CMSC 27200 - Winter 2021 — Andrew Drucker, Lorenzo Orecchia
due 6 hours after first access via gradescope

M1.Hamlet

- Submit on Gradescope within **6 hours of your first access**. Please note down the time of your access now on the top of this page, above the M1 heading.
- Accesses are logged by the system. **Your timeliness is expected.**
- Timestamps in the form of carefully, completely-photographed work sent by email are admissible in a technical emergency.
- Do not share or make the exam accessible to anyone other than yourself. Any suspected violation of this policy will constitute a breach of the University's academic integrity policy and will be reported.
- You can either type your solutions using LaTeX or scan your handwritten work. You will find a LaTeX template in the midterm folder. If you are writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will facilitate the grading.
- **No collaboration and no use of external resources** are allowed for this midterm. Any suspected violation will be investigated and reported. You may consult the textbook, the material and videos linked from the Canvas Modules page and any notes that you took for the course.
- If you absolutely need to contact the teaching team, please use a private message on Ed.

PROBLEM 1 (25 POINTS)

Time started: 3:57 Here is an algorithm that attempts to produce a stable matching between N hospitals and N students.

1. Let M be an arbitrary initial matching between hospitals and students;
2. Let (h, s) and (h', s') be arbitrary pairs in M such that (h, s') form an unstable pair with respect to M, i.e., an unmatched pair such that both parties prefer each other over their current match. If no such pair exists, **halt** with output M.
3. Modify M by swapping the assignments; remove the previous two pairs, and add two new pairs matching (h, s') and (h', s)
4. Goto line 2.

We will show that this algorithm can run forever without producing a stable matching even for $N = 3$. To accomplish this task, denote the hospitals as $H = \{h_1, h_2, h_3\}$, and the students as $S = \{s_1, s_2, s_3\}$. Construct a counterexample by completing the following parts.

(a) Fill in the blanks in the following preference lists for your counterexample. The leftmost entry is the *most* preferred; the rightmost is the *least*.

Solution:

h_1:	s_2,	s_1,	s_3.	s_1:	h_1,	h_3,	h_2.
h_2:	s_1,	s_2,	s_3.	s_2:	h_3,	h_1,	h_2.
h_3:	s_1,	s_2,	s_3.	s_3:	h_1,	h_2,	h_3.

(b) We assume, without loss of generality, that the initial matching is $\{(h_1, s_1), (h_2, s_2), (h_3, s_3)\}$. Fill in the following table with with the execution of the algorithm that leads to an infinite loop. In the "Iteration #" column, write down the iteration number; in the "Unstable Pair" column, write down the unstable pair that triggers a swap; in the "Matching After Swap" column, write down the matching after the swap is performed.

Solution:

Iteration #	Unstable Pair	Matching After Swap
1	(h_1, s_2)	(h_1, s_2), (h_2, s_1), (h_3, s_3)
2	(h_3, s_2)	(h_1, s_3), (h_2, s_1), (h_3, s_2)
3	(h_3, s_1)	(h_1, s_3), (h_2, s_2), (h_3, s_1)
4	(h_1, s_1)	(h_1, s_1), (h_2, s_2), (h_3, s_3)
5	(h_1, s_2)	(h_1, s_2), (h_2, s_1), (h_3, s_3)
6	(h_3, s_2)	(h_1, s_3), (h_2, s_1), (h_3, s_2)
7	(h_3, s_1)	(h_1, s_3), (h_2, s_2), (h_3, s_1)

problem 2 (25 points)

Give an algorithm that takes as input an undirected, weighted graph $G = (V, E, w)$, with distinct edge weights w, and a subset of edges of $S \subseteq E$, which does not contain a cycle, and returns the spanning tree of G of minimum weight out of all spanning trees containing all edges in S. Prove that your algorithm is correct and give an explicit Big-O bound on its running time as a function of $|E|$ and $|V|$.

Solution:

We will be using Kruskal's algorithm and the Union-Find Abstract Data Structure
Let $|V| = n$ and $|E| = m$ and $|S| = o$
We know $w(e_1) < w(e_2) < \ldots < w(e_m)$
Let $K = S$, where Kruskal's algorithm begins with the edges already in set S.
Let e' be in $E - S$

```
for $i = 1$ to $|E - S|$ do
    if $K \cup e'_i$ is cycle free then
        $K \leftarrow K \cup e'_i$
    end if
end for
Return $T \leftarrow K$
```

Proof of Runtime The Runtime for our algorithm is $O((m - o)logn)$. Firstly, Kruskal's algorithm works on a sorted array of edges. Since we are given that w is the set of the weights of all edges, and all weighst are distinct, we know we can order the weights such like $w(e_1) < w(e_2) < \ldots < w(e_m)$. This sorting takes at most $O(mlogn)$ when Kruskal's algorithm begins with an empty set of edges. However, our greedy alrgotihm begins with the given set S of size o and so the number of edges *left* that must be sorted is equal to $m - o$. Therefore, our runtime $O((m - o)logn)$.

Proof of Correctness We will now prove our algorithm is correct. We will do this using an exchange argument where we prove that Kruskal's algorithm retruns a Minimum Spanning Tree even when starting with a non-empty set of edges.

Proof of Feasibility First we want to show that this is feasible. According to Kruskal's algorithm, it builds a subtree, K, of graph G by adding edges that do not create any cycles within the tree previously built. We are given that S does not contain a cycle. Therefore, it checks out. However, given a set of edges we must still establish the partitions of which the edges in set S will be a part of. For that, we can use the function (as per the KT textbook) *MakeUnionFind* where we can define the partitions over set S and set $E - S$ such that $P_{e_1}, P_{e_2}, \ldots P_{e_m} \cup P_{e'_1}, P_{e'_2}, \ldots P_{e'_o}$

Proof of Optimality Now we want to show that even when starting with a given set of edges, one that does not contain any cycles, Kruskal's algorithm still outputs an MST. We will set up an exchange argument and prove inductively. First we will define our working solutions. Let K be the spanning tree of our algorithm and O be some other optimal spanning tree. Our base is our minimum requirement, our given S. If K = S and S = E then we are done, for the MST is equal to the E. However, if $K = S \neq E$ then we cannot assume our current partial tree is connected and will move onto our inductive hypothesis. We want to assume that for k iterations of our algorithm $K = S$ holding the current minimum weight, we want to check if the next edge is the minimum edge. Let's assume that if our K does not output the MST, then there must exist some edge e in $E - S$ that can be swapped out for some e' such that e does not add a cycle and the total weight of the spanning tree is at least as small as the weight of O. Let $w(e) < w(e')$. Well, we know it is feasible to Find(i), Find(j) on the edge e because it does not creat a cycle. Since edge e does not belong to a partition already, we can merge it into our spanning tree depending on if vertex i or vertex j are in a partion or different partitions. Since we can merge *e* into our spanning tree then the weight of our spanning tree K is at least as small as the weight of the optimal spanning tree O. We know that our algorithm is optimal because there was no loss of weight.

Extra Space for your solution

PROBLEM 3 (25 POINTS)

Suppose that we have an array $A = [a_0, a_1, a_2, ..., a_{n-1}]$ of integers and an array $S = [s_0, s_1, ..., s_{n-1}]$ of partial sums of A, i.e., each element $s_j = \sum_{i=0}^{j} a_i$ of S is the sum of the first $j+1$ elements of A. For example, we could have that $A = [2, 3, -4, 2, 1, 6, -8, 5, -12, 4]$ and $S = [2, 5, 1, 3, 4, 10, 2, 7, -5, -1]$.

Give a fast algorithm which, provided that the sum of the elements of A is odd, finds an odd element of A. Show that your algorithm is correct and analyze its running time by proving an explicit Big-O bound as a function of n. A solution that runs in $\Omega(n)$ time will receive no credit, as this is trivially achieved by checking elements one-by-one.

Solution:

in case i don't finish this- this is a divide and conquer problem where we can use the (mathematical axiom??) where only odd + even numbers = odd numbers. Given that the total sum of A is odd, we can always divide the given by 2 and recurse over the side with an odd partial sum. I got confused with indexing

```
$Algorithm(A, S)$
Base Case:
if $A[j-1]$ then
    return $A[j-1]$ if $A[j-1]\%2 == 1$
else
    return $A[j]$
end if
Split:
m := median of Set $S$ (assuming $\frac{|S|}{2}$ and taking the floor)
S′ := the list of partial sums for the first $m+1$ values of A.
S″ := the list of partial sums for the last $n-m+1$ values of A.
Recursion:
if $S[m]$ == odd then
    return $Algorithm(A, S')$.
else
    return $Algorithm(A, S'')$ .
end if
```

Proof of Runtime This algorithm has runtime similar to a binary search since for each recursion, we split the list of partial sums S in half, check if the the median is an odd number, and if it is we recurse over the first half of S and if it is not we recurse over the last half of S. We know we can "throw away" either side of S because odd sums are always composed of even and odd sums. Thus, simply checking the median tells us if there exists an odd sum earlier or later. This allows our branching constant to be 1, such that we only check 1 of the branches at most. Next, we know that we are dividing our list of partial sums by even-ness or odd-ness so at each recursion we decide between 2 branches. This means our input is only ever divided by 2. Lastly, our work value is O(n) because we do linear-time work to find the median of list S (in other words, finding the median of S is a function of the size of S). Thus our recurrence relation is $T(n) \le T(\lceil \frac{n}{2} \rceil) + O(1)$. By the master theoreom we know our algorithm runs in $T(n) \le O(n)$.

Proof of Correctness We Will prove this algorithm by induction such that $Algorithm(A, S)$ on any input with $size(x) \le k$ returns a correct output. The size(x) we have chosen carefully for our problem.

Base Case Correctness follows very simply from our base case. Let set S be of size 1 and be an odd partial sums, then we know that S directly reflects an odd value in A. For example, let $A = 1, 2$ and $S = 3$. Then we know $A[j-1] == A[0] == 1$ is odd.

Inductive Step We will show using strong induction. Assume correct for k ¿ 1 iterations and has been proved for all values k' ¿ k. We'll show it for k. Consider x of size at most k. If size(x) ¡ k then correctness for x has already been showen, so assume now that size(x) = k. We claim the generated instances of S', S'' are of size less than k. Thus the recursive calls return correct solutions for A[j].

Lemma 1 If S', S'' are of size less than k, then we know that $S'[m]$ or $S''[m]$ is an odd value. Since, m represents the median of the new sets we have split our input by at most $\lceil \frac{k}{2} \rceil$. Since S' and S'' are sets of running sums then their respctive lists have a total of an odd value. Since only odd and even values together can make a sum of odd, there must exist an odd value in at most one of these sets at a time. Thus this yields a correct solution. the median of either set is an odd value. At each recursive step, to define S' and S'' we must take the median of parent set S. This is a function that linearly computes over all k values once. This has runtime $O(n)$.

From Lemma 1 and our inductively assumed correctness of S' and S'' it follows that our output for x is correct/ The induction extends to value k. $Algorithm(A, S)$ has been proven for all inputs.

Efficiency From our runtime bound in Lemma 1 an dthe fact in our non-base we make two recursive calls in this code to inputs of $\lceil \frac{k}{2} \rceil < k$ we conclude that the function T(n) defined as maximum runtime on any input of size at most n obeys a recurrence of form $T(n) \leq T(\lceil \frac{n}{2} \rceil) + O(n)$. We now freely quote the standard case of the Master Theorem which informs us that $T(n) \leq T(\lceil \frac{n}{2} \rceil) + O(n) \Rightarrow T(n) \leq O(n)$

We have proved correctness and efficiency.

Extra Space for your solution

PROBLEM 4 (25 POINTS)

Suppose that an emergency vehicle drives from city A to city B over a straight highway that is split in n segments. Each segment i has a length l_i and a speed limit u_i. The driver has special permissions that allow her to violate the speed limit by v units for a total time T. Specifically, for each segment $i \in \{1, 2, \ldots, n\}$, the driver has to choose for which duration $t_i \geq 0$ she will travel with the higher speed $u_i + v$, under the constraint that the total violation time $\sum_{i=1}^{n} t_i$ is at most T.

Devise a polynomial-time greedy algorithm that will allow the driver to go from city A to city B in a minimum amount of time, if she is allowed to violate the speed limit for a total time T and by v units of speed. Prove the correctness of your algorithms and give a Big-O bound on the running time as a function of n.

Hint: Write a closed-form expression for the time needed to traverse a segment i, if the driver violates the speed limit and travels at speed $u_i + v$ for a duration t_i. Notice that t_i cannot be arbitrarily large.

Solution: Your solution goes here.

Extra Space for your solution