`tag` — Version 1.0.1 ***due:*** *Mon, Oct 29th @ 11:59pm*

**Disclaimer:**
We will not grade non-compiling code.

# 1 Introduction

You have recently been hired as a TA for the course Parallel Cats and Data Structures at the prestigious Carnegie Meow-llon University. There is a lot of work that you need to do in order to keep this course afloat. Obviously, most of this work will come in the form of writing and reasoning with graph algorithms. Fortunately, your time in 15-210 has prepared you well for this.

In this assignment you will identify graph bridges, find maximal independent sets, come up with an approximate solution to the exam-scheduling problem, and read lots of bad cat puns. (Strictly speaking, that last one is optional; you won't receive any point deductions for being a sourpuss.)

## 1.1 Submission

Submit your solutions by placing you solution files in your handin directory, located at

`/afs/andrew.cmu.edu/course/15/210/handin/<yourandrewid>/assn6/`

Name the files exactly as specified below. You can run the check script located at

`/afs/andrew.cmu.edu/course/15/210/bin/check/06/check.pl`

to make sure that you've handed in appropriate files. Do not submit any other files.

Your written answers must be in a file called `hw06.pdf` and must be typeset. You do not have to use LaTeX, but if you do, we have provided the file `defs.tex` with some macros you may find helpful. This file should be included at the top of your `.tex` file with the command `\input{<path>/defs.tex}`.

For the programming part, the only files you're handing in are

`Bridges.sml`
`BridgesTest.sml`
`SequenceMIS.sml`
`SequenceMISTest.sml`
`Coloring.sml`
`ColoringTest.sml`
`Schedule.sml` (*extra credit*)

These files must contain all the code that you want to have graded for this assignment and must compile cleanly. If you have a function that happens to be named the same as one of the required functions but does not have the required type, it will not be graded.

### 1.2 Style

As always, you will be expected to write readable and concise code. If in doubt, you should consult the style guide at `http://www.cs.cmu.edu/~15210/resources/style.pdf` or clarify with course staff. In particular:

1. **Code is Art.** Code *can* and *should* be beautiful. Clean and concise code is self-documenting and demonstrates a clear understanding of its purpose.
2. **If the purpose or correctness of a piece of code is not obvious, document it.** Ideally, your comments should convince a reader that your code is correct.
3. **You will be required to write tests for any code that you write.** In general, you should be in the habit of thoroughly testing your code for correctness, even if we do not explicitly tell you to do so.
4. **Use the check script and verify that your submission compiles.** We are not responsible for fixing your code for errors. If your submission fails to compile, you will lose significant credit.

## 2 Bridge Finding

On your way to lecture, a professor stops you in the hallway. He introduces himself as a cat-strophysicist and asks for help solving a tricky computational problem that has been purr-plexing his research group.

He is writing code for a cat-body simulation (a much cuter variant of the N-body simulation used by normal astrophysicists). In such a simulation it is not uncommon to track upwards of $3072^3$ cats across hundreds of time steps. In order to study large-scale feline structure in the midst of this deluge of data, cat-strophysicists will group large clumps of cats together into objects called "halos".

Your professor friend has attempted to identify halos by forming a graph in which pairs of cats separated than some distance less than $\delta$ are connected. The forest of graphs is then reduced so that all connected particles can be identified as a single halo (this approach is somewhat confusingly called the Friends of Friends or FoF algorithm). However, he laments, if two separate, massive halos are connected by a thin filament of closely connected cats, FoF will register both those halos as a single entity. Clearly, this is an inescapable flaw in FoF! "Nonsense," you say, "that thin filament of cats is a graph-bridge and can be identifed relatively quickly."

Let $(u, v) \in E$ for some undirected graph $G = (V, E)$ be given. $(u, v)$ is a *bridge* if it is not contained in any cycles (equivalently, if a bridge is removed there will no longer be a path which connects its endpoints). All code for the first two tasks in this section should go in the functor `Bridges` in the file `bridges.sml`.

**Task 2.1** (5%). Define the type `ugraph` representing an undirected graph and write the function `makeGraph (edges: (int * int) seq): ugraph` which takes in a sequence representing the edges of an *undirected* graph and returns that same graph under your `ugraph` representation. You may assume that `edges` is somewhat sane, i.e. that `edges` contains no self-loops or duplicated elements and that no node is labeled by an integer larger than $|V| - 1$ or smaller than 0. Your graph should not contain any vertices which are not explicitly in `edges`. `makeGraph` should have $O(|E| \log |V|)$ work and $O(\log^2 |E|)$ span.

You will receive no points for this task if you do not also include a comment explaining why you chose to represent `ugraph` in the way that you did. This comment does not need to be a novel, but it must be sufficiently detailed that a TA can understand your `ugraph` representation from the comment and type signature alone.

You will need to implement `makeGraph` several times throughout this assignment (possibly with different underlying representations). This will be the only time that you receive points for it.

**Task 2.2** (20%). Implement the function

```
findBridges (G: ugraph):  (int * int) seq
```

It should take in an undirected graph and return a sequence containing all the edges of $G$ which are bridges (and only those edges). `findBridges` should run in $O(|E|)$ work and span.

**Task 2.3** (5%). Write the test functor `BridgesTest` for a structure subscribing to the `BRIDGES` signature in the file `BridgesTest.sml`.

- `BridgesTest` should not run any tests at compile time.
- `BridgesTest.all()` should return true if all tests pass and false otherwise. Your code should not crash just because the argument structure failed a test.
- `BridgesTest` should test a wide range of edge cases.
- `BridgesTest` should be written in such a way that boiler plate code is kept to a bare minimum. Adding test cases should be easy and repeated work should be dealt with elegantly.

It is possible to get full credit for this task even without completing the previous two tasks.

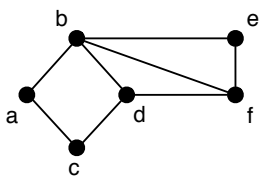
# 3 Maximal Independent Set

## 3.1 Problem Definition

In graph theory, an *independent set* is a set of vertices on an undirected graph that do not neighbor one another. More formally, let a graph $G = (V, E)$ be given. We say that a set $I \subseteq V$ is an independent set if and only if $(I \times I) \cap E = \emptyset$.

Unfortunately, the problem of finding the overall largest independent set—known as the Maximum Independent Set problem—is **NP**-hard. Its close cousin Maximal Independent Set, however, admits efficient algorithms and is a useful approximation to the harder problem.

The *Maximal Independent Set* (MIS) problem is: given an undirected graph $G = (V, E)$, find an independent set $I \subseteq V$ such that for all $v \in (V \setminus I)$, $I \cup \{v\}$ is not an independent set. Such a set $I$ is maximal in the sense that we can't add any other vertex and keep it independent, but it easily may not be a maximum—i.e. largest—independent set in the graph.



For example, in the graph above, the set $\{a, d\}$ is an independent set, but not maximal because $\{a, d, e\}$ is also an independent set. On the other hand, the set $\{a, f\}$ is a maximal independent set because there's no vertex that we can add without losing independence. Note that in MIS, we are *not* interested in computing the overall-largest independent set: while maximum independent sets are maximal independent sets, maximal independent set are not necessarily maximum independent sets! Staying with the example above, $\{a, f\}$ is a maximal independent set but not a maximum independent set because $\{a, d, e\}$ is independent and larger.

### 3.2 Algorithm and Code

You have recently assigned MIS as a homework question for the students in Parallel Cats and Data Structures. You supplied your students with the signature `MIS` in the file `MIS.sig`.

Holding your pre-deadline office hours was like herding cats! Fortunately, the students playing cat-and-mouse with the due date have finally all submitted their code and it is time to begin grading.

**Task 3.1** (5%). A student submitted the file `TableMIS.sml` as a solution to this homework. The functions in this file form a valid solution to the MIS problem. As part of the grading process, write down a concise (but careful) description of the algorithm the student uses—a hierarchical enumerated list may help tremendously here.

**Task 3.2** (15%). Although the student's table-based implementation is correct, you notice that it runs in a sequence of $O(\log |V|)$ steps in expectation, each of which takes $O(|E| \log |V|)$ work and has $O(\log^2 |V|)$ span. As a TA you must one-up your student by writing a more efficient version that uses single-threaded or normal sequences and runs in $O(|E|)$ work and $O(\log |V|)$ span per step with the same expected total number of steps.

Implement another solution to MIS in a functor `SequenceMIS` in `SequenceMIS.sml` that uses either single-threaded or normal array sequences. Be sure to carefully document the representations of any data structures you use in comments in your SML file.

You must use the same algorithm that the student used in `TableMIS.sml` and obtain the better costs by working with a different representation. You must ensure that if `SequenceMIS` and `TableMIS` are given the same graph and seed, they will return the same `int seq`.

**Task 3.3** (5%). Write a functor `MISTest` in `MISTest.sml` that thoroughly tests its argument structure against the `TableMIS` solution.

Your test battery should include some hard coded small examples, but for full credit, it must also include code to generate larger graphs as stress tests.

You will still need to observe all the restrictions outlined in task 2.2.

### 3.3 MIS Analysis

The MIS homework was a cat-tastrophe, so you plan to review it during your recitation this week. As part of this review, you want to discuss the cost of your reference solution.

Because you have taken more advanced algorithm courses, you know that there is a beautiful analysis which shows that the code in `SequenceMIS` has $O(|E|+|V|)$ expected total work and $O(\log^2 |V|)$ expected

total span on arbitrary graphs. To whet your students' appetites, you decide to analyze the performance of the algorithm for the special case of graphs where the maximum degree of any vertex is bounded by a constant.

Let $G = (V, E)$ be an input graph. Assume that the maximum degree in $G$, denoted by $\Delta$, is bounded by a constant positive integer $c_\Delta$.

**Task 3.4** (5%). Consider a single step of the MIS algorithm. Calculate the probability $p_v$ that a vertex $v$ is selected as part of the independent set during that step. $p_v$ should be written in terms of $\deg(v)$.

**Task 3.5** (10%). Show that the total cost of the MIS algorithm has $O(|V|)$ expected total work and has $O(\log^2 |V|)$ expected total span when implemented as specified.

## 4 Graph Coloring

The final exam period at Carnegie Meow-llon draws near. You have been tasked with drawing up a schedule for the exams in such a way that no student has two exams scheduled at the same time.

Exams at CM-Mew are structured very similarly to those at CMU: throughout the duration of the exam period, there are $n$ time-slots of uniform length. Any number of exams can take place during a single time slot as long as they do not lead to time conflicts. All exams last exactly as long as a single time slot and must be contained within exactly one time slot.

Formally, given a set of students $S$, each of which is taking some set of exams $X_s$, your goal is to create a sequence of time slots $T$ whose elements are sets of exams $X_t$ such that $\forall X_t \in T$, $\forall X_s \in S$, $|X_t \cap X_s| \leq 1$. The *Exam Scheduling* problem is: to find a $T$ such that $|T|$ is as small as possible. This is by no means a trivial problem (in fact it is NP-hard: 3-COLOR can be reduced to it). However during a late-night study session, you realize that the scheduling problem can be reduced to graph-coloring and that graph-coloring can be reduced to MIS. Explaining this revelation will require a bit more formalism.

Let a graph $G = (V, E)$ be given. If $C$ is a set of colors, a *coloring function* for $G$, $c_G : V \to C$, maps every vertex to a color such that adjacent vertices do not have the same color. That is to say, for all $(u, v) \in E$, $c_G(u) \neq c_G(v)$.

A graph is said to be *k-colorable* if there exists a coloring function for that graph whose range has size at most $k$. For example, every graph is $|V|$-colorable by mapping each vertex to a unique color. A star graph is 2-colorable by picking one color for the fringe vertices and another for the central vertex. The *Graph Coloring* problem is: to produce a coloring function that uses the smallest possible number of colors.

**Task 4.1** (10%). Describe a reduction from the Exam Scheduling problem to the Graph Coloring problem using the tightest cost bounds that you can. State and prove these cost bounds. While this reduction doesn't need to be overly formal, it must be precise enough to be unambiguous. You may include pseudocode if you wish, but just writing a raw SML function that implements this reduction will be insufficient.

## $(\Delta + 1)$-coloring

As some would put it, it appears that you have jumped out of the frying pan and into the fire. Although graph coloring is easier to work with, The Graph Coloring problem is also NP-complete. The best we can do at this point is an approximation of the solution. Let $\Delta$ denote the maximum degree in a graph. There is an algorithm that reduces finding a $(\Delta + 1)$-coloring to solving MIS.

**Task 4.2** (15%). Implement the function

```
graphColor (E : (vertex*vertex) seq):  (vertex*int):  seq,
```

in the structure `MISColoring` in the file `Coloring.sml`. The input sequence contains the graph's *undirected* edges and the output sequence maps every vertex to a color. For your sanity's sake, you will use the SML type `int` to represent colors. Your algorithm should run in $O(\Delta|E|\log|V|)$ work and $(\Delta\log^2|V|)$ span. Be sure to justify why your algorithm meets these bounds in `assn06.pdf`.

It is strongly suggested that you use `TableMIS` as the underlying `MIS` representation in this task.

**Task 4.3** (5%). Write a functor `ColoringTest` in `ColoringTest.sml` that thoroughly tests its argument structure. (**Note**: the distributed functor takes in an `MIS` as its argument structure instead of a `COLORING`. You should ignore this and hard-code `MISColoring` in as the tested structure.)

You will still need to observe all the restrictions outlined in task 2.2.

**Task 4.4** (5%). **(Extra Credit)** If you ignore the cost of converting the input edge seqeunce into a graph in task 4.2, it is possible to implement this function in $O(\Delta|E|)$ work and $O(\Delta\log^2|V|)$ span. To recieve full credit for this task, your implementation of `graphColor` in `Coloring.sml` should meet these stricter cost bounds, you should justify why it meets these bounds in `assn06.pdf`, and should indicate with a comment in `Coloring.sml` that you are attempting task 4.4.

If you write an $O(\Delta|E|)$ solution, you do not also need to also write an $O(\Delta|E|\log|V|)$ solution.

**Task 4.5** (10%). **(Extra Credit)** Implement the function `scheduleExams (S: exam seq seq): exam seq seq` in the structure `Schedule` in the file `schedule.sml`. Each element of `S` represents the collection of exams taken by single student. Each element of the output sequence represents a single time slot, where its elements are the exams scheduled during that time slot. No explicit time bound is given. Document and justify the work and span of your function in `assn06.pdf`. Slower asymptotic times will receive fewer points. You are not required to submit a test structure for this question.