Report for assignment 2

Shashwat Gupta (14IE10028)

January 14, 2016

Fill a nxn board with one cell missing with L shaped tiles

1. **Implementing structure** The structure is implemented to store two integers x and y which are the coordinates on the board

2. **Implementing recursive function to split the board into 4 equal squares of size n/2**

   This problem can be solved using Divide and Conquer. Below is the recursive algorithm. Function **rec_fill** is the recursive function doing the following things:

   - Find the Centre.
   - Call function to find the quadrant that the missing/defective cell lies in.
   - Call function to fill L tile appropriately.

   Base Condition for termination of recursion
   if(size==2)
   return;

   Finding centre
   centre.x=start.x+size/2-1;
   centre.y=start.y+size/2-1;

   If the Base condition is not satisfied, the function first checks for which zone the defect is in and fills the L tile accordingly. It then effectively divides the board into 4 equal squares each of size $\frac{n}{2}$, and implements recursion by calling each of the 4 sub-boards recursively. Before calling each sub-board, it updates the start point and the defect point and passes them.

   Sample code for calling the four sub-boards recursivley. This code is for when the defect is in zone 1.

   startCopyy.x=start.x;
   startCopyy.y=start.y;
   defectCopyy.x=defect.x;
   defectCopyy.y=defect.y;
   recfill(startCopyy,size/2,defectCopyy);

startCopyy.x=start.x+size/2;
startCopyy.y=start.y;
defectCopyy.x=start.x+size/2;
defectCopyy.y=start.y+size/2-1;
recfill(startCopyy,size/2,defectCopyy);
startCopyy.x=start.x+size/2;
startCopyy.y=start.y+size/2;
defectCopyy.x=start.x+size/2;
defectCopyy.y=start.y+size/2;
recfill(startCopyy,size/2,defectCopyy);
startCopyy.x=start.x;
startCopyy.y=start.y+size/2;
defectCopyy.x=start.x+size/2-1;
defectCopyy.y=start.y+size/2;
recfill(startCopyy,size/2,defectCopyy);

3. **Function that checks for quadrant**
int checkdefectzone(point centre, point defect)
if(defect.x more than centre.x)
if(defect.y more than centre.y)
return 3;
else
return 2;
else
if(defect.y more than centre.y)
return 4;
else
return 1;

4. **Function that fills the L tile appropriately**
void fill4operation(point a, int z)
if(z==1)
arr[a.x+1][a.y+1]=printer;
arr[a.x][a.y+1]=printer;
arr[a.x+1][a.y]=printer;
else if(z==2)
arr[a.x][a.y]=printer;
arr[a.x][a.y+1]=printer;
arr[a.x+1][a.y+1]=printer;
else if(z==3)
arr[a.x][a.y]=printer;
arr[a.x+1][a.y]=printer;
arr[a.x][a.y+1]=printer;
else

arr[a.x][a.y]=printer;
arr[a.x+1][a.y]=printer;
arr[a.x+1][a.y+1]=printer;
printer++;

5. **Printing the Board**
   void printarray(int size)
   printf("printing board: ");
   int i,j;
   for (i = 0; i less than size; ++i)
   for (j = 0; j less than size; ++j)
   printf("d",arr[i][j]);
   printf(" ");

   While printing the board, we observe a series of number in the board, where we observe all integers three times except the integer -1. The -1 represents the original missing cell and all the other integer triplets represent the L tile.

6. **Some Notes** The baord has been assumed to be a 2-D array and its has been defined globally. Its size is user defined with an upper limit m which is macro defined and can be changed. A printer variable is also globally defined to store the running triplet number.

7. **Time Complexity Calculation** For each value of size, the board checks if the size is 2X2 and if it isn't then it recursively calls the 4 sub-boards of size $\frac{n}{2}$

$$T(n) = \begin{cases} a & : n = 1 \\ 4T(\frac{n}{2}) + c & : n > 1 \end{cases}$$

   Solving this equation by recursive method yields T(n) $\in O(n^2)$.

Finding the closest pair of points

1. **Implementing structure** The structure is implemented to store two integers x and y which are the coordinates on the board

2. **Implementing function to find the distance between 2 points**
   float givedistance(point a, point b)
   return sqrt((a.x-b.x)(a.x-b.x)+(a.y-b.y)*(a.y-b.y));*

3. **Implementing function to randomly populate the set of coordinate points with whole numbers less than 100**
   void fillcoordinates()
   srand(time(NULL));
   int i;

printf(”Enter number of points : ”);
scanf(”d”,n);
for (i = 0; i ¡ n; ++i)
a[i].x=rand() mod 100;
a[i].y=rand() mod 100;

4. **Implementing function to print**
   void printarray()
   int i;
   for (i = 0; i ¡ n; ++i)
   printf(”(d,d) ”,a[i].x,a[i].y);
   printf(” ”);

5. **Implementing function to return least distance**
   float bruteforce()
   int i,j;
   float min=givedistance(a[0],a[1]);
   for(i=0;i¡n-1;i++)
   for(j=i+1;j¡n;j++)
   if(givedistance(a[i],a[j])¡min)
   min=givedistance(a[i],a[j]);
   return min;

6. **Time Complexity Calculation** For each value the board checks the distance with all the other values and finds the minimum

   Solving this equation by recursive method yields T(n) $\in O(n^2)$.