1. **Graph Shortest Paths:** Consider the following directed, weighted graph:

![Graph Diagram]

(a) Even though the graph has negative weight edges, step through Dijkstra’s algorithm to calculate *supposedly* shortest paths from A to every other vertex. Show your steps in the table below. At the end of each step show the distances as they are after relaxation (in the beginning only d[A] is 0, the rest are ∞, shown for your convenience). Also list for each vertex its Edge To (which is the edge that marks the final step in the shortest path, as shown in class). Under “Vertex” show the vertex being added to the tree (taken out of the priority queue) at that stage.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td></td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>∞</td>
<td>12</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>12</td>
<td>6</td>
<td>∞</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>12</td>
<td>6</td>
<td>∞</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>12</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>G</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>E</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>2*</td>
</tr>
</tbody>
</table>

| Final Dist: | 0 | 2 | 7 | 4 | 9 | 6 | 2* |
| EdgeTo:     | – | A-B | A-C | B-D | C-E | D-F | E-G* |

* This is what Dijkstra’s algorithm would do, although it’s technically not OK, since we modify the path length to a vertex that’s already been processed.

(b) Due to the negative edges, Dijkstra’s algorithm found the wrong path to some of the vertices. For just the vertices where the wrong path was computed, write both the path that was computed by Dijkstra’s and the correct path and their weights. There are three such paths.

<table>
<thead>
<tr>
<th>path 1</th>
<th>path 2</th>
<th>path 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>computed (weight)</td>
<td>A-B-D-F-G</td>
<td>A-B-D</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>correct (weight)</td>
<td>A-C-E-G</td>
<td>A-C-E-G-D</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

(Many people found two paths because one of them, path 1, is calculated by Dijkstra’s algorithm. The other ones aren’t. When I gave this question in an exam I accepted any two out of the three though).

(c) What single edge could be removed from the graph such that Dijkstra’s algorithm would show the correct path for all vertices? Explain.
If you look at the table above, the main “culprit” is the $E \rightarrow G$ edge, with a weight of -7.

2. **Greedy Algorithms:** The maximum independent set in a graph or a tree is a maximal set of mutually disconnected nodes in a graph. To solve the problem we should find a maximum set of nodes such that no two nodes in the set have an edge between them. When the graph is a tree, the maximum size independent set can be found efficiently using a greedy algorithm.

(a) Show that every leaf in a tree is included in at least one maximum independent set. **Hint:** Given a leaf, show that every maximum independent set can either have this leaf or be made to have this leaf with one simple exchange.

**Answer:** Let us look at a maximum independent set $S$ and a leaf $l$. If $l \in S$ we are done. Otherwise it means that $l$’s parent $p$ is in $S$ (if it weren’t we could add $l$ to $S$ and get a bigger set, contradicting the assumption that $S$ is maximum). We can take out $p$ and add $l$, getting a set of the same size (thus maximum) and containing $l$. Notice that it doesn’t mean every max independent set has $l$, just that at least one such set exists.

(b) Given (a) above, we can define a greedy algorithm as follows:

- Initialize an empty set $S$.
- Pick a random leaf and add it to $S$.
- Remove its parent from the tree because the parent cannot be part of the set (leaving possible descendants of that parent in the tree. Notice that the tree may now be disconnected but that’s ok and that now some nodes may become leaves).
- Repeat until the tree is empty.
- Return $S$

What is the runtime of this algorithm as a function of $n$, the number of nodes in the tree?

**Answer:** The runtime is linear in the number of nodes.
3. **Dynamic Programming:** The *Rod cutting* problem is defined as follows: Suppose you have a rod of length \( n \), and you want to cut up the rod and sell the pieces in a way that maximizes the total amount of money you get. A piece of length \( i \) is worth \( p_i \) dollars. For example, given a rod of length 4, these are the ways to cut it:

\[
(4,0) \quad (3,1) \quad (2,2) \quad (1,3) \\
(1,1,2) \quad (1,2,1) \quad (2,1,1) \quad (1,1,1,1)
\]

(a) A recursive algorithm to solve the problem is to examine all the possible cuts:

- First, cut a piece of size \( i \) off the left end of the rod, and sell it.
- Then, recursively find the optimal way to cut the remainder of the rod (of size \( n - i \)).

We don’t know how large a piece we should cut off. So we try all possible cases. First we try cutting a piece of length 1, and combining it with the optimal way to cut a rod of length \( n - 1 \), then cut a piece of length 2 and combining it with the optimal way to cut a rod of length \( n - 2 \), etc. Notice that we can also ”cut” a piece of size \( n \) which is the same as not cutting at all and just use the entire rod. **Question:** Formulate the recursive algorithm as an equation by filling in the missing parts. We define \( r_n \) as the optimal solution for a rod of size \( n \):

\[
 r_n = \max_{1 \leq i \leq n} \{ r_{n-i} + c_i \}
\]

It’s just a summary of the above description. Boundary condition of \( r_0 \) (where \( n = 0 \)): \( c_n \) (when you don’t cut anything, you just sell the whole piece for its full price)

(b) Draw the recursion tree resulting in calculating the solution for a rod of size 4. The first level is given. Each node is the step where you cut a piece of \( i \), sell it and calculate recursively the optimal solution for the remaining \( n - i \). Draw the rest (as a function of \( i \) at every level based on the formula you calculated above).

(c) The problem can be solved using dynamic programming by first solving small subproblems and memoizing them, using the formula as above. Given the following prices, solve the problem for a rod of size 4 by filling the table below:

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>
(d) What is the runtime of the DP algorithm as a function of \( n \), the rod size? Explain briefly.

**Answer:** It is a quadratic algorithm, because for every \( i \) you need to test \( i-1 \) previous answers.

4. Character coding:

   a. Which of the following sets of codes are prefix-free codes for a file containing only the characters, \( a \), \( b \), \( c \) and \( d \)?
      a. \( a = 1, b = 0, c = 01, d = 10 \)
      b. \( a = 10, b = 01, c = 00, d = 11 \)
      c. \( a = 1, b = 01, c = 101, d = 0001 \)
      d. All of the above are valid
      e. None of the above are valid

   **Answer:** (b) is a prefix-free set. The rest are not.

   b. Draw the trie that results from the following character distribution: No need to show the stages of the algorithm. You can assign left and right as you wish, as long as the trie is algorithmically correct.

   \[
   \begin{array}{c|c|c|c|c|c}
   \text{char} & 'z' & 'x' & 'a' & 'b' & 'c' \\
   \text{freq} & 10 & 7 & 14 & 8 & 4 \\
   \end{array}
   \]

   c. Fill the code for each character in the following table:

   \[
   \begin{array}{c|c}
   \text{char} & \text{binary code} \\
   'z' & 10 \\
   'x' & 010 \\
   'a' & 00 \\
   'b' & 11 \\
   'c' & 011 \\
   \end{array}
   \]

5. Network Flow:

   a. What is the maximum flow/minimum capacity and of an s-t cut in the flow network in the figure? The capacity of each edge appears as a label next to the edge. Explain your answer by running the max-flow algorithm.

   b. Show the min-cut corresponding to your answer.

   **Answer:** The flow is 4, the cut is \( \{S,V\} \) vs. \( \{U,T\} \). Augmenting paths are \( S \to u \to t \) (flow 2), and \( S \to v \to t \) (flow 2).