Algorithm Final

Xinyu Jiang

December 2018

1

1. (15 pts) Doctor Jean needs your help! She's been relabeling her collection of chromosome sequences and has found three different sequences that were displaced from their original locations in the lab. She knows that two of the sequences represent two chromosomes from a human (Homo Sapiens) and the other sequence represents a chromosome from a soybean (Glycine Max).

   Help Doctor Jean by parsing in the following sequences into memory (be careful of newlines!) and applying the Longest Common Subsequence algorithm learned in recitation to each unique pair of sequences.

   Unzip the "sequence_data.zip" file located on Canvas to access the data. When backtracing through the computed two dimensional matrix to find the longest common subsequence, break ties uniformly at random. Compare the results from your implementation to determine which species the sequences pertain to (try to come up with a sensible metric that will compare the results).

   (a) Show a table that maps the sequence to the species.

   (b) Submit your separate python (.py) file along with your PDF submission.

   a).By using lcs function(longest common substring) and comparing the longest common substring sequence A with B, sequence A with C and sequence B with C, the AB result is 2017, the AC result is 1985 and BC result is 2434, then because sequence B and C have the longest common substring, then B and C are both human, and A is soybean.

Sequence A	Soybean
Sequence B	Human
Sequence C	Human

analysis.png

b). Implementation on separate file.

2. (15 pts) Draco Malfoy is struggling with the problem of making change for n cents using the smallest number of coins. Malfoy has coin values of $v_1 < v_2 < \ldots < v_r$ for r coins types, where each coin's value v_i is a positive integer. His goal is to obtain a set of counts $\{d_i\}$, one for each coin type, such that $\sum_{i=1}^{r} d_i = k$ and where k is minimized.

   (a) A greedy algorithm for making change is the **wizard's algorithm**, which all young wizards learn. Malfoy writes the following pseudocode on the whiteboard to illustrate it, where n is the amount of money to make change for and v is a vector of the coin denominations:

```
wizardChange(n,v,r) :
   d[1 .. r] = 0       // initial histogram of coin types in solution
   while n > 0 {
      k = 1
      while ( k < r and v[k] > n ) { k++ }
      if k==r { return 'no solution' }
      else { n = n - v[k] }
   }
   return d
```

   Hermione snorts and says Malfoy's code has bugs. Identify the bugs and explain why each would cause the algorithm to fail.

   (b) Sometimes the goblins at Gringotts Wizarding Bank run out of coins,[1] and make change using whatever is left on hand. Identify a set of U.S. coin denominations for which the greedy algorithm does not yield an optimal solution. Justify your answer in terms of optimal substructure and the greedy-choice property. (The set should include a penny so that there is a solution for every value of n.)

   (c) On the advice of computer scientists, Gringotts has announced that they will be changing all wizard coin denominations into a new set of coins denominated in powers of c, i.e., denominations of $c^0, c^1, \ldots, c^\ell$ for some integers $c > 1$ and $\ell \geq 1$. (This will be done by a spell that will magically transmute old coins into new coins, before your very eyes.) Prove that the wizard's algorithm will always yield an optimal solution in this case.

   Hint: first consider the special case of $c = 2$.

   a).
   1. The algorithm did not use the variable d, according to the write up, d is corresponding to v, therefore, once we use any kind of coin denominations, we also need to update d in the corresponding index.
   2.In the second while loop($v[k] > n$), the condition is always checking the

[1] It's a little known secret, but goblins like to *eat* the coins. It isn't pretty for the coins, in the end.

coin from the smallest to largest, but we need to check from largest to smallest, so we need to change it from $v[k] > n$, to $v[len(k) - k] > n$, so that it could check from largest denominations to smallest denominations.
3. In the second while loop, the algorithm set if $k < r$ (suppose $v[k] > n$ is correct here), increase the value of k. However, in the if statement, the algorithm also said if k==r, return no solution, but since the condition to increase k value is $k < r$, therefore, k will never equal to r. In addition, if $k < r$, the algorithm will never touch the smallest denominations of the coins, therefore, we need to change it to $k <= r$, so that it go through entire coin denominations.
4.The condition of no solution is also wrong, it should be $k > r$ (because k could equal to r which indicates k we go the last coin denomination)

b).
For greedy algorithm, if we need to fulfill the greedy choice property, we always have to choose from the largest to second largest to the smallest(if we need to go until smallest to fulfill the requirement) until the sum value of the coins is the same with the value given to, by choosing coin using greedy algorithm it will not always have the optimal solution(does not provide an optimal substurcture). For example we have coin denominations 30, 17, 1 and the money we need to exchange is 34, by using greedy algorithm, we need to use one 30 denomination coin and four 1 denomination coins(30+1+1+1=34), but the optimal solution is choosing two 17 denomination coins(17+17=34), in this case the greedy will not always have the optimal solution.
c).
proposition: For all $c > 0$ and c is an integer, $i >= 0$ and i is an integer, $\sum_0^i c^i$, the $(\sum_0^{i-1} c^i) < c^i$
Proof by indcution:
Base case: there are two elements in the summation, in this case $\sum_0^i c^k$=c^0+ c^1, $c^0 = 1$ and since c is an positive integer, $c^1 > c^0$, which holds the proposition.
Inductive hypothesis: Suppose it is true that for all $c > 0$ and c is an integer, $i >= 0$ and i is an integer, $\sum_0^i c^i$, the $(\sum_0^{i-1} c^i) < c^i$
Inductive step: Suppose the inductive hyphthesis is true. Then we need to prove that it is also true that for $\sum_0^{i+1} c^i$, the $(\sum_0^i c^i) < c^{i+1}$, $(\sum_0^i c^i)$=c^0+ c^1+c^2+...+c^i=$(\sum_0^{i-1} c^i)$+c^i, and $c^{i+1} = c*c^i$, because$(\sum_0^{i-1} c^i)$ is positive interger and $c^{i+1} = c^i$ is also positive integer, by the inductive hyphthesis, $(\sum_0^{i-1} c^i) < c^i$, because c is a positive integer and greater than 1, therefore $(\sum_0^i c^i) < c^{i+1}$
Conclusion:For all $c > 0$ and c is an integer, $i >= 0$ and i is an integer, $\sum_0^i c^i$, the $(\sum_0^{i-1} c^i) < c^i$

Proposition: $c^0, c^1, \ldots, c^\ell$ for some integers $c > 1$, greedy algorithm support an optimal solution Proof by induction:

Base case: For only c^0 coin denomination, to exchange n(n is an positive integer) amount money, its need n coins, so greedy algorithm is the optimal solution, which the greedy algorithm holds.
Inductive step: For c^n coin denomination, because $c^0 + c^1 + \ldots + c^{n-1} < c^n$(proved above), therefore, by always choosing the largest coin denomination that is samller or equal to the money we want to exchange, using greedy algorithm will have the least amount of coin use.
Conclusion: $c^0, c^1, \ldots, c^\ell$ for some integers $c > 1$, greedy algorithm support an optimal solution.

3. In the two-player game "Pandas Peril", an even number of cards are laid out in a row, face up. On each card, is written a positive integer. Players take turns removing a card from either end of the row and placing the card in their pile. The player whose cards add up to the highest number wins the game. One strategy is to use a greedy approach and simply pick the card at the end that is the largest. However, this is not always optimal, as the following example shows: (The first player would win if she would first pick the 4 instead of the 5.)

   4 2 10 5

   (a) (10 pts) Write a dynamic programming algorithm for a strategy to play Pandas Peril. Player 1 will use this strategy and Player 2 will use a greedy strategy of choosing the largest card.

   (b) (10 pts) Prove that your strategy will do no worse than the greedy strategy for maximizing the sum of each hand.

   (c) (10 pts) Implement your strategy and the greedy strategy in Python and simulate a game, or two, of Pandas Peril. Your simulation should include a randomly generated collection of cards and show the sum of cards in each hand at the end of the game.

   a).

```
optimal(arr){
    size=length(arr)
    if(size%2!=0){
        return NULL
        //The game have to have even number of cards
    }
    solution=[[0 for i in range(size)] for j in range(size)]
    //create a size* size matrix and initialize all to 0
    for i in (0,size){
        for j in (i,size){
            temp=j-i
            if(i>=2){
            //if it is comparing at least 3 numbers
```

```
                    first=solution[temp+1][j-1]
                    second=solution[temp][j-2]
                    third=solution[temp+2][j]
                    solution[temp][j]=max(arr[temp]+min(first,third),arr[j]+min(second,
                }else{
                    solution[temp][j]=max(arr[temp],arr[j])
                }
            }
        }
        return solution[0][size-1]
    }
```

b).
Proof by Induction:
Base case: For 2 cards, if the value of two cards are the same, then, it the value for player 1(optimal) and player 2(greedy) are the same which holds dynamic programming is not worse than greedy algorithm. If the value of two cards are different, because the dynamic programming always picking the larger one by comparing the two value, and the player use dynamic programming pick first, so dynamic programming is better than greedy algorithm which also holds dynamic programming is not worse than greedy algorithm.
Inductive hypothesis: Suppose for 2n cards, the dynamic programming algorithm will do no worse than the greedy strategy for maximizing the sum of each hand.
Inductive step: Suppose the inductive hypothesis is true. Then for 2(n+1) cards, because the dynamic programming algorithm also add the two cases and comparing the value of two cases and picking the lager one by back tracking ((solution[temp][j]=max(arr[temp]+min(first,third),arr[j]+min(second,first))), and greedy algorithm is just picking the larger one, so greedy algorithm is a subset of dynamic programming algorithm (It is possible that greedy algorithm and dynamic programming algorithm have the same sum value, but dynamic programming algorithm is still no worse than greedy algorithm).
Conclusion: Therefore, for 2(n+1) cards, the dynamic programming algorithm will do no worse than the greedy strategy for maximizing the sum of each hand.
c). Implementation on separate file

4. A common problem in computer graphics is to approximate a complex shape with a bounding box. For a set, S, of n points in 2-dimensional space, the idea is to find the smallest rectangle, R. with sides parallel to the coordinate axes that contains all the points in S. Once S is approximated by such a bounding box, computations involving S can be sped up. But, the savings is wasted if considerable time is spent constructing R, therefore, having an efficient algorithm for constructing R is crucial.

(a) (10 pts) Design a divide and conquer algorithm for constructing R in $O(\frac{3n}{2})$ comparisons.

(b) (10 pts) Implement your algorithm in Python. Generate 50 points randomly and show that your bounding box algorithm correctly bounds all points generated. Your code should output the list of points, as well as the coordinates of R. You should include an explanation of the results in your pdf file with your algorithm.

a).

```
maxMin(arr){
    tempMax=0
    tempMin=0
    startIndex=0
    if(len(arr)%2==0){
        //if there are even number of elements in the array
        //compare the first two element, and set it to min and max
        if(arr[0]>arr[1]){
            tempMax=arr[0]
            tempMin=arr[1]
        }else{
            tempMax=arr[1]
            tempMin=arr[0]
        }
        isEven=2
    }else{
        //if there is odd number of element in the array
        //set the first element as both max and min
        tempMax=arr[0]
        tempMin=arr[1]
        startIndex=1
    }
    for(i=startIndex;i<=length(arr)-1); i+2){
        if(arr[i+1]>arr[i]){
            //check the value of arr[i+1] and arr[i]
            if(arr[i+1]>tempMax){
                tempMax=arr[i+1];
            }
            if(arr[i]<tempMin){
                tempMin=arr[i]
            }
        }
        else if(arr[i+1]<arr[i]){
            if(arr[i]>tempMax){
                tempMax=arr[i]
            }
```

```
                if(arr[i+1]<tempMin){
                    tempMin=arr[i+1]
                }
            }
        }
    }
```

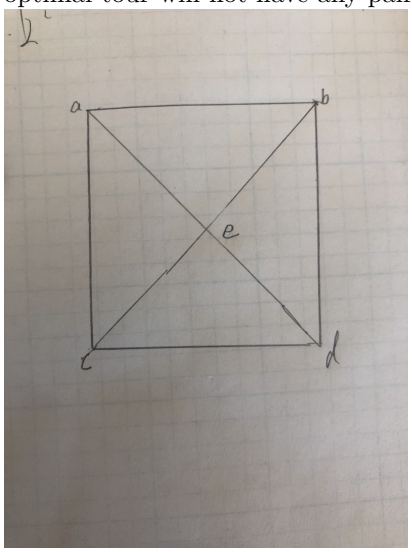
The algorithm constructs an R in O($\frac{3n}{2}$) comparisons. For example, an array $x = [4, 9, 1, 10]$, then the function first sets the min value to be 4 and max value to be 9(1 comparsion). Then the function compare the value or 1 and 10(1 comparsion), and use the larger value to compare with current min and max value(4 and 9), if the current min is smaller than the local min(which is 1), then change the min value to 1(1 camparsion), if the current max is smaller than the local max(which is 10), then change the min value to 10(1 camparsion), so for this case, the algorithm did total of 4 camparsions, which is smaller than O($\frac{3*4}{2}$)=6 camparsions.
b).Implementation on seperate file
For the implementation of the the algorithm, it first checks if the length of the input array is even of odd. If it is even, set the bigger one of the first two elements to be max(temporarily) and the small one to of the first elements to be min(temporarily). If it is odd, set the first element of array to be both min and max(both temporarily). Then, in the for loop, if the number of elements in array is even, starts from the third element of the array, check two elements one time, use the bigeer one to compare the temporarily max value(dividing and conquer), if it is bigger than that, change the temporarily max value to the local max. Then compare the local min value with the temporarily min value, if the local min value is smaller than temporarily min value, change the temporarily min value to the local min value. By keep doing that until the end of the array, the temporarily max and min will be the max and min value for the whole array. If the number of elements in array is odd, starts from the second elements of the array, and repeat the steps in the for loop, until its go through the whole array, then the temporarily max and min is the max and min value for the whole array.

5. (10 pts) Professor Voldemort has designed an algorithm that can take any graph G with n vertices and determine in $O(n^k)$ time whether G contains a clique of size k. Has the Professor just shown that $P = NP$? Why or why not?

   The algorithm that the professor provided does not show $P = NP$, because the value of k is not a constant, since k is a variable, so we could not decide what k value would be (there are infinite possible solution since k's value is not unique). In addition, for example n and k's relationship are exponential, then P will no longer equal to NP(because it is not polynomial). Therefore, by having an undicided k value(variable k), P is not

equal to NP

6. (10 points) Consider the special case of TSP where the vertices correspond to points in the plane, with the cost defined for an edge for every pair (p, q) being the usual Euclidean distance between p and q. Prove that an optimal tour will not have any pair of crossing edges.



Proof by contradiction:
For the sake of contradiction, suppose an optimal tour will have any pair of crossing edges.
Suppose there are four vertices in a grapha(as the picture above), by saying there are crossing edges in the graph means the vertices in the graph cross each other diagonally (because in order to have a crossing edge in a 4 vertices graph, the vertices have to go to the other one diagonally). Therefore, there have to be an intersection point in the graph(showing in the graph above). By the definition of triangle (the sum of two side have to greater than the other side), $ec + ed > cd$, $ae + ec > ac$, $ae + eb > ab$, $eb + ed > bd$, by add all these up, $ec + ed + ae + ec + ae + eb + eb + ed > cd + ac + ab + bd$, $ec + ed + ae + ec + ae + eb + eb + ed = ad + cb$, since the sum of all crossing edge path is greater than the path creates non-crossing edges. Therefore, the crossing edge path is not optimal tour.