数据分析与算法设计
第9章 贪婪技术
（Greedy Technique）

李旻
百人计划研究员
浙江大学 信息与电子工程学院
Email: min.li@zju.edu.cn

贪婪技术

- 基本思想：通过一系列步骤来构造问题的解，每一步对目前构造的部分解做一个扩展，直到获得问题的完整解为止。所做的每一步选择必须满足：
  - 可行性：必须满足问题的约束
  - 局部最优：它是当前步骤中所有可行选择中最佳的局部选择
  - 不可取消：选择一旦做出，在后面的步骤中无法改变
- 贪婪技术并没有固定的算法解决框架，算法的关键是贪婪策略的选择，根据不同的问题选择不同的策略
- 贪婪技术往往只能获得近似最优解，但是某些特定问题中可以证明其最优性

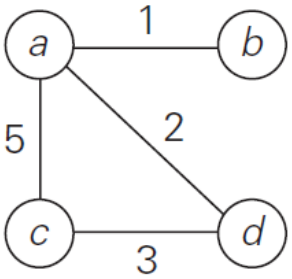
目录

- 最小生成树的构造
  - Prim算法
  - Kruskal算法
- 求解加权图最短路径
  - Dijkstra算法
- 数据压缩
  - 哈夫曼树及编码

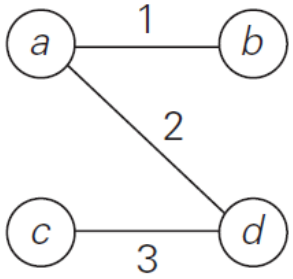
最小生成树的构造

最小生成树 (MST: Minimum Spanning Tree)

- 连通图G的生成树：包括G所有顶点的连通无环子图；
- 树的权重：所有边的权重总和；
- 连通图G带权最小生成树：权重总和最小的生成树T

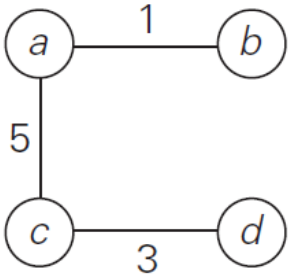


graph

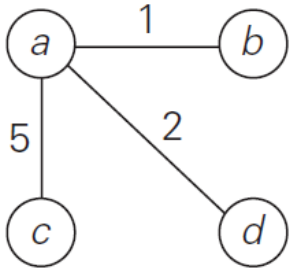


$w(T_1) = 6$

最小生成树



$w(T_2) = 9$



$w(T_3) = 8$

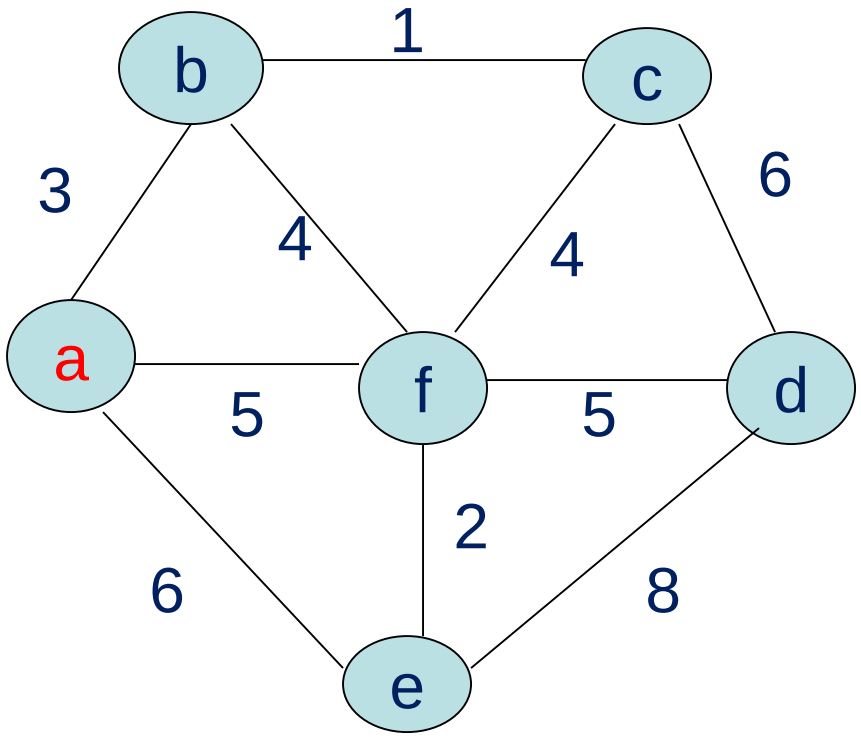
Prim算法

- 基本思想：通过一系列不断扩张的子树来构造
  - 从图的顶点集合中任选一个顶点，作为初始子树
  - 以一种贪婪的方式扩张当前的生成树，将不在树中的最近顶点添加到树中（最近顶点是指与树中的顶点相连，且边权重最小的一个顶点）
  - 上述操作迭代n-1次后，完成生成树的构造

- 迭代过程中，需要更新的信息
  - 生成树中的顶点集合、不在生成树中的顶点集合
  - 不在生成树中的顶点到生成树顶点的最短距离

Prim算法

初始a为已选顶点，初始已选顶点集合为{a}

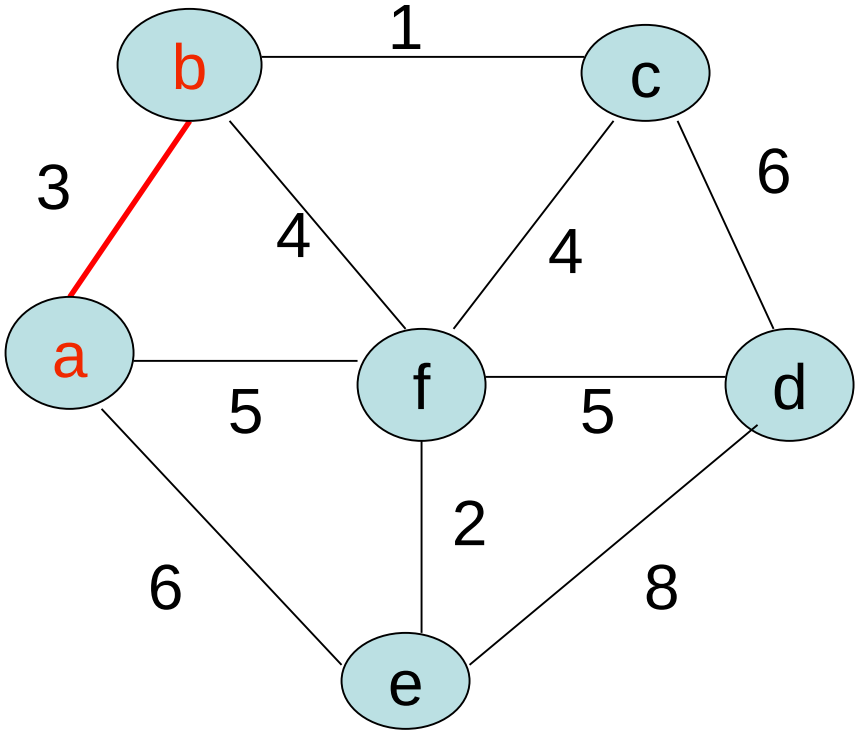


a(-,-)	**b(a,3)**, c(-,∞),d(-,∞),e(a,6),f(a,5)

先选中一个顶点作为已选顶点，然后从未选顶点集合中选一个顶点到已选顶点集合中有最小权值的边，标记该顶点。

Prim算法

已选顶点集合为{a,b}

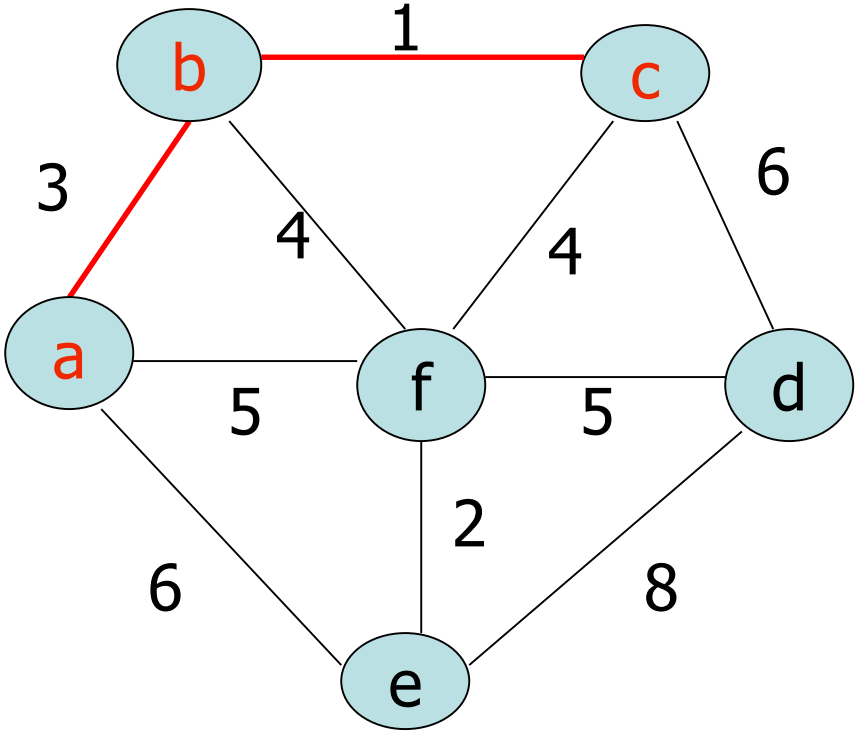


a(-,-)	**b(a,3)**, c(-,∞),d(-,∞),e(a,6),f(a,5)
b(a,3)	

因为b到已选顶点集合有最小权值边w(a,b)，所以把b加入已选顶点集合。

Prim算法

已选顶点集合为{a,b,c}

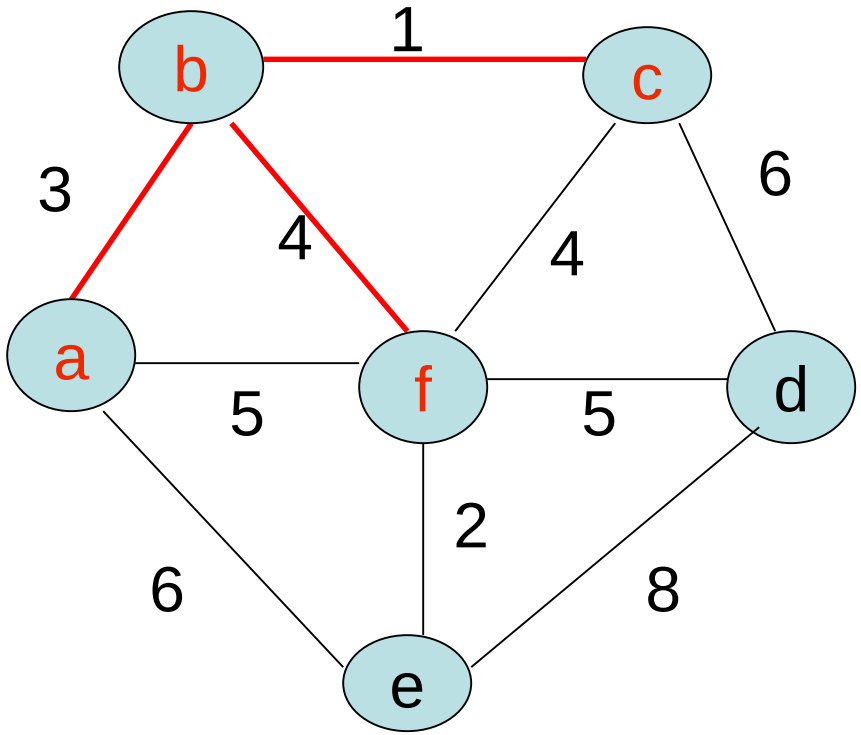


a(-,-)	**b(a,3)**, c(-,∞),d(-,∞),e(a,6),f(a,5)
b(a,3)	**c(b,1)**,d(-, ∞), e(a,6) ,f(b,4)
c(b,1)	

因为c 到已选顶点集合有最小权值边w(b,c），所以把c加入已选顶点集合。

Prim算法

已选顶点集合为{a,b,c,f}

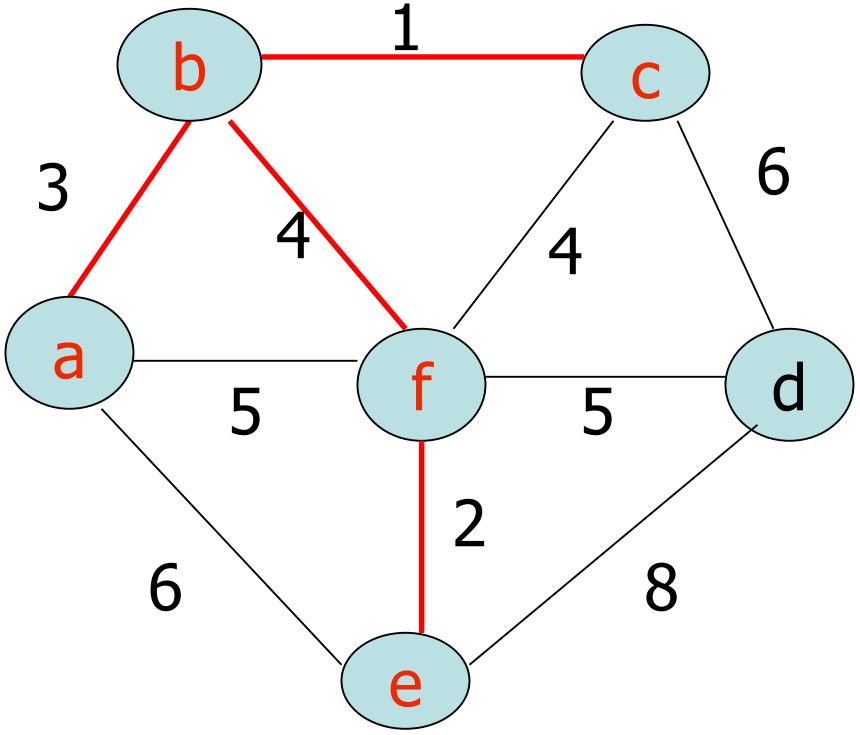


a(-,-)	**b(a,3)**, c(-,∞),d(-,∞),e(a,6),f(a,5)
b(a,3)	**c(b,1)**,d(-, ∞), e(a,6) ,f(b,4)
c(b,1)	d(c,6),e(a,6),**f(b,4)**
f(b,4)	

因为f 到已选顶点集合有最小权值边w(b,f),
所以把f 加入已选顶点集合。

Prim算法

已选顶点集合为{a,b,c,f,e}

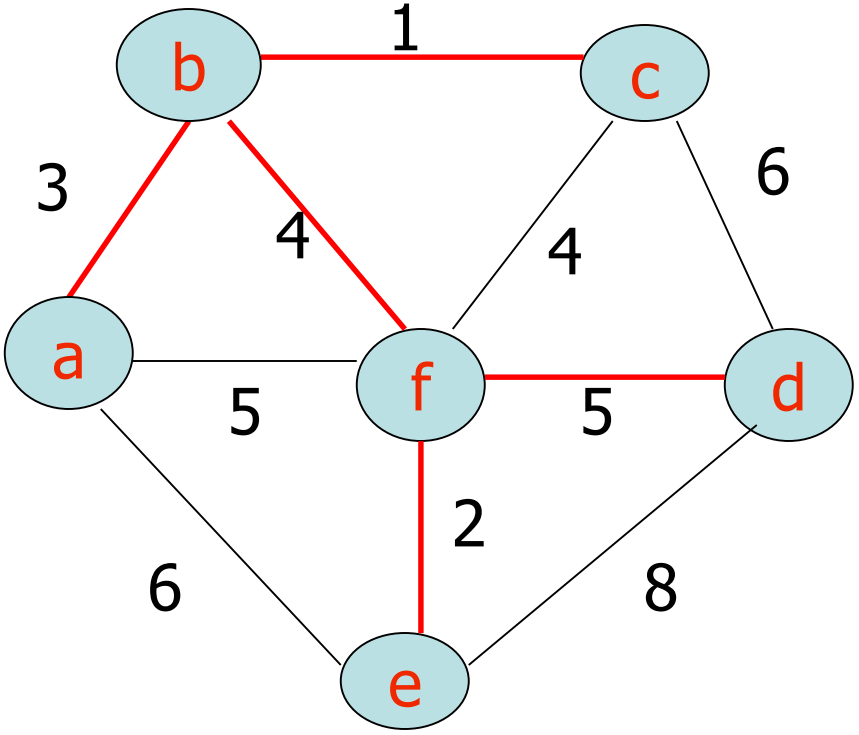


a(-,-)	**b(a,3)**, c(-,∞),d(-,∞),e(a,6),f(a,5)
b(a,3)	**c(b,1)**,d(-, ∞), e(a,6) ,f(b,4)
c(b,1)	d(c,6),e(a,6),**f(b,4)**
f(b,4)	d(f,5),**e(f,2)**
e(f,2)	

因为e 到已选顶点集合有最小权值边w(f ,e)，所以把f 加入已选顶点集合。

Prim算法

因为d 到已选顶点集合有最小权值边w(f ,d)，所以把d加入已选顶点集合



a(-,-)	**b(a,3)**, c(-,∞),d(-,∞),e(a,6),f(a,5)
b(a,3)	**c(b,1)**,d(-, ∞), e(a,6) ,f(b,4)
c(b,1)	d(c,6),e(a,6),**f(b,4)**
f(b,4)	d(f,5),**e(f,2)**
e(f,2)	**d(f,5)**
d(f,5)	

已选顶点集合{a,b,c,f,e,d}

未选顶点集合为空集，算法结束。

Prim算法

- 算法伪代码

算法 Prim (G)

//构造最小生成树的 Prim 算法

//输入：加权连通图 $G=<V, E>$

//输出：E_T，组成 G 的最小生成树的边的集合

$V_T \leftarrow \{v_0\}$　　//可以用任意顶点来初始化树的顶点集合

$E_T \leftarrow \varnothing$

for $i \leftarrow 1$ **to** $|V|-1$ **do**

在所有的边 (v, u) 中，求权重最小的边 $e^*=(v^*, u^*)$，

使得 v 在 V_T 中，而 u 在 $V-V_T$ 中

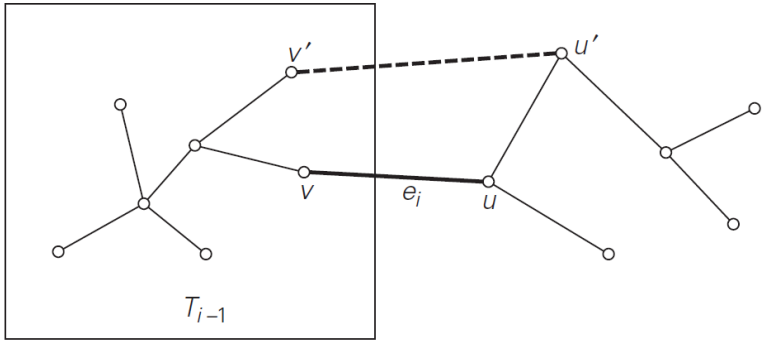
$V_T \leftarrow V_T \cup \{u^*\}$

$E_T \leftarrow E_T \cup \{e^*\}$

return E_T

Prim算法

- 正确性（最优性）证明：
  - T_0（只含一个顶点的初始树），显然是最小生成树的一部分
  - 假设T_{i-1}是最小生成树T的一部分，则需证明：用Prim算法从T_{i-1}生成的T_i也是最小生成树的一部分。记$e_i = (v,u)$是从T_{i-1}拓展到T_i时添加的边。
    - 反证法：如果e_i不属于包括T在内的任何最小生成树，则把e_i加到T中，就会形成一条回路



    - 除了边$e_i = (v,u)$，该回路必定还包含另一条边(v',u')，它把一个顶点$v' \epsilon T_{i-1}$和不在T_{i-1}中的顶点u'连接起来。而e_i的权重小于或等于(v',u')的权重，如果删除(v',u')，则可得到另一棵生成树，且权重小于等于T的权重；与假设矛盾！

Prim算法

- 效率：取决于表示图本身的数据结构、表示集合$V - V_T$的优先队列的数据结构
  - 用权重矩阵表示图，用一个无序数组来实现优先队列时，为$\Theta(|V|^2)$
  - 用邻接链表表示$|V|$个顶点、$|E|$条边的图，用最小堆实现优先级队列时，为$(|V| - 1 + |E|)\mathrm{O}(\log|V|) = \mathrm{O}(|E|\log|V|)$

- 是否还有其它的贪婪策略呢？

目录

- 最小生成树的构造
  - Prim算法
  - Kruskal算法
- 求解加权图最短路径
  - Dijkstra算法
- 数据压缩
  - 哈夫曼树及编码

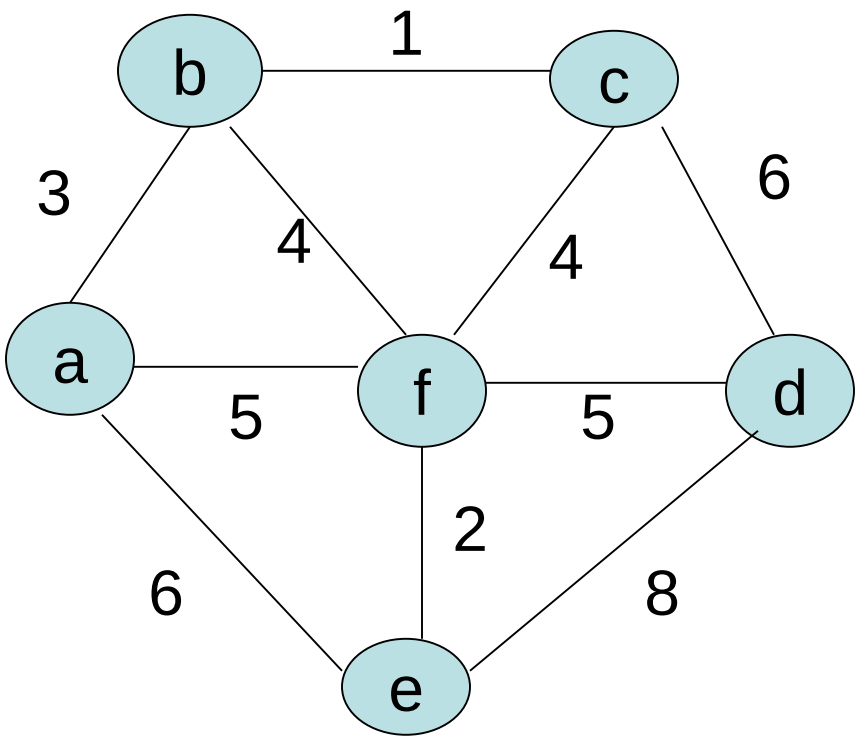
Kruskal算法

- 基本思想：通过一系列不断扩展的子图来构造
  - 排序：将图的边集E中所有边按权值从小到大排序
  - 用权值最小的边初始化边集E_T
  - 按权值顺序访问后续每条边，若将相应的边加入子图中不会产生回路，则把该边加入最小生成树的边集E_T中
  - 当边集E_T的边个数为n-1时，得到的$G_T=(V_T, E_T)$即为最小生成树

- 与Prim算法一样，该贪婪策略也可以保证最优性

Kruskal算法

bc	1
ef	2
ab	3
bf	4
cf	4
af	5
fd	5
ae	6
cd	6
ed	8

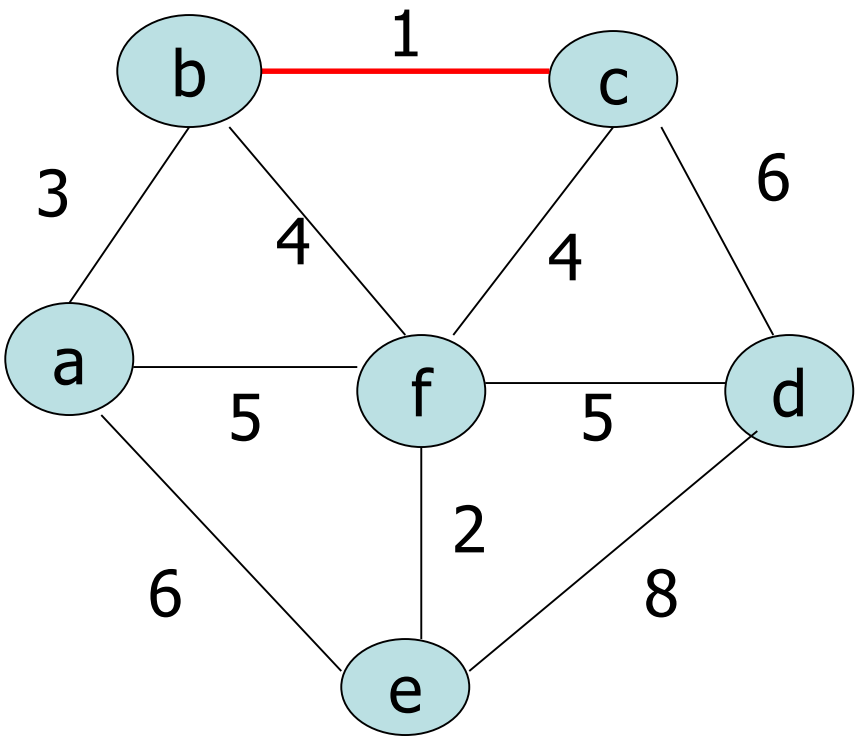


初始已选边集合为空集

每次从未选边集合中选一条权值最小的边连接到已选边集合，当连成的新图包含环路时，抛弃此次选择，不含环路则加入该边到已选边集合。

Kruskal算法

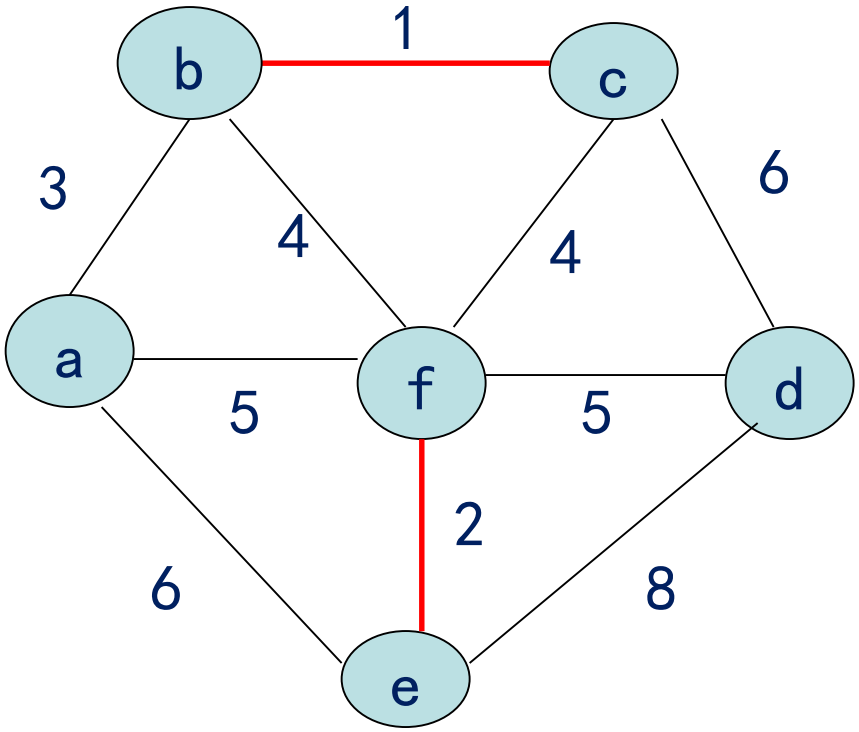
bc	1
ef	2
ab	3
bf	4
cf	4
af	5
fd	5
ae	6
cd	6
ed	8



已选边集合为
{ (b,c) }

Kruskal算法

bc	1
ef	2
ab	3
bf	4
cf	4
af	5
fd	5
ae	6
cd	6
ed	8

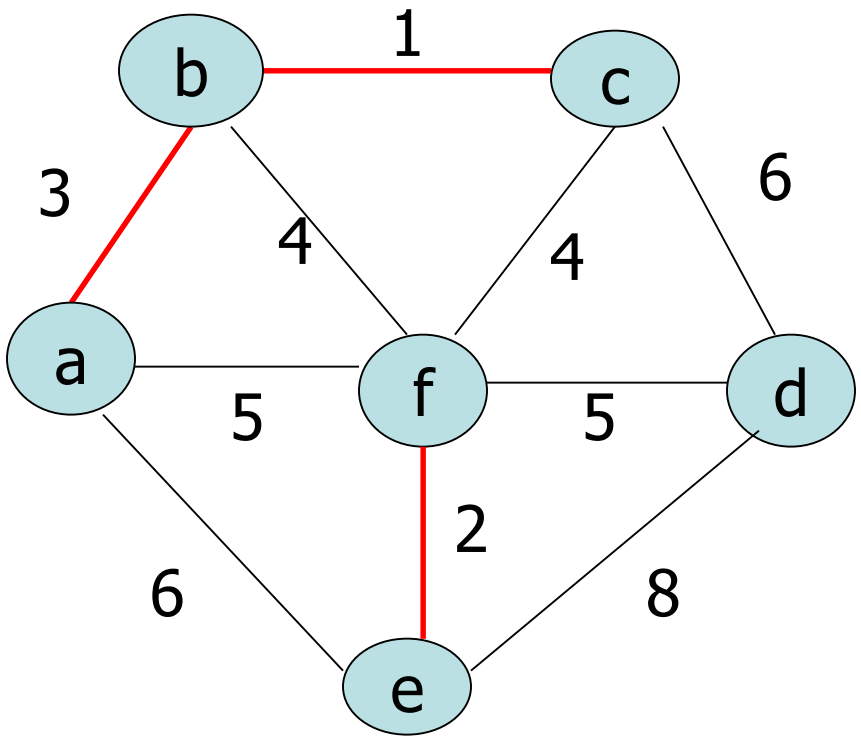


已选边集合为
{ (b, c), (f, e) }

因为边w(f, e)为未选边集合中最小权值的边，且加入后，已选边集合中不含回路。

Kruskal算法

bc	1
ef	2
ab	3
bf	4
cf	4
af	5
fd	5
ae	6
cd	6
ed	8

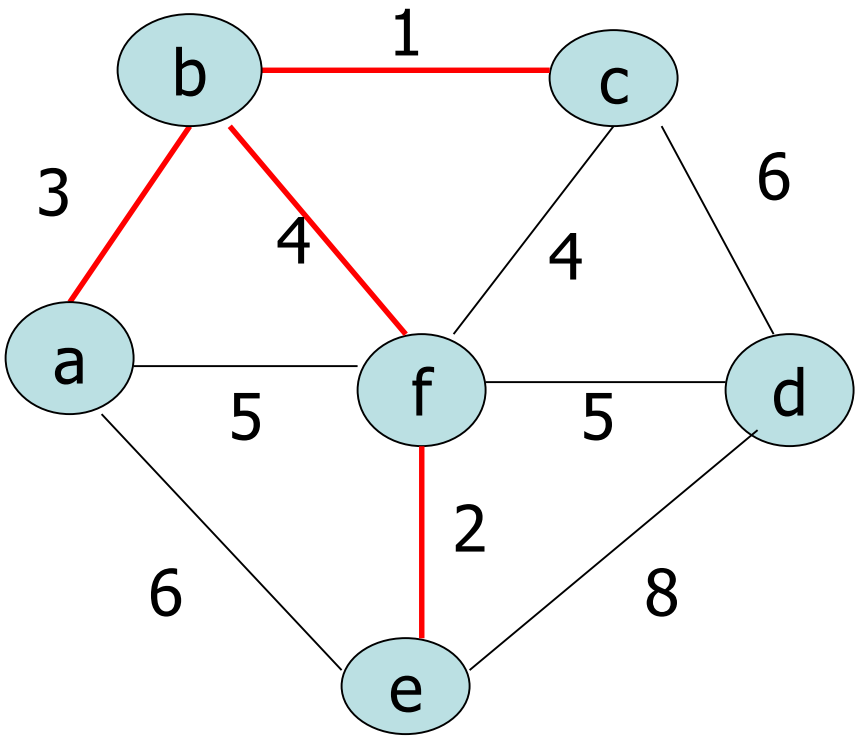


已选边集合为
{ (b,c),(f,e) ,
(a,b) }

因为边w(a,b)为未选边集合中最小权值的边，且加入后，已选边集合中不含回路。

Kruskal算法

bc	1
ef	2
ab	3
bf	4
cf	4
af	5
fd	5
ae	6
cd	6
ed	8



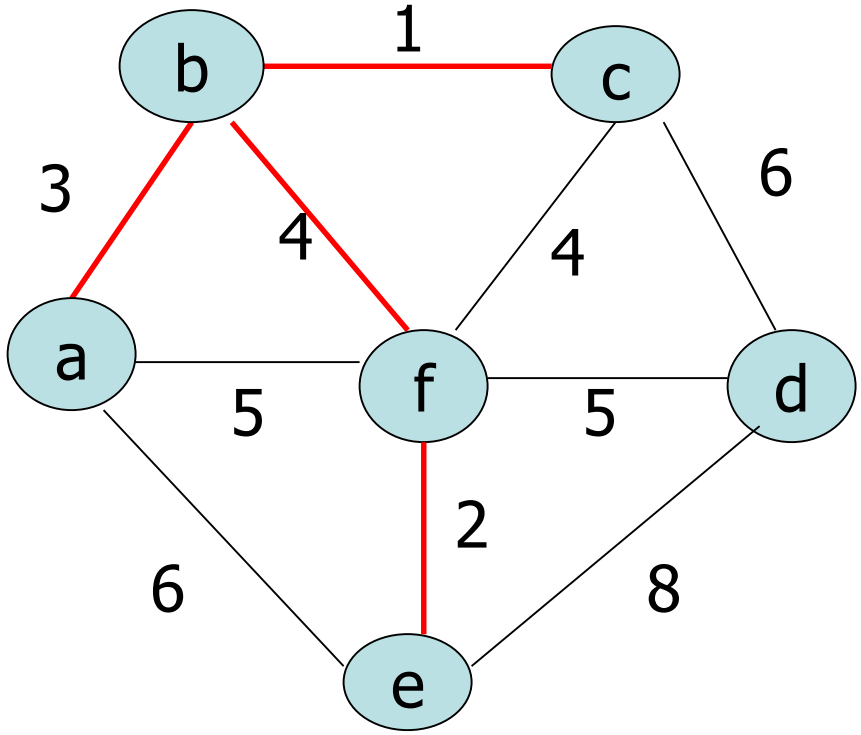
已选边集合为
{ (b,c),(f,e)

(a,b), (b,f) }

因为边w(b,f)为未选边集合中最小权值的边，且加入后，已选边集合中不含回路。

Kruskal算法

bc	1
ef	2
ab	3
bf	4
~~cf~~	~~4~~
~~af~~	~~5~~
fd	5
ae	6
cd	6
ed	8



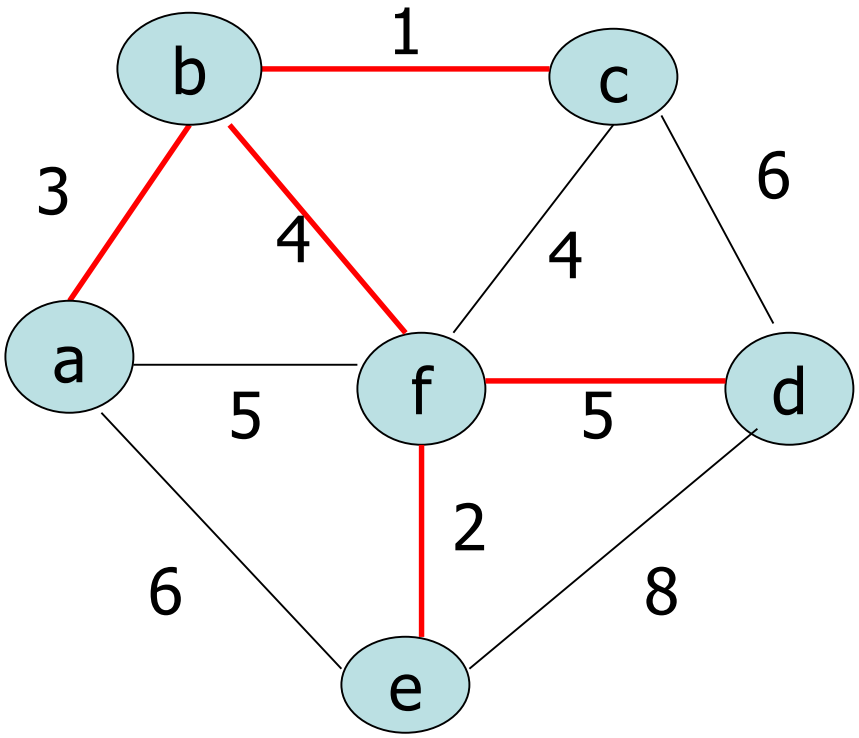
已选边集合为
{ (b,c),(f,e)

(a,b), (b,f) }

因为在加入边w(f,c)时，已选边集合存在回路b->c>f->b，加入边w(a,f)时，已选边集合存在回路a->b>f->a，所以抛弃。

Kruskal算法

bc	1
ef	2
ab	2
bf	4
cf	4
af	5
fd	5
ae	6
cd	6
ed	8



已选边集合为{ (b,c),(f,e),

(a,b), (b,f) ,

(f,d) }，此时已选边集合中元素个数为n-1,算法结束。

因为边w(f,d)为未选边集合中最小权值的边，且加入后，已选边集合中不含回路。

Kruskal算法

算法 Kruskal (G)

//构造最小生成树的 Kruskal 算法

//输入：加权连通图 $G=<V, E>$

//输出：E_T，组成 G 的最小生成树的边的集合

按照边的权重 $w(e_{i_1}) \leqslant \cdots \leqslant w(e_{i_{|E|}})$ 的非递减顺序对集合 E 排序

$E_T \leftarrow \varnothing$； $ecounter \leftarrow 0$ //初始化树中边的顶点集合以及集合的规模

$k \leftarrow 0$ //初始化已处理的边的数量

while $ecounter < |V| - 1$ **do**

$k \leftarrow k+1$

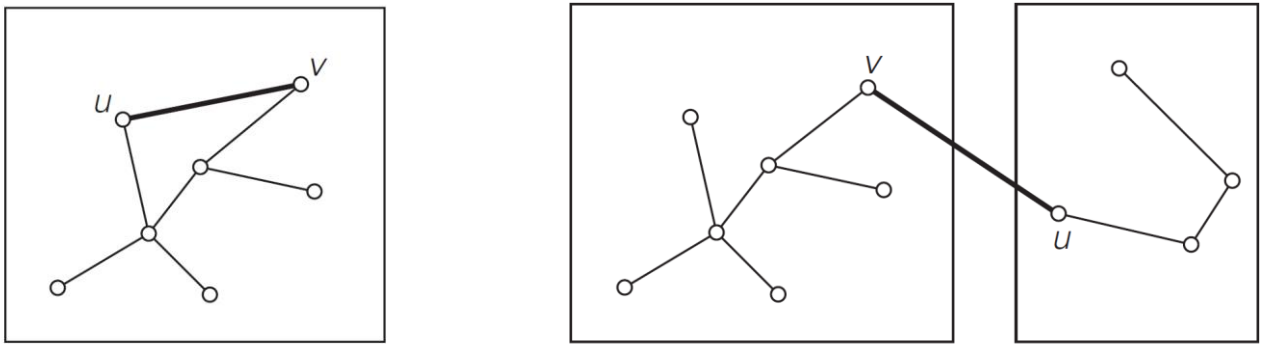
if $E_T \cup \{e_{i_k}\}$ 无回路

$E_T \leftarrow E_T \cup \{e_{i_k}\}$； $ecounter \leftarrow ecounter + 1$

return E_T

关于判定有/无回路

- 每次迭代，取出边(u, v)，并找到包含顶点u和v的树，如果它们不是同一棵树，则加入边(u, v)不会形成回路，且该边会把两棵树连成一棵更大的树



(a) (b)

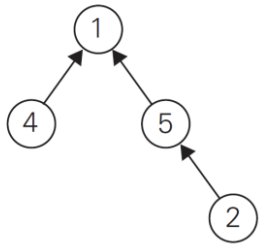
- 相关实现算法：并查(union-find)算法

并查算法

- 对于一个n元素集合S的子集做一系列求并集和查找的操作
- 基本操作
  - makeset(x)：生成一个单元素的集合{x}
  - find(x)：查找并返回一个包含x的子集
  - Union(x,y)：构造分别包含x和y的不相交子集S_x和S_y的并集，并把它加到子集的集合中，以代替被删除后的S_x和S_y
- 举例：S={1, 2, 3, 4, 5, 6}
  - makeset(i)：{1}, {2}, {3}, {4}, {5}, {6}
  - union(1,4)和union(5,2): {1, 4}, {5, 2}, {3}, {6}
  - union(4,5)和union(3,6): {1, 4, 5, 2}, {3, 6}

并查算法

- 快速求并
  - 用一棵有根树来表示每个子集
    - 树中的每个节点代表一个元素
    - 树根为该子集的代表（如每个子集的最小元素为代表）
    - 树中的边由子女指向父母节点



集合{1，4，5，2}

  - makeset(x)：创建一棵单节点的树；$\Theta(1)$ 时间复杂度
  - find(x)：沿着一条指针链，从包含x的节点开始找到树的根，并返回根元素；$O(n)$时间复杂度
  - union(x,y)：把y树的根附加到x树的根；$\Theta(1)$ 复杂度

用来表示子集{1, 4, 5, 2}和{3, 6}的森林　　union(5, 6)的结果

目录

- 最小生成树的构造
  - Prim算法
  - Kruskal算法
- 求解加权图最短路径
  - Dijkstra算法
- 数据压缩
  - 哈夫曼树及编码

Dijkstra算法

- 单起点最短路径问题：加权连通图的某个顶点作为起点(源点)，计算源点到所有其它点之间的最短路径

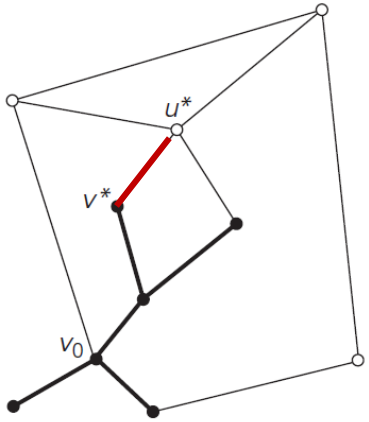
埃德斯加·狄克斯特拉 (Edsger W.Dijkstra**)** ：1972年获得图灵奖，1974年获AFIPS Harry Goode Memorial Award、1989年获ACM SIGCSE计算机科学教育教学杰出贡献奖。狄克斯特拉因最早指出“Goto是有害的”以及首创结构化程序设计而闻名于世。事实上，他对计算机科学的贡献并不仅仅限于程序设计技术。在算法和算法理论、编译器、操作系统诸多方面，狄克斯特拉都有许多创造，作出了杰出贡献。1983年，ACM为纪念Communications of ACM创刊25周年，评选出从1958—1982年的四分之一世纪中在该杂志上发表的25篇有里程碑意义的论文，每年一篇，狄克斯特拉一人就有两篇入选，是仅有的这样的两位学者之一(另一位是英国学者C.A.R **Hoare**，也是图灵奖、计算机先驱奖获得者)。

Dijkstra算法

- 基本思想：由近而远，依次寻找离源点最近的顶点
  - i=1次迭代：根据权重矩阵，找到距离源点最短路径长度及相应的顶点；由源点、该顶点及源点到该顶点的路径构成一棵子树T_1
  - i∈ $[2:n]$次迭代：从与T_{i-1}的顶点相邻的顶点集合（边缘顶点）中找到与源点距离最近的顶点，并将该顶点（如图中的u^*）加入子树中，得T_i
    - 确定加入顶点u^*后，还需以下两个操作
      - 把u^*从边缘顶点集合移到树顶点
      - 更新每个边缘顶点u的标记，如果

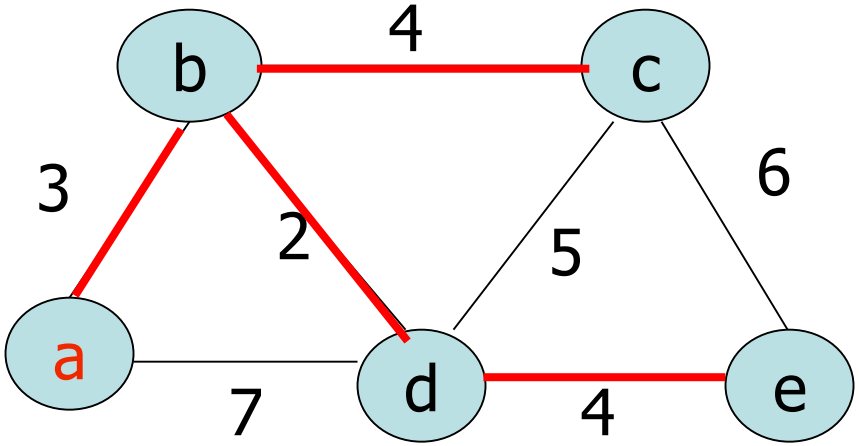
$$d_{u^*} + w(u^*, u) < d_u$$

则u的标记更新为$u(u^*, d_{u^*} + w(u^*, u))$



Dijkstra算法

树中顶点	边缘顶点
a(-,-)	b(a,3), c(-, ∞), d(a,7), e(-, ∞)
b(a,3)	c(b,3+4) , d(b,3+2) , e(-,∞)
d(b,5)	c(b,3+4), e(d,5+4)
c(b,7)	e(d,5+4)
e(d,9)	



每个顶点的标记：
1. 该顶点的父节点
2. 从源点到该点的最短路径长度

Dijkstra算法

算法 Dijkstra (G, s)

//单起点最短路径的 Dijkstra 算法

//输入：具非负权重加权连通图 $G=<V, E>$以及它的顶点 s

//输出：对于 V 中的每个顶点 v 来说，从 s 到 v 的最短路径的长度 d_v

//以及路径上的倒数第二个顶点 p_v

Initialize(Q) //将顶点优先队列初始化为空

for V 中每一个顶点 v

$d_v \leftarrow \infty$; $p_v \leftarrow$ **null**

Insert(Q, v, d_v) //初始化优先队列中顶点的优先级

$d_s \leftarrow 0$; Decrease(Q, s, d_s) //将 s 的优先级更新为 d_s

$V_T \leftarrow \varnothing$

for $i \leftarrow 0$ **to** $|V|-1$ **do**

$u^* \leftarrow$ DeleteMin(Q) //删除优先级最小的元素

$V_T \leftarrow V_T \cup \{u^*\}$

for $V-V_T$ 中每一个和 u^* 相邻的顶点 u **do**

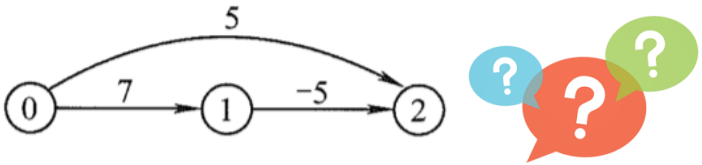
if $d_{u^*}+w(u^*,u)<d_u$

$d_u \leftarrow d_{u^*}+w(u^*,u)$; $p_u \leftarrow u^*$

Decrease(Q, u, d_u)

Dijkstra算法

- 既适用于无向图又适用于有向图，但权重必须非负



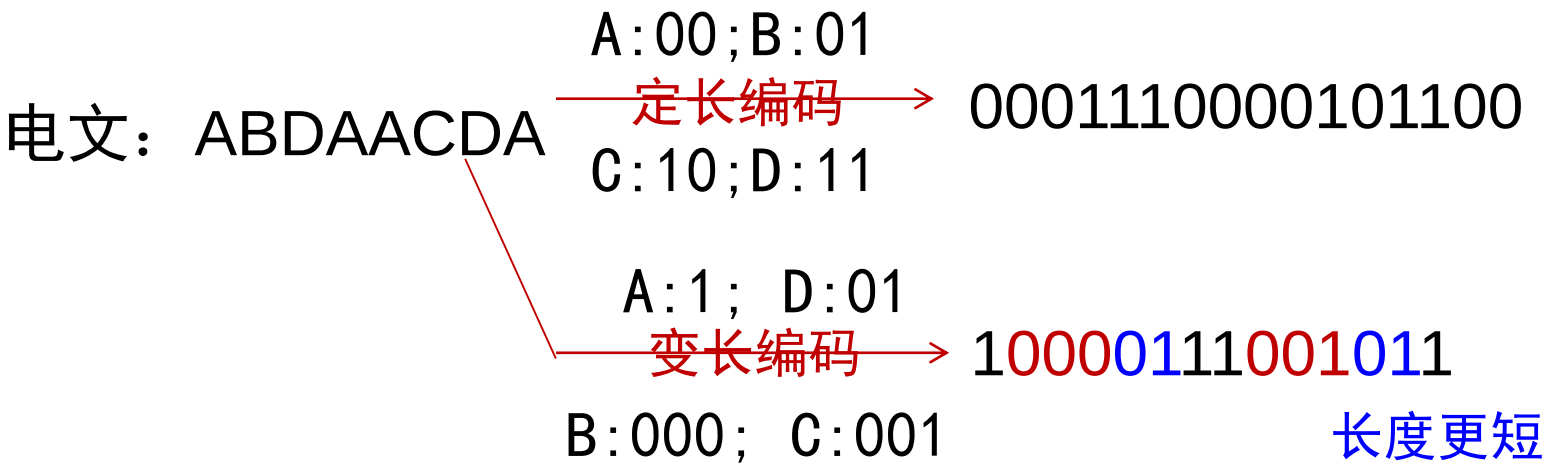
- 效率：
  - 当图用权重矩阵表示，优先队列用无序数组实现时，为$O(|V|^2)$
  - 当图用邻接链表表示，优先队列用最小堆实现时，为$O(|E|\log|V|)$
- Dijkstra算法 vs Prim算法
  - 两者顶点的标记结构类似
  - 但前者比较的是路径长度，因此必须把边的权重值累加，而后者则是直接比较给定的边权重

目录

- 最小生成树的构造
  - Prim算法
  - Kruskal算法
- 求解加权图最短路径
  - Dijkstra算法
- 数据压缩
  - 哈夫曼树及编码

哈夫曼树及编码

- 字符编码
  - 将文本中的每个字符编码成一个比特串（码字）
  - 假定文本中不同字符个数为n，则
    - 定长编码：至少需要长度为$m \geq log_2 n$的码字（为什么）
    - 变长编码：不同字符的码字长度不同，但不影响解码

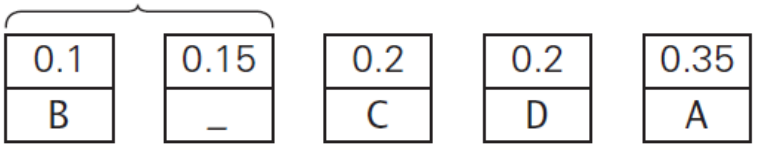


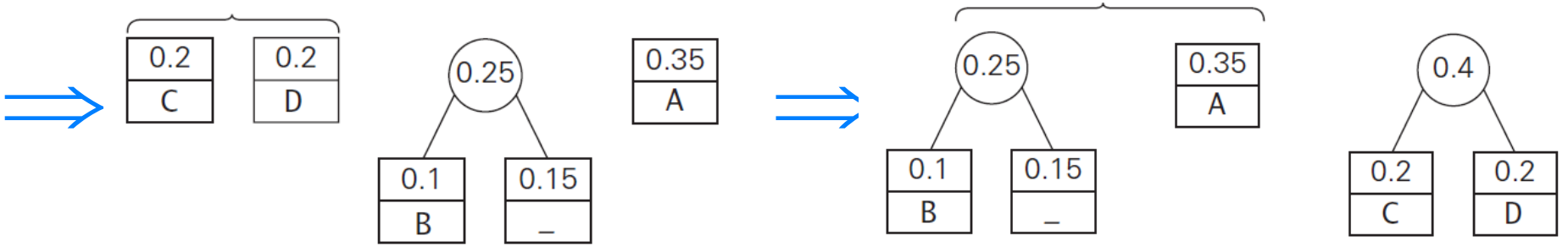
哈夫曼编码

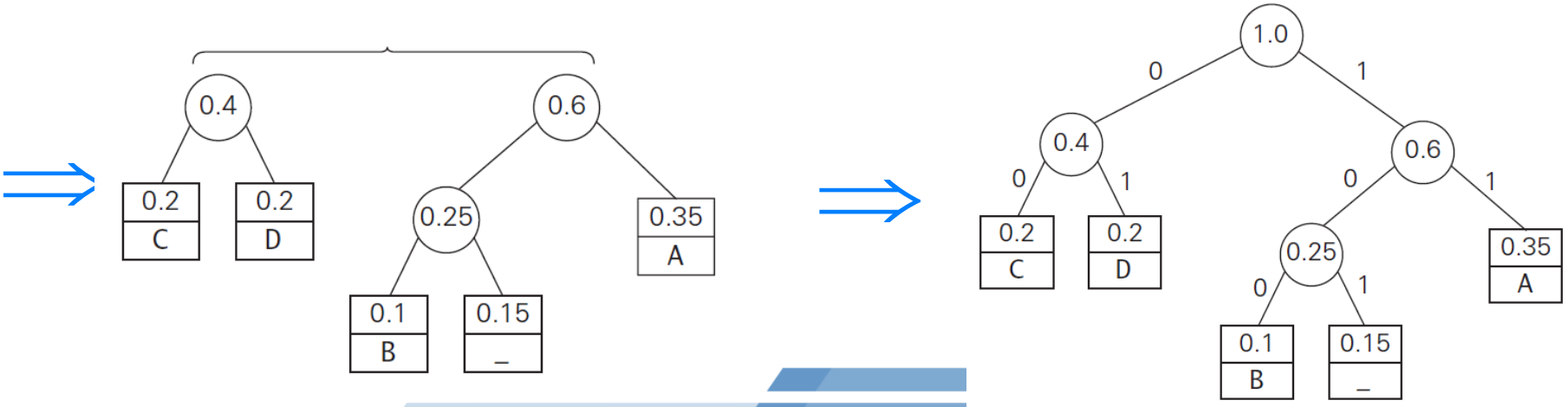
- 基本思想：假定字符出现的概率已知，则可将短码字分配给高频字符，将长码字分配给低频字符
- 算法（哈夫曼树的构造）：
  - 第一步：初始化n个单节点的树，并为它们标上字母表中的字符，并把每个字符的概率记录在树的根中，用来指出树的权重
  - 第二步：找到两棵权重最小的树，把它们作为新树中的左右子树，并把其权重之和作为新的权重记录在新树的根中。重复上述步骤，直到只剩一棵单独的树
- 哈夫曼编码：对于所构造哈夫曼树，将所有的左向边标记为0，右向边标记为1，则树根到叶子字符的路径对应的比特串即为该字符的码字

哈夫曼编码

- 举例


0.1
B
0.15
_
0.2
C
0.2
D
0.35
A


0.2
C
0.2
D
0.25
0.1
B
0.15
_
0.35
A
0.25
0.1
B
0.15
_
0.35
A
0.4
0.2
C
0.2
D


0.4
0.2
C
0.2
D
0.6
0.25
0.1
B
0.15
_
0.35
A
1.0
0
1
0.4
0.6
0
1
0
1
0.2
C
0.2
D
0.25
0.35
A
0
1
0.1
B
0.15
_

哈夫曼编码

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

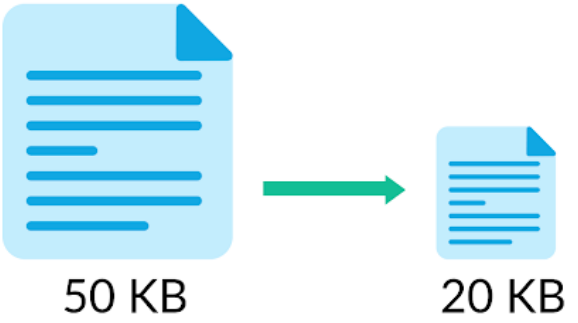
- 上述例子的哈夫曼编码平均码长：

$$2 \cdot 0.35 + 3 \cdot 0.1 + 2 \cdot 0.2 + 2 \cdot 0.2 + 3 \cdot 0.15 = 2.25.$$

  - 定长编码至少需要3比特
  - 哈夫编码压缩率：$(3 - 2.25)/3 \cdot 100\% = 25\%.$

哈夫曼编码

- 是一种重要的文件压缩方法：先扫描给定的文本，统计字符出现的频率，再构造哈夫曼树进行编码
  - 但需要将编码树的信息包含在编码文本中，以便解码
- 可能的改进
  - 动态哈夫曼编码：每次从文本中读入一个字符就更新编码树
  - Lempel-Ziv编码：不是对单个字符编码，而是对一串字符编码



哈夫曼编码压缩的文件类型

.mpq, .ace, .jpeg, .png, .zip

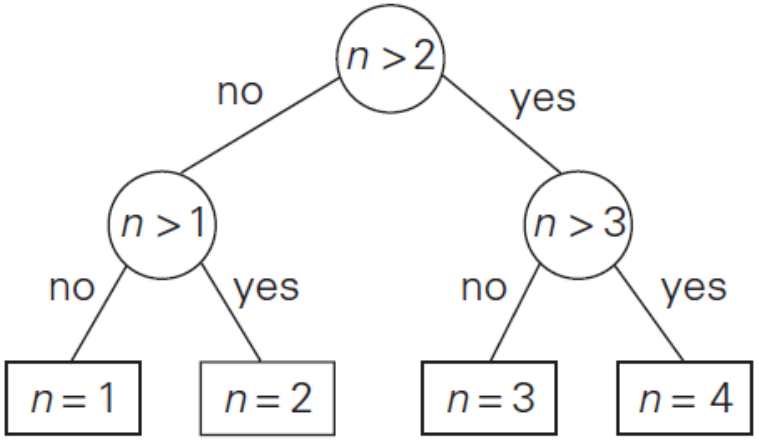
Lempel-Ziv压缩的文件类型

.lzma, .lzo, .lz, .lzh

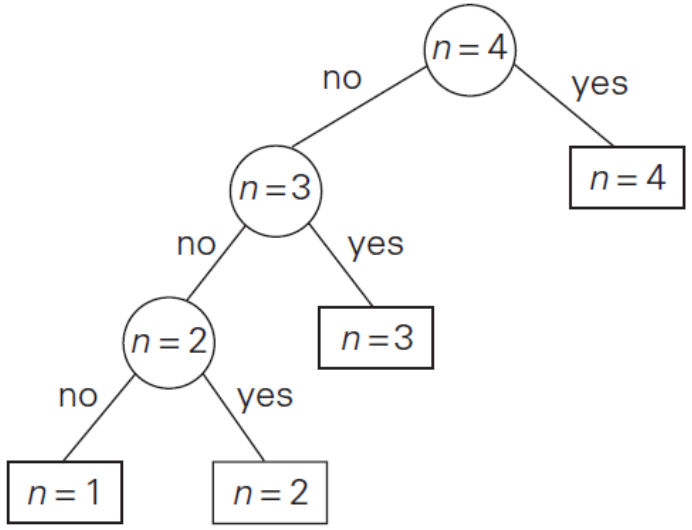
RAR 是一种专有的文件格式，用于文件的压缩、归档与打包。RAR 的全名是：Roshal Archive（即"罗谢尔的归档"之意），其开发者是尤金·罗谢尔（Eugene Roshal）。首个公开版本 RAR 1.3 发布于1993年。Roshal 最初编写了 RAR 的 Dos 版本的程序用以编码和解码文件，后来该程序被移植到其他的平台，比较著名的是 Windows 平台上移植版的 Win RAR。Eugene Roshal 后来公开了解码程序的源代码，但是编码程序仍然是私有的。RAR 因为其独特的压缩算法，基本可以做到无损压缩，而且还能够满足较高的压缩比，同时保证一定的压缩速度。但是RAR压缩算法也存在一定的缺陷，由于RAR文件头需要占用一部分的空间，二档压缩的原始数据量较小，而且压缩的空间较小的情况下，可能会出现压缩后的文件反而比原始文件更大。RAR 文件有较多的冗余记录，主要是考虑到在压缩过程中，压缩数据受损，为了保证无损压缩，会有较多的恢复记录，这些恢复记录也占用了一定的空间。但是分卷压缩是 RAR 非常突出的一个优点，将源文件分割为多个小文件，从而有利于解压出源文件。如果将所有的数据压缩到同一个数据区，就可以大大加大压缩比，但是这种压缩方式在解压其中一个单独的文件时就必须解压同一数据区中位于它之前的所有的文件，不利于文件的单独解压。RAR 拥有成熟的加密算法，2.0 版本以后使用 AES 算法来加密，AES 算法的破解难度比较大，在没有密码的情况下只能采取暴力破解的办法，对于数据的安全性有一定的保证。 [1]

哈夫曼算法的其它应用

- 决策树
  - 比如从1到n整数中猜测一个数，玩家可以提问，并根据回答（是或否）来调整策略。那么，平均意义上，需要多少次提问才可以确定这个数？



n=4：4个数字等概情况下的决策树

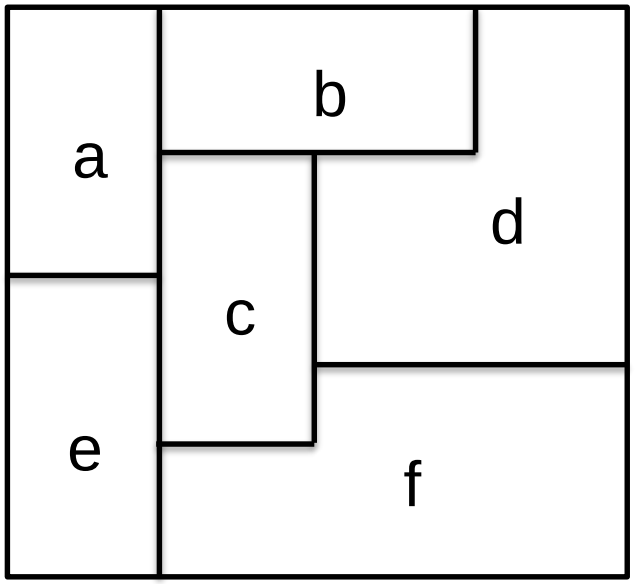


$p_1 = 0.1$，$p_2 = 0.2$，$p_3 = 0.3$，$p_4 = 0.4$

讨论：图着色问题

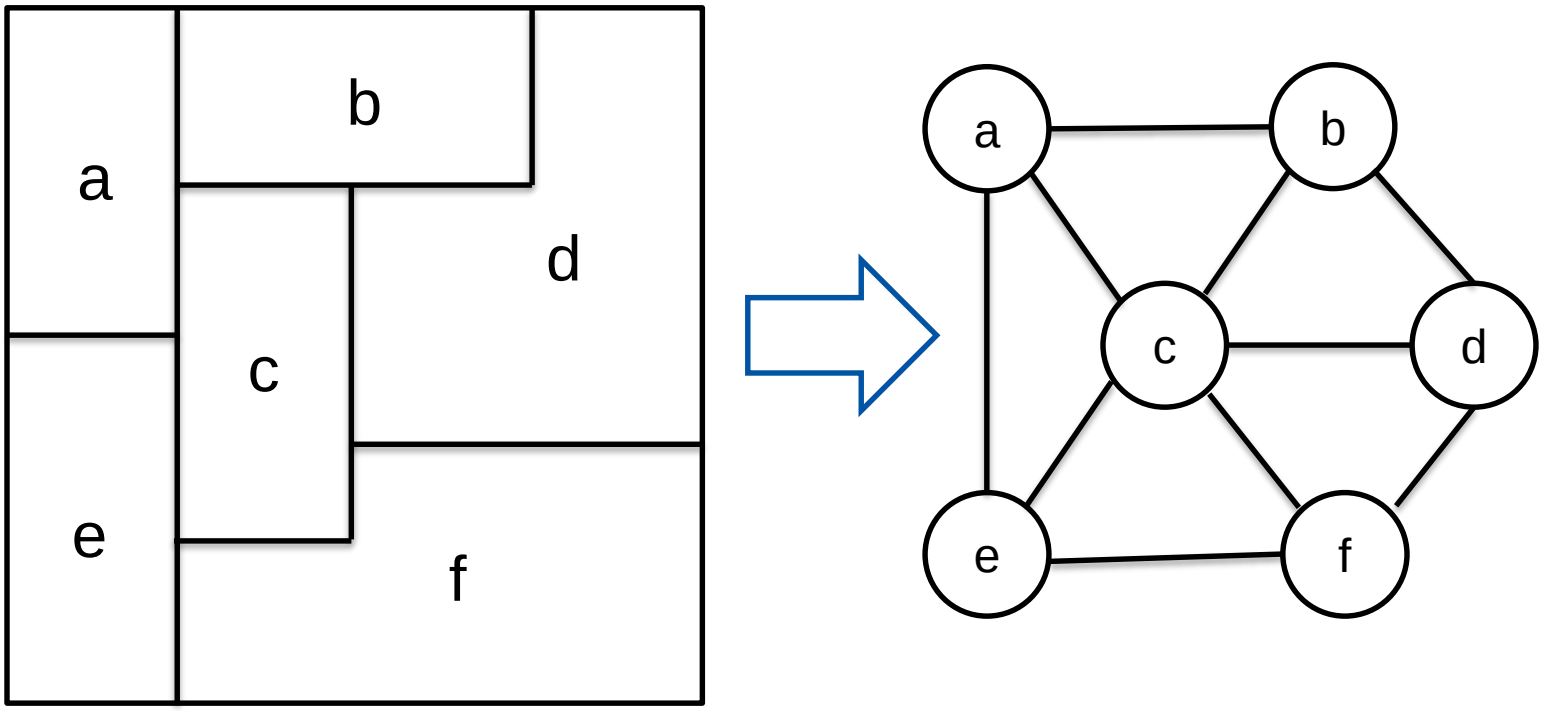
- 考虑以下地图：

  请设计一个贪婪算法，使用最少种类的颜色对该地图着色，并保证相邻区域颜色不同



讨论：图着色问题

- 解决思路：
  - 用无向图来表征不同区域的邻接关系



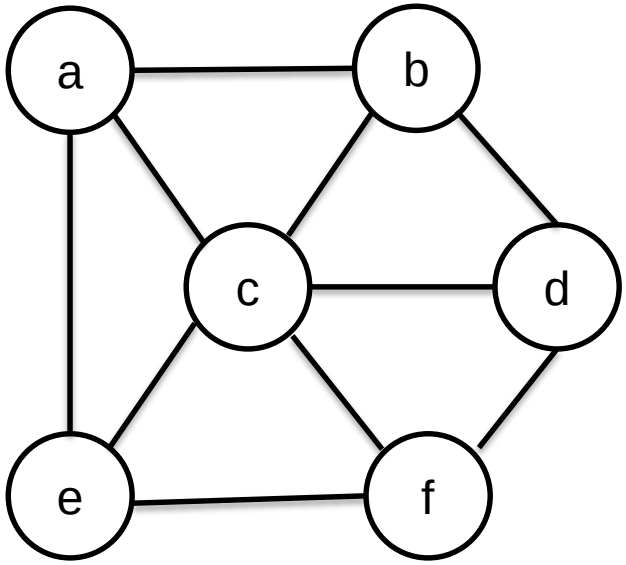
讨论：图着色问题

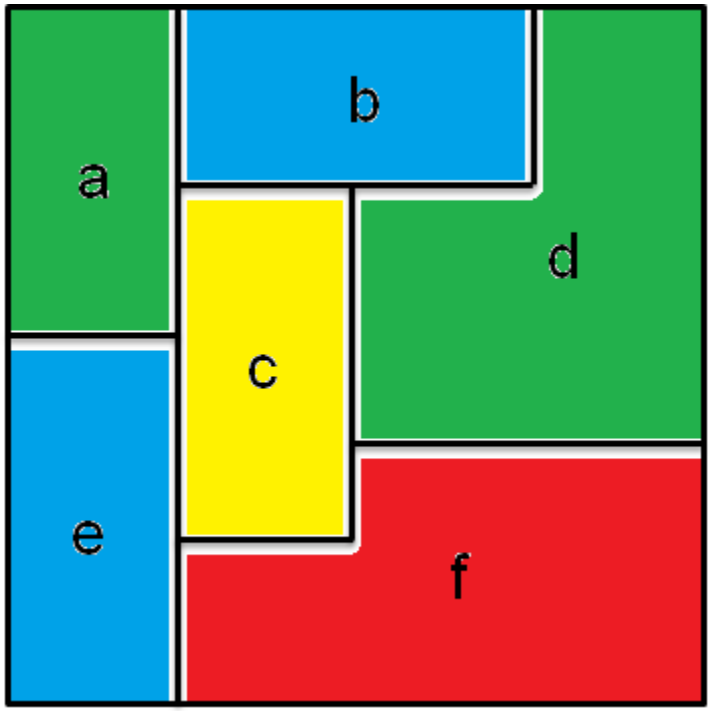
- 解决思路：
  - 用无向图来表征不同区域的邻接关系
  - 贪婪思想：用同一种颜色着尽量多的顶点（区域）

```
graphcoloring（A[0,n-1]）
//用贪心算法来解决
//c表示最少用的填充颜色个数
c=1
for i = 0 to n-1 do
  if(可以在已有的c种颜色中找到符合条件的颜色填充第i个区域)
    i++
  else
    c++;
    i++
c既是最少需要的颜色个数
```

讨论：图着色问题

a → 1
b → 2
c → 3
d → 1
e → 2
f → 4

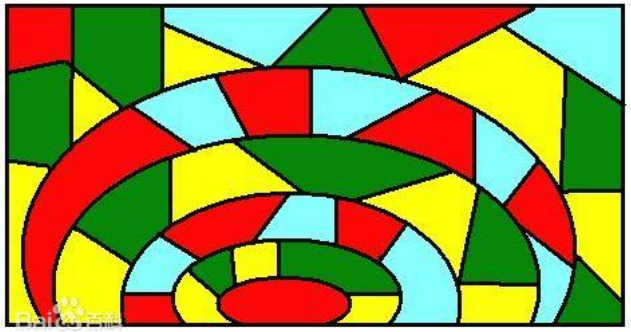




四色定理

- 如果在平面上划出一些邻接的有限区域，那么可以用四种颜色来给这些区域着色，使得每两个邻接区域所着的颜色都不一样

- 也就是说，任何一张地图只用四种颜色就能使具有共同边界的国家着上不同的颜色



课后作业

章 X	节 X. Y	课后作业题 Z	思考题 Z
9	9.1	9	6,8
	9.2	1,6	5,11
	9.3	2a	
	9.4	1	10

注：只需上交“课后作业题”；以“学号姓名_chX. pdf ”规范命名，提交到“学在浙大”指定文件夹。DDL：2024年4月30日

大数据处理实验内容

- 需要基于华为云平台
- 请大家在4月23日前
  - 注册华为云账号
  - 注册完后，查询华为云账号ID，返回ID信息给助教（石滨溥同学），便于后续发放代金券

步骤 2 · 如何查询自己的华为云账号 ID。

登录华为云账号，访问以下链接，红框处即为华为云账号 ID。

https://console.huaweicloud.com/iam/?agencyId=0bc8d306f880f2c21f28c01b3710deb1&region=cn-north-1&locale=zh-cn#/mine/apiCredential

注意：华为云账号 ID 是在华为云控制台的“我的凭证”-“API 凭证”处查看，由字母和数字组成，无特殊符号，唯一且不可更改。