CPSC 320 2017W2: Assignment 3

Please submit this assignment via GradeScope at `https://gradescope.com`. Be sure to identify everyone in your group if you're making a group submission. (Reminder: groups can include a maximum of three students; we strongly encourage groups of two.)

Remember that this assignment is based on the collected quizzes and quiz solutions. You will likely want to refer to those where you need more details on assignment questions.

Submit by the deadline **Monday 5 Mar at 10PM**. For credit, your group must make a **single** submission via one group member's account, marking all other group members and the pages associated with each problem in that submission **using GradeScope's interface**. Your group's submission **must**:

- Be on time.
- Consist of a single, clearly legible file uploadable to GradeScope with clearly indicated solutions to the problems. (PDFs produced via LaTeX, Word, Google Docs, or other editing software work well. Scanned documents will likely work well. **High-quality** photographs are OK if we agree they're legible.)
- Include prominent numbering that corresponds to the numbering used in this assignment handout (not the individual quizzes). Put these **in order** starting each problem on a new page, ideally. If not, **very clearly** and prominently indicate which problem is answered where!
- Include at the start of the document the **ugrad.cs.ubc.ca e-mail addresses** of each member of your team. (Please do **NOT** include your name on the assignment, however.[1])
- Include at the start of the document the statement: "All group members have read and followed the guidelines for academic conduct in CPSC 320. As part of those rules, when collaborating with anyone outside my group, (1) I and my collaborators took no record but names (and GradeScope information) away, and (2) after a suitable break, my group created the assignment I am submitting without help from anyone other than the course staff." (Go read those guidelines!)
- Include at the start of the document your outside-group collaborators' ugrad.cs.ubc.ca IDs, but **not** their names. (Be sure to get those IDs when you collaborate!)

1 Ol' McDonald is safer, he hi he hi ho

Here is pseudocode for a correct greedy algorithm that solves the McDonald's safety committee problem:

```
define produce_safety_committee(array_of_shifts):
    $S_{max}$ = none           # where s(none) = f(none) = $-\infty$
    previous_$S_{max}$ = none

    list_current = empty list
```

[1]If you don't mind private information being stored outside Canada and want an extra double-check on your identity, include your student number rather than your name.

```
all_endpoints = all shift endpoints, sorted in increasing order

for endpoint p in all_endpoints:
    let S be the shift to which p belongs
    if p is a left endpoint:
        set $S_{max}$ to S if f(S) > f($S_{max}$)
        #
        # Shifts starting before f(previous_$S_{max}$) are already covered.
        # so we don't need to worry about covering them.
        #
        if p < f(previous_$S_{max}$):
            mark S as covered
        else:
            add S to list_current

    else if S is not marked as covered:
        previous_$S_{max}$ = $S_{max}$
        add $S_{max}$ to the committee
        mark all shifts from list_current as covered
        list_current = empty list
```

In this question you will complete a proof of correctness of this algorithm.

1. Let $S_1, \ldots, S_k$ be the set of shifts returned by our algorithm. Assume without loss of generality that $S_1, \ldots, S_k$ is sorted by finishing times. We will denote the starting and finishing times of shift S_j by $s(S_j)$ and $f(S_j)$ respectively. We first prove that $S_1, \ldots, S_k$ is a valid safety committee, by proving that:

   Fact 1 *If S is a shift such that $s(S) \leq f(S_j)$, then S overlaps one of $S_1, \ldots, S_j$.*

   Proof: The proof is by induction on j. For $j = 1$, observe that **FILL IN THIS PART**.

   Suppose now that every shift whose starting point is $\leq f(S_j)$ overlaps one of $S_1, \ldots, S_j$. Consider a shift S such that $s(S) \leq f(S_{j+1})$.

   - If $s(S) \leq f(S_j)$ then **FILL IN THIS PART**.
   - Otherwise, **FILL IN THIS PART**. Thus $f(S) \geq f(T)$, which means that it overlaps with S_{j+1}.

   End Proof

2. Let $T_1, \ldots, T_m$ be the shifts in a smallest safety committee, sorted by increasing finishing time. Because our algorithm returns a complete safety committee (select the correct statement):

   ○ $k < m$
   ○ $k \leq m$
   ○ $k = m$
   ○ $k \geq m$
   ○ $k > m$

   First we establish that

   Fact 2 *For $j = 1, \ldots, m$, $f(S_j) \geq f(T_j)$.*

Proof: This is clear for $j = 1$, because **FILL IN THIS PART**

Suppose now that it is true for S_j and T_j, and consider S_{j+1} and T_{j+1}. Let S be the shift with the earliest finishing time amongst those that start after $f(S_j)$.

FILL IN THIS PART

Hence $f(T_{j+1}) \leq f(S_{j+1})$.

End Proof

In order to prove that (select the correct statement)

◯ $k < m$
◯ $k \leq m$
◯ $k \geq m$
◯ $k > m$

we only need to prove that every shift overlaps an element of $S_1, \ldots, S_m$. Indeed, if there were a shift S that does not overlap any of them, then **FILL IN THIS PART**

This is clearly impossible, since in that case none of $T_1, \ldots, T_m$ would overlap with S.

2 Recurrences resolve runtimes

1. Consider the following sorting algorithm:

```
define sloth_sort(A, first, last):
  if A[first] < A[last]:
      exchange A[first] and A[last]

  if (first + 1 < last):
    mid = (last - first + 1) // 3     # integer division
    sloth_sort(A, first + mid, last)  # sort last two-thirds
    sloth_sort(A, first, last - mid)  # sort first two-thirds
    sloth_sort(A, first + mid, last)  # sort last two-thirds again
```

a. Write a recurrence relation that describes the worst-case running time of function `sloth_sort` in terms of n, where $n = \texttt{last} - \texttt{first} + 1$. You can ignore floors and ceilings.

b. What is the worst case running time of algorithm `sloth_sort` ?

2. Consider now this much less useful algorithm:

```
define not_useful(A, first, last):
  if last < first + 4:
      x = A[last] - A[first]
  else:
    x = not_useful(A, first+1, last-1) - not_useful(A, first+2, last-2)
  return x
```

Write a recurrence relation that describes the worst-case running time of function `not_useful` in terms of n, where $n = \texttt{last} - \texttt{first} + 1$.

3. Finally, consider the following Python algorithm, which we may see again later this term:

```
def deterministic_select(A, i):
  if len(A) < 5:
    sorted_A = sorted(A)
    return sorted_A[i-1]

  blocks = [ ]
  for b in range((len(A)-1) // 5 + 1):  # That is, the ceiling of len(A)/5
    blocks.append(sorted(A[b*5:(b+1)*5]))

  medians = [block[len(block)//2] for block in blocks]
  median_of_medians = deterministic_select(medians, len(medians) // 2 + 1)

  lesser  = [v for v in A if v <  median_of_medians]
  greater = [v for v in A if v >  median_of_medians]
  moms    = [v for v in A if v == median_of_medians]

  if len(lesser) < i <= len(lesser) + len(moms):
    return median_of_medians
  elif len(lesser) >= i:
    return deterministic_select(lesser, i)
  else:
    return deterministic_select(greater, i - len(lesser) - len(moms))
```

Knowing that `0.3 len(A) <  len(lesser) <  0.7 len(A)`, write a recurrence relation that describes the worst-case running time of function `deterministic_select` in terms of n, where $n =$ `len(A)`.

3 Playing the Blame Game

A distributed computing system composed of n nodes is responsible for ensuring its own integrity against attempts to subvert the network. To accomplish this, nodes in the system can assess each others' integrity, which they always do in pairs. A node in such a pair with its integrity intact will correctly assess the node it is paired with to report either "intact" or "subverted". However, a node that has been subverted may freely report "intact" or "subverted" regardless of the other node's status.

The goal is for an outside authority to determine which nodes are intact and which are subverted. If $n/2$ or more nodes have been subverted, then the authority cannot necessarily determine which nodes are intact using any strategy based on this kind of pairing. However, if more than $n/2$ nodes are intact, it is possible to confidently determine which are which.

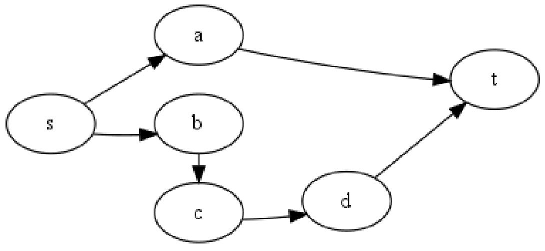
Throughout this problem, we assume that **more** than $n/2$ of the nodes are intact. Further, we let one "round" of pairings be any number of pairings overall as long as each node participates in at most one pairing. (I.e., a round is a matching that may not be perfect.)

ONLY PROBLEM IN THIS PART: Give an algorithm in clear, simple pseudocode that runs in $O(\lg n)$ rounds and identifies an intact node, and briefly but clearly justify the correctness of your algorithm via clear comments interspersed into the algorithm explaining what it does and why. (**HINT:** the quiz problems should lead you in a useful direction!)

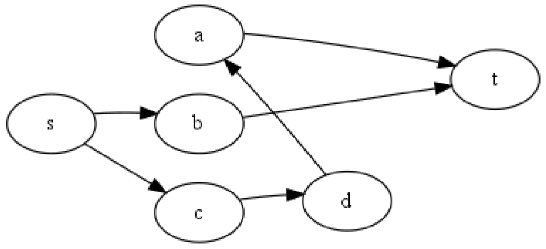
4 Mixed Nets

You're working on the routing for an anonymization service called a "mixnet" in which a network of computers pass messages through a sequence of handoffs from one source computer to eventually reach another target computer.

To represent this, you have a weakly-connected, directed acyclic graph (DAG) $G = (V, E)$ composed of designated source and target vertices $s, t \in V$ and a set of $p > 0$ simple paths (along which p messages pass) each of which starts at s, ends at t, and includes at least one vertex in between s and t. The paths are also vertex disjoint besides s and t (i.e., no two paths share any other vertex). There are no other vertices or edges in the graph. For example, here are two different graphs both over the same set of vertices and both with $p = 2$:



Graph #1



Graph #2

In fact, your mixnet actually involves a single set of computers (with a designated start and target computer) and two entirely separate sets of paths among those computers, as with these two sample graphs. At some point as each message passes along its path among the first set of paths, it switches to using one of the second set of paths instead (never switching back).

Specifically, you have an overall graph G made up of two subgraphs G_1 and G_2 like those specified in the previous part, where one subgraph's vertices is an exact copy of the other's, i.e., $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where a vertex $v_1 \in V_1$ if and only if $v_2 \in V_2$. (s_1 and t_1 are the start and target vertices in G_1 and their corresponding vertices s_2 and t_2 are the start and target in G_2.) Each subgraph is based on its own set of p paths, but p is the same for both. There is **also** a directed edge (v_1, v_2) for each vertex $v_1 \in V_1$ **except** s_1 and t_1 leading *from* G_1 *to* G_2. (There is no edge from s_1 to s_2 or t_1 to t_2.)

So, for the examples above, the overall graph would include both Graph 1 (with each node subscripted like a_1) and Graph 2 (with each node subscripted like a_2) plus 4 more edges: $(a_1, a_2), (b_1, b_2), (c_1, c_2), (d_1, d_2)$.

Your goal is to find p vertex-disjoint paths in the overall graph that start at s_1 in G_1 and end at t_2 in G_2. Thus, no two paths visit the exact same vertex (besides s_1 and t_2). **Additionally**, you want to ensure that no two paths even visit the same vertex in *different* subgraphs (i.e., no path contains v_1 if any other path contains v_2). We call two paths visiting the same vertex but in different graphs a "conflict".

1. In a valid instance of this problem, there is **always** a path from s_1 to t_2. Sketch the key points in a proof of this fact. (Recall that $p > 0$.)

2. In a valid instance of this problem, there is **never** any path from t_1 to t_2. Sketch the key points in a proof of this fact.

3. We now add the constraint to a valid instance of this problem that it must have **some** valid solution. Finish the following solution to this problem via reduction to STP (i.e., SMP with ties allowed).

   Let the first vertex along each of the paths out of s_1 be $v_{1,1}, v_{1,2}, \ldots, v_{1,p}$. Let the last vertex along each of the paths into t_2 be $u_{2,1}, u_{2,2}, \ldots, u_{2,p}$. Let the men $m_1, \ldots, m_p$ be the vertices $v_{1,1}, v_{1,2}, \ldots, v_{1,p}$. Let the women $w_1, \ldots, w_p$ be the vertices $u_{2,1}, u_{2,2}, \ldots, u_{2,p}$.

   Let man m_i prefer women w_j to woman w_k when there is a path from s_1 through $v_{1,i}$ to $u_{2,j}$ which transitions from G_1 to G_2 after a steps (and no such path that transitions after fewer than a steps), there is a path from s_1 through $v_{1,i}$ to $u_{2,k}$ which transitions from G_1 to G_2 after b steps (and no such path that transitions after fewer than b steps), and $\boxed{}$. (Let all women w_k for which there is no path from s_1 through $v_{1,i}$ to $u_{2,j}$ be tied at the end of m_i's preference list.)

Let woman w_i prefer man m_j to man m_k when... **COMPLETE THIS PART**. (Hint: think about an arrangement that in some sense mirrors the previous set of preferences.)

Solve the resulting STP instance to produce the perfect matching $S = \{(w_i, m_j), (w_k, m_l), \ldots\}$.

For each pair (w_i, m_j), use $\boxed{}$, always exploring edges that transition from G_1 to G_2 before edges that remain in G_1, to find (and add to the solution set of paths) the first path leading from s_1 to $v_{1,i}$ and then via some number of edges to $u_{2,j}$ and then to t_2.

4. Prove that since some valid solution exists, then some solution to the STP instance produced by the reduction exists which contains no instabilities (weak or strong).

   (This doesn't complete the proof that the reduction is correct, but it will present all the key insights.)

5 Cover Charge

In the "minimum edge cover" problem, the input is a simple, undirected, connected graph $G = (V, E)$ with $|V| \geq 2$, and the output is the smallest possible set E' such that $E' \subseteq E$ and for all vertices $v \in V$, there is an edge $\{v, u\}$ (which is the same as $\{u, v\}$) in E'. That is, every vertex in the graph is the endpoint of some edge in E'.

A maximum matching in a graph $G = (V, E)$ is the largest set E'' such that $E'' \subseteq E$ and there are no three vertices $v_1, v_2, v_3 \in V$ such that $\{v_1, v_2\}$ and $\{v_1, v_3\}$ are in E''. That is: E'' "marries off" as many vertices as possible without having any one vertex "married" to two or more vertices.

In this part, assume you have a connected graph $G = (V, E)$ and a maximum matching for the graph E''.

1. Prove that if $|E''| = \frac{|E|+1}{2}$, then E'' is a minimum edge cover.

2. Give an efficient, optimal greedy algorithm to find a minimum edge cover for $G = (V, E)$, given a maximum matching in the graph E'', and briefly but clearly justify the correctness of your algorithm via clear comments interspersed into the algorithm explaining what it does and why.

6 Bonus (OPTIONAL)

Worth 1 bonus point each:

1. Returning to the problem "Playing the Blame Game", give a short, extremely clear, and complete proof that if exactly half of the nodes are subverted (assuming an even number of nodes), no strategy whatsoever is guaranteed to find an intact node.

2. Returning to the problem "Cover Change", consider the following greedy algorithm for this problem:

```
Sort the vertices in increasing order by degree
Mark all vertices as uncovered
Let the cover E' be empty.
While there are vertices remaining, pick the next one v:
    If v is uncovered:
        If there is any neighbour u of v that is uncovered:
            Find the uncovered neighbour u of v with lowest degree
            Add {u, v} to E' and mark u and v as covered
        Else:
            Pick an arbitrary edge {u, v}, add it to E', and mark v as covered
```

Prove that this algorithm is not optimal by providing a **very** cleanly drawn and clearly explained counterexample. Your counterexample may not rely for its correctness (as a counterexample) on tie-breaking behaviour in the algorithm.