CS106X Instructor: Cynthia Lee

Autumn 2015 Practice Exam

PRACTICE MIDTERM EXAM – SOLUTIONS

---

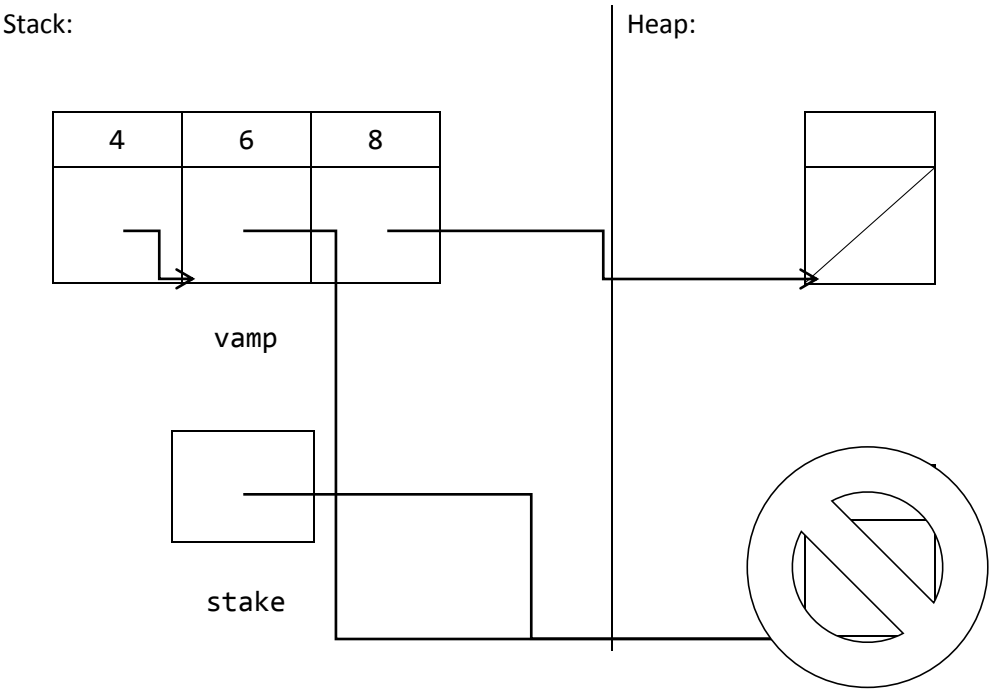
1. **ADTs (30pts).**

```
int shortestLivingPath(Grid<int>& board, Point infected, Point dest) {
       Queue<Vector<Point>> toExplore;
       Set<Point> visited;
       Vector<Point> startPath;
       startPath.add(infected);
       toExplore.enqueue(startPath);

       while (!toExplore.isEmpty()) {
           Vector<Point> currentPath = toExplore.dequeue();
           Point end = currentPath[currentPath.size() - 1];
           if (!board.inBounds(end.row, end.col) ||
               board[end.row][end.col] == 0 ||
               visited.contains(infected)) {
               continue;
           }
           if (end == dest) return currentPath.size();

           visited.add(end);

           for (int row = end.row - 1; row < end.row + 1; row++) {
               for (int col = end.col - 1; col < end.col + 1; col++) {
                   if (row == end.row && col == end.col) continue;
                   Vector<Point> newPath = currentPath;
                   Point next(row, col);
                   newPath.add(next);
                   toExplore.enqueue(newPath);
               }
           }
       }

       return -1;
}
```

---

2. **Pointers and Memory (21pts).**

DRAWING:



---

3. **Classes (30pts).**

a) (5 Points)

```
class Lecture {
public:
  Lecture(DayPattern daysOfWeek, int startTime, int duration);

  int startTime() const;
  int durationInMinutes() const;
  DayPattern daysOfWeek() const;

  int endTime() const;
  bool overlapsWith(const Lecture& other);

private:
  int startTime;
  int durationInMinutes;
  DayPattern daysOfWeek;
};
```

b) **(10 Points)**

```
Lecture::Lecture(DayPattern daysOfWeek, int startTime, int duration) {
```

```
  this->daysOfWeek = daysOfWeek;
  this->startTime = startTime;
  this->duration = duration;
}

int Lecture::startTime() {
  return this->startTime;
}

int Lecture::durationInMinutes() const {
  return this->duration;
}

DayPattern Lecture::daysOfWeek() const {
  return this->daysOfWeek;
}
```

c) **(15 Points)**

```
int Lecture::endTime() const {
  int startHour = startTime / 100;
  int startMinute = startTiem % 100;
  int endHour = (duration + startMinute) / 60 + startHour;
  int endMinute = (startMinute + duration) % 60;
  return 100 * endHour + endMinute;
}

bool Lecture::overlapsWith(const Lecture& other) {
  if (this->daysOfWeek() != other->daysOfWeek()) return false;
  return !(this->endTime()   <= other->startTime() ||
           other->endTime()  <= this->startTime());
}
```

---

4. **Recursion (30pts).**

```
bool canTakeAtLeastKUnits(const Vector<string>& interestingClasses,
                          const Map<string, Lecture>& schedule,
                          const Map<string, int>& units,
                          int k,
                          Vector<string>& mySchedule) {
    Vector<string> coursesCopy = interestingClasses;
    return canTakeAtLeastKUnitsRec(coursesCopy, schedule, units, k, mySchedule);

}

bool canTakeClass(const string& courseName, const Vector<string>& mySchedule,
                  const Map<string, Lecture>& schedule) {
    Lecture thisLecture = schedule[courseName];
    for (string otherLecture : mySchedule) {
        if (thisLecture.overlapsWith(schedule[otherLecture])) return false;
```

```
    }
    return true;
}

bool canTakeAtLeastKUnitsRec(Vector<string>& interestingClasses,
                             const Map<string, Lecture>& schedule,
                             const Map<string, int>& units,
                             int k,
                             Vector<string>& mySchedule) {
    if (k <= 0) return true;
    if (interestingClasses.isEmpty()) return false;

    string nextCourseName = interestingClasses[0];
    interestingClasses.remove(0);
    if (canTakeClass(nextCourseName, mySchedule, schedule)) {
        mySchedule += nextCourseName;
        int remainingUnits = k – units[nextCourseName];
        if (canTakeAtLeastKUnitsRec(interestingClasses, schedule, units,
                                    remainingUnits, mySchedule)) {
            return true;
        }
        mySchedule.remove(mySchedule.size() – 1);
    } else if (canTakeAtLeastKUnitsRec(interestingClasses, schedule, units,
                                       k, mySchedule)) {
        return true;
    }
    interestingClasses.insert(0, nextCourseName);
    return false;
}
```

---

5. **Big-O (9pts).**

O(1)
O(NlogN)
O(N^2)