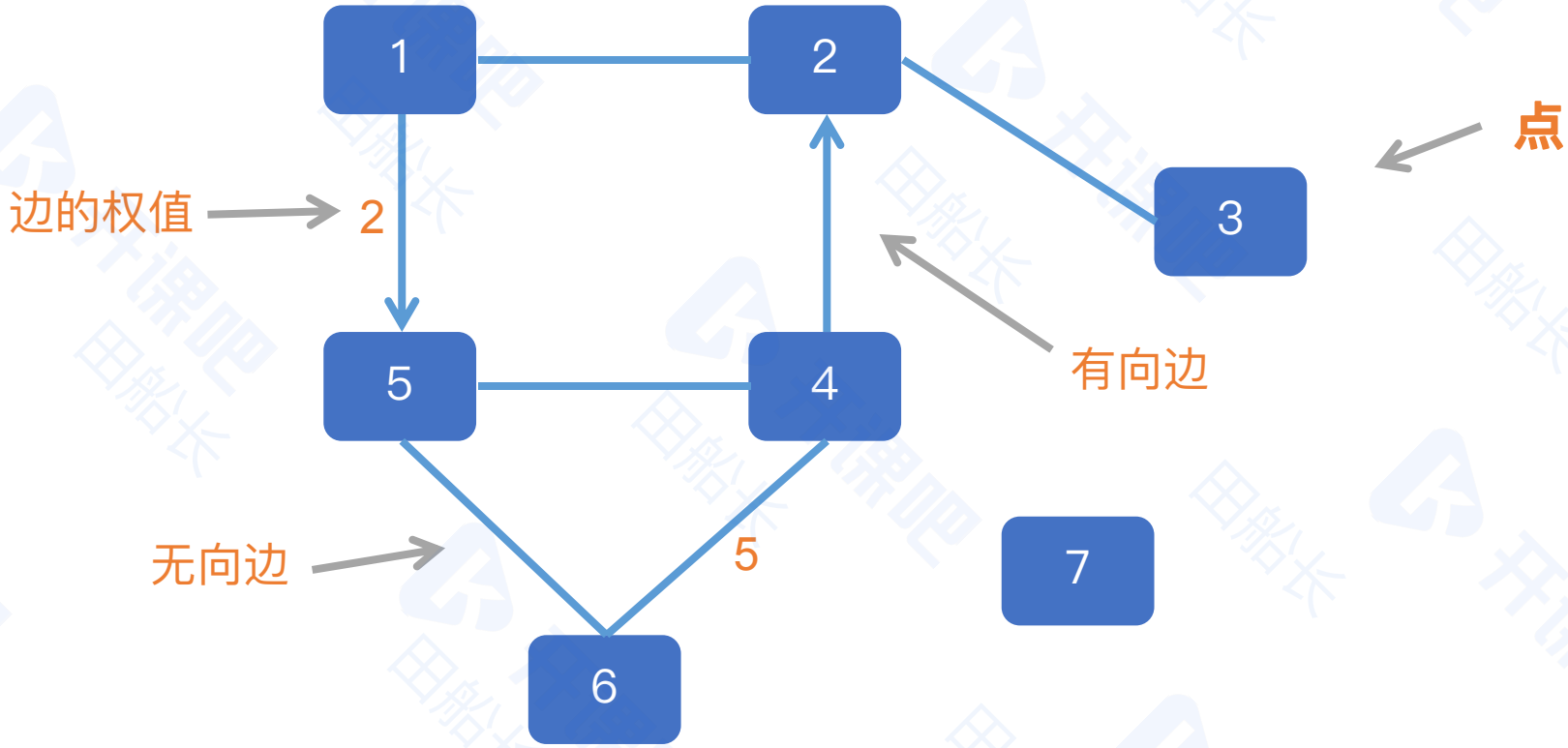
开课吧

图

田船长

普通的图

数据结构中的图



图是由点和边组成的集合，边记录的是节点之间的关系

图的基本术语

1. 节点，边（弧）
2. 有向边，无向边，有向图，无向图，完全图，子图
3. 度，入度，出度
4. 路径，回路
5. 连通图，强连通图，连通分量，强连通分量
6. 权值，带权图（网）

图的存储——邻接矩阵

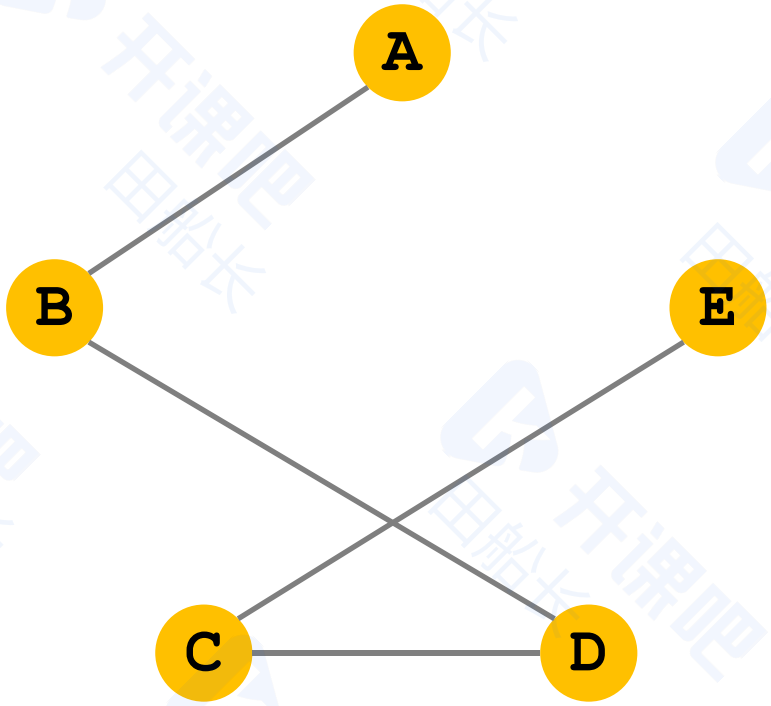
```
#define MaxCnt 50 //节点最大数量
struct Graph { //图的定义
    EleType Vex[MaxCnt]; //节点表
    int arr[MaxCnt][MaxCnt]; //邻接矩阵
    int vexnum, edgnum; //节点元素数量与边的数量
};
```

邻接矩阵是一个多维数组，存储节点间边的关系

设图的邻接矩阵为A，An的元素An[i][j]等于

由顶点i到顶点j的长度为n的路径数目（不绝对）

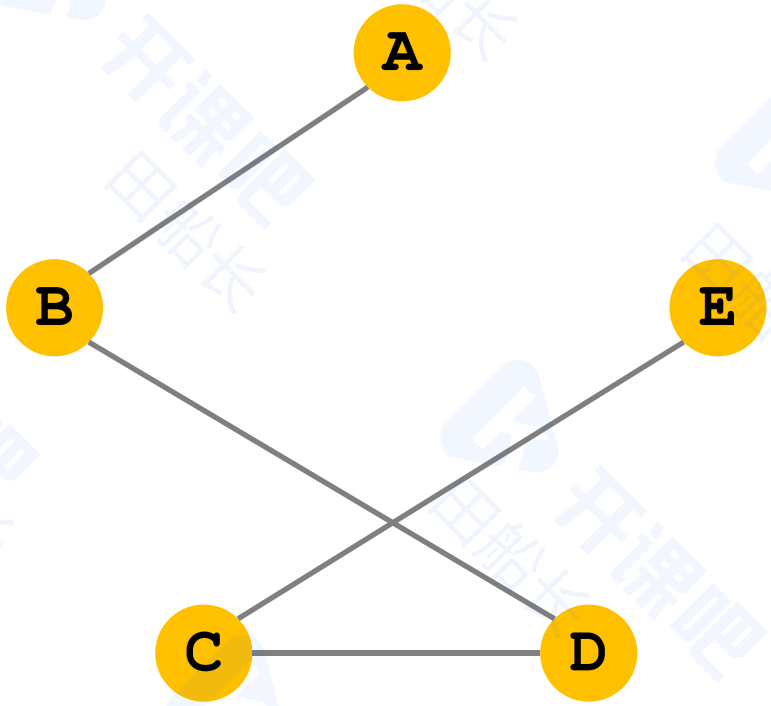
邻接矩阵（无向图）



	A	B	C	D	E
A					
B					
C					
D					
E					

邻接矩阵（无向图）

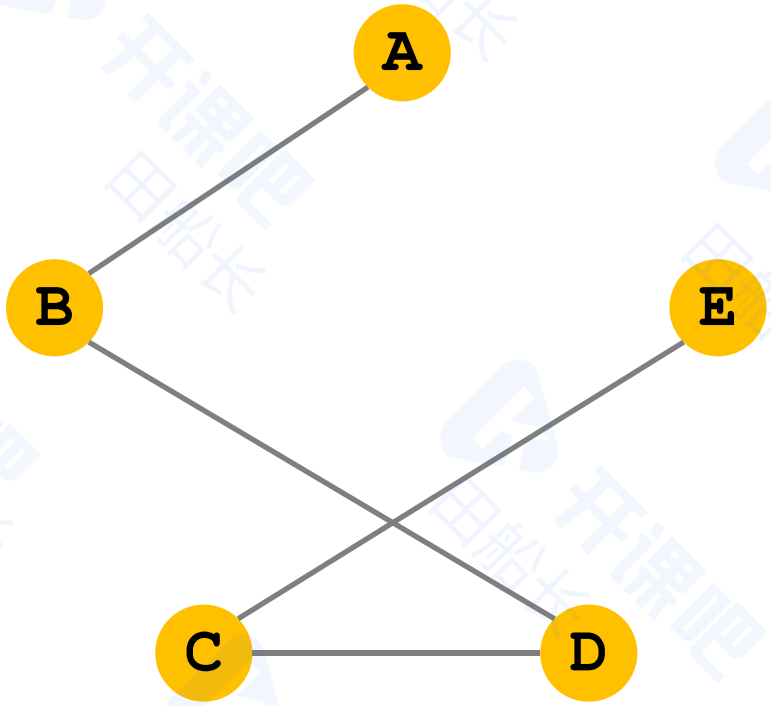
[任务]构建邻接矩阵（无向图）



	A	B	C	D	E
A					
B					
C					
D					
E					

邻接矩阵（无向图）

[任务]构建邻接矩阵（无向图）

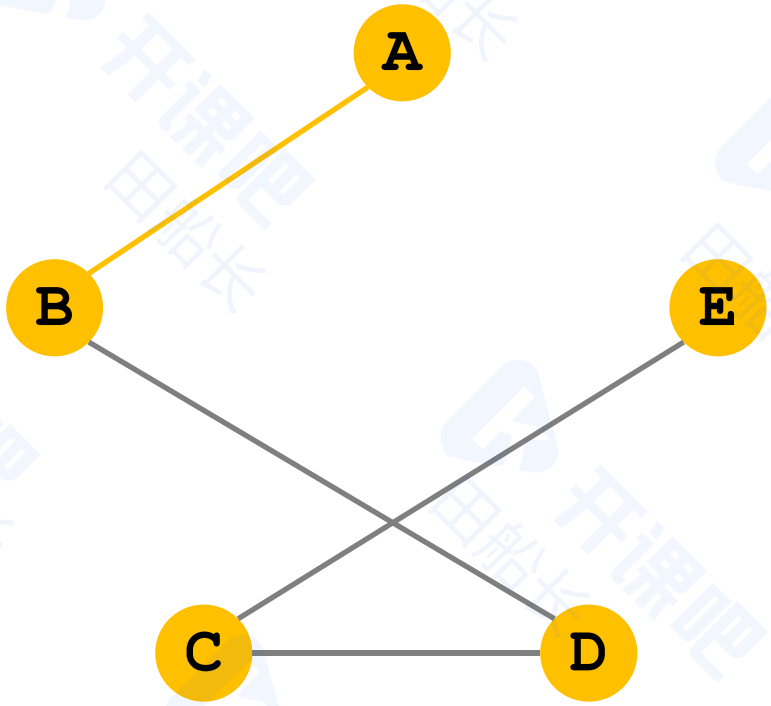


初始化邻接矩阵（无向图）

	A	B	C	D	E
A	0	0	0	0	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

邻接矩阵（无向图）

[任务]构建邻接矩阵（无向图）

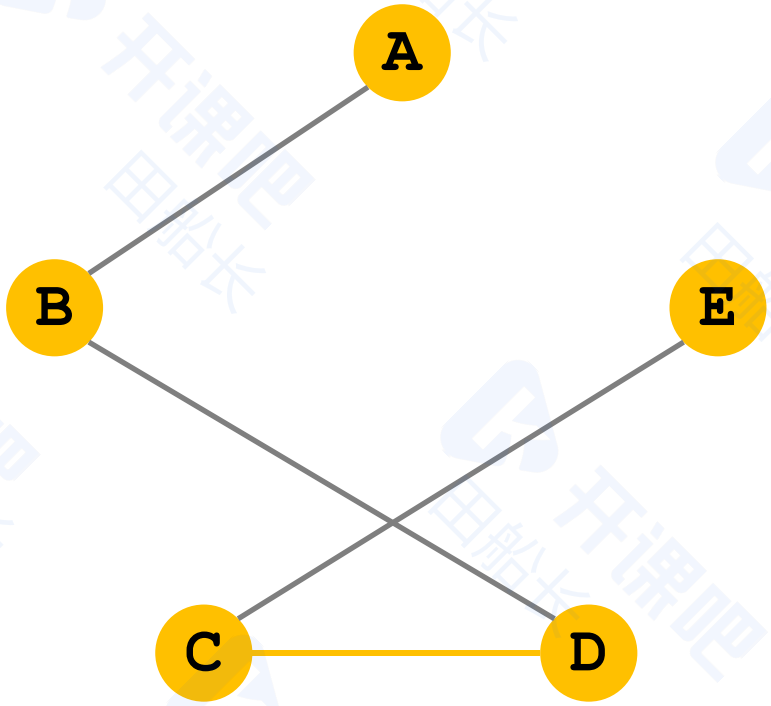


邻接矩阵中记录无向边AB

	A	B	C	D	E
A	0	1	0	0	0
B	1	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

邻接矩阵（无向图）

[任务]构建邻接矩阵（无向图）

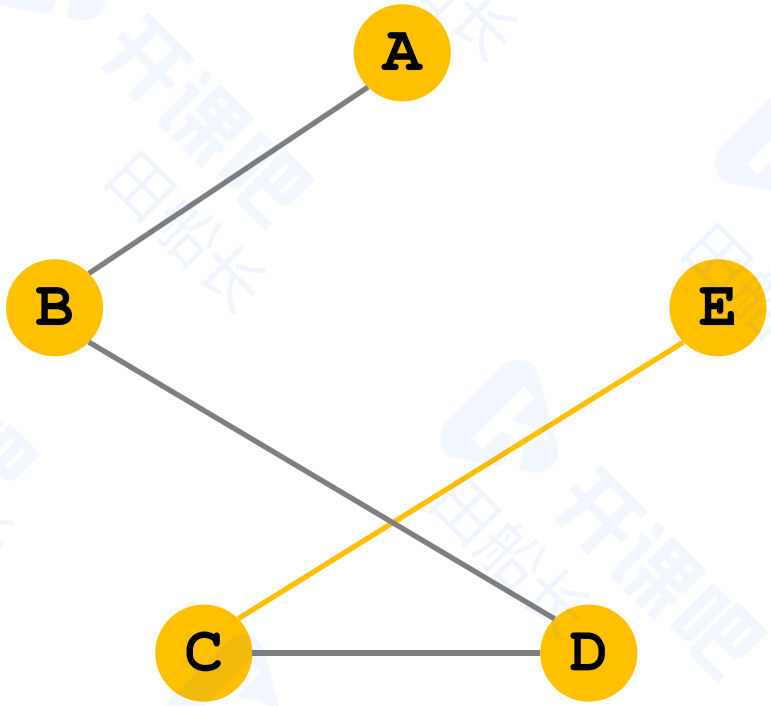


邻接矩阵中记录无向边CD

	A	B	C	D	E
A	0	1	0	0	0
B	1	0	0	0	0
C	0	0	0	1	0
D	0	0	1	0	0
E	0	0	0	0	0

邻接矩阵（无向图）

[任务]构建邻接矩阵（无向图）

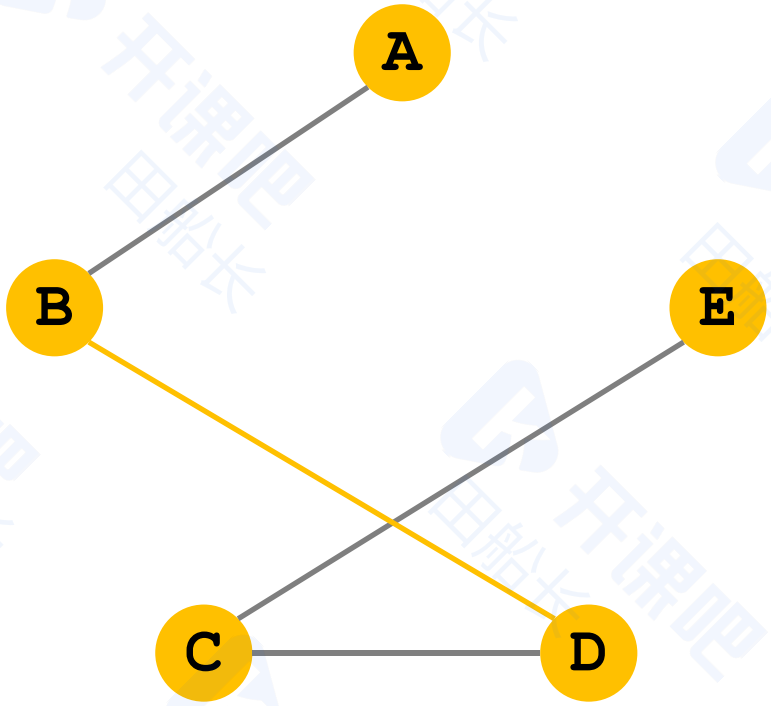


邻接矩阵中记录无向边CE

	A	B	C	D	E
A	0	1	0	0	0
B	1	0	0	0	0
C	0	0	0	1	1
D	0	0	1	0	0
E	0	0	1	0	0

邻接矩阵（无向图）

[任务]构建邻接矩阵（无向图）

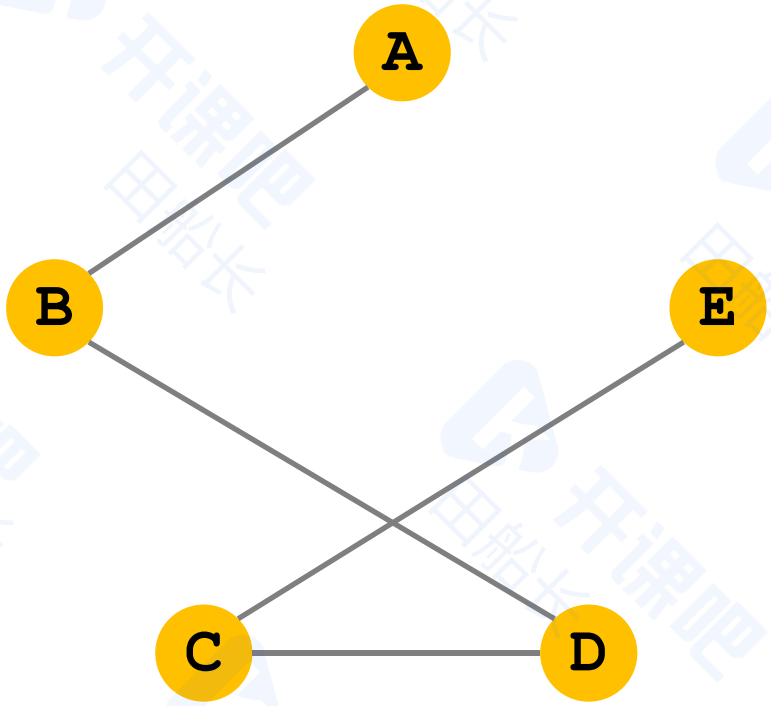


邻接矩阵中记录无向边BD

	A	B	C	D	E
A	0	1	0	0	0
B	1	0	0	1	0
C	0	0	0	1	1
D	0	1	1	0	0
E	0	0	1	0	0

邻接矩阵（无向图）

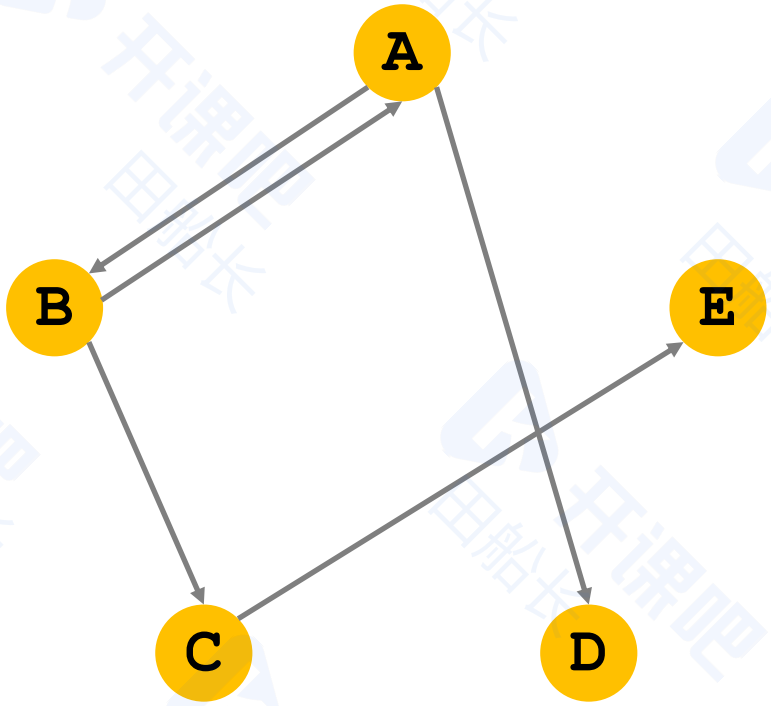
[任务]构建邻接矩阵（无向图）



邻接矩阵（无向图）构建完毕

	A	B	C	D	E
A	0	1	0	0	0
B	1	0	0	1	0
C	0	0	0	1	1
D	0	1	1	0	0
E	0	0	1	0	0

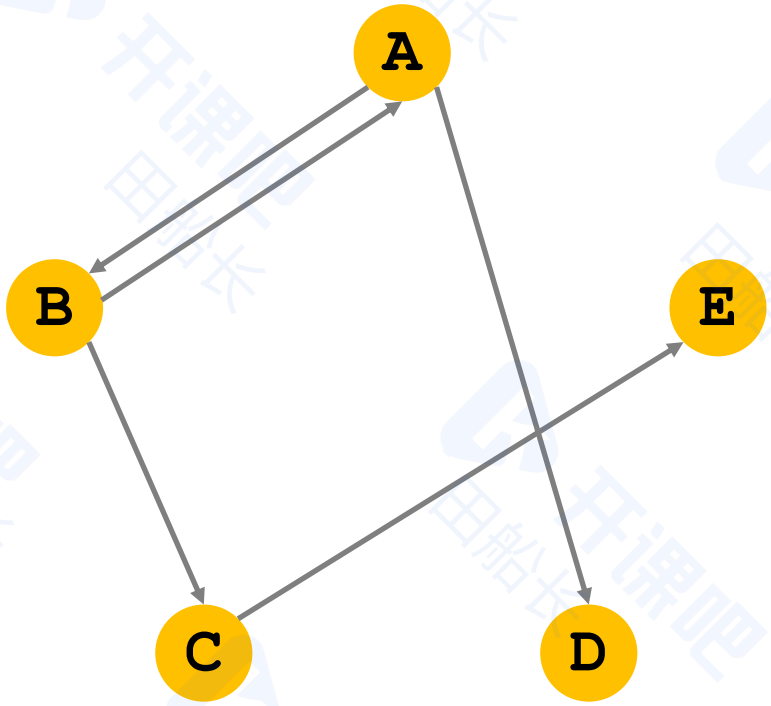
邻接矩阵（有向图）



	A	B	C	D	E
A					
B					
C					
D					
E					

邻接矩阵（有向图）

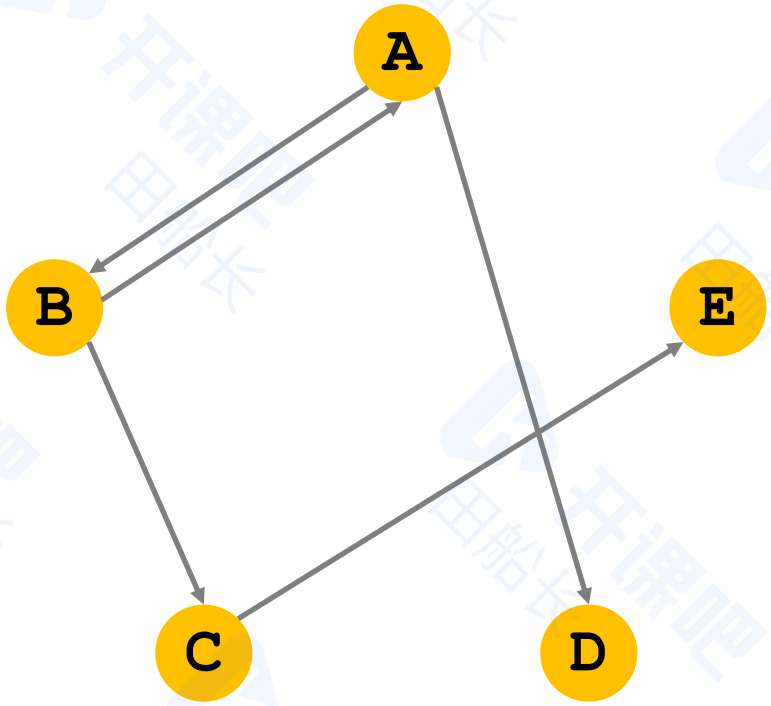
[任务]构建邻接矩阵（有向图）



	A	B	C	D	E
A					
B					
C					
D					
E					

邻接矩阵（有向图）

[任务]构建邻接矩阵（有向图）

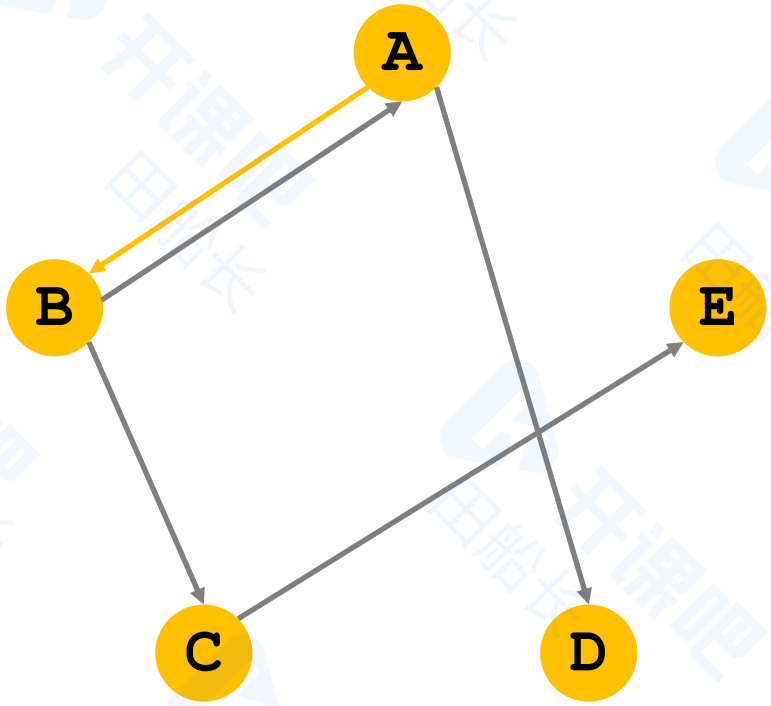


初始化邻接矩阵（有向图）

	A	B	C	D	E
A	0	0	0	0	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

邻接矩阵（有向图）

[任务]构建邻接矩阵（有向图）

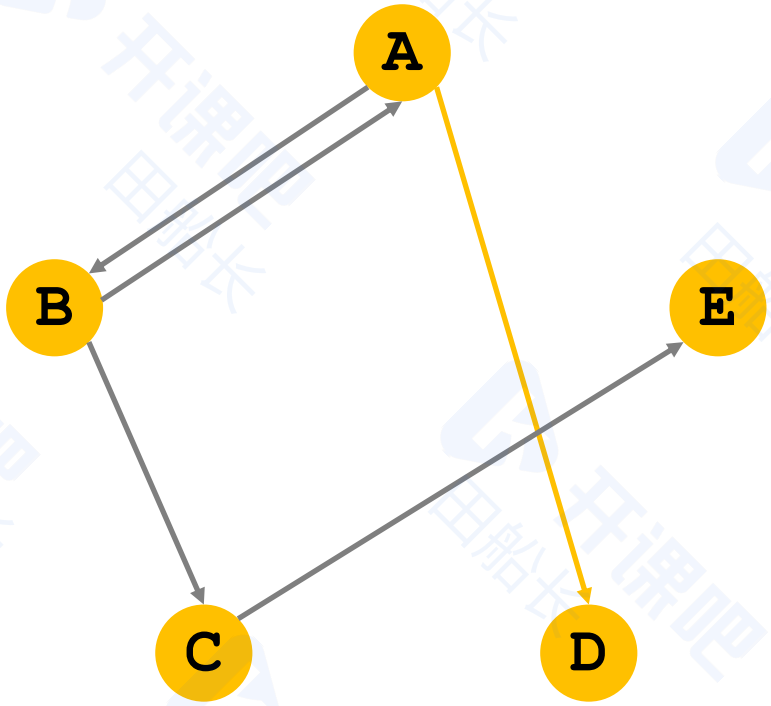


邻接矩阵中记录有向边AB

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

邻接矩阵（有向图）

[任务]构建邻接矩阵（有向图）

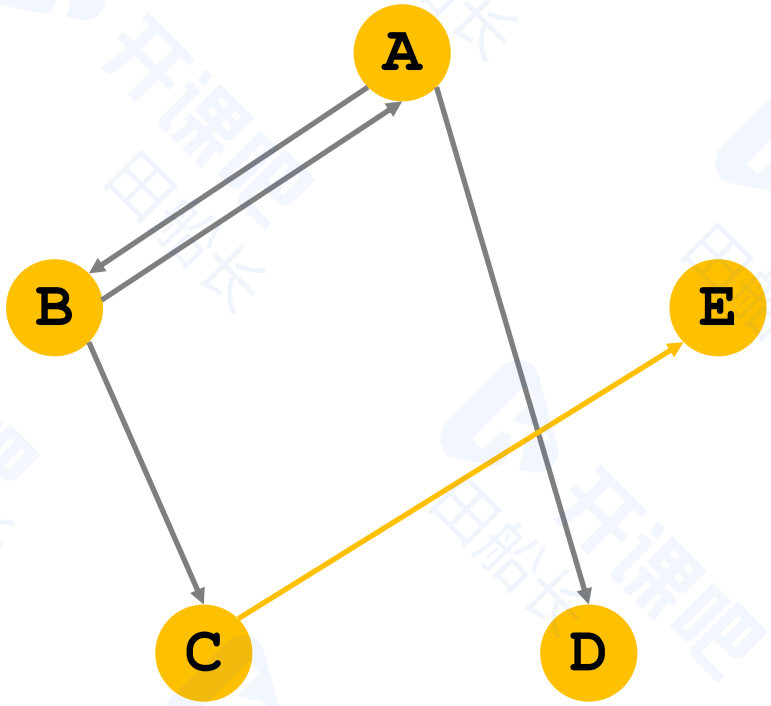


邻接矩阵中记录有向边AD

	A	B	C	D	E
A	0	1	0	1	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

邻接矩阵（有向图）

[任务]构建邻接矩阵（有向图）

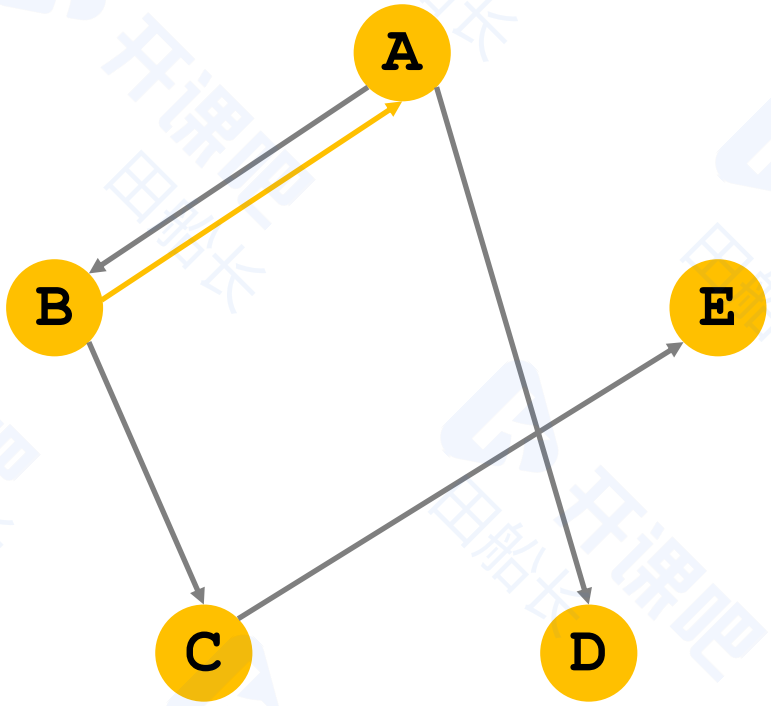


邻接矩阵中记录有向边CE

	A	B	C	D	E
A	0	1	0	1	0
B	0	0	0	0	0
C	0	0	0	0	1
D	0	0	0	0	0
E	0	0	0	0	0

邻接矩阵（有向图）

[任务]构建邻接矩阵（有向图）

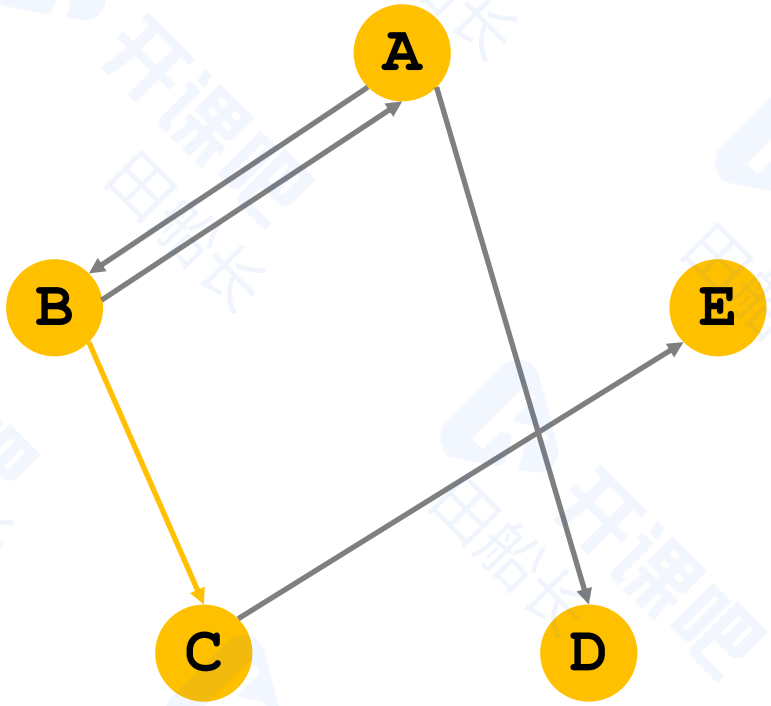


邻接矩阵中记录有向边BA

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	0	0	0
C	0	0	0	0	1
D	0	0	0	0	0
E	0	0	0	0	0

邻接矩阵（有向图）

[任务]构建邻接矩阵（有向图）

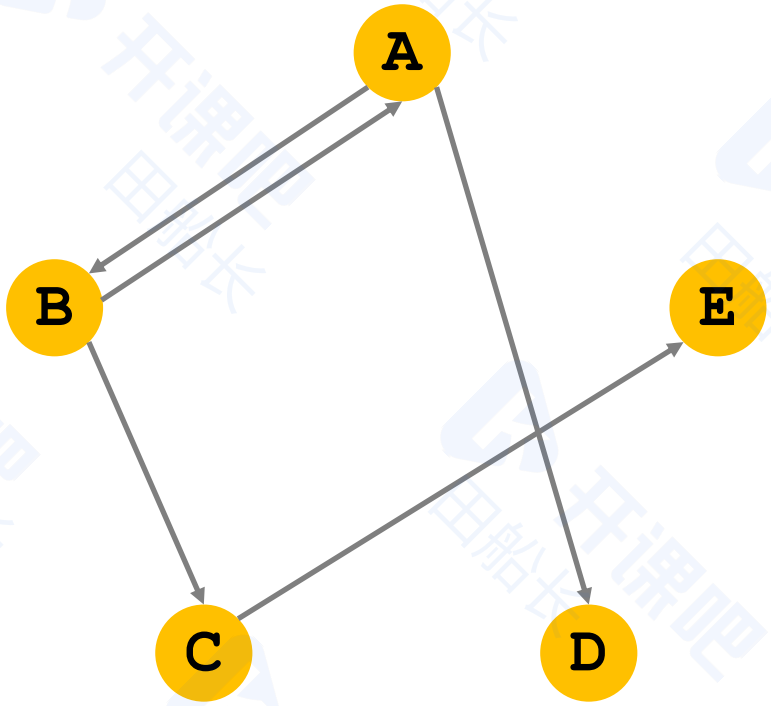


邻接矩阵中记录有向边BC

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	0	0
C	0	0	0	0	1
D	0	0	0	0	0
E	0	0	0	0	0

邻接矩阵（有向图）

[任务]构建邻接矩阵（有向图）



邻接矩阵（有向图）构建完毕

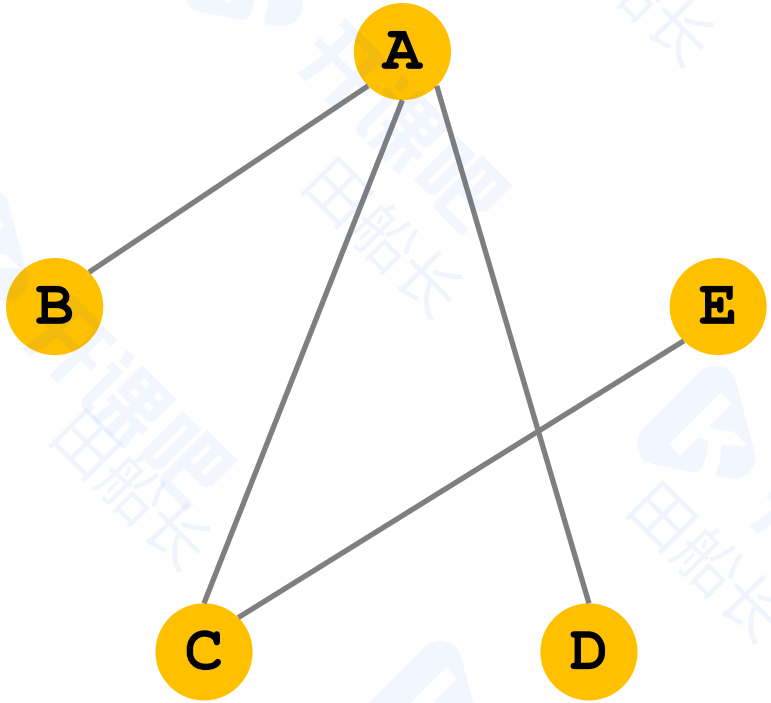
	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	0	0
C	0	0	0	0	1
D	0	0	0	0	0
E	0	0	0	0	0

图的存储——邻接表

```
#define MaxCnt 50 //节点最大数量
struct Node { //边在链表中的结点
    int adjvex; //该条边的终点
    Node *next; //指针域（同起点的下一条边）
};
struct Graph {
    EleType Vex[MaxCnt]; //节点表
    Node *first[MaxCnt]; //以每个节点为起点的所有边组成的链表
};
```

邻接表存储了与每个节点有关的所有边
与同一个节点有关的边都存到同一个链表中

邻接表（无向图）

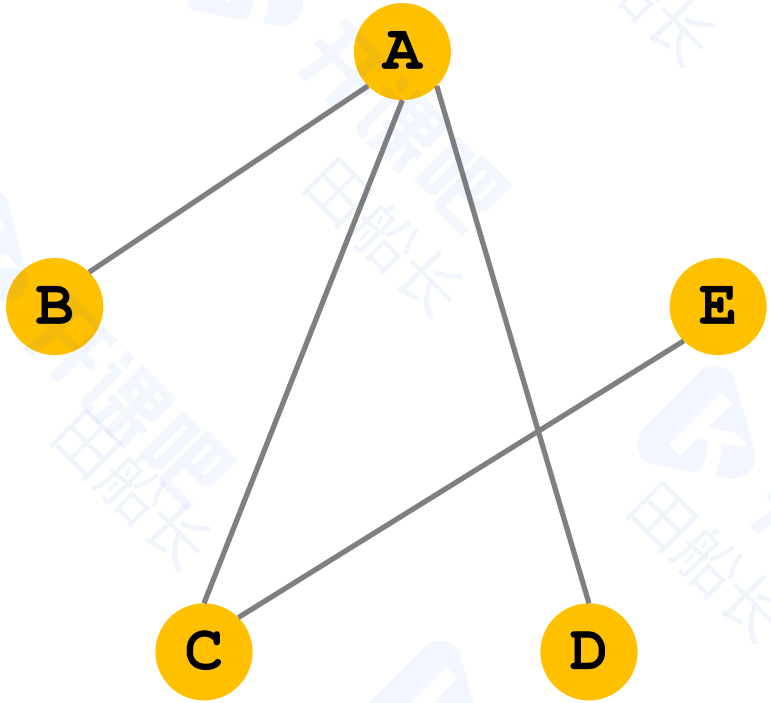


指向第一条边

	data	*first
0	A	^
1	B	^
2	C	^
3	D	^
4	E	^

邻接表（无向图）

[任务]构建邻接表（无向图）

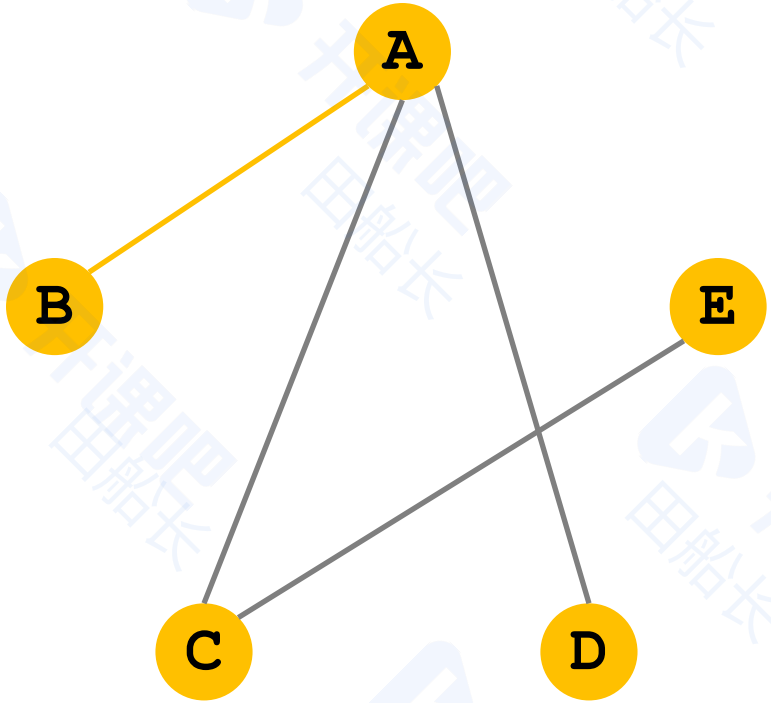


指向第一条边

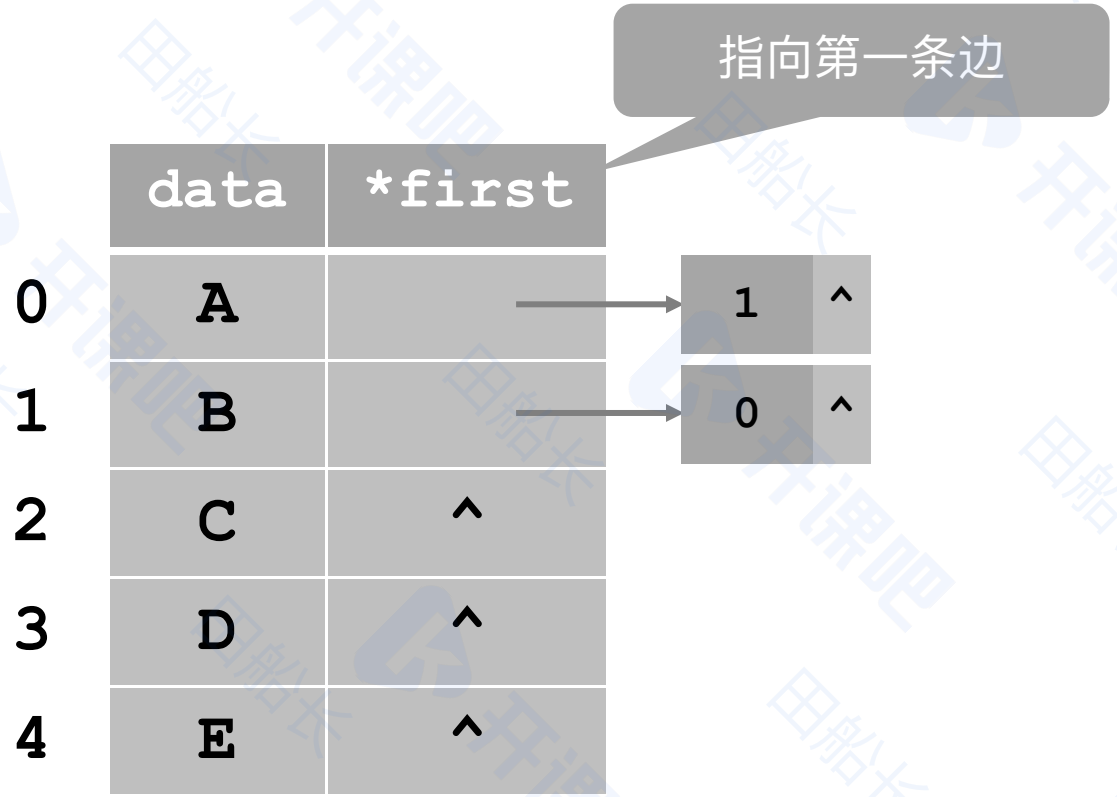
	data	*first
0	A	^
1	B	^
2	C	^
3	D	^
4	E	^

邻接表（无向图）

[任务]构建邻接表（无向图）

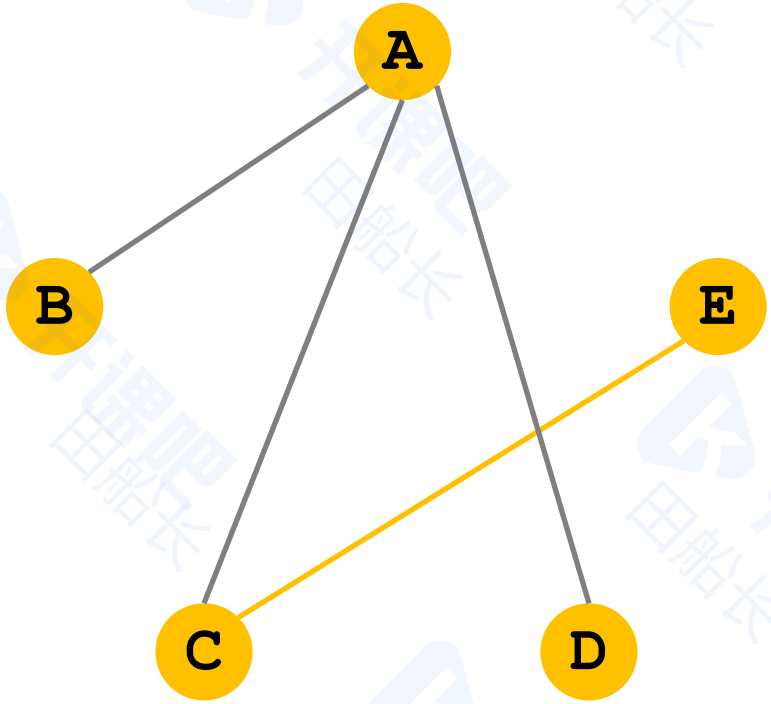


邻接表中记录无向边AB



邻接表（无向图）

[任务]构建邻接表（无向图）



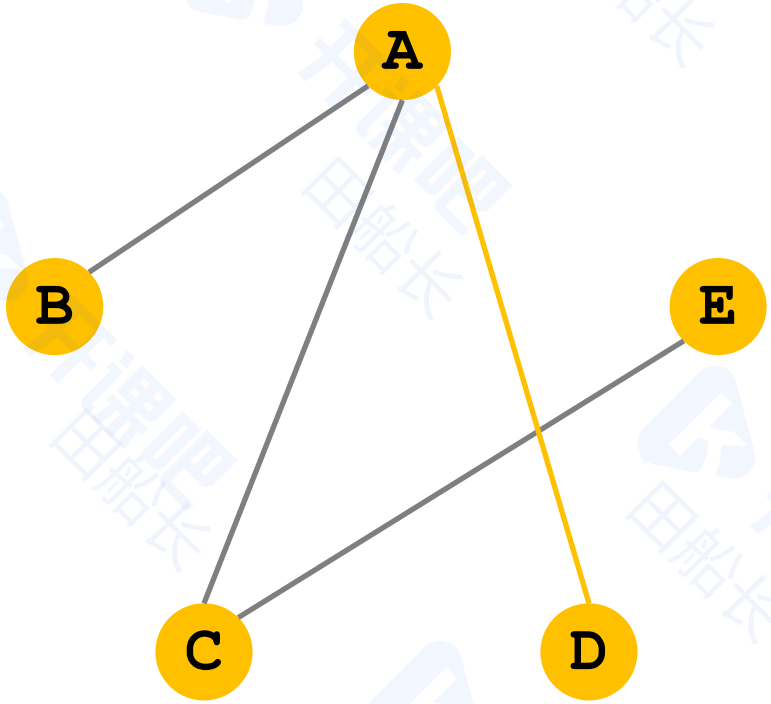
邻接表中记录无向边CE

指向第一条边

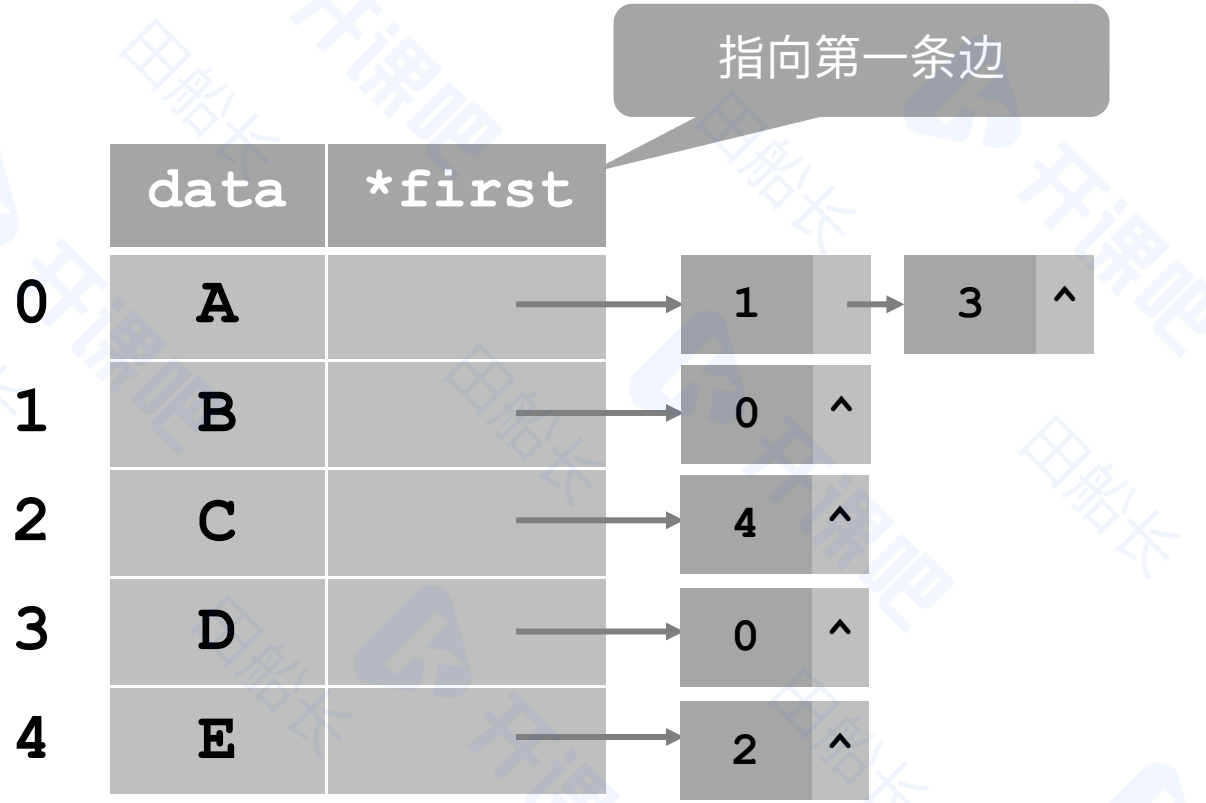
	data	*first
0	A	→ 1 ^
1	B	→ 0 ^
2	C	→ 4 ^
3	D	^
4	E	→ 2 ^

邻接表（无向图）

[任务]构建邻接表（无向图）

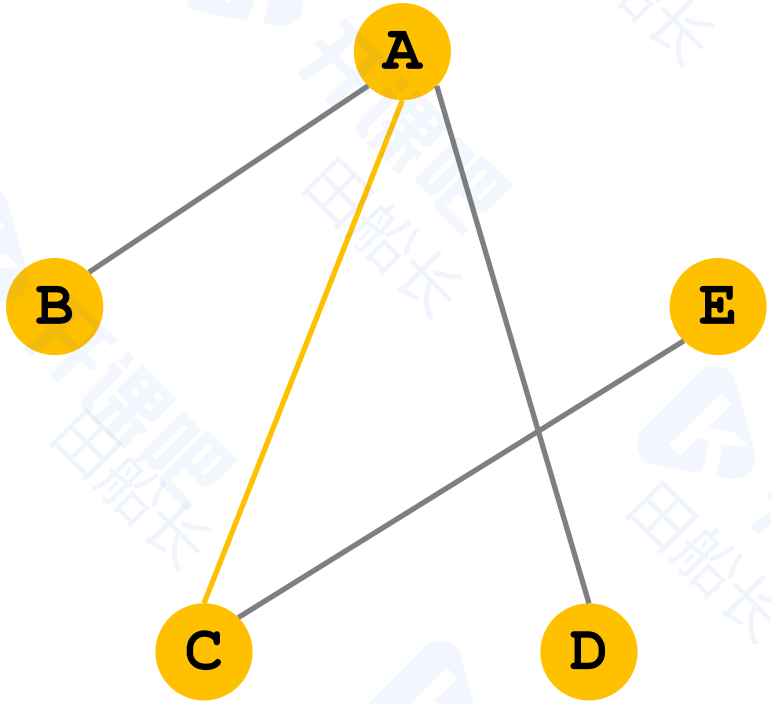


邻接表中记录无向边AD

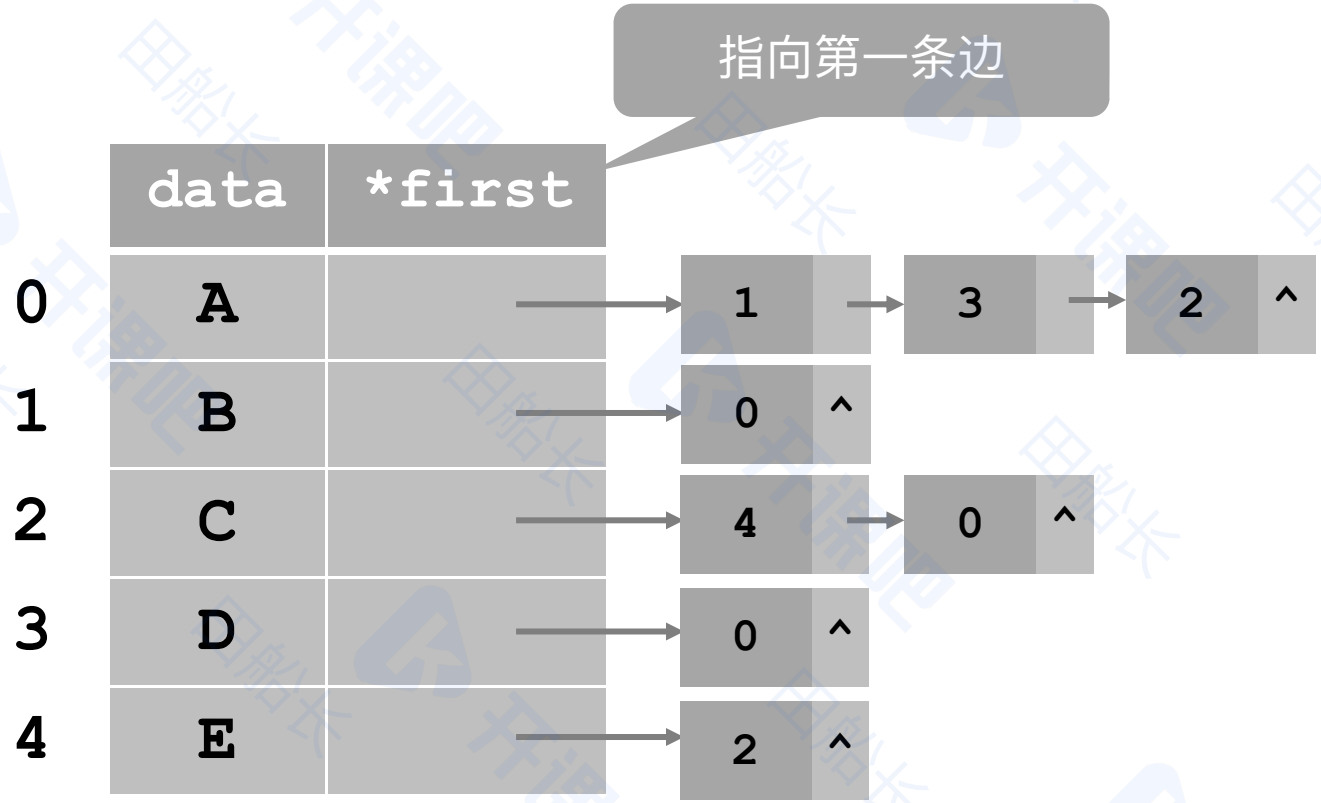


邻接表（无向图）

[任务]构建邻接表（无向图）

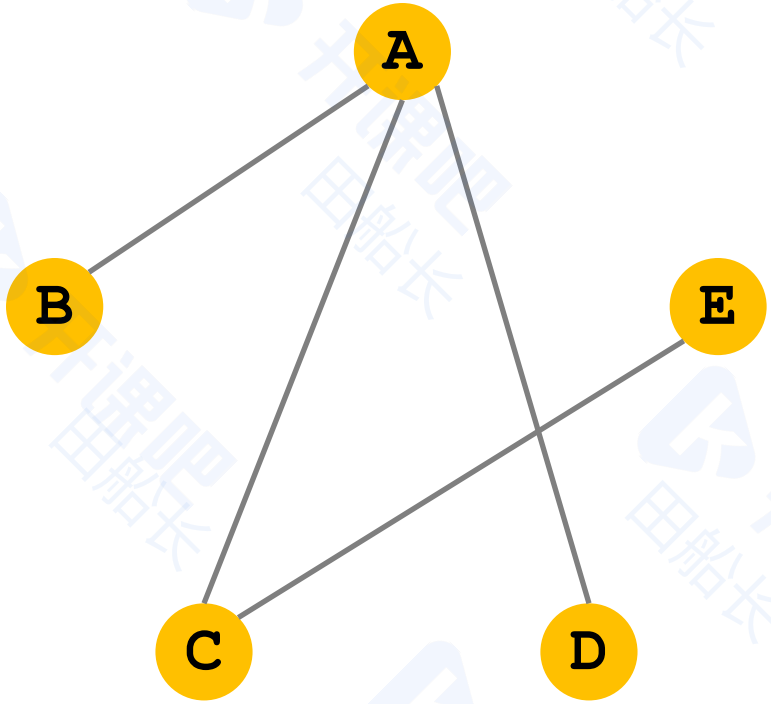


邻接表中记录无向边AC

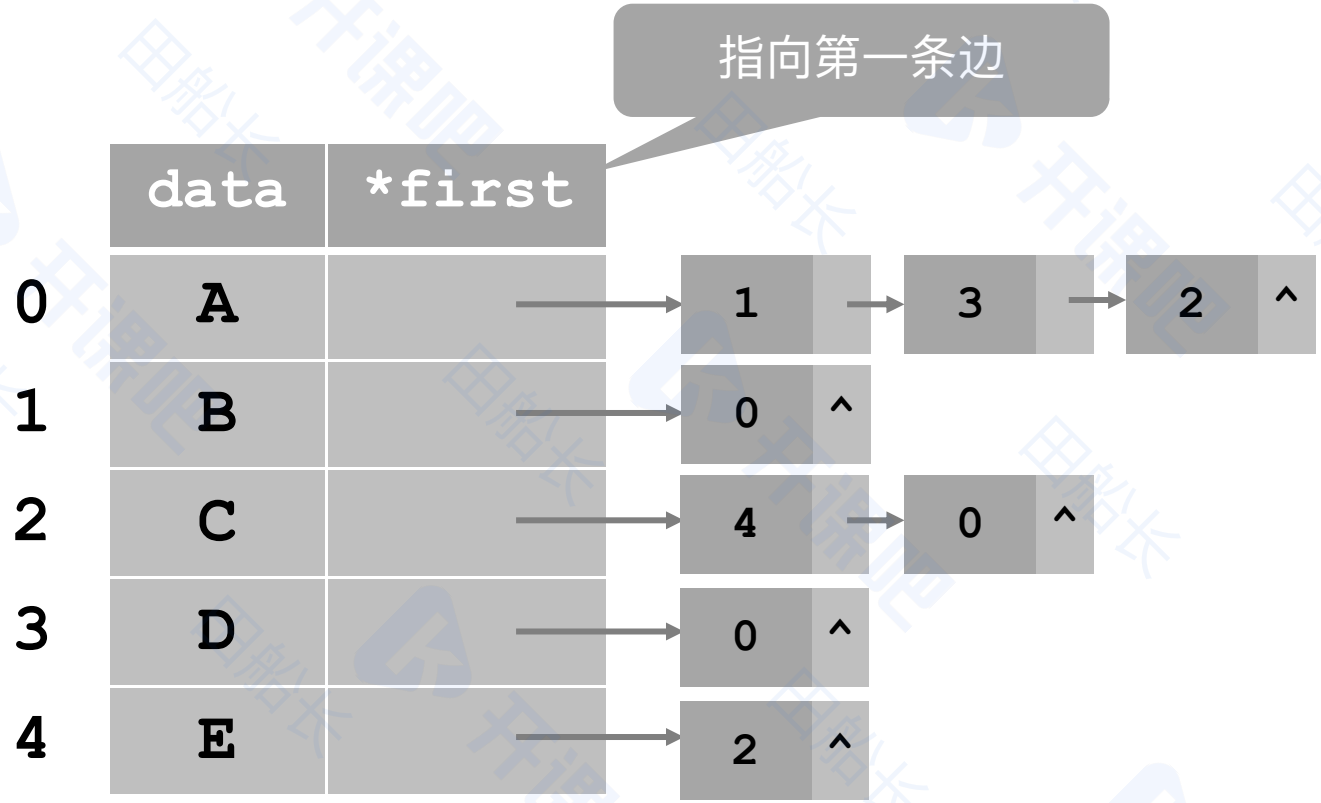


邻接表（无向图）

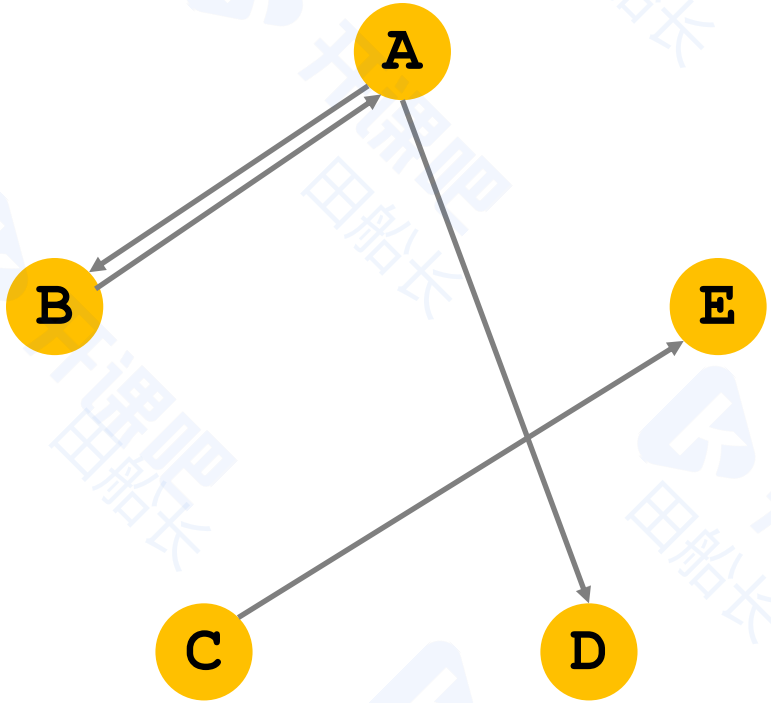
[任务]构建邻接表（无向图）



邻接表（无向图）构建完毕



邻接表（有向图）

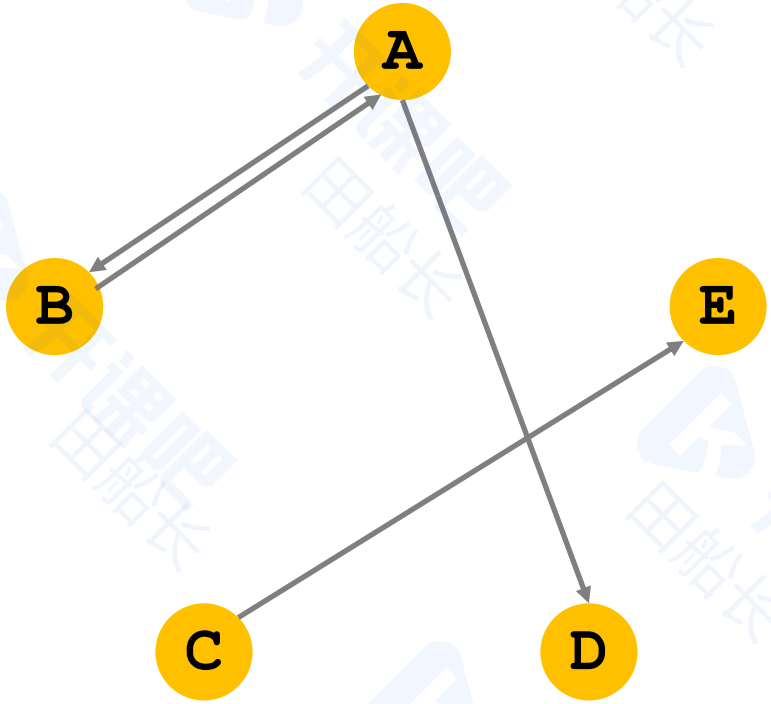


指向第一条边

	data	*first
0	A	^
1	B	^
2	C	^
3	D	^
4	E	^

邻接表（有向图）

[任务]构建邻接表（有向图）

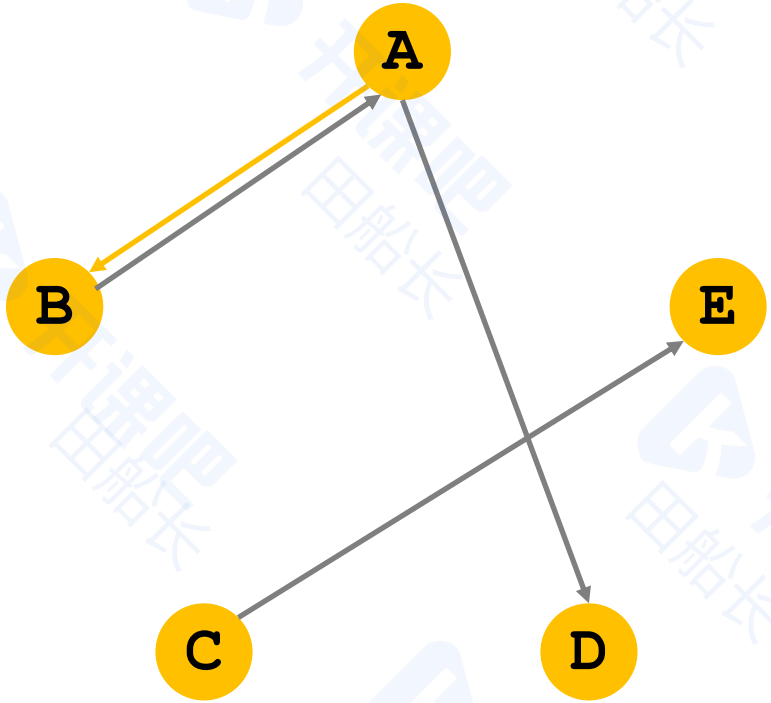


指向第一条边

	data	*first
0	A	^
1	B	^
2	C	^
3	D	^
4	E	^

邻接表（有向图）

[任务]构建邻接表（有向图）



邻接表中记录有向边AB

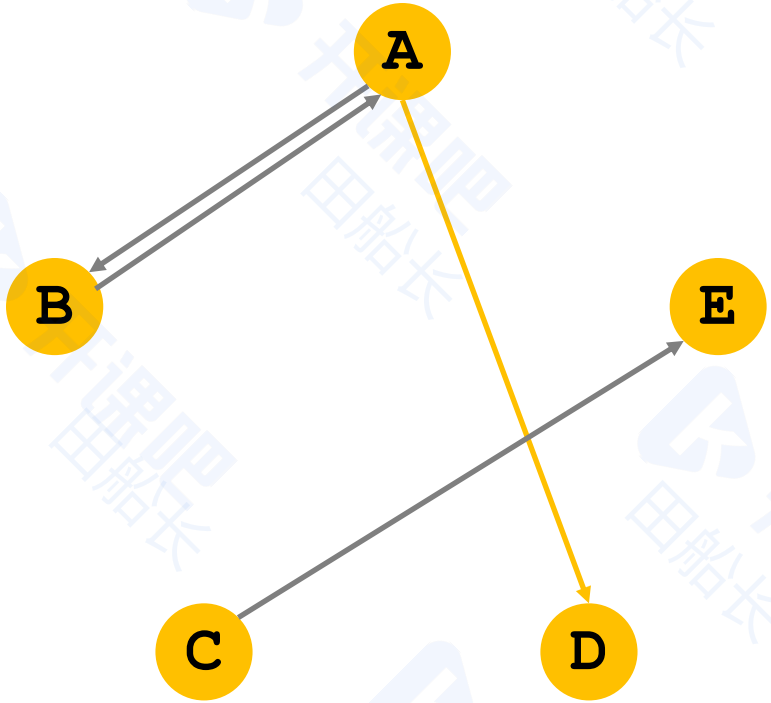
指向第一条边

	data	*first
0	A	
1	B	^
2	C	^
3	D	^
4	E	^

1 ^

邻接表（有向图）

[任务]构建邻接表（有向图）



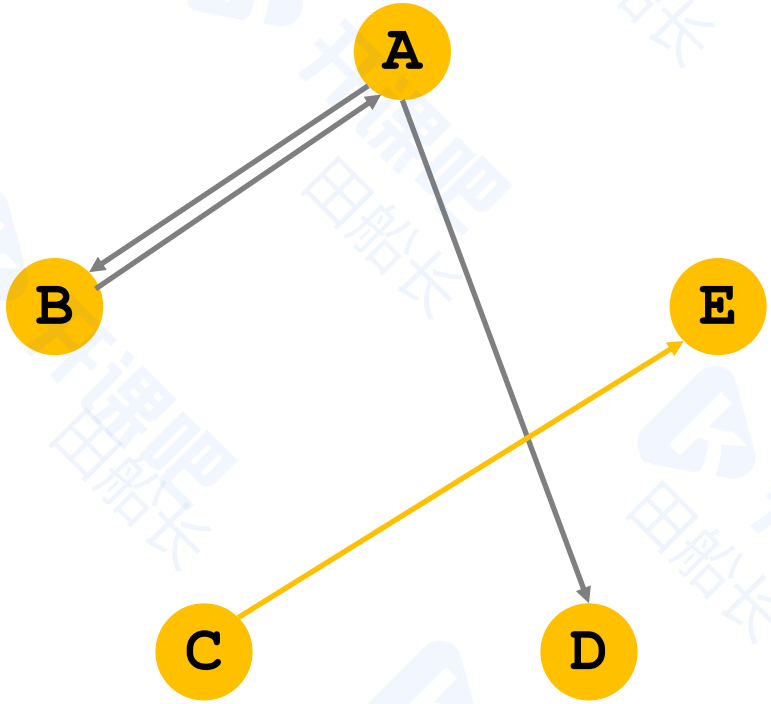
邻接表中记录有向边AD

指向第一条边

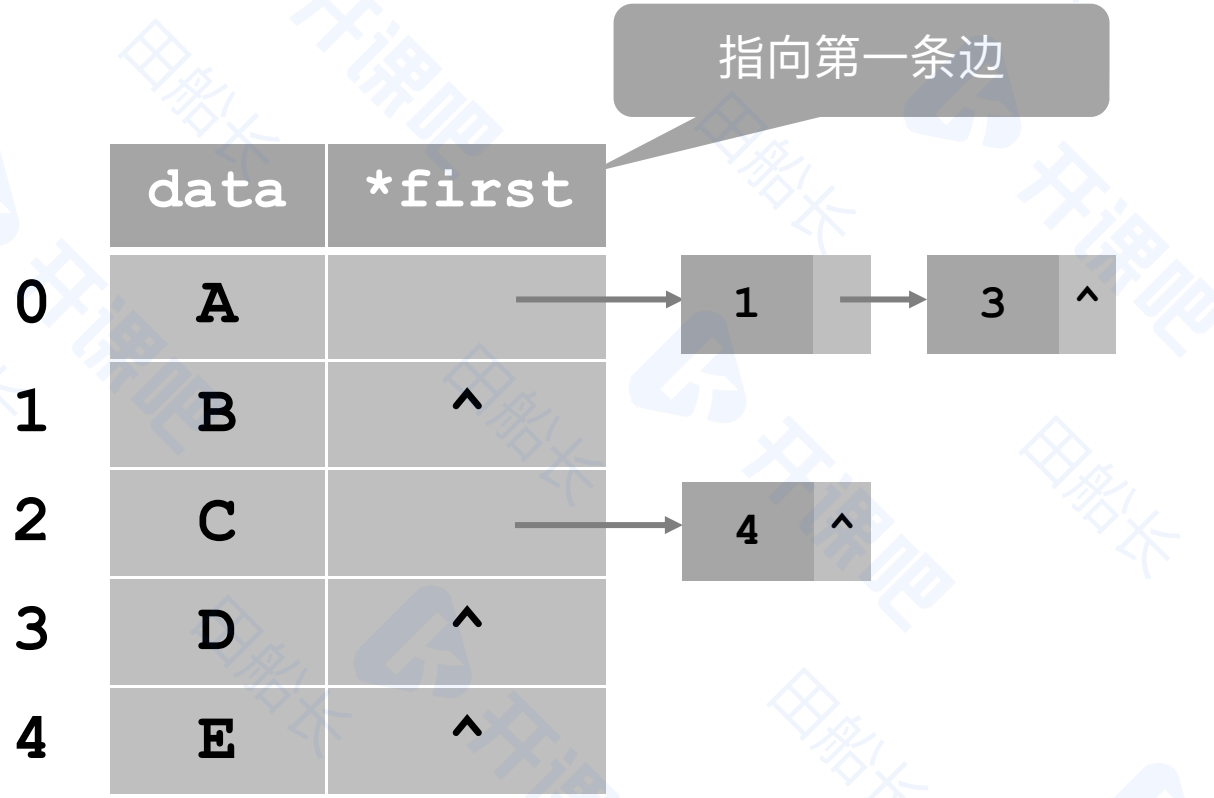
	data	*first
0	A	
1	B	^
2	C	^
3	D	^
4	E	^

邻接表（有向图）

[任务]构建邻接表（有向图）

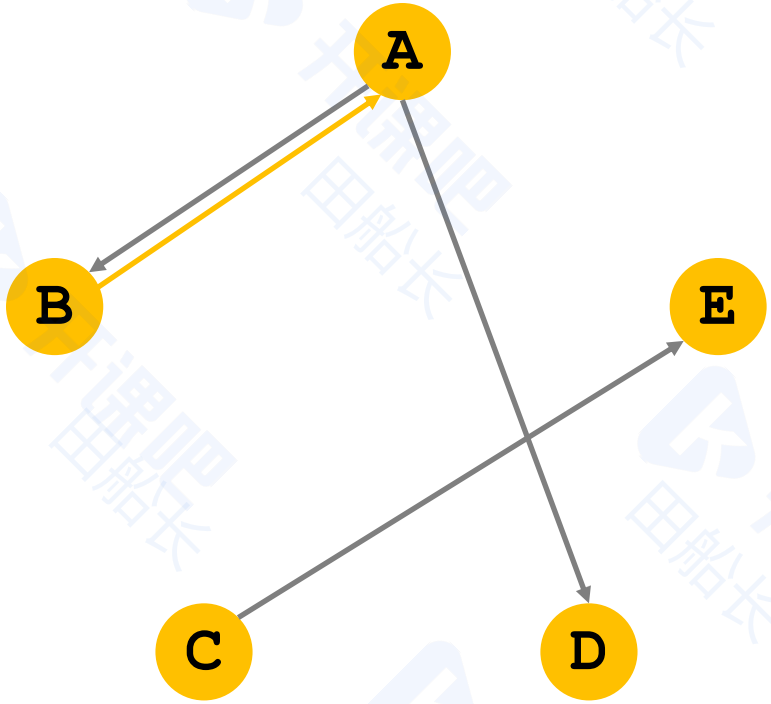


邻接表中记录有向边CE

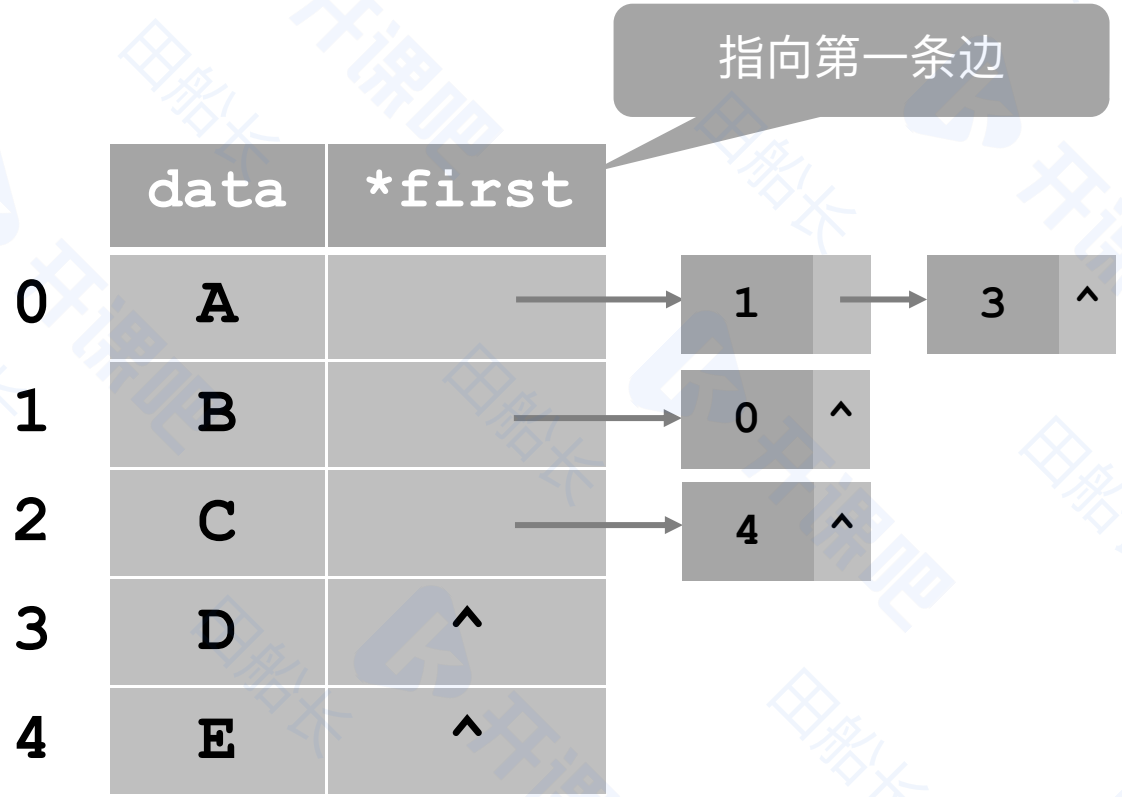


邻接表（有向图）

[任务]构建邻接表（有向图）

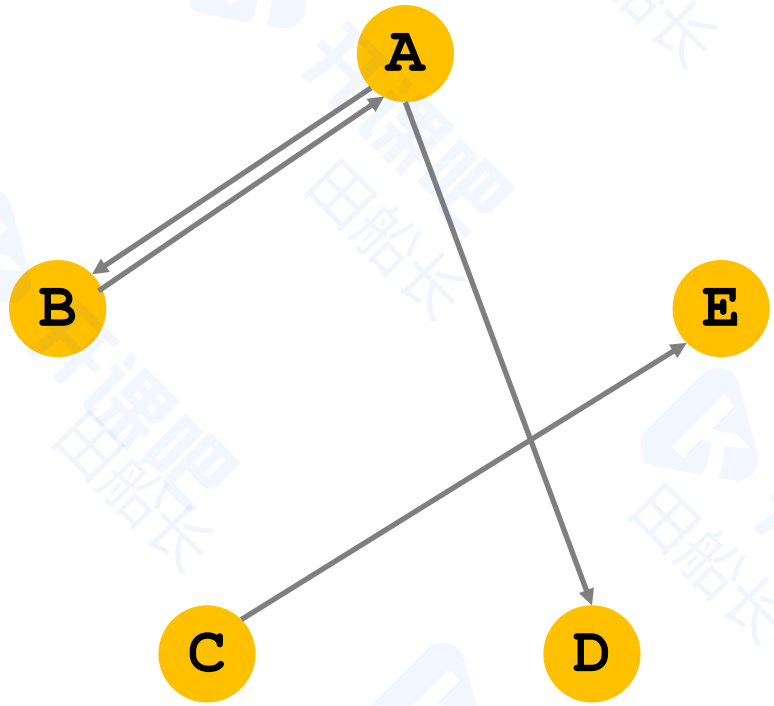


邻接表中记录有向边BA

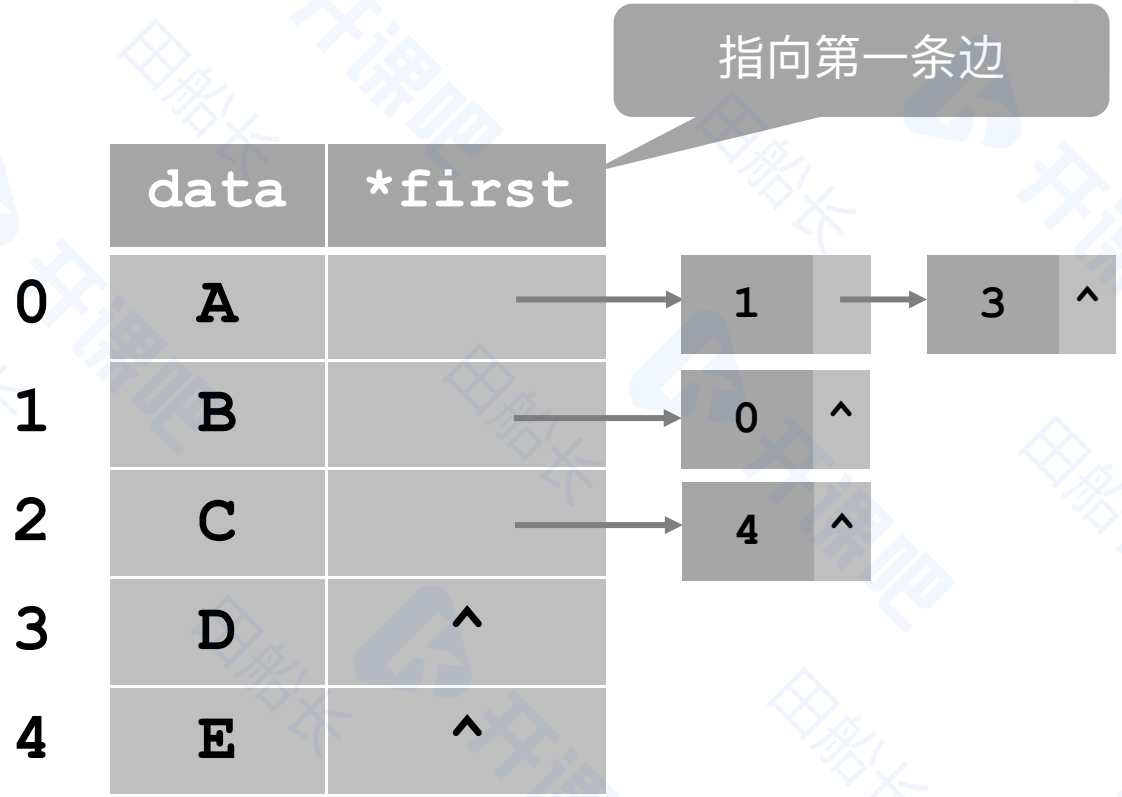


邻接表（有向图）

[任务]构建邻接表（有向图）



邻接表（有向图）构建完毕



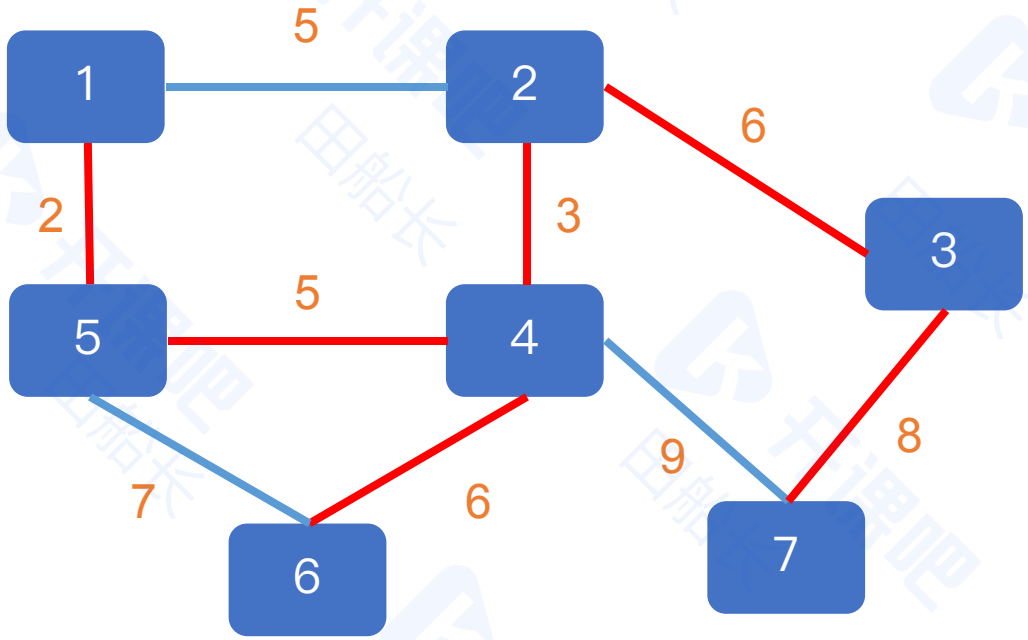
图的遍历——深度优先

从起点出发，每次走到一个新的点时
就以新的点为起点继续向下走
当走到不能走时，再进行回溯
深度优先遍历一般使用递归实现

图的遍历——广度优先

从起点出发，先遍历所有连通的点
接下来按照遍历的顺序
继续遍历每个点相连通的点
广度优先遍历一般使用队列实现

最小生成树



一个图的一种最小生成树

对于一个图来说，删掉一些边，使得每两个节点之间只有一条路径
那么，删掉边以后，可以得到一棵树，这棵树就叫做该图的生成树
所有生成树中，所有边权相加后，总权值最小的生成树，叫做最小生成树

最小生成树的性质

1. 若图中有 N 个节点，那么生成树中应该有 N-1 条边

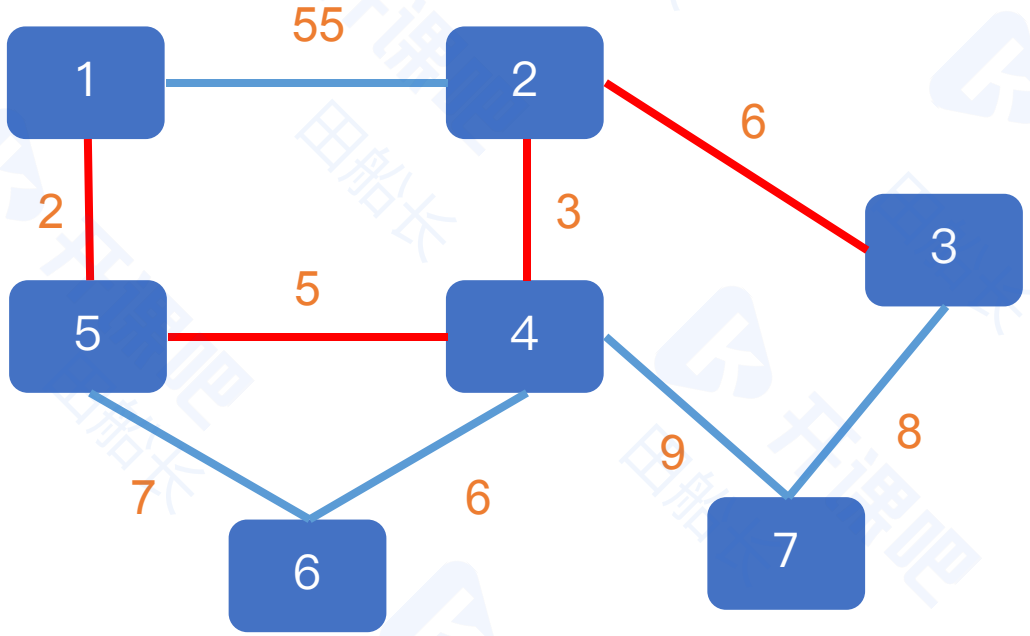
2. 最小生成树的权值总和唯一，但最小生成树不一定唯一

最小生成树的求解方法

1．Prim算法（以点为基准）

2．Kruskal算法（以边为基准）

最短路径



从1号点到3号点的最短路

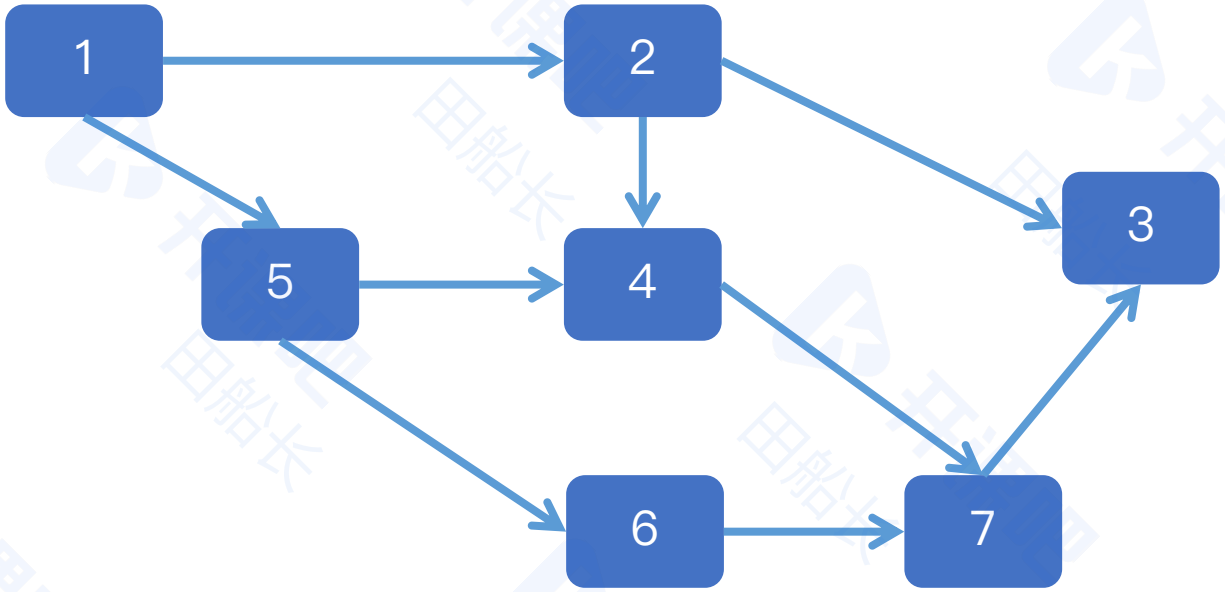
对于从节点A到节点B的所有路径
总权值最小的路径称为从A到B的最短路径，简称最短路
最短路也不一定唯一，但最短权值唯一

最短路径的求解方法

1．Dijkstra算法（单源最短路）

2．Floyd算法（多源最短路）

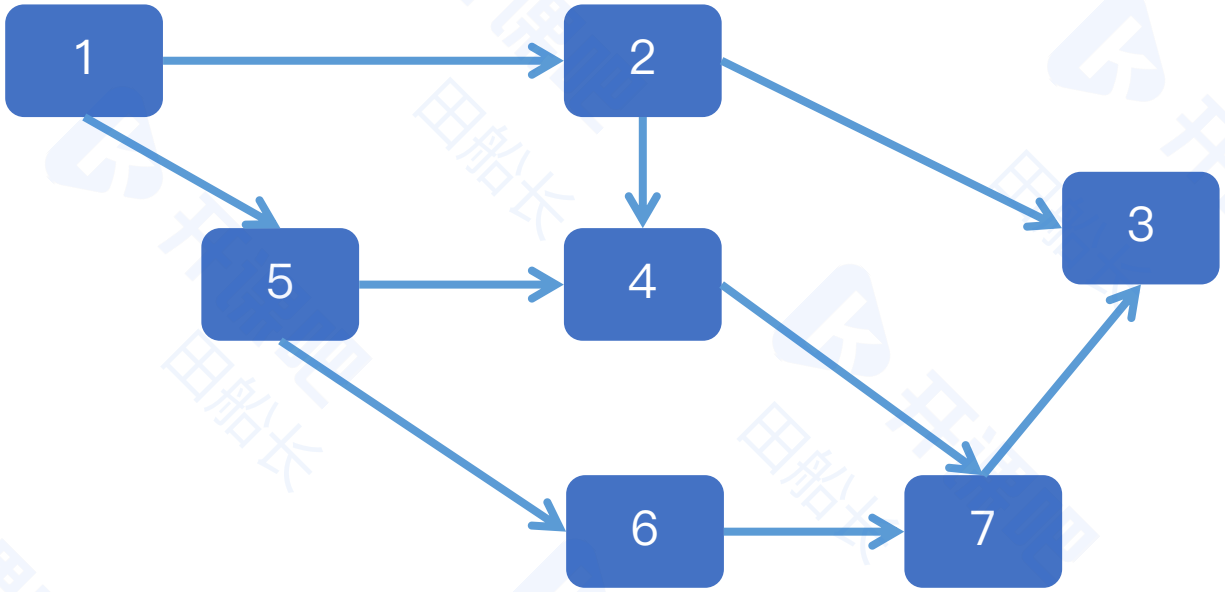
AOV网



一个AOV网

用有向图表示一个工程，其节点表示活动
有向边 1 —> 5 表示 1活动必须先于 5活动这样一种关系
则称这个图为AOV网（顶点表示活动的网络）

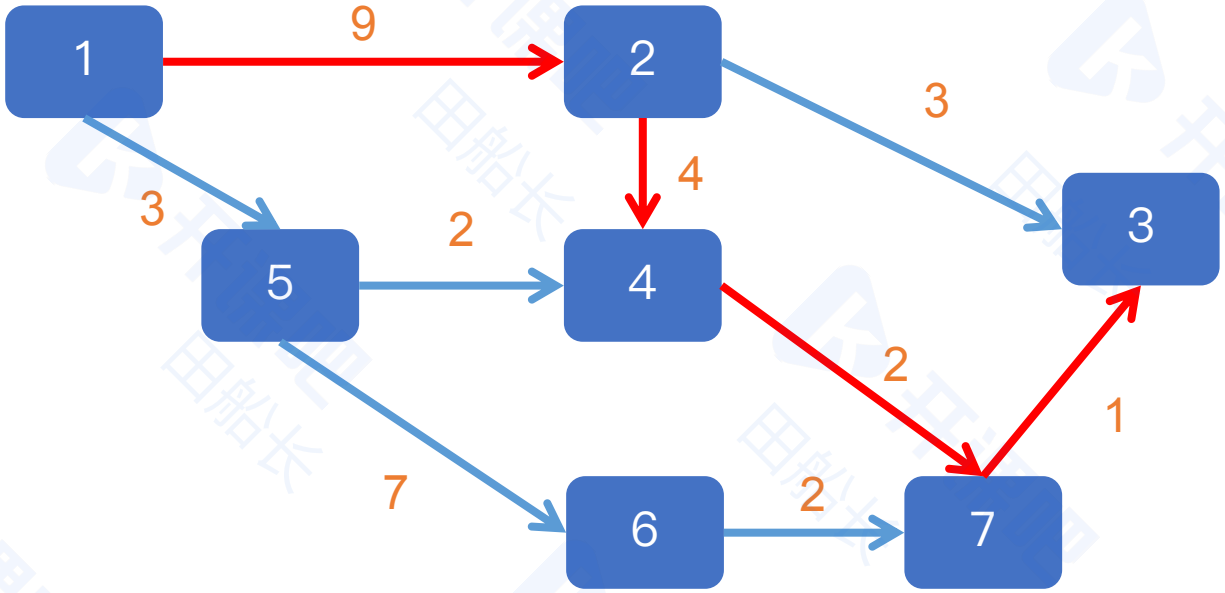
拓扑排序



1 2 5 4 6 7 3
左图的一种拓扑排序

对于AOV网来说，找到所有活动的一种排序
使得按照排序序列进行活动时，不违反网中活动先后顺序
在求解过程中通常使用入度计数来实现

AOE网与关键路径



一个AOE网及它的关键路径

用有向图表示一个工程，其边表示活动
有向边 1 —> 5 的权值3表示该活动的开销（通常为活动所需时间）
则称这个图为AOE网（边表示活动的网络）

AOE网与关键路径的性质

1.AOE网只有一个入度为0的点（源点），表示工程的开始

2.AOE网只有一个出度为0的点（汇点），表示工程的结束

3.只有在某点所代表的事情发生以后，从该点出发的活动（有向边）才能开始

4.只有在进入某点的各个活动都以结束时，该点的事件才能发生

5.从源点到汇点的最长路径称为关键路径，完成整个工程的最短时间就是关键路径的长度

6.所有在关键路径上的活动（边）称为关键活动，缩短关键活动可以缩短工期