CS180 Exam 2

Nicole Wong

TOTAL POINTS

20.25 / 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 **1.25 / 3**

- **1.75 Moderate attempt**

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	NICOLE WONG
UID	[redacted]
Section	FRI 12-2 ANG LI (DIS 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]

   Let $R \leftarrow V$. Let $T \leftarrow \emptyset$.
   While R is not empty:
   choose the edge e with smallest weight in R
   If adding e to T does not create a cycle,
   add e to T
   remove e from R

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

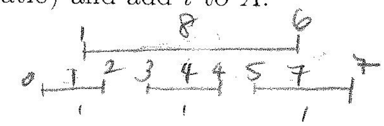
   For a cut $S \subseteq V$, the minimum weight edge crossing the cut is part of a minimal spanning tree.

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]

   $\boxed{\text{True}}$ For MSTs, only the order of edgeweights matter and squaring does not change order.

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 $(s_1, f_1, v_1), \ldots, (s_n, f_n, v_n)$.) 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 $A = \emptyset$, $R = \{1, 2, \ldots, n\}$.
   (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]

$\boxed{\text{False}}$

Given the jobs (1, 6, 8) (0, 2, 1) (3, 4, 4) (5, 7, 7)

The job with the highest v_i/f_i is (1, 6, 8)
All of the jobs conflict with this one, so we remove all the jobs from R.

A contains only job (1, 6, 8) with value = 8.
However, the set of jobs with highest total value that are nonconflicting is (0, 2, 1) (3, 4, 4) (5, 7, 7) with value = 12.

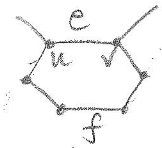
5. You have n items with the i'th item having weight w_i. You also have a knapsack with total weight capacity W (i.e., it can safely hold items whose total weight is at most W). Describe an algorithm for picking a *largest* possible subset of items that can be placed safely in the knapsack. That is, describe an algorithm to find a subset $S \subseteq \{1, 2, \ldots, n\}$ 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 algoritm. [2 points]

Maximize $|S|$ while maintaining $\sum_{i \in S} w_i \leq W$

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

let R ← list of n items sorted in increasing order of weight (use Mergesort)
let S ← $\emptyset$, available weight ← W.
While R is not empty:
    Select item i from R with least weight
    If available weight − w_i > 0:
        add i to S
        remove i from R
        available weight ← available weight − w_i
    Else
        return S.

Return S.



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:

Assume that the statement is false and let T be a MST that contains the edge e. Suppose that another edge f is also part of C, but with less weight than e. Since they belong to the same cycle, you can take a path through f instead of e to achieve the MST, and this would have a smaller weight. This contradicts the statement that T is an MST.

$$OPT = \begin{cases} v_j + OPT(j-1, w) - w_j \\ OPT(j-1, w) \end{cases}$$

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 $w_1, \ldots, w_n$ respectively and *value* $v_1, \ldots, v_n$ respectively and you have **three** copies of each item. Suppose you have a knapsack of total weight capacity W. We a say configuration $(a_1, \ldots, a_n)$ is *safe* if $0 \leq a_i \leq 3$ and $a_1 w_1 + a_2 w_2 + \ldots + a_n w_n \leq W$ (i.e., it is safe to pack a_1 copies of item 1, a_2 copies of item 2, ..., 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 $(a_1, \ldots, a_n)$, its value is $v_1 a_1 + v_2 a_2 + \cdots + v_n a_n$.

Give an algorithm which given the numbers $w_1, \ldots, w_n$, $v_1, \ldots, v_n$, 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]

let OPT(j, w) = max value achievable over j items with weight limit w

$$= \max \begin{cases} OPT(j-1, w) \\ v_j + OPT(j-1, w) \text{ if } w_j < w \\ 2v_j + OPT(j-1, w-2w_j) \text{ if } 2w_j < w \\ 3v_j + OPT(j-1, w-3w_j) \text{ if } 3w_j < w \end{cases}$$

Implentation //Initialize

OPT(0, w) = 0 ∀ w from 0 to W

OPT(j, 0) = 0 ∀ j from 0 to n

For j = 1 ... n

  For w = 1 ... W

    compute OPT using recurrence above

Return OPT(n, W)

edit dist

$$\underset{(i,j)}{opt} = \min \begin{cases} OPT(i-1, j-1) + \mathbb{1}(x_i \neq y_j) \\ 1 + OPT(i-1, j) \\ 1 + OPT(i, j-1) \end{cases}$$

index i of X

index j of Y

Opt soln θ

4 Problem

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

let OPT (i, j) = best possible value achievable over $[X[0] \cdots X[i]]$ and $[Y[0] \cdots Y[j]]$

$$= \max \begin{cases} \frac{1}{1+|X[i]-Y[j]|} + OPT(i-1, j-1) \\ OPT(i-1, j) \\ OPT(i, j-1) \end{cases}$$

Implementation : initialize OPT : $OPT(0,j) = 0 \ \forall j \{1 \ldots n\}$, $OPT(i,0) = 0 \ \forall i \{1 \ldots m\}$

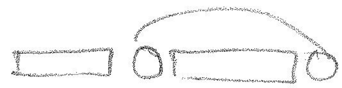
For $i = 0 \ldots m$

For $j = 0 \ldots n$

compute OPT (i,j) using recurrence above

return OPT(m,n)

RNA-Seq.



$$OPT(i,j) = \max \begin{cases} \max\limits_{t} \; 1 + \\ \text{for } t < j-4 \\ \\ OPT(i, j-1) \end{cases}$$

5 Problem

Consider the following variant of the RNA sequencing question. Given a sequence $X = (x_1, \ldots, x_n)$, a set of pairs $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)\}$ 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 $1 \le p \le m$, $\{x_{i_p}, x_{j_p}\}$ 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:

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

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 $X = (x_1, \ldots, x_n)$ computes the maximum possible *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]

let OPT(i,j) = max possible stability over feasible sets of pairs M from x_i to x_j

$$= \max \begin{cases} OPT(i, j-1) \\ \max\limits_{t \text{ for } t<j-4 \text{ where } t \text{ matches } x_j} OPT(i, t-1) + OPT(t+1, j-1) + (j-1)^2 \end{cases}$$

Implementation:

initialize: OPT$(i, i+k) = 0$ $\forall k = \{0, 1, 2, 3, 4\}$

For $k = 5 \ldots n$

For $i = 1 \ldots n-k$

$j = i+k$

compute OPT(i,j) using recurrence above, iterating over all possible t

return OPT$(1,n)$