CSE3500-001 Homework #6

Exercise 6

1, 2, 3, 4, 17, 28.

Exercises

1. Let $G = (V, E)$ be an undirected graph with n nodes. Recall that a subset of the nodes is called an *independent set* if no two of them are joined by an edge. Finding large independent sets is difficult in general; but here we'll see that it can be done efficiently if the graph is "simple" enough.

Call a graph $G = (V, E)$ a *path* if its nodes can be written as $v_1, v_2, \ldots, v_n$, with an edge between v_i and v_j if and only if the numbers i and j differ by exactly 1. With each node v_i, we associate a positive integer *weight* w_i.

Consider, for example, the five-node path drawn in Figure 6.28. The *weights* are the numbers drawn inside the nodes.

The goal in this question is to solve the following problem:

Find an independent set in a path G whose total weight is as large as possible.

(a) Give an example to show that the following algorithm *does not* always find an independent set of maximum total weight.

```
The "heaviest-first" greedy algorithm
  Start with S equal to the empty set
  While some node remains in G
    Pick a node v_i of maximum weight
    Add v_i to S
    Delete v_i and its neighbors from G
  Endwhile
  Return S
```

(b) Give an example to show that the following algorithm also *does not* always find an independent set of maximum total weight.

```
Let S_1 be the set of all v_i where i is an odd number
Let S_2 be the set of all v_i where i is an even number
(Note that S_1 and S_2 are both independent sets)
Determine which of S_1 or S_2 has greater total weight,
  and return this one
```

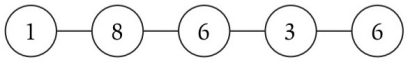

Figure 6.28 A paths with weights on the nodes. The maximum weight of an independent set is 14.

(c) Give an algorithm that takes an n-node path G with weights and returns an independent set of maximum total weight. The running time should be polynomial in n, independent of the values of the weights.

2. Suppose you're managing a consulting team of expert computer hackers, and each week you have to choose a job for them to undertake. Now, as you can well imagine, the set of possible jobs is divided into those that are *low-stress* (e.g., setting up a Web site for a class at the local elementary school) and those that are *high-stress* (e.g., protecting the nation's most valuable secrets, or helping a desperate group of Cornell students finish a project that has something to do with compilers). The basic question, each week, is whether to take on a low-stress job or a high-stress job.

If you select a low-stress job for your team in week i, then you get a revenue of $\ell_i > 0$ dollars; if you select a high-stress job, you get a revenue of $h_i > 0$ dollars. The catch, however, is that in order for the team to take on a high-stress job in week i, it's required that they do no job (of either type) in week $i-1$; they need a full week of prep time to get ready for the crushing stress level. On the other hand, it's okay for them to take a low-stress job in week i even if they have done a job (of either type) in week $i-1$.

So, given a sequence of n weeks, a *plan* is specified by a choice of "low-stress," "high-stress," or "none" for each of the n weeks, with the property that if "high-stress" is chosen for week $i > 1$, then "none" has to be chosen for week $i-1$. (It's okay to choose a high-stress job in week 1.) The *value* of the plan is determined in the natural way: for each i, you add ℓ_i to the value if you choose "low-stress" in week i, and you add h_i to the value if you choose "high-stress" in week i. (You add 0 if you choose "none" in week i.)

The problem. Given sets of values $\ell_1, \ell_2, \ldots, \ell_n$ and $h_1, h_2, \ldots, h_n$, find a plan of maximum value. (Such a plan will be called *optimal*.)

Example. Suppose $n = 4$, and the values of ℓ_i and h_i are given by the following table. Then the plan of maximum value would be to choose "none" in week 1, a high-stress job in week 2, and low-stress jobs in weeks 3 and 4. The value of this plan would be $0 + 50 + 10 + 10 = 70$.

	Week 1	**Week 2**	**Week 3**	**Week 4**
ℓ	10	1	10	10
h	5	50	5	1

(a) Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

```
For iterations i = 1 to n
  If h_{i+1} > ℓ_i + ℓ_{i+1} then
    Output "Choose no job in week i"
    Output "Choose a high-stress job in week i+1"
    Continue with iteration i+2
  Else
    Output "Choose a low-stress job in week i"
    Continue with iteration i+1
  Endif
End
```

To avoid problems with overflowing array bounds, we define $h_i = \ell_i = 0$ when $i > n$.

In your example, say what the correct answer is and also what the above algorithm finds.

(b) Give an efficient algorithm that takes values for $\ell_1, \ell_2, \ldots, \ell_n$ and $h_1, h_2, \ldots, h_n$ and returns the *value* of an optimal plan.

3. Let $G = (V, E)$ be a directed graph with nodes $v_1, \ldots, v_n$. We say that G is an *ordered graph* if it has the following properties.

(i) Each edge goes from a node with a lower index to a node with a higher index. That is, every directed edge has the form (v_i, v_j) with $i < j$.

(ii) Each node except v_n has at least one edge leaving it. That is, for every node v_i, $i = 1, 2, \ldots, n-1$, there is at least one edge of the form (v_i, v_j).

The length of a path is the number of edges in it. The goal in this question is to solve the following problem (see Figure 6.29 for an example).

Given an ordered graph G, find the length of the longest path that begins at v_1 and ends at v_n.

(a) Show that the following algorithm does not correctly solve this problem, by giving an example of an ordered graph on which it does not return the correct answer.

```
Set w = v_1
Set L = 0
```

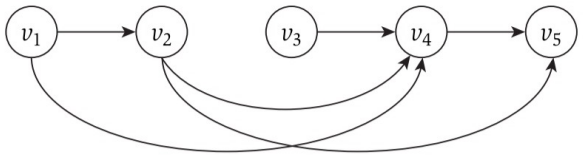

Figure 6.29 The correct answer for this ordered graph is 3: The longest path from v_1 to v_n uses the three edges (v_1, v_2),(v_2, v_4), and (v_4, v_5).

```
While there is an edge out of the node w
  Choose the edge (w, v_j)
     for which j is as small as possible
  Set w = v_j
  Increase L by 1
end while
Return L as the length of the longest path
```

In your example, say what the correct answer is and also what the algorithm above finds.

(b) Give an efficient algorithm that takes an ordered graph G and returns the *length* of the longest path that begins at v_1 and ends at v_n. (Again, the *length* of a path is the number of edges in the path.)

4. Suppose you're running a lightweight consulting business—just you, two associates, and some rented equipment. Your clients are distributed between the East Coast and the West Coast, and this leads to the following question.

Each month, you can either run your business from an office in New York (NY) or from an office in San Francisco (SF). In month i, you'll incur an *operating cost* of N_i if you run the business out of NY; you'll incur an operating cost of S_i if you run the business out of SF. (It depends on the distribution of client demands for that month.)

However, if you run the business out of one city in month i, and then out of the other city in month $i + 1$, then you incur a fixed *moving cost* of M to switch base offices.

Given a sequence of n months, a *plan* is a sequence of n locations—each one equal to either NY or SF—such that the i^{th} location indicates the city in which you will be based in the i^{th} month. The *cost* of a plan is the sum of the operating costs for each of the n months, plus a moving cost of M for each time you switch cities. The plan can begin in either city.

The problem. Given a value for the moving cost M, and sequences of operating costs $N_1, \ldots, N_n$ and $S_1, \ldots, S_n$, find a plan of minimum cost. (Such a plan will be called *optimal*.)

Example. Suppose $n = 4$, $M = 10$, and the operating costs are given by the following table.

	Month 1	**Month 2**	**Month 3**	**Month 4**
NY	1	3	20	30
SF	50	20	2	4

Then the plan of minimum cost would be the sequence of locations

$$[NY, NY, SF, SF],$$

with a total cost of $1 + 3 + 2 + 4 + 10 = 20$, where the final term of 10 arises because you change locations once.

(a) Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

```
For i = 1 to n
  If N_i < S_i then
    Output "NY in Month i"
  Else
    Output "SF in Month i"
End
```

In your example, say what the correct answer is and also what the algorithm above finds.

(b) Give an example of an instance in which every optimal plan must move (i.e., change locations) at least three times.

Provide a brief explanation, saying why your example has this property.

(c) Give an efficient algorithm that takes values for n, M, and sequences of operating costs $N_1, \ldots, N_n$ and $S_1, \ldots, S_n$, and returns the *cost* of an optimal plan.

17. Your friends have been studying the closing prices of tech stocks, looking for interesting patterns. They've defined something called a *rising trend*, as follows.

They have the closing price for a given stock recorded for n days in succession; let these prices be denoted $P[1], P[2], \ldots, P[n]$. A *rising trend* in these prices is a subsequence of the prices $P[i_1], P[i_2], \ldots, P[i_k]$, for days $i_1 < i_2 < \ldots < i_k$, so that

- $i_1 = 1$, and
- $P[i_j] < P[i_{j+1}]$ for each $j = 1, 2, \ldots, k-1$.

Thus a rising trend is a subsequence of the days—beginning on the first day and not necessarily contiguous—so that the price strictly increases over the days in this subsequence.

They are interested in finding the longest rising trend in a given sequence of prices.

Example. Suppose $n = 7$, and the sequence of prices is

$$10, 1, 2, 11, 3, 4, 12.$$

Then the longest rising trend is given by the prices on days 1, 4, and 7. Note that days 2, 3, 5, and 6 consist of increasing prices; but because this subsequence does not begin on day 1, it does not fit the definition of a rising trend.

(a) Show that the following algorithm does not correctly return the *length* of the longest rising trend, by giving an instance on which it fails to return the correct answer.

```
Define i = 1
       L = 1
For j = 2 to n
  If P[j] > P[i] then
    Set i = j.
    Add 1 to L
  Endif
Endfor
```

In your example, give the actual length of the longest rising trend, and say what the algorithm above returns.

(b) Give an efficient algorithm that takes a sequence of prices $P[1], P[2], \ldots, P[n]$ and returns the *length* of the longest rising trend.

28. Recall the scheduling problem from Section 4.2 in which we sought to minimize the maximum lateness. There are n jobs, each with a deadline d_i and a required processing time t_i, and all jobs are available to be scheduled starting at time s. For a job i to be done, it needs to be assigned a period from $s_i \geq s$ to $f_i = s_i + t_i$, and different jobs should be assigned nonoverlapping intervals. As usual, such an assignment of times will be called a *schedule*.

In this problem, we consider the same setup, but want to optimize a different objective. In particular, we consider the case in which each job must either be done by its deadline or not at all. We'll say that a subset J of the jobs is *schedulable* if there is a schedule for the jobs in J so that each of them finishes by its deadline. Your problem is to select a schedulable subset of maximum possible size and give a schedule for this subset that allows each job to finish by its deadline.

(a) Prove that there is an optimal solution J (i.e., a schedulable set of maximum size) in which the jobs in J are scheduled in increasing order of their deadlines.

(b) Assume that all deadlines d_i and required times t_i are integers. Give an algorithm to find an optimal solution. Your algorithm should run in time polynomial in the number of jobs n, and the maximum deadline $D = \max_i d_i$.