CS180 Exam 2

Koyoshi Shindo

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	Koyoshi Shindo
UID	004630357
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]

Initialize $T = \phi$, TEMP_E = E
while TEMP_E $\neq \phi$ {
pick edge e with least c_e from TEMP_E
if e does not create a cycle in T
add e into T
remove e from TEMP_E
}

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

Cut property: the edge that crosses cut (a set of connected vertices) with minimum weight among all crossing edges has to be part of MST

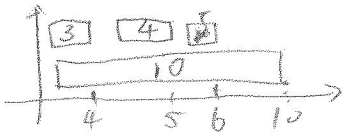
3. Suppose we are given an instance of the Minimum Spanning Tree Problem on a graph G, with edge costs that are all positive and distinct. Let T be a minimum spanning tree for this instance. Now suppose we replace each edge cost c_e by its square, c_e^2, thereby creating a new instance of the problem 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. consider Kruskal's algorithm, where it picks edge from lowest edge to largest appropriate edge. there is one unique way how kruskal goes. Squaring edge cost does not change the relationship which edge is larger, ie. if $C_a < C_b$ then $C_a^2 < C_b^2$ must hold true.

doesn't change how Kruskal algorithm goes

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:



v_i
f_i = 1

(a) Set $A = \emptyset$, $R = \{1, 2, \ldots, n\}$.

(b) While $R \neq \emptyset$:

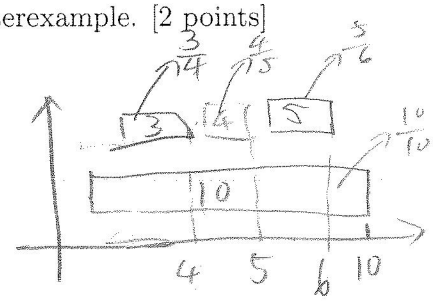
i. Pick job $i \in R$ with highest v_i/f_i (value to finish time ratio) and add i to A.

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]

False.



Algorithm will get 10

But optimal is 3+4+5=12.

#3

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 algorithm. [2 points]

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

Sort the item in increasing order of w_i $\leftarrow O(n \log n)$

go from left to right, choose least weight item each time

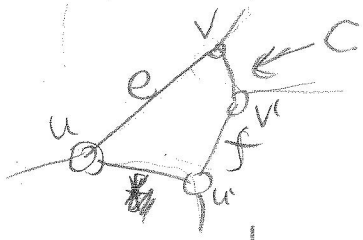
until $\sum w_i \in W$ can no longer be satisfied $\leftarrow O(n)$

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.]

Assume that the statement is false, and there exists an edge e that is part of C that is heaviest in its cycle that is part of MST, let T be that MST.



Because T is a tree, there must exist an edge "f" such that f is in cycle C but $f \notin T$. (If above is false, T will have all edges in cycle C, which means T is acyclic, by contradiction, so such "f" must exist)

By definition, weight(e) > weight(f)

Say that $e = \{u, v\}$, $f = \{u', v'\}$.

Because T is connected, there must exist path $\{u, \cdots, u'\}$ and $\{v, \cdots, v'\}$

if we remove e and replace it with f, obtain T'

T' is still connected because every path that uses $\{u, v\}$ can use $\{u, \cdots u', v', \cdots v\}$ (f)

T' is still a spanning tree.

And weight of T decreases $\because e > f$, weight(T) > weight(T')

~~hence original T~~

which is a contradiction $\because T$ is supposed to be minimum,

by contradiction, e cannot be part of MST in G.

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 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]

Recurrence relation:

$$OPT(n,W) = \max \begin{cases} OPT(n-1, W) \\ v_n + OPT(n-1, W - w_n) \\ 2v_n + OPT(n-1, W - 2w_n) \\ 3v_n + OPT(n-1, W - 3w_n) \end{cases} \quad \text{if } W \geq 3w_n$$

$$= \max \begin{cases} OPT(n-1, W) \\ v_n + OPT(n-1, W - w_n) \\ 2v_n + OPT(n-1, W - 2w_n) \end{cases} \quad \text{if } W \geq 2w_n$$

$$= \max \begin{cases} OPT(n-1, W) \\ v_n + OPT(n-1, W - w_n) \end{cases} \quad \text{if } W \geq w_n$$

$$= OPT(n-1, W) \quad \text{if } W < w_n$$

$$= 0 \quad \text{if } n = 0 \text{ or } W = 0$$

Recursive implementation: Call Compute-Opt(n, W) to solve. Iterated sol...

Initialize $M[n][W]$, $M[0][:] = 0$, $M[:][0] = 0$

```
Compute-Opt(n, w) {
    if M[n][w] ≠ φ
        return M[n][w]
    if w < w_n
        M[n][w] = Compute-Opt(n-1, w)
        return M[n][w]
    if w ≥ w_n
        M[n][w] = max(Compute-Opt(n-1, w), v_n + Compute-Opt(n-1, w-w_n))
        return M[n][w]
    if w ≥ 2w_n
        M[n][w] = max(Compute-Opt(n-1, w), v_n + Compute-Opt(n-1, w-w_n), 2v_n + Compute-Opt(n-1, w-2w_n))
        return M[n][w]
    if w ≥ 3w_n
        M[n][w] = max(Compute-Opt(n-1, w),
                      v_n + Compute-Opt(n-1, w-w_n),
                      2v_n + Compute-Opt(n-1, w-2w_n),
                      3v_n + Compute-Opt(n-1, w-3w_n))
    return M[n][w]
}
```

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 $0 \le i_1 < i_2 < \ldots < i_k \le m$ and $0 \le j_1 < j_2 < \ldots < j_k \le n$, 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.]

Recurrence:

$$OPT(m,n) = \begin{cases} \max \begin{cases} OPT(m-1,n-1) + \frac{1}{1+|x_m - y_n|} \\ OPT(m-1,n) \\ OPT(m,n-1) \end{cases} & \text{otherwise} \\ 0 & \text{if } m<0 \text{ or } n<0 \end{cases}$$

Recurse: call Compute-Opt(m,n) to solve the problem.

Initialize array with size M[m+1][n+1]

@ M[0][:] = 0, M[:][0] = 0 → indicate negative entry eg x[-1], does not exist.

```
Compute-Opt(m,n){
  if M[m+1][n+1] ≠ 0
    return M[m+1][n+1]
  M[m+1][n+1] = max(Compute-Opt(m-1,n-1) + 1/(1+|x_m - y_n|),
                    Compute-Opt(m-1,n),
                    Compute-Opt(m,n-1))
  return M[m+1][n+1]
}
```

Explanation:

x_0 would be stored in $M[1][1]$

y_0 would be stored in $M[1][1]$

x_m would be stored in $M[m+1][n+1]$

y_n would be stored in $M[m+1][n+1]$

$M[0][:], M[:][0]$ is reserved for 0.

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 $1 \leq p \leq m$, $\{x_{i_p}, x_{j_p}\}$ 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:

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

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]

recurse: $OPT(i,j) = \max \begin{cases} OPT(i, j-1) \\ \max_k \{ OPT(i, k-1) + OPT(k+1, j-1) + (k-j)^2 \} \end{cases}$

↳ k constrained by: $i \leq k < j-4$, $\{x_k, x_j\}$ are one of $\{G,C\}$ or $\{A,U\}$

$= 0$ for all $i > j-4$

recursive implementation: Call Compute-Opt(1, n)

initialize $M[n][n]$, set $M[i][j] = 0$ for all $i > j-4$

```
Compute-Opt(i, j) {
  if M[i][j] ≠ φ {
    return M[i][j]
  } else { for k = i ... j-5 {
      if {x_k, x_j} are one of {G,C} or {A,U}
        SUM[k] = max(Compute-Opt(i,k-1) + Compute-Opt(k+1,j-1) + (k-j)^2)
      else
        SUM[k] = -∞
    }
    M[i][j] = max(Compute-Opt(i, j-1), max(SUM[k]))
    return M[i][j]
  }
}
```