Problem 1. (25 points [greedy])

Find a maximum-size subset of non-overlapping intervals (or non-conflicting tasks) from the set of 26 intervals shown below, according to the greedy strategy we discussed in class for Activity Selection. Each letter labels a single interval, e.g., --A-- is an interval five units long; intervals --A--, ---H--, ---O---- and ---V--- are overlapping.

Circle the intervals you take. Put an X through each interval you don’t take.

Answer: We select tasks by early finish time, as explained in class. Solution is marked with ====.

==A== ---B---- --C-- ---D---- --E-- --F--- ----G----
---H-- ==I== ----J--- ---K--- --L-- ---M-- --N--
----O---- ----P---- --Q-- ---R---- ==S== ==T=== ====U====
-----V-- --W-- ==X== ===Y==== =Z=

Problem 2. (25 points [greedy])

The Activity Selection problem we discussed in class requires one to find the maximum-size subset of non-overlapping intervals (or non-conflicting tasks). Consider the following greedy algorithm.

1. For each interval $i$ compute the number of other intervals overlapping $i$
2. Sort the task by the number of overlaps, in increasing order (break ties arbitrarily)
3. Pick the task $i$ with the smallest number of overlaps, schedule it, and remove from further consideration tasks that are overlapping with $i$
4. Repeat Step 3 until all tasks are scheduled/discarded (no task is left)

This greedy algorithm is not optimal. Show a counter-example in which this strategy gives a suboptimal solution.

Answer: As said, the strategy is not optimal. Consider the following seven tasks:

The optimal number of tasks is four (boxed), but the algorithm can pick the dashed tasks (the two on the sides have one conflict so they are picked first).

Problem 3. (25 points [dynamic programming])

In this problem you are asked to traceback the solution(s) of an instance of LCS for the strings $x = abcda$, $y = cdabac$. Recall that we define the table $C[i, j]$ as the length of LCS of between the prefix of $x$ of length $i$ and the prefix of $y$ of length $j$. 
\[
C[i,j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
C[i-1,j-1] & \text{if } i > 0, j > 0 \text{ and } x[i] = y[j] \\
\max\{C[i-1,j], C[i,j-1]\} & \text{if } i > 0, j > 0 \text{ and } x[i] \neq y[j]
\end{cases}
\]

1. Draw the traceback pointers for the optimal solution(s)

2. Write the corresponding optimal solution(s)

**Note:** guessing the optimal solutions without showing the traceback won’t give any credit.

**Answer:** There are three LCSs. LCS \text{abc} corresponds to the path \((5,6) \rightarrow (4,6) \rightarrow (3,6) \rightarrow (2,5) \rightarrow (2,4) \rightarrow (1,3) \rightarrow (0,2)\); LCS \text{aba} corresponds to \((5,6) \rightarrow (5,5) \rightarrow (4,4) \rightarrow (3,4) \rightarrow (2,4) \rightarrow (1,3) \rightarrow (0,2)\); LCS \text{cda} corresponds to \((5,6) \rightarrow (5,5) \rightarrow (4,4) \rightarrow (4,3) \rightarrow (4,2) \rightarrow (3,1) \rightarrow (2,0)\).

**Problem 4.** (25 points [dynamic programming (black-box)])

You are given a set \(A = \{a_1, a_2, \ldots, a_n\}\) of \(n\) positive integers such that \(a_1 + a_2 + \ldots + a_n = N\). Design a \(O(nN)\)-time dynamic programming algorithm for determining whether there is subset \(B \subset \{1,2,\ldots,n\}\) such that \(\sum_{i \in B} a_i = \sum_{i \in \{1,2,\ldots,n\}\setminus B} a_i\). For example, if \(A = \{9,2,1,3,5,4\}\) the algorithm should return True because \(1 + 2 + 5 + 4 = 9 + 3\). **Hint:** you can use one of the known dynamic programming algorithms as a black-box.

**Answer:** If \(N\) is odd, return False. If \(N\) is even, then the “target” is \(N/2\). Simply call \text{SUBSET\_SUM}(A,N/2) where \text{SUBSET\_SUM} is the algorithm in Homework 7 (Problem 1). The time complexity is \(O(nN)\).