Programming Team Practice: Transitioning from Voluntary Sessions to a Sanctioned Course

Ian Barland, Maung Htay
{ibarland,mhtay}@radford.edu

Outline

- Establishing a course
- Course structure
- Some Contest Tactics

Outline

- Establishing a course
- Course structure
- Some Contest Tactics

Original Organization

- Practice each Saturday, 12:00-17:00
- Providing lunch helps!
- Only dedicated students attend
  (is this good or bad?)
- Six on campus days per week – might weary the coach

Meeting as a class

- Better attendance; auto-advertising.
- Gives some formal recognition of students' effort.
- Does not count towards degree (as prev.)
- Taught as faculty overload (as prev.)
- Adds to departmental credit-production (reflecting work/resources already being committed)

Choices, in course design

- One credit hour; one evening lab.
- Pre-req: CS2 suggested
- Title: "Programming Practicum"
- Independent of participating in contest
- A glitch: 19th credit hour?
- Emphasis shifted from "time to just solve" to mix of "discuss algorithms, *and* solve"

Outline

- Establishing a course
- Course structure
- Some Contest Tactics

Fall semester

- Focus on practice for contest
- Go over solutions one hour in class; one hour to start next set.
- Discussion of tactics: managing the keyboard, delegating problems, *etc*.

Spring semester

- More focus on algorithms, and less on quick&dirty solutions
- Digress on related algorithms (longest-path, or Steiner tree)
- Much more wide-ranging conversations
  - arrays vs lists
  - library design (Java's `Scanner`, or `String.split` oddities)

Comparing Languages

- Might assign same problem in different langs
- `java.util.Map` vs php arrays vs javascript object properties – which do they prefer?
- Passing a function vs `java.util.Comparable` *vs.* `Comparator` *vs.* php string-as-function-name
- Pass-by-ref, vs Java's passing references as values – even top students can be fuzzy here

Example: Dynamic Programming

- Dynamic programming as a last resort:
  - raw recursive solution; *if* that fails then:
  - memoize (rote); *if* that's mem-intensive then:
  - dynamic programming (*ad hoc*).
- Might preview a language where you can write 'memoize' as a function/macro.

Whole-problem analysis

- Given a problem/scenario (e.g. a strategy game)
  - heuristic vs sure-move
  - different types of greedy algs
  - weaknesses of existing interface
  - termination?
  - Estimate size of state-space or solution-size
  - formal upper or lower bounds
  - approaches to features (e.g. undo)
- MVC – writing V or C is extra credit

Outline

- Establishing a course
- Course structure
- Some Contest Tactics
  - coding approaches
  - social approaches

tactic: in grid problems, wrap array-lookup

- ```
  <T> T safeLookup( T[] arr, int I, T dflt ) {
    return (0 <= i && i < arr.length) ? arr[i] : dflt;
  }
  ```

- Can greatly simplify grid problems and boundary conditions (though not always)
  ```

tactic: Scanner as argument

- E.g.

```
double solve( List<City> cities ) { … }
double solve( Scanner s ) {
    List<City> cs = new ArrayList<City>();
    while (s.hasNext("city: ")
      cs.add( new City(s) );
    solve(cs);
}
```

- Dis-entangle parsing from solving.
- Helpful when helper-functions involved.

tactic: Unit Tests

- Adds overhead, but when it helps it helps big.
- Consider whenever you a helper function (e.g. anything more than a nested loop)
- Consider if sub-problem has many corner cases
- Helps reinforce the notion for other courses.
- Doesn't need to be full junit or heavyweight:

```
void checkExpect( Object actl, Object expct ) {
  print( actl.equals(expct) ? "." : "FAIL" ); }
```

tactic: Avoid State

- For searching, consider immutable data:
  - Much simpler to program
    (no worrying about a sub-call changing your data underneath you)
  - overhead of copying/sharing not as bad as I always fear
- Overheard: "Solving Missionaries&Cannibals [breadth-first with immutable lists] was the biggest eye-opener in my programming."

Outline

- Establishing a course
- Course structure
- Some Contest Tactics
  - coding approaches
  - social approaches

tactic: The first 30min

- Get on scoreboard first!
- Only *after* 30min have all members discuss each problem.
- Each member rates:
  - difficulty;
  - type (search, greedy, dynamic prog, …);
  - any structs/classes, subroutines *vs.* one-big-main

tactic: Communication

- Soft-spoken coder can get ignored by brasher egos.
  - soln?
- A stumbling block: asking for help from busy teammates.
  - Perhaps a codeword: e.g. "Sphinx!" for "I need to bounce some ideas off you, when you finish your train of thought."

tactic: watch your diet

- Avoid morning donuts, or 2nd afternoon cookie.
- Even just morning fruit can give a lot of sugar.
- Do exclaim loudly, how good the doughnuts are when other teams are within earshot.

Questions/Thoughts/Tips?

Ian Barland

ibarland@radford.edu

RU RADFORD UNIVERSITY