Introduction to Algorithms	**Problem Set 4 (3 problems)**
CS 4820 Fall 2018	**Due 11:59pm Thursday, September 20**

(1) Pretty Printing. Given an ordered list of words W and a right margin position L, find an efficient algorithm to partition the words into valid lines such that the sum of the *squares* of the slacks of the lines—not including the last line—is minimized. (This choice of loss function reflects that we prefer to have a number of lines with small slack to a few lines with a lot of slack.) You may assume that a valid printing exists, i.e. that no word is longer than L.

Solution A: Reduction to Segmented Least Squares This problem is very much like segmented least squares; in this case, the line segments are lines of text, and the "error" for each line is the slack for that line, with no additional cost C per line. The only difference is that we don't consider the slack of the last line, so our optimal solution can come from any entry entry in the DP table such that the words after that entry form exactly one valid line. So the important parts of this reduction are identifying how to compute the error table and how to make sure the last line doesn't count.

Solution B: Reinvent Solution Student may also reinvent effectively a least-squares-like solution. In this case, the optimal subproblem should still be that $OPT(i)$ is the best solution to minimize slack for *all* lines including the last for words $w_1, \ldots, w_i$. To set up the recurrence such that $OPT(i)$ doesn't include the error of the last line requires some extra bookkeeping to keep track of the error that must be added in when $OPT(i)$ is used as part of the solution for some $OPT(j)$ with $j > i$, as the last line of $OPT(i)$ will no longer be the last line of the full solution and thus we must consider its slack.

Common pieces of both To handle the penalty for each segment (equivalent to error in SLS), we will replace the SSE table with a table of slack penalties S, where s_{ij} is the slack of a line ℓ containing words w_i through w_j. For a valid line, this should be defined for $i \leq j$ as $(L - C_\ell)^2$, or the margin

$$s_{ij} = (L - c_j - \sum_{k=i}^{j-1}(c_k + 1))^2.$$

If the line formed by words $w_i, \cdots, w_j$ is not valid, then the penalty should be prohibitively high; one can use ∞ or some larger penalty than could possibly be accrued by a valid solution (e.g. $(\sum_{i=1}^{n}(c_i + 1))^2$). Note that if they use a finite constant, it must scale with the total number of characters in the document, not just the margin L, as in a very long document the sum of slacks for valid lines must still remain under the total cost to put everything into one line.

Students may observe that they should simply not consider possible candidate lines where the line length would exceed L instead of storing an error value, in which case they need to show how they would decide which segments $w_i, \ldots, w_j$ are invalid.

The recurrence with these slack computations is therefore

$$OPT(i) = min_{1 \leq j \leq i}(s_{ji} + OPT(j-1)).$$

The only necessary base case should be $OPT(0)$, which is 0 (as no lines are needed, no slack is accumulated).

To find the solution, we can look in the error table to find all words that could be part of the final line, i.e. all i such that s_{im} is not ∞. The best solution weight is in one of these i. To reconstruct the schedule, one can trace through (as in segmented least squares) to find which prior line solution gives the combined slack observed.

The complexity to compute the slacks is the most expensive part of this at $O(n^3)$ time. The DP itself takes $O(n^2)$.

Common Mistakes:

- Incorrect runtime analysis: didn't account for the $O(n^3)$ cost to compute slacks. Many students attempted to avoid this by only calculating slacks when they needed to (instead on populating the table with them), but even then the slack calculations contribute $O(n^3)$ runtime.
- Failure to define the subproblem, or defining a subproblem different from the one used in their solution
- Not calculating a value for the slack, s_{ij}

(2) Counting Shortest Paths. Consider a directed graph $G = (V, E)$ where the cost of edge e, c_e, can be negative. Given two nodes s and t, we have seen in class how to compute the length of the min-cost path in G assuming G has no negative cost cycles. In this problem, assume that all cycles have strictly positive costs. This problem concerns two extensions of this problem:

Suppose you have found a new apartment in Ithaca, and you want to assess how long your commute would be from there before you sign a lease. You could initially use the algorithm above to measure the shortest path from your apartment to class, but because of the frequency of construction in Ithaca, you also want to make sure that there is more than one shortest path available to work. In this case, in addition to the length (cost) of the shortest path from s to t, you would also like to know how many path there are from s to t of this shortest length.

(a) *(3 points)* Previously, we computed the length of the shortest path from s to v using at most i edges and stored the value in $OPT(i, v)$. Here, consider the min-cost path using *exactly* i edges (rather than at most i edges). Give a polynomial time algorithm to find the minimum cost path from s to t using *exactly* i edges.

Solution To compute the length of the shortest path using exactly i edges, we need to change the recurrence of (6.23) in the book to only include options that will produce exactly i-length paths, excluding the $M[i-1, v]$ term that includes paths from the previous step:

```
EXACT EDGE COUNT SHORTEST PATH($G, s, t$)
$n =$ number of nodes in $G$
Define $M[0, t] = 0$ and $M[0, v] = \infty$ for all other $v \in V$
For $i = 1, 2, \ldots, n-1$
    For $v \in V$
        Compute $M[i, v] = \min_{(v,w)\in E} c_{vw} + M[i-1, w]$
    Endfor
Endfor
Return $M[i, s]$ as the cost of the min-cost path using exactly $i$ edges.
```

The running time of the i-length shortest path algorithm stays the same as the Bellman-Ford shortest path $O(n^3)$ (or in fact $O(mn)$ if optimized as described in Section 6.8 of the book).

(b) *(5 points)* In addition to the length of the shortest path with i edges, also compute the number of different shortest paths with the same length that use exactly i edges. (There can be exponentially many shortest paths, so outputting them all may not be possible in polynomial time.)

Solution To compute the number of shortest paths, we can augment the table we used for computing the shortest paths. Recall that we previously used $M[i, v]$ to be the minimum cost of a path v to t using i edges. To know how many paths achieved the minimum cost $M[i, v]$, we should also have a table N of the same dimensions as M, where $N[i, v]$ contains the number of min-cost paths from v to t using i edges. Now $M[i, v]$ is set and updated as before. To see how to update $N[i, v]$, we find out which edges $e = (v, w)$ leaving v are part of any shortest path. For the edges leaving v that are part of these shortest paths, we add the values $N[i-1, w]$. Written out as code:

```
COUNT OF SHORTEST PATH($G, s, t$)
$n =$ number of nodes in $G$
Define $M[0, t] = 0$ and $M[0, v] = \infty$ for all other $v \in V$
Define $N[0, t] = 1$ and $N[0, v] = 0$ for all other $v \in V$
For $i = 1, 2, \ldots, n-1$
    For $v \in V$
        Compute $M[i, v] = \min_{v,w\in E} c_{vw} + M[i-1, w]$
        Set $N[i,v] = \sum_{w:M[i,v]=c_{vw}+M[i-1,w],v\neq w} N[i-1,w]$
    Endfor
Endfor
Return $N[i,s]$ as the number of the shortest paths using $i$ edges.
```

We prove correctness of N by induction, showing that $N[i,v]$ is the number of shortest paths from v to t using at most i edges. The base case of $N[0,v]$ is valid by definition. For a shortest path using at most i edges, the first edge is to a node w that is part of the shortest path, so it satisfies $M[i,v] = c_{vw} + M[i-1,w]$, and the number of these paths adds up the number of paths through each of the possible w nodes. In this case, we never will accidentally reconsider an edge that was used in a prior shortest path; to produce a shorter path than other existing options where an edge is reconsidered would require a cycle in the path, and we know that all cycles are positive, so the option with a cycle would be strictly more expensive than another option.
The N array values $N[i,v]$ can each be set as Bellman-Ford runs, proportional to the number of edges leaving the node v. As with M, populating N takes a worst-case $O(n)$ time for each entry to check which edges are part of shortest paths and add together their combined shortest paths, or $O(n^3)$ for the total loop. (This can again be improved to $O(nm)$ running time.)

Common Mistakes:

- A fair number of solutions did not initialize the new table N correctly.
- Most proofs did not fully explain how the algorithm was counting all different paths without double-counting.

(c) *(2 points)* Show how to compute the total number of shortest paths, this time using any number of edges.

Solution To combine these, we can first look at the entries of $M[i,s]$ for all i to find the minimum shortest path length ℓ. Then, we sum the counts of shortest paths in $N[i,s]$ corresponding to the entries where $M[i,s] = \ell$: $M[i,s] = \sum_{i:M[i,s]=\ell} N[i,s]$. In effect, we are summing over how many shortest paths have exactly i edges for all i, where if the shortest paths for some i are all longer than the true shortest path, we add nothing.
It is possible to also do this by redefining N to include the number of paths less than or equal to a certain number of edges; in this case, the proof of the recurrence of N is important, as one needs to ensure paths won't be double-counted. Redefining M to give the shortest path cost with i or fewer edges makes it challenging to find paths with the shortest path cost but more segments than other shortest paths; if you see students doing this, ask them to talk through their recurrence on N and why it is correct.

Common Mistakes:

- There were a few solutions that added all of the $N[i,s]$ without checking the minimum shortest path length ℓ.
- Some created an algorithm analogous to part (b), which instead counts the number of different shortest paths that use at most i edges, and then suggested that its correctness can be proven similarly as in part (b) without saying much else. As said above, we needed to ensure that redefining M and N as such does not lead to double-counting paths.

(3) Graph Coloring (probability review). Next week we'll start on randomized algorithms: taking advantage of random selection in algorithm design. Recall that probability (conditional probability, expectation, variance) was discussed in CS 2800, which is a pre-requisite of this course. For a quick review of probability you may want to read Sections 13.1 and 13.3. We will also generate a quick summary/reminder by Monday of things you need to remember about probabilities, and two of the TAs will run an optional session to help you review the probability basics we need.

Consider an undirected graph G on n nodes. G is called k-colorable if there is a way to color the n nodes with k colors so that the two ends of each edge get different colors. For example, a cycle is always 3-colorable, but only even cycles are 2-colorable. As we will see latter in the course, deciding if a graph is 3-colorable is computationally hard. Instead, we can aim to color the nodes with 3 colors, so that as few edges get identical colors as possible. In this problem, we test out a naive algorithm to do this: choose a color for each node uniformly at random among the three options, with the color sampled independently for each node.

(a) *(3 points)* For a given edge $e = (v, w)$, what is the probability that the two ends of edge e get the same color?

Solution: For any given selected color of v, the probability that w selects the same color is $1/3$. An alternate way to do this is to say that there are 3 colors. For each color c the probability that v and w both get color c is $1/3 * 1/3$(due to independence), and these events are disjoint, so we get $3 * 1/9 = 1/3$ as the probability.

Common Mistakes:

- Sometime student failed to state that that the probability is $1/9$ due to independence.
- Although not a mistake, sometimes student would list all 9 possibilities and saying that 3 out of the 9 have the same color instead of calculating the probabilities analytically. This may be a problem in the future when you can't list every possibility. For example, what would be the answer if there are n colors?

(b) *(3 points)* Assuming G has m edges (that is $m = |E|$), what is the expected number of edges that are colored properly (with the two ends of each edge different colors), and what is the expected number of edges that are badly colored?

Solution: Let $X_e = 1$ if edge e if the edge e badly colored and 0 otherwise, and then $X = \sum_e X_e$ is the number of badly colored edges. By linearity of expectation, the expected number of badly colored edges is $m/3$, and of properly-colored edges is $2m/3$.

Common Mistakes:

- Sometimes students assumed that edges were independent in their coloring and attempted to write out the definition of the expectation: $\sum_j jPr(X = j)$ with $Pr(X = j) = \binom{m}{j}(2/3)^j(1/3)^{m-j}$. Although the answer turns out to be the same, this solution assumes independence. Edges being badly colored are dependent random variables since if two edges in a triangle are badly colored, so is the third.

(c) *(4 points)* Show that the probability that more than half of the edges are badly colored is less than 67%.

Solution: By Markov's inequality, $Pr(X > E(X) * a) \leq 1/a$. We can use this with $a = 3/2$, and get that the probability that more than $1/2$ the edges are badly colored is at most $2/3$.