Solutions to Final Exam

Instructions: Print you name clearly and sign above. Papers without a name or a signature will not be graded. Answer all parts. Extremely short answers that are intended to hide ignorance will get extremely low scores. Obvious wild guesses will not be viewed favorably as well. This exam requires you to think more than remember.

1. A manufacturer has produced an assembly that is to be used on an upcoming special secret mission of the space shuttle. When the shuttle is launched, the assembly goes into operation and stays in operation until the shuttle returns to earth. Every minute the assembly is operating there is a .002% chance that it will fail, independently of past history. If the assembly fails before the shuttle returns to earth, the mission will be scrubbed and no results will be obtained. NASA needs to determine the longest flight, in hours, that it can schedule to secure a less than 10% chance of mission scrub due to an assembly failure. You will write a Matlab function that will answer this question for NASA. The function has one input parameter which is the per minute chance that the assembly fails. The function outputs the maximum number of hours that achieves the 10% goal. It is safe to assume that no shuttle mission lasts more than 10 days.

(a) First explain in English how you are going to approach this problem.

Acceptable approach 1: Call the input parameter \( p \). We will collect statistics on a large number of experiments then evaluate the results to determine the answer to the question. Each experiment proceeds as follows: repeatedly invoke \( \text{rand} \) until a value is returned that is no greater than \( p \) and let \( \text{count} \) be the number of times \( \text{rand} \) was called until that happened. Dividing \( \text{count} \) by 60 and rounding down gives the number of hours before failure for that experiment. We determine the number of experiments for which failure occurred in each hour from 0 to 240 (maximum mission is 10 days). Finally, we sum all these counts from hour 0 to hour \( i \) where \( i \) starts at 1 and is increased by 1 until the sum is greater than 10% of the number of experiments. The result returned by the Matlab function is \( i-1 \).

Acceptable approach 2: The probability that a failure will occur in the \( i^{th} \) minute is \((1-p)^{i-1}p\). The probability that a failure will occur on or before the \( i^{th} \) minute is therefore

\[
\sum_{k=1}^{i}(1-p)^{k-1}p = p \cdot \sum_{k=0}^{i-1}(1-p)^{k} = 1 - (1-p)^{i}
\]

So all we need to do is determine \( i \) so that \( 1 - (1-p)^{i} = 0.1 \). This is

\[
i = \frac{\ln(0.9)}{\ln(1-p)}
\]

and the value that is returned is this value divided by 60, rounded down.
(b) Now write the Matlab function.

Alternative 1:

```matlab
function result = nasa(p)
    nexp = 10000;
    bin = zeros(1,241); % bin(i) = number of failures in hour i-1
    for i=1:nexp
        count = 0;
        while rand > p count = count+1; end
        n = floor(count/60); % number of hours before failure
        if n <= 240 bin(n+1) = bin(n+1)+1; end
    end
    s = 0;
    result = 240; % in case 10% is not reached in 10 days
    for i=1:length(bin)
        s = s + bin(i);
        if s >= .1*nexp
            result = i-1;
            break;
        end
    end
end
```

Alternative 2:

```matlab
function result = nasa_alt(p)
    result = floor((log(0.9)/log(1-p))/60);
end
```
2. NASA needs to get the assembly that was mentioned in the previous question from the manufacturer’s plant in Berkeley, California to Cape Canaveral, Florida. The assembly will be shipped by rail. Because every city is presumed to have spies, NASA wants to choose a route that passes through the minimum number of cities (that is, that takes the minimum number of hops). You are going to provide a Matlab function for NASA which does this. Your input is a railroad map which consists of a collection of pairs of cities: if two cities are a pair in the railroad map then there is a rail link between these cities such that no other city is encountered along that link; and there is no such rail link between two cities that are not listed as a pair in the railroad map (that is, any travel between an unlisted pair of cities requires passing through at least one other city). Numbers from 1 to 1000 will be used to identify cities. Assume Berkeley is city number 1 and Cape Canaveral is city number 1000. What we want to do is illustrated by the small example shown below which consists of 9 cities and where the city of origin is city 1 and the city of destination is city 9. Cities are dots and map pairs are shown as lines connecting pairs of dots.

In this case, the best route (to minimize the number of hops) is 1-6-9.

Your function will read a railroad map from a file and output the route as a sequence of cities in the order of travel from city 1 to city 1000. The file contains 1000 lines, the $i^{th}$ line in sequence corresponds to city number $i$ and specifies those cities that may be reached from city $i$ without passing through another city. All lines end with 0. For the above example, the file looks like this:

```
2 3 6 7 0
1 3 4 6 0
1 2 4 8 0
2 3 9 0
8 9 0
1 2 9 0
1 8 0
3 5 7 9 0
4 5 6 8 0
```

The idea needed to solve this problem is simple. Call a pair of cities in the map neighbors. Mark city 1 visited. All neighbors of city 1 are 1 hop away from that city. Mark them visited. All unmarked neighbors of those cities are 2 hops away from city 1. Mark them visited. All unmarked neighbors of those cities are 3 hops away from city 1 and so on until city 1000 is reached. When that happens we need to trace a path back to city 1. More on that later. We will build a Matlab function based on this idea in stages. In each stage we will implement some function until finally we will put them all together to solve the problem. The first function, call it `readfile`, takes as input a file name and outputs a 1000 by 1000 matrix C such that the entry in the $i^{th}$ row and $j^{th}$ column of C is 1 if the $i^{th}$ line of the file contains the number $j$, and is 0 otherwise.
(a) Complete the function `readfile` below:

```matlab
function C = readfile(filename)

    C = zeros(1000,1000);
    fid = fopen(filename,'rt');
    for i=1:1000
        token = fscanf(fid,'%d',1);
        while token
            C(i,token) = 1;
            token = fscanf(fid,'%d',1);
        end
    end
end
```

Next we build functions to support a structure known as a Queue. A Queue is a list of numbers that starts out empty. Numbers can be added to and removed from a Queue: when a number is added it becomes the rightmost number in the Queue’s list and the number that is removed is always the leftmost number of the list. We are going to use variable `queue` to hold a Queue’s list.

(b) Complete the Matlab expression below which is intended to initialize the list.

```matlab
queue = []; % Initialize the Queue
```

(c) Complete the Matlab function below, called `push`, that takes as input a Queue’s list `queue` and a number `n` to add to that list and outputs the Queue’s list modified to contain `n`.

```matlab
function queue = push(queue,n)
    queue = [queue n];
end
```

(d) Complete the Matlab function below, called `pop`, that takes as input a Queue’s list `queue` and outputs the first number of `queue` and a modified version of `queue` that is exactly as the input version except the first number is missing.

```matlab
function [n queue] = pop(queue)
    n = queue(1);
    queue = queue(2:length(queue));
end
```
(e) While running the main function we will need to see whether a particular city is visited and, if so, mark it visited so we will never visit it again. We will use a vector called visited to do this. Complete the following Matlab expression for initializing visited.

\[
\text{visited} = \text{zeros}(1,1000);
\]

(f) We will need a vector, call it back, of numbers, one for each city, that tells us what city to proceed to next along the route of minimum hops. For example, if \( \text{back}(12) \) is 4 and \( \text{back}(4) \) is 116, then the best route from city 12 to city 1 will continue to city 4 then to city 116 and so on until reaching city 1. Complete the following Matlab expression for initializing back.

\[
\text{back} = \text{zeros}(1,1000);
\]

(g) When city 1000 has been reached we can trace a route back to city 1 using the back vector. Thus, the first city on the best route, we call it city1 here, is given by \( \text{city1} = \text{back}(1000) \). The next, we will call it city2, would be \( \text{city2} = \text{back}(\text{city1}) \), and so on. Eventually, \( \text{back}(\text{next_to_last_city}) \) will be 1 and the best route would have been traversed. Write a Matlab function, called getroute, that traverses a path to 1 and outputs a vector of cities in the order they were visited on that route. The input to this function is the back vector.

\[
\begin{align*}
\text{function route} &= \text{getroute(back)} \\
\text{city} &= 1000; \\
\text{route} &= []; \\
\text{while} \text{ city} \\
\quad &\quad \text{route} = \text{push(route,city)}; \\
\quad &\quad \text{city} = \text{back(city)}; \\
end \\
end
\end{align*}
\]

(h) Now we put all of the above together to complete the main function, called bestroute, for finding the route of minimum hops between cities 1 and 1000. We use the following simple procedure outline.

Read the contents of the file into matrix C.
Initialize back, visited, route, and queue.
Push 1 into queue and mark city 1 visited.
Repeat the following until queue has no numbers:
Pop a number off queue and call it i.
For each non-zero entry in column j of row i of C such that city j is not yet marked visited, do the following:
Set back(j) to i and mark city j visited.
If j is 1000 then
Output the result of invoking getroute(back).
Otherwise
Push j into queue.
function route = bestroute(filename)
    C = readfile(filename);
    visited = zeros(1,1000);
    back = zeros(1,1000);
    queue = [];
    route = [];
    queue = push(queue,1);
    visited(1) = 1;
    while length(queue)
        [i queue] = pop(queue);
        for j = 1000:-1:1
            if C(i,j) && ~visited(j)
                back(j) = i;
                visited(j) = 1;
                if j == 1000
                    route = getroute(back);
                    queue = [];
                    break;
                else
                    queue = push(queue,j);
                end
            end
        end
    end
end