1. (20 points) Consider the following variant of the single-source shortest path (SSSP) problem. One natural application of SSSP is finding shortest driving routes to various places from a single starting point (for example, edges are roads and nodes are intersections and possible destinations). One issue the model in class would not capture, however, is traffic: specifically, we had constant edge weights, but traffic could cause the amount of time it takes to traverse a road fluctuate even within the time during which we are driving. This problem considers one approach to this issue; one can think of it as the problem faced by an individual who begins a car trip early in the morning and expects to see more and more traffic as people begin their morning commutes.

Say we have a directed graph $G = (V, E)$, with edge weight functions $f_e : \mathbb{Z}_{\geq 0} \rightarrow \mathbb{Z}_{\geq 0}$ for each directed edge $e \in E$, and a start node $s \in V$. We want to find shortest paths from $s$.

For any edge $e$, $f_e(t)$ represents the number of minutes it will take to traverse edge $e$ if we start to do so $t$ minutes after we began moving from node $s$. Assume all of the functions are nonnegative and monotone nondecreasing, that is for all directed edges $e \in E$, and all times $t' > t \geq 0$ we have that $f_e(t') \geq f_e(t) \geq 0$. Will Dijkstra’s algorithm still give shortest paths in this model? Either prove that it does, or provide an example showing that it does not.

2. (20 points / 10 points each part) For the following two sets of character frequencies, compute a Huffman code. Give both a table showing the encoding of each character, and either the steps of the algorithm or the tree corresponding to the code with internal nodes annotated with the order in which they are created during the algorithm.

```
(a)    u  v  w  x  y  z
    15  22  7  1  29  11
(b)    a  b  c  d  e  f  g  h
    13  5  24  7  19  11  30  21
```

3. (20 points) Write code for the attached problem. Submission info for code will be sent to the course list. Your code should read from standard input and write to standard output. Please use Java, C, or C++. It should pass the test cases given in the problem statement (as well as any others that match the input specification), and do so in a reasonable amount of time.
4. (20 points) Consider the following problem. Say we have an \( n \times n \) grid, with a (different) nonnegative integer written in each square. We say that a square is a local minimum if none of the squares immediately above, below, to the right, or to the left of it (if they exist) have lower values. For example, of the below examples in the first two 10 and 20 would be local minimums, while in the third 7 would not be. (The figures below represent excerpts from larger grids; the first and last would be portions in the interior of grids, while the middle one represents an excerpt from along the right edge of a grid.)

\[
\begin{array}{ccc}
5 & 20 & 7 \\
17 & 10 & 11 \\
25 & 35 & 15 \\
\end{array}
\quad
\begin{array}{ccc}
5 & 25 \\
22 & 20 \\
27 & 35 \\
\end{array}
\quad
\begin{array}{ccc}
20 & 5 & 19 \\
17 & 7 & 10 \\
18 & 40 & 23 \\
\end{array}
\]

Assume that rather than having access to the whole grid, we can only ask for the number at a certain column and row. We call such a request a **probe**. Give an algorithm that finds the location of a local minimum of such a grid with only \( O(n) \) probes.

**Challenge Problem** (10 extra points) Say we are given a set of \( n \) objects, and the ability to test whether any two of them are equivalent. The property of equivalence is transitive: if object \( A \) is equivalent to object \( B \), and object \( B \) is equivalent to object \( C \), we know that \( A \) must be equivalent to \( C \) as well. We want to answer the following question with as few equivalence tests as possible: is there a subset of half or more of the items that are all equivalent to each other? Find an algorithm that answers this using only \( O(n) \) equivalence tests (find an algorithm that does so using \( O(n \log n) \) tests for half credit).
2 Fibonacci (fib.{c,cc,java})

2.1 Description
In the Fibonacci integer sequence, \( F_0 = 0 \), \( F_1 = 1 \), and \( F_n = F_{n-1} + F_{n-2} \) for \( n \geq 2 \). For example, the first ten terms of the Fibonacci sequence are:

\[ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots \]

An alternative formula\(^1\) for the Fibonacci sequence is

\[
\begin{bmatrix}
F_{n+1} & F_n \\
F_n & F_{n-1}
\end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{\text{n times}}.
\]

Given an integer \( n \), your goal is to compute the last 4 digits of \( F_n \).

2.2 Input
The input test file will contain multiple test cases. Each test case consists of a single line containing \( n \) (where \( 0 \leq n \leq 1,000,000,000 \)). The end-of-file is denoted by a single line containing the number -1.

\[ 0 \\
9 \\
999999999 \\
1000000000 \\
-1 \]

2.3 Output
For each test case, print the last four digits of \( F_n \). If the last four digits of \( F_n \) are all zeros, print ‘0’; otherwise, omit any leading zeros (i.e., print \( F_n \mod 10000 \)).

\[ 0 \\
34 \\
626 \\
6875 \]

---

\(^1\)As a reminder, matrix multiplication is associative, and the product of two \( 2 \times 2 \) matrices is given by

\[
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix} \begin{bmatrix}
b_{11} & b_{12} \\
b_{21} & b_{22}
\end{bmatrix} = \begin{bmatrix}
a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\
a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22}
\end{bmatrix}.
\]

Also, note that raising any \( 2 \times 2 \) matrix to the 0th power gives the identity matrix:

\[
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix}^0 = \begin{bmatrix} 1 & 0 \\
0 & 1
\end{bmatrix}.
\]