



Part3：贪心算法

- 贪心策略
- 最小生成树—— Prim算法
- 最小生成树—— Kruskal算法
- 哈夫曼树及编码
- 背包问题



9.1 贪心策略——几个例子

找零问题

问题描述

假设有面额为25, 10, 5, 1美分的硬币，用这些硬币给出48美分的找零。

解决办法

不超过总额的情况下优先选择面额最大的， $25 * 1 + 10 * 2 + 3 * 1 = 48$ 。

9.1 贪心策略——几个例子

找零问题

注意

- 对于上面的实例算法给出了最优解；
- 考虑面额为25, 10, 1美分的硬币，找零30每分；
- 算法给出解： $25 * 1 + 5 * 1 = 30$ ；
- 实际最优解： $10 * 3 = 30$ ；



9.1 贪心策略——几个例子

排课问题

问题描述

假设有如下课表，你希望尽可能多的课程安排在某间教室上。

课程	开始时间	结束时间
美术	9:00	10:00
英语	9:30	10:30
数学	10:00	11:00
计算机	10:30	11:30
音乐	11:00	12:00



9.1 贪心策略——几个例子

排课问题

解决办法

在时间不冲突的情况下，每一次都选择最早结束的课程。

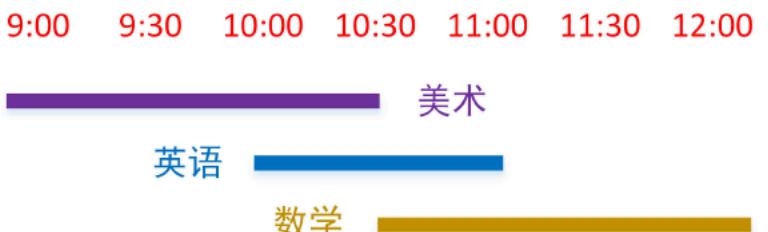
美术	9:00	10:00
数学	10:00	11:00
音乐	11:00	12:00



9.1 贪心策略——几个例子

- 贪心算法每步都采取当前最优（局部最优），最终得到一个全局最优的方案；
- 选择最早结束的课程的策略可以找到教室调度问题最优解；
- 选择最小占用时间的课程得不到最优解；

美术	9:00	10:30
英语	10:00	11:00
数学	10:30	12:00



9.1 贪心策略——几个例子

- 选择最早开始的课程得不到最优解；

美术	9:00	11:00
英语	9:30	10:30
数学	10:30	12:00



- 贪心算法不是任何情况下都有效，但是容易实现。

9.1 贪心策略——几个例子

活动安排问题

- 设有n个活动的集合 $E=\{1,2,\dots,n\}$ ，其中每个活动都要求使用同一资源，而在同一时间内只有一个活动能使用这一资源。
- 活动安排问题就是要在所给的活动集合中选出最大(尽可能多)的相容活动子集合
- 每个活动*i*都有一个要求使用该资源的起始时间 s_i 和一个结束时间 f_i ,且 $s_i < f_i$,若区间 $[s_i, f_i)$ 与区间 $[s_j, f_j)$ 不相交, 即当 $s_i \geq f_j$ 或 $s_j \geq f_i$ 时, 则称活动*i*与活动*j*相容

活动安排问题

```
template<class Type>
void GreedySelector(int n,Type s[],Type f[],bool A[])
{
    A[1]=true;
    int j=1;
    for (int i=2;i<=n;i++) {
        if (s[i]>=f[j]) { A[i]=true; j=i; }
        else A[i]=false;
    }
}
```

按结束时间的非减序排列

当输入的活动已排列，算法只需
 $O(n)$ 的时间实现活动安排问题；
否则用 $O(n\log n)$ 的时间重排。

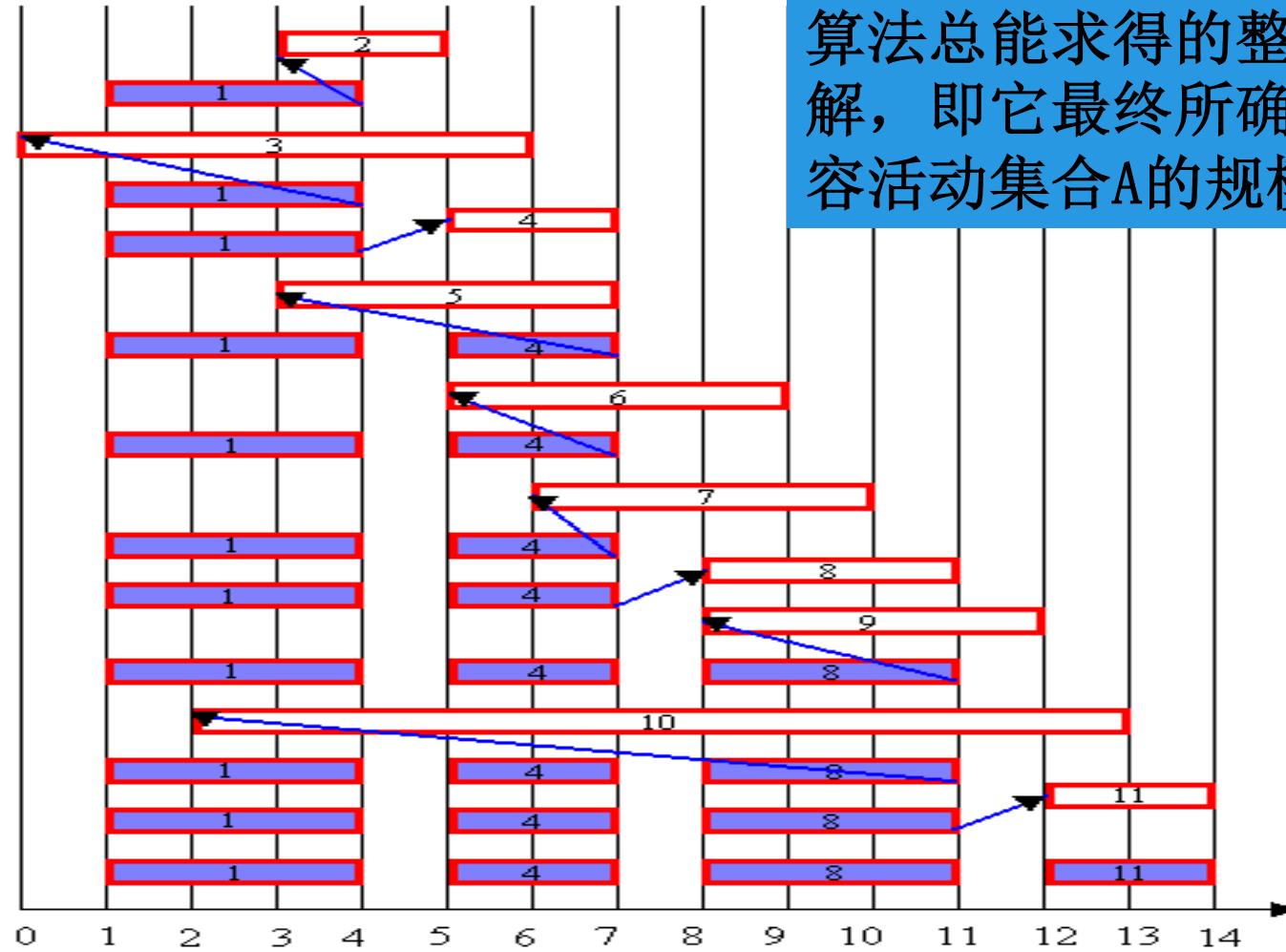
该算法每次选择具有最早完成时间的相容活动加入集合A中。该算法的贪心选择的意义是使剩余的可安排时间段极大化，以便安排尽可能多的相容活动。

活动安排问题

设待安排的11个活动的开始时间和结束时间按结束时间的非减序排列如下

i	1	2	3	4	5	6	7	8	9	10	11
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

活动安排问题例子



对于活动安排问题，贪心算法总能求得的整体最优解，即它最终所确定的相容活动集合A的规模最大。

贪心算法的特点

贪心算法特点

- 可行性：贪心算法每一步选择都必须满足问题的约束；
- 局部最优：算法不考虑全局最优，仅做出当前看来最优的选择；
- 不可取消：一旦做出选择，再算法后面的步骤中无法改变；

证明贪婪算法可以获得最优解的方法（本课程不涉及）

- 数学归纳法证明算法每一步获得的部分解能够扩展到全局最优解；
- 证明算法在接近目标的过程中，每一步的选择不会比其他算法差；
- 基于算法的输出，证明算法得到的解的最优化；

贪心算法的基本要素1——贪心选择性质

贪心选择性质

- 求问题的**整体最优解**可以通过一系列**局部最优**的选择，即贪心选择来达到，即一个最优解以贪心选择开始。
- 贪心算法则通常以**自顶向下**的方式进行，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题简化为规模更小的子问题。

贪心算法的基本要素2——最优子结构性质

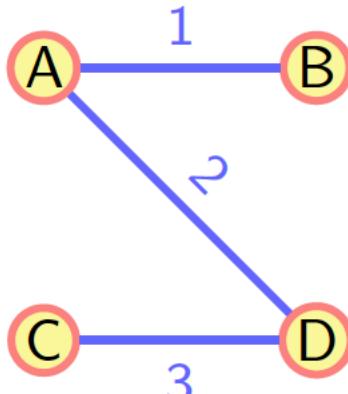
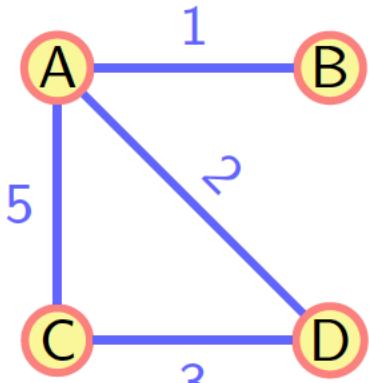
最优子结构性质

- 当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。
- 问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

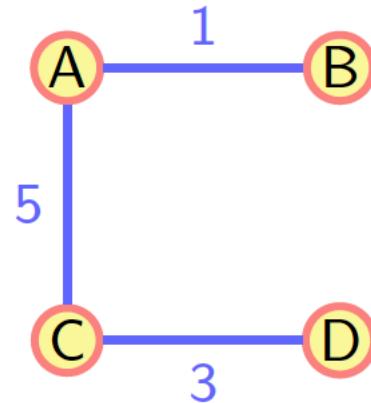
最小生成树

生成树

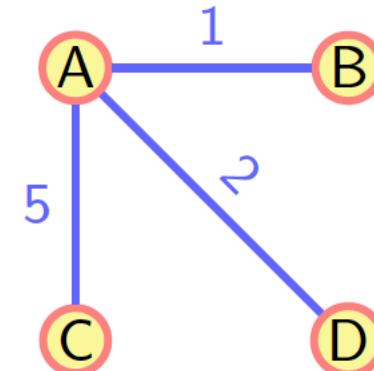
连通图的一棵生成树是包含图的所有顶点的连通无环子图（一棵树）。



$$W(T_1) = 6$$



$$W(T_2) = 9$$



$$W(T_3) = 8$$

最小生成树

加权连通图的一棵最小生成树是图的一棵权重最小 (T_1) 的生成树。

最小生成树——Prim算法

图的定义回顾

图 (Graph) 是有一些称为顶点的点 (V) 构成的集合，其中某些顶点由一些称为边的线段 (E) 相连， $G = \langle V, E \rangle$ 。

In an *undirected graph* $G = (V, E)$, the edge set E consists of *unordered* pairs of vertices.

In either case, we have $|E| = O(V^2)$. Moreover, if G is connected, then $|E| \geq |V| - 1$, which implies that $\lg |E| = \Theta(\lg V)$.

最小生成树——Prim算法

Prim算法流程

- ① 任意选一点 v_0 , 集合 V 被分割成两个集合 $V_T = \{v_0\}$ 和 $V - V_T$;
- ② 从连通 V_T 和 $V - V_T$ 的边中挑选一条权重最小的边 $e^* = (v^*, u^*)$,
 $v^* \in V_T, u^* \in V - V_T$, 将 u^* 加入 V_T 中, 从 $V - V_T$ 删除 u^* ;
- ③ 重复步骤2, 直到集合 $V - V_T$ 为空;

为什么Prim算法是贪心算法?

以贪婪的方式生成树, 每次选择不在当前树中最短的边。

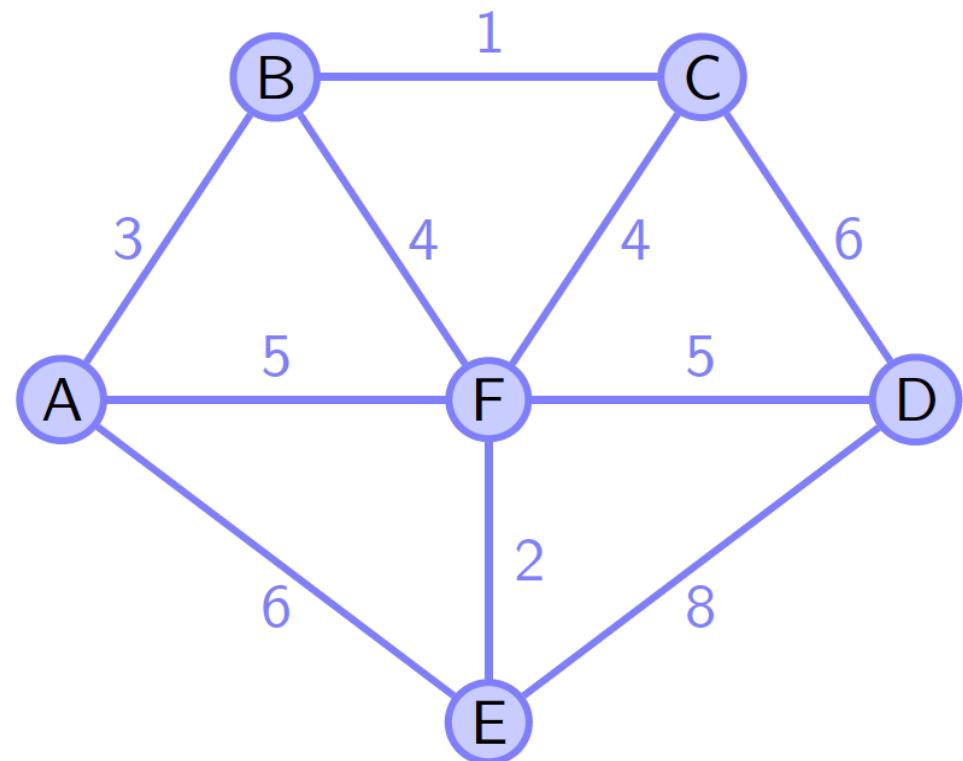
最小生成树——Prim算法实例

算法核心

从连通 V_T 和 $V - V_T$ 的边中挑选一条权重最小的边。

计算过程

- $V_T = \{\emptyset\}$
- $V - V_T = \{A, B, C, D, E, F\}$



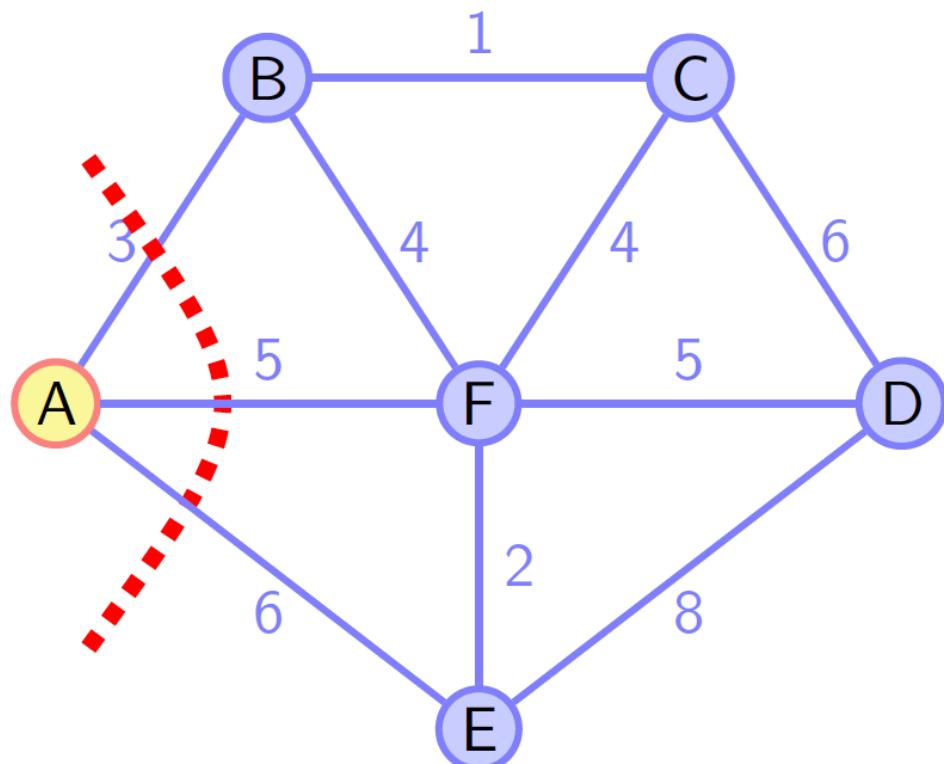
最小生成树——Prim算法实例

算法核心

从连通 V_T 和 $V - V_T$ 的边中挑选一条权重最小的边。

计算过程

- $V_T = \{A\}$
- $V - V_T = \{B, C, D, E, F\}$
- $(A, B) = 3$
- $(A, F) = 5$
- $(A, E) = 6$



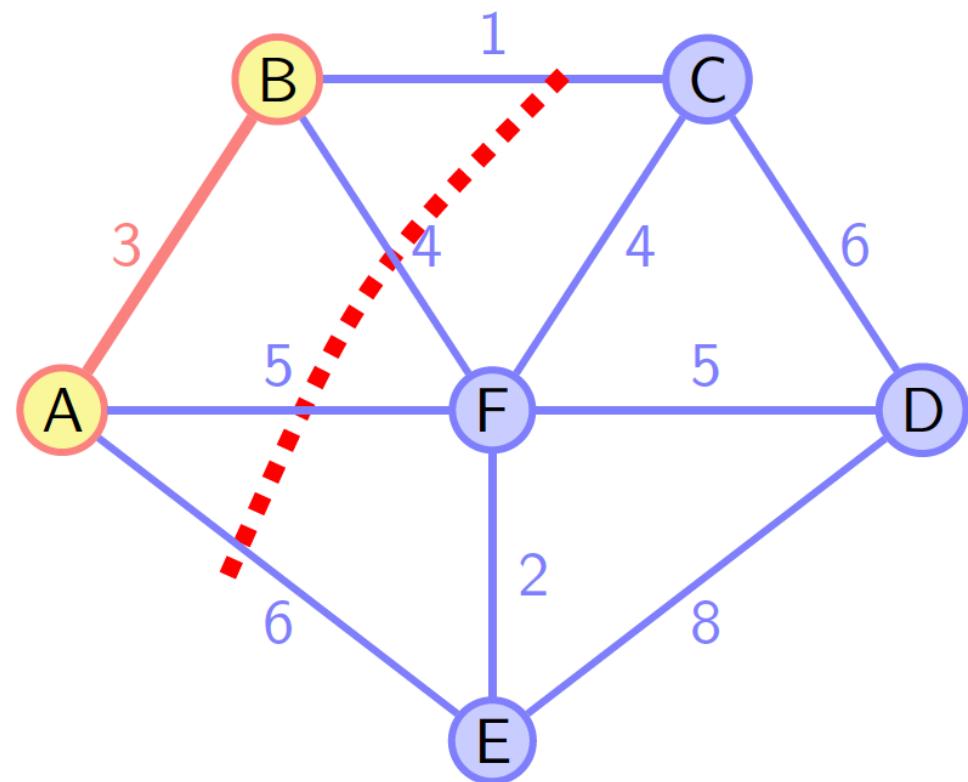
最小生成树——Prim算法实例

算法核心

从连通 V_T 和 $V - V_T$ 的边中挑选一条权重最小的边。

计算过程

- $V_T = \{A, B\}$
- $V - V_T = \{C, D, E, F\}$
- $(B, C) = 1$
- $(B, F) = 4$
- $(A, F) = 5$
- $(A, E) = 6$



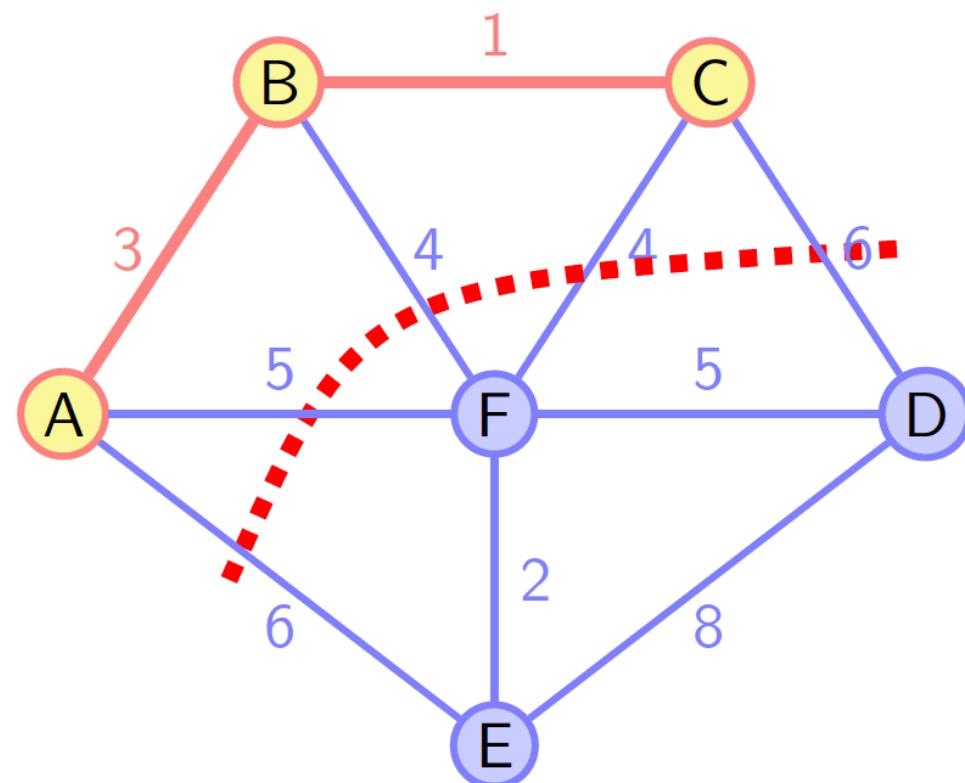
最小生成树——Prim算法实例

算法核心

从连通 V_T 和 $V - V_T$ 的边中挑选一条权重最小的边。

计算过程

- $V_T = \{A, B, C\}$
- $V - V_T = \{D, E, F\}$
- $(C, F) = 4$
- $(C, D) = 6$
- $(B, F) = 4$
- $(A, F) = 5$
- $(A, E) = 6$



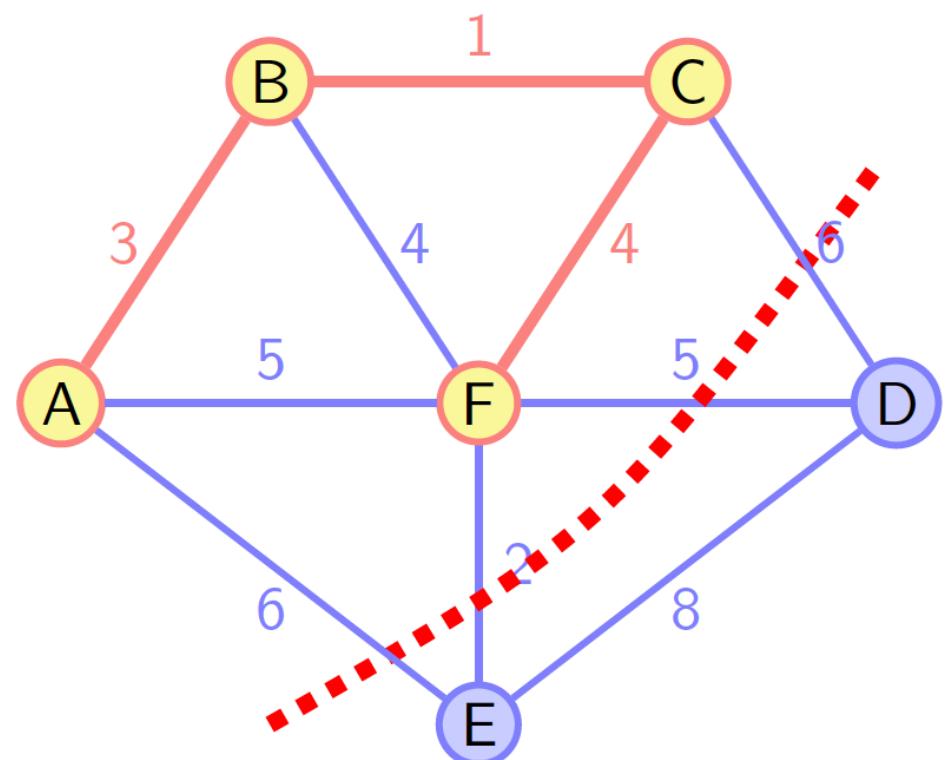
最小生成树——Prim算法实例

算法核心

从连通 V_T 和 $V - V_T$ 的边中挑选一条权重最小的边。

计算过程

- $V_T = \{A, B, C, F\}$
- $V - V_T = \{D, E\}$
- $(C, D) = 6$
- $(F, D) = 5$
- $(F, E) = 2$
- $(A, E) = 6$



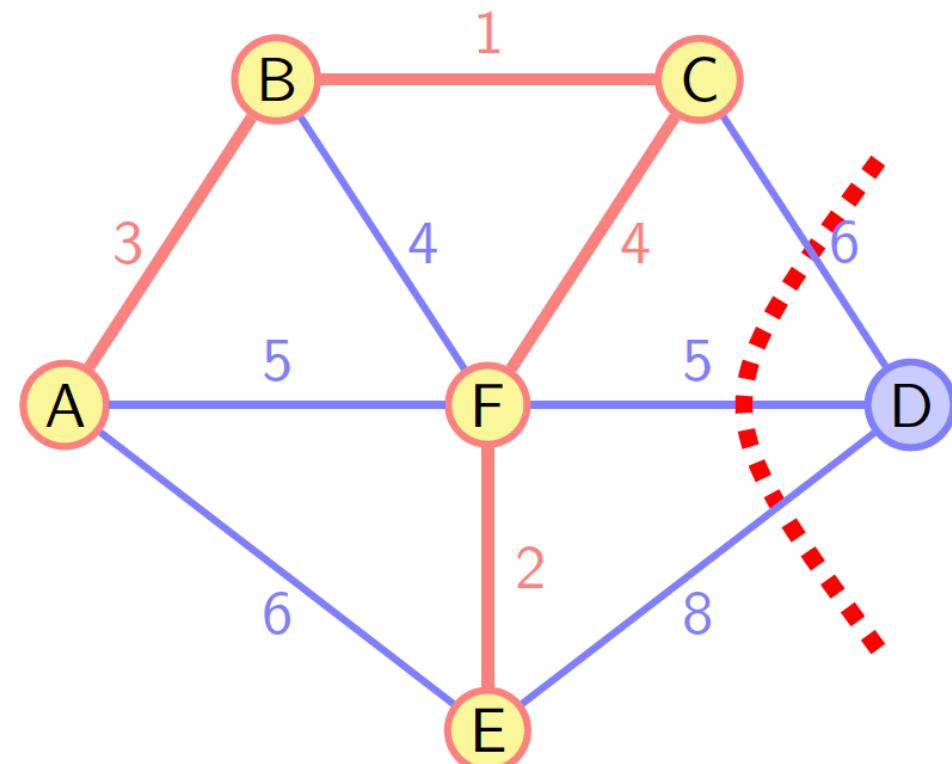
最小生成树——Prim算法实例

算法核心

从连通 V_T 和 $V - V_T$ 的边中挑选一条权重最小的边。

计算过程

- $V_T = \{A, B, C, F, E\}$
- $V - V_T = \{D\}$
- $(C, D) = 6$
- $(F, D) = 5$
- $(E, D) = 8$



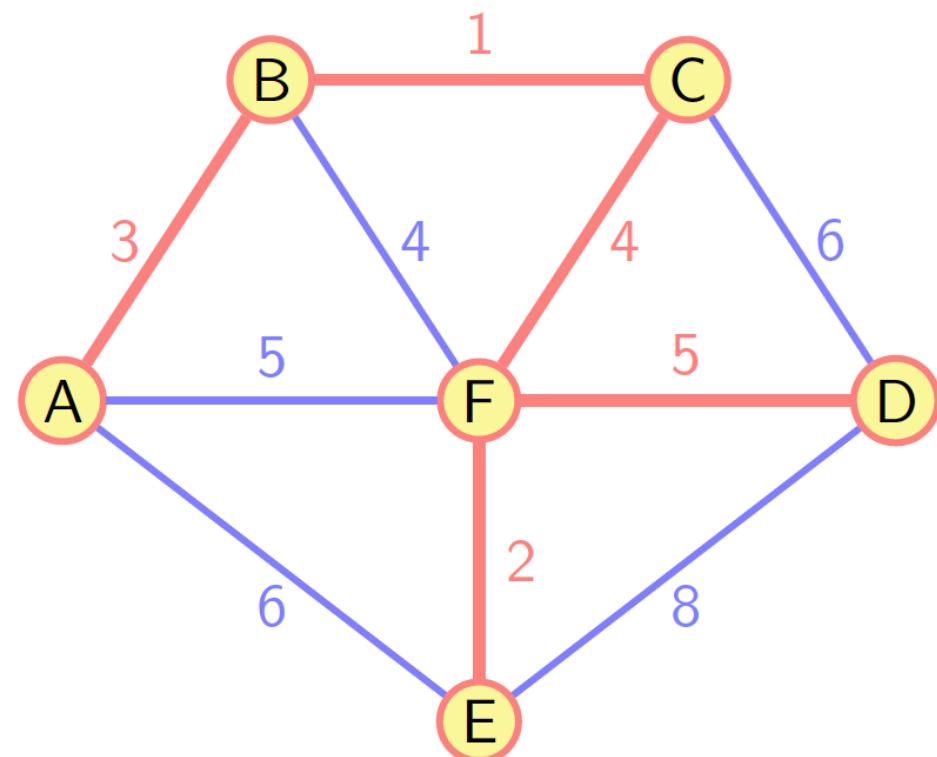
最小生成树——Prim算法实例

算法核心

从连通 V_T 和 $V - V_T$ 的边中挑选一条权重最小的边。

计算过程

- $V_T = \{A, B, C, F, E, D\}$
- $V - V_T = \{\emptyset\}$
- $W(T) = 15$

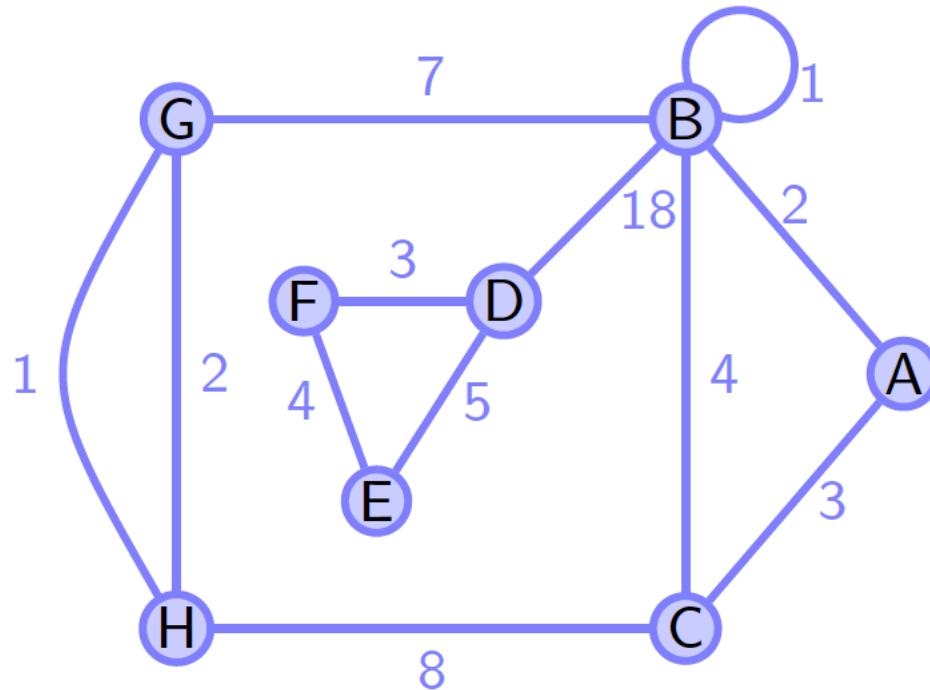


最小生成树——Prim算法实例

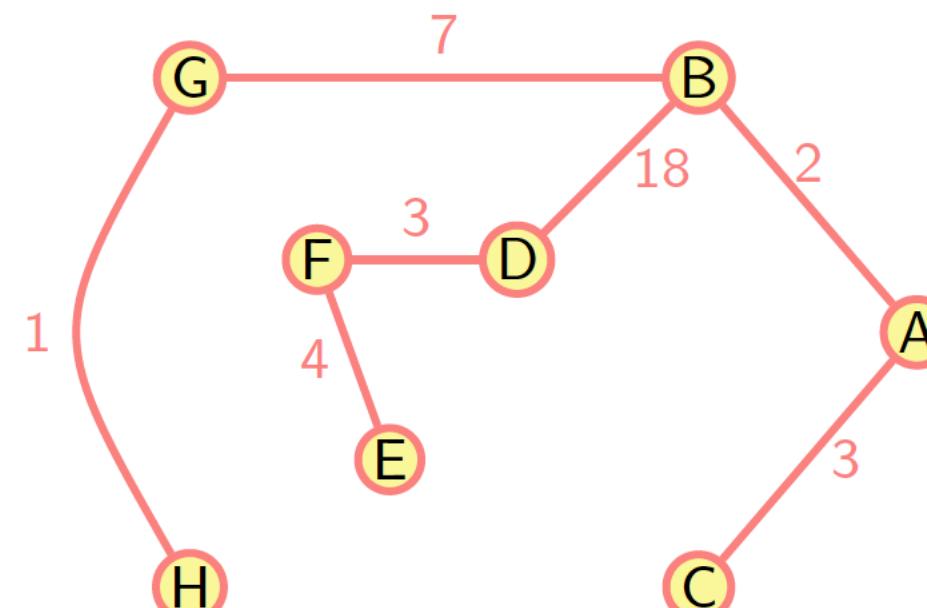
算法核心

从连通 V_T 和 $V - V_T$ 的边中挑选一条权重最小的边。

图



最小生成树



最小生成树——Prim算法伪代码

构造最小生成树的Prim算法

```
1: function Prim( $G$ )
2:    $V_T \leftarrow v_0$  // $v_0$ 为 $G$ 中任一顶点
3:    $E_T \leftarrow \emptyset$ 
4:   for  $i \leftarrow 1 \rightarrow |V| - 1$  do
5:     for ( $v \in V_T$ ) and ( $u \in V - V_T$ ) do
6:       求权重最小的边  $e^* = (v^*, u^*)$ ;
7:     end for
8:      $V_T \leftarrow V_T \cup \{u^*\}$ 
9:      $E_T \leftarrow E_T \cup \{e^*\}$ 
10:    end for
11:    return  $E_T$ 
12: end function
```

最小生成树——Prim算法效率分析

计算过程中用到的数组

邻接矩阵

	A	B	C	D	E	F
A	∞	3	∞	∞	6	5
B	3	∞	1	∞	∞	4
C	∞	1	∞	6	∞	4
D	∞	∞	6	∞	8	5
E	6	∞	∞	8	∞	2
F	3	4	4	5	2	∞

V_T 和 $V - V_T$ 连接权重 $Near$

序号	V_T	A	B	C	D	E	F
0	A	∞	3	∞	∞	6	5
1	A,B	∞	3	∞	∞	6	5
2	A,B	∞	∞	1	∞	6	4
3	A,B,C	∞	∞	1	∞	6	4
4	A,B,C	∞	∞	∞	6	6	4
5							...

复杂度分析

- 步骤1和3，从 $Near$ 中选出最小的权值，复杂度为 $O(n)$ ；
- 步骤2和4，根据 V_T 中节点，更新 $Near$ 的权值，复杂度为 $O(n)$ ；
- 一共需要向 V_T 中加入 $n - 1$ 个节点，总的复杂度为 $O(n^2)$ ；

最小生成树——Kruskal算法

算法思想

算法贪婪地选择最小权重的边，扩展无环的子图构造最小生成树。

算法流程

- ① 按照权重非递减顺序对图中的边 E 进行排序；
- ② 扫描以排序的列表，如果下一条边加入到当前的子图中不导致一个回路，则加入该边到子图中，否则跳过该边；
- ③ 重复步骤2，直到子图中有 $|V| - 1$ 条边；

最小生成树——Kruskal算法

算法的另一种解释

Kruskal算法可看作是对包含给定图所有顶点和某些边的一系列森林所做的连续动作。初始森林是 $|V|$ 颗只包含一个顶点的树，通过加入边，将小数连通成大树，最终构造出一棵最小生成树。

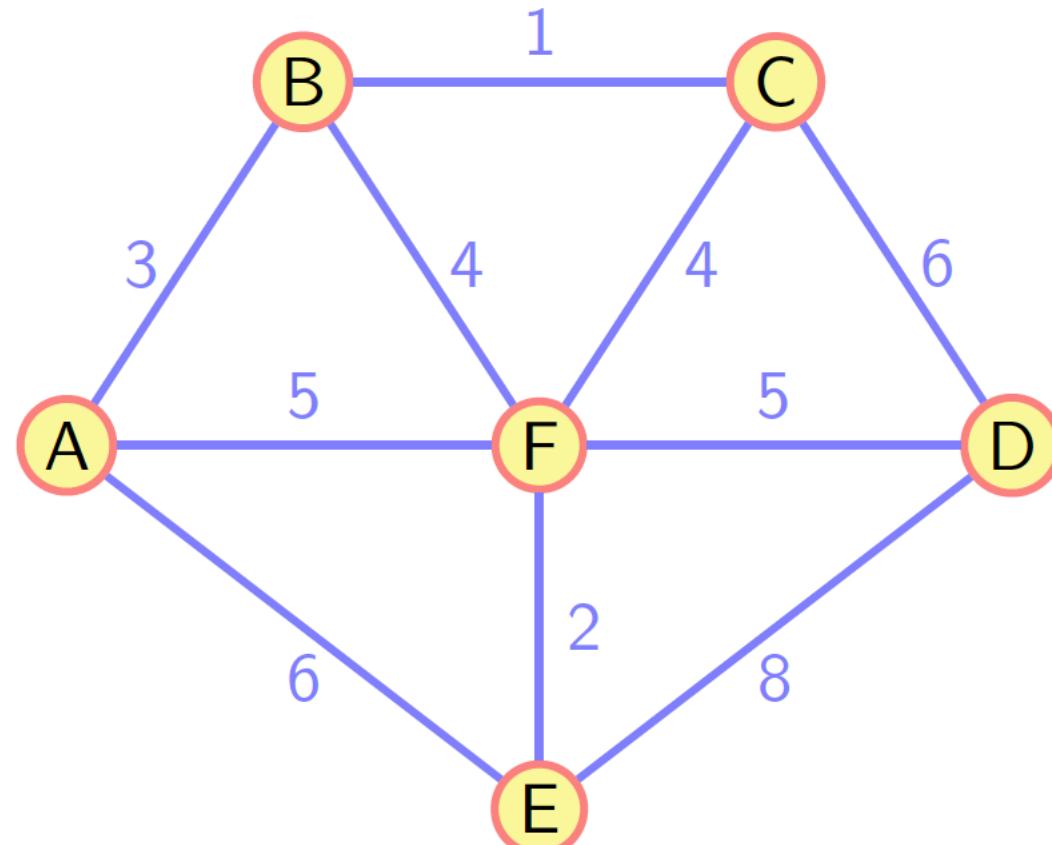
算法核心

算法贪婪地选择最小权重的边构造生成树，所选择的边不能构成回路。

最小生成树——Kruskal算法实例

计算过程

- ① $(B, C) = 1$
- ② $(F, E) = 2$
- ③ $(A, B) = 3$
- ④ $(B, F) = 4$
- ⑤ $(C, F) = 4$
- ⑥ $(A, F) = 5$
- ⑦ $(D, F) = 5$
- ⑧ $(A, E) = 6$
- ⑨ $(C, D) = 6$
- ⑩ $(D, E) = 8$



最小生成树——Kruskal算法实例

计算过程

$$① (B, C) = 1$$

$$② (F, E) = 2$$

$$③ (A, B) = 3$$

$$④ (B, F) = 4$$

$$⑤ (C, F) = 4$$

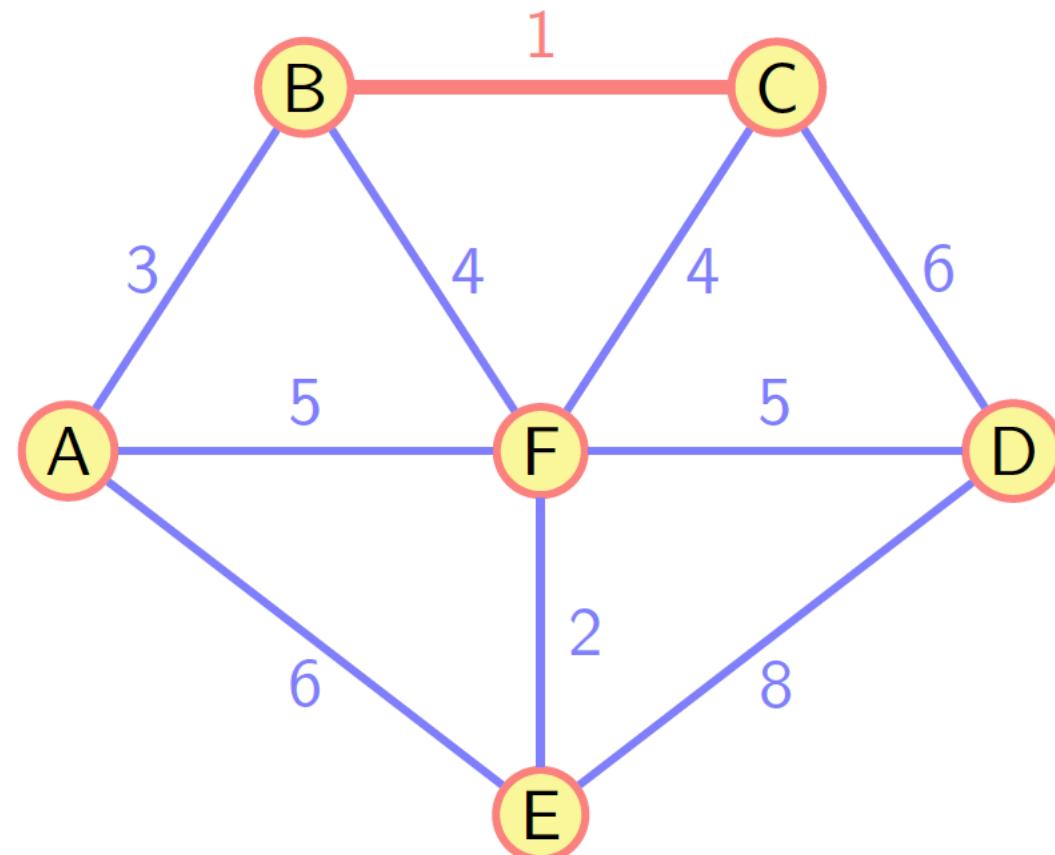
$$⑥ (A, F) = 5$$

$$⑦ (D, F) = 5$$

$$⑧ (A, E) = 6$$

$$⑨ (C, D) = 6$$

$$⑩ (D, E) = 8$$



最小生成树——Kruskal算法实例

计算过程

$$① (B, C) = 1$$

$$② (F, E) = 2$$

$$③ (A, B) = 3$$

$$④ (B, F) = 4$$

$$⑤ (C, F) = 4$$

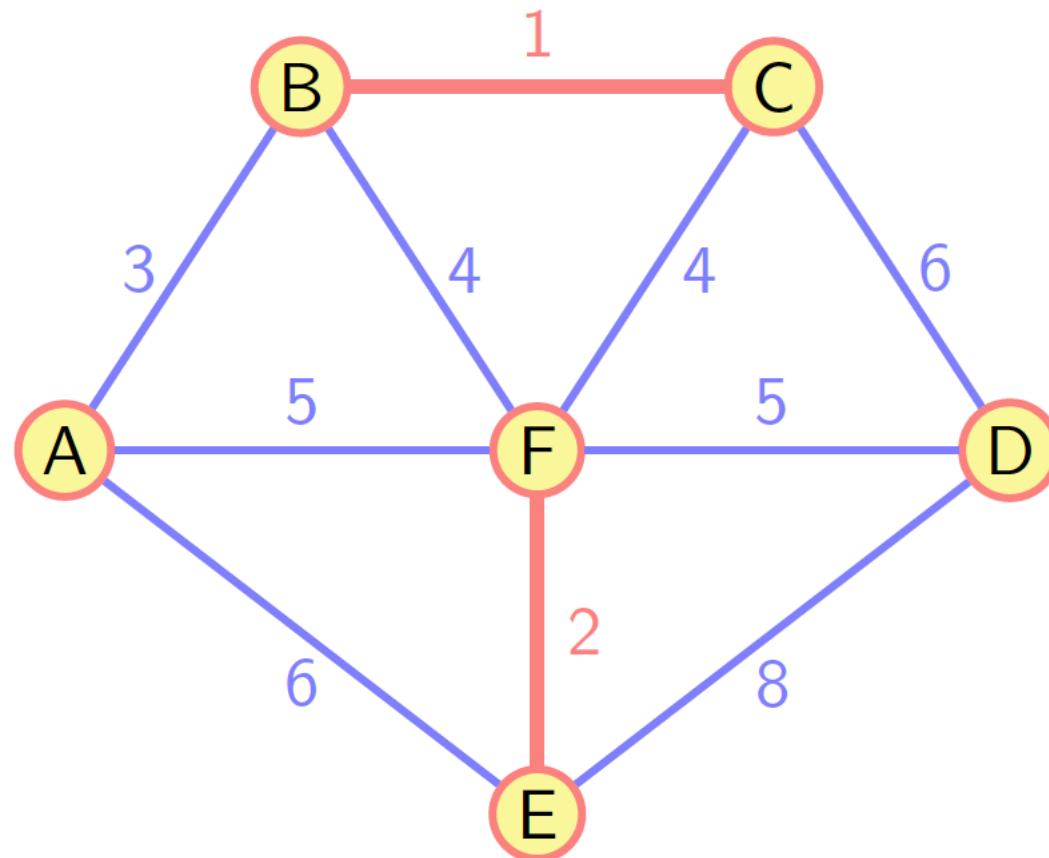
$$⑥ (A, F) = 5$$

$$⑦ (D, F) = 5$$

$$⑧ (A, E) = 6$$

$$⑨ (C, D) = 6$$

$$⑩ (D, E) = 8$$



最小生成树——Kruskal算法实例

计算过程

$$① (B, C) = 1$$

$$② (F, E) = 2$$

$$③ (A, B) = 3$$

$$④ (B, F) = 4$$

$$⑤ (C, F) = 4$$

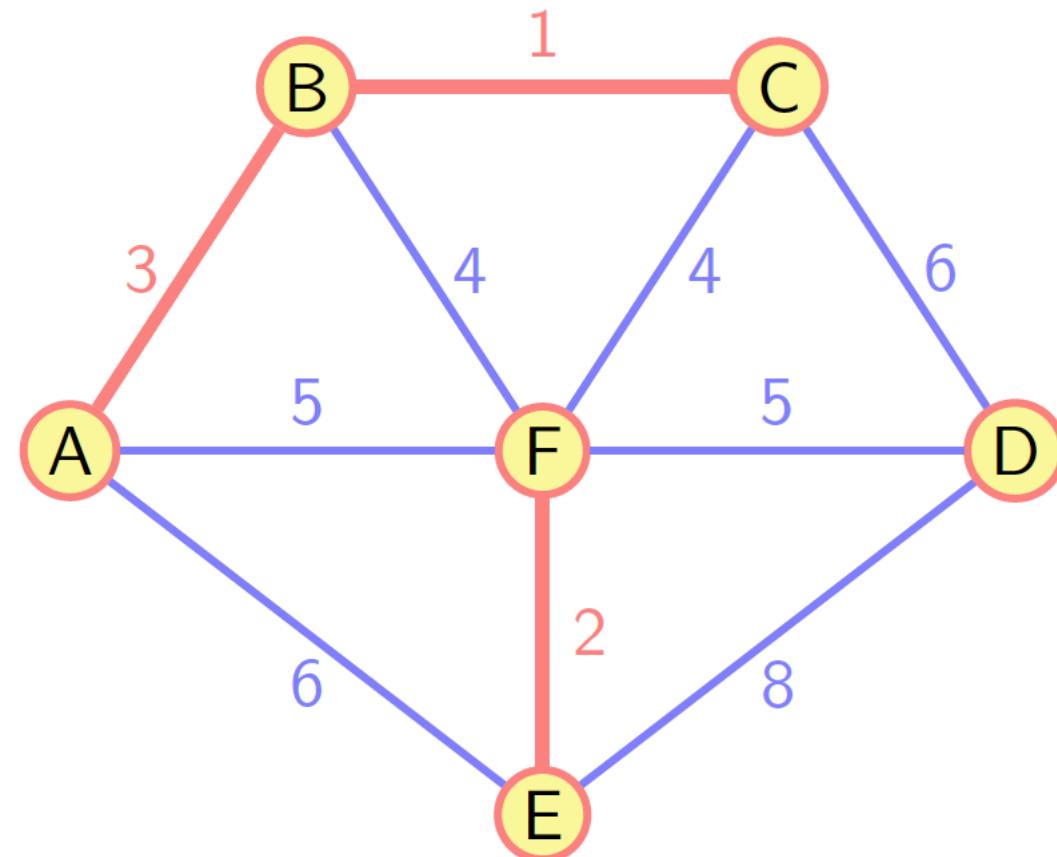
$$⑥ (A, F) = 5$$

$$⑦ (D, F) = 5$$

$$⑧ (A, E) = 6$$

$$⑨ (C, D) = 6$$

$$⑩ (D, E) = 8$$

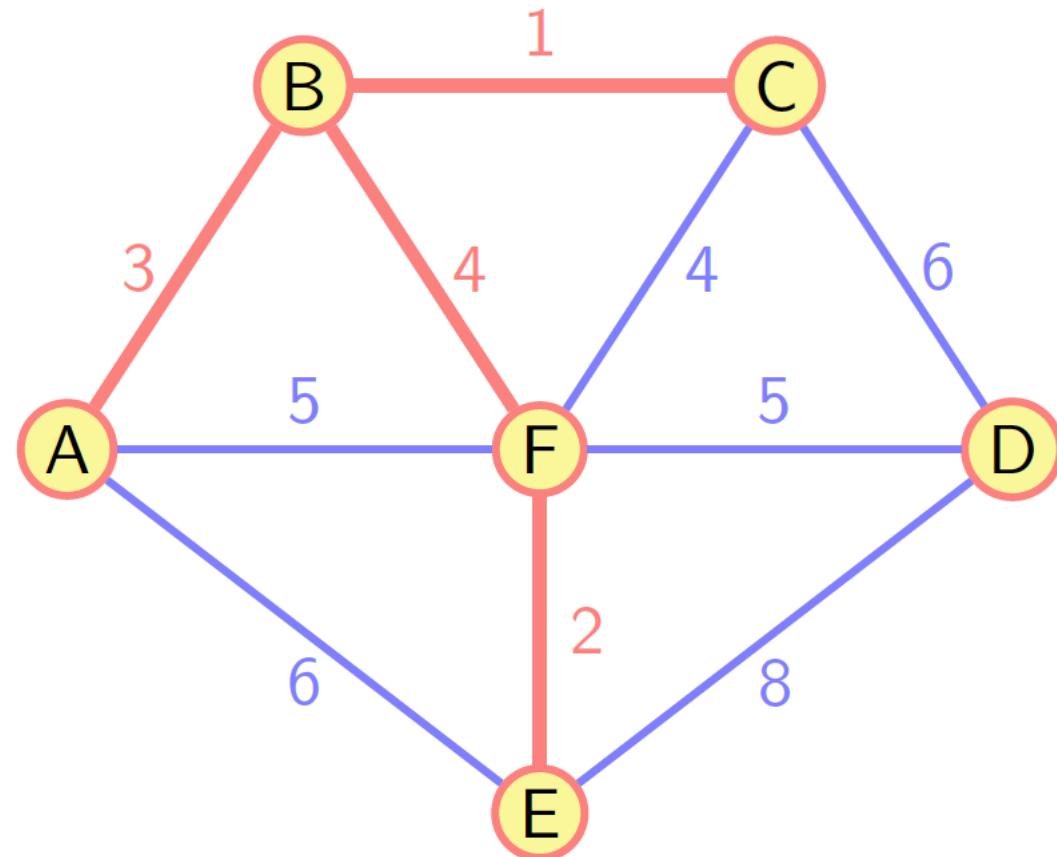


最小生成树——Kruskal算法实例

计算过程

- ① $(B, C) = 1$
- ② $(F, E) = 2$
- ③ $(A, B) = 3$
- ④ $(B, F) = 4$
- ⑤ $(C, F) = 4$

- ⑥ $(A, F) = 5$
- ⑦ $(D, F) = 5$
- ⑧ $(A, E) = 6$
- ⑨ $(C, D) = 6$
- ⑩ $(D, E) = 8$



最小生成树——Kruskal算法实例

计算过程

$$① (B, C) = 1$$

$$② (F, E) = 2$$

$$③ (A, B) = 3$$

$$④ (B, F) = 4$$

$$⑤ (C, F) = 4$$

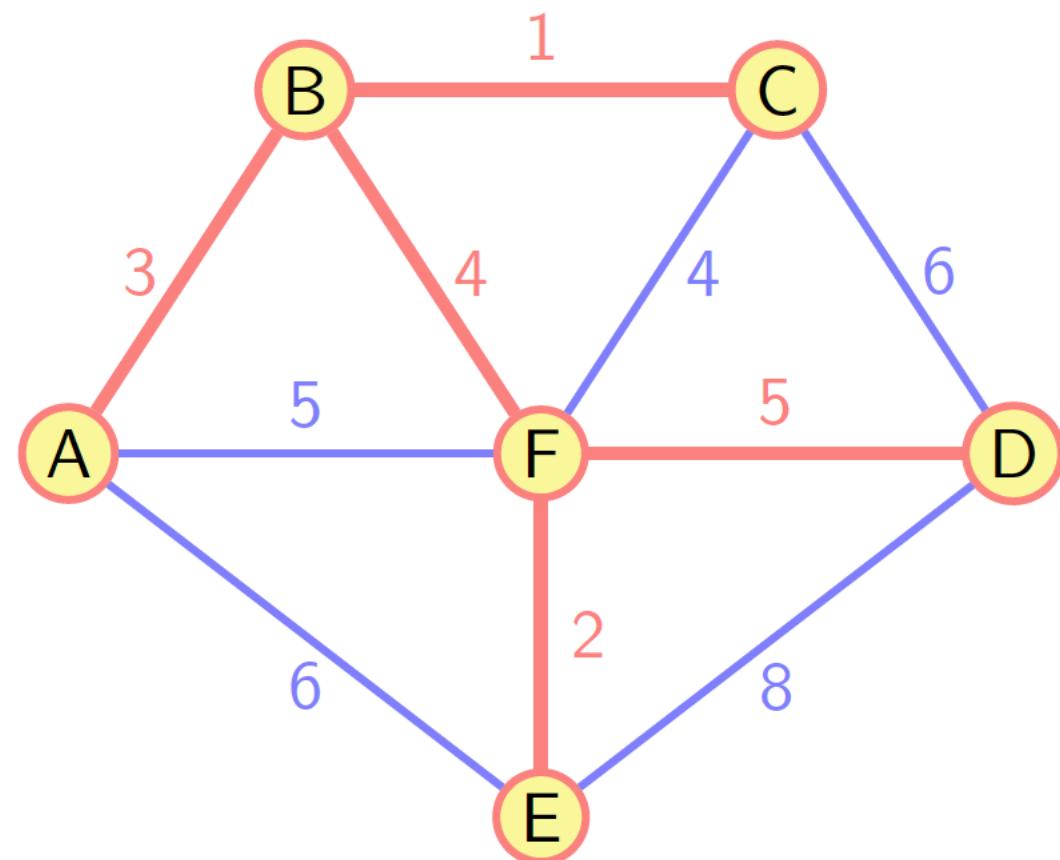
$$⑥ (A, F) = 5$$

$$⑦ (D, F) = 5$$

$$⑧ (A, E) = 6$$

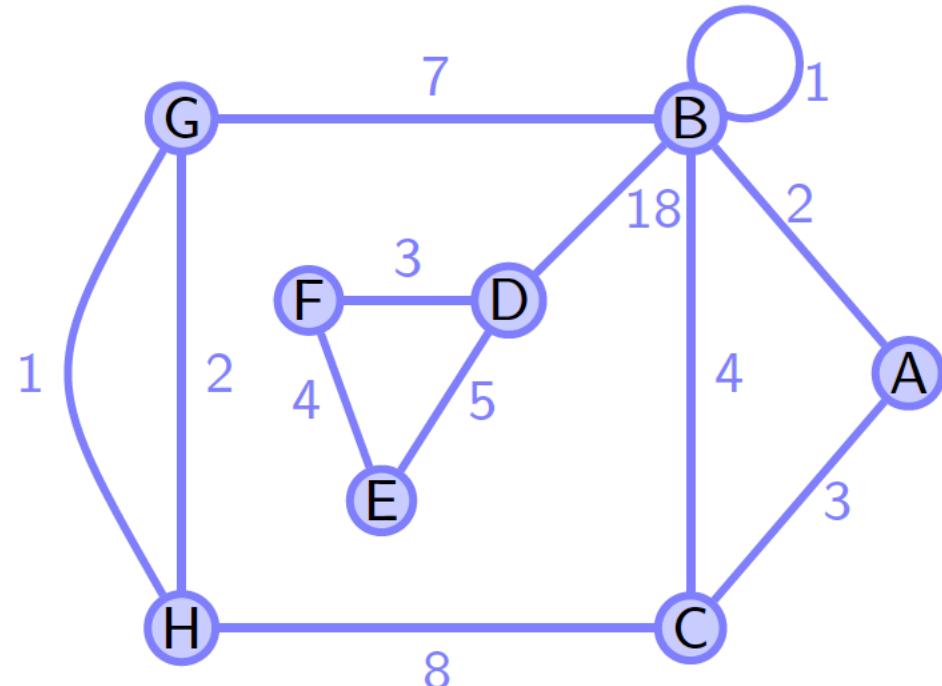
$$⑨ (C, D) = 6$$

$$⑩ (D, E) = 8$$

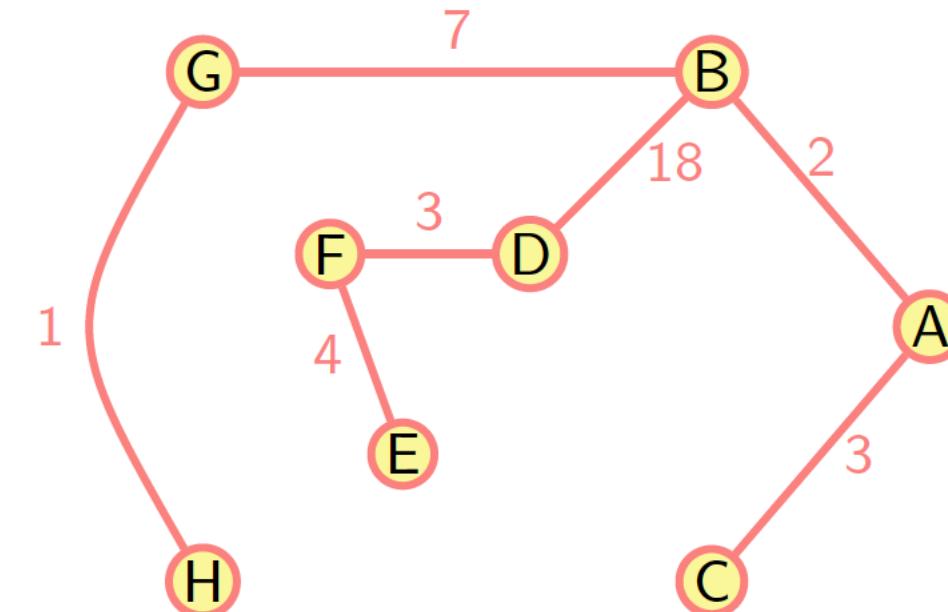


最小生成树——Kruskal算法实例

图



最小生成树



最小生成树——Kruskal算法伪代码

```
1: function Kruskal( $G$ )
2:    按照权重  $w(e_1) \leq \dots \leq w(e_{|E|})$  的非递增顺序对集合  $E$  排序;
3:     $E_T \leftarrow \emptyset$ ;
4:     $eCounter \leftarrow 0$ ;  $k \leftarrow 0$ ;
5:    while  $eCounter < (|V| - 1)$  do
6:         $k \leftarrow k + 1$ ;
7:        if  $E_T \cup \{e_k\}$  无回路 then
8:             $E_T \leftarrow E_T \cup \{e_k\}$ ;
9:             $eCounter \leftarrow eCounter + 1$ ;
10:       end if
11:    end while
12:    return  $E_T$ 
13: end function
```

最小生成树——判断新加入的边是否构成回路（难点）

抽象数据类型

包含一些列不相交的子集和以下的操作：

- $\text{makeset}(x)$ 生成一个单元素集合 x ;
- $\text{find}(x)$ 返回一个包含 x 的子集;
- $\text{union}(x, y)$ 构造分别包含 x 和 y 的不相交子集的并集;

操作 $\text{makeset}(x)$

集合： $\{a, b, c, d, e, f\}$

- | | |
|-------------------------------|-------------------------------|
| • $\text{makeset}(a) = \{a\}$ | • $\text{makeset}(d) = \{d\}$ |
| • $\text{makeset}(b) = \{b\}$ | • $\text{makeset}(e) = \{e\}$ |
| • $\text{makeset}(c) = \{c\}$ | • $\text{makeset}(f) = \{f\}$ |

集合： $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}$

最小生成树——判断新加入的边是否构成回路（难点）

操作 $union(x, y)$ 和 $find(x)$

集合: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}$

- $union(a, b) = \{a, b\}$
- $union(c, d) = \{c, d\}$
- $union(a, e) = \{a, b, e\}$
- $union(c, f) = \{c, d, f\}$

集合: $\{a, b, e\}, \{c, d, f\}$

- $find(a) = \{a, b, e\}$
- $find(c) = \{c, d, f\}$
- $find(b) = \{a, b, e\}$
- $find(f) = \{c, d, f\}$

思考

如何用操作 $find$ 和 $union$ 判断回路?

最小生成树——判断新加入的边是否构成回路（难点）

边权重排序

$$① (B, C) = 1$$

$$③ (A, B) = 3$$

$$⑤ (C, F) = 4$$

$$⑦ (D, F) = 5$$

$$② (E, F) = 2$$

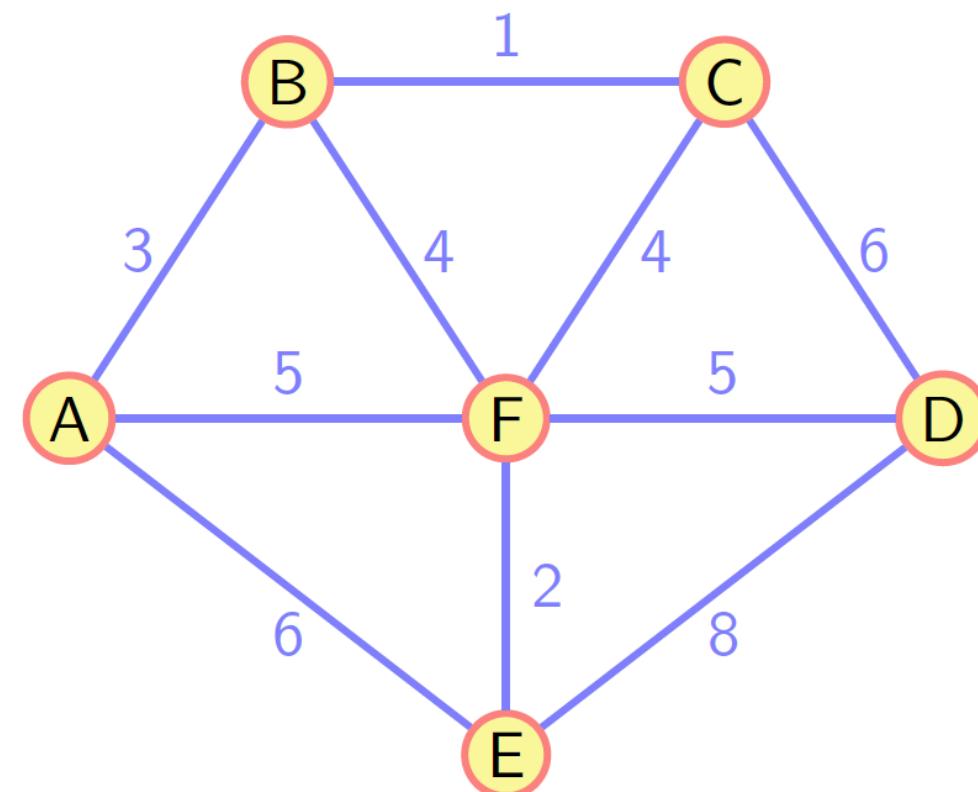
$$④ (B, F) = 4$$

$$⑥ (A, F) = 5$$

⑧ ...

操作过程

- $\text{makeset}(A) = \{A\}$
- $\text{makeset}(B) = \{B\}$
- $\text{makeset}(C) = \{C\}$
- $\text{makeset}(D) = \{D\}$
- $\text{makeset}(E) = \{E\}$
- $\text{makeset}(F) = \{F\}$



最小生成树——判断新加入的边是否构成回路（难点）

边权重排序

$$① (B, C) = 1$$

$$③ (A, B) = 3$$

$$⑤ (C, F) = 4$$

$$⑦ (D, F) = 5$$

$$② (E, F) = 2$$

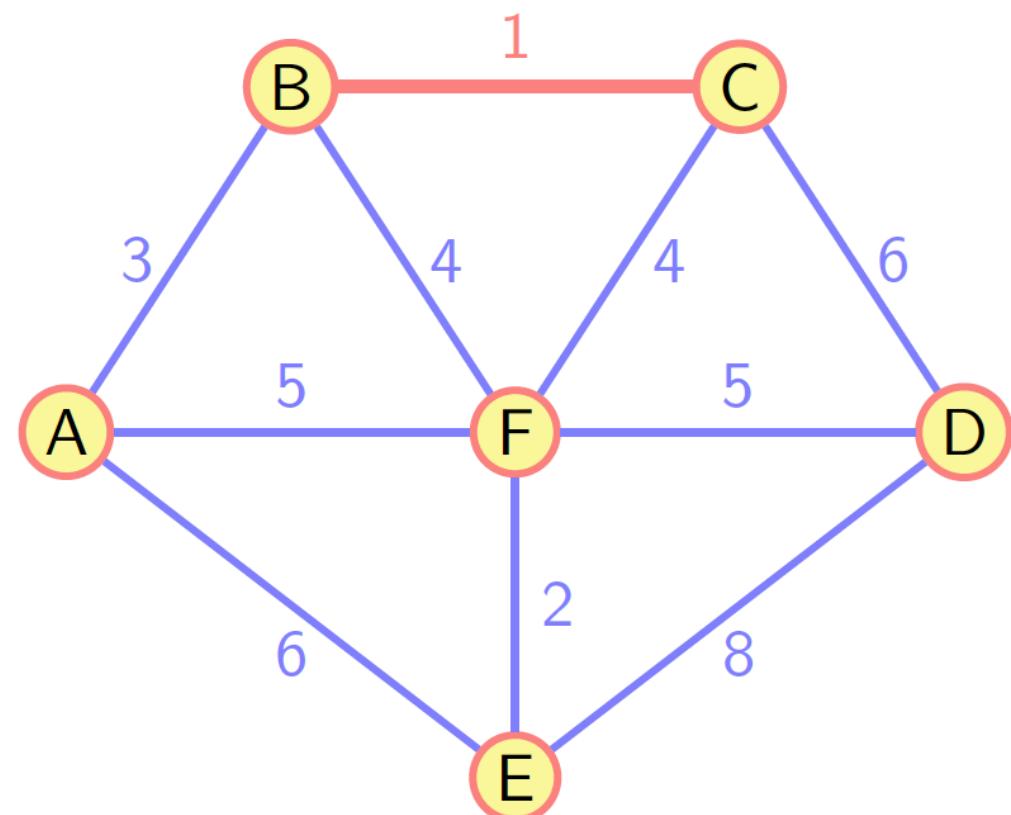
$$④ (B, F) = 4$$

$$⑥ (A, F) = 5$$

$$⑧ \dots$$

操作过程

- $\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}$
- $find(B) = \{B\}$
- $find(C) = \{C\}$
- $union(B, C) = \{B, C\}$
- $\{A\}, \{B, C\}, \{D\}, \{E\}, \{F\}$



最小生成树——判断新加入的边是否构成回路（难点）

边权重排序

$$① (B, C) = 1$$

$$③ (A, B) = 3$$

$$⑤ (C, F) = 4$$

$$⑦ (D, F) = 5$$

$$② (E, F) = 2$$

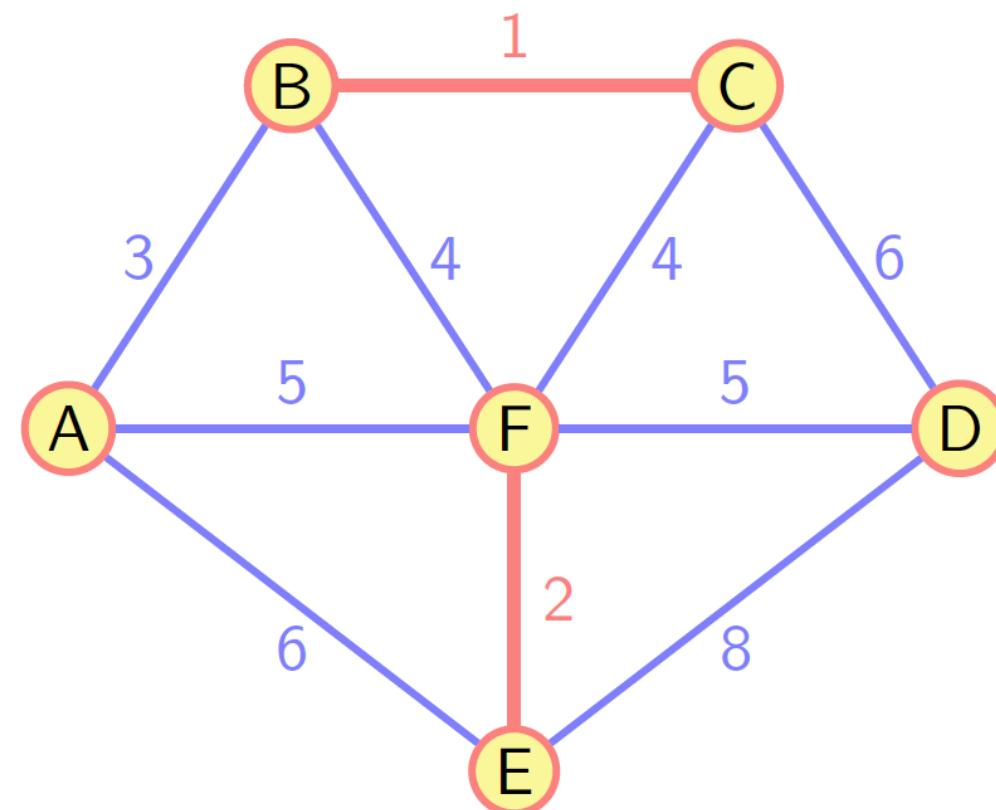
$$④ (B, F) = 4$$

$$⑥ (A, F) = 5$$

⑧ ...

操作过程

- $\{A\}, \{B, C\}, \{D\}, \{E\}, \{F\}$
- $find(E) = \{E\}$
- $find(F) = \{F\}$
- $union(E, F) = \{E, F\}$
- $\{A\}, \{B, C\}, \{D\}, \{E, F\}$



最小生成树——判断新加入的边是否构成回路（难点）

边权重排序

$$① (B, C) = 1$$

$$③ (A, B) = 3$$

$$⑤ (C, F) = 4$$

$$⑦ (D, F) = 5$$

$$② (E, F) = 2$$

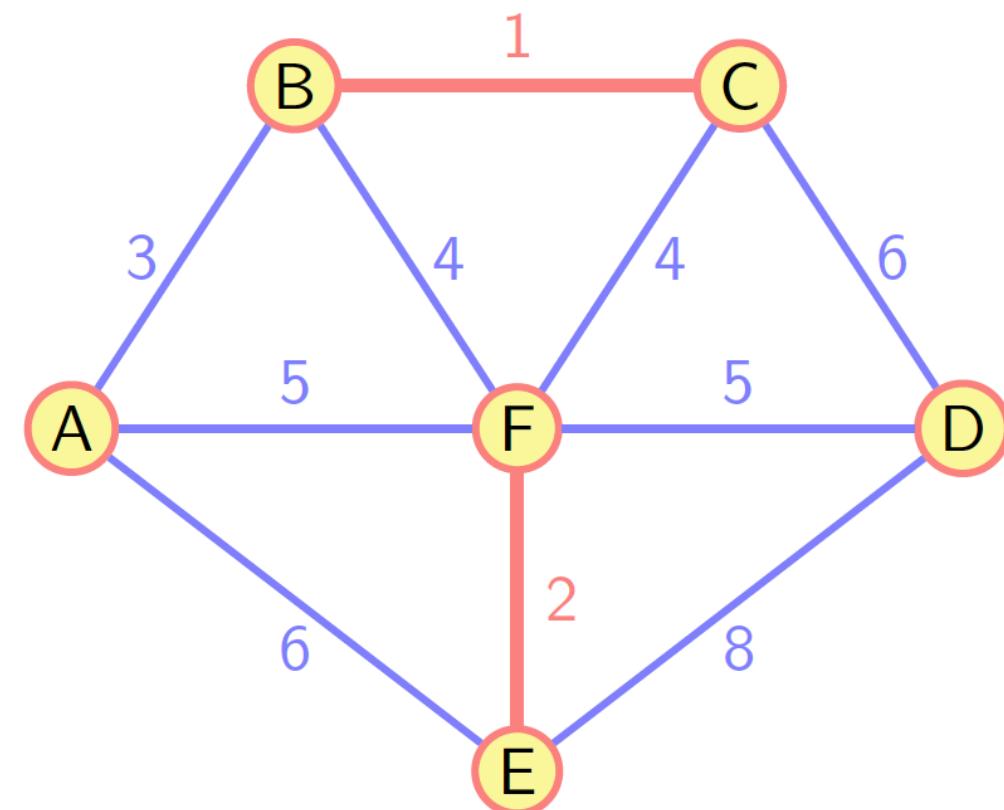
$$④ (B, F) = 4$$

$$⑥ (A, F) = 5$$

⑧ ...

操作过程

- $\{A\}, \{B, C\}, \{D\}, \{E\}, \{F\}$
- $find(E) = \{E\}$
- $find(F) = \{F\}$
- $union(E, F) = \{E, F\}$
- $\{A\}, \{B, C\}, \{D\}, \{E, F\}$



最小生成树——判断新加入的边是否构成回路（难点）

边权重排序

$$① (B, C) = 1$$

$$③ (A, B) = 3$$

$$⑤ (C, F) = 4$$

$$⑦ (D, F) = 5$$

$$② (E, F) = 2$$

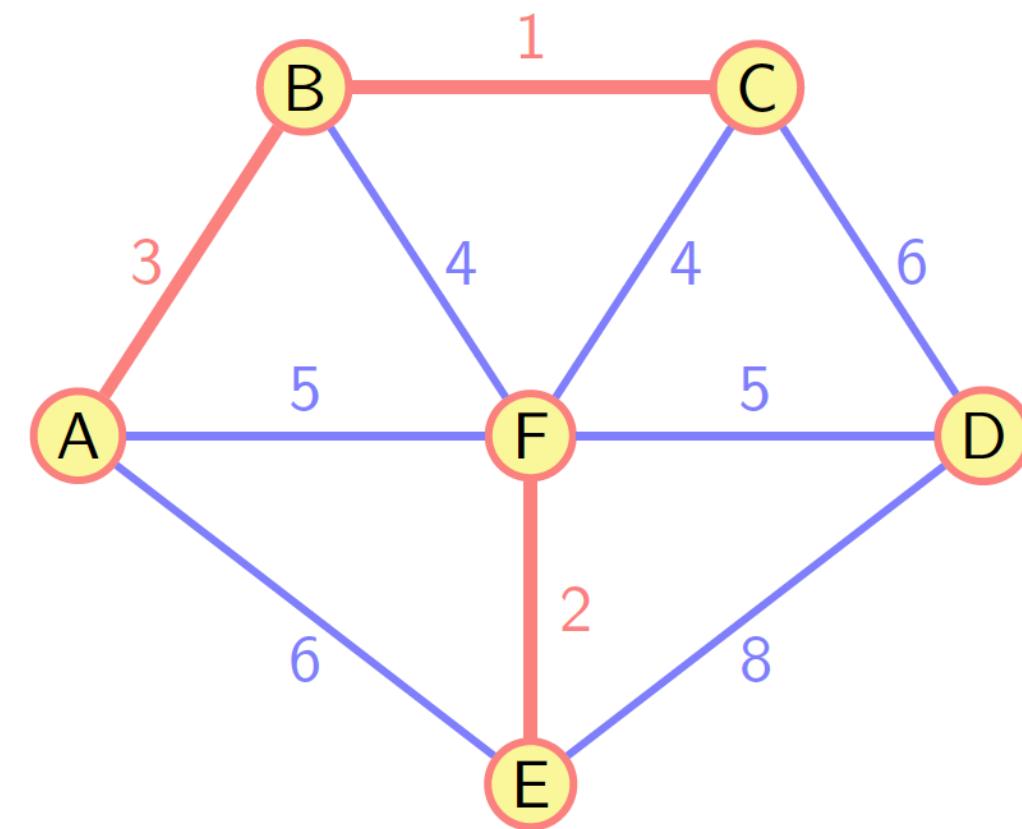
$$④ (B, F) = 4$$

$$⑥ (A, F) = 5$$

⑧ ...

操作过程

- $\{A\}, \{B, C\}, \{D\}, \{E, F\}$
- $find(A) = \{A\}$
- $find(B) = \{B, C\}$
- $union(A, B) = \{A, B, C\}$
- $\{A, B, C\}, \{D\}, \{E, F\}$



最小生成树——判断新加入的边是否构成回路（难点）

边权重排序

$$① (B, C) = 1$$

$$③ (A, B) = 3$$

$$⑤ (C, F) = 4$$

$$⑦ (D, F) = 5$$

$$② (E, F) = 2$$

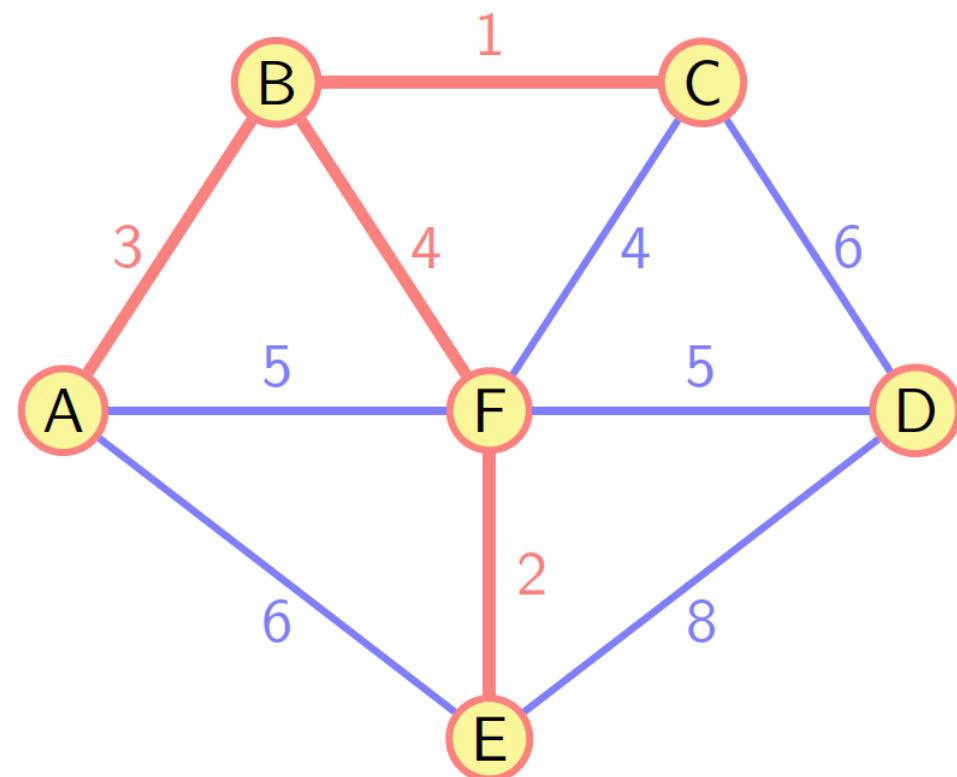
$$④ (B, F) = 4$$

$$⑥ (A, F) = 5$$

⑧ ...

操作过程

- $\{A, B, C\}, \{D\}, \{E, F\}$
- $find(B) = \{A, B, C\}$
- $find(F) = \{E, F\}$
- $union(B, F) = \{A, B, C, E, F\}$
- $\{A, B, C, E, F\}, \{D\}$



最小生成树——判断新加入的边是否构成回路（难点）

边权重排序

$$① (B, C) = 1$$

$$③ (A, B) = 3$$

$$⑤ (C, F) = 4$$

$$⑦ (D, F) = 5$$

$$② (E, F) = 2$$

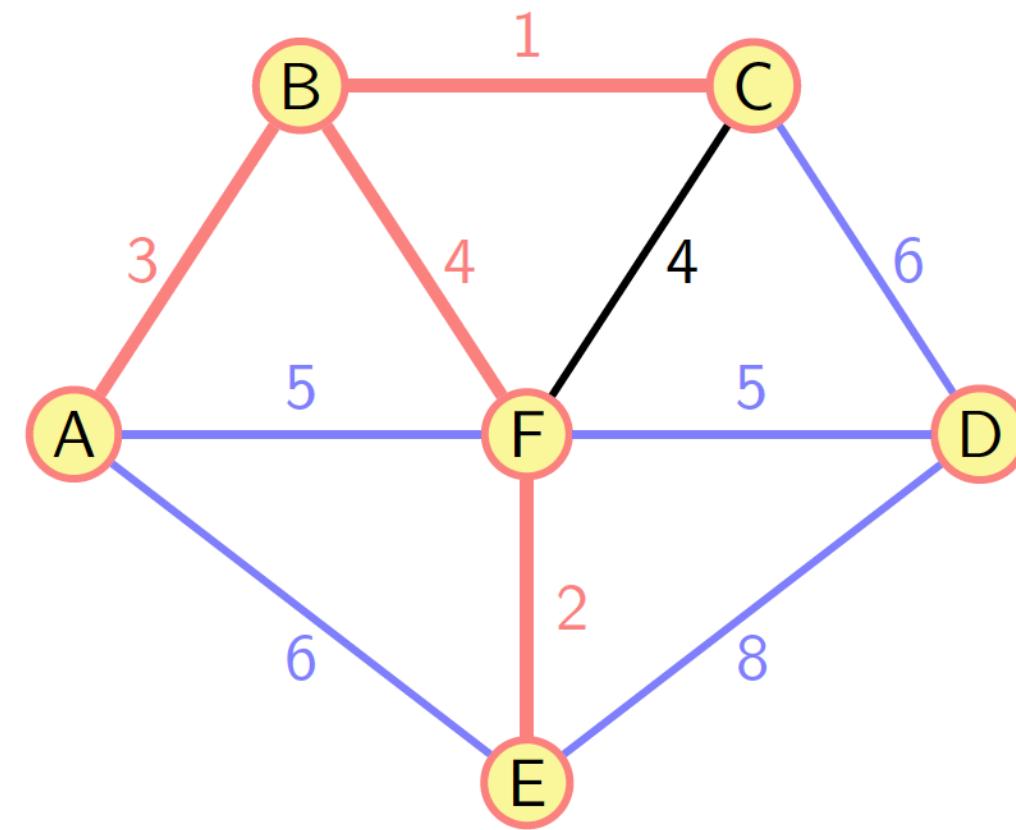
$$④ (B, F) = 4$$

$$⑥ (A, F) = 5$$

⑧ ...

操作过程

- $\{A, B, C, E, F\}, \{D\}$
- $find(C) = \{A, B, C, E, F\}$
- $find(F) = \{A, B, C, E, F\}$
- $find(C) = find(F)$
- 构成回路



最小生成树——判断新加入的边是否构成回路（难点）

边权重排序

$$① (B, C) = 1$$

$$③ (A, B) = 3$$

$$⑤ (C, F) = 4$$

$$⑦ (D, F) = 5$$

$$② (E, F) = 2$$

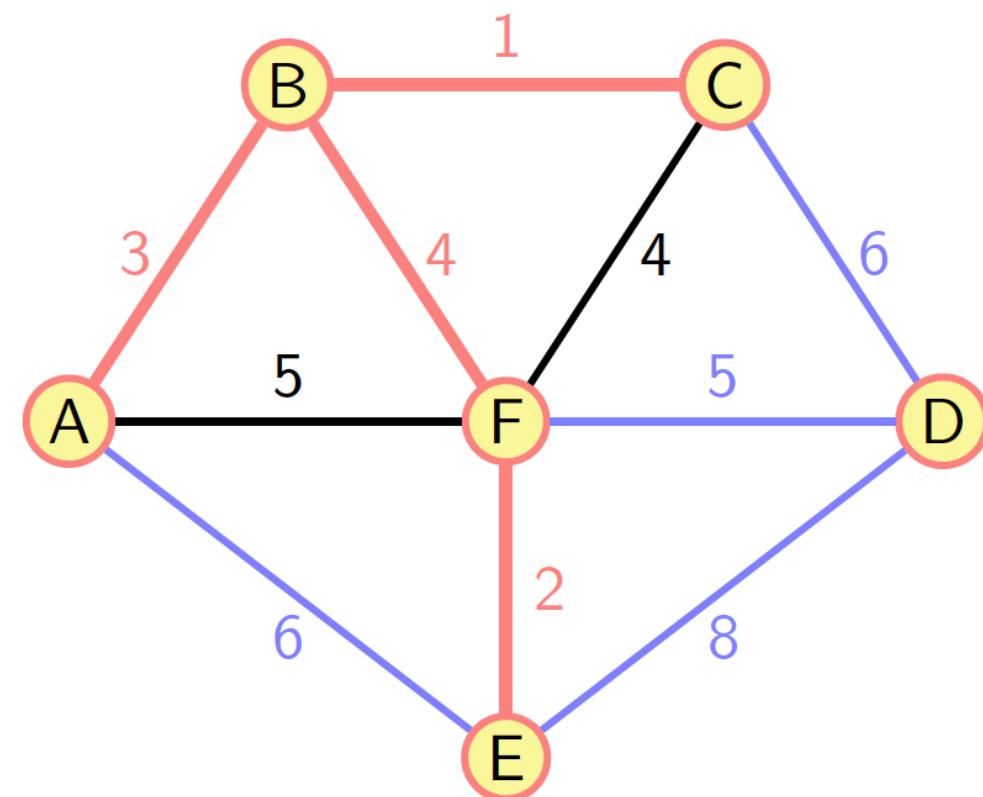
$$④ (B, F) = 4$$

$$⑥ (A, F) = 5$$

⑧ ...

操作过程

- $\{A, B, C, E, F\}, \{D\}$
- $find(A) = \{A, B, C, E, F\}$
- $find(F) = \{A, B, C, E, F\}$
- $find(A) = find(F)$
- 构成回路



最小生成树——判断新加入的边是否构成回路（难点）

边权重排序

$$① (B, C) = 1$$

$$③ (A, B) = 3$$

$$⑤ (C, F) = 4$$

$$⑦ (D, F) = 5$$

$$② (E, F) = 2$$

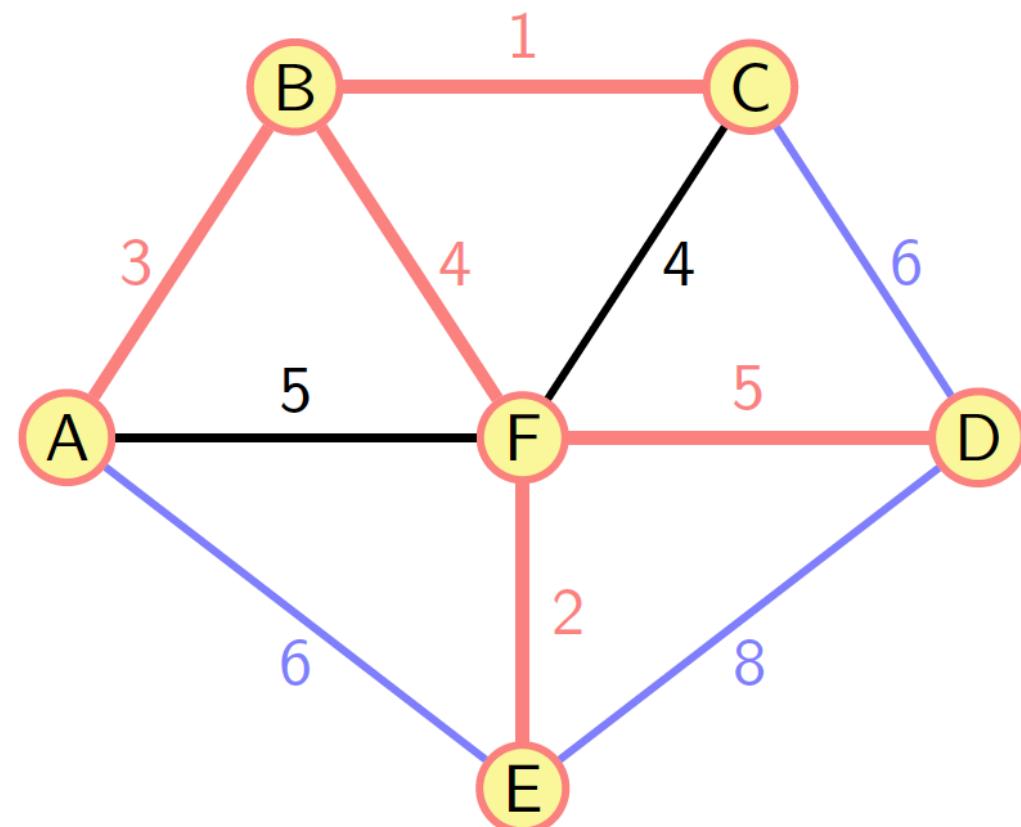
$$④ (B, F) = 4$$

$$⑥ (A, F) = 5$$

$$⑧ \dots$$

操作过程

- $\{A, B, C, E, F\}, \{D\}$
- $find(D) = \{D\}$
- $find(F) = \{A, B, C, E, F\}$
- $union(D, F) =$
 $\{A, B, C, D, E, F\}$
- $\{A, B, C, D, E, F\}$



最小生成树——算法效率分析

算法要点

- ① 按照权重 $w(e_1) \leq \dots \leq w(e_{|E|})$ 的非递增顺序对集合 E 排序；
- ② 判断 $find(a)$ 是否等于 $find(b)$ ，如果不相等， $union(a, b)$ ；

效率分析

- 第1步的时间复杂度为 $O(|E| \log |E|)$ （快速排序）；
- 第2步 $find(x)$ 和 $union(a, b)$ 的复杂度取决于实现方式；
- 快速查找（P253）： $T(find) \in O(1), T(union) \in O(n \log n)$ ；
- 快速求并（P254）： $T(find) \in O(\log n), T(union) \in O(1)$ ；
- 快速查找下总的效率为 $O(|E| \log |E|) + O(m + |V| \log |V|)$ ；
- 快速求并下总的效率为 $O(|E| \log |E|) + O(m \log |V| + |V|)$ ；

哈夫曼树及编码



使用哈夫曼编码进行数据压缩

- 目的
 - 有限的网络带宽
 - 有限的磁盘空间
- 哈夫曼编码被广泛应用于数据压缩，通常情况可以节省20%到90%的存储空间

到90%的存储空间

哈夫曼树及编码——编码策略

两种编码方式

- 定长编码：对每个字符赋予固定长度的编码；
- 变长编码：不同字符的编码长度不同，希望平均长度最短；

二元前缀码

- 为正确解码，要求任何字符的代码不能作为其他字符的前缀；
- $Q = \{001, 00, 010, 01\}$ 不是二元前缀码，假设码字分别代表字符：

$a : 001, b : 00, c : 010, d : 01$

如果接收到序列0100001，那么可能有两种译码方法：

分解为：01, 00, 001，译作： d, b, a

分解为：010, 00, 01，译作： c, b, d

哈夫曼树及编码——数据压缩实例

问题：

假设有一个包含100K字符串的文件

- 每个字符从 a 到 f
- 需要3bit来表示一个字符， 所以这个文件需要300K bits 来存储.
- 可以更好吗？
- 假设我们知道文件中每个字符出现的频率

哈夫曼树及编码——数据压缩实例

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

- 定长编码需要 300K bits
- 变长编码需要 $45*1 + 13*3 + 12*3 + 16*3 + 9*4 + 5*4 = 224K$ bits
- 节省 25% 存储空间.

哈夫曼树及编码——最优前缀码

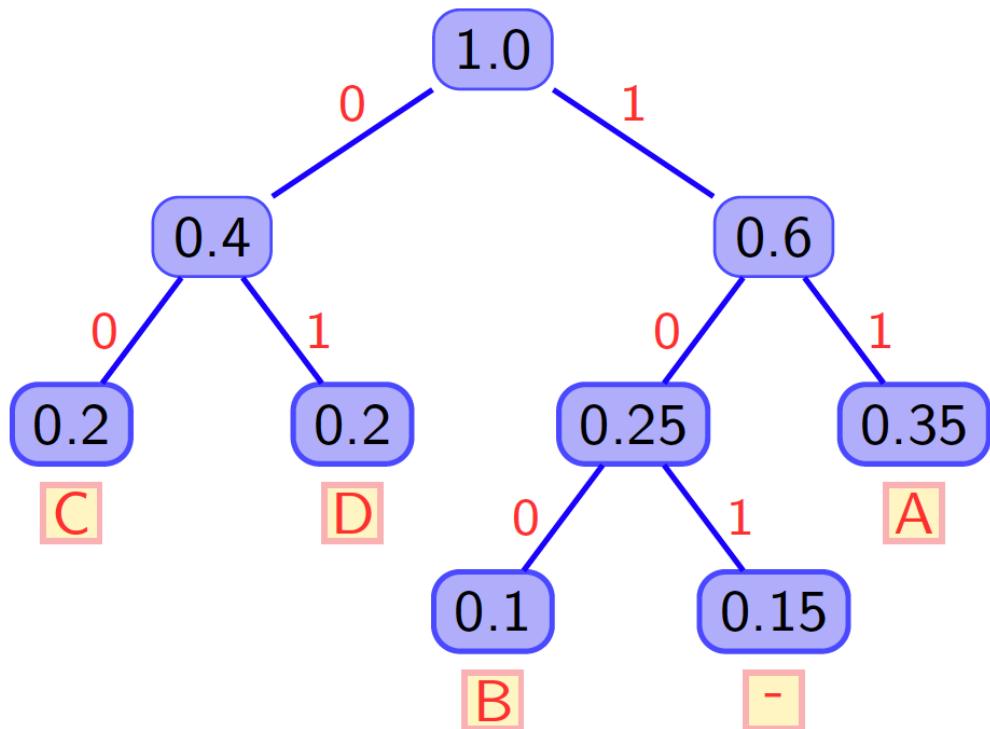
最优二元前缀码

平均使用二进制位数 B 最少的前缀码， $B = \sum_{i=1}^n p(x_i)f(x_i)$ ， $f(x_i)$ ，其中 $f(x_i)$ 表示 x_i 的编码长度， $p(x_i)$ 表示 x_i 出现的概率)。

例子

字符	出现概率	代码字
A	0.35	11
B	0.1	100
C	0.2	00
D	0.2	01
-	0.15	101

如何构造最优二元前缀码的二叉树？



每个字符平均长度 $B = 2.25$

哈夫曼算法

算法思想

构造一棵树，将较短编码（树较浅）配给高频字符，较长编码（树较深）配给低频字符。

算法流程

- ① 初始化 n 个只包含一个节点的树，每个树的权重为节点的概率；
- ② 找到两颗权重最小的树，把它们作为新树的左右子树，新树的权重为左右子树的权重之和；
- ③ 重复步骤2，直到只剩一棵单独的树；

注

用哈夫曼算法构造的树称为哈夫曼树，所得到的编码称为哈夫曼编码。

哈夫曼算法——算法过程

0.2

C

0.2

D

0.35

A

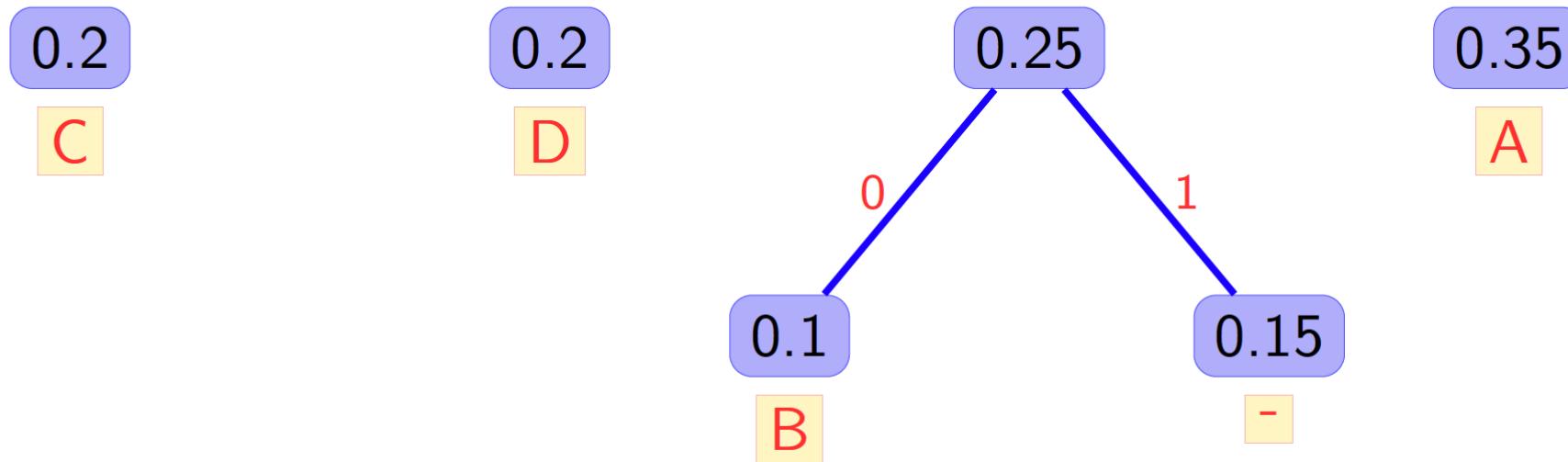
0.1

B

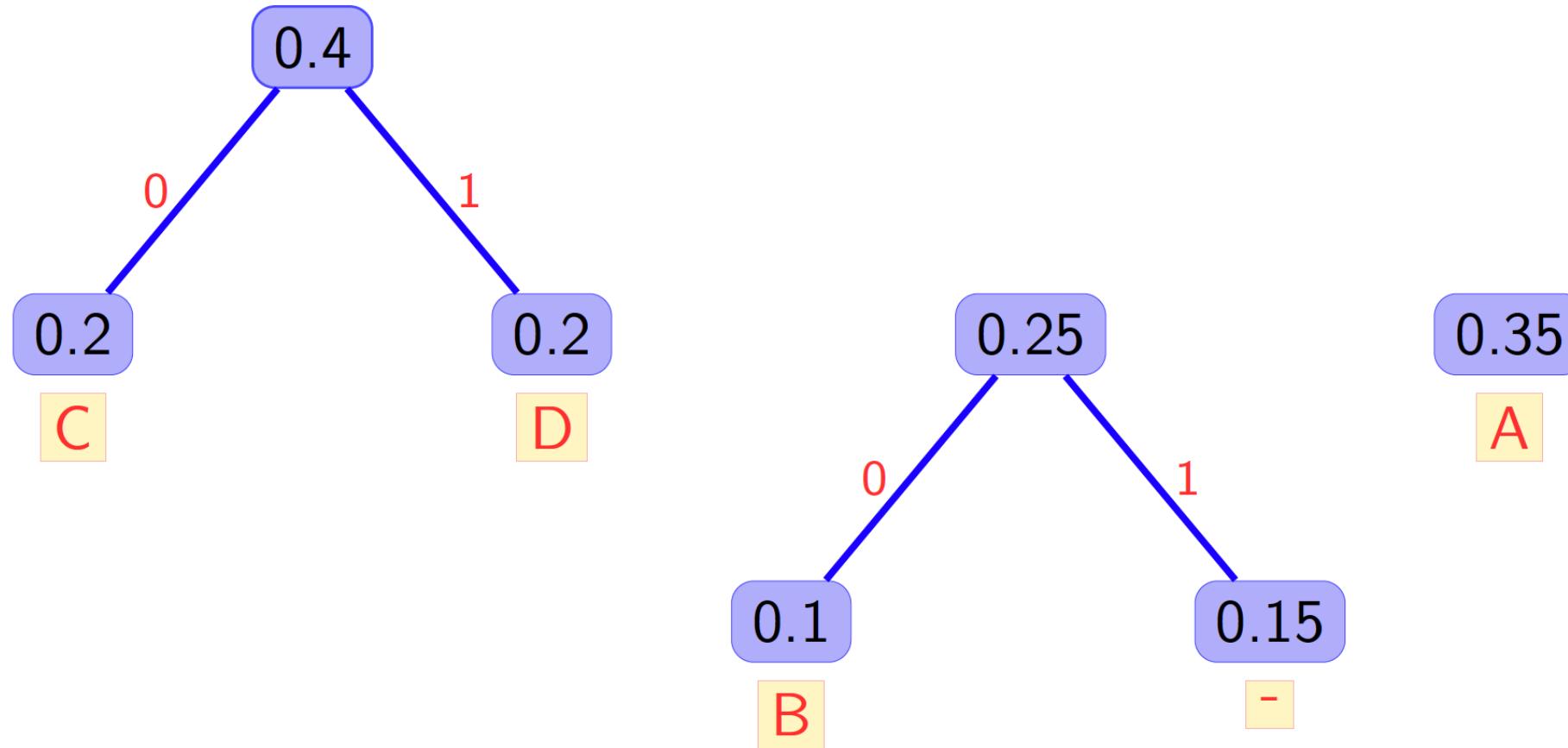
0.15

-

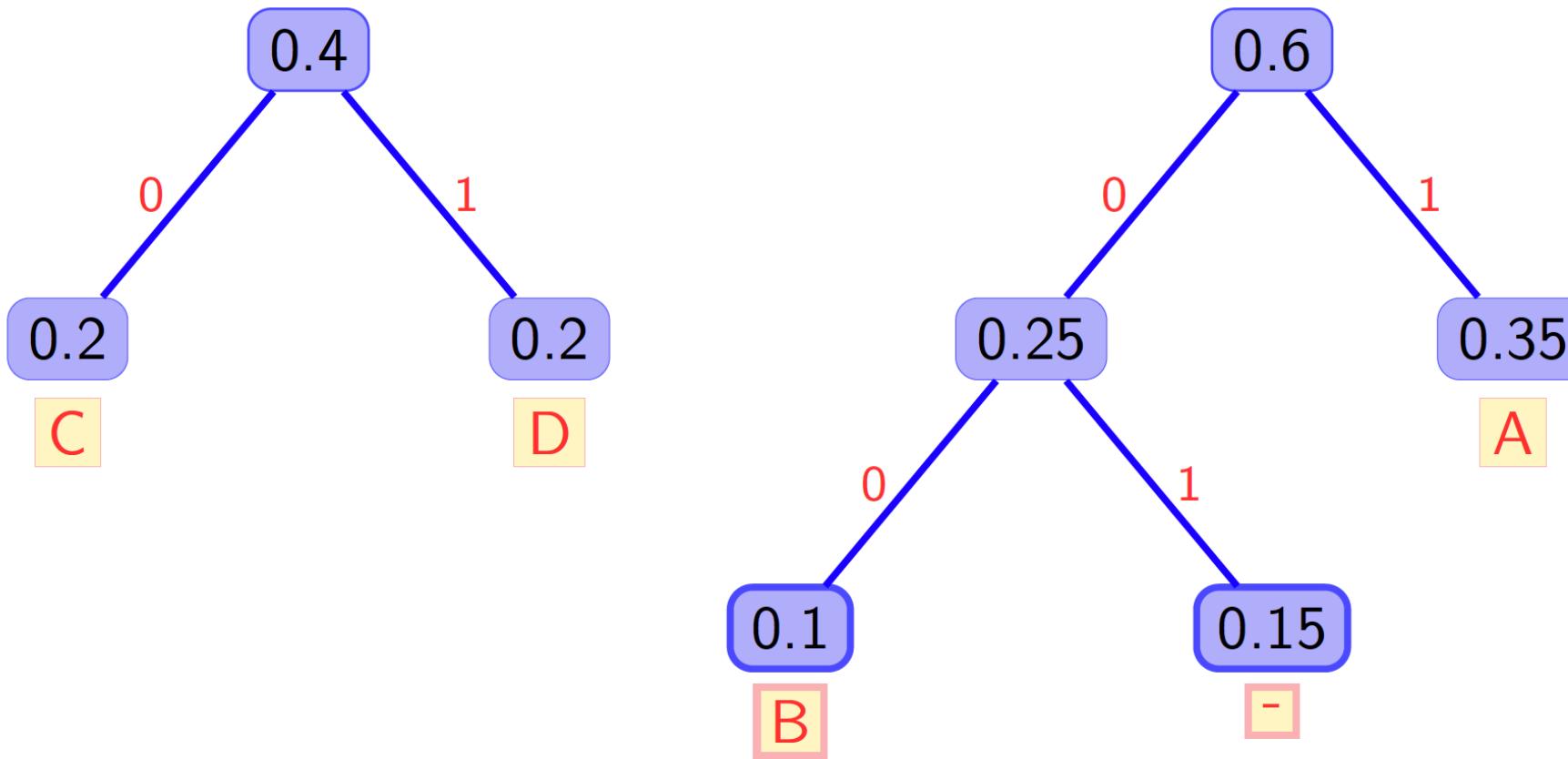
哈夫曼算法——算法过程



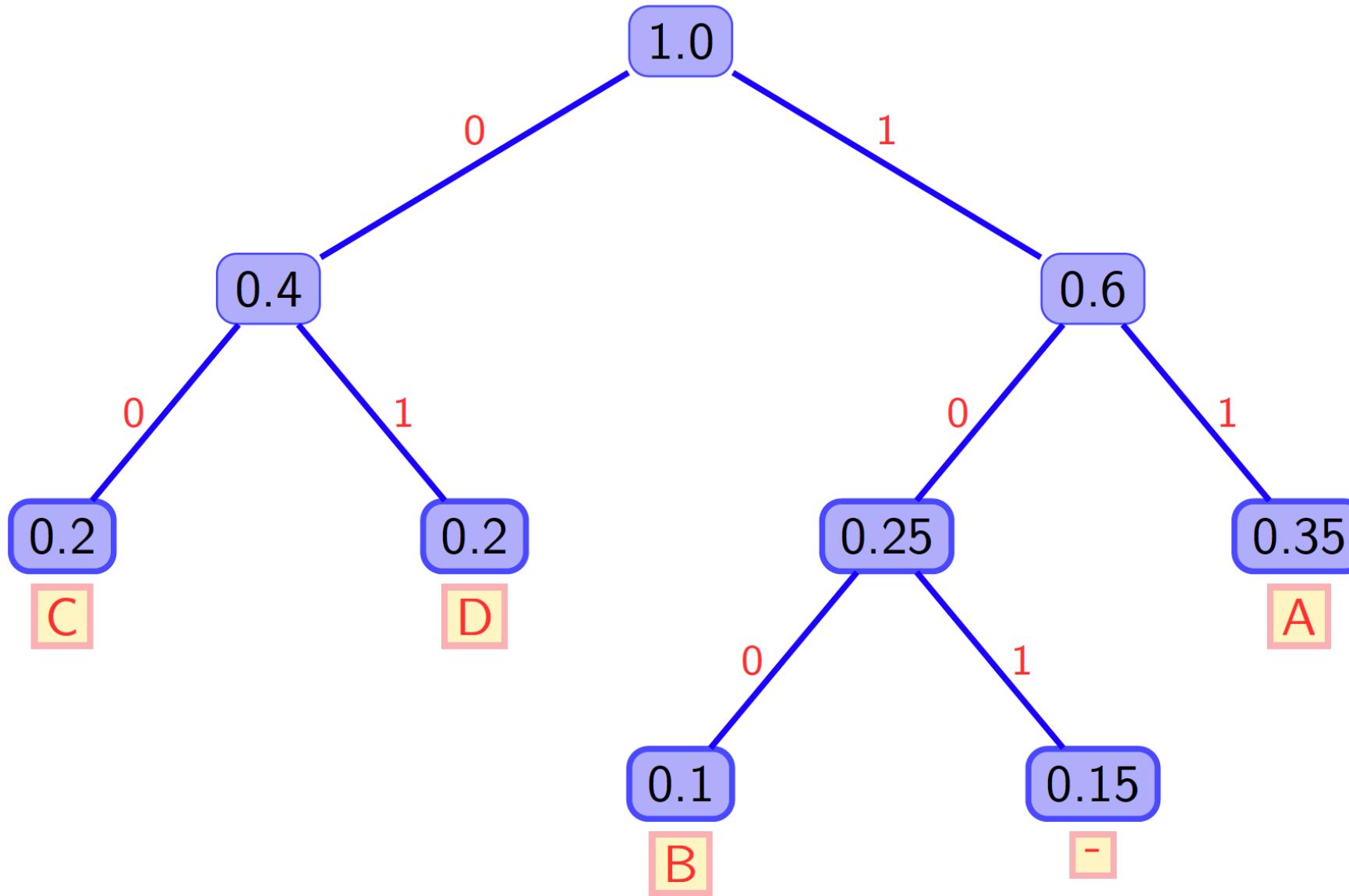
哈夫曼算法——算法过程



哈夫曼算法——算法过程



哈夫曼算法——算法过程



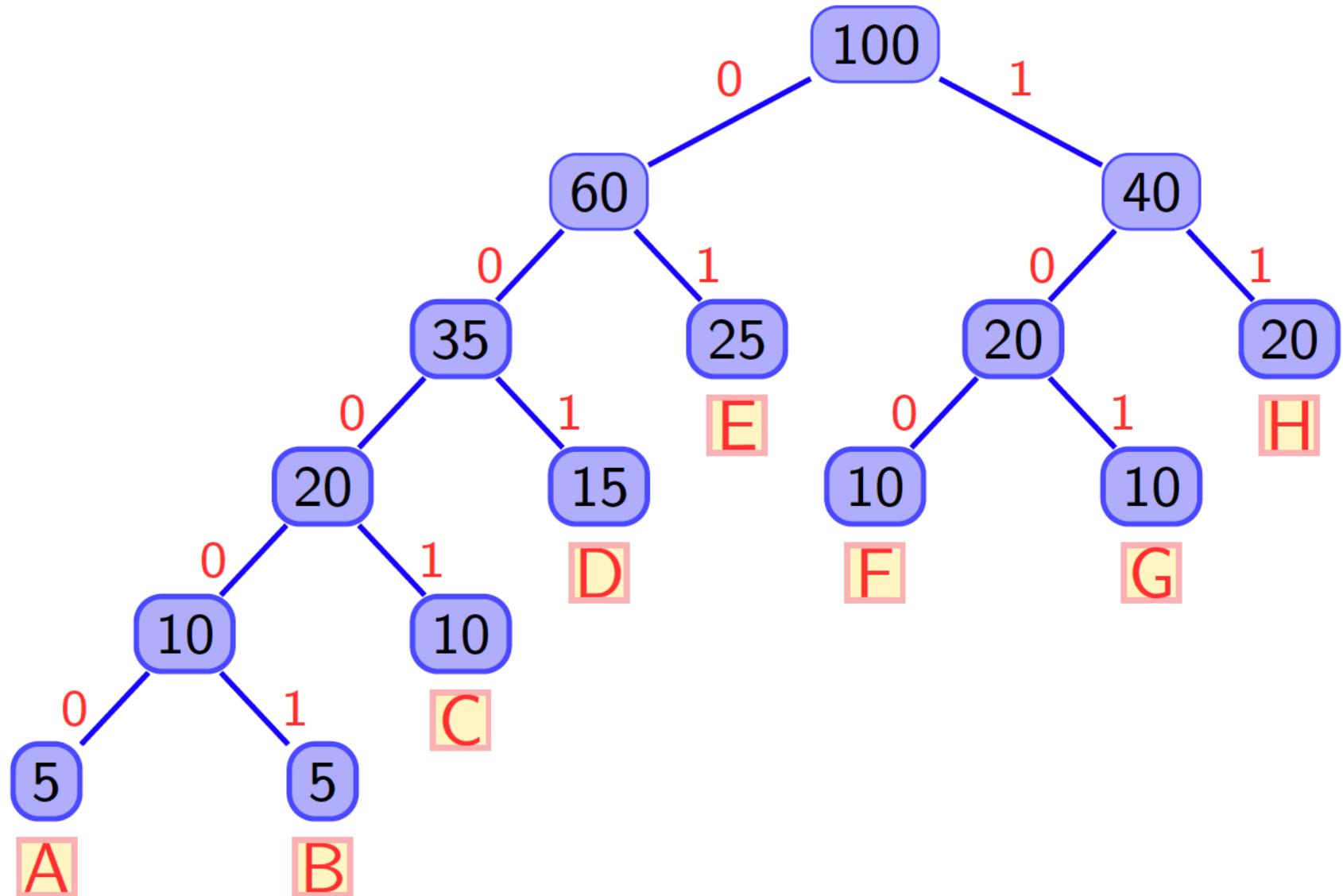
哈夫曼算法——算法练习

字符及出现概率

字符	A	B	C	D	E	F	G	H
概率 (%)	5	5	10	15	25	10	10	20

哈夫曼算法——算法练习

哈夫曼树



哈夫曼算法——伪代码

构造最优二元前缀码的二叉树的哈夫曼算法

```
1: function Haffman( $C$ ) // $C = \{x_1, x_2 \dots x_n\}$ 为字符集, 每个字符频率为 $f_i$ 
2:    $n \leftarrow |C|;$ 
3:    $Q \leftarrow \text{sort}(C)$  ;    //按照频率递增构成队列Q
4:   for  $i \leftarrow 1 \rightarrow n - 1$  do
5:      $z \leftarrow \text{AllocateNode}();$       //生成节点 $z$ 
6:      $z.left \leftarrow Q$ 中最小元;
7:      $z.right \leftarrow Q$ 中次小元;
8:      $f(z) \leftarrow f(x) + f(y);$ 
9:     Insert( $Q, z$ );      //将 $z$ 插入 $Q$ 
10:   end for
11:   return  $Q$ 
12: end function
```

效率分析

主要步骤

```
1: function Haffman( $C$ )
2:    $Q \leftarrow \text{sort}(C)$  ;    //按照频率递增构成队列Q
3:   for  $i \leftarrow 1 \rightarrow n - 1$  do
4:     构造节点 $z$ ;
5:     Insert( $Q, z$ ) ;      //将 $z$ 插入 $Q$ 
6:   end for
7: end function
```

- 行2排序时间复杂度为 $O(n\log n)$;
- 行3执行循环 $n - 1$ 次，行5插入操作的复杂度为 $O(\log n)$;
- 算法的时间复杂度为 $O(n\log n)$;

贪心算法的基本要素——0-1背包问题

- 0-1背包问题：给定n种物品和一个背包。物品*i*的重量是 W_i ，其价值为 V_i ，背包的容量为C。应如何选择装入背包的物品，使得装入背包中物品的总价值最大？
- 对每种物品*i*只有2种选择，即装入背包或不装入背包。
- 不能将物品*i*装入背包多次，也不能只装入部分的物品*i*

具有**最优子结构性质**，0-1背包问题却不能用贪心算法求解。应比较选择该物品和不选择该物品所导致的最终方案，然后再作出最好选择。由此就导出许多**互相重叠的子问题**——用**动态规划算法**求解的另一重要特征。

贪心算法的基本要素——背包问题

与0-1背包问题类似，不能将物品 i 装入背包多次，所不同的是在选择物品 i 装入背包时，可以选择物品 i 的一部分，而不一定要全部装入背包， $1 \leq i \leq n$ 。

背包问题

有5样东西的重量和价值如下：背包的最多能装100磅的物品

value(\$US)	20	30	65	40	60
weight(Lbs)	10	20	30	40	50
value/weight	2	1.5	2.1	1	1.2

方法1：首先选择重量最小的

$$\text{总重量} = 10 + 20 + 30 + 40 = 100$$

- 总价值 = $20 + 30 + 65 + 40 = 155$

具有**最优子结构性质**，但背包问题可以用贪心算法求解——**贪心选择性质**。

方法3：选择单位重量价值最高的

- 总重量 = $30 + 10 + 20 + \textcolor{red}{40} = 100$

- 总价值 = $65 + 20 + 30 + \textcolor{red}{48} = 163$

可以得出什么结论？

贪心算法解背包问题的基本步骤

- 计算每种物品单位重量的价值 V_i/W_i
- 依贪心选择策略，将尽可能多的**单位重量价值最高的**物品装入背包
- 若将这种物品全部装入背包后，背包内的物品总重量未超过 C ，则选择**单位重量价值次高的**物品并尽可能多地装入背包
- 依此策略一直地进行下去，直到背包装满为止

贪心算法解背包问题具体算法

GREEDY KNAPSACK

```
GREEDY-KNAPSACK( p, w, W, x, n )  
1 x ← 0 //initialize solution to zero //  
2 c ← W // c = remaining knapsack capacity //  
3 for i ← 1 to n do  
4   if w(i) ≤ c  
5     then x(i) ← 1  
6       c ← c - w(i)  
8   else exit // exit for loop  
9 if i ≤ n  
10 then x(i) ← c/w(i)  
11 return x
```

算法时间上界
为O(nlogn)

作业 3

- 阅读
 - 9.1-9.2, 9.4
- 习题
 - 9.1: 1,3,
 - 9.2: 1