# EC 504
# Fall, 2018
# Exam 3 Solutions

## Monday, December 17, 2018

**Instructions**

- Put your name on the first page.
- This exam is open notes, but no consultation with anyone other than the instructors are permitted during the exam.
- You have one hour and 45 minutes to complete this exam. There are 7 questions. All work must be shown on the paper to be counted.

1. (20 pts) Answer True or False to each of the questions below, **AND YOU MUST PROVIDE SOME FORM OF EXPLANATION (very brief is OK)**. Each question is worth 2 points. Answer true only if it is true as stated, with no additional assumptions.

   (a) Consider a minimum spanning tree $T$ in a connected, weighted undirected graph $(V, E)$ where the weights have different values. Then, for every node $i$, the minimum spanning tree must contain the arc $\{i, j\}$ with the smallest weight connected to node $i$.
   **Solution:** True. Otherwise, we can add that arc, and disconnect a current arc adjacent to node $i$, and get a smaller weight spanning tree.

   (b) Dynamic programming can be used to solve the integer knapsack problem in pseudo-polynomial time.
   **Solution:** True.

   (c) Assume that there is a decision problem which can be described in an input with $n$ bits. Assume that the worst-case instance of this decision problem of size $n$ bits requires $O(2^n)$ operations to solve. Then, the decision problem belongs to class NP.
   **Solution:** False. It might be in NP, but there is no certification that a solution can be evaluated in polynomial time.

   (d) Finding the minimum-distance tour in a traveling salesperson problem is an NP-Hard problem.
   **Solution:** True. Indeed, it is an optimization problem, which is equivalent to an NP-complete decision problem. Thus, it is NP-Hard.

   (e) The problem of determining whether a graph has an Eulerian cycle is in class P.
   **Solution:** True. We just have to look compute the degrees of every node and verify that they are even.

   (f) Suppose we have an R-tree of order $M$, with minimum keys $m$, where the R-tree stores data that is indexed by points in 2-d space. If the R-tree is of height (Root level plus next level plus leaf level), then the maximum number of data entries stored in the R-tree is $M^2$.
   **Solution:** False. There are $M^2$ leaf nodes, but each leaf node has a maximum of $M$ data entries.

   (g) Suppose we have $n$ keys stored in an R-tree. Then, the worst-case time to find a key is $O(\log(n))$.
   **Solution:** False. Unfortunately, one may have to search multiple children.

   (h) Given any instance of a traveling salesperson problem, we can find an approximate algorithm that computes a tour of all the nodes with distance no worse than 2 times the distance of the optimal tour.
   **Solution:** False. Would be true if the distances satisfied the metric property.

2. (10 pts) Consider the following 2-dimensional set of points:

$$(4, 8), (3, 4), (2, 9), (1, 2), (2, 3), (4, 4), (6, 7), (7, 9), (1, 10)$$

   Arrange the elements in a PR quadtree with value range 0-10 for both the first and second coordinates.
   **Solution:**

```
Sorting to coarsest level, we get nodes that need expansion
                Root
                 |
        -------------------------
     00|    01|      |10         |11
    (3,4)   (2,9)              (6,7)
    (1,2)   (4,8)              (7,9)
    (2,3)   (1,10)
    (4,4)
```

```
We now subdivide to next level: squares of 2.5 x 2.5
                            Root
                             |
                 ---------------------------
              00|        01|    |10         |11
     ---------------      -----           -------
   00|  01|   |10  |11   |01  |11       00|    01|
  (1,2) (2,3)     (3,4) (2,9) (4,8)     (6,7)  (7,9)
                  (4,4) (1,10)
```

```
Subdivide to the next level (squares size 1.25 by 1.25)

                            Root
                             |
                 ---------------------------
              00|        01|    |10         |11
     ---------------      -----          -------
   00|  01|   |10  |11   |01  |11      00|    01|
  (1,2) (2,3)      |     |   (4,8)     (6,7)  (7,9)
                   |     |
           ----------   -------
           |01    11|   01|    |11
          (3,4)   4,4) (1,10) (2,9)
```

3. (10 pts) Consider the following 2-dimensional set of points:

$$(4,8),(3,4),(2,9),(1,2),(2,3),(4,4),(6,8),(7,9),(1,10)$$

Form a balanced 2-d binary search tree using the median of the elements to split, with all the keys at the leaves of the tree. To make things unique, when you have to

nd a median of a list with an even number of entries, choose the smaller of the two entries as the median. Assume also that when a key is equal to the navigation key, it is added to the left subtree (that is, less than or equal).

**Solution:** First, find the median of the first coordinates. Sorting by the first coordinate yields:

$$(1,2),(1,10),(2,3),(2,9),(3,4),(4,4),(4,8),(6,7)(7,9)$$

The median value is 3, resulting in

```
                                    3
                 -------------------|-------------------
                |                                       |
(1,2),(1,10),(2,3),(2,9),(3,4)               (4,4),(4,8),(6,7)(7,9)
```

The median of the second coordinate on the right is 4, so we branch on 4. For the second list, the median is 7 (because we branch left), so we branch on 7. We get:

```
                                    3
                 -------------------|-------------------
                |                                       |
                4                                       7
       ---------|----------                ------------|-----------
      |                    |               |                        |
 (1,2),(2,3),(3,4)   (1,10),(2,9)      (4, 4),(6,7)        (4, 8),(7,9)
```

Repeat the process for each leaf above, branching onthe median of the $x$-values.

```
                                    3
                 -------------------|-------------------
                |                                       |
                4                                       7
       ---------|----------                ------------|-----------
      |                    |               |                        |
      2                    1               4                        4
  ----|--------       -----|-----     -----|-----              ----|----
 |             |     |           |   |           |            |         |
(1,2),(2,3)   (3,4) (1,10)     (2,9) (4,4)      (6,7)        (4,8)      (7,9)
```

Finally, one more branch:

```
                                    3
                 -------------------|-------------------
                |                                       |
                4                                       7
       ---------|----------                ------------|-----------
      |                    |               |                        |
      2                    1               4                        4
  ----|--------       -----|-----     -----|-----              ----|----
 |             |     |           |   |           |            |         |
 2            (3,4) (1,10)     (2,9) (4,4)      (6,7)        (4,8)      (7,9)
 _\-------
|         |
(1,2)   (2,3)
```

4. Consider the following 2-dimensional set of points:

$$(4,8),(3,4),(2,9),(1,2),(2,3),(4,4),(6,7),(7,9),(1,10)$$

Below is an R-tree of order $M=3, m=2$, where the minimum number of children (and keys) is 2, formed using linear splitting, with the following variation: when splitting an area defined by points, use as seeds the two points that are farthest in distance from each other. The R-tree already contains the first six points: $(4,8),(3,4),(2,9),(1,2),(2,3),(4,4)$.

```
                 -----------------------
                |(3,4) |  (1,2) | (2,8) |
                |(4,4) |  (2,3) | (4,9) |
            -------/         |       \---------------
 ----------/------    -------|--------       --------|-------
|(3,4)|(4,4)|     |  |(2,3)|(1,2)|    |     |(4,8)|(2,9)|    |
|(3,4)|(4,4)|     |  |(2,3)|(1,2)|    |     |(4,8)|(2,9)|    |
```

(a) (8 pts) Insert the next three points in order, $(6,7),(7,9),(1,10)$, showing the resulting R-tree at each step. Use minimum increase in perimeter as the rule for navigation when selecting which leaf to insert.

**Solution:**

Insert (6,7). Into the first rectangle increases perimeter by 12. Into the second rectangle, increases perimeter by 16. Into the 3rd rectangle, increases perimeter by 6, so it goes into 3rd leaf, and the is room.

```
                       ________________________
                      |(3,4) |  (1,2) | (2,7) |
                      |(4,4) |  (2,3) | (6,9) |
                  _______/          |       \________________
      __________/______     _______|________        _________|________
     |(3,4)|(4,4)|     |  |(2,3)|(1,2)|    |     |(4,8)|(2,9)|(6,7)|
     |(3,4)|(4,4)|        |(2,3)|(1,2)|    |     |(4,8)|(2,9)|(6,7)|
```

Next, insert (7,9). Inserting into first rectangle increases perimenter by 16. Into the second increases perimeter by 20. Into the 3rd increases perimeter by 2. We get overflow

```
                       ________________________
                      |(3,4) |  (1,2) | (2,7) |
                      |(4,4) |  (2,3) | (6,9) |
                  _______/          |       \________________
      __________/______     _______|________        _________|________
     |(3,4)|(4,4)|     |  |(2,3)|(1,2)|    |     |(4,8)|(2,9)|(6,7)|(7,9)
     |(3,4)|(4,4)|        |(2,3)|(1,2)|    |     |(4,8)|(2,9)|(6,7)|(7,9)
```

The two farthest points are (2,9) and (7,9). It is clear that (6,7) gets added to (7,9), and (4,8) gets added to (2,9), resulting in a split:

```
                       ________________________
                      |(3,4) |  (1,2) | (2,8) | (6,7)
                      |(4,4) |  (2,3) | (4,9) | (7,9)
                         /         |        \      \_____________________________
                 _______/          |         \_______________                    |
      _________/_______     ______|_________        _________|_______    _______|________
     |(3,4)|(4,4)|     |  |(2,3)|(1,2)|    |     |(4,8)|(2,9)|    |  |(6,7)|(7,9)|    |
     |(3,4)|(4,4)|        |(2,3)|(1,2)|    |     |(4,8)|(2,9)|    |  |(6,7)|(7,9)|    |
```

We have overflow at the top level, so we have to split that! These are rectangles, so we use the linear split rule for rectangles. For the x-axis, the maximum white space is 4, and the maximum range is 6, for a ratio of 2/3. For the y axis, the maximum white space is 5, and the maximum range is 7. So we split on the y axis, with seeds [(1,2),(2,3)] and [(2,8),(4,9)].

To add [(3,4),(4,4)] to [(1,2),(2,3)] increases area by 5. To add it to [(2,8),(4,9)] increases area by 8. So, we add it to [(1,2),(2,3)] to get rectangle [(1,2),(4,4)], and add [(6,7),(7,9)] to [(2,8),(4,9)] to get rectangle [(2,7),(7,9)]. The result is:

```
                                  __________________
                                 |(1,2)|(2,7)|     |
                                 |(4,4)|(7,9)|     |
                                    /      \
                             _____/         \______
                       _____|__________      _____|___________
                      |(3,4)|(1,2)|    |    |(2,8)|(6,7)|     |
                      |(4,4)|(2,3)|    |    |(4,9)|(7,9)|     |
                         /      \                \     \_________________________
                 _______/        \                \_________                     |
      _________/_______     ______|_________        _________|_______    _______|________
```

```
|(3,4)|(4,4)|      |  |(2,3)|(1,2)|     |     |(4,8)|(2,9)|     |  |(6,7)|(7,9)|     |
|(3,4)|(4,4)|         |(2,3)|(1,2)|     |     |(4,8)|(2,9)|     |  |(6,7)|(7,9)|     |
```

Now, insert (1,10). Inserting into [(1,2),(4,4)] increases perimeter by 12. Inserting it into [(2,7),(7,9)] increases perimeter by 4, so insert it there. Go down to the next level, and consider inserting into [(2,8),(4,9)], which increases by 4. Inserting into [(6,7),(7,9)] increases perimeter by 12. So insert into the leaf [(2,8),(4,9)], which has room! Final tree is:

```
                             ------------------
                            |(1,2)|(2,7)|      |
                            |(4,4)|(7,9)|      |
                              /       \
                        -----/         \------
                  -----|-----------      -----|-----------
                 |(3,4)|(1,2)|     |    |(1,8) |(6,7)|     |
                 |(4,4)|(2,3)|     |    |(4,10)|(7,9)|     |
                    /      \                \     \__________________________
            -------/        \                \_________                      |
 ---------/-------     ------|---------        ---------|---------    -------|--------
|(3,4)|(4,4)|      |  |(2,3)|(1,2)|     |     |(4,8)|(2,9)|(1,10)|  |(6,7)|(7,9)|    |
|(3,4)|(4,4)|         |(2,3)|(1,2)|     |     |(4,8)|(2,9)|(1,10)|  |(6,7)|(7,9)|    |
```

(b) (2 pts) For the above tree, show the tree that results after deleting the point (4,8).

**Solution:** We find it, and find that we can simply delete it because we have enough data keys there. We do have to update the navigation value. The final tree is:

```
                             ------------------
                            |(1,2)|(2,7)|      |
                            |(4,4)|(7,9)|      |
                              /       \
                        -----/         \------
                  -----|-----------      -----|-----------
                 |(3,4)|(1,2)|     |    |(1,9) |(6,7)|     |
                 |(4,4)|(2,3)|     |    |(2,10)|(7,9)|     |
                    /      \                \     \__________________________
            -------/        \                \_________                      |
 ---------/-------     ------|---------        ---------|---------    -------|--------
|(3,4)|(4,4)|      |  |(2,3)|(1,2)|     |     |     |(2,9)|(1,10)|   |(6,7)|(7,9)|    |
|(3,4)|(4,4)|         |(2,3)|(1,2)|     |     |     |(2,9)|(1,10)|   |(6,7)|(7,9)|    |
```

5. Consider the integer knapsack problem, described as follows: There are $n$ objects $i = 1, \dots, n$, each of which has integer size $c_i$ and value $V_i$. You are given a bag of total integer size $C$. You are allowed to put objects in the bag, as long as the total size of the objects is less than or equal to $C$. The goal is to find the set of objects that you will put in the bag that give you the maximum value.

One way to solve this problem is to use dynamic programming in an interesting, clever way. Define the following function: $T(k, c)$ for non-negative integers $k, c$ is the highest value you can put in a bag of size $c$ when you can only consider objects $0, \dots, k$. You can compute this function recursively as follows:

$$T(1, c) = \begin{cases} 0 & \text{if } c < c_1 \\ V_1 & \text{if} c \geq c_1 \end{cases}$$

We can compute this recursively now as follows: when you consider an extra object, then either it is optimal to put this in the bag, so the rest of the object in the bag must fit in the leftover space, or it is best to leave this object out. This leads to this recursion:

$$T(k,c) = \begin{cases} T(k-1,c) & \text{if } c < c_k \\ \max\{T(k-1,c), V_k + T(k-1,c-c_k)\} & \text{otherwise} \end{cases}$$

Once you compute $T(n,C)$, this is the optimal value when all objects are considered for a bag of size $C$.

(a) (5 pts) Consider the following list of objects: Object 1 has value 3, size 2. Object 2 has value 2, size 3. Object 3 has value 3, size 4. Use the above recursive algorithm to compute $T(3,2), T(3,3), T(3,4), T(3,5)$ and $T(3,6)$.

**Solution:** What you need appears in the third column below:

| T(c\k) | 1 | 2 | 3 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 2 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 |
| 4 | 3 | 3 | 3 |
| 5 | 3 | 5 | 5 |
| 6 | 3 | 5 | 5 |

(b) (3 pts) Is this problem (integer knapsack) in class $NP$? Please explain why.

**Solution:** Given a set of objects that are claimed to be in the bag, one can verify in $O(n)$ whether the object fit in the bag, and what their value is, in $O(n)$.

(c) (3 pts) Provide an expression for the worst case complexity of the above algorithm in terms of $n$ and $C$. Justify your answer

**Solution:** Filling the table, you get a complextiy $O(nC)$.

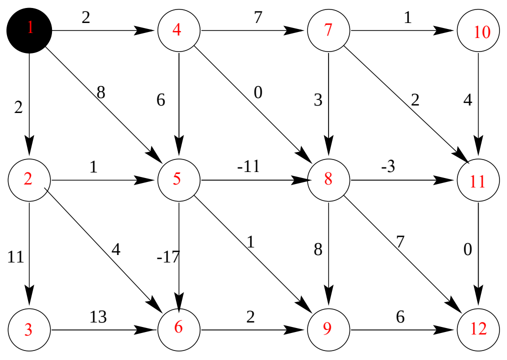
(d) (2 pts) Is this problem (integer knapsack) in class $P$? Please explain why.

**Solution:** It is not. Look at what happens for $C$ growing. To specify $C$ takes $\log(C)$ bits. The complexity is exponential in the size required to specify $C$. Thus, the above algorithm is not polynomial complexity.

(e) 2 pts Suppose you are allowed to divide a task and get partial credit for the fraction of the task that you put in the bag. Find the optimal value that you can put into a bag of size 5.

**Solution:** Insert in order of value/size. So insert object 1, then 3/4 of object 3, for total value $3 + 9/4 = 21/4$.

6. Consider a directed acyclic graph with weights on the arcs, with an origin node 1 and a destination node $n$, as illustrated in the figure on the side. We want to find the ***longest*** path from node 1 to node $n$.



(a) (3 pts) Is the decision version of this problem in class NP? If so, explain why.

(b) (4 pts) Is the decision version of this problem in class P? Explain why or why not.

(c) (3 pts) Compute the longest path from node 1 to node 8.

**Solution:** The decision version of the problem is in class NP because, given a path from node 1 to node $n$, computing the length of the path can be done in time linear in the number of nodes in the path, and hence we can certify in polynomial time whether the answer should be true or not. Remarkably, the problem is also in class P, because there are no cycles. Hence, reversing the sign of every distance converts the problem to that of finding a shortest path from node 1 to node $n$. Since there are no cycles, there are no negative cost cycles, so this can be solved by any version of Bellman-Ford algorithm, with complexity polynomial in the number of arcs and nodes.

Finally, the longest path from 1 to 8 can be computed by doing scan iterations using maximization instead of minimization. We illustrate the distance computations below, scanning the nodes left to right, top to bottom. We ignore the last row and the last column, as none of those nodes can be in a path to 8.

| | Init: | Scan 1 | Scan 2 | Scan 4 | Scan 5 | Scan 7 |
|---|---|---|---|---|---|---|
| D(1) | 0 | 0 | 0 | 0 | 0 | 0 |
| D(2) | -inf | 2 | 2 | 2 | 2 | 2 |
| D(4) | -inf | 2 | 2 | 2 | 4 | 4 |
| D(5) | -inf | 8 | 8 | 8 | 8 | 8 |
| D(7) | -inf | -inf | -inf | 9 | 9 | 9 |
| D(8) | -inf | -inf | -inf | 2 | 2 | 12 |

The longest distance is 12, the path is 1-4-7-8.