Midterm 1

Name: Targaryen

SID: 0123456789

Name and SID of student to your left: Lannister

Name and SID of student to your right: Stark

Circle One: **Pimentel** **Wheeler** **Wozniak**

Rules and Guidelines

- **The exam is out of 80 points and will last 80 minutes.**
- Answer all questions. Read them carefully first. Not all parts of a problem are weighted equally.
- Write your student ID number in the indicated area on each page.
- Be precise and concise. **Write in the solution box provided.** You may use the blank page on the back for scratch work, but it will not be graded. Box numerical final answers.
- Any algorithm covered in the lecture can be used as a blackbox.
- Throughout this exam (both in the questions and in your answers), we will use ω_n to denote the first n^{th} root of unity, i.e., $\omega_n = e^{2\pi i/n}$. So ω_{16} will denote the first 16^{th} root of unity, i.e., $\omega_{16} = e^{2\pi i/16}$.
- Good luck!

Discussion Section

Which of these do you consider to be your primary discussion section(s)? Feel free to choose multiple, or to select the last option if you do not attend a section. **Please color the checkbox completely. Do not just tick or cross the boxes.**

☐ Aarash, Tuesday 5 - 6 pm, Barrows 151

☐ Nick T., Tuesday 5 - 6 pm, Wheeler 202

☐ Chinmay, Wednesday 9 - 10 am, Dwinelle 215

☐ Chinmay, Wednesday 10 - 11 am, Moffitt 150D

☐ Simin, Wednesday 10 - 11 am, Mulford 240

☐ Aditya M., Wednesday 11 am - 12 pm, Barrows 56

☐ Nick W., Wednesday 11 am - 12 pm, Giannini 141

☐ Yuxiang, Wednesday 1 - 2 pm, Dwinelle 215

☐ Nikhil, Wednesday 1 - 2 pm, Wheeler 108

☐ James, Wednesday 1 - 2 pm, Soda 405

☐ Aditya B., Wednesday 2 - 3 pm, Wheeler 200

☐ Owen, Wednesday 2 - 3 pm, Etcheverry 3105

☐ James, Wednesday 2 - 3 pm, Soda 310

☐ Aditya B., Wednesday 3 - 4 pm, Wheeler 202

☐ Harley, Wednesday 3 - 4 pm, Wheeler 220

☐ Michael, Wednesday 3 - 4 pm, Soda 310

☐ Vinay, Wednesday 4 - 5 pm, Dwinelle 223

☐ Benjamin, Wednesday 5 - 6 pm, GPB 107

☐ Mudit, Wednesday 5 - 6 pm, Moffitt 150D

☐ I do not attend a discussion section

1 Multiple Choice [1 point per problem]

Fill in a single circle for each problem. Fill it in completely.

(a) Suppose we run DFS on the complete graph K_n (ie, the graph on n vertices where all $\binom{n}{2}$ edges are present). What is the depth of the resulting DFS tree? That is, what is the length of the longest root to leaf path?

- ☐ 1
- ☐ $\lceil \frac{n}{2} \rceil$
- ☐ $n-1$
- ☐ n

Solution: $n-1$

(b) Suppose that in the previous problem, we ran BFS instead of DFS. What is the depth of the resulting BFS tree?

- ☐ 1
- ☐ $\lceil \frac{n}{2} \rceil$
- ☐ $n-1$
- ☐ n

Solution: 1

(c) $T(n) = T(n-1) + \Theta(f(n))$ runs in $\Theta(n^2)$ time for what value of $f(n)$?

- ☐ 1
- ☐ $\log n$
- ☐ n
- ☐ n^2

Solution: n

(d) $T(n) = 7T(n/2) + \sqrt{n}$. What is $T(n)$?

- ☐ $O(\sqrt{n})$
- ☐ $O(n^2)$
- ☐ $O(n^{log_2 7})$
- ☐ $O(n^{log_7 2})$

Solution: $n^{\log_2 7}$

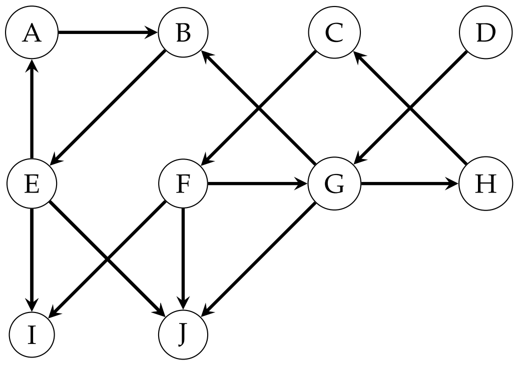
(e) Let ω be one of the fourth roots of unity. What is ω^{-4}?

- ☐ 1
- ☐ i
- ☐ Depends on the root of unity

Solution: 1

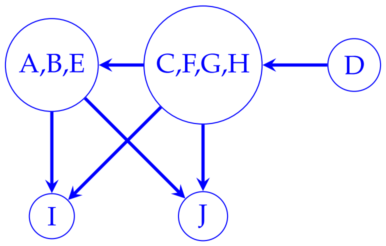
2 Strongly Connected Components [5 points per part]

Use the following directed graph to answer parts (*a*) and (*b*) of this question.:



(a) Find the strongly connected components of the above graph and use these components to draw its corresponding "meta-graph" (you don't need to use the SCC-algorithm).

Solution:



Solution:

(b) Give a topological sorting of the meta-graph you drew in part (*a*).

Solution:

Solution:

$$[\{D\},\{C,F,G,H\},\{A,B,E\},\{I\},\{J\}]$$

or

$$[\{D\},\{C,F,G,H\},\{A,B,E\},\{J\},\{I\}]$$

3 Ancestor Queries [5 points]

You are given a tree $T = (V, E)$ (in adjacency list format), along with a designated root node $r \in V$. Recall that u is said to be an ancestor of v in the rooted tree, if the path from r to v in T passes through u (note that r is an ancestor of v for every $v \in V$). You wish to preprocess the tree so that queries of the form is u an ancestor of v? can be answered in constant time. The preprocessing itself should take linear time. How can this be done?

You only have to provide a main idea.

Solution:

Solution: Run a Depth First Search on the graph and note a pre-visit time w_{pre} and post-visit time w_{post} to each vertex $w \in V$ (linear time algorithm). Then it suffices to check that $u_{\text{pre}} \le v_{\text{pre}} < v_{\text{post}} \le u_{\text{post}}$.

4 Cubed Fourier [10 points]

(a) Cubing the 9^{th} roots of unity gives the 3^{rd} roots of unity. Next to each of the third roots below, write down the corresponding 9^{th} roots which cube to it. The first has been filled for you. *We will use ω_9 to represent the primitive 9^{th} root of unity, and ω_3 to represent the primitive 3^{rd} root.*

$\omega_3^0 : \omega_9^0$, ,

$\omega_3^1 :$, ,

$\omega_3^2 :$, ,

$\omega_3^0 : \omega_9^0, \omega_9^3, \omega_9^6$

$\omega_3^1 : \omega_9^1, \omega_9^4, \omega_9^7$

$\omega_3^2 : \omega_9^2, \omega_9^5, \omega_9^8$

(b) You want to find the Fourier transform of a degree-8 polynomial, but you don't like having to pad it with 0s to make the (degree+1) a power of 2. Instead, you realize that 9 is a power of 3, and you decide to work directly with 9th roots of unity and use the fact proven in part (a). Say that your polynomial looks like $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_8x^8$. **How do you split $P(x)$ to use the fact proven in part (a) to your advantage?** Provide either the polynomial, or explain how the vector can be divided to recurse on. *Recall that for the FFT algorithm shown in the book, we split a given polynomial $Q(x) = A_e(x^2) + xA_o(x^2)$, and we define what $A_e(x^2)$ and $A_o(x^2)$ are. Correspondingly, in lecture you saw the $\vec{a}$ split into $\vec{a}_{even}$ and $\vec{a}_{odd}$.*

Solution:

Let $P(x) = P_1(x^3) + xP_2(x^3) + x^2P_3(x^3)$
where $P_1(x^3) = a_0 + a_3x^3 + a_6x^6$.
and $P_2(x^3) = a_1 + a_4x^3 + a_7x^6$.
and $P_3(x^3) = a_2 + a_5x^3 + a_8x^6$.

(c) If we were to recurse (on a polynomial of degree n-1) according to part (b), what would the recurrence relation of the algorithm be? In $O(\cdot)$ notation, how long would it take to compute the Fourier transform?

Solution:

The recurrence relation would be $T(n) = 3T(n/3) + O(n)$. The solution to this recurrence relation is still $O(n \log n)$

5 A Network of Roads [20 points]

There is a network of undirected roads $G = (V, E)$ connecting a set of cities V. Each road $e \in E$ has an associated length l_e. There is a proposal to add one new road to this network, and there is a list F of candidate pairs of cities between which the new road can be built. Each such potential road $e' \in F$ has an associated length $l'(e')$. As a designer for the public works department you are asked to determine the road $e' \in F$ whose addition to the existing network G would result in the maximum decrease in the driving distance between two fixed cities s and t in the network. If none of the roads offer an improvement, you will decide to not add any of them. Since the set F of candidate roads can be quite large, running Dijkstra $|F|$ times is way too slow, and your challenge is to design a more clever algorithm.

Provide a 4 part solution (main idea, pseudocode, proof of correctness, runtime).

Main Idea:

Pseudocode:

Proof of Correctness: (A few sentences should suffice)

Runtime:

Solution: There were two main solutions to this problem, with some variations.

Solution 1

The first solution is to create 2 copies of the graph G, say G_A and G_B. First, convert all edges in both of these graphs to be bi-directed from undirected. Next, for each edge $e : (u, v) \in F$: add an edge from u in G_A to v in G_B. Also, add an edge from v in G_A to u in G_B. Now run Dijkstra's starting from s in G_A, and find the distance to t in G_B. If this distance is longer than the original distance from s to t (checked using another run of Dijkstra's), then return None.

The correctness for this algorithm in guaranteed by the fact that we end in G_B, and the only way to get to G_B is by using one and only one edge in F.

The runtime is going to be that of a single run of Dijkstra's on a graph with $2|V|$ vertices, and $2|E| + |F|$ edges, which is $O((2|V| + 2|E| + |F|) \log 2|V|)$.

Solution 2

The second solution is to run Dijkstra's starting from s, and once again starting from t. This computes shortest distances from s to every other vertex, and from t to every other vertex. Store these distances in an array. Then, for every $e : \{u, v\} \in F$, compute $d(s, u) + \ell(e) + d(v, t)$ and $d(s,)v + \ell(e) + d(u, t)$ find the minimum of these, and return the corresponding edge e. If $d(s, t)$ is shorter, then return None.

The correctness of this algorithm relies on the fact that the shortest path from s to t using $e = \{u, v\}$ is the minimum of $d(s, u) + \ell(e) + d(v, t)$ and $d(s,)v + \ell(e) + d(u, t)$, and running Dijkstra's from s and t guarantees optimal $d(s, u)$ and optimal $d(t, v)$ for every u, v.

The runtime of this algorithm is two runs of Dijkstra's on the original graph, followed by $|F|$ computations to add the distances, since querying $d(s, u)$ and $d(s, t)$ is constant time. This takes $O((|V| + |E|) \log V + |F|)$ time.

Common mistakes

- Note that for the first solution, the edges going across the edges must be directed. This is because otherwise, the algorithm could end up using multiple roads from F. It is possible that the shortest path jumps around between the 2 graphs.
- Note that for the first solution, it must be clear that we are starting in G_A and ending in G_B.
- Note that for the second solution, running Dijkstra's from s alone is not enough, since that does not compute $d(v, t)$.
- Note that for the second solution, running Dijkstra's from t alone is also not enough. Some students argued that the path from s to u must be the same if the path improves by adding an edge from F. This is not necessarily true. It is possible that the path from v to t is instead the same, for instance.

Common incorrect solutions

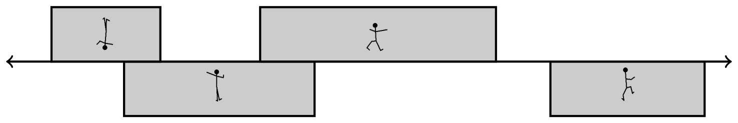
- Note that in the problem statement, we explicitly asked to do better than running Dijkstra's $|F|$ times. Submissions that called Dijkstra's F times received no credit.
- Additionally, an alternate solution is to make F copies of the graph instead of just 2. This results in a runtime of $O(|F|(|V| + |E|)log(|F||V|)$, which is even worse than running Dijkstra's $|F|$ times, and so such a solution also received no credit.
- Many students also argued that all the edges in F can be added to the graph, and then as soon as Dijkstra's uses the first edge in F, remove all other edges from F in the graph. This modification does not work, because it is not necessary that the first edge countered from F is the correct edge to use.

6 Placing Bus Stations [15 points]

A city is interested in designing a new bus line that runs on a straight road. However, they are unsure of where to place the bus stops. The city surveyed bus users along the route for (a) where they live and (b) how far they would be willing to walk to a bus station. Design a greedy algorithm that minimizes the number of bus stops required to cover all surveyed bus users. Give an exchange argument to prove that your algorithm is correct.

Assume the survey included n users with user i living at x_i and willing to walk a distance d_i (in either direction). You may assume that all elementary arithmetic operations take unit time.

For example, here is a picture of 4 users and the ranges they are willing to walk for a bus.



Main Idea:

Proof of Correctness:

Solution:

For each user, there is a strict range $[\ell_i = x_i - d_i, r_i = x_i + d_i]$ of the x axis in which a stop must be put to satisfy user i.

Now, if we sort all users by r in increasing order, and begin looking at the first user. Where should we choose z_1 for the first stop? In order to cover the first user, we need $z_1 \leq r_1$. However, placing it at any $z_1 < r_1$ wastes resource, as it covers no more users than placing it at $z_1 = r_1$.

Now that the first stop is placed, it covers some users. Where should we place the next router? We can simply repeat the above process and find among the uncovered users the one with the smallest r coordinate. The process goes on until we cover all the users.

Exchange argument: consider an optimal solution $y_1, y_2, ..., y_k$ which maximizes j such that the first j stops are the same as in greedy. i.e. $y_i = z_i$ for $1 \leq i \leq j$. Let r_k be the right side of the next (i.e., leftmost) range not covered by $y_1, \ldots, y_j$. By our greedy algorithm $z_{j+1} = r_k$. If $y_{j+1} > z_{j+1}$, then the k^{th} user isn't covered in the optimal solution (a contradiction). So $y_{j+1} \leq z_{j+1}$. Moving y_{j+1} right so it equals z_{j+1} cannot uncover some previously covered interval, since it would have to move right of some interval not covered by $y_1, \ldots, y_j$ (contradicting the definition of r_k). Therefore, we can replace y_{j+1} with z_{j+1}. By induction, we can replace each y_i with z_i without uncovering any interval, so $z_1, \ldots, z_k$ is an optimal solution.

The runtime is $O(n \log n)$ as it takes $O(n)$ time to compute the ℓ and r values, $O(n \log n)$ to sort with respect to r, and then $O(n)$ to perform a linear sweep assigning stops.

7 Interweaving an array in place [15 points]

You are given an array of length $2n$, where n is a power of 2, and you wish to apply the transformation

$$[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n] \mapsto [x_1, y_1, x_2, y_2, \ldots, x_n, y_n].$$

Provide a 3 part solution (main idea, pseudocode, runtime analysis) for an algorithm that applies the transformation **in place** (using at most $O(1)$ additional memory). Note that this effectively means that you are only allowed to swap elements within the array.

Main Idea:

Pseudocode:

Runtime analysis:

Solution:

Apply a divide and conquer approach. First apply the following transformation using $n/2$ swaps.

$$[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n] \mapsto [x_1, \ldots, x_{n/2}, y_1, \ldots, y_{n/2}, x_{n/2+1}, \ldots, x_n, y_{n/2+1}, \ldots, y_n]$$

Then recurse on the two half sized arrays. This satisifes reccurance relation $T(n) = 2T(n/2) + O(n)$ which solves to $T(n) = \Theta(nlogn)$.

Blank page: If using this page to write an answer, clearly specify on the question that your work should be found here.

Blank page: If using this page to write an answer, clearly specify on the question that your work should be found here.