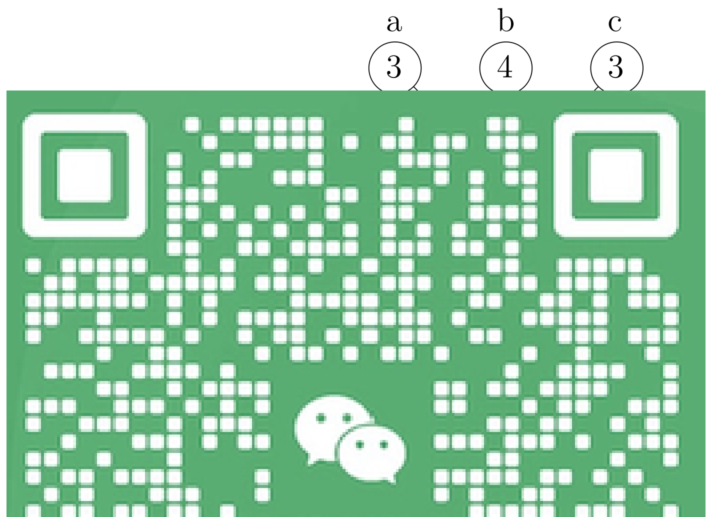
CSCI 3110 Final Exam

posted: 7 am ADT 31.07.2020

Instructor: Travis Gagie

due: midnight ADT 31.07.2020

1. Describe a polynomial-time divide-and-conquer algorithm that, given a tree T with a weight assigned to each vertex, returns a vertex cover with minimum total weight. For example, in the tree shown below, the vertex cover with the minimum total weight is d, f, g, h, i, k — even though the smallest vertex cover is e, g, i. You need not prove your algorithm correct.



(Hint: Becaus [illegible] ver includes either e or f or both. [illegible] including e and T_f including f. I [illegible] htest vertex cover of T_e, and the we [illegible] *includes* e, and the weight $w_3 = 7$ [illegible] $= 10$ of the lightest vertex cover o [illegible] vertex cover of T is $\min(w_1 + w_4,$ [illegible] More interestingly, how do you work out w_2 and w_4?)

Idea: If T is a single vertex, return the empty set. Otherwise, pick an edge (u, v) and remove it. Let T_u be the remaining component containing u, let T_v be the remaining component containing v, let F_u be the forest that results from deleting u and all its incident edges from T_u, and let F_v be the forest that results from deleting v and all its incident edges from T_v.

Recursively find the lightest vertex covers $C(T_u)$, $C(T_v)$, $C(F_u)$ and $C(F_v)$ of T_u, T_v, F_u and F_v, respectively. (To find $C(F_u)$ and $C(F_v)$, recurse on all the trees in F_u and union the results and on all the trees in F_v and union the results.) Return whichever of $C(T_u) \cup \{v\} \cup C(F_v)$ and $C(F_u) \cup \{u\} \cup C(T_v)$ is lighter.

2. Recall that for the NP-complete problem Knapsack we are given a sequence $S = (w_1, p_1), \ldots, (w_n, p_n)$ of weight-profit pairs, a target T and a capacity C, and asked if there exists a subsequence $S' = (w'_1, p'_1), \ldots, (w'_{|S'|}, p'_{|S'|})$ of S such that

$$\sum_{1 \leq i \leq |S'|} w'_i \leq C$$
$$\sum_{1 \leq i \leq |S'|} p'_i \geq T\,.$$

For the problem Pockets we are again given a sequence $S = (w_1, p_1), \ldots, (w_n, p_n)$ of weight-profit pairs and a target T but now, instead of a single capacity C, we are given a sequence $C = c_1, \ldots, c_m$ of capacities. Assume $w_1 \geq \cdots \geq w_n$ and $c_1 \geq \cdots \geq c_m$. We are asked if there exists a subsequence $S' = (w'_1, p'_1), \ldots, (w'_{|S'|}, p'_{|S'|})$ of S with $|S'| \leq m$ such that

In other words ... a capacity, and we want to put a ... tal profit at least T without overfil...

Describe a pol... xample, if

then your algo... $,6),(4,4)$ we obtain profit $5+6+$ ... , $5 \leq 5$ and $4 \leq 5$). You need not ... proof is a good way to catch mistakes anyway).

(Hint: Assuming the pockets are sorted in decreasing order by capacity made it easier to state the problem, but it's not the best order for solving it. What should you put in the smallest pocket?)

Idea: We consider the pockets in increasing order by capacity and, in each pocket, put the most profitable remaining item that fits. This is all the students have to say but it's pretty easy to see why it's correct, so some of them may give a proof anyway.

Before we fill any pocket, our empty subsolution can be extended to an optimal solution. Assume our subsolution after i steps — i.e., after we've filled i pockets — can be extended to an optimal solution S. If S also puts in the $(i+1)$st pocket the most profitable remaining item x that fits, then our subsolution after $i+1$ steps can also be extended to S. Suppose S doesn't put x in the $(i+1)$st pocket. If S puts x in a later pocket then we can change S by swapping the contents of the $(i+1)$st pocket and of that later pocket (i.e., x), to obtain an optimal solution S' that extends our subsolution after $i+1$ steps. If S doesn't put x in a later pocket then we can change S by replacing the contents of of the $(i+1)$st pocket, to obtain a solution S' that extends our subsolution after $i+1$ steps; since x is the most profitable remaining item that fits in the $(i+1)$st pocket, the total profit of S' is at least that of S, so S' is also optimal. By induction, we find an optimal solution.

3. A *semi-wildcard* in a string is special character representing a non-empty subset of the normal alphabet, and a string containing a mix of normal characters and semi-wildcards represents the set of all normal strings that can be obtained by replacing each semi-wildcard by a character from the subset it represents. For example, if the normal alphab[illegible] enting $\{\texttt{B}, \texttt{D}\}$ and ! representing $\{$[illegible] $\texttt{BADCAAD}, \texttt{BADCDAD}\}$.

   Describe a poly[illegible] given a string $\mathcal{S}[1..n]$ containing a m[illegible] rmal string $T[1..m]$, computes

   where ED$(S, T$[illegible] example, given $\mathcal{S} =$ $\texttt{BA?C!AD}$ as des[illegible] d return 1, since the edit distance f[illegible] otice that in general the set of stri[illegible] ny strings, so trying them one by o[illegible] m correct.

   (Hint: This q[illegible] with the "grid with diagonal arrow[illegible] in some rows.)

   Idea: Followi[illegible] an $(m+1)\times(n+1)$ grid with rows [illegible] draw an arrow from cell (i, j) to ce[illegible] to $T[j]$, or if $\mathcal{S}[i]$ is a semi-wildcard representing a set that contains $T[j]$. We then compute the distance from the top left corner to the bottom right corner when moving down or right costs 1, and moving diagonally down and right costs 1 if there is no arrow or 0 if there is.

   Formally, we fill in a matrix $A[0..m][0..n]$ by setting $A[0][0] = 0$, $A[i][0] = i$ for $i > 0$ and $A[0][j] = j$ for $j > 0$, and then using the recurrence

$$A[i][j] = \begin{cases} \min(A[i-1][j]+1, A[i-1][j-1], A[i][j-1]+1) \\ \quad \text{if } T[j] = \mathcal{S}[i] \text{ or } T[j] \in \mathcal{S}[i], \\ \min(A[i-1][j]+1, A[i-1][j-1]+1, A[i][j-1]+1) \\ \quad \text{otherwise.} \end{cases}$$

4. For the problem FAIR 3-COLOURABILITY we are given a graph G on n vertices and asked if it is possible to colour exactly $n/3$ vertices red, exactly $n/3$ vertices green and exactly $n/3$ vertices yellow such that no vertex shares an edge with a vertex of the same colour. Show that FAIR 3-COLORABILITY is NP-complete by both showing it is in NP and reducing a known NP-complete problem to it.

   (Hint: Don't reduce from 3-SAT again; it's easier than that.)

   Idea: FAIR 3-COLOURABILITY is in NP because, given a colouring of G we can check in polynomial time that exactly $n/3$ vertices are red, exactly $n/3$ vertices are green, exactly $n/3$ vertices are yellow and no vertex shares an edge with a vertex of the same colour.

   We reduce 3-COLOURABILITY to FAIR 3-COLOURABILITY. Suppose we are given a graph G on n vertices as an instance of 3-COLOURABILITY. We make a graph G' on $n' = 3$ … nstance of FAIR 3-COLOURABILI…

   Suppose there … 3-colouring of G' if

   (a) we colour …

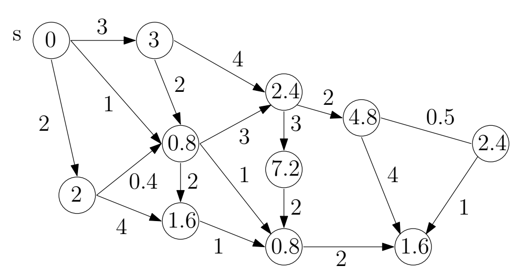
   (b) in the sec… all the green vertices yellow an…

   (c) in the thi… all the green vertices red, and …

   Since each ver… yellow in the third, there are exact…

   Now suppose t… G in G' is 3-coloured by C'.

5. Describe a pol… clic graph G with a positive weight… from s to each other vertex when the length of a path is defined to be the *product* of the weights along that path, instead of their sum. Can you find a faster algorithm if all the edge weights are at least 1? For example, the distances from s to the other vertices are shown in the graph below. You need not prove your algorithm(s) correct.



(Hint: You can solve this either by modifying algorithms you already know or by reducing to the problems they solve.)

Idea: The two ways to do this with positive weights are either to change the Bellman-Ford algorithm to use products instead of sums, which isn't hard but is a mess, or to replace each edge weight by its logarithm, run Bellman-Ford, and then replace each distance by c to that distance, where c is the base of the logarithm (which the students can choose however they like). Similarly, to get a faster algorithm when all the weight are at least 1, they can either modify Dijkstra's algorithm to use products instead of sums, or do the same reduction but using Dikstra's algorithm instead of Bellman-Ford.

6. Suppose that due to various catastrophes, next semester your professor ends up teaching both 3110 and 3136 with exactly the same n students in each one; the 2-hour final
exams are sche oom with exactly n
immovable, ar ly for those 4 hours.
The pandemic worry about social
distancing, bu exam to be sitting
within two me together, so he can't
have all the st t he decides to have
some students the 3136 exam, and
then switch ha for example, there's
a triangle of t other two. Describe
a polynomial-t idea works with the
room's seating

Bonus (1 ma fessor is in the same
situation but xams in eight hours
(perish the tho

(Hint: You sh by yourself, but for
three or more, ndard name for this
problem.)

Idea: This is just graph colouring, where the number of colours is the number of exams. For two exams it's easy, via BFS, but for three or more colours it's NP-complete, according to `https://doi.org/10.1007/PL00009196`.