

CS375 Assignment 3

Due Sep 22d 10PM ET

A general note for CS375: When writing up your homework, please write neatly and explain your answers clearly, giving all details needed to make your answers easy to understand. Graders may not award full credit to incomplete or illegible solutions.

You are encouraged to discuss the problems with your classmates, **but your submission should be your own**. When writing your solutions up, you should not be looking at any other student's submission.

Clear communication **is the point**, on every assignment. In general in CS375, unless explicitly specified otherwise, answers should be accompanied by explanations. Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Necessary acknowledgements: many of these questions are reproductions or adaptations of questions of Eric Aaron's devising. Others are from our course texts as listed on the syllabus.

1. We briefly discussed this in class, but you really ought to know and understand the words **(sub)set** and **permutation**.

- (a) A **set** A of items is a collection of items without any repeats. A **subset** B of A is any set of items all of which belong to A . For example, if $A = \{1, 2, 3\}$ and $B = \{2, 3\}$, then B is a subset of A .

- i. Write out the 8 subsets of $\{1, 2, 3\}$:

- ii. For a given set of n items, how many distinct subsets are there?

- iii. If you haven't seen it before, you should also know the following: given a set A of n items, the number of distinct ways to write a subset of A of size $1 \leq k \leq n$ is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{\prod_{j=1}^k (n-j+1)}{\prod_{j=1}^k j} \leq n^k$$

- iv. The following is pseudo-code for generating a list of all the subsets of a given array of items:

```
def GenerateAllSubsets(A):
    output = [[]] # A list containing an empty list
    for x in A:
        newList = [] # Another new list
        for lst in output:
            add lst to newList
            copyList = copy(lst) # A copy of list
            add x to copyList
            add copyList to newList
        output = newList
    return output
```

Analyze the code to show that its runtime is $\Theta(n2^n)$ (both worst and best case). You can assume the `copy` function takes linear time.

- (b) A **permutation** of an ordered list is just a reordering of that list. For example, $[2, 1, 3]$ is a permutation of the list $[1, 2, 3]$

- i. Write out the 6 permutations of $[1, 2, 3]$
 - ii. How many permutations of a list of n items are there?
 - iii. Write out an algorithm `GENERATEALLPERMUTATIONS` to create a list of all the permutations of a given array. It should look pretty similar to the algorithm above for `GENERATEALLSUBSETS`. Analyze your algorithm and show that it has a runtime of $\Theta(n \times n!)$.
2. In the following list, I'm borrowing a bunch of questions from various texts. I'll use Erickson/-CLRS to refer to our two course texts as listed on our class homepage, but I'll also borrow some questions from *Algorithm Design*, by Jon Kleinberg and Eva Tardos, which I'll refer to by AD.
- Do 2 of the following problems; **your answer should use an exchange argument if you are showing an algorithm is correct.**
- i) (AD 4.3) You are consulting for a trucking company that does a large amount of business shipping packages between New York and Boston. The volume is high enough that they have to send a number of trucks each day between the two locations. Trucks have a fixed limit W on the maximum amount of weight they are allowed to carry. Boxes arrive at the New York station one by one, and each package B_i has a weight w_i . The trucking station is quite small, so at most one truck can be at the station at any time. Company policy requires that boxes are shipped in the order they arrive; otherwise, a customer might get upset upon seeing a box that arrived after his make it to Boston faster. At the moment, the company is using a simple greedy algorithm for packing: they pack boxes in the order they arrive, and whenever the next box does not fit, they send the truck on its way. But they wonder if they might be using too many trucks, and they want your opinion on whether the situation can be improved. Here is how they are thinking: maybe one could decrease the number of trucks needed by sometimes sending off a truck that was less full, and in this way allow the next few trucks to be better packed. Prove that, for a given set of boxes with specified weights, the greedy algorithm currently in use actually minimizes the number of trucks that are needed.
 - ii) (Erickson 4.1) (**This problem counts as two problems** - a midterm question will look exceptionally similar to this one almost certainly, so I'd recommend doing it regardless) The `GREEDYSCHEDULE` algorithm we described for the class scheduling problem is not the only greedy strategy we could have tried. For each of the following alternative greedy strategies, either prove that the resulting algorithm always constructs an optimal schedule, or describe a small input on which it fails. Assume that all algorithms break ties arbitrarily (that is, in a manner that is completely out of your control). **Hint: Three of these algorithms are actually correct.**
 - i. Choose the course x that ends last, discard classes that conflict with x , recurse.
 - ii. Choose the course x that starts first, discard classes that conflict with x , recurse.
 - iii. Choose the course x that starts last, discard classes that conflict with x , recurse.
 - iv. Choose the course x with the shortest duration, discard classes that conflict with x , recurse.
 - v. If none of the courses conflict, choose them all; otherwise, discard the course of longest duration, recurse.
 - vi. If none of the courses conflict, choose them all; otherwise, discard the course which conflicts with the most other courses, recurse.
 - vii. Let x be the class with the earliest start time, and let y be the course with the second earliest start time.
 - If x and y are non-overlapping, choose x and recurse on everything but x (including y).
 - Else if x completely contains y , discard x and recurse.

- Otherwise, discard y and recurse.
- viii. If any course x completely contains another course, discard x and recurse. Otherwise, choose the class y that ends last, discard all classes that conflict with y , and recurse.
- iii) (Erickson 4.5) Let X be a set of n intervals on the real line. A proper coloring of X assigns a color to each interval, so that any two overlapping intervals are assigned different colors. Describe and analyze an efficient algorithm to compute the minimum number of colors needed to properly color X . Assume that your input consists of two arrays $L[1..n]$ and $R[1..n]$, representing the left and right endpoints of the intervals in X .
- iv) (Erickson 4.17) Suppose we have n skiers with heights given in an array $P[1..n]$, and n skis with heights given in an array $S[1..n]$. Describe an efficient algorithm to assign a ski to each skier, so that the average difference between the height of a skier and her assigned ski is as small as possible. (**Hint: You'll probably want to do some drawing for this one**)
- v) (AD 4.5) Let's consider a long, quiet country road with houses scattered very sparsely along it. (We can picture the road as a long line segment, with an eastern endpoint and a western endpoint.) Further, let's suppose that despite the bucolic setting, the residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within four miles of one of the base stations. Give an efficient algorithm that achieves this goal, using as few base stations as possible.
- vi) (Erickson 4.19) Suppose we are given two arrays $C[1..n]$ and $R[1..n]$ of positive integers. An $n \times n$ matrix of 0s and 1s agrees with R and C if, for every index i , the i th row contains $R[i]$ 1s, and the i th column contains $C[i]$ 1s. Describe and analyze an algorithm that either constructs a matrix that agrees with R and C , or correctly reports that no such matrix exists.
- vii) (AD 4.6) Your friend is working as a camp counselor, and he is in charge of organizing activities for a set of junior-high-school-age campers. One of his plans is the following mini-triathlon exercise: each contestant must swim 20 laps of a pool, then bike 10 miles, then run 3 miles. The plan is to send the contestants out in a staggered fashion, via the following rule: the contestants must use the pool one at a time. In other words, first one contestant swims the 20 laps, gets out, and starts biking. As soon as this first person is out of the pool, a second contestant begins swimming the 20 laps; as soon as he or she is out and starts biking, a third contestant begins swimming... and so on. Each contestant has a projected swimming time (the expected time it will take them to complete the 20 laps), a projected biking time (the expected time it will take them to complete the 10 miles of bicycling), and a projected running time (the time it will take them to complete the 3 miles of running). Your friend wants to decide on a schedule for the triathlon: an order in which to sequence the starts of the contestants. Let's say that the completion time of a schedule is the earliest time at which all contestants will be finished with all three legs of the triathlon, assuming they each spend exactly their projected swimming, biking, and running times on the three parts. (Again, note that participants can bike and run simultaneously, but at most one person can be in the pool at any time.) What's the best order for sending people out, if one wants the whole competition to be over as early as possible? More precisely, give an efficient algorithm that produces a schedule whose completion time is as small as possible.
- viii) (AD 4.7) The wildly popular Spanish-language search engine El Goog needs to do a serious amount of computation every time it re-compiles its index. Fortunately, the company has at its disposal a single large supercomputer, together with an essentially unlimited supply of high-end PCs. They've broken the overall computation into n distinct jobs, labeled $J_1, J_2, \dots, J_n$, each of which can be performed completely independently of one another. Each job consists of two stages: first it needs to be preprocessed on the supercomputer, and then it needs to be finished on one of the PCs. Let's say that job J_i needs p_i seconds of time on the supercomputer, followed by f_i seconds of time on a PC. Since there are at least n PCs available on the premises, the finishing of the jobs can be performed fully in parallel all the jobs can be processed at the same time. However, the supercomputer

can only work on a single job at a time, so the system managers need to work out an order in which to feed the jobs to the supercomputer. As soon as the first job in order is done on the supercomputer, it can be handed off to a PC for finishing; at that point in time a second job can be fed to the supercomputer; when the second job is done on the supercomputer, it can proceed to a PC regardless of whether or not the first job is done (since the PCs work in parallel); and so on. Let's say that a schedule is an ordering of the jobs for the supercomputer, and the completion time of the schedule is the earliest time at which all jobs will have finished processing on the PCs. This is an important quantity to minimize, since it determines how rapidly El Goog can generate a new index. Determine the ordering which obtains a minimal completion time of jobs.

- ix) (AD 4.15) The manager of a large student union on campus comes to you with the following problem. She's in charge of a group of n students, each of whom is scheduled to work one shift during the week. There are different jobs associated with these shifts (tending the main desk, helping with package delivery, rebooting cranky information kiosks, etc.), but we can view each shift as a single contiguous interval of time. There can be multiple shifts going on at once. She's trying to choose a subset of these n students to form a supervising committee that she can meet with once a week. She considers such a committee to be complete if, for every student not on the committee, that student's shift overlaps (at least partially) the shift of some student who is on the committee. In this way, each student's performance can be observed by at least one person who's serving on the committee. Give an efficient algorithm that takes the schedule of n shifts and produces a complete supervising committee containing as few students as possible.

- x) (AD 4.19) A group of network designers at the communications company CluNet find themselves facing the following problem. They have a connected graph $G = (V, E)$, in which the nodes represent sites that want to communicate. Each edge e is a communication link, with a given available bandwidth b_e . For each pair of nodes u, v in V , they want to select a single path P between u and v on which this pair will communicate. The bottleneck rate $b(P)$ of this path P is the minimum bandwidth of any edge it contains; that is, $b(P) = \min_{e \in P} b_e$. The best achievable bottleneck rate for the pair u, v in G is simply the maximum, over all paths P between u and v in G , of the value $b(P)$.

It's getting to be very complicated to keep track of a path for each pair of nodes, and so one of the network designers makes a bold suggestion: Maybe one can find a spanning tree T of G so that for every pair of nodes u, v , the unique path between u and v in the tree actually attains the best achievable bottleneck rate for u, v in G . (In other words, even if you could choose any path between u and v in the whole graph, you couldn't do better than the path between u and v in T .) This idea is roundly heckled in the offices of CluNet for a few days, and there's a natural reason for the skepticism: each pair of nodes might want a very different-looking path to maximize its bottleneck rate; why should there be a single tree that simultaneously makes everybody happy? But after some failed attempts to rule out the idea, people begin to suspect it could be possible.

Show that such a tree exists, and give an efficient algorithm to find one. That is, give an algorithm constructing a spanning tree T in which, for each u, v in V , the bottleneck rate of the path between u and v in T is equal to the best achievable bottleneck rate for the pair u, v in G .

- xi) (AD 4.29) Given a list of n non-negative integers $d_1, d_2, \dots, d_n$, show how to decide in polynomial time whether there exists an undirected graph $G = (V, E)$ whose node degrees are precisely the numbers $d_1, \dots, d_n$. (That is, if $V = \{v_1, \dots, v_n\}$, then the degree of v_i should be exactly d_i .) G should not contain multiple edges between the same pair of nodes, or loop edges with both endpoints equal to the same node.
- xii) (Erickson 4.24) Congratulations! You have successfully conquered Camelot, transforming the former battle-scarred hereditary monarchy into an anarcho-syndicalist commune, where citizens take turns to act as a sort of executive-officer-for-the-week, but with all the decisions of that officer ratified at a special bi-weekly meeting, by a simple majority in the case of purely internal affairs, but by a two-thirds majority in the case of more major...

As a final symbolic act, you order the Round Table (surprisingly, an actual circular table) to be split into pizza-like wedges and distributed to the citizens of Camelot as trophies. Each citizen has submitted a request for an angular wedge of the table, specified by two angles for example: Sir Robin the Brave might request the wedge from 17.23° to 42° , and Sir Lancelot the Pure might request the 2° wedge from 359° to 1° . Each citizen will be happy if and only if they receive precisely the wedge that they requested. Unfortunately, some of these ranges overlap, so satisfying all the citizens' requests is simply impossible. Welcome to politics.

Describe and analyze an algorithm to find the maximum number of requests that can be satisfied. **Hint: The output of your algorithm should not change if you rotate the table. Do not assume that angles are integers.**

- xiii) (AD 1.6, kinda) For this problem, imagine you own a fleet of freight trains and an equal number of stations. Each one of these trains T_i has a schedule S_i for this month (you can assume months are arbitrarily long in this universe), during which every train visits every station exactly once. You've been informed by your engineers that every one of these trains has a problematic flaw in design and will need to be held in one of your stations for the rest of the month. These problems aren't exactly immediate - they can be held until the end of the month. Unfortunately, your stations aren't very big - each station can only hold one train, and another train cannot pass through the station while another is held there. You've been asked to truncate (end early) each train's schedule at a unique station: whatever this station is, it will remain there for the remainder of the month. Your goal is to do this so that:

- i. No other train passes through a station that holds another train
- ii. The number of stops made by all trains is maximized - equivalently, the number of stops lost due to truncation is minimized.

As an example, imagine that we had two trains and two stations, where the schedule was:

Train	Day 1	Day 2	Day 3	Day 4
T_1	S_1	.	S_2	.
T_2	.	S_1	.	S_2

If T_1 were truncated at S_1 , then T_2 wouldn't be able to pass through the station, and thus no valid truncation would follow. Hence T_1 must be truncated at S_2 , meaning T_1 visits both S_1 and S_2 while T_2 must truncate at S_1 , losing its visit to S_2 .

Determine an algorithm that successfully achieves your goals. (**Hint: How is this just a Stable-Marriage problem?**)

- xiv) (AD 1.7) Some of your friends are working for CluNet, a builder of large communication networks, and they are looking at algorithms for switching in a particular type of input/output crossbar. Here is the setup. There are n input wires and n output wires, each directed from a source to a terminus. Each input wire meets each output wire in exactly one distinct point, at a special piece of hardware called a junction box. Points on the wire are naturally ordered in the direction from source to terminus; for two distinct points x and y on the same wire, we say that x is upstream from y if x is closer to the source than y , and otherwise we say x is downstream from y . The order in which one input wire meets the output wires is not necessarily the same as the order in which another input wire meets the output wires. (And similarly for the orders in which output wires meet input wires.) Figure 1.8 gives an example of such a collection of input and output wires. Now, here's the switching component of this situation. Each input wire is carrying a distinct data stream, and this data stream must be switched onto one of the output wires. If the stream of Input i is switched onto Output j , at junction box B , then this stream passes through all junction boxes upstream from B on Input i , then through B , then through all junction boxes downstream from B on Output j . It does not matter which input data stream gets switched onto which output wire, but each input data stream must be switched onto a different output wire. Furthermore - and this is the tricky constraint - no two data streams can pass through the same junction box following the switching operation. Finally, here's the problem. Show that for any specified pattern in which the input wires and output wires meet each other (each pair meeting exactly

once), a valid switching of the data streams can always be found – one in which each input data stream is switched onto a different output, and no two of the resulting streams pass through the same junction box. Additionally, give an algorithm to find such a valid switching. (**Hint: This one is also a stable-marriage problem**)

- xv) If looking for more problems, there are a bunch of really good ones in Erickson that I didn't bother copying over – however, Erickson covers Greedy algorithms after a bunch of other techniques, and so he actually made his Greedy chapter about discerning which types of problems a greedy strategy will even work for. So if you're stuck on one and you're not sure if a greedy strategy would even work, feel free to come chat with me about it!