CS2223, HW4:
Dynamic Programming and Recurrences*

Feel free to work in groups, discuss, debate, and dissect this homework. More enjoyable, and good for learning. Problem A follows on from Problem B in the last homework.

Even though I'm out of town Weds–Fri, the TAs/SA are available, and I'm available via email on the weekend.

A. A Dynamic Algorithm. Unfortunately, the Golden Smacks finance people decide that the projects are not all of equal value. They ask you to rewrite your algorithm using the procedure `payoff(p)` to compute the value of a project p.

You know to use a dynamic algorithm for this kind of problem.

Assume your projects are already sorted in the `projects[]` array in the same order you chose in B.1.

To find the subproblems to solve a given problem, you need to know the *last compatible predecessor* of any job j. The procedure `prev_compat` finds the largest index $i < j$ in `projects[]` such that `projects[i]` is compatible with `projects[j]`. It returns -1 if there is none.

The worst case runtime of `prev_compat` is $\in \Theta(\log_2 k)$, where k is the length of the vector `projects[]`.

A.1. Choosing Subproblems. Suppose that you knew the optimal values `opt(i)` for each subset of the projects of the form

`projects[0]`,..., `projects[i]`.

Give a recursive condition that tells how to compute `opt(i+1)` using this information and the `payoff` function.

*Due: Sunday night, 2 Dec.

Answer. Let $f(i+1) = 0$ if $\texttt{prev\_compat}(i+1) = -1$, and let $f(i+1) = \texttt{opt}(\texttt{prev\_compat}(i+1))$ otherwise.

$$\texttt{opt}(i+1) = \max(\texttt{opt}(i), \quad \texttt{payoff}(\texttt{projects}[i+1]) + f(i+1))$$

A.2. Pseudocode. *Write pseudocode* to fill in an array `M` that stores, for each `M[i]`, the optimal value that you can achieve using compatible projects selected from

`projects[0]`,..., `projects[i]`.

Answer.

```
M[0] = payoff(projects[0]);
for i from 1 to k-1 {
  prev = prev_compat(i);
  if prev = -1 {
    prev_opt = 0
  } else {
    prev_opt = M[prev]
  };
  M[i] = max(M[i-1], payoff(projects[i])+M[prev])
}
```

A.3. Pseudocode to Determine Project List Modify or add to your pseudocode so that it will compute not just the optimal payoff from compatible jobs, but also computes the list of projects to accept to achieve the optimal payoff.

Answer. Work backwards from k. Whenever $M[i] = M[i-1]$, just print out the solution for $i-1$. Whenever $M[i] = M[\text{prev\_compat}(i)]+\text{projects}[i]$, first print the solution for prev_compat(i), and then print i.

A.4. Estimate Runtime. *Estimate the runtime* for your code from A.2 by choosing a f such that it will complete in time $\Theta(f(k))$. You may assume that the procedure call to `payoff` completes in constant time. However, be sure to reflect the time that may be required for the call to your code in `prev_compat`.

Answer. If the runtime of `prev_compat` is $\in \Theta(\log_2 k)$, the runtime will be $\in \Theta(k \log_2 k)$.

Some Problems from *CLRS*.

Sec. 4.1 Do exercises **4.1-1**, **4.1-4**, p. 74. (Related to project 4.)

Sec. 4.4 Do exercises **4.4-1**, **4.4-2**, **4.4-3**, **4.4-4**, p. 92.

Sec. 15.1 On pp. 370, do exercises **15.1-3**, **15.1-4**, **15.1-5**.

Sec. 15.4 On pp. 390, do exercises **15.4-1**, **15.4-2**, **15.4-4**, **15.4-5**.