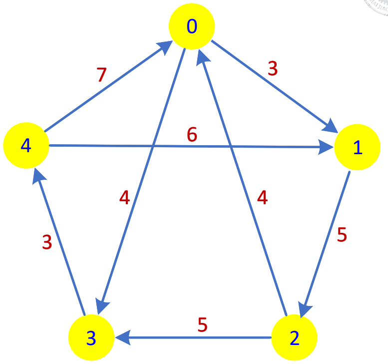
优化子结构及重叠子问题

- 子问题 1：从 0 点出发，到 1 点，然后从 1 出发，经过 {2,3,4}，最后回到 0 点的最短路径，使得总代价最小
- 子问题 2：从 0 点出发，到 3 点，然后从 3 出发，经过 {1,2,4}，最后回到 0 点最短路径，使得总代价最小
- 需要从上述两个子问题中选择一个代价小的解
- 一般来说，假设某一优化解从 0 点到 i 点，则该优化解遍历剩余结点的路径必是从 i 点出发，途经剩余结点恰好一次，并回到 0 点的最短路径。因此，具有优化子结构



显然，求解从当前结点 i 遍历集合 V 中结点恰好一次并返回 0 点的最短路径过程中，会多次求解从某结点 j 经集合 $V' \subseteq V$ 到 0 点的更小子问题，因此具有重叠子问题

TSP 问题的求解思路

- 假设起点为 s，并假设已确定了由 s 到达当前结点 i 的最佳路径，用集合 V' 表示剩余尚未遍历的结点集合 (认为 V' 包含 s)，则问题为求解由 i 出发，遍历 V' 并返回 s 的最短路径
- 令 $d(i, V')$ 表示从当前结点 i 出发，遍历 V' 中结点恰好一次并返回 s 的最小代价，据此，递归方程为：

$$d(i, V') = \min_{k \in V', k \neq s} \{d(k, V' - k) + c_{ik}\}$$

  其中 c_{ik} 为 i 到 k 的代价
- $d(s, \{s\}) = 0$
- 问题的解为求解 $\min_{i \in V, i \neq s} \{d(i, V - i) + c_{si}\}$

TSP 问题动态规划求解的难点

- $d(i, V')$ 中，V' 为集合，无论是自顶向下还是自底向上求解，记录子问题解的表格均需要以集合作为下标
- 如果用 STL 的 set 作为集合，占用空间会很大，而且难以设计结果存储的表格
- 可以考虑采用位向量来表示结点集合，比如对于 4 个城市，可用"0101" 表示第一个城市和第三个城市包含在集合中，这样对于较小的城市数，比如小于 64 个城市，完全可以用整数来表示城市的集合
- 采用上述方法的动态规划算法称为状态压缩的动态规划算法
- 伪代码及具体实现从略