CS 1115: Prelim 2 Solution

November 14, 2013

(Name)

(Cornell ID)

Problem 1	20 points	17.7
Problem 2	20 points	18.4
Problem 3	20 points	18.6
Problem 4	20 points	18.2
Problem 5	20 points	18.3
		91.3

Rough: A=[85,100], B = [70,80] , C = [55 65].

1. Complete the following function so that it performs as specified:

```
function V = TopValue(Inv,Price,k)
% Inv is an m-by-n matrix of nonnegative integers with the property that
%    Inv(i,j) is the maximum amount of vegetable j (in pounds) that can be purchased
%    from supermarket i.
% Price is an m-by-n matrix of non-negative real numbers with the property
%    that Price(i,j) is the price (in dollars) of one pound  of vegetable j
%    if purchased from supermarket i.
% k is an integer that satisfies 1<=k<=m
% V is the most one can spend purchasing all the vegetables from k different
%    supermarkets.
```

Vectorization is optional. FYI: If `x` is a length-r vector and `y = sort(x)`, then `y` is a length-r vector whose components are a permutation of the values in `x` arranged from smallest to largest.

```
[m,n] = size(Inv);
p = zeros(m,1);
for i=1:m
   for j=1:n
      p(i) = p(i) + Inv(i,j)*Price(i,j);
   end
end
q = sort(p);
V = sum(q(m-k+1));
```

2. Assume the availability of

```
  function H = MakeLocation(i,j)
% x and y are integers and H is a 2-field structure with
%      H.i = x
%      H.j = y
% H can be thought of as the location of a robot on a grid.
H = struct('i',x,'j',y);
```

Complete the following function so that it performs as specified and is efficient:

```
  function alfa = Intersect(P1,P2)
% P1 and P2 are structure arrays of locations with equal length.
% For all valid indices k, P1(k) is the location of robot 1 at time k
%   and P2(k) is the location of robot 2 at time k.
% alfa = 1 if there is a time when the two robots have the same location
% alfa = 0 if there is never a time when the two robots have the same location

k=1;
n = length(P1);
alfa = 0;
while k<=n && ~alfa
   alfa = (P1(k).i==P2(k).i && P1(k).j==P2(k).j)
   k = k+1;
end
```

3. Assume that `S` is a length-48 structure array with the property that `S(k).name` is a string that encodes the name of the k-th state and `S(k).v` is a vector whose components are the indices of the states that it touches. For example, if Washington is state 46, Idaho is state 30, and Oregon is state 35, then the value of `S(46).name` would be `'Washington'` and the value of `S(46).v` would be `[30 35]` since Washington is bordered by Idaho and Oregon.

Write a MATLAB fragment that sets up a length-48 cell array `C` where the k-th cell is the string obtained by concatenating to the name of the k-th state, the names of all the states that it touches. A blank should separate the state names and `C{k}` should begin with the name of the kth state. For the above example, the value of `C{46}` would be `'Washington Idaho Oregon'`.

```
C = cell(48,1);
for k=1:48
   C{k} = S(k).name;
   for j=1:length(S(k).v)
      i = S(k).v(j);                % Index of the jth border state
      C{k} = [C{k} ' ' S(i).name];
   end
end
```

4. This problem is about "enlarging" a black and white image in the horizontal direction. Suppose the original image matrix is

×	×	×	×	×	×
×	×	×	×	×	×
×	×	×	×	×	×

Each "×" is a `uint8` pixel value.

Using interpolation we can expand in the horizontal direction obtaining

×	a	b	×	a	b	×	a	b	×	a	b	×	a	b	×
×	a	b	×	a	b	×	a	b	×	a	b	×	a	b	×
×	a	b	×	a	b	×	a	b	×	a	b	×	a	b	×

Here, each "a" and "b" is a `uint8` pixel value obtained by interpolating between the neighboring "×" values in the same row. Thus, if

10	a	b	70

we set

$$\begin{aligned} \text{a} &= 10 + (1/3) * (70 - 10) \\ \text{b} &= 10 + (2/3) * (70 - 10). \end{aligned}$$

Complete the script below so that it assigns to `B` the widened version of `A`. Don't forget that `double()` can be used to convert to floating point format and `uint8()` can be used to convert to uint8 format. The use of vector-level operations is optional.

```
 A = rgb2gray(imread('Cornell_Clock.jpg'))
 [m,n] = size(A);
 B = zeros(1:m,3*n-2,'uint8');

for i=1:m
   for j=1:n
      B(i,3*j-2) = A(i,j);
      if j<n
         alfa = double(A(i,j));
         beta = double(A(i,j+1));
         B(i,3*j-1) = uint8(alfa+(1/3)*(beta-alfa));
         B(i,3*j)   = uint8(alfa+(2/3)*(beta-alfa));
      end
   end
end
```

5. Suppose `A` is an m-by-n array of distinct positive integers. An entry in `A` is said to be a *saddle point entry* if it is the largest entry in its row and the smallest entry in its column *or* if it is the smallest entry in its row and the largest entry in its column. Thus if

```
          80  12  14  26  90  56  15
          55  20  50  40  70  96  67
    A =   99  17  14  51  82  72  44
          10  13  25  36  43  51  19
          30  11  49  63  61  81  37
```

then `A(2,2)` and `A(4,6)` are saddle point entries. By making effective use of the built-in functions `max` and `min`, give an efficient implementation of the following function so that it performs as specified:

```
  function [ivec,jvec] = FindSaddle(A)
% A is an m-by-n matrix with distinct entries that are positive integers.
% If A has no saddle point entries, then ivec and jvec are empty vectors.
% Otherwise, ivec and jvec are row vectors of equal length that specify
% all the saddle point entries. In particular, if ivec and jvec have length q,
% then A(ivec(1),jvec(1)),...,A(ivec(q),jvec(q)) are the saddle point entries.
```

Thus, if this function is applied to the above example, then `ivec = [2 4]` and `jvec = [2 6]`. (Order is not important so it would be OK to return `ivec = [4 2]` and `jvec = [6 2]`.) FYI, if the components of a vector `v` are distinct, then

```
[y,idx] = max(v)  assigns the maximum value in v to y and y = v(idx).

[y,idx] = min(v)  assigns the minimum value in v to y and y = v(idx).
```

```
[m,n] = size(A);
ivec = [];
jvec = [];
for i=1:m
   % Check for saddle entries in row i
   [y,j] = max(A(i,:))
   % Check if A(i,j) is the smallest entry in its column
   if y == min(A(:,j))
      ivec = [ivec i];
      jvec = [jvec j];
   end
   [y,j] = min(A(i,:))
   % Check if A(i,j) is the largest entry in its column
   if y == max(A(:,j))
      ivec = [ivec i];
      jvec = [jvec j];
   end
end
```

```
      [y,j] = min(A(i,:));
      % Check to see if A(i,j) is the largest entry in its column
```

```
      if y = max(A(:,j))
         ivec = [ivec i];
         jvec = [jvec j];
      end
   end
end
```