Problem 1

Consider the following scheduling problem. There are $N$ tasks. Each task $T$ has requires unit time and has value $T.V$. There is an overall deadline $D$ and a target value $M$. The tasks are structured in a DAG; no task can be executed until all its predecessors have been completed. There is a single processor that executes one task at a time. The problem is to find a valid schedule that achieves at least the target value. You may assume that $D < N$; otherwise, the problem is trivially solved by doing a topological sort.

For instance, given the 5 tasks with $D = 3$, $M = 8$, and the following structure

```
A 3
   ↗   ↘
   B 1   C 2
   ↗   ↘
   D 2   E 6
```

a correct schedule is B,C,E.

Characterize this as a tree-structured state space search problem. In particular:

- What are the states?
- What are the operators?
- What is the branching factor?
- What is the depth of the tree?
- Is the depth of the goal node known initially? If it is not, give reasonable lower and upper bounds on the depth of the shallowest goal node.

Answer these for the general problem, not just for the particular example above, though you may use this example as an illustration.
Problem 2

Modify problem 1 by allowing task $T$ to require time $T.R$. The constraint that $D < N$ is changed to the requirement that $D < \sum_T T.R$.

For instance, suppose that we have the same graph structure as above, but with the following values and time requirements:

<table>
<thead>
<tr>
<th>Task</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Time</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Then with a target value $M = 7$ and a deadline $D = 10$, a valid solution is $[A, C, D]$.

Characterize this new problem as a tree-structured search problem, answering the same questions as above.

Problem 3

Consider the following greedy algorithm for solving problem 2.

```java
Schedule = empty;
TotalValue = 0;
TotalTime = 0;
while (TotalValue < M) {
    Candidates = { T | T is a task that is ready to run such that TotalTime + T.R <= D }
    if (Candidates == empty) return FALSE;
    T = the task in Candidates with the maximal value of T.V/T.R;
    Add T to Schedule, scheduled to start at time TotalTime;
    TotalValue += T.V;
    TotalTime += T.R;
}
return Schedule;
```

Construct an example where there exists a solution, but this greedy algorithm does not find it.