Chapter 5

1. Come up with an example and ask one of your colleagues to mark it. Then mark the example of your colleague.

3.

	1	2	3	4	5
1	0	200	1200	320	1320
2		0	400	240	640
3			0	200	700
4				0	2000
5					0

	1	2	3	4	5
1	0	1	1	1	4
2		0	2	2	4
3			0	3	4
4				0	4
5					0

So the optimal ordering is $(A_1(A_2(A_3A_4)))A_5$

	1	2	3	4
1	0	5785	1530	2856
2		0	1335	1845
3			0	9078
4				0

	1	2	3	4
1	0	1	1	3
2		0	2	3
3			0	3
4				0

So the optimal ordering is $(A_1(A_2A_3))A_4$

5. Come up with an example and ask one of your colleagues to mark it. Then mark the example of your colleague.

7. Assuming the sequences are $a_1, \ldots, a_m$ and $b_1, \ldots, b_n$ and that the table $P[0..m][0..n]$ is such that $P[i][j]$ is the size of a longest common subsequence of $a_1, \ldots, a_i$ and $b_1, \ldots, b_j$ for every i and j, the following pseudocode assembles a longest common sequence s of $a_1, \ldots, a_m$ and $b_1, \ldots, b_n$:

```
k ← P[m][n] {the size of the longest common subsequence}
Initialize a sequence s_1, ..., s_k of size k.
i ← m
j ← n
while k ≠ 0 do
  if a_i = b_j then
    s_k ← a_i
    k ← k − 1
    i ← i − 1
    j ← j − 1
  else if P[i − 1][j] = k then
    i ← i − 1
```

else
$j \leftarrow j - 1$
end if
end while

8.

$k=0$	1	2	3	4	5	6	7
1	0	4	∞	∞	∞	10	∞
2	3	0	∞	18	∞	∞	∞
3	∞	6	0	∞	∞	∞	∞
4	∞	5	15	0	2	19	5
5	∞	∞	12	1	0	∞	∞
6	∞	∞	∞	∞	∞	0	10
7	∞	∞	∞	8	∞	∞	0

$k=1$	1	2	3	4	5	6	7
1	0	4	∞	∞	∞	10	∞
2	3	0	∞	18	∞	13	∞
3	∞	6	0	∞	∞	∞	∞
4	∞	5	15	0	2	19	5
5	∞	∞	12	1	0	∞	∞
6	∞	∞	∞	∞	∞	0	10
7	∞	∞	∞	8	∞	∞	0

$k=2$	1	2	3	4	5	6	7
1	0	4	∞	22	∞	10	∞
2	3	0	∞	18	∞	13	∞
3	9	6	0	24	∞	19	∞
4	8	5	15	0	2	18	5
5	∞	∞	12	1	0	∞	∞
6	∞	∞	∞	∞	∞	0	10
7	∞	∞	∞	8	∞	∞	0

$k=3$	1	2	3	4	5	6	7
1	0	4	∞	22	∞	10	∞
2	3	0	∞	18	∞	13	∞
3	9	6	0	24	∞	19	∞
4	8	5	15	0	2	18	5
5	21	18	12	1	0	31	∞
6	∞	∞	∞	∞	∞	0	10
7	∞	∞	∞	8	∞	∞	0

$k = 4$	1	2	3	4	5	6	7
1	0	4	37	22	24	10	27
2	3	0	33	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	15	0	2	18	5
5	9	6	12	1	0	19	6
6	∞	∞	∞	∞	∞	0	10
7	16	13	23	8	10	26	0

$k = 5$	1	2	3	4	5	6	7
1	0	4	36	22	24	10	27
2	3	0	32	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	14	0	2	18	5
5	9	6	12	1	0	19	6
6	∞	∞	∞	∞	∞	0	10
7	16	13	22	8	10	26	0

$k = 6$	1	2	3	4	5	6	7
1	0	4	26	22	24	10	20
2	3	0	32	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	14	0	2	18	5
5	9	6	12	1	0	19	6
6	∞	∞	∞	∞	∞	0	10
7	16	13	22	8	10	26	0

$k = 7$	1	2	3	4	5	6	7
1	0	4	26	22	24	10	20
2	3	0	32	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	14	0	2	18	5
5	9	6	12	1	0	19	6
6	26	23	32	18	20	0	10
7	16	13	22	8	10	26	0

9. For each vertex v, the adjacency list of v contains the outgoing edges from v. For each vertex v, add an empty list *ingoing* that will contain the ingoing edges coming to v. Before the very first line of the algorithm that was presented in class, add a step where you scan all adjacency lists and update the different ingoing lists. Scanning all adjacency lists takes $O(|V| + |E|)$ as we discussed multiple times in class.

11. (a) i. Let $b_1, \ldots, b_k$ be any increasing subsequence of $a_1, \ldots, a_n$. Then $-\infty, b_1, \ldots, b_k, \infty$ is an increasing subsequence of size $k+2$ of $-\infty, a_1, \ldots, a_n, \infty$. Therefore

$$\mathrm{LIS}(a_1, \ldots, a_n) \leq \mathrm{LIS}(-\infty, a_1, \ldots, a_n, \infty) - 2.$$

Conversely, any longest increasing subsequence of $-\infty, a_1, \ldots, a_n, \infty$ must contain both the infinite entries, otherwise we could append $-\infty$ to the left of the sequence or ∞ to the right of it to obtain an even longer sequence. Thus removing these two infinite entries yields an increasing subsequence of $a_1, \ldots, a_n$ that is two entries shorter, which shows that

$$\mathrm{LIS}(a_1, \ldots, a_n) \geq \mathrm{LIS}(-\infty, a_1, \ldots, a_n, \infty) - 2.$$

ii. The key idea is that there is a correspondence between longest increasing subsequences of $-\infty, a_1, \ldots, a_n, \infty$ and longest paths from v_0 to v_{n+1} in the graph. Indeed, the vertices corresponding to the entries of any increasing subsequence have edges between them and form a path, and we also know any longest increasing subsequence of $-\infty, a_1, \ldots, a_n, \infty$ must use the first and last entries. Furthermore, any path from v_0 to v_{n+1} in the graph is increasing in both the vertex indices and the vertex keys, corresponding therefore to an increasing subsequence of $-\infty, a_1, \ldots, a_n, \infty$ of same size.
We can modify the algorithm from section 5.1 to compute longest paths: simply look for the incoming edge of largest path length instead of smallest path length. The reason this algorithm works is that the dynamic programming actually considers all possible paths. We also need to use the fact that the graph acyclic. Do you see why?

iii. In the worst case, the graph has $\binom{n}{2} = O(n^2)$ edges. Do you see why? Therefore, building the graph can be done in $O(n^2)$ time. The running time of the longest path finding algorithm, as in the course notes, is then $O(|V| + |E|) = O(n + n^2) = O(n^2)$.

(b) We will scan the list $(a_1, a_2, ..., a_n)$ from a_1 to a_n. As we scan the list, we will maintain a set S of candidate lists. We call the lists in S the *active lists*. When we look at a new element a_i, we will look at the lists in S to see if any of them can be improved. This can be done by focusing only on the last element of each list. Here is how to do it. We consider three cases.

(1) If a_i is smallest among all end candidates of active lists, we will start new active list of length 1.

(2) If a_i is largest among all end candidates of active lists, we will clone the largest active list, and extend it by a_i.

(3) If a_i is in between, we will find a list with largest end element that is smaller than a_i. Clone and extend this list by a_i. We will discard all other lists of same length as that of this modified list.

Here is an example. Take $(a_1, a_2, ..., a_n) = (3, 6, 10, 5, 1, 8)$. We start with $S = \{\,\}$.

- We start with $a_1 = 3$. Since there are no lists, we are in case (1). Hence, S becomes

$$(3)$$

- Then we have $a_2 = 6$. We are in case (2). Hence, S becomes

$$\begin{aligned}&(3)\\&(3,6)\end{aligned}$$

- Then we have $a_3 = 10$. We are in case (2). Hence, S becomes

$$\begin{aligned}&(3)\\&(3,6)\\&(3,6,10)\end{aligned}$$

- Then we have $a_4 = 5$. We are in case (3). Hence, S becomes

$$\begin{aligned}&(3)\\&(3,5)\\&(3,6,10)\end{aligned}$$

- Then we have $a_5 = 1$. We are in case (1). Hence, S becomes

$$\begin{aligned}&(1)\\&(3)\\&(3,5)\\&(3,6,10)\end{aligned}$$

- Then we have $a_6 = 8$. We are in case (3). Hence, S becomes

$$\begin{aligned}&(1)\\&(3)\\&(3,5)\\&(3,5,8)\end{aligned}$$

Therefore, $(3, 5, 8)$ is a longest increasing subsequence.

Each time we consider a new element a_i, we need to deal with one of the three cases. This is done only by considering the last element in each active list. This can be done by doing binary search. Therefore, the total running time is $O(n \log(n))$.

13. (a) Let the denomination values be $d_1, \ldots, d_n$ and $C[j]$ denote the minimum number of coins you need to get a total amount of j. Then

$$C[j] = \begin{cases} \infty & \text{if } j < 0, \\ 0 & \text{if } j = 0, \\ 1 + \min_{1 \le k \le n} C[j - d_k] & \text{if } j \ge 1. \end{cases}$$

Filling the table takes $O(n)$ time for each j. Therefore, the total running time is $O(nv)$.

(b) Then the problem can be reduced to the knapsack problem studied in Assignment 3. Do you see how?

15. fib2.

17. By induction.

Base case: If $n = 1$, we do not need any parentheses and $1 - 1 = 0$.

Suppose we need $n - 1$ pairs of parentheses to fully parenthesize an expression having n matrices.

Consider the product $A_1 A_2 \ldots A_{n+1}$. Let k be the index where the product is cut. We get

$$\Big((A_1 A_2 \ldots A_k)(A_{k+1} A_{k+2} \ldots A_{n+1})\Big).$$

By the induction hypothesis, we need $k - 1$ pairs of parentheses to fully parenthesize the first group of matrices and we need $(n - (k + 1) + 1) - 1 = n - k - 1$ pairs of parentheses to fully parenthesize the first group of matrices. With the extra pair we need around the two groups, we get

$$1 + (k - 1) + (n - k - 1) = n - 1.$$