COMS 311: Homework 8 Due:
Dec 8, 11:59pm Total Points:
20

Late submission policy. **Submissions after Dec 8, 11:59PM will not be graded without explicit permission from the instructors.**

Submission format. Your submission should be in pdf format. Name your submission file: <Your-net-id>-311-hw8.pdf. For instance, if your netid is asterix, then your submission file will be named asterix-311-hw8.pdf.

If you are planning to write your solution and scan+upload, please make sure the submission is readable (sometimes scanned versions are very difficult to read - poor scan quality to blame).

If you plan to snap pictures of your work and upload, please make sure the generated pdf is readable - in most cases, such submissions are difficult to read and more importantly, you may end up submitting parts of your work.

If you would like to type your work, then one of the options you can explore is latex (Overleaf).

Rules for Algorithm Design Problems.

1. Unless otherwise mentioned, for all algorithm design problems, part of the grade depends on runtime. Better runtime will lead to higher grade.
2. Unless otherwise mentioned, for all algorithm design problems, you are required to write some justification of why your algorithm correctly addresses the given problem.
3. You will write pseudo-code for your algorithm. It should not be some code in a specific programming language (e.g., Java, C/C++, Python).
4. You can use any operation already covered in class-lectures in your solution without explicitly writing the algorithm. However, you need to use them appropriately (i.e., indicate the inputs to and outputs from these algorithms). For instance, you can use heapifyUp with input as the index of a heap element and you do not need to write the code for heapifyUp.

Learning outcomes.

1. RSA
2. Greedy Algorithms
3. Dynamic Programming

Problems 1: 20pts. Each Extra Credit Problem is 10pts.

1. (a) Consider the prime numbers 7 and 17. Verify that (5, 119) is a correct public key for RSA crypto system. Derive a corresponding private key for RSA crypto-system. Present your derivation steps.

n = 7*17 =119

φ(n) = 6*16 = 96

e = 5 (where e ∈ {2, 3, …, φ(n)})

These φ(n) and e gcd satisfies (e, φ(n)) = 1

∴ (5, 119) is a correct public key for RSA crypto.

e = 5

x = m^e mod n = 5 ^ 2 mod 119 = 3125

m = 5 = x ^ d mod n = 25 ^ d mod 119

Brute force d from 2….

When d = 77 it satisfies e * d mod φ(n) = 1

e * d mod φ(n) = 119 * 77 mod 96 = 1

∴ Kprivate = 77

(b) Using these keys, verify that decryption of encrypted message 4 produces 4. You can use calculator.

m = 4

x = m ^ e mod n = 4 ^ 5 mod 119 = 72

72 ^ d mod n = 4^77 mod 119 = 4

x = 4

∴ the statement is correct

(c) State the sequence multiplications and squaring operations necessary to efficiently compute 1024^{17}. Justify your answer.

$1024^{17} = 2^{170}$

The runtime of the sequence multiplications and squaring operations is O(log(n))

The runtime without the operation is O(n)

2^{170} is a huge enough number that makes a difference in those algorithm.

∴ The sequence multiplications and squaring operations necessary to efficiently compute 1024^{17}.

2. **Extra Credit.** You got n assignments to complete with exactly the same deadline. Unfortunately, due to various other commitments, you were not able to start any of them before the deadline date. But you know that each day from now on, you can complete one assignment per day.

   However, if you submit any assignment after the deadline, there will be penalty. For assignment i, the penalty increases by r_i per day, where $r_i > 1$. That is, if you submit the assignment d days after the deadline, then the penalty you will incur is r_i^d.

   Fortunately, you have gained some knowledge of algorithmic strategies, and you are able to develop a greedy strategy to minimize the *sum of the penalties* that you may incur for n assignments. As per the greedy strategy if you complete the assignments in decreasing order of r_is, you will be able to achieve your minimization objective. Prove that you are correct.

   Assume if the statement is not correct there exist r_n and r_m such that $r_n > r_m$ by swapping the order of n and m, it is possible to minimize the sum of the penalties…i

   Let the penalties before n and m is A and the penalties After n and m is B

   ∵i

   $A + (r_n \wedge d) + (r_m \wedge (d+1)) + B$

   $> A + (r_n \wedge (d+1)) + (r_m \wedge d) + B$

   $\therefore (r_n \wedge d) + (r_m \wedge (d+1)) > (r_n \wedge (d+1)) + (r_m \wedge d)$

   However,

   ∵ $r_n > r_m$

   $(r_n \wedge d) + (r_m \wedge (d+1)) >/ (r_n \wedge (d+1)) + (r_m \wedge d)$…ii

   Here ii and the assumption contradict.

   ∴The statement is correct.

3. **Extra Credit.** You have n programming assignments in a dystopian algorithms class. You have a total of H hours that you can allocate to all the assignments. The points you can secure in these assignments depend on the time (in hrs) you allocate for them. This information is present in the form of an array. For the i-th programming assignment, there is an array G_i of size H + 1, such that $G_i[j]$ indicates the points you can secure for the i-th programming assignment if you spend j hours on it (e.g., $G_i[0]$ is likely to be 0 and $G_i[H]$ is likely to be close to 100). Your objective is to find some allocation of time for each of the n assignments such that the sum of the time allocation is $\leq H$ (no partial hour allocation allowed) and your point-earnings is maximized. Write a dynamic-programming based solution to realize your objective. You need to present (a) a recursive formulation, (b) an iterative DP program for finding the maximal earnings achievable, and (c) an algorithm for finding the allocations that maximize the earnings.

   opt(i,j): max point-earning within the hour

   opt(i,j) = if(i<=j): max(opt(i-1,j),opt(i-1,j-H[j])+H[j])
   else: opt(i-1,j)

   Base case: first column and row
   Set first column and row 0

   Dictionary array(dp) (n+1)*(H+1):
   dp[i][j] keeps tracks of the valuations for opt[i]

   The solution will be stored at dp[n][H]

   Order of computation: i must be computed before i+1 and j must be computed before j+1.

```
for i = 1 to n{
        for j = 1 to H{
                if(i<=j){
                        opt[i][j]=max(opt[i-1][j],opt[i-1][j-H[j]]+H[j])
                }
                else{
                        opt[i][j]=opt(i-1,j)
                }
        }
}
return dp[n][H]
```

4. **Extra Credit.** You are given a list of n (positive and negative) integers. You start with selecting the first or the second element. Subsequent to selecting the first or second element, you have the following choices for selection. If you have selected the i-th element ($i \geq 1$), then the next element that you can select is either the i+2 or i+3. You will continue this selection process until you cannot select any more elements (your choice of selection goes beyond n). The value of your selection is the sum of the integers you have selected. Here is an example scenario.

   4 5 – 1 – 2 8 7

   You may select 4 – 1 8 and then the value of your selection is 11. You may select 5 8 and then the value of your selection is

   Your objective to make selection in such a way that the value of your selection is maximized. Write a dynamic-programming based solution to realize your objective. You need to present (a) a recursive formulation, (b) an iterative DP program for finding the maximal earnings achievable, and (c) an algorithm for finding the allocations that maximize the earnings.

   Recursive formulation:

   opt(i): maximum value until i (inclusive)

   opt(i) = max(opt(i-2), opt(i-3))+List[i]

   Base case: when i = 0,1,2

   dictionary array (dp):

   dp[i]: keeps track of the valuations for opt[i]

   The solution will be stored at either dp[n-1] or dp[n-2] depends on which is bigger.

   Order of computation: i must be computed before i+1

```
if(i==0||i==1){
  dp[i]=List[i]
}
else if(i==2){
  dp[2] = max(dp[0]+List[2], List[2])
}
else{
  dp[i] = max(dp[i-2]+List[i], dp[i-3]+List[i])
}
return max(dp[n-1], dp[n-2]);
```

iterative:

Base case: when i = 0,1,2

dictionary array (dp):

dp[i]: keeps track of the valuations for opt[i]

The solution will be stored at either dp[n-1] or dp[n-2] depends on which is bigger.

Order of computation: i must be computed before i+1

```
dp[0]=List[0]
dp[1]=List[1]
dp[2] = max(dp[0]+List[2], List[2])
for(int i = 3;i<n;i++){
    dp[i] = max(dp[i-2]+List[i], dp[i-3]+List[i])
}
return max(dp[n-1], dp[n-2]);
```