CS 580 Algorithm Design, Analysis, And Implementation
Fall 2021 Vassilis Zikas HW 4

Problem 5

Collaborators: List students that you have discussed problem 5 with: Vishnu Teja Narapareddy, Tulika Sureka

We can use the dynamic programming approach to solve this problem.
Our objective is to repair the damaged parts of the road using minimal area being covered in the patches used for repairing, given the constraint that we can use atmost P patches, rectangular in shape.
Let say the road is given to us in the form of 2-dimensional matrix, road[2][n], where damaged parts are represented by 0 and the good parts are represented by 1.

SubProblem:
opt[i][j] has a structure which denotes the minimum area covered using 'i' patches for a road of length upto 'j' units, along with the dimensions of the last patch. The constraints for i are i $\in$ [0,P] and j are j $\in$ [0,N].

Recursion:
At every step, we have two options when we try to increase the length of road by 1 unit:

1. We use the last patch and try to extend it if the new column has a damaged portion
2. We start a new patch to cover the damaged portion in the new column

For this approach to work, we need to store the coordinates of our patch so that we know about the orientation of the last patch. It will help us to see how much area needs to be covered while extending the length of the road covered by 1 unit. The structure used to store the patches is as follows:
Let m be left column from where the patch starts and n be the ending column for the patch. We also need to take into account the orientation of the patch. So, we store another integer to take care of that. Let say, field 'o' denotes the orientation of the patch. If the patch covers just the first row, we store the value 0 in the orientation field. If it covers just the second row, we store the value 1 in the orientation field. If it covers both the rows, we store the value 2 in the orientation field.
This orientation of the last patch, along with the orientation of the new column being added in the road is required to get the overall dimension and orientation of the updated patch afterwards.
So, we have the following cases:
Here, i goes from 1 to P and j goes from 1 to N.

1. if road[0][j-1]==1 and road[1][j-1]==1 // new column is good
   opt[i][j] = opt[i][j-1] // no need to extend the patch or start a new patch, just use the value for the last column
2. if road[0][j-1]==0 and road[1][j-1]==1 // upper element in new column is damaged
   This means the orientation of the new column is 0 as only the upper element is damaged in the column being added

(a) extending the last patch

i. orientation of last patch is 0:
last_patch = opt[i][j-1].patch
opt[i][j].area = opt[i-1][last_patch.m - 1].area + (j-last_patch.m)
opt[i][j].patch = last_patch.m,j,0

ii. orientation of last patch is not 0:
last_patch = opt[i][j-1].patch
opt[i][j].area = opt[i-1][last_patch.m - 1].area + 2*(j-last_patch.m)
opt[i][j].patch = last_patch.m,j,2

Take the min area value for the two cases discussed above

(b) starting a new patch
opt[i][j].area = opt[i-1][j-1].area + 1 // no need to take into account the orientation of last patch
opt[i][j].patch = (j-1,j,0) // start col, end col, orientation

Take the min area value for the two cases discussed above (extending the last patch, starting a new patch)

3. if road[0][j-1]==1 and road[1][j-1]==0 // lower element in new column is damaged
This means the orientation of the new column is 1 as only the lower element is damaged in the column being added

(a) extending the last patch

i. orientation of last patch is 1:
last_patch = opt[i][j-1].patch
opt[i][j].area = opt[i-1][last_patch.m - 1].area + (j-last_patch.m)
opt[i][j].patch = last_patch.m,j,1

ii. orientation of last patch is not 1:
last_patch = opt[i][j-1].patch
opt[i][j].area = opt[i-1][last_patch.m - 1].area + 2*(j-last_patch.m)
opt[i][j].patch = last_patch.m,j,2

Take the min area value for the two cases discussed above

(b) starting a new patch
opt[i][j].area = opt[i-1][j-1].area + 1 // no need to take into account the orientation of last patch
opt[i][j].patch = (j-1,j,1) // start col, end col, orientation

Take the min area value for the two cases discussed above (extending the last patch, starting a new patch)

4. if road[0][j-1]==0 and road[1][j-1]==0 // new column is fully damaged
This means the orientation of the new column is 2 as both the elements are damaged in the column being added

(a) extending the last patch

i. orientation of last patch does not matter in this scenario:
last_patch = opt[i][j-1].patch
opt[i][j].area = opt[i-1][last_patch.m - 1].area + 2*(j-last_patch.m)
opt[i][j].patch = last_patch.m,j,2

(b) starting a new patch
opt[i][j].area = opt[i-1][j-1].area + 2 // no need to take into account the orientation of last patch
opt[i][j].patch = (j-1,j,2) // start col, end col, orientation

Take the min area value for the two cases discussed above (extending the last patch, starting a new patch)

Base Case:
For j $\in$ [0,n]:
opt[0][j].area = 0 // area that can be covered is zero when there is no patch
opt[0][j].patch = null, null, null
for i $\in$ [0,P]:
opt[i][0].area = 0 // area to be covered is zero when there is no road
opt[i][0].patch = null, null, null

For the recursive cases, we make a check that if the last patch was null, we can't extend the patch and have to go for the case - starting a new patch.

The result to our problem is opt[P][N] as it would give us the min area which would be required by P patches to cover the damaged portion of road of length N.

Analysing TC:
We are traversing through the complete matrix of size (P+1)*(N+1). So, the time complexity for the overall problem becomes O(NP). Maximum value for P should be 2N as the size of road, considering all the blocks were damaged and we used individual patches to cover each block.
Analysing SC:
We require to store the matrix of size (P+1)*(N+1) as per the algorithm. So, the overall SC is O(NP)

Proof of Correctness:
We can use strong induction to prove the correctness of the algorithm.

For base cases (N=0 or P=0), it is trivial to observe that we would not be covering any road area through patches, so their value is 0.

Now, let say our algorithm gives the optimum area value for all the values in the row opt[i] and for the sub-problem opt[i][j-1]. We need to show that the algorithm provides the optimal value for opt[i][j] too.

opt[i][j] denotes the min area to be covered using i patches up till j column. So we can see that for the area to be minimised, we have two choices, either to extend the last patch used or to start a new patch from this column.

1. we extend the patch
   We need to find the area required to cover till the start of the last patch using i-1 patches and then add the new area after extension of the patch as described for each of the possible orientations in the algorithm.
   We already know that all the previous computed values are optimal. So, we get the corresponding value from the matrix and add the new computed area.

2. we start a new patch
   We know the area required to cover the road uptil j-1 column using i-1 patches (already computed before). We also know that it is optimal (using strong induction).
   We add the new patch area corresponding to the orientation of the column being added to this value.

Finally, we take the min of the two values from the possible cases discussed above. As those values were computed using the optimal values till the previous point, we can say that the area value which we get for this cell in the matrix is also optimal. We see that if it stores any other value, then it won't be the min area covered using i patches till j column. This ensures that opt[i][j] also stores the optimal value.
The result would be stored in the cell opt[P][N], which will give us the minimum area to be covered using P patches for a 2-lane road of length N units.
Hence, proved.