UIT2201: CS & the IT Revolution
Tutorial Set 2 (Fall 2016)
[Note changes made to T2-Q2]
(D-Problems discussed on Friday, 19-Aug-2016)
(Q-Problems due on Tuesday, 23-Aug-2016, before 5pm, @COM1-03-17)

Practice Problems: (not graded)

I have included a number of practice problems for you to try out. These problems will help you to "ease into" the materials covered in the course. *(If you have difficulties with these practice problems, please **quickly** see your classmates or the instructor for help.)*

T2-PP1: [Your first Scratch program]
(a) After downloading Scratch (See T1-D2), write your first Scratch program that will make the cat (the *default* sprite/actor) "move around, change costumes, change colour, and say Meow" a few times.
(b) Learn how to add a *new random* sprite, and also choose your own new sprite.
(Hint: See the "New Sprite:" just below the stage and above the sprite panel and experiment with it.)
(c) Learn to add a different background. Inside the sprite panel, click on stage (which is also "a sprite"), then under the "New background:", choose "Import" to import one of many, many pre-designed background. You can also import your own custom background.

T2-PP2: [Doing-something-many-times in Scratch program]
After you had *fun* with your first Scratch program, now is time of get more serious. Instead of just randomly moving around, how about getting the cat to "do something many times", such as doing the following 18 times: (move forward 10 steps; change costume, say "Meow" for 0.5s;)
(**Note:** You can add the "commands-blocks" 18 times, but there is a *simpler* way. *Find* & use it!)

Discussion Problems: -- Prepare (individually) for tutorial discussion.

T2-D0: Read [SG] Ch. 1 & 2

T2-D1: [Having *FUN* with Scratch]
(a) After you have downloaded Scratch, you can now goto the UIT2201 mini-page on Scratch and download some of the demo Scratch programs and some of those that implements algorithms covered in the class.

The UIT2201 Mini-Page on Scratch -- || here ||

- A first Simple Scratch program -- || forward-backward.sb -- (*Learn how each cmd works.*)
- **Some Simple HowTo Scratch Introductions** -- look through all of them quickly. (You will outgrow these.)
- **Some sample Scratch programs** -- look through them. Learn how they work.
  Look first at BreakDanceWithComments.sb, BeachCrab.sb
- **Some sample Demo Scratch Cards** -- look through all of them quickly.
  But, look especially at 03_keymoves_v14.pdf

(b) And there are some Scratch program that implements algorithms covered in UIT2201:

- sum-1-to-5-no-loop.sb -- (This Scratch program computes the sum of 1+2+3+4+5. It does it the straightforward way, without using loops! This works, but is not scalable for large sums such as (1+2+3+. . . + 99 + 1000).
- sum-1-to-5-loop.sb -- (This Scratch program computes the sum of 1+2+3+4+5. It does it with a loop, like the algorithm in the lecture notes. This works and is *scalable* for large sums such as (1+2+3+. . . + 99 + 1000).
- sum.sb -- (The Scratch program computes sum of 1..100 and is more concise.)

Examine these Scratch programs carefully to see how they work. Make small changes (by modifying the Scratch blocks) and see what happens (you should save as your own versions). You can also play around with the other Scratch blocks to see what they do. Try combining some of the blocks.
Also, try right-clicking the different "objects" to see what other functionalities are available. Try them out to see how they work.

T2-D2: (Algorithm: Step by Step Instruction)
Write out an informal algorithm for going from the Dean's Office of your home faculty to the USP-JAR (Joint Appointee Room in the USP) Room 02-09. The USP-JAR has the following "logo" on the door.
[**Note:** Your algorithm should be given in a mixture of precise English instructions and instructions similar to those found in the algorithm for "sum of 1 to 100" in lecture notes or like that in Figure 1.1 of [SG]. You may need to carefully *define* your own "primitive operations".]

T2-D3: (Computing Device)
Consider one computing system that you encounter in your everyday life and that has *not* been discussed in the lectures (preferably one related to *your own discipline*). Discuss how it might have been designed and note the *similarities* and *difference* with other examples we considered in class.

T2-D4: (All kinds of graphs or networks)
Graph model (also called networks) are used in many different discipline to model all kinds of things. Do a search and find a problem (not from CS-IT-Engg field) that has been modeled and solved as a graph-related problem (it does not have to be graph coloring problem).

Problems to be Handed in for Grading by the Deadline:

(**Note:** Please submit *hard copy* to me. Not just soft copy via email.)

T2-Q2: (5 points) (Algorithm Odd-13-to-5039) [Updated: Odd, not even]
In the lecture notes, algorithm **Sum-1-to-100** finds the sum of all the integers from 1 to 100, namely, 1 + 2 + 3 + ... + 99 + 100.

(a) Modify this algorithm *very slightly* to get Algorithm **Odd-13-to-5039** that finds the sum of all the odd numbers from 17 to 5039 (inclusive), namely, 13 + 15 + 17 + ... + 5037 + 5039.
(Note: Please make *only small changes* to the original algorithm. Also explore other ways, creative ways to find the solution / solve the same problem.)
[**FUN** Curiosity-Question: Why 13 and 5039?]

T2-Q1: (10 points) (Cat moving...)
Write a Scratch program that starts with the instructions given in Scratch card 03_keymoves_v14.pdf. Then add more features to your program.

- whenever "*space bar*" is pressed, make the cat "walk a 40x40 diamond", and then say "Meow" whenever the "space bar" is pressed.
- Make it return to the "*base*" (position (0,0)) when "b" key is pressed.
- Make it *change* the background when "c" key is pressed.
- Add another two more function (of your choice) to it.
- When the "green flag" is clicked, have an introduction segment where the cat will say what *your two additional functions* are, and how to activate them (and so that I will know about it).
- Beyond this, you are free to make it MORE FUN. Be creative.

IMPORTANT: Name your Scratch program according to the following *standard naming convention*: `T2-Q1-Yourname.sb`, where you should replace `YourName` with your own own name as it appears on IVLE (separated by "-"), for example `T2-Q1-Leong-Hon-Wai.sb`. Then submit your program into the appropriate IVLE Workbin.
(**You WILL LOSE MARKS if your Scratch program is given other names.**)

Adding More FUN to your Scratch Programs: What I give above is *merely the minimum requirements*. Do feel free to *use your creativity and imagination* to make your Scratch program do MORE, make it more FUN.)

T2-Q3: (10 points) [Having an IT-enabled frame of mind] (a continuation of T1-D2.)
Imagine that you are acting as an IT-aware CEO (*not* CTO) and you want to implement an e-Greeting card web-site. Write down roughly (and based on your current know-how) the "*feature list*" (or functionality, look-and-feel, human-computer-interface, etc) you would like to have in your own e-Greeting card web-site.
(Examples of features are (i) "ability to send reminder if the recipient has not viewed the card in 3 days", (ii) convenient way for recipient to reply to sender upon viewing the card".)

Don't worry about *actual implementation*. Assume you will hire people with the right expertise later. Don't worry about the *how*, focus on the *what*, remember ITEM, the *IT-Enabled Mindset*.

T2-Q4: (15 points) [Greedy Heuristic Algorithms for Graph Colouring]
In class, we showed several colourings of the conflict graph, but we did not explain *how* the colouring step was done. (The examples in lectures were done in an ad-hoc fashion, not via a proper algorithm.)

Now we want a systematic algorithm for this colouring step. One class of *graph coloring algorithm* does the following:

> It picks a colour (#1 say) and tries to colour as many vertices as it can with that colour, until we cannot colour any more vertex. Then it repeats with a new colour on the remaining uncolored vertices. And it keeps repeating (with new colours, as necessary) until all the vertices are colored.

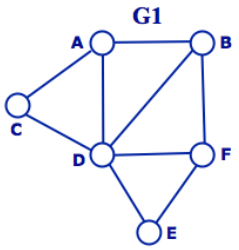
Next, we specify the step (or *algorithm) to colour as many vertices as possible using one colour*. Researchers have put forth many algorithms for this step, but none can guarantee an *optimal colouring* of the graph. We list three simple *greedy heuristic* algorithms.

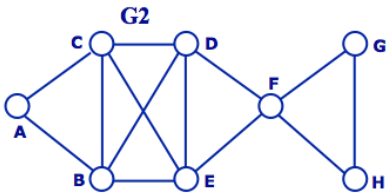
[**Note: In all three heuristic,** after colouring a vertex v, we remove (from the graph) vertex v and all those vertices connected to v. Whenever we remove a vertex w, we also remove all the edges connect to w. Hence, degree of a vertex *changes dynamically* during the coloring process.]

(i) Alphabetic order heuristic: Whenever there is a choice of several vertices (to colour), it chooses the *smallest vertex in alphabetical order*.

(ii) Smallest-degree first heuristic: Whenever there is a choice of several vertices (to colour), it chooses the vertex with the smallest degree (smallest number of edges connected to it) and in the case of a tie, it chooses in alphabetical order.

(iii) Largest-degree first heuristic: Whenever there is a choice of several vertices (to colour), it chooses the vertex with the largest degree (largest number of edges connected to it) and in the case of a tie, it chooses in alphabetical order.





Given the graph G1 shown below, an *optimal solution* is given by
C_1:{D}, C_2:{B,C,E}, C_3:{A,F}, that uses 3 colours.
If we use the *alphabetic order heuristic*, then we get the coloring given by
C_1:{A,E}, C_2:{B,C}, C_3:{D}, C_4:{F}, that uses 4 colours.
If we use the *smallest-degree first heuristic*, then we get the coloring given by
C_1:{C,E,B}, C_2:{A,F}, C_3:{D}, that uses 3 colours.
(*Here, the vertices are listed in the order that they are colored.*)
If we use the *highest-degree first heuristic*, then we get the coloring given by
C_1:{D}, C_2:{A,E}, C_3:{B,C}, C_4:{F}, that uses 4 colours.

(a) For the graph G2 given above, give an optimal coloring.

(b) For the graph G2 given above, give the coloring obtained by applying all the three heuristic algorithms given above. (*Note: In your answer, list the vertices in the order that they are colored.*)

A-Problems: OPTIONAL Challenging Problems for Further Exploration
A-problems are usually *advanced* problems for students who like a challenge; they are optional. No deadline for A-problems -- try them if you are interested and turn in your attempts.

A2: [Black and White Colouring]
Give an algorithm that will determine if a given graph can be coloured with two colours, black and white, and, if so, give such a black-and-white colouring.

A3: [Better Colouring Heuristic]
The heuristic algorithms for coloring given in T2-Q4 are quite simple. They do not give very good results for large graphs. Can you think up better heuristic algorithms for graph coloring? Try to give some informal justification for why you think they may be better than those in T2-Q4.

UIT2201: CS & IT Revolution; (Fall 2016); LeongHW