第 5 章 贪心算法

5.1 简介

例子： 有 4 种硬币，面值分别为 2 角 5 分、一角、五分和一分。现在要求以最少的硬币个数找给顾客 6 角三分。

通常是：先拿两个 2 角 5 分的＋1 个一角＋3 个一分

这种找法所拿出的硬币个数最少。这种算法其实就是贪心算法。顾名思义：该算法总是作出再当前看来最好的选择，并且不从整体上考虑最优问题，所作出的选择只是在某种意义上的局部最优解。上面的解法得到的恰好也是最优解。**因此，贪心方法的效率往往是很高的**。

假如，面值有一角一分、五分和一分，要找给顾客一角五分。如果还是使用上述贪心算法，得到的解是：一个一角一分＋4 个一分。而最优解是 3 个 5 分。

算然贪心算法不能对所有问题都能得到最优解，但是对很多问题（包括很多著名的问题）都能产生整体最优解。例如，我们今天要讲的图的单元最短路径问题、最小生成树问题。在有些情况下，算然不能得到整体最优解，但是结果确是最优解的很好近似。

使用贪心法的难点在于：证明所设计的算法确实能够正确解决所求解的问题。

5.2 哈夫曼编码

哈夫曼编码是广泛地用于数据文件压缩的十分有效的编码方法。其压缩率通常在 20%～90%之间。哈夫曼编码算法用字符在文件中出现的频率表来建立一个用 0，1 串表示各字符的最优表示方式。

给出现频率高的字符较短的编码，出现频率较低的字符以较长的编码，可以大大缩短总码长。

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。这种编码称为**前缀码**。

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**的二叉树总是一棵**完全二叉树**，即树中任一结点都有 2 个儿子结点。

平均码长定义为：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**。**构造哈夫曼编码**

哈夫曼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**哈夫曼编码**。

哈夫曼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。

在书上给出的算法 huffmanTree 中，编码字符集中每一字符 c 的频率是 f(c)。**以 f 为键值的优先队列 Q** 用在**贪心选择**时有效地确定算法当前要合并的 2 棵具有最小频率的树。一旦 2 棵具有最小频率的树合并后，产生一棵新的树，其频率为合并的 2 棵树的频率之和，并将新树插入优先队列 Q。经过 n－1 次的合并后，优先队列中只剩下一棵树，即所要求的树 T。

算法 huffmanTree 用最小堆实现优先队列 Q。初始化优先队列需要 O(n)计算时间，由于最小堆的 removeMin 和 put 运算均需 O(logn)时间，n－1 次的合并总共需要 O(nlogn)计算时间。因此，关于 n 个字符的哈夫曼算法的**计算时间**为 O(nlogn)

哈夫曼算法的正确性

要证明哈夫曼算法的正确性，只要证明最优前缀码问题具有**贪心选择性质**和**最优子结构性质**。

(1)贪心选择性质

(2)最优子结构性质

5.3 单源最短路径问题

$G=(V,E)$是一个有向图，每条边上有一个非负整数表示长度值，其中有一个顶点 s 称为源节点。所谓的单源最短路径问题就是：求解该源节点到所有其它节点的最短路径值。不失去一般性，我们假设$V=\{1,2,3,...,n\}$ 并且 $s=1$。那么该问题可以使用 Dijkstra's 算法来求解，该算法是一种贪心算法，并且能求得最优解。

开始时，我们将所有的顶点划分为两个集合 $X=\{1\},Y=\{2,3,4,..,n\}$。所有已经计算好的顶点存放在 X 中，Y 中表示还没有计算好的。Y 中的每个顶点 y 有一个对应的量 $\lambda[y]$，**该值是从源顶点到 y（并且只经由 X 中的顶点）的最短路径值。**

（1） 下面就是选择一个 $\lambda[y]$ 最小顶点 $y\in Y$，并将其移动到 X 中。

（2） 一旦 y 被从 Y 移动到 X 中，Y 中每个和 y 相邻的顶点 w 的 $\lambda[w]$ 都要更新：表示经由 y 到 w 的一条更短的路径被发现了。

对于任意一个顶点 $v\in V$，假如我们使用 $\delta[v]$ 表示源顶点到顶点 v 的最短路径，最后，我们可以证明上述的贪心法结束后有 $\delta[v]=\lambda[v]$。

我们先来看一下算法的框架：

1. $X\leftarrow\{1\};Y\leftarrow V-\{1\}$

2. 对于任意一个 $v\in V$，如果存在一条边从 1 到 v，那么 $\lambda[v]$ ＝该边的长度。否则 $\lambda[v]=\infty$；并设定 $\lambda[0]=0$。

3. while $Y\neq\Phi$

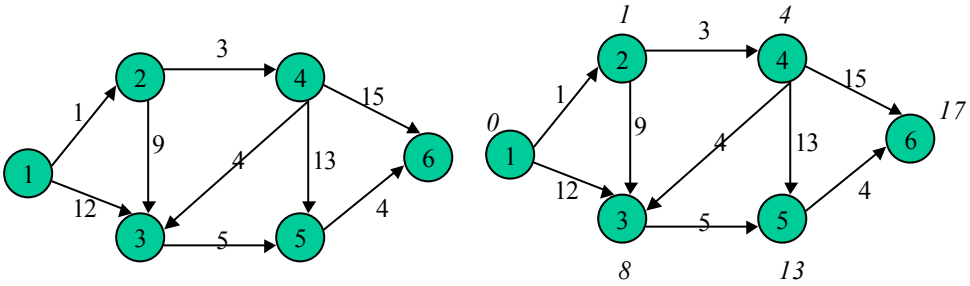
对于 $y\in Y$，找到最小 $\lambda[y]$

将 y 从 Y 移动到 X 中

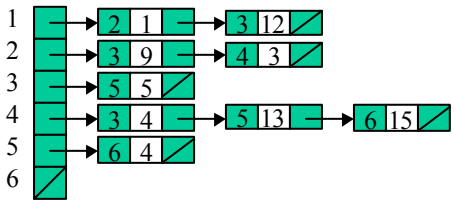
更新 Y 中和 y 相邻顶点的 λ 值

end while

举例说明：



（手工解该题目）



算法的详细实现：

（1） 有向图用邻接矩阵来表示 如图

（2） 假设每条边是非负的

（3） X 和 Y 用两个向量来表示 X[1..n] ,Y[1..n]。初始时 X[1]=1,Y[1]=0.并且对于 2<=i<=n, X[i]=0,Y[i]=1. 因此，执行 $X \leftarrow X \cup \{y\}$ 的操作，就是 X[y]=1, $Y \leftarrow Y - \{y\}$ 的操作就是，Y[y]=0。

详见 PAGE 235

算法 DIJKSTRA

下面来证明，使用该方法得到的是最优解，也就是 $\lambda[y] = \delta[y]$ 。

证明: 归纳法

（1） 显然 $\lambda[1] = \delta[1] = 0$

（2） 假设，当前将 y 移动到 X 中，并且在 y 之前移动到 X 中的任何一个顶点 c，都有 $\lambda[c] = \delta[c]$ 。由于 $\lambda[y]$ 是有限的，也就是说必定存在一条从 1 到 y 路径，长度为 $\lambda[y]$（我们需要来证明 $\lambda[y] = \delta[y]$ ）。

那么这条路径总可以写作：

1→[ ]→[ ]→。。。→[ ]→[ ]→[ ]→y

在上述序列中，我们总是可以找到一个顶点，不妨称之为 x （ $x \in X$ ），

使得 x 之后的顶点均属于 Y 。所以有以下两种情形：

1→[ ]→[ ]→。。。→[ ]→[x]→y　　(A)

或是

1→[ ]→[ ]→。。。→[x]→[ ]…→y　　(B)

√对于情形(A)，

$\lambda[y] \le \lambda[x] + length[x, y]$　　**算法要求**

$= \delta[x] + length(x, y)$　　**归纳假设**

$= \delta[y]$　　**(A)是最短路径**

√对于情形(B)，

我们将 x 之后的顶点不妨称之为 w，即：

1→[ ]→[ ]→。。。→[x]→[w]…→y　　(B)

于是有：

$\lambda[y] \le \lambda[w]$　由于 y 在 w 之前离开 Y

$\le \lambda[x] + length(x, w)$　算法要求

$= \delta[x] + length(x, w)$　归纳假设

$= \delta[w]$　因为 π **是最短路径**

$\le \delta[y]$ 因为 π **是最短路径**

我们已经说了，$\delta[y]$ 是最短路径了，因此 $\lambda[y] = \delta[y]$ 。

时间复杂性分析：

5.4 最小生成树

设 G =(V,E)是无向连通带权图，(V,T)是是 G 的一个子图，并且 T 是一颗树，那么称(V,T)是 G 的生成树。如果 T 的权之和是所有生成树中最小的，那么则称之为最小生成树。

我们假定 G 是连通的，如果 G 是非连通的，那么，可以对 G 的每个子图应用求解最小生成树的算法。

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，则最小生成树就给出了建立通信网络的最经济的方案。

5.4.1 最小生成树性质

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。本节介绍的构造最小生成树的 **Prim 算法**和 **Kruskal 算法**都可以看作是应用贪心算法设计策略的例子。尽管这 2 个算法做贪心选择的方式不同，它们都利用了下面的**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)∈E，且 u∈U，v∈V-U，且

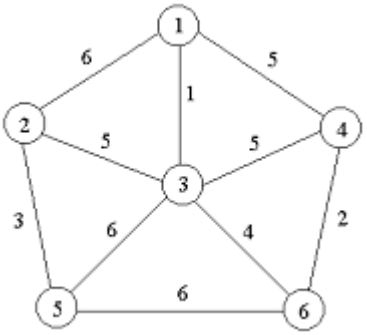
在所有这样的边中，(u,v)的权 c[u][v]最小，那么一定存在 G 的一棵最小生成树，它以(u,v)为其中一条边。这个性质有时也称为 **MST 性质**。

5.4.2 Prim 算法

设 G=(V,E)是连通带权图，V={1,2,…,n}。

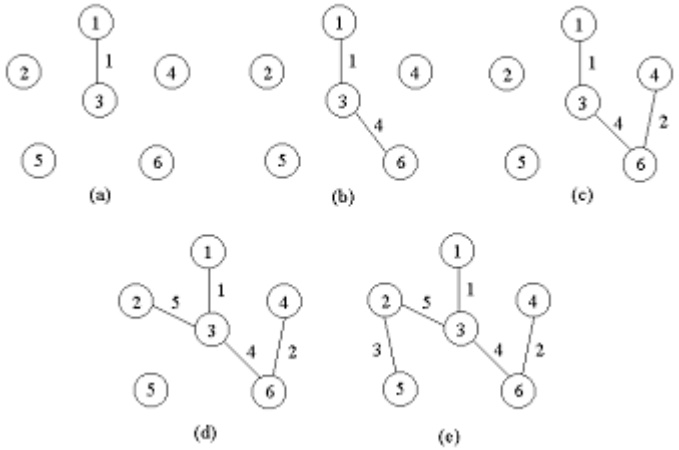
构造 G 的最小生成树的 Prim 算法的**基本思想**是：首先置 S={1}，然后，只要 S 是 V 的真子集，就作如下的**贪心选择**：选取满足条件 i∈S，j∈V-S，且 c[i][j]最小的边，将顶点 j 添加到 S 中。这个过程一直进行到 S=V 时为止。

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**。



利用最小生成树性质和数学归纳法容易证明，上述算法中的**边集合 T 始终包含 G 的某棵最小生成树中的边**。因此，在算法结束时，T 中的所有边构成 G 的一棵最小生成树。

例如，对于右图中的带权图，按 **Prim 算法**选取边的过程如下页图所示



在上述 Prim 算法中，还应当考虑**如何有效地找出满足条件 i∈S,j∈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。

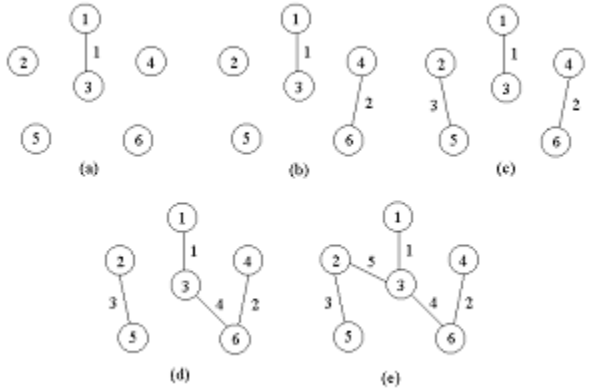
在 Prim 算法执行过程中，先找出 V-S 中使 lowcost 值最小的顶点 j，然后根据数组 closest 选取边(j,closest[j])，最后将 j 添加到 S 中，并对 closest 和 lowcost 作必要的修改。

用这个办法实现的 Prim 算法所需的**计算时间**为 $O(n^2)$.

5.4.3 Kruskal 算法

Kruskal 算法构造 G 的最小生成树的**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 T1 和 T2 中的顶点时，就用边(v,w)将 T1 和 T2 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。

例如，对前面的连通带权图，按 Kruskal 算法顺序得到的最小生成树上的边如下图所示。



KRUSKAL 算法

输入： 具有 n 个顶点的带权连通无向图 $G(V,E)$

输出：G 的最小生成树 T

1. 以非降序的方式对 E 中各条边的权进行排序
2. for each $v \in V$
3. MAKESET($\{v\}$);
4. end for
5. $T = \{\}$
6. while $|T| < n-1$
7. let (x,y) 为 E 中的下一条边
8. if $FND(x) \neq FND(y)$ then
9. $Add(x,y)$ to T
10. $UNION(x,y)$
11. end if

12. end while

关于**集合的一些基本运算**可用于实现 Kruskal 算法。

按权的递增顺序查看等价于对**优先队列**执行 **removeMin** 运算。可以用**堆**实现这个优先队列。

对一个由连通分支组成的集合不断进行修改，需要用到抽象数据类型**并查集 UnionFind** 所支持的基本运算。

当图的边数为 e 时，Kruskal 算法所需的**计算时间**是 $O(e\log e)$。当 $e=\Omega(n^2)$ 时，Kruskal 算法比 Prim 算法差，但当 $e=o(n^2)$ 时，Kruskal 算法却比 Prim 算法好得多。

Kruskal 算法的正确性。

证明：我们只要证明，使用 Kruskal 算法过程中，每次循环所得到的 T (从空集增至最小生成树)总是图 G 的最小生成树的子集即可。使用归纳法＋反证法。

(1) G 总是具有一个最小生成树，不妨记为 T^*。

(2) 当前要加入的边为 $e=(x,y)$。

(3) 包含 x 的那颗子树的所有顶点用 X 表示。

(4) 假设在 $e=(x,y)$ 加入之前得到的 T 均满足 $T\subset T^*$，其中 $x\in X, y\in V-X$。

令 $T'=T\cup\{e\}$，下面我们要证明 T' 也是图 G 的最小生成树的子集。

依据归纳假设，有 $T\subset T^*$。

(A) 如果 $e\in T^*$：显然有 $T'\subset T^*$

(B) 如果 $e\notin T^*$：~~那么 $T^*\cup\{e\}$ 必将构成一个环，也就是 T^* 不再是树~~。我们知道 T^* 中必定包含这样一条边 $e'=(w,z)$，且 $w\in X, z\in V-X$，且 $\text{cost}(e')\geq\text{cost}(e)$。否则，$e'$ 将被选择加入。

下面我们来证明 e' 就是 e。

定义 $T^{**}=T^*-\{e'\}\cup\{e\}$，那么 T^{**} 也是图的一个生成树，并且 $\text{cost}(T^{**})<\text{cost}(T^*)$。也就是说，$T^{**}$ 是最小生成树，那么 T^* 不是最小生成树。矛盾。所以 e **必定是属于** T^* 的。也就必定有 $T'\subset T^*$。

证明完毕。

5.5 数列极差问题

问题描述：

在黑板上写了 N 个正整数作成的一个数列，进行如下操作：每一次擦去其中的两个数 a 和 b，然后在数列中加入一个数 a×b+1，如此下去直至黑板上剩下一个数，在所有按这种操作方式最后得到的数中，最大的 max，最小的为 min，则该数列的 **极差** 定义为 M=max-min。

任务：对于给定的数列，编程计算出极差 M。

分析：仔细分析，我们会发现求 max 与求 min 是两个相似的过程。我们把求解 max 与 min 的过程分开，着重探讨求 max 的问题。

下面我们以求 max 为例来讨论此题用贪心策略求解的合理性。

讨论：

（1）假设经（N－3）次变换后得到 3 个数：

a,b,max′ （ 不妨设 max′ ≥ a ≥ b ）

其中 max′ 是（N－2）个数经（N－3）次 f 变换后所得的最大值，此时有两种求值方式，设其所求值分别为 Z_1，Z_2，则有：

Z_1＝（a × b ＋ 1）×max′ ＋ 1，

Z_2＝（a ×max′ ＋ 1）× b＋1

所以 $Z_1－Z_2$＝max′ － b ≥ 0

（2）若经（N－2）次变换后所得的 3 个数为：m， a， b （m≥ a ≥ b）且m不为（N－2）次变换后的最大值，即m＜max′ 则此时所求得的最大值为：

Z_3＝（a × b＋1）×m＋1

此时

$Z_1－Z_3$＝（1＋a b）（max′ －m）＞ 0

所以此时不为最优解。

所以若使第 k （1 ≤ k ≤N－1）次变换后所得值最大，必使（k－1）次变换后所得值最大，在进行第 k 次变换时，只需取在进行（k－1）次变换后所得数列中的两最小数 p， q 施加 f 操作：p ← p × q ＋1， q ←∞即可，因此此题可用贪心策略求解。讨论完毕。

在求m i n时，我们只需在每次变换的数列中找到两个最大数 p， q 施加作用 f：p ← p × q＋1， q ←-∞即可．原理同上。