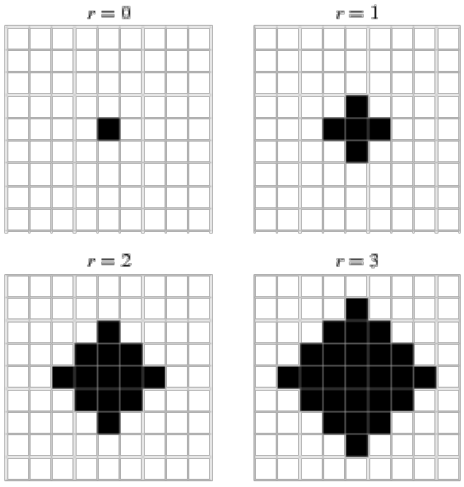
CS350 — Spring 2013
Homework 2

Due Tuesday 23rd April, on paper, at the start of class. This assignment will be graded.

1. **von Neumann neighborhood.** Recall the definition of von Neumann neighborhood from Levitin Ex 2.3, problem 11.



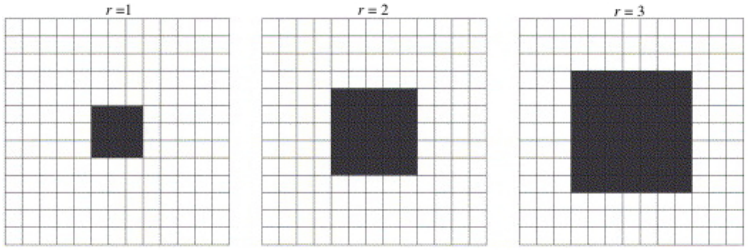
On the n^{th} iteration, n squares are *added* to each of the four symmetric sides of the von Neumann neighborhood. Hence we obtain the following recurrence for $S(n)$, the total number of squares in the neighborhood after the n^{th} iteration:

$$S(n) = S(n-1) + 4n \quad \text{for } n > 0 \text{ and } S(0) = 1.$$

Solve this recurrence relation using backwards substitution to find a closed-form formula for the number of cells in the von Neumann neighborhood of range n.

Hint: Make sure that the working for your solution contains at least one instance of the back-substitution. If it's at all unclear where a line of your derivation comes from, add a comment, such as "replace n by $n-1$", or "substitute for $S(n-1)$".

2. **Moore neighborhood**. The Moore neighborhood is similar to the von Neumann neighborhood, except that on each iteration, squares are added on the diagonals as well. The starting point (0^{th} iteration) is the same; the first three iterations in the development of the Moore neighborhood are shown here:



Write down a recurrence relation for the total number of squares in the Moore neighborhood after the n^{th} iteration. Also write down the initial condition.

Hint: Be sure to describe your reasoning.

3. **Fibonacci**. Levitin §2.5 gives two algorithms for computing the Fibonacci numbers. Algorithm $F(n)$ uses $\Theta(n)$ time; Algorithm $Fib(n)$ uses $\Theta(n)$ time, but also uses $\Theta(n)$ space.

Improve algorithm $Fib(n)$ so that it uses only $\Theta(1)$ space.

Hint: Write out your complete improved algorithm in the same style of presentation as is used by Levitin for Algorithm $Fib(n)$. Comments should not be necessary if you use decretive names for the variables.

4. **Brute-force Exponentiation.** Write down, using Levitin's pseudocode, a simple, brute-force algorithm for computing a^n by multiplying a by itself n times.

What is the efficiency of this algorithm as a function of n? Derive its efficiency as a function of the number of bits, b, in the binary representation of n.

Hint: If your algorithm uses a loop, then your derivation should be in the form of a summation; if your algorithm is recursive, your derivation should start with a recurrence relation. When transforming from n to b, recall the formula for the number of bits in the binary representation of a number from Levitin Equation 2.1.

5. **Distances.** There are several ways to define a distance between two points p and q. In particular, the *Manhattan distance* is defined as

$$d_M(p, q) = |p.x - q.x| + |p.y - q.y|$$

(a) Prove that d_M satisfies the following axioms that every distance function must satisfy:

i. $d_M(p, q) \geq 0$ for any two points p and q, and $d_M(p, q) = 0$ if and only if $p = q$;

ii. $d_M(p, q) = d_M(q, p)$; and

iii. $d_M(p, r) \leq d_M(p, q) + d_M(q, r)$ for any p, q and r

Hint: The questions says "Prove", so be sure to write a formal proof, using the definition of d_M.

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

(c) True or false: the answer to an instance of the closest-pair problem does not depend on which of the two metrics — d_E (Euclidean) or d_M(Manhattan) — is used.

Hint: Justify your answer, for example, by providing a counter-example if you said "false".