COMP20003 Algorithms and Data Structures

Worksheet 3

[week starting 8 of August]

Second (Spring) Semester 2016

Overview

The workshop for Week 3 will start with a mini-tutorial on computational complexity and static/dynamic arrays.

Tutorial Questions

Question 2.1 Given the following functions $f(n)$ and $g(n)$, is f in $O(g(n))$ or is f in $\Theta(g(n))$, or both?

	$f(n)$	$g(n)$
(a)	$n + 100$	$n + 200$
(b)	$log_2(n)$	$log_{10}(n)$
(c)	2^n	2^{n+1}
(b)	2^n	3^n

Question 2.2 big-O notation gives an upper bound for a function. In Computer Science, we often loosely use big-O to mean the `least` upper bound.

First of all: make sure you can describe the difference between `any` upper bound and the `least` upper bound. big-Θ(theta) is an exact bound: the function is bounded from below and from above by g(n).

The following table uses big-O notation and big-Θ notation to approximate the running time of algorithms. Given the descriptions of the run time in the two left hand columns, what can you say, in each instance, about the relative performance of the two algorithms when:

1. big-O is used in the usual Computer Science sense, as the least upper bound;
2. big-O is used in its strictest sense to mean any upper bound

Algorithm 1	Algorithm 2	Relative Performance: CS sense of big-O	Relative performance: strict sense of big-O
$O(n \ log \ n)$	$O(n^3)$		
$\Theta(n \ log \ n)$	$O(n^3)$		
$O(n \ log \ n)$	$\Theta(n^3)$		
$\Theta(n \ log \ n)$	$\Theta(n^3)$		

Question 2.3 What is the difference between the two following declarations?

```c
int a[10][20];
```

```c
int *b[10];
```

1. How could you use them both as 2-dimensional arrays? (write the code)
2. what advantages might there be to declaring an array like `*b[]` above, instead of like `a[][]` above?
3. how could you make a variable declared as `int **c` into a 2-dimensional array? (write the code)

Question 2.4 Give a characterization, in terms of big-O, big-Ω and big-Θ, of the following loops:

```c
int p = i;
for(int i = 0; i < 2*n; i++){
     p = p * i;
}
```

```c
int s = 0;
for(int i = 0; i < 2*n; i++){
     for(int j = 0; j < i; j++){
          s = s + i;
     }
}
```

Programming exercises

Programming 2.1 In Mathematics, the **powerset** of any set S, is the set of all subsets of S including the empty set and S itself.

For example, the powerset of the set {A,B,C} is {{}, {A}, {B}, {C}, {A,B}, {A,C}, {B,C}, {A,B,C}}. We often write the powerset as a boolean array, where each element in the original set has its own column, each row represents a subset of the powerset, where 1 means "in the subset", and 0 means not "in the subset". For example, the power set of {A,B,C} could be written:

```
000 /* empty set */
100 /* only A */
010 /* only B */
001 /* only C */
110 /* only A,B */
...
111 /* A,B,C */
```

Your task in this laboratory is to write a small C program that takes as input a number representing the alphabet size (in the example above with A,B,C, the alphabet size is 3), and outputs the boolean matrix representing the powerset.

The program will give you experience using 2-dimensional matrices, dynamic memory allocation, and the argc/argv arguments to main() that are used for passing command line arguments to the program.

Your program should take the alphabet size as a command line argument argv[1], and you will allocate memory for the 2-dimensional array dynamically. (One dimension is clearly the alphabet size; you will need to figure out what the other dimension is before you allocate the memory. Hint: you might find the C library function `pow()` useful – you will need to `#include<math.h>` and compile with the -lm option to include the math library).

Because the main point of this exercise is for you to gain experience with memory allocation and command line arguments, rather than with generating powersets, in the first instance you should aim to allocate an array based on a command line argument, fill it with something simpler than the powerset, e.g. all zeros, and print it out. Try to print it out reversing columns and rows. Finally you just have to compute the powerset and print it out.

August 5, 2016