CS317: Algorithms — Homework Assignment #3

Due: See Canvas for Assignment Due Dates (20 points)

NOTE: NO LATE ASSIGNMENTS WILL BE ACCEPTED because we often review them in class on the due date.

Please upload the document containing your answers. They can be handwritten and scanned, but they must be clearly legible to receive a grade on the assignment. PDF is the best format for canvas. You DO NOT Need to include this cover sheet in your upload. It is formatted for the grader to use for me, as needed.

Chapter 3: Work the following problems. Point values are provided for each problem.

Problem #	Points	Grader's Notes
Sec 3.2, #5 pg 1	2	*Read section on Brute-force pattern matching*
Sec 3.3, #1 pg 2	1	
Sec 3.3, #2 pg 2	1	
Sec 3.3, #4b pg 2	1	
Sec 3.4, #7 pg 2	3	
Sec. 3.4, #9a-c pg 3	3	
Sec 3.5, #1 a-b pg 3	4	
Sec 3.5, #4 pg 3	2	

Chapter 4: Work the following problems. Point values are provided for each problem.

Problem #	Points	Grader's Notes
Sec 4.1, #4 pg 3	3	

Other ungraded practice problems:

Sec 3.1: #7, #9, #13

Sec 3.3: #6, #7

Sec 3.5: #2, #7

Sec 4.1: #12

3.2

5. How many comparisons (both successful and unsuccessful) will be made by the brute-force algorithm in searching for each of the following patterns in the binary text of one thousand zeros?

a. 00001 **b.** 10000 **c.** 01010

a) Looking for 00001 in 1'000 zero's

(000... 000) length of 1000

00001 length of 5

1000 - 5 + 1 = 996 iterations

First 4 will be successful = 996x4 = 3984

Last 1 will be unsuccessful = 996x1 = 996

total = 4980

b) Length of 5, 1000-5+1 996 iterations

First, unsuccessful = 996x1 = 996

∴ no successful comparisons

c) 1000-5+1 = 996 iterations

First successful: 996x1 = 996

Second unsuccessful: 996 x1 = 996

996+996 = 1992

3.3

1. Assuming that *sqrt* takes about 10 times longer than each of the other operations in the innermost loop of *BruteForceClosestPoints*, which are assumed to take the same amount of time, estimate how much faster the algorithm will run after the improvement discussed in Section 3.3.

Original Algorithm: $\min(d, \text{sqrt}((x_i - x_j)^2 + (y_i - y_j)^2))$

$\left.\begin{array}{l} \to 1 \text{ min} \\ \to 2 \text{ sub} \\ \to 2 \text{ square} \\ \to 1 \text{ sqrt} \end{array}\right\}$ $1 + 2 + 2 + 10 \times 1 = 15$

↳ 10 times longer

W/out sqrt: just 5 units.

3x faster!

2. Can you design a more efficient algorithm than the one based on the brute-force strategy to solve the closest-pair problem for n points $x_1, x_2, \ldots, x_n$ on the real line?

ALGORITHM *BruteForceClosestPair*(P)

//Finds distance between two closest points in the plane by brute force
//Input: A list P of n ($n \geq 2$) points $p_1(x_1, y_1), \ldots, p_n(x_n, y_n)$
//Output: The distance between the closest pair of points
$d \leftarrow \infty$
for $i \leftarrow 1$ **to** $n - 1$ **do**
  for $j \leftarrow i + 1$ **to** n **do**

return d

$d \leftarrow \min(d, \text{abs}(x_i - x_j) + \text{abs}(y_i - y_j))$

Use the manhattan distance formula. essentially replaces squaring and square root with absolute values.

4. a. There are several alternative ways to define a distance between two points $p_1(x_1, y_1)$ and $p_2(x_2, y_2)$ in the Cartesian plane. In particular, the ***Manhattan distance*** is defined as

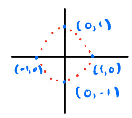
$$d_M(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|.$$

Prove that d_M satisfies the following axioms, which every distance function must satisfy:

i. $d_M(p_1, p_2) \geq 0$ for any two points p_1 and p_2, and $d_M(p_1, p_2) = 0$ if and only if $p_1 = p_2$
ii. $d_M(p_1, p_2) = d_M(p_2, p_1)$
iii. $d_M(p_1, p_2) \leq d_M(p_1, p_3) + d_M(p_3, p_2)$ for any p_1, p_2, and p_3

b. Sketch all the points in the Cartesian plane whose Manhattan distance to the origin (0, 0) is equal to 1. Do the same for the Euclidean distance.

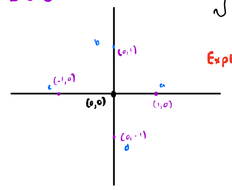
Manhattan $\text{abs}(x_1 - x_2) + \text{abs}(y_1 - y_2)$



All the red and blue points are approximately 1 unit away from the origin

Euclidian

$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} = 1$



Explanation: a, b, c, and d are all one unit away. There are less points because that is now squaring fractions works

3.4

7. Consider the ***clique problem***: given a graph G and a positive integer k, determine whether the graph contains a ***clique*** of size k, i.e., a complete subgraph of k vertices. Design an exhaustive-search algorithm for this problem.

Step 1: Subset "S" is to be "k" sized subset

Step 2: Search ↳ for an edge for each pair of vertices S.

Step 3: If it cannot find an edge go back to step 1 to find k subset else step 4

Step 4: Stop and return successful edge.

9. *Eight-queens problem* Consider the classic puzzle of placing eight queens on an 8×8 chessboard so that no two queens are in the same row or in the same column or on the same diagonal. How many different positions are there so that

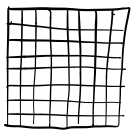
a. no two queens are on the same square?

b. no two queens are in the same row?

c. no two queens are in the same row or in the same column?

Also estimate how long it would take to find all the solutions to the problem by exhaustive search based on each of these approaches on a computer capable of checking 10 billion positions per second.

a. no two queens are on the same square?

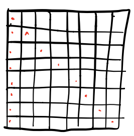


64 possible locations, 64-8= 56 blank spots. 8 queens.

$$\frac{64!}{56!\,8!} = 4,426,165,368$$ positions

b. no two queens are in the same row?

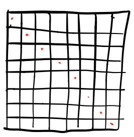
doesnt work LOL! ->



64 positions, when we place one we have 7x8 board = 56...

64+56+48+40+32... = 288 positions

c. no two queens are in the same row or in the same column?



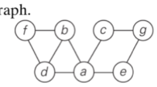
Same as b, but row and column gone.

64 + 49... = 204 positions

8x8 7x7

3.5

1. Consider the following graph.

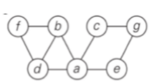


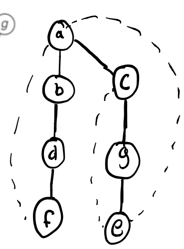
a. Write down the adjacency matrix and adjacency lists specifying this graph. (Assume that the matrix rows and columns and vertices in the adjacency lists follow in the alphabetical order of the vertex labels.)

	a	b	c	d	e	f	g
a	0	1	1	1	1	0	0
b	1	0	0	1	0	1	0
c	1	0	0	0	0	0	1
d	1	1	0	0	0	1	0
e	1	0	0	0	0	0	1
f	0	1	0	1	0	0	0
g	0	0	1	0	1	0	0

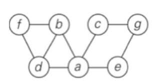
a →	d b c e
b →	f d a
c →	a g
d →	a b f
e →	a g
f →	b d
g →	c e

b. Starting at vertex *a* and resolving ties by the vertex alphabetical order, traverse the graph by depth-first search and construct the corresponding depth-first search tree. Give the order in which the vertices were reached for the first time (pushed onto the traversal stack) and the order in which the vertices became dead ends (popped off the stack).

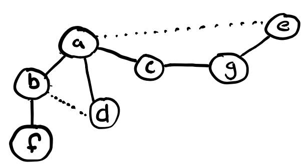




	Push	Pop
a	1	7
b	2	3
d	3	2
f	4	1
c	5	6
g	6	5
e	7	4



4. Traverse the graph of Problem 1 by breadth-first search and construct the corresponding breadth-first search tree. Start the traversal at vertex *a* and resolve ties by the vertex alphabetical order.



4.1

4. Design a decrease-by-one algorithm for generating the power set of a set of n elements. (The power set of a set S is the set of all the subsets of S, including the empty set and S itself.)

Power Set $A = \{1, 2, 3\}$

$P(A) = \{\{\emptyset\}, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

If n=0
  return empty set

Recursion list of sub sets

$T(N-1) = \text{PowerSet}(\{a_1 \ldots a_n\}, n-1)$

$T(N) = \{a_n\} \cup \{a_n\ T(n-1)\}$ -> append a_n to each $T(n-1)$

return $T(N)$ Return subset