Introduction to Algorithm
ID: 519021911248
Name: ZhuoHao Li

Homework 4

Question 1.

1. 假设你到某个著名风景区旅游，风景区内有 n 个景点，各景点间以单向路径连通 (比如 $A \to B$ 与 $B \to A$ 是两条不同路径)，且连接两个景点间的路径长度不等。同时，管理方特意避免了在景点间形成环路。风景区有一个入口 s，有一个出口 t。假设你为了尽量延长在景区内逗留的时间，从而可以尽情欣赏路边景色，希望设计一个动态规划算法来找到一条从 s 出发，到 t 结束，满足路径指向，且长度最长的路径。请给出问题分析过程、优化子结构和重叠子问题的证明、伪代码，并分析时间复杂度和空间复杂度。

My Answer 1. 将这 n 个景点抽象为图的 n 个节点，这个图是一个有向图，并且图中不包含环，后面会讨论如果包含环会带来什么糟糕的影响。这个问题和 *ATSP* 问题很像（在有向完全图中从给定起点 S 遍历图中所有节点后回到 S 的最小权重 *Hamilton* 环)，可以参考他的思路。

首先对问题进行形式化：给定一个有向无环图 V，且任意一条边的长度都不相等，从图中给定的一个节点 s 开始到图中给定的一个节点 t，寻找一个路径最长的路径。

优化子结构 这个问题具有优化子结构：假设已经在 V 中找到了这样一条最长路径，并且该路径经过一个节点 i，则该优化解包含了从节点 i 出发，**经过** V 最后到达 t 的优化解。(我没有用"遍历"的原因，是因为考虑到最后的解不一定包括所有在 V 中的节点，比如，如果一个节点只能被指向，那么它就不应该出现在这个问题的优化解中 (除非他是 t))(注意：这里和 *TSP* 问题的差异是 *TSP* 问题要求一个 *Hamilton loop*，也就是说每个节点只能遍历一次，而本题没有这个选项，也就没有 *classic TSP* 问题证明中未遍历节点 *V'* 的存在了。此外，从节点 i 出发的这个子问题的优化解必然不会遍历到已经在原问题优化解从 s 到 i 的已访问节点上，否则，如果存在节点 j, j 在该路径上，且从 i 出发的子问题优化解路径包含 j，那么必然存在一个从 i 到 j 的环路，矛盾！)

证明：通常地，采用反证法。如果原问题的优化解不包含从 i 出发的子问题的优化解，那么采用 *cut-paste* 策略，将原问题的解从 i 出发的部分 *cut* 掉，并将子问题的优化解 *paste* 到原问题的解上，这样得到的新的原问题解必然比预先假设的优化解更优！这与"*an optimal solution*"的含义是违背的！

重叠子问题 这个问题也具有重叠子问题：显然，求解从当前节点 i 经过 V 并到达 t 的最长路径过程中，会多次求解从某节点 j 经过集合 *V'* V 并到达 t 的规模更小的子问题。在这里，*V'* 可以是 *V-i*，因为子问题的优化解不可以再包含访问过的节点 i 了。

求解思路（分析过程） 起点为 s，假设已经确定了从 s 到当前节点 i 的最长路径，用集合 *V'* 表示剩余的不在 s 到 i 路径上的节点集合（*V'* 当中是包含终点 t 的)，则问题转化为求解从 i 出发，经过 *V'* 并到达 t 的最长路径

令 *d(i,V')* 表示从当前节点 i 出发，经过 *V'* 并到达 t 的最长路径距离，可以得到递归方程：

$$d(i, V') = max_{k \in V'}\{d(k, V' - k) + C_{ik}\}\text{如果有有向边 } ik \text{ 的话}$$

其中 C_{ik} 表示从 i 到 k 的路径长度

$d(t, \{t\}) = 0$

原问题的解是

$$max_{k \in V}\{d(k, V - k) + C_{sk}\}$$

具体的实现，用一个邻接矩阵来存储图，矩阵规模为 $n \times n$, 再用一个二维矩阵 $n \times n$ 保存每一个点到点的最长路径，对第 n 列进行排序，取出第 n 列中最大的一个数 a，记录下行号 R 和列号 C（在 *v1—>v2*

中，行对应的是 *v1*，列对应的是 *v2*)，然后把列号 C 压进栈里（输出的时候直接输出就是最长路线了），接着按照这个数字 a 的行号 R 找到相同数字的列号 *R'*（比如 a 数字在的位置是 *[4][5]*，行号是 *4*，则找到第 *4* 列），令 *N=R*‘，重复以上步骤即可。

伪代码如下所示：

Algorithm 1 LONGESTPATH(V,s,t)

Require: $n \geq 0$

Initialize d[0...n-1]= -∞ ▷ 将每个节点到 t 的初始距离设置成负无穷
if d[k] ≥ 0 **then**
  return d[k]
end if
if k == t **then**
  d[k]=0
  return d[k]
else
  q=$W_j t$
  for i from k+1 to t and $W_j t$ exists **do**
    q=max(q,$W_k i$+LONGESTPAH(V,i,t)
  end for
d[k]=q
return d[k]
end if
return LONGESTPATH

复杂度分析*:*

Q 是一个 *HEAP*，第 *2* 步用堆排序的 *BUILD-HEAP* 构造 *HEAP:O(n);* 后续每一次堆重构都会带来 *O(logn)* 复杂度，循坏 *n-1* 次需要 *O(nlogn)*，所以最终的时间复杂度为：*T(n)=O(n)+O(nlogn)*，当 $n \to \infty$, *T(n)=O(nlogn)*.