

# CS180 Exam 2

Hao Wang

TOTAL POINTS

**22 / 22**

QUESTION 1

7 pts

1.1 Kruskal's algorithm 1 / 1

- 0 Correct algorithm

1.2 Cut property 1 / 1

- 0 Property stated correctly

1.3 MST change when squaring weights 1 / 1

- 0 Correct answer

1.4 WIS: value-to-finish time 2 / 2

- 0 Correct answer, with a valid example showing why the greedy algorithm won't work

1.5 Greedy for same value knapsack 2 / 2

- 0 Correct algorithm, that sorts the items by weight, and fills up the knapsack

QUESTION 2

2 Proof of cycle property 3 / 3

- 0 Correct proof.

QUESTION 3

3 Knapsack with 3 copies 4 / 4

- 0 Correct algorithm

QUESTION 4

4 Most valuable subsequence 4 / 4

- 0 Correct algorithm.

QUESTION 5

5 RNA with squared norm stability 4 / 4

- 0 Correct.

# Mid-term. February 24, 2017

CS180: Algorithms and Complexity  
Winter 2017

Guidelines:

- The exam is closed book and closed notes. Do not open the exam until instructed to do so.
- Write your solutions clearly and when asked to do so, provide complete proofs.
- Unless told otherwise you may use results and algorithms we proved in class without proofs or complete details as long as you state what you are using.
- I recommend taking a quick look at all the questions first and then deciding what order to tackle to them in. Even if you don't solve the problems fully, attempts that show some understanding of the questions and relevant topics will get reasonable partial credit.
- You can use extra sheets for scratch work, but try to use the white space (it should be more than enough) on the exam sheets for your final solutions.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly and any cheating reported.

Problem	Points	Maximum
1		7
2		3
3		4
4		4
5		4
Total		22

Name	Hao Wang
UID	204681263
Section	1C



# 1 Problem

The answers to the following should fit in the white space below the question.

1. Write down Kruskal's algorithm. It is sufficient to write down the main while loop and the rule describing how the algorithm proceeds. [1 point]

while  $R \neq \emptyset$   
pick the edge  $e$  of least weight.  
check if adding  $e$  to our output  $T$  creates a cycle. if not, add  $e$  to  $T$ .  
delete  $e$ .

2. State the *cut property* we used in class to analyze Kruskal's and Prim's algorithms. [1 point]

( ① Adding an edge that crosses the cut does not create a cycle. )

② If an edge  $e$  is the minimum weight edge across some cut, then it is in a MST.

3. Suppose we are given an instance of the Minimum Spanning Tree Problem on a graph  $G$ , with edge costs that are all positive and distinct. Let  $T$  be a minimum spanning tree for this instance. Now suppose we replace each edge cost  $c_e$  by its square,  $c_e^2$ , thereby creating a new instance of the problem with the same graph but different costs.

True or false:  $T$  must still be a minimum spanning tree for this new instance. [1 point]

True.



4. Consider the weighted interval scheduling problem where we are given  $n$  jobs as input with the  $i$ 'th job having start time  $s_i$ , finish time  $f_i$ , and value  $v_i$ . (Thus, the input to the problem is  $n$  triples  $(s_1, f_1, v_1), \dots, (s_n, f_n, v_n)$ .) Recall that our goal is to find the set of non-conflicting jobs with the highest possible total value. Consider the following greedy algorithm for the question:

- (a) Set  $A = \emptyset$ ,  $R = \{1, 2, \dots, n\}$ .
- (b) While  $R \neq \emptyset$ :
  - i. Pick job  $i \in R$  with highest  $v_i/f_i$  (value to finish time ratio) and add  $i$  to  $A$ .
  - ii. Remove  $i$  and all jobs that conflict with  $i$  from  $R$ .
- (c) Return  $A$ .

True or false:  $A$  achieves the highest possible total value. If true, provide a brief explanation. If false, provide a counterexample. [2 points]

False Consider  $j_1 = (1, 3, 6)$  and  $j_2 = (2, 10, 10)$

$$\frac{v_1}{f_1} = 2 > \frac{v_2}{f_2} = 1$$



but picking  $j_2$  would result in higher total possible value

5. You have  $n$  items with the  $i$ 'th item having weight  $w_i$ . You also have a knapsack with total weight capacity  $W$  (i.e., it can safely hold items whose total weight is at most  $W$ ). Describe an algorithm for picking a largest possible subset of items that can be placed safely in the knapsack. That is, describe an algorithm to find a subset  $S \subseteq \{1, 2, \dots, n\}$  of maximum possible size such that  $\sum_{i \in S} w_i \leq W$ . For full-credit, your algorithm should run in time  $O(n \log n)$ . You don't have to prove correctness or analyze the time complexity of the algorithm. [2 points]

[Hint: One approach is to give a greedy algorithm.]

Sort the items according to its weight  $w_i$ .  $A = \emptyset$

While the item list is not empty  
pick the item with the smallest weight  $w_i$

if  $W \geq w_i$

add item  $i$  to  $A$

remove item  $i$  from the item list

else  $W = W - w_i$

Return  $A$

Return  $A$



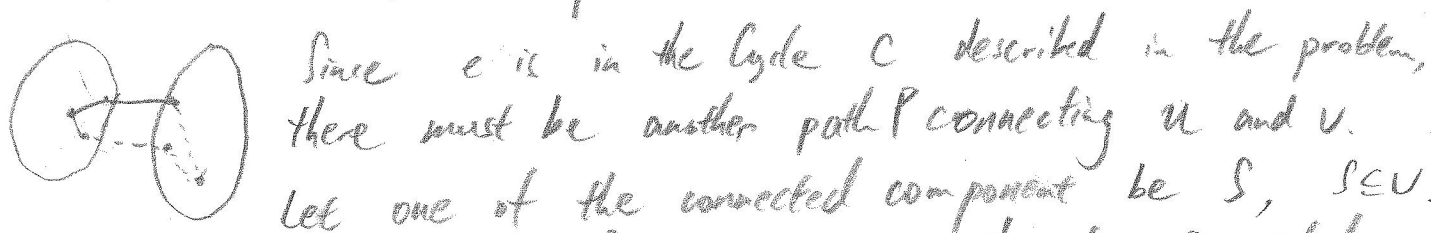
## 2 Problem

Suppose you have a weighted undirected graph  $G = (V, E)$  where all the weights are distinct. Prove that if an edge  $e$  is part of a cycle  $C$  and has weight more than every other edge in the cycle, then  $e$  cannot be part of the minimum spanning tree in  $G$ . [3 points]

[Hint: Assume that the statement is false for the sake of contradiction and let  $T$  being a MST that contains the edge  $e$ . Arrive at a contradiction by a swapping argument as we did in class for proving the cut property.]

Proof by contradiction. Suppose  $T$  is an MST that contains an edge  $e = \{u, v\}$  that is described in the problem.

If we remove  $e$  from  $T$ , we will get two connected components because  $T$  is an MST.



Since  $e$  is in the cycle  $C$  described in the problem, there must be another path  $P$  connecting  $u$  and  $v$ .

Let one of the connected components be  $S$ ,  $S \subseteq V$ .

$S$  is a cut of  $T \setminus \{e\}$  since the two connected

components are not connected. Note that  $e$  is the maximum weight edge on cycle  $C$ , so on the other path  $P$  that connects  $u, v$ , there must be an edge  $e'$  that has smaller weight than  $e$ , and connects the two connected components.

① observe  $w(e) > w(e')$

$$w(T) > w(T \setminus \{e\} + \{e'\})$$

②  $T \setminus \{e\} + \{e'\}$  is still a spanning tree, because  $e'$  is an edge across the cut and adding  $e'$  doesn't create cycles. Also since  $e'$  connects two connected components,  $T \setminus \{e\} + \{e'\}$  spans all vertices.

Thus it is a spanning tree with smaller weight than  $T$ , and thus  $T$  is not an MST. Contradiction! ☹





### 3 Problem

Give a dynamic programming algorithm for the following version of knapsack where you have three copies of each item. There are  $n$  types of items with weights  $w_1, \dots, w_n$  respectively and value  $v_1, \dots, v_n$  respectively and you have three copies of each item. Suppose you have a knapsack of total weight capacity  $W$ . We say configuration  $(a_1, \dots, a_n)$  is safe if  $0 \leq a_i \leq 3$  and  $a_1 w_1 + a_2 w_2 + \dots + a_n w_n \leq W$  (i.e., it is safe to pack  $a_1$  copies of item 1,  $a_2$  copies of item 2,  $\dots$ ,  $a_n$  copies of item  $n$  into the knapsack). The value of a configuration is the total value of the items in the configuration: for a configuration  $(a_1, \dots, a_n)$ , its value is  $v_1 a_1 + v_2 a_2 + \dots + v_n a_n$ .

Give an algorithm which given the numbers  $w_1, \dots, w_n, v_1, \dots, v_n, W$  as input computes the maximum value achievable over all safe configurations. For full-credit it is sufficient to give a correct algorithm for the problem which runs in time  $O(nW)$  and it is not required to prove correctness or analyze the time-complexity of the algorithm. You must provide full description of the algorithm. [4 points]

Initialize array  $M(i, j)$ , with  $M(i, 0) = 0$  for  $\forall i = 0, \dots, 3n$   
 $M(0, w) = 0$  for  $\forall w = 1, \dots, W$   
and the rest of  $M = \text{null}$ .

$j = 0$

For  $k = 1$  to  $n$

For  $i = 1$  to  $3$

$w'[i] = w_k$

$v'[i] = v_k$

Let  $m = 3n$

$\text{OPT}(m, W)$

If  $M(m, W)$  is not null

Return  $M(m, W)$ .

Else

$M(m, W) = \max\{v'[m] + \text{OPT}(m-1, W - w'(m)), \text{OPT}(m-1, W)\}$

Return  $M(m, W)$



## 4 Problem

You are given two arrays of integers  $X = [x[0], x[1], \dots, x[m]]$  and  $Y = [y[0], y[1], \dots, y[n]]$  as input. For two subsequences of  $X, Y$  of the same length, i.e., sequences of indices  $0 \leq i_1 < i_2 < \dots < i_k \leq m$  and  $0 \leq j_1 < j_2 < \dots < j_k \leq n$ , the value of the subsequences is defined as

$$\sum_{\ell=1}^k \frac{1}{1 + |x[i_\ell] - y[j_\ell]|}$$

Give an algorithm that given  $X, Y$  as input computes the maximum possible value achievable over all subsequences. For full-credit, your algorithm should run in time  $O(mn)$  (ignoring the cost of arithmetic, i.e., adding numbers). You don't have to prove correctness or analyze the time-complexity of the algorithm. [4 points]

Example:  $X = [1, 4, 2, 5]$ ,  $Y = [1, 2, 10, 4, 100]$ . Here, if you look at subsequences  $x[0], x[2], x[3]$  and  $y[0], y[1], y[3]$  you get value  $1/1 + 1/1 + 1/2 = 2.5$ . Whereas, if look at subsequences  $x[0], x[1], x[2], x[3]$  and  $y[0], y[1], y[2], y[3]$ , you get value  $1/1 + 1/3 + 1/9 + 1/2 \sim 1.9444$ . So the first subsequence has better value. Your goal is to find the best possible value achievable over all subsequences.

[Hint: Create subproblems like we did for edit-distance in class and develop the appropriate recurrence.]

$m, n \geq 0$      $OPT(m-1, n-1) + \frac{1}{1 + |x[m] - y[n]|}$   
 $n \neq 0$      $OPT(m, n-1)$   
 $m \neq 0$      $OPT(m-1, n)$

Initialize an array  $M[i, 0] = 0$  for  $\forall i = 1, \dots, m$   
 $M[0, j] = 0$  for  $\forall j = 1, \dots, n$

$OPT(m, n)$   
 For  $i = 1, \dots, m$   
 For  $j = 1, \dots, n$   
 $M[i, j] = \max \left\{ M[i-1, j-1] + \frac{1}{1 + |x[i] - y[j]|}, M[i, j-1], M[i-1, j] \right\}$

Return  $M[m, n]$



## 5 Problem

Consider the following variant of the RNA sequencing question. Given a sequence  $X = (x_1, \dots, x_n)$ , a set of pairs  $M = \{(i_1, j_1), (i_2, j_2), \dots, (i_m, j_m)\}$  is an *allowed* set of pairs if the following hold:

1. Each index appears in at most one pair in  $M$  (i.e., no repetitions).
2. Each pair is one of  $\{G, C\}$  or  $\{A, U\}$ . That is, for all  $1 \leq p \leq m$ ,  $\{x_{i_p}, x_{j_p}\}$  is one of  $\{G, C\}$  or  $\{A, U\}$ .
3. No sharp edges: For all pairs  $(i, j) \in M$ ,  $i < j - 4$ .
4. No crossing edges: If pairs  $(i, j), (k, \ell) \in M$ , then we cannot have  $i < k < j < \ell$ .

(These are the same rules as we worked with in class.)

The *stability* of an allowed set of pairs  $M$  is given by the following formula:

$$\text{stability}(M) = \sum_{p=1}^m (j_p - i_p)^2.$$

That is, the stability of the collection of pairs is the sum of squares of the number of characters between each pair. Give an efficient algorithm that given a sequence  $X = (x_1, \dots, x_n)$  computes the maximum possible  $\text{stability}(M)$  over all feasible sets of pairs  $M$ . For full-credit, your algorithm should run in  $O(n^3)$  time. You do not have to prove correctness or analyze the time complexity of the algorithm. [4 points]

AGUC AUUCCU

Initialize an array  $M$ , with  $M[i, i+k] = 0$  for  $\forall i = 1, \dots, n$   
 $k = 0, 1, 2, 3, 4$   
 $M[i, j] = 0$  if  $j < i$  for  
 $\forall i, j = 0, 1, \dots, n+1$

For  $k = 5, \dots, n-1$   
 For  $i = 1, \dots, n-k$   
 $j = i+k, M[i, j] = 0$   
 For  $t = i+1, \dots, j$   
 If  $x_t$  forms  $\{G, C\}$  or  $\{A, U\}$  with  $x_i$   
 $\text{temp} = (t-i)^2$   
 Else  
 $\text{temp} = 0$   
 If  $M[i, j] < \text{temp} + M[i+1, t-1] + M[t+1, j]$   
 $M[i, j] = \text{temp} + M[i+1, t-1] + M[t+1, j]$

Return  $M[1, n]$

