算法设计与分析笔记

wr按：
这一部笔记，主要记录学习过程中的一些想法，以便后续查看时能回忆起当时的理解，并加深印象。
附：☞算法导论第三版参考答案（English Ver.）
~~这个答案对于很多题目都没有给出完整回答，参考一下就好~~

*Part*1　分治策略

主方法用来求解递归式，获得一些渐进紧确界：
主定理　令$a \geq 1$和$b > 1$是常数，$f(n)$是一个函数，$T(n)$是定义在非负整数上的递归式

$$T(n) = aT(n/b) + f(n) \tag{1.1}$$

其中我们将n/b解释为$\lfloor n/b \rfloor$或$\lceil n/b \rceil$.那么：
1.若对于某个常数$\varepsilon > 0$有$f(n) = O(n^{\log_b a - \varepsilon})$，则$T(n) = \Theta(n^{\log_b a})$.
2.若$f(n) = \Theta(n^{\log_b a})$，则$T(n) = (n^{\log_b a} \lg n)$.
3.若对于某个常数$\varepsilon > 0$有$f(n) = \Omega(n^{\log_b a + \varepsilon})$，且对于某个常数$c < 1$和足够大的$n$有$af(n/b) \leq cf(n)$，则$T(n) = \Theta(f(n))$.

*Part*2　动态规划与贪心算法

动态规划的两个核心：最优子结构性的证明，状态转移方程的建立。动态规划运行的核心是最优子结构性，证明了这个底层逻辑，动态规划算法就是正确的。

> 使用动态规划算法解决的经典问题：
> 一维DP：钢条切割问题。
> 二维DP：矩阵链乘法问题、最长公共子序列（LCS，Longest Common Subsequence）问题，最优二叉搜索树问题，0-1背包问题。
> DP维度，就是开辟的DP数组的维数。可能要使用三维DP的问题：POJ 1185

贪心算法，在某种程度上可以看做是动态规划的一个特例。动态规划更偏向于穷举，每条路都试一下；贪心算法则是绕开了这种穷举，抄一条近路直接到达目的地，所以很多情况下并不能保证最优解。
贪心算法运行的核心是贪心选择性质——可以通过做出局部最优解（贪心）来构造全局最优解。只要你证明了这个底层逻辑，贪心算法的正确性一般就能够保证。

> Huffman树的生成原理，使用了贪心算法：每次调两个最小的结点合并，直到只剩下一个节点，它就是Huffman树的根节点。
> 图论中也有很多算法的底层逻辑是贪心算法。比如最小生成树的Prim算法和Kruskal算法，单源最短路径的Dijkstra算法。

*Part*3　基本图算法

一、最小生成树
最小生成树的概念，是针对于无向连通图的。有两个贪心算法：

Prim算法：从一个源点S开始。在不形成环的情况下，每次选择连接S所在的连通分支的边中，权重最小的那个。直到只剩下一个连通分支。

Kruskal算法：在不形成环的情况下，从所有的边中选择权重最小的，直到只剩下一个连通分支。

二、单源最短路径

Bellman-Ford算法：如果图G有n个节点，在初始化之后，执行$n-1$轮松弛。每轮松弛都是按照特定的顺序松弛G中所有的边。根据操作后结点的d值是否满足三角不等式，返回bool值，代表该图中最短路径是否能够定义。

> 初始化操作（后面的Dijkstra也是）：对所有的结点$v \in V$，置$v.d = \infty$ and $v.\pi = \text{NIL}$.然后置$s.d = 0$，其中s是源节点。
>
> Bellman-Ford返回的bool值，也代表“图G中是否含有可以由s到达的负权重回路”。一旦返回True，那么对于所有的结点$v \in V$，都有$\delta(s,v) = v.d$.
>
> 在手动运行Bellman-Ford算法的时候，有两种松弛方式：
>
> （1）用每轮松弛操作前的d值进行松弛，是一种“并行松弛”。每轮松弛后的结果与边松弛的先后顺序无关，我们可以按照任意的顺序对这些边进行松弛。
>
> （2）每次松弛时采用最新的d值，是一种“串行松弛”。这个依赖于每条边的松弛顺序，也就是说，松弛顺序不同，一轮松弛得到的结果也会有区别。
>
> 比如我在第一次松弛的时候，得到了$u.d = 6$，$v.d = 7$，图中有$w(u,v) = 3$.在进行第二轮松弛的时候，(u,v)被松弛之前，先更新了$u.d = 2$.如果是采取松弛方式（2），在后续对(u,v)进行松弛的时候，$v.d$会被更新为$2+3=5$；如果是采取松弛方式（1），在后续对(u,v)进行松弛的时候，使用的$u.d$仍然是第一轮松弛得到的6，所以$v.d$不会改变。
>
> Bellman-Ford实际的运行方式是松弛方式（2），但不理解有的题目要我们用方式（1）。不过，虽然方式（2）每轮松弛的结果依赖边松弛的顺序，但是在算法返回True的情况下，最终的结果和这个松弛顺序是无关的——最终$v.d$都会收敛到$\delta(s,v)$.

算法的优点是可以处理含有负权重边的图，缺点是时间复杂度$O(VE)$有点大。简单图的边数E可以达到$O(V^2)$的规模，这个时候Bellman-Ford算法的时间复杂度就是$O(n^3)$.

Dijkstra算法：维持一个集合S.在初始化之后，不断重复这样的操作：

1.对于不在S中的结点，找到d值最小的那个v加入S.

2.松弛所有从v发出的边。

最后，所有的结点都会进入S.此时算法终止。Dijkstra算法在时间复杂度上优于Bellman-Ford算法，不经优化就能达到$O(n^2)$的复杂度，但是不能处理含有负权重边的图。

SPFA算法：维持一个队列Q，初始时有且只有源节点s.重复这样的操作：每次从Q中出队队首节点v，松弛所有从v发出的边，在松弛中d值改变的结点都入队。直到Q空时，算法终止。它的优点是可以处理含有负权重边的图，并且时间复杂度也显著优于Bellman-Ford算法。但是貌似泛用性不强，容易被卡。

三、多源最短路径

有一种自然的想法，对从u到v的路径长度l进行动态规划：求出所有节点对在路径长度不大于$l = l_0$时的最短路径，就可以自底向上地推导所有结点对在路径长度不大于$l = l_0 + 1$时的最短路径。这种动态规划的时间复杂度是$O(n^3 \lg n)$.

Floyd-Warshall算法也是一个动态规划的算法，但思路是先求出所有结点对在经过中间结点v_c编号$c \in [1,k]$时的最短路径，再自底向上地推导出所有节点在经过中间节点编号$c \in [1,k+1]$时的最短路径。时间复杂度可以降至$O(n^3)$.

> Floy-Warshall算法可以看作是三维DP。数组$dp[i][j][k]$表示在中间节点编号$c \in [1,k]$的时候，从结点i结点到j的最短距离。状态转移方程是：

$$dp[i][j][k] = \min(dp[i][j][k-1], dp[i][k][k-1] + dp[k][j][k-1]) \tag{3.1}$$

由于状态转移方程中$dp[\ldots][\ldots][k]$只和$dp[\ldots][\ldots][k-1]$有关，其实可以把dp数组的第三维压掉，变成二维DP.

Johnson算法的设计还是比较巧妙的，它的核心思想就是在确保最短路径不变的情况下，把图G的边权w转化成非负的权重$\hat{w}$，让Dijkstra能跑起来。

Johnson算法引入一个外部节点s加入V中，同时向E中添加n条由s发出、到V中的结点的边，生成图G'.在图G'中以s为源点运行Bellman-Ford算法，给每个节点标记一个h值$h(v) = \delta(s, v)$.

如果Bellman-Ford算法返回True，说明图G不含负权重回路，以$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$运行Dijkstra算法，能够得到各结点对之间的最短距离$\hat{\delta}(u, v)$.修正后的$\delta(u, v) = \hat{\delta}(u, v) - h(u) + h(v)$就是我们需要求的结果。

Johnson算法适用于稀疏图，究其原因还是Bellman-Ford在非稀疏图上跑起来很需要时间。

$Part4$　图树周游

这里涉及到了一个求连通图G关节点的方法。

关节点，就是割点。对于连通图G，其连通分支$p(G) = 1$.割点指的就是满足$p(G - \{v\}) > 1$的这些点v.

对图G进行深度优先遍历，能够得到一颗深度优先生成树T，T中包含了G中所有的顶点和部分边。在图G中，但是不在生成树T中的边称为逆边。

使用深度优先遍历的好处是，图G中不存在这样一条边(u, v)，使得在T中u和v没有祖孙关系（互相不是对方的祖先），也可以说是图G中不含相对T的**交叉边**。如果是使用广度优先生成树T，G就可能存在相对T的交叉边。

图G中每一个结点v都有深度优先数$\mathrm{DFN}(v)$，表示结点v是深度优先遍历图G时第几个被遍历到的结点。每个节点v还有一个最低深度优先数$L(v)$，它的定义是：

$$L(v) = \min\{\mathrm{DFN}(v), \min\{L(w)|w是v的儿子\}, \min\{\mathrm{DFN}(w)|(v, w)是一条逆边\}\} \tag{4.1}$$

直观地，$L(v)$就是v通过一条子孙路径且至多后随一条逆边所可能到达的最低深度优先数。判定割点的规则是，在深度优先生成树T中，根节点如果至少有两个儿子就是割点，非根节点v如果存在一个儿子w满足$L(w) \geq \mathrm{DFN}(v)$就是割点。

$\boxed{\mathbb{The\ End.}}$