CS3343 Analysis of Algorithms Fall 2017

Homework 6

Due 11/3/17 before 11:59pm (Central Time)

1. Franchise Restaurants (5 points)

A fast food company is considering a bunch of bids to open franchises on a road and they want to develop an algorithm to decide which to accept.

For each of the n miles along the road, the bid for that location is recorded (if there is no bid it is recorded as 0). The company wants to accept the bids which maximizes their total earnings. One extra caveat is that two franchises must be at least 3 miles apart (for example if you accept the bid at mile 5 then you can't accept the bids at miles $3, 4, 6, 7$).

Mile:	1	2	3	4	5	6	7	8	9
Bid:	50,000	60,000	30,000	45,000	40,000	10,000	40,000	0	45,000

(1) (1 point) Example: For the bids given above, which would you accept to maximize the earnings? (in this case, $n = 9$)
I would accept the bids at miles 2, 5, and 9 for the maximum total of 145,000.

(2) (2 points) Let $E(n)$ denote your maximum earnings for the bids in the first n miles. Give a recursive definition of E:

(a) Base case
(Hint: this will occur when $n = 0$ since there are no bids to consider).
$E(0) = 0$

(b) Recursive case
(Hint: compare the total value obtained from accepting the n^{th} bid to the total value obtained from not accepting the n^{th} bid).

$$E(n) = \max \begin{cases} E(n-1) \\ \max\limits_{1 \le i \le n-3} (E(i) + b[n]) \end{cases}$$

(3) (2 points) Give pseudo-code for an algorithm which uses memoization to compute $E(n)$ based on the above recurrence (assume you are passed an array b of the bids).

Algorithm 1 int getMaxProfits(int n, int[] b)

```
let p[0..n] be a new array
for i = 0 to n do
    p[i] = −1;
end for
return getMaxProfitsAux(n, b, p);
```

Algorithm 2 int getMaxProfitsAux(int n, int[] b, int[] p)

```
if p[n] ≥ 0 then
    return p[n];
end if
m = −1;
for i = n − 3; i ≥ 1; i − − do
    m = max(getMaxProfitsAux(n − 1), getMaxProfitsAux(i) + b[n]);
end for
p[n] = m;
return m;
```

2. Inventory Management (7 points)

Suppose you are playing a video game. In this game you can only carry a limited amount of items. Every item has a value and your goal is to maximize the total value of items you are carrying.

Specifically, you are choosing which of m items to carry where item number i has value $v[i]$ (for your recurrence/pseudo-code you can assume you are passed the item values in an array, v, of size m).

(1) Suppose you can only carry n items of the m available.
   (a) (1 point) Example: Let $n = 3$, $m = 5$. If the item values are $v = \{5, 30, 17, 32, 40\}$ which 3 should you choose to carry?
   I would choose to carry the items numbered 2, 4, and 5 for a maximum value of 102.
   (b) (1 point) Give a short description of a greedy algorithm which maximizes your total value for any given n,m, and v.
   The algorithm would, at each step i through the m items, choose that item if n items have not been chosen yet or replace the minimum of the chosen n items so far with the ith item if $v[i]$ is greater than that minimum item of the n chosen so far.

(2) Suppose, the game is updated so that every item, i, now has weight, $w[i]$, in kilograms (you can assume you are passed the item weights in an array, w, of size m). You can only carry n kilograms of weight.
   (a) (1 point) Example: Let $n = 15$, $m = 5$. If the item values are $v = \{5, 30, 17, 32, 40\}$ and the item weights are $w = \{2, 4, 3, 6, 15\}$ which should you choose to carry? Which would your greedy algorithm choose to carry?
   I would choose to carry the first 4 items for a maximum value of 84 and weight of 15. My greedy algorithm would select the items with the highest value obeying the constraints of weight, which would select the 5th item only for a value of 40 and weight of 15.
   (b) (2 points) Let $V(n, m)$ denote your maximum total value for any given n, m, v, and w. Give a recursive definition of V:

(i) Base case
(Hint: your base case will occur when $m = 0$ since you have no items left to consider).
$V(n, 0) = 0$ and $V(0, m) = 0$

(ii) Recursive case
(Hint: compare the total value obtained from taking the m^{th} item to the total value obtained from not taking the m^{th} item).

$$V(n,m) = \begin{cases} V(n, m-1) & w[m] > n \\ \max \begin{cases} V(n, m-1) \\ V(n - w[m], m-1) + v[m] \end{cases} & otherwise \end{cases}$$

(c) (2 points) Based on your recurrence, write the pseudo-code for a dynamic programming algorithm to compute the maximum total value (you can use bottom up dynamic programming or memoization).

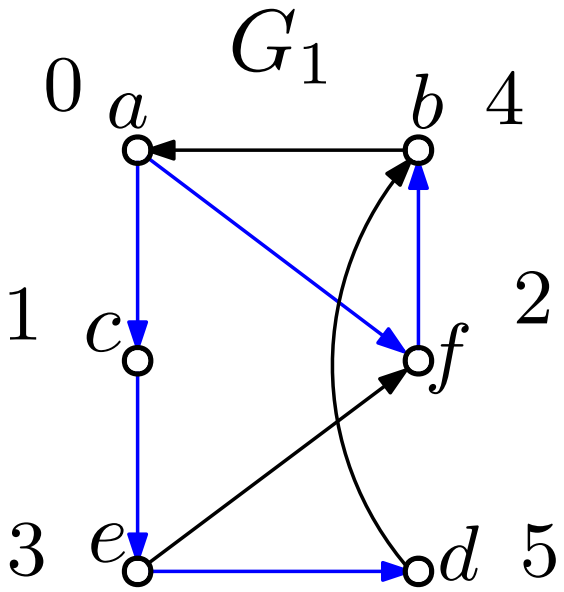
Algorithm 3 int getMaxValue(int n, int m, int $v[]$, int $w[]$)

```
let p[0..m][0..n] be a new array
for i = 0 to m do
    for j = 0 to n do
        if i == 0 or j == 0 then
            p[i][j] = 0;
        else if w[i − 1] > j then
            p[i][j] = p[i − 1][j];
        else
            p[i][j] = max(p[i − 1][j], p[i − 1][j − w[i − 1]] + v[i − 1]);
        end if
    end for
end for
return p[m][n];
```

3. DFS and BFS (6 points)

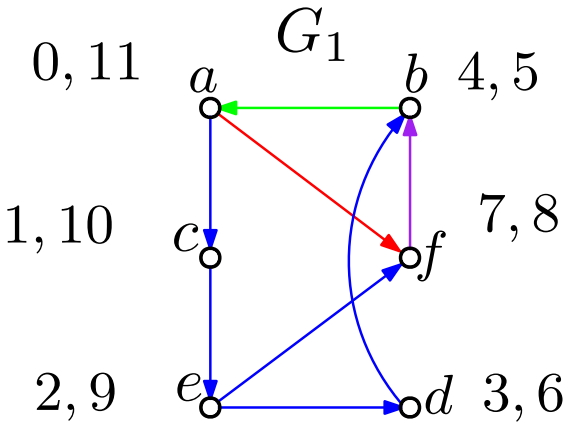
For the following problems, assume that vertices are ordered alphabetically in the adjacency lists (thus you will visit adjacent vertices in alphabetical order).

(1) (2 points) Execute a Breadth-First Search on the graph G_1, starting on vertex a. Specifiy the visit times for each node of the graph.



(2) (2 points) Execute a Depth-First Search on the graph G_1, starting on vertex a. Specifiy the visit and finish times for each node of the graph

(a) (1 point) For each edge in your graph identify whether it is a tree edge, back edge, forward edge, or a cross edge.



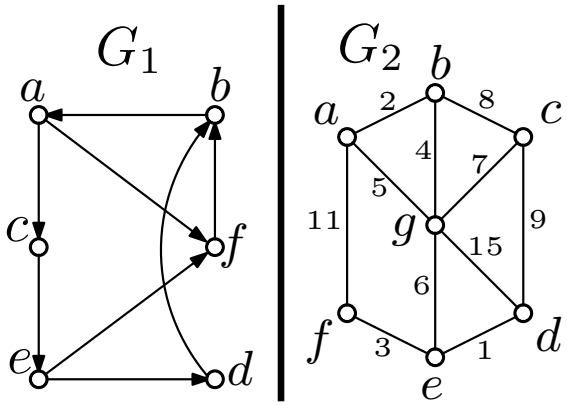
(b) (1 point) Which edges would you remove to make your graph into a DAG? (hint: use your edge classification to justify your choice)

I would remove all back edges, because a cycle is defined when a path exists where the first and last vertices are the same. Thus, there would be an edge from a vertex to its ancestor in the path and, by definition, that is a back edge in the edge classifications of a depth first search. Thus if you remove all back edges, you remove all cycles.

4. Prim's Algorithm (4 points)

(4 points) Run Prims algorithm on the graph G_2, with start vertex a. Assume that vertices are ordered alphabetically.

For each step of the algorithm specify the current vertex weights (you can use a table to represent this data). Draw the minimum spanning tree the algorithm finds.



step	a	b	c	d	e	f	g
1	0	∞	∞	∞	∞	∞	∞
2	0	2	∞	∞	∞	11	5
3	0	2	8	∞	∞	11	4
4	0	2	7	15	6	11	4
5	0	2	7	1	6	3	4
6	0	2	7	1	6	3	4
7	0	2	7	1	6	3	4
8	0	2	7	1	6	3	4