我保证没有抄袭别人作业

1. 证明：只要适当地排列顶点的次序，就能使有向无环图的邻接矩阵中主对角线以下的元素全部为 0。

证明:(归纳法)对图的顶点个数进行归纳,当图中只有一个顶点时,是可以做到的,假设在图中有 n-1 个顶点时,原命题成立,则有 n 个顶点时,图中必然存在一个顶点没有入边(因为是有向无环图),将它作为第一个顶点则邻接矩阵形如 0 a ,A'为(n - 1)*(n - 1)的矩阵 0 A'
则由归纳假设,也可以化为主对角线以下为零的矩阵,所以有 n 个顶点时,存在一个排列使得邻接矩阵主对角线以下元素全部为零.

2. Dr.Stranger 的电脑染上了一种特殊的病毒，该病毒发作时会将字母用其他字母代替，但不会将单词顺序交换，也不会产生新的字母。在病毒发作前文档 D 中的单词顺序为字典排序（比如在字典序下单词 ab 会排在单词 ac 前），且文档 D 中所有的单词只由字母集合{a，b，c，d，e}中的字母组成，病毒发作后文档 D 中的所有单词顺序如下：{cebdbac，cac，ecd，dca，aba，bac}，求文档 D 在病毒发作前的内容，并说明解题思路和步骤。

由首字母及首字母相同时第二个字母的的顺序可得病毒发作后字母的排序为
c -> e -> d -> a -> b, 且 e -> a
故病毒将 a 替换为 c,e 替换为 b,c 替换为 d,d 替换为 a,e 替换为 b
所以文档在病毒发作前的内容为
abeceda, ada, bac, cad, ded, eda

3. Toole 教授提出了一种新的分治算法来计算最小生成树，该算法是这样的：给定一个图 $G=(V, E)$，将顶点集合 V 划分成两个集合 V_1 和 V_2，使得$|V_1|$和$|V_2|$至多差 1。设 E_1 为一个边集，其中的边都与 V_1 中的顶点关联，E_2 为另一个边集，其中的边都与 V_2 中的顶点关联。在两个子图 $G_1=(V_1, E_1)$和 $G_2=(V_2, E_2)$上，分别递归地解决最小生成树问题。最后，从 E 中选择一条通过割(V_1, V_2)（两个端点分别在 V_1 和 V_2 上）的最小权边，并利用该边，将所得的两棵最小生成树合并成一棵完整的生成树。请论证该算法能正确地计算出 G 的最小生成树，或者给出一个使该算法不能正确工作的例子。

反例

```
    A
    |
    B
  /   \
 C     D
```

则任意将顶点集分成两半的划分都存在一个集合没有最小生成树,且合并时只取一条割也不能保证得到的"生成树"经过所有顶点

4. 单源最短路径问题中，当路径中存在负权边时(不存在负权回路)不可以使用 Dijkstra 算法：(注：题中分析时间复杂度时，默认存储结构是邻接表，且使用最小堆存储源点到各点的路径值)
   a) 一种想法是：先在每条边上加上一个常数，从而消除负权边，然后再利用 Dijkstra 算法。请问这种算法可行吗？若可行，请说明理由并分析时间复杂度；若不可行，请举反例。
   b) 另外一种想法是：利用 Dijkstra 算法，在算法运行过程中，若 w 已经在 S 中，但是

由于与 w 相邻的点加入 S 中而新计算得到的 dist[w]比 S 中存储的 dist[w]还要小，那么将 w 从 S 中剔除，重新加入未知集合 T 中。请问这种算法可行吗？若可行，请说明理由并分析时间复杂度；若不可行，请举反例。

a) 不可行 反例

A -3- B -(-3)- D 如果每条边的权重都加上 3,A-B 的最短路径会选择

\1 /1 A-B 而不是 A-C-B

C

b) 可行

(反证)记源点为 u,如果存在 u-v 的一条路径(记为 r : u->x1->x2->...->xk->v)且它的长度比 dist[v]还要小,则分成两种情况讨论

1 如果 r 中没有负权边,则由 dijkstra 算法的证明可知这与每一次选择最短距离的算法矛盾

2 如果 r 中有负权边,不妨假定只有一条负权边(有更多负权边的情形可以归纳证明),则由算法过程可知将 v 加入 S 时,xk 必不在 S 中(否则这一步会选择路径 r 而不是选择 dist[v]确定的路径),但当 xk 加入 S 时,经计算得到新计算出来的 dist[v]比存储的 dist[v]小,会把 v 剔除出集合 S,下一步会选择路径 r 将 v 加入集合 S 中,则最终会以路径 r 的长度作为 dist[v]的值,这与 r 的长度比 dist[v]小是矛盾的,假设不成立

综上,这种想法是可行的

时间复杂度

考虑最坏的情况,即每次加入一个新的结点进入 S 都会驱逐 S 中原有的所有结点.则每处理一个结点需要

1 取出堆中路径长度最小的结点(O(logn), 堆中最多只有 n 个顶点(也可以是 loge, 因为这种最坏的情况下 $e = O(n^2)$)

2 更新 dist 数组(O(n), 因为之前的所有结点都会更新)

3 剔除之前的所有结点

下面计算每加入一个结点会剔除多少次其他结点

$f(n) = f(1) + f(2) + ... + f(n - 1)$ 因为每次加入第 n 个节点会剔除所有之前的结点,之后再重新加入第一个结点,第二个结点(加入第二个节点之后还会再剔除第一个结点,再将更新后的第一个结点加回来,这些都计入 f(2)中)...直到更新到第 n-1 个节点,又 f(1) = O(1), 则有 $f(n) = O(2^n)$

因为每剔除一个结点就意味着之后还要加回来一次,且每一次加入是 O(n)的开销,所以总的时间复杂度为 $O(n * 2^n)$

5. 套汇是指利用汇率差异将一个单位的货币转换为大于一个单位的同种货币。例如，假设 1 美元兑换 7.51 人民币，1 元人民币兑换 0.07 英镑，1 英镑兑换 2.03 美元，那么如果一个人拿 1 美元先兑换成人民币，再把人民币兑换成英镑，最后把英镑兑换成美元，则他最后能够得到 1*7.51*0.07*2.03=1.07 美元，从而获得 1.07-1=0.07 美元的利润，这就是套汇。假设有 n 种货币 $v_1,v_2,\cdots,v_n$ 和有关汇率的 n*n 矩阵，其中 A[i,j]是一单位货币 v_i 兑换成货币 v_j 的单位数，要求设计一个程序判断是否存在一个货币序列 $v_{i1},v_{i2},\cdots,v_{ik}$ 使得 $A[i_1,i_2]*A[i_2,i_3]*\cdots*A[i_k,i_1] > 1$，并确定算法的时间复杂度。

```
bool f(){
  for(int k = 0; k < n; k += 1){
    for(int i = 0; i < n; i += 1){
      for(int j = 0; j < n; j += 1){
        if(A[i, j] < A[i, k] * A[k, j]){
          A[i, j] = A[i, k] * A[k, j];
        }
      }
    }
  }
  if(A[0, 0] > 1){
    return 1;
  }
  return 0;
}
```

利用 floyd 算法,进行 n 次迭代,如果最后 A[0, 0]比 1 大,即存在一条路径使得可以由 1 单位 i_1 货币得到超过 1 单位的货币,即存在一个货币序列

时间复杂度同 floyd 算法的时间复杂度, 为 $O(n^3)$