CS361 Algorithm Lab 3

Nathan Stark

CS 361 Lab 3

What to do

1. Implement the DP version of MCM algorithm. Show commented code.

```
17      //A matrix to be shown. eno get it from the matrix lol.
18      private static int[][] eno;
19
20
21
22      /************************************************************ LAB 3 ****************************
23
24      /**
25       * This in my DP matrix chain multiplication method for lab 3
26       * of CS 361 I had found this code on GeeksForGeeks.com at
27       * http://www.geeksforgeeks.org/dynamic-programming-set-8-matrix-chain-multiplication/
28       * This code is contributed by Rajat Mishra.
29       *
30       * @param p is the array of the sizes of the matrixes.
31       * @param n is the length of the array of p.
32       * @return
33       */
34      public int matrixChainOrder(int[] p, int n){
35
36          eno = new int[n][n];                          // Initialize the matrix.
37
38          int i,j,k,L,q;                                // Variables to be descried and/or used later.
39
40          for(i=1;i<n;i++){                             // i will be used through out as just an indexing.
41              eno[i][i] = 0;                            // Fill in the diagonal of the matrix.
42          }
43
44          for(L=2;L<n;L++){                             // L is to keep track of the length of the chain.
45              for(i=1;i<n-L+1;i++){
```

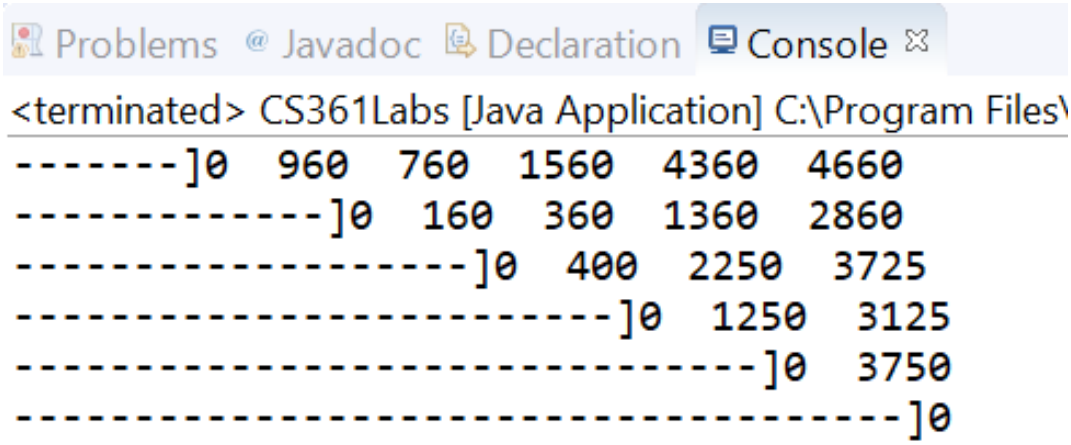
```
44          for(L=2;L<n;L++){                             // L is to keep track of the length of the chain.
45              for(i=1;i<n-L+1;i++){
46                  j=i+L-1;
47                  if(j==n) continue;
48                  eno[i][j] = Integer.MAX_VALUE;
49                  for(k=i;k<=j-1;k++){                  // k and j are also just indexers.
50                      q = eno[i][k] + eno[k+1][j] +     // q is the cost to multiply.
51                              p[i-1]*p[k]*p[j];
52                      if(q<eno[i][j]){
53                          eno[i][j] = q;
54                      }
55                  }
56              }
57          }
58          return eno[1][n-1];
59      }
```

```
440     /**
441      * @param args
442      */
443     public static void main(String[] args) {
444 /************************************************************ DP_MCM ****************************************************************
445         CS361Labs lab3DPMCM = new CS361Labs();
446         arr = new int[] {30,4,8,5,10,25,15};                          // initialize the p array.
447         int size = arr.length;
448         lab3DPMCM.matrixChainOrder(arr, size);
449
450         for(int t=1;t<size;t++){                                      // the following will print the top half of the matrix.
451             int c= t+1;
452             String repeat = new String(new char[t*2]).replace("\0", "---");
453             System.out.print(repeat+"-]"+eno[t][t]);
454             for(c=t+1;c<size;c++){
455                 System.out.print("  "+eno[t][c]);
456             }
457             System.out.println();
458         }
```

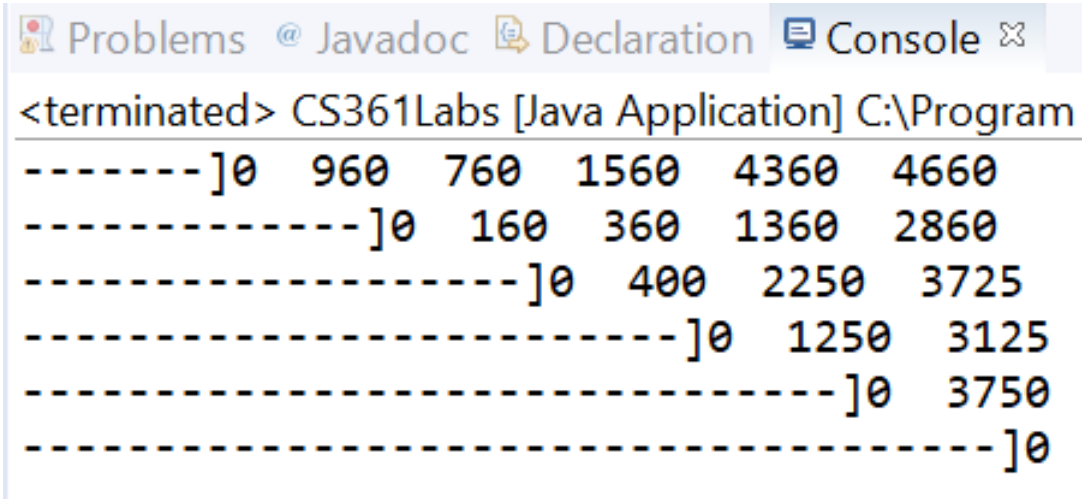
2. Implement the memoization version of MCM algorithm, using -1 for ∞. Show commented code.

```
19    //A test field that will indicate weather the matrix has been made.
20    private static boolean matrixMade = false;
21
22
23
24    /************************************************************ LAB 3 ***********************************************
25    /**
26     * This in my memoization matrix chain multiplication method for lab 3
27     * of CS 361 I was helped with this code from GeeksForGeeks.com
28     * http://www.geeksforgeeks.org/dynamic-programming-set-8-matrix-chain-multiplication/.
29     * Slite changes have been mad by Nathan Stark.
30     *
31     * @param p is the array of the sizes of the matrixes.
32     * @param i is the lower bound index.
33     * @param j is the upper bound index.
34     * @return the value of the work needed to multiply the matrices.
35     */
36    public int matrixCainMemo(int[] p, int i, int j){
37
38        if(!matrixMade){
39            eno = new int[j + 1][j + 1];            // Initialize the matrix.
40            matrixMade = true;                      // Once the matrix in made don't make another.
41        }
42        if(i==j){                                   // Check to see if you are on the diagonal
43            return eno[i][j] = 0;                   // If you are set the value to 0 and return the value.
44        }
45        int min = -1;                               // Set the min to an impossible value so that it is easily checked for.
46
47        for(int k=i;k<j;k++){
48            int count = matrixCainMemo(p,i,k) +     // Do the math using recursive calls.
48            int count = matrixCainMemo(p,i,k) +     // Do the math using recursive calls.
49                    matrixCainMemo(p,k+1,j) + p[i-1]*p[k]*p[j];
50            if((min == -1) || (count<min)){         // Check the min to see if it makes sense and check in the count is less than the min.
51                min=count;                          // Change the min to the count value.
52                eno[i][j]=min;                      // Set the value at the ith and jth place.
53            }
54        }
55        return min;                                 // Return the min.
56    }
```

3. Show the output for your DP version of MCM algorithm for p being < 30, 4, 8, 5, 10, 25, 15>, including where the parenthesis should be located.



4. Show the output for your memoization version of MCM algorithm for p being < 30, 4, 8, 5, 10, 25, 15>, including where the parenthesis should be located.



5. Implement a breadth first search using an adjacency list. Show commented code.

```
21      // The number of vertices.
22      private  int V;
23      // The adjacency list.
24      private LinkedList<Integer> adj[];
25
26
27
28      /************************************************************ LAB 3 ***************************************
29
30      /**
31       * This is the setting up method for the graph the be traversed.
32       * This code is found on GeeksForGeeks.com
33       * https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
34       *
35       * @param vert the number of vertices there are in the given graph.
36       */
37      @SuppressWarnings("unchecked")
38      public void graph(int vert){
39
40          V = vert;
41
42          adj = new LinkedList[vert];                     //Set the adjacency list to have vert number of vertices
43          for(int i=0;i<vert;i++){
44              adj[i] = new LinkedList<Integer>();         // Initialize the linkedLists into the adj.
45          }
46      }
47
48      /**
49       * The method to add an edge in in the graph.
50       * This code is found on GeeksForGeeks.com
51       * https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
52       *
53       * @param vert which vertex the edge is getting a set to.
54       * @param w the other vertex to which will be connected.
55       */
56      public void addEdge(int vert, int w){
57          adj[vert].add(w);
58      }
59
60      /**
61       * This is the breath first search using the adjacency list.
62       * This code is found on GeeksForGeeks.com
63       * https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
64       *
65       * @param s the root of the breath first search.
66       */
67      public void bfsAdjLinked(int s){
68
69          boolean visited[] = new boolean[V];                 // By default all values will be set to false a loop could used but is unneeded.
70
71          LinkedList<Integer> queue = new LinkedList<Integer>();  // The queue for the bfs is initiated.
72
73          visited[s]=true;                                    // The first/root vertex is visited.
74          queue.add(s);                                       // The first/root in now in the queue.
75
76          while (queue.size() != 0)
77          {
78              s = queue.poll();                               // Take the fist vertex out of the queue and print it
79              System.out.print(s+" ");
80
81              // Get all adjacent vertices of the dequeued vertex s
```

```
79              System.out.print(s+" ");
80
81              // Get all adjacent vertices of the dequeued vertex s
82              // If a adjacent has not been visited, then mark it
83              // visited and enqueue it
84              Iterator<Integer> i = adj[s].listIterator();
85              while (i.hasNext())
86              {
87                  int n = i.next();
88                  if (!visited[n])
89                  {
90                      visited[n] = true;
91                      queue.add(n);
92                  }
93              }
94          }
95      }
96
```

6. Implement a breadth first search using an adjacency matrix. Show commented code.
7. Show the output for the bfs using the adjacency list on the graph below.
   Let it be noted that I am converting the alphabetical letters to numeric values. Therefore, letting a=0 and so on. Thus, my results are as follows a, b, d, c, e, f, g, n, m, h, j, i, k, and l.

```
<terminated> CS361Labs [Java Application] C:\Prog
0 1 3 2 4 5 6 13 12 7 9 8 10 11
```

8. Show the output for the bfs using the adjacency matrix on the graph below.