Problem 1: [15pts] Design a linear-time algorithm that, given an undirected graph $G$ and an edge $e$ in $G$, determines whether $G$ has a cycle containing $e$. 
Problem 2: [10+10pts] A path cover of a directed graph $G = (V, E)$ is a set $P$ of vertex-disjoint paths such that every vertex in $V$ is included in exactly one path in $P$. Paths may start and end at any vertex, and they may be of any length, including 0. A minimum path cover of $G$ is a path cover containing the fewest possible paths.

(a) Give a polynomial-time algorithm to find a minimum path cover of a directed acyclic graph $G = (V, E)$.

(Hint: Assuming that $V = \{1, 2, \ldots, n\}$, construct the graph $G' = (V', E')$, where $V' = \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$ and $E' = \{(x_i, y_j) : (i, j) \in E\}$. How does the size of maximum matching of $G'$ relate to the minimum path cover of $G$?)

(b) Show that the problem of finding a minimum path cover is NP-Complete if the input graph contains cycles. (You can use a well-known path problem for the reduction.)
Problem 3: [15pts] Let $X$ and $Y$ be two strings over a given alphabet $\Sigma = \{A, C, G, T\}$. An alignment of $X$ and $Y$ is a way of measuring similarity between $X$ and $Y$: it consists of inserting spaces at arbitrary locations in $X$ and $Y$ (including at either end) so that the resulting sequences $X'$ and $Y'$ have the same length but do not have a space in the same position. For example, suppose $X = \text{ATGCC}$ and $Y = \text{TACGCA}$. Here is an alignment:

\[
X': \quad - \ A \ T \ - \ G \ C \ C \\
Y': \quad T \ A \ - \ C \ G \ C \ A
\]

Here “$-$” indicates a gap. The score of an alignment is specified by a $(|\Sigma| + 1) \times (|\Sigma| + 1)$ matrix $\Delta$, called scoring matrix; extra row and column are to accommodate gaps, where $\Delta(a, b)$ for $a, b \in \Sigma \cup \{-\}$ gives the score of aligning character $a$ in $X$ with $b$ in $Y$. The score of an alignment $(X', Y')$, with $|X'| = |Y'| = m$ is $\sum_{i=1}^{m} \Delta(X'[i], Y'[i])$. For instance, the preceding alignment has the following score:

$$\Delta(-, T) + \Delta(A, A) + \Delta(T, -) + \Delta(-, C) + \Delta(G, G) + \Delta(C, C) + \Delta(C, A)$$

Describe a dynamic programming algorithm that takes as input two strings $X[1 \ldots n]$ and $Y[1 \ldots m]$ and a scoring matrix $\Delta$, and returns in $O(mn)$ time the highest-scoring alignment.
Problem 4: [5+5+5pts] Let $V$ be a set of $n$ cities in US, and let $C$ be a set of airline companies. We are given a set $E = \{e_1, \ldots, e_m\}$ of flight information, where $e_i = (a_i, b_i, c_i, f_i)$ denotes a flight from the city $a_i \in V$ to the city $b_i \in V$ by the airlines $c_i \in C$ and the fare of the flight is $f_i$. Note that multiple airlines may have flights from $a_i$ to $b_i$. Given $E$ and two cities $s, t \in V$, describe efficient algorithms for finding

- the connection from $s$ to $t$ that uses the minimum number of flights;
- the cheapest connection from $s$ to $t$;
- the connection from $s$ to $t$ such that one switches the airlines between two consecutive segments minimum number of times.
Problem 5: [10+10pts] An array \(A[1\ldots n]\) is said to have a majority element if more than half of its entries are same. Given an array, task is to design an efficient algorithm to tell whether array has a majority element, and if so, find the element. The elements of the array are not necessarily from some ordered domain, so only allowed operation is query of the form \(A[i] = A[j]\).

- Show how to solve this problem in \(O(n \log n)\) time.
- Give a linear-time algorithm for the same problem.
  (Hint: Here is another approach. Pair up the elements of array to get \(\frac{n}{2}\) pairs. In each pair, if elements are different discard both of them. If they are same, then keep one of them. Show that after this procedure, there are at most \(\frac{n}{2}\) elements left and they have a majority element if \(A\) does.)
Problem 6: [15pts] Chicago has many tall buildings, but only some of them have a clear view of Lake Michigan. Suppose we are given an array $A[1..n]$ that stores the height of $n$ buildings on a city block, indexed from west to east. Building $i$ has a good view of Lake Michigan if and only if every building to the east of $i$ is shorter than $i$.

Here is an algorithm that computes which buildings have a good view of Lake Michigan. $\text{TOP}(S)$ is the top element in the stack $S$. What is the running time of your algorithm? Justify your answer.

```
GOODVIEW ($A[1 : n]$)
    initialize a stack $S$
    for $i = 1$ to $n$
        while ($S \neq \emptyset \land A[i] > A[\text{TOP}(S)]$)
            POP($S$)
            PUSH($S$)
    return ($S$)
```