CSCC73
Algorithm design & analysis

Week 8 Tutorial
Question 1

Another DP algorithm for 0/1 knapsack.

Recall the problem:

**Input:** For each $i$, $1 \leq i \leq n$, the value $v_i$ and the weight $w_i$ of item $i$; and the capacity $C$ of the knapsack.

**Output:** The maximum value of a subset of the items whose total weight does not exceed $C$. (Also, the actual subset of items, but we won’t worry about that here.)
In the DP algorithm in class we defined the following subproblems:

For \( i = 0, 1, \ldots, n \), and \( c = 0, 1, \ldots, C \),

\[ K(i, c) = \text{the maximum value of a subset of items} \{1, 2, \ldots, i\} \text{ whose weight is } \leq c. \]

We found a recursive formula to compute these subproblems, and used that to design a DP algorithm for the 0/1 knapsack problem.

Running time: \( O(nC) \) — pseudopolynomial.
Now we will define the subproblem differently.
For \( i = 0, 1, \ldots, n \), let
\[
V_i = \sum_{t=1}^{i} v_t. \quad (V_0 = 0.)
\]

For \( i = 0, 1, \ldots, n \), and \( v = 0, 1, \ldots, V_i \),
\[
W(i, v) = \text{the minimum weight of a subset of items } \{1, 2, \ldots, i\} \text{ whose value is } \geq v.
\]

Compare to the subproblems we defined before:

\[
K(i, c) = \text{the maximum value of a subset of items } \{1, 2, \ldots, i\} \text{ whose weight is } \leq c.
\]
Question 1 (cont’d)

For $i = 0, 1, \ldots, n$, and $v = 0, 1, \ldots, V_i$,
$W(i, v)$ = the minimum weight of a subset of items
$\{1, 2, \ldots, i\}$ whose value is $\geq v$.

• Give a recursive formula to compute the subproblems.

• Describe your DP algorithm in pseudocode.

• Analyze the running time of your algorithm.

• Modify the algorithm to find the actual set of items
of maximum value whose weight does not exceed
the knapsack capacity $C$. 
Question 2

Give a pseudo-polynomial time DP algorithm to solve the “knapsack with replacement” problem:

For each item \( i = 1, 2, \ldots, n \), let \( v_i \) be the value and \( w_i \) be the weight of \( i \), where each weight is a positive integer. Let \( C \) be a positive integer (the knapsack capacity).

Assume that there is an unlimited number of each item. We want to find how many copies \( S(i) \) of each item \( i \) to steal, where \( S(i) \) is a non-negative integer, so that the stolen items have maximum value and fit in the knapsack: find \( S \) that maximizes \( \sum_{i=1}^{n} S(i) \cdot v_i \), so that \( \sum_{i=1}^{n} S(i) \cdot w_i \leq C \).
Question 2

Focus on the problem of finding the maximum value of a multiset of items from 1, 2, ..., n whose weight does not exceed $C$.

• Define the subproblems to solve, and state how solving these subproblems helps solving the above problem.

• Give a recursive formula to compute the subproblems.

• Write pseudocode to solve the problem of finding the number $S(i)$ of each item $i$ in an optimal knapsack with replacement.
Let $A$ be a sequence. A subsequence of $A$ is a sequence $A'$ obtained by removing zero or more elements from $A$, leaving the remaining elements in their original order.

A sequence is **palindromic**, if it is equal to its reverse.

Describe a polynomial time DP algorithm that, given a sequence $A[1..n]$, finds the length of the **longest palindromic subsequence of** $A$. Analyze the running time of your algorithm.
Question 3 (cont’d)

• Define the subproblems of your DP algorithm.

• Give a recursive formula to compute the subproblems.

• Describe your DP algorithm in pseudocode.

• Analyze the running time of your algorithm.

• Retrofit your algorithm to compute the actual longest palindromic subsequence of the given sequence \( A \).
Question 4

Run the Bellman-Ford algorithm on the graph below where the start node is 1. Show the L-value of every node in every iteration. (Shown are the L-values after initialization.) Don’t trace BF line-by-line; use your knowledge of what the L-values represent!

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 5
Run the Floyd-Warshall algorithm on the graph below. Shown next to the graph is the matrix C(-,-,0). Show the matrix C(-,-,k) for every other relevant value of k. Don’t trace FW line-by-line; use your knowledge of what C(-,-,-) represents!

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0 & 3 & 8 & 5 \\
2 & \infty & 0 & \infty & 1 \\
3 & \infty & 1 & 0 & \infty \\
4 & \infty & \infty & 2 & 0 \\
\end{array}
\quad
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & \quad & \quad & \quad \\
2 & \quad & \quad & \quad \\
3 & \quad & \quad & \quad \\
4 & \quad & \quad & \quad \\
\end{array}
\quad
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & \quad & \quad & \quad \\
2 & \quad & \quad & \quad \\
3 & \quad & \quad & \quad \\
4 & \quad & \quad & \quad \\
\end{array}
\]
Question 6

To think about on your own:

1) Does the Bellman-Ford algorithm find a shortest path from node $s$ to every node with the minimum number of edges? If not, can you change it so that it does?

2) Does the Floyd-Warshall algorithm find a shortest path between every pair of nodes with the minimum number of edges? If not, can you change it so that it does?