CSCI 4020 Spring 2021 Design & Analysis of Algorithms

Midterm Exam Solutions

Problem 2

(a) False: Consider a cycle of size 4, with one of the edges e having cost 4 and the rest having cost 1,2,3. Let u and v be the endpoints of e. Then $P = \{e\}$, but the only MST does not contain it.

(b) False: Consider a cycle of edges with costs 1, 2, and 3, attached to an edge of cost 10 leaving the cycle. The edge with cost 10 is in the unique MST, but the edge of cost 3 is not.

Problem 3 Let $OPT[i]$ be the optimum amount of fun you can obtain if you are only given the first i days. Let $j \leq i$ be the last time when you went to a concert. For this to be true, it must be that there is enough time to rest afterwards, i.e., $x_j \leq |i - j|$. Then, the total amount of fun is equal to f_j plus the maximum possible before day j. Then, we know that

$$OPT[i] = \max_{x_j \leq |i-j|} \{f_j + OPT[j-1]\}.$$

We can set the base case to be $OPT[0] = 0$, compute $OPT[i]$ in order of increasing i, and then return $OPT[n]$ as the final answer. The total runtime is $O(n^2)$.

Another correct solution would be to consider $L[i]$ to be the optimum solution if we were given only days $i \ldots n$. Then, on day i we either attend a concert or not (which is possible as long as $x_i \leq n - i$). If we attend the concert, the value we get is $f_i + L[i + x_i + 1]$; otherwise it is just $L[i + 1]$. Thus

$$L[i] = \max\{f_i + L[i + x_i + 1], L[i + 1]\}$$

if $x_i \leq n - i$, and $L[i] = 0$ otherwise. Here we would compute $L[i]$ in order of decreasing i, and the final answer would be $L[1]$.

Finally, another correct approach would be to reduce this problem to Weighted Interval Scheduling with intervals starting at i, ending at $i + x_i$, and weight f_i.

Problem 4 There are many correct ways to solve this problem, here is one. Let $OPT[i, j]$ denote the maximum number of fish that can be caught during days $1 \ldots i$ if the skill level at the end of day i equals exactly j. On day i, there are only two options: to train or to fish. If you train on day i, then $OPT[i, j] = OPT[i - 1, j - 2]$. Otherwise, $OPT[i, j] = OPT[i - 1, j] + min(j, x_i)$. Thus, in total,

$$OPT[i, j] = \max\{OPT[i - 1, j - 2], OPT[i - 1, j] + \min(j, x_i)\}.$$

The subproblems can be solved starting with smaller i values. We can set $OPT[i, 0] = 0$ for all i, and $OPT[i, j] = -\infty$ for $j/2 > i$ since we cannot have more than $2i$ skill on day i. The final answer requires looking over all j in the last column, i.e., returning $\max_j OPT[n, j]$. Filling in each entry $OPT[i, j]$ takes constant time, and there are $2n^2$ such entries (since j can never get larger than $2n$), giving us a $O(n^2)$ runtime.

Common Mistakes for Problem 4

0.1 Subproblems, Recurrence and Runtime

You need a 2D recurrence to consider all relevant (i, j) pairs. A 1D recurrence only considers one possible j value for each day and is greedy/fails. If you defined $OPT[i, j]$ to take the maximum between training on day $i - 1$ and training on day $i - 2$, this is also greedy/fails (see counterexamples below). The question explicitly asks for an $O(n^2)$ runtime, so if your algorithm runs in $O(n^3)$ you lost a few points.

Many people tried to pattern-match a previous homework problem and define $OPT[i, k]$ = max fish that can be caught by day i if the last training was on day k. This neglects the skill level, even if you tried to include it implicitly in the recurrence. Rule of thumb: If your recurrence equation has a variable which changes in each instance of the subproblem (not a constant), then it should be a parameter in your subproblem definition.

0.2 Algorithm: Base Case, Array Order and Return Values

There are two ways to approach the problem: Your subproblems can consider days from $1 \ldots i$ or days $i \ldots n$. For both cases, either j is the skill at the beginning or at the end of day i, and this determines your recurrence. You start day 1 with no fishing skill and always fish on day n (base case: $OPT[1, 0]$ and/or $i = 1$, $OPT[0, 0]$). Since you can only gain a skill of 2 for each day you train, this means that your fishing skill can NEVER be higher than twice the number of the day ($j \leq 2i$). If your table tries to fill in all n^2 possible values, this is an error (need base cases for $OPT[i, i]$ and/or $OPT[i, j]$ where $j > i$ or something like $OPT[0, j]$ for all j depending on your approach). Many people, especially those who looked at days $i \ldots n$, filled in their tables in the wrong order. If your subproblem considers days $1 \ldots i$, your algorithm should fill in your table from low-to high so that OPT[i,j] depends on OPT[i-1,j-2], with something like: $for(i = 1, i \leq n, i++)\{for(j = 0, j \leq i, j++)\{\ldots\}\ldots\}$. If your subproblem considers days $i \ldots n$, your algorithm should fill in your table from high-to-low so that OPT[i,j] depends on OPT[i+1,j+2], as in: $for(i = n, i \geq 0, i--)\{for(j = i-1, j \geq 0, j--)\{\ldots\}\ldots\}$. If you consider days $1 \ldots i$ in each subproblem, it is important to return $\max_j OPT[n, j]$ and not $OPT[n, n]$. If your subproblem considers days $i \ldots n$, ideally you should return $OPT[1, 0]$ or $OPT[2, 1]$. Note that your return value depends on whether you treated j as the skill at the beginning or the end of i. If j is the skill at the beginning of day i then you cannot return $OPT[1, 1]$ because $j < i$ rather than $j \leq i$.

0.3 Counterexamples

Most incorrect algorithms either fail on example 1 or 2 (depending on your approach to the problem). Effectively all incorrect algorithms fail on example 3, which is created to show several different kinds of pitfalls.

1) x : 0 0 0 6 8 8 8 8 8 (Mostly for days 1...i approach)

2) x : 0 0 4 4 4 10 10 10 (Mostly for days i...n approach)

3) x : 0 0 0 8 10 6 8 2 10 12 12 12 12