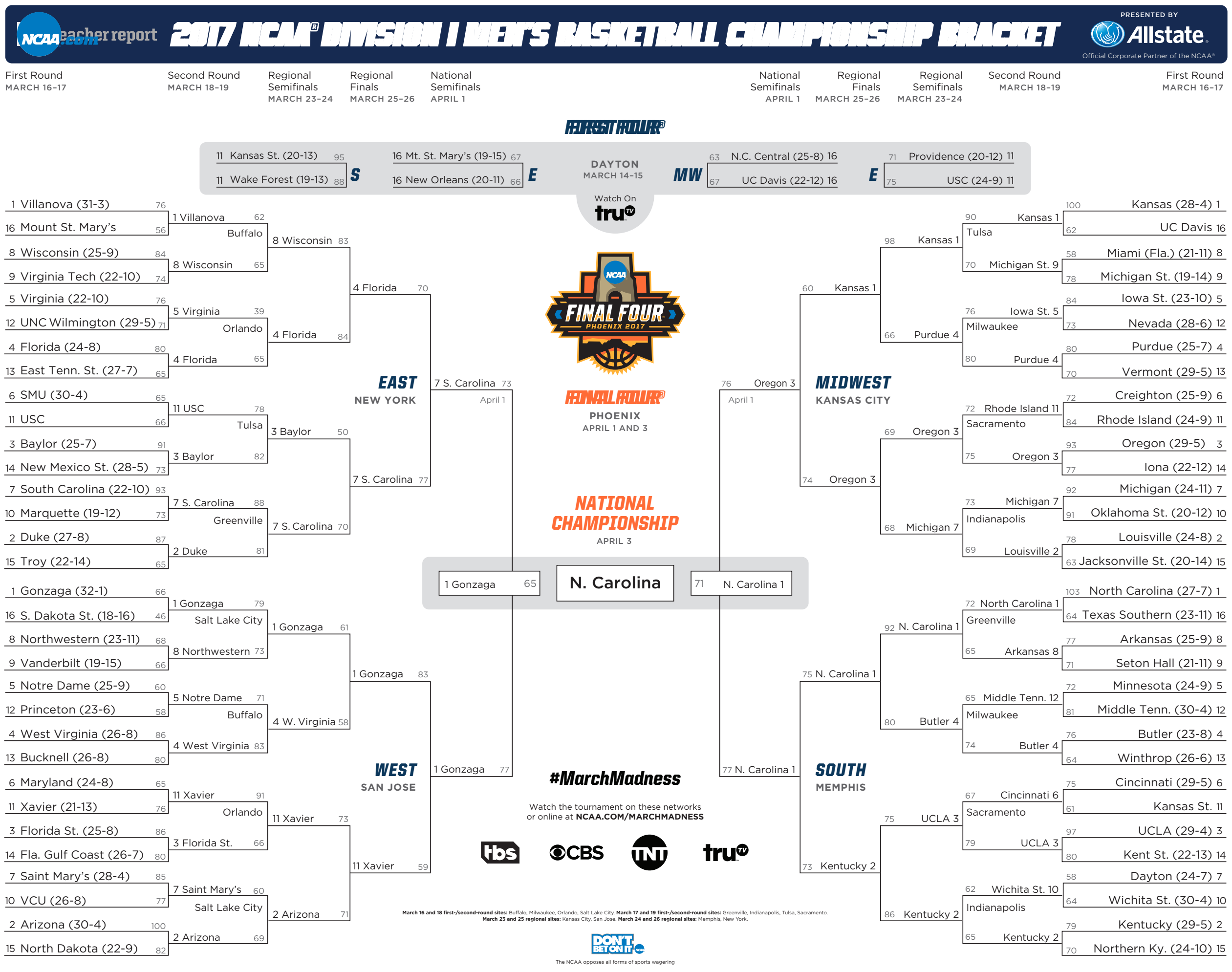
CS 1110 Spring 2017, Assignment 4: Tournaments*

Due on CMS **Thursday April 20th, 2017, 11:59pm**. As usual, we strongly recommend that you submit a preliminary version by 2pm the day of the deadline, for the usual reasons.[1]

Necessary files: download and unzip http://www.cs.cornell.edu/courses/cs1110/2017sp/assignments/assignment4/a4.zip

1 Introduction

In March and early April, *bracket* diagrams like the one depicted below[2] sprout up everywhere:



And it's not just basketball; there's brackets for Shakespeare's plays, Miramax films, Pokémon, and, of course, brackets. Brackets have a naturally recursive structure that we'll take advantage of in this assignment.

*Authors: Lillian Lee, Erik Andersen

[1]Reminders: you can replace older submissions with improved ones up to the actual deadline. Since you've been warned to submit early, do not expect that we will accept work that doesn't make it onto CMS on time — by CMS's definition of time, which may differ from your watch's or your computer's — for whatever reason, including server delays stemming from many other students trying to submit at the same time.

[2]The bracket diagram for the 2017 NCAA Men's Division I Basketball tournament. Image from http://i.turner.ncaa.com/sites/default/files/external/printable-bracket/2017/bracket-ncaa.pdf. Sorry, Zags.

Contents

2 Rules (All The Same As For A3)

This assignment, and all subsequent ones unless otherwise noted, will receive a single grade; there is no revise-and-resubmit like there was for Assignment 1.

You may do this assignment in groups of size **one or two**. All previous groups have been dissolved by CMS for this assignment; if you want to work with someone (whether or not you've worked with them before), you need to *form your group on CMS* before submitting.[6]

Reminder: If your partnership dissolves, see the course Academic Integrity description about "group divorce" on what to do.

Our policies are laid out in full on the course Academic Integrity page, but we re-state here **the main rules**: where "you" means you and, if there is one, your one CMS-registered group partner,

1. Never look at, access or possess any portion of another group's work in any form.
2. Never show or share any portion of your work in any form to anyone except a member of the course staff.
3. Never request solutions from outside sources such as online services like StackOverflow.
4. DO specifically acknowledge by name all help you received, whether or not it was "legal" according to (1)-(3).

3 Representing Tournament Outcomes

In a typical tournament set-up, competitors play against each other, and the winners of one game are allowed to proceed to the next round to play other winners. So, a tournament can be thought of as based on the outcomes of individual head-to-head games, where the outcomes in one round depend on the outcomes of the games in the previous round.

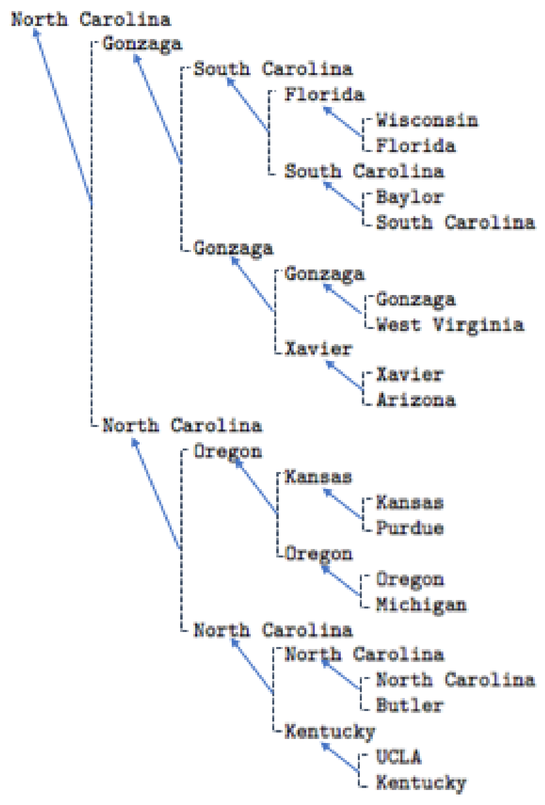
To make things concrete, we'll use Figure 1 as a running example.

[3]Throughout, by "finish" a method we mean: write test cases for it, then complete its body, and then test it

[4]Throughout, by "finish" a method we mean: write test cases for it, then complete its body, and then test it

[5]Throughout, by "finish" a method we mean: write test cases for it, then complete its body, and then test it

[6]Reminder: This requires actions on the part of both parties. Brief instructions for how to form a CMS group are on the course Assignment page.



```
North Carolina
        Gonzaga
                South Carolina
                        Florida
                                Wisconsin
                                Florida
                        South Carolina
                                Baylor
                                South Carolina
                Gonzaga
                        Gonzaga
                                Gonzaga
                                West Virginia
                        Xavier
                                Xavier
                                Arizona
        North Carolina
                Oregon
                        Kansas
                                Kansas
                                Purdue
                        Oregon
                                Oregon
                                Michigan
                North Carolina
                        North Carolina
                                North Carolina
                                Butler
                        Kentucky
                                UCLA
                                Kentucky
```

Figure 1: Left: A "bracket-like" depiction of the Outcome tree for the so-called "Sweet Sixteen" portion of the bracket diagram in Section 1. Each arrow-and-dotted-bracket represents an Outcome, where `input1` and `input2` are the top and bottom ends of the dotted bracket, and the `winner` is the string pointed to by the arrow. Right: the string representation for that Outcome that is produced by the `__str__` method we've provided you.

- The outcome of the championship game was that North Carolina (the line with no indentation) was the winner, in a game between Gonzaga (the first line that's indented one level) and North Carolina (the other line that's indented one level).
  - The reason that Gonzaga got to play in the championship was that the outcome of the final-four game between South Carolina (the first line under the one-level Gonzaga line that's indented two levels) and Gonzaga (the other line under the one-level Gonzaga line that's indented two levels) was that Gonzaga was the winner.
  - The reason that North Carolina got to play in the championship was that the outcome of the final-four game between Oregon (the first line under the one-level North Carolina line that's indented two levels) and North Carolina (the other line under the one-level North Carolina line that's indented two levels) was that North Carolina was the winner.
    - The reason North Carolina got to play in the final-four was that the outcome of the elite-eight game between North Carolina and Kentucky was that North Carolina was the winner.

... and so on. Sounds recursive, doesn't it?

Hence, we've created a class Outcome to represent outcomes.

4 TODO (aim for end-of-the-day (EOD) Wednesday the 12th): Understand the basics of the Outcome class design

Carefully read over the Outcome class specification given in `a4.py`.

4.1 Attributes

Why do we allow attributes `input1` and `input2` to be strings as well as Outcomes? Because we can't have an infinite recursion where it's Outcomes all the way down: we need to be able to create "bottom-level" Outcomes where we aren't recording why an input is in the competition. For instance, in our running example, we aren't bothering to specify what Outcome led to North Carolina playing Butler in the sweet-sixteen round. And in general, the participants in the first round of a tournament might not have had to play anyone in order to participate.

4.2 `__init__`

Our decision about attribute types has repercussions for the complexity of the specification of the `__init__` method, in `a4.py`. Read that specification now.

You see that we must handle both the situation where `input1` should be an Outcome and the situation where it should be a string, and similarly for `input2`. And, suppose the winner comes from `input1`. Then, the name of the winner must extracted differently from `input1` depending on whether `input1` is a string or an Outcome.

4.2.1 Initializing the winner non-explicitly and with an optional parameter

For the convenience of human callers, the `__init__` method is specified to set the `winner` attribute using an optional parameter, `one_won`, rather than directly taking a value to set `winner` to. To see why this is nice, look at the function `standard_outcome()` in file `a4test.py`, which will create the Outcome depicted in Figure 1 once `Outcome.__init__()` has been properly implemented:

- We can write `Outcome("Gonzaga", "West Virginia")` to create the Gonzaga-vs-West Virginia matchup in Figure 1 and automatically have Gonzaga be the winner. If we'd been required to explicitly name the winner, we'd have to have said `Outcome("Gonzaga", "West Virginia", "Gonzaga")` which is unattractively redundant. Similarly, compare `Outcome("Wisconsin", "Florida", False)` with `Outcome("Wisconsin", "Florida", "Florida")`.
- Additionally, if we'd had to explicitly name the winner, we'd be vulnerable to typos: in fact, in typing the item above, the first time around, one of us accidentally wrote `Outcome("Wisconin", "Florida", "Florida")`.[7]

4.3 `__str__`: An example of a recursive method on Outcomes

We've mentioned before that Outcomes have a naturally recursive structure. You can see this principle in action in the Outcome `__str__` method, which we've written for you.

Recall that if variable `x` is an object, then `str(x)` runs the `__str__` method of the (deepest, most sub-y) class that `x` belongs to. This means we can write `str(x)` below and in our code instead of having to type in the underscores.

The core idea behind our implementation is:

1. Get this Outcome's winner's name — namely, `self.winner` — and move to a new line.
2. Recursive step: get the output of `str(self.input1)`, and indent each of its lines one level.[8]
3. Recursive step: get the output of `str(self.input2)`, and indent each of its lines one level.

5 TODO (aim for EOD Thursday the 13th): Finish[9] `__init__`

5.1 The `test_init` test procedure in `a4test.py`

In this assignment, you are required to develop (and submit) some test cases on your own — *we hope you've learned by now how useful test cases are, and thus, when you program "in real life", we hope you always write test cases early in the process.*

In the case of testing the initializer, we're simplifying this process by:

[7] Sorry, Badgers.
[8] The indentation stuff is a little tricky, and we do not expect you to have come up with this Python on your own.
[9] Throughout, by "finish" a method we mean: write test cases for it, then complete its body, and then test it

- Letting you use our pre-written `__str__` method to test `__init__` with. A test input consists of an Outcome `tc` that you create, since the process of creating an object implicitly calls the initializer method. The correct answer for `tc` is what you *expect* `str(tc)` to return, and you check whether what you get when you apply `str` to `tc` is what you expected.
- Providing a reasonably-sized test case for you. Function `standard_outcome()` in a4test.py returns the Outcome in Figure 1, and function `standard_outcome_str()` returns the expected string. Thus, one test case is to compare `str(standard_outcome())` against `standard_outcome_str()`. That is the purpose of the lines of code we've given already you in `test_init()`, in `a4test.py`.

What do you still have to do?

1. Read the comments at the top of a4test.py, which tell you the three sorts of Outcomes that need testing. Since we've supplied a standard outcome, those comments tell you that you need to...
2. Add code to create a base-case Outcome, figure out what its string representation should be, and check whether applying `str()` to that base-case Outcome gives you that right answer. And,
3. ... do the same for an "unbalanced" Outcome.

5.2 Implementation notes

You then need to complete the body of `__init__` so that it obeys the given specification.

As mentioned in Section 4.2, this specification looks complex in part because extracting a winner's name requires handling both Outcomes and strings. We abstract away this issue by giving you a helper function, `_extract_name`. Because it is a function that is not a method, it is defined *after* all the Outcome methods in a4.py: look around line 125. Read its specification, and determine how to use it in your implementation of the initializer.

Note that we made _extract_name a non-method — it's not indented inside the Outcome class definition — because it doesn't need to be associated with particular Outcome objects; indeed, it's designed explicitly to be applied to either Outcomes or strings.

The function name has a leading single underscore because we don't anticipate it being used anywhere outside the a4.py file; we've therefore designated it a hidden function.

6 TODO (aim for EOD Monday the 17th): Finish[10] recursive methods `teams()` and `pathToVictory()`

You'll need to supply your own testcases in the test procedures `test_teams()` and `test_path()` in a4test.py. You can use `standard_outcome()` again; in fact, we've provided functions `standard_outcome_teams` and `standard_outcome_path()` that return the *expected* results of `standard_outcome().teams()` and `standard_outcome().pathToVictory()`, respectively.

You'll need to check the types of `self.input1` and `self.input2`. Take a look at `_extract_name()`, which we wrote for you: it uses `isinstance(x, Outcome)` instead of `type(x) == Outcome` and you should too.[11]

The alphabetization requirement mentioned in the specification for `teams()` can be taken care of with the standard sorting methods/functions for lists.

In `pathToVictory()`, you have to decide whether to perform recursion on `self.input1` and record the name of `self.input2`, or vice versa. Rather than have really long if-else blocks, we suggest that you first have an if-else clause that correctly sets two variables, one corresponding to the recursion target and one corresponding to the loser's name. This allows you to write more generic code, referring directly to your new variables, after and outdented at the same level as the main body of `pathToVictory()`.

7 TODO (aim for afternoon Thursday the 20th): Finish[12] recursive method `hasHeadToHead()`

For testing, we've provided you some testing for the standard test Outcome in test procedure `test_hasHeadToHead()`. **You must decide whether to add test cases to the `test_cases` dictionary for this standard Outcome.**

[10] Throughout, by "finish" a method we mean: write test cases for it, then complete its body, and then test it

[11] Reason: what if someone has defined a subclass of Outcome, and is using that instead? Here is a Stack Overflow post on this topic.

[12] Throughout, by "finish" a method we mean: write test cases for it, then complete its body, and then test it

You may lose points for not including missing important test cases for the standard Outcome.

You also need to add testing for a base-case Outcome and an unbalanced Outcome. You can use our code for the standard Outcome as a model.

8 TODO (by the submission deadline): Edit your code

Recursive solutions at their best tend to be compact and elegant, and can be a pleasure to read. However, they don't tend to look their best on their first draft, even if they pass all test cases, just as most pieces of writing can be improved by editing.

The above observation is not just idle philosophizing: **the fewer lines a piece of code has, the fewer places it can have bugs.** Hence, once you get a recursive solution working, read it back over and try to find redundancies you can eliminate.

- Do you have more special cases than you need? A bad sign is lots of if-elifs or nested if-statements.
- Are you creating variables that you don't end up needing?
- Can you tell what your variables mean after being away from your computer for an hour? If not, you might not be using a variable consistently — *changing the meaning of your variables without (ahem) meaning to is a* **major** *source of bugs.* Or, it could be just a case of poor choice of variable names; but then why not make them better?

Also, the above observation pertains not just to recursive code, but to all code. We make lots of drafts of code before we distribute it to you; and we even update almost every piece of lab code each semester, even though the labs are "the same". Revision is key to all great writing — be it in English or in code.

9 Before You Submit

Remove all extraneous comments from `a4.py` and `a4test.py`, including comments we provided that do not document what your code is doing, and comments you've been told to remove.

Remove all debugging print statements.

Put your (and your partner's, if you have one) name and netid(s) at the top of the submitted file(s).

Acknowledge by name all help you received in comments at the top of your submitted file(s).