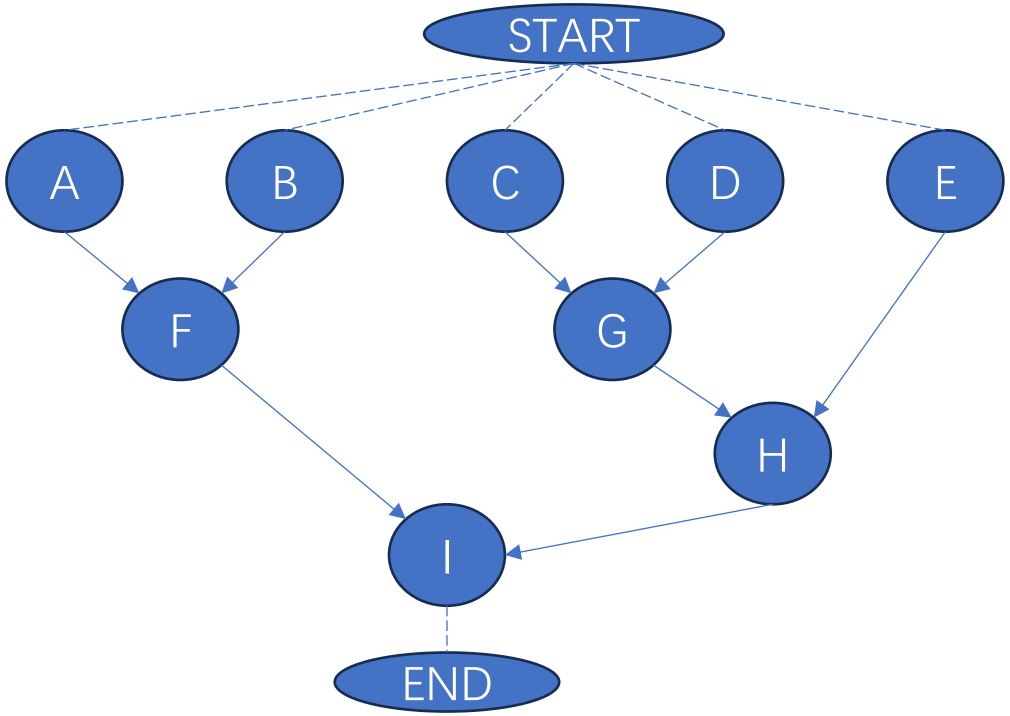
}

变量	当前值
l	7
U	{END}
S	{END}

作业2

in-class assignment

已知输入的图如下（每个都是加法操作）：



假设我们有两个加法器，请写出HU调度下的程序**的输出文件的内容**，输出文件格式如下:

每一行由一个字符串和一个数字组成，以空格隔开，分别代表顶点名和开始的调度周期