

算法分析与设计

贪心算法

主要内容

- 贪心算法的基本思想
- 活动安排问题
- 贪心策略的基本要素
- 贪心算法实例——单源最短路径
- 贪心算法实例——最小生成树
- Huffman编码

贪心算法的基本思想

- 适用于求解最优化问题的算法往往包含一系列步骤，每一步都有一组选择
- 贪心算法总是作出在当前看来是最好的选择
- 贪心算法并不从整体最优上加以考虑，它所作出的选择只是在某种意义上的局部最优选择

贪心算法的基本思想

- 贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解
- 在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似
- 与动态规划方法相比，贪心算法更简单，更直接

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，如报告厅等
- 而在同一时间内只有一个活动能使用这一资源
- 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 $s_i < f_i$
- 如果选择了活动 i ，则它在半开时间区间 $[s_i, f_i)$ 内占用资源，若区间 $[s_i, f_i)$ 与区间 $[s_j, f_j)$ 不相交，则称活动 i 与活动 j 是相容的
- 也就是说，当 $s_i \geq f_j$ 或 $s_j \geq f_i$ 时，活动 i 与活动 j 相容
- 目标：要在所给的活动集合中选出最大的相容活动子集合

活动安排问题

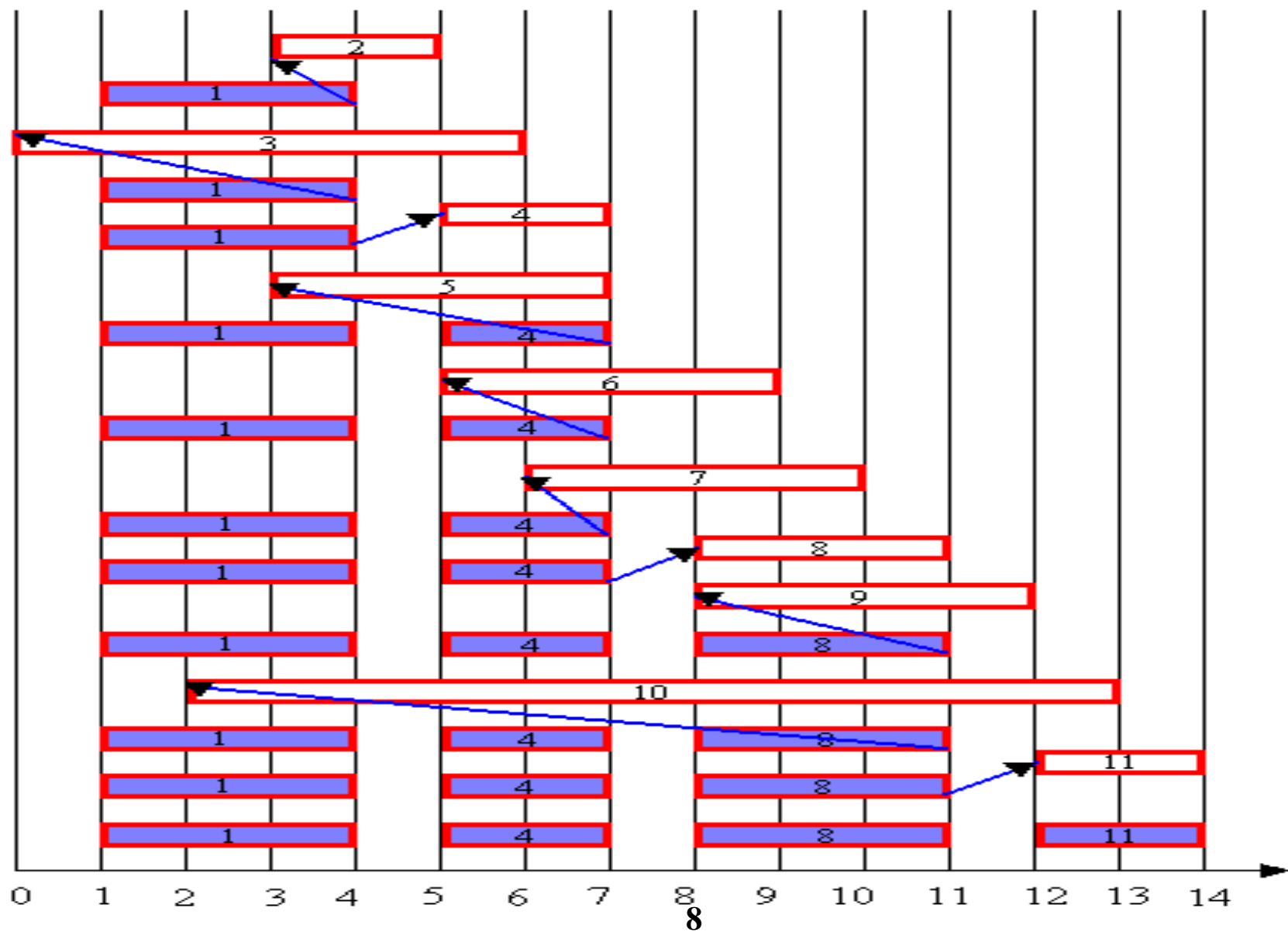
●贪心策略求解方案

- 将活动按照结束时间的增序 $f_1 \leq f_2 \leq \dots \leq f_n$ 排列
- 一开始选择活动1，然后依次检查后面的活动 i 是否与前面已选择的活动相容，若相容，则将活动 i 加入到已选择的活动集合A中；若不相容，则继续检查下一活动与已选择活动的相容性
- 由于活动已按结束时间排序，设刚加入已选择活动集合的活动是 j ，因此只需检查活动 i 的是否与 j 相容；也就是检查是否满足 $s_i \geq f_j$ ，若满足，则活动 i 与 j 相容
- 贪心体现在总是选择具有最早完成时间的相容活动

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已按结束时间的增序排列：

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

GREEDY-ACTIVITY-SELECTOR(s, f)

1 $n \leftarrow \text{length}[s]$

2 $A \leftarrow \{\text{activity } 1\}$

3 $j \leftarrow 1$

4 for $i \leftarrow 2$ to n

5 do if $s_i \geq f_j$

6 then $A \leftarrow A \cup \{\text{activity } i\}$

7 $j \leftarrow i$

8 return A

活动安排问题

●复杂度分析

- 如果已经排序，算法的时间复杂度为 $\Theta(n)$
- 如果事先没有按照结束时间增序排列，排序需 $O(n\lg n)$

●贪心算法可以获得该问题的整体最优解

- 可证明：**1.活动安排问题有一个最优解以贪心选择开始；**
- 2.做出贪心选择之后，原问题简化为比原问题更小的但与原问题形式相同的子问题。**

贪心算法的基本要素

- 贪心算法通过做一系列的选择来给出某一问题的最优解。它所作出的每一个选择当前状态下的最好选择（局部），即贪心选择
- 这种贪心选择并不总能产生最优解，但对于一些问题，比如活动安排问题，可以给出最优解

贪心算法的基本要素

●可以根据下列步骤设计贪心算法

- 将最优化问题转化为这样的一个问题，即先做出选择，再解决剩下的一个子问题
- 证明原问题总有一个最优解是做贪心选择得到的，从而说明贪心选择的安全
- 说明在做出贪心选择之后，子问题的最优解与所作出的贪心选择联合起来，可以得出原问题的一个最优解

贪心算法的基本要素

- 许多可以用贪心算法求解的问题，具备以下两个性质
 - 贪心选择性质
 - 最优子结构性

贪心算法的基本要素

●贪心选择性质

- 是指所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
- 这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别
- 动态规划算法通常以自底向上的方式解各子问题
- 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

●最优子结构性质

- 当一个问题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

贪心算法的基本要素

●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？
- 限制：在选择装入背包的物品时，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

贪心算法的基本要素

●背包问题

- 与0-1背包问题类似，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， $1 \leq i \leq n$

贪心算法的基本要素

●0-1背包问题和背包问题的求解分析

- 这2类问题都具有最优子结构性质，极为相似
- 但背包问题可以用贪心算法求解
- 而0-1背包问题却不能用贪心算法求解

贪心算法的基本要素

●用贪心算法解背包问题的基本步骤

- 首先计算每种物品单位重量的价值 V_i/W_i
- 然后，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

贪心算法的基本要素

●对于0-1背包问题，贪心选择不能得到最优解

- 因为在这种情况下，它无法保证最终能将背包装满，部分闲置的背包空间使单位重量背包空间的价值降低
- 事实上，在考虑0-1背包问题时，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（子问题重叠）
- 这正是该问题可用动态规划算法求解的重要特征

最优装载问题

●问题

- 有一批集装箱要装上一艘载重量为 c 的轮船。其中集装箱 i 的重量为 W_i 。最优装载问题要求确定在装载体积不受限制的情况下，将尽可能多的集装箱装上轮船

●算法

- 最优装载问题可用贪心算法求解
- 采用重量最轻者先装的贪心选择策略，可产生最优装载问题的最优解

贪心算法的重要实例

●几个重要的图算法

- 单源最短路径
 - Dijkstra算法
- 最小生成树
 - Prim算法
 - Kruskal算法

单源最短路径 Dijkstra算法

●单源最短路径问题

- 给定带权有向图 $G=(V,E)$ ，其中每条边的权是非负实数
- 给定 V 中的一个顶点，称为源
- 要求计算从源到所有其他各顶点的最短路长度（这里路的长度是指路上各边权之和）

单源最短路径 Dijkstra算法

● 基本思想

- 设置顶点集合S，初始时，S中仅含有源，此后不断作贪心选择来扩充这个集合
- 一个顶点属于集合S当且仅当从源到该顶点的最短路径长度已知
- 设u是G的某一个顶点，把从源到u且中间只经过S中顶点的路称为从源到u的特殊路径，并用数组dist记录当前每个顶点所对应的最短特殊路径长度
- Dijkstra算法每次从V-S中取出具有最短特殊路长度的顶点u，将u添加到S中，同时对数组dist作必要的修改，检查 $\text{dist}(u) + [u, j]$ 与 $\text{dist}[j]$ 的大小，若 $\text{dist}(u) + [u, j]$ 较小，则更新
- 一旦S包含了所有V中顶点，dist就记录了从源到所有其他顶点之间的最短路径长度

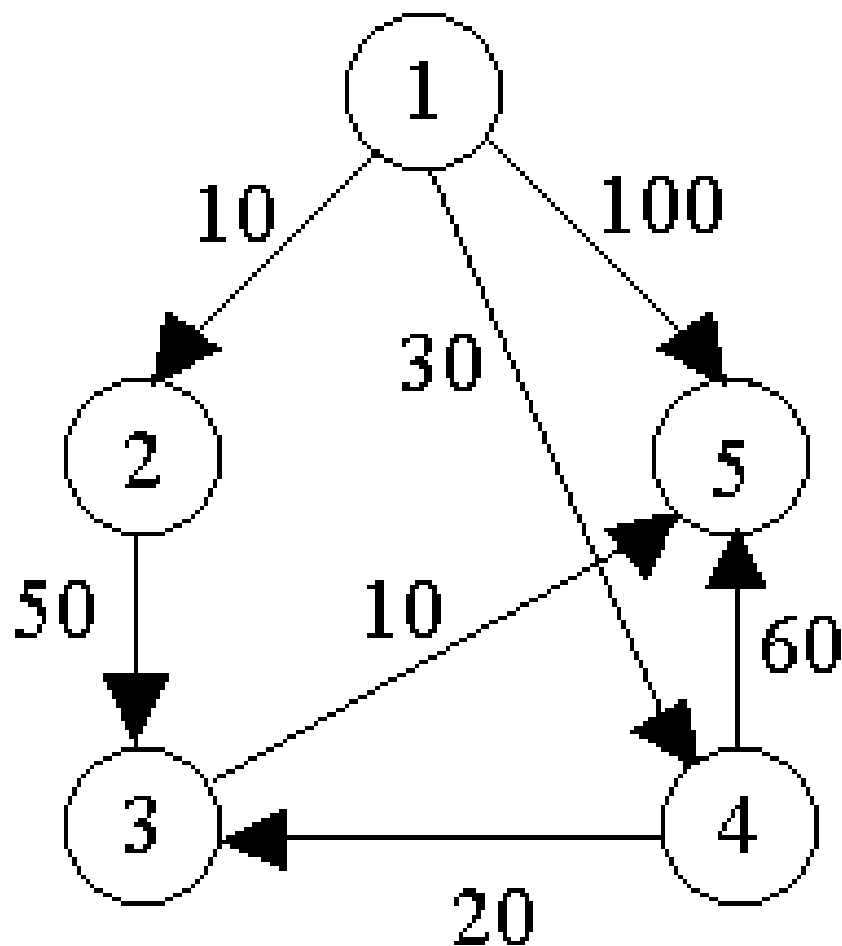
单源最短路径 Dijkstra算法

●贪心策略

- 因为Dijkstra算法总是在 $V-S$ 中选择“最近”（具有最短特殊路长度）的顶点，所以说Dijkstra算法使用了贪心策略

单源最短路径 Dijkstra算法

●实例，源点1



单源最短路径 Dijkstra算法

●实例：Dijkstra算法的计算过程：

迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

单源最短路径 Dijkstra算法

- 分析

- 贪心策略体现在对V-S中的点的选择上

- 时间复杂度

- 对于具有n个顶点和e条边的带权有向图，如果用带权邻接矩阵表示这个图， $O(n^2)$

最小生成树

- 设 $G = (V, E)$ 是无向连通带权图，即一个网络
- E 中每条边 (v, w) 的权为 $c[v][w]$
- 如果 G 的子图 G' 是一棵包含 G 的所有顶点的树，则称 G' 为 G 的生成树
- 生成树上各边权的总和为该生成树的耗费
- 在 G 的所有生成树中，耗费最小的生成树称为 G 的最小生成树

最小生成树

- 网络的最小生成树在实际中有广泛应用
- 例如，在设计通信网络时，用图的顶点表示城市，用边 (v,w) 的权 $c[v][w]$ 表示建立城市 v 和城市 w 之间的通信线路所需的费用
- 最小生成树就给出了建立通信网络的最经济的方案

最小生成树

●性质

➤ 设 $G=(V,E)$ 是连通带权图， U 是 V 的真子集。如果 $(u,v) \in E$ ，且 $u \in U$ ， $v \in V-U$ ，且在所有这样的边中， (u,v) 的权 $c[u][v]$ 最小，那么一定存在 G 的一棵最小生成树，它以 (u,v) 为其中一条边

● 用贪心算法设计策略可以设计出构造最小生成树的有效算法

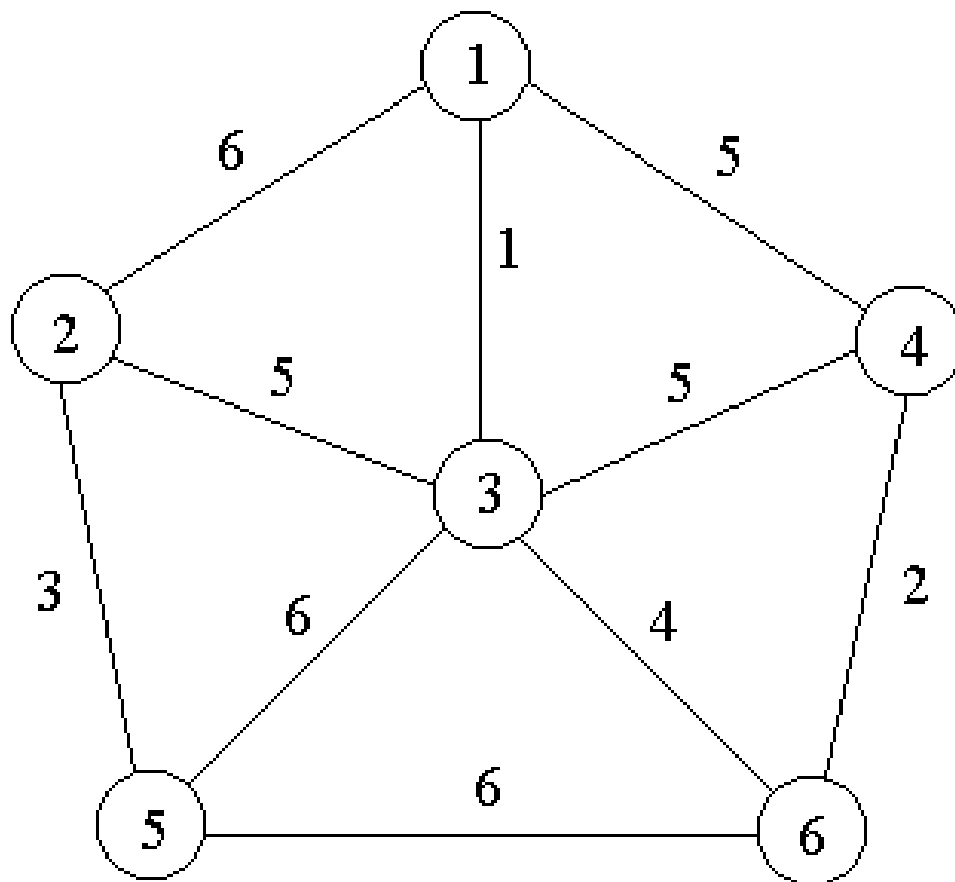
● 构造最小生成树的Prim算法和Kruskal算法用到了以上性质

最小生成树 Prim算法

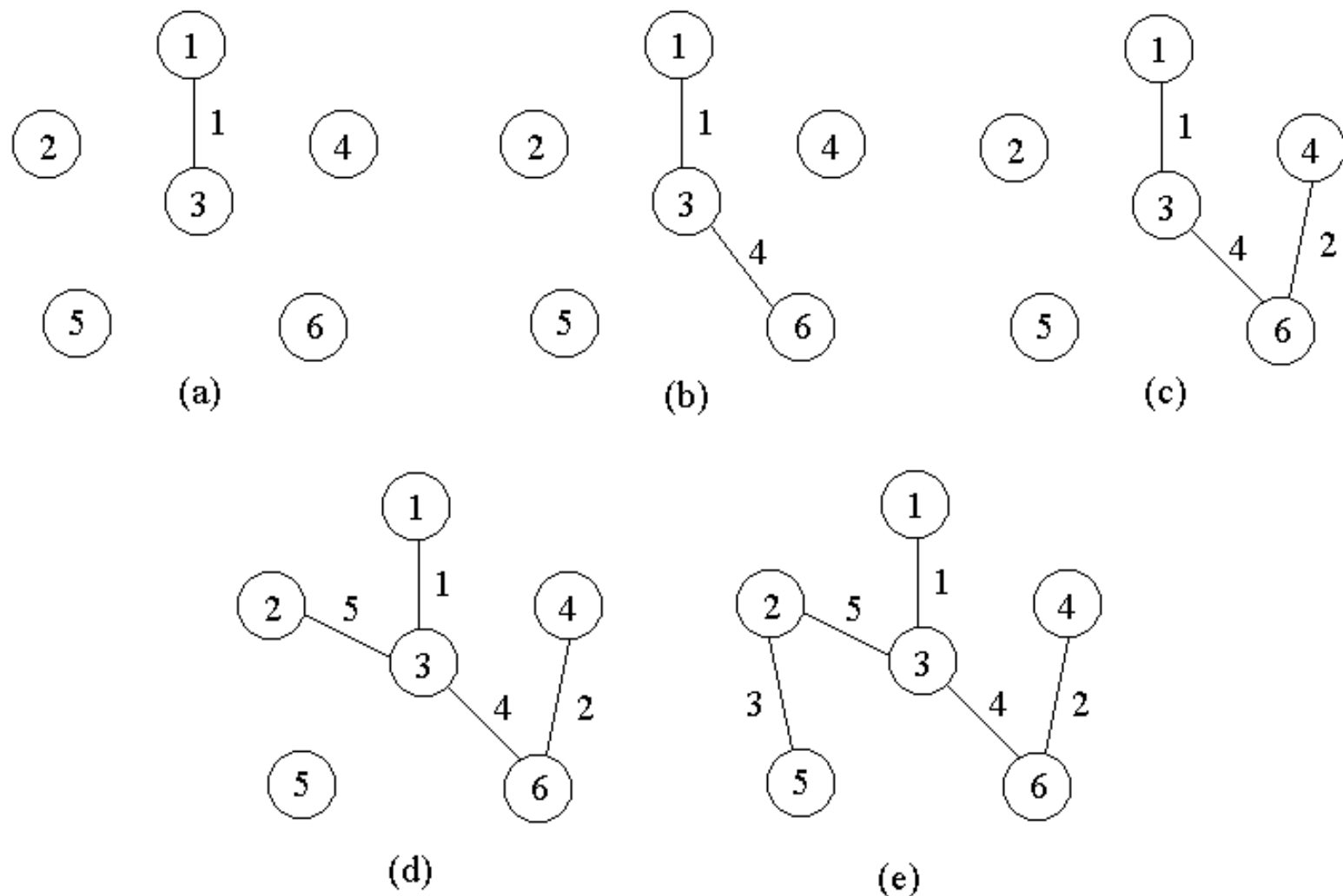
- 设 $G=(V,E)$ 是连通带权图, $V=\{1,2,\dots,n\}$
- Prim算法的基本思想
 - 首先置 $S=\{1\}$
 - 然后, 只要 S 是 V 的真子集, 就作如下的贪心选择:
选取满足条件 $i \in S, j \in V-S$, 且 $c[i][j]$ 最小的边, 将顶点 j 添加到 S 中
 - 这个过程一直进行到 $S=V$ 时为止
 - 在这个过程中选取到的所有边恰好构成 G 的一棵最小生成树

最小生成树 Prim算法

●Prim算法求解实例



最小生成树 Prim算法



最小生成树 Prim算法

- 在上述Prim算法中，还应当考虑如何有效地找出满足条件 $i \in S, j \in V-S$ ，且权 $c[i][j]$ 最小的边 (i,j) ，实现这个目的的较简单的办法是设置2个数组closest和lowcost
- closest[j]是j在S中的邻接顶点，它与j在S中的其它邻接顶点k相比有 $c[j][closest[j]] \leq c[j][k]$
- lowcost[j]的值就是 $c[j][closest[j]]$

最小生成树

- 在Prim算法执行过程中，先找出V-S中使lowcost值最小的顶点j，然后根据数组closest选取边(j,closest[j])，最后将j添加到S中，并对closest和lowcost作必要的修改
- 用这个办法实现的Prim算法所需的时间复杂度为 $O(n^2)$

最小生成树 Kruskal算法

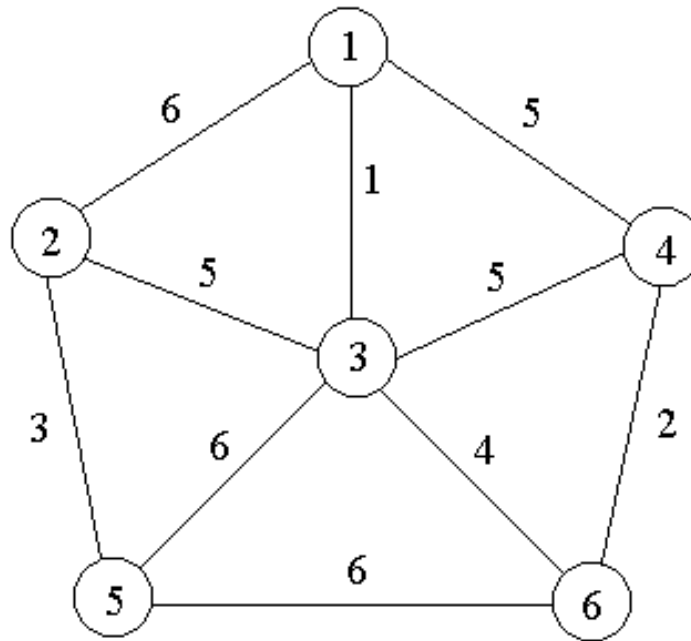
●Kruskal算法的基本思想是

- 首先将 G 的 n 个顶点看成 n 个孤立的连通分支。将所有的边按权值从小到大排序
- 然后从第一条边开始，依边权递增的顺序查看每一条边，并按下述方法连接2个不同的连通分支：当查看到第 k 条边 (v, w) 时，如果端点 v 和 w 分别是当前2个不同的连通分支 T_1 和 T_2 中的顶点时，就用边 (v, w) 将 T_1 和 T_2 连接成一个连通分支，然后继续查看第 $k+1$ 条边；如果端点 v 和 w 在当前的同一个连通分支中，就直接再查看第 $k+1$ 条边
- 这个过程一直进行到只剩下一个连通分支时为止

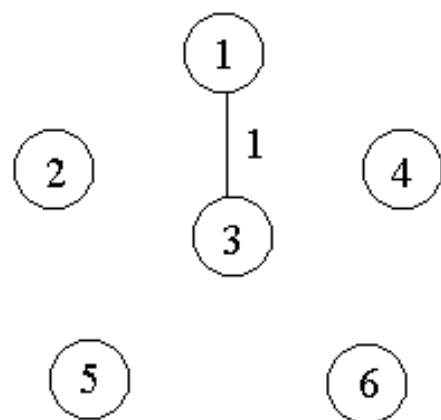
最小生成树

●Kruskal算法的贪心策略

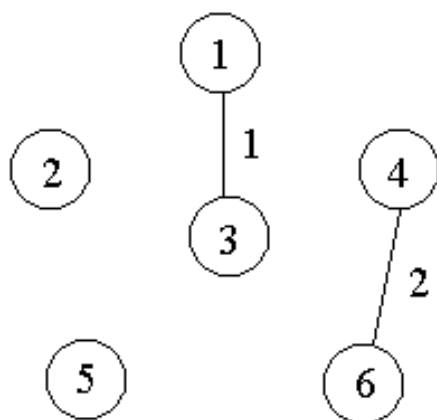
- 将图中的边按权值由小到大排序，由小到大顺序选取各条边，若选某边后不形成回路，则将其保留作为树的一条边



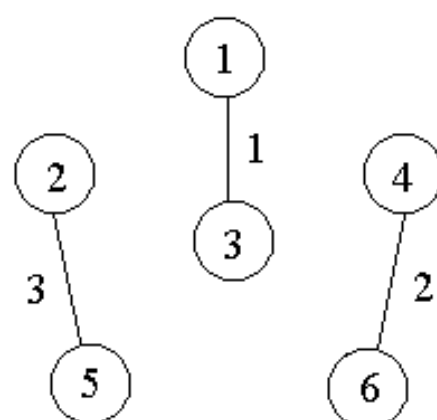
最小生成树



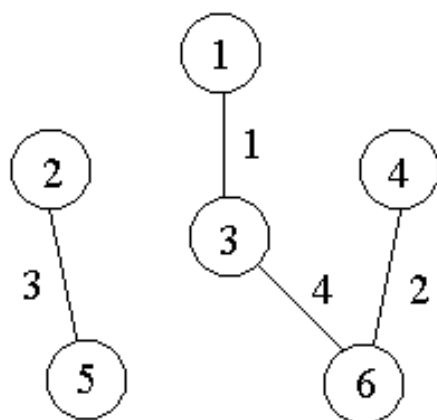
(a)



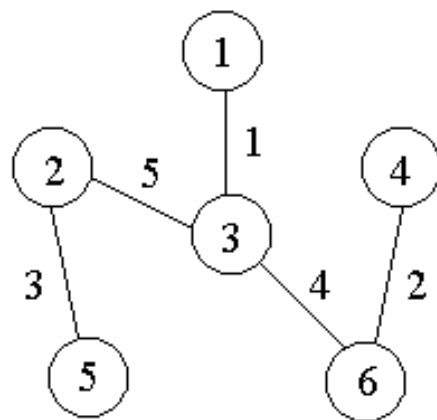
(b)



(c)



(d)



(e)

Huffman编码

●Huffman编码

- 一种可变字长编码(VLC)方式
- Huffman于1952年提出
- 该方法依据字符出现的概率来构造字符编码，一般称为Huffman编码
- 广泛应用于数据文件的压缩

Huffman编码

●前缀码

- 对每一个字符规定一个0,1串作为其代码，并要求任一字符的代码都不是其他字符代码的前缀
- 编码的前缀性质可以使译码方法非常简单
- 表示最优前缀码的二叉树中任一结点都有2个儿子结点
- 平均码长定义为 $B(T) = \sum_{c \in C} f(c)d_T(c)$
- 使平均码长达到最小的前缀码编码方案称为给定编码字符集C的最优前缀码

Huffman编码

●构造方式

- Huffman提出构造最优前缀码的贪心算法，由此产生的编码方案称为Huffman编码
- Huffman算法以自底向上的方式构造表示最优前缀码的二叉树T
- 算法以 $|C|$ 个叶结点开始，执行 $|C|-1$ 次的“合并”运算后产生最终所要求的树T

Huffman编码

●构造过程

- 编码字符集 C 中每一字符 c 的频率是 $f(c)$
- (1) 根据给定的 $|C|$ 个频率，构造 n 棵二叉树的集合 $F=\{T_1, T_2, \dots, T_n\}$ ，其中 T_i 中只有一个根结点，左右子树为空；
- (2) 在 F 中选取两棵根结点频率最小的树作为左、右子树构造一棵新的二叉树，且置新的二叉树的根结点的频率为左、右子树上根结点的频率之和；
- (3) 将新的二叉树加入到 F 中，删除原两棵根结点频率最小的树；
- (4) 重复 (2) 和 (3) 直到 F 中只含一棵树为止

Huffman编码

●贪心策略

- 每次都选择根节点“频率”值最小的两棵树合并