1. **Graph Shortest Paths:** Given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex $u$ to vertex $v$. We interpret $r(u, v)$ as the probability that the channel from $u$ to $v$ will not fail, and we assume that these probabilities are independent. This assumption allows us to estimate the total probability of an entire path as the product of its edge probabilities. So for example if we have a path as follows:

```
0.5
s → a → b → t
0.8
```

Then its total probability is $0.5 \times 0.8 \times 0.8 = 0.32$

Give an efficient algorithm to find the most reliable path between two given vertices $s$ and $t$. In other words, the path whose probability product is maximum. (**Hint:** You should eventually use Dijkstra’s algorithm, but not before you convert the graph from a ”maximum product” to a ”minimum sum”. We already learned how to do it...).

**Answer:** We are looking for the graph with the total biggest product of path weights. We can convert products to sums by using log, so we can construct a new graph $G'$ with the same set of vertices, such that the vertices and edge locations are the same, and every edge weight $w' = -\log(w)$, where $w$ is the original weight. We need the negative of the log, since: 1) all weights are $< 1$, so the log is negative. Using -log guarantees all positive weighs 2) Given two paths in the original graphs, say $p_1, p_2$, whose weights are $w_1, w_2$ respectively and are in the range of $(0, 1]$ if $w_1 > w_2$, then $-\log(w_1) < -\log(w_2)$. Therefore, we can find the shortest path in $G'$ between $s$ and $t$ using Dijkstra’s algorithm.

2. **Greedy algorithms**

(a) Show that the greedy cashier (change making) algorithm gives the optimal solution for the following set of coins: $\{2.5, 5, 10, 25, 45\}$. **Hint:** Cap every combination of smaller coins by a combination of larger coins that has the same amount but at most the same number of coins, and show that this is what the greedy algorithm would select.

**Answer:** The greedy algorithm would select the largest number of big coins it can get. We can show that an optimal algorithm has at most one 2.5 coin (because we can always replace two 2.5 with one 5, getting a better solution), at most one 5 coin (we can replace every two 5’s by a 10, getting a better solution), at most three 10 and 5 coins, out of which at most one is 5 (just like standard US coins). As for 25’s – any 2 25’s can be replaced by 45+5, getting the same number of coins. Therefore, the greedy algorithm performs at least as well as any optimal solution for any combination of coins.

(b) Show that when you add a 35 coins, so that now you have the following coins: $\{2.5, 5, 10, 25, 35, 45\}$, the greedy method does not give you the minimum number of coins for every combination. **Hint:** Don’t try integer multiples of that 35, but a combination of it + a smaller one.

**Answer:** A combination of 35 + 25 gives 60. The greedy method would give 45 + 10 + 5, one coin more.
3. **Dynamic Programming:** In class, we discussed the Independent Set problem being NP complete. As a reminder, an maximum independent set is a maximal set of mutually disconnected nodes in a graph. In other words, 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 binary tree, the maximum size independent set can be found efficiently using a dynamic programming algorithm.

(a) First, notice that every node \( v \) in the tree can either be picked or not picked for the independent set:
If \( v \) is picked, its children obviously cannot be picked but its grandchildren can be considered. If \( v \) is not picked, its children can be considered. Whether we should pick \( v \) or not, depends on which one of the choices (pick or not pick) will result in the largest independent set. Define the recursive formulation to the problem. Don’t forget the boundary conditions. Don’t assume the binary tree is complete.

**Answer:** For every node, either you select it or not. If you select a node you can’t select its children so you have to recursively select the maximum independent sets of the subtrees rooted at its grandchildren and add 1 (the node). Otherwise, recursively calculate the maximum independent sets of the trees rooted at the children. The boundary condition is – if the tree is empty, return 0. If it’s a leaf, return 1 (that’s the size of the set we’re interested in). Otherwise:

\[
I_{Set}(n) = \max(1 + \sum_{i \in \text{grandchildren}(n)} I_{Set}(i), \sum_{i \in \text{children}(n)} I_{Set}(i))
\]

(b) The algorithm you described above have many overlapping subproblems. Show a dynamic programming formulation for the problem. What is the runtime?

**Answer:** To make the solution DP, we use the same formula but go bottom up instead of top down. We first calculate the maximum independent sets for all the leaves and then climb our way up.

(c) Show the maximum independent set for the tree in the figure below. Mark by * the nodes that are picked for the maximum independent set.

See following tree. Every node shows the size of the independent set for the tree rooted in it. Nodes marked by * are picked for the max. independent set. The size of the entire set is 5, as marked at the root.
4. Character coding:

a. Which of the following sets of codes are prefix 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 is 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.

<table>
<thead>
<tr>
<th>char</th>
<th>z</th>
<th>x</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>10</td>
<td>7</td>
<td>14</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

![Trie diagram]

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

<table>
<thead>
<tr>
<th>char</th>
<th>binary code</th>
</tr>
</thead>
<tbody>
<tr>
<td>'z'</td>
<td>10</td>
</tr>
<tr>
<td>'x'</td>
<td>010</td>
</tr>
<tr>
<td>'a'</td>
<td>00</td>
</tr>
<tr>
<td>'b'</td>
<td>11</td>
</tr>
<tr>
<td>'c'</td>
<td>011</td>
</tr>
</tbody>
</table>

5. Network Flow: 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 and show the cut.

![Flow network diagram]

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

6. Reducibility and NP: Hamiltonian path

   (a) In class we saw the directed Hamiltonian cycle problem, which is the question: Given a directed graph \(G = (V, E)\), is there a simple directed cycle that contains every node in \(G\)? Similarly, a directed Hamiltonian path is a simple path that contains every node in the graph. Show that Hamiltonian path is NP.
(b) We can show that the directed Hamiltonian Path problem is NP complete by reducing the Directed Hamilton Cycle: Given a graph $G = (V, E)$ create a graph $G' = (V', E')$: Pick an arbitrary vertex $v$ in the graph, and split it into two vertices, $v_{in}$ and $v_{out}$. All the edges $(u, v)$ in $G$ become $(u, v_{out})$ in $G'$ and all the edges $(v, u)$ in $G$ become $(v_{in}, u)$ in $G'$. Show this is a polynomial time construction.

**Answer:** We split one vertex in two (add a vertex in total) and add at most $O(|E|)$ number of edges. So we add a polynomial number of elements to the graph, in polynomial time (we only have to go over the edge list).

(c) Show that $G$ has a Hamiltonian cycle if and only if $G'$ has a Hamiltonian path.

**Answer:** $\Rightarrow$ If there is a Hamiltonian cycle in $G$, then it means there is an ordered list of the vertices $v_1, v_2, \ldots, v_n, v_1$ that constitute a simple cycle. Since it’s a cycle and it doesn’t matter where we start, let’s say that $v_1 = v$. Then the vertices $v_{out}, v_2, \ldots, v_n, v_{in}$ constitute a Hamiltonian path in $G'$, since that’s the only difference between the two graphs... All the edges still exist.

$\Leftarrow$ If there is a Hamiltonian path in $G'$, it must start at $v_{out}$ and end in $v_{in}$ since the former has no incoming edges and the latter has no outgoing edges. Therefore, the path looks something like: $v_{out}, v_2, v_3, \ldots v_n, v_{in}$. Take the same order of vertices and replace $v_{out}$ and $v_{in}$ by $v$, and we got a Hamiltonian Cycle in $G$.

(d) (Bonus) Show that in a DAG the HAM-Path problem can be solved in polynomial time.

**Answer:** This was a question in HW3. A DAG has a Hamiltonian Path iff it has only one topological sorting. So you just need to topologically sort the DAG and see whether the vertices, in the topological order, form a Hamiltonian path.