Midterm 1 Solution

Name: Targaryen

SID: 0123456789

Name of student to your left: Lannister

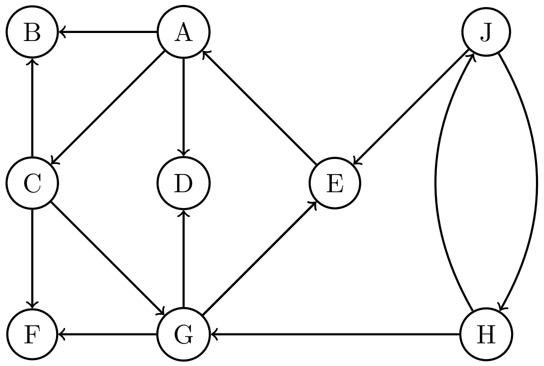
Name of student to your right: Stark

GSI and section time: Westeros 300 AC

Rules and Guidelines

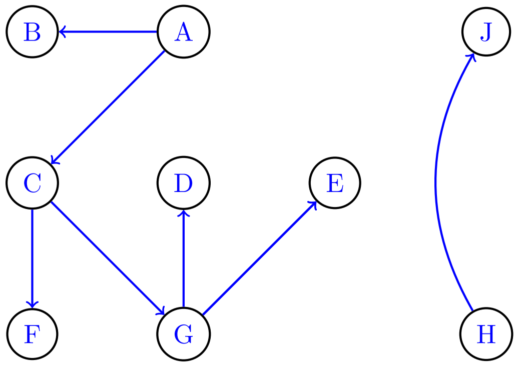
- Answer all questions. Read them carefully first. Not all parts of a problem are weighted equally.
- Write your student ID number in the indicated area on each page.
- Be precise and concise.**Write in the solution box provided.** You may use the blank page on the back for scratch work, but it will 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!

1. (**6 points**) Execute a DFS on the graph shown below starting at node A and breaking ties alphabetically. Draw the DFS tree/forest. Mark the pre and post values of the nodes with numbering starting from 1.



Node	pre	post
A		
B		
C		
D		
E		
F		
G		
H		
J		

Draw the DFS Tree/Forest in the box below:

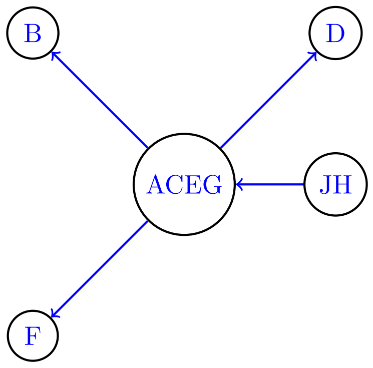


Node	pre	post
A	1	14
B	2	3
C	4	13
D	8	9
E	10	11
F	5	6
G	7	12
H	15	18
J	16	17

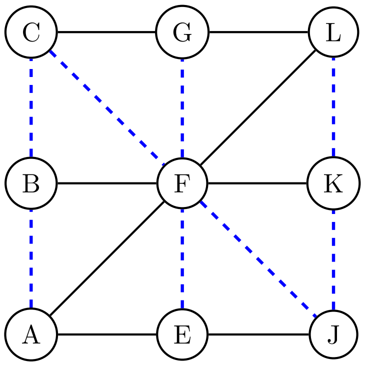
2. (**4 points**) In the DFS execution from above, mark the following edges as as **T** for Tree, **F** for Forward, **B** for Back and **C** for Cross.

Edge	Type
$C \to G$	T
$J \to E$	C
$G \to F$	C
$E \to A$	B

3. (**4 points**) Draw the DAG of strongly connected components for the above graph.



4. Suppose the dashed edges represent the unique MST in the graph below.



(a) (**4 points**) List all edges necessarily heavier than the edge BC.

BF, AF, AE, all the other edges across the cut

(b) (**4 points**) List all edges that are necessarily lighter than the edge GL.

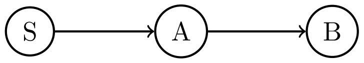
LK, KJ, JF, FG, all the other edges along the cycle

5. Recall the update operation on distances used in the Bellman-Ford algorithm:

$$\begin{aligned}&\textit{update}\,(\,\textit{edge}\,(\,u \to v\,)\,)\\&\qquad \textit{dist}[v] = \min\,(\,\textit{dist}[v],\ \textit{dist}[u] + \ell(u,v)\,)\end{aligned}$$

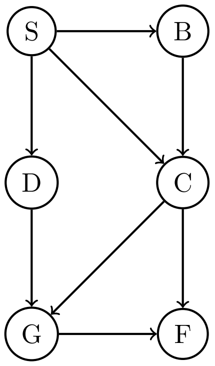
In the following graphs, find the shortest sequence of *update* operations that will ensure that all the distance values from S are correctly computed, irrespective of the weights on the edges. (For simplicity, assume that all edge weights are positive) We have solved the first one as an example.

(a) (Example)



Answer: $S \to A,\ A \to B$

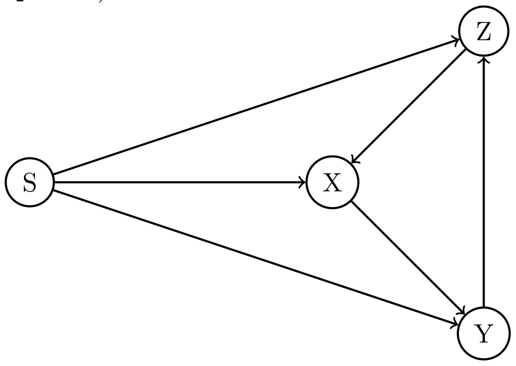
(b) (**6 points**)



One valid sequence: $S \to B, S \to C, S \to D, B \to C, C \to G, C \to F, D \to G, G \to F$, exactly once for all 8 edges.

All patterns should match: $S \to B \to C \to G \to F$, $S \to D \to G$ has to be before $G \to F$.

(c) (**8 points**)



One valid sequence: $S \to X, S \to Z, S \to Y, X \to Y, Y \to Z, Z \to X, X \to Y$, length 7, one of the edges in the cycle XYZ must be updated twice.

All patterns should match one of:

$S \to X \to Y \to Z \to X$,

$S \to Z \to X \to Y \to Z$,

$S \to Y \to Z \to X \to Y$.

6. Solve the following recurrence relations:

(a) (**6 points**)

$$T(n) = 8T(n/2) + O(n^3)$$

$T(n) \in T(n) \in O(n^3 \log n)$

Show work:

We find that $a = 8$, $b = 2$, and $d = 3$. Using Master Theorem, we find that $d = \log_b a$ so that results in $O(n^3 \log n)$

(b) (**9 points**)

$$T(n) = 2T(\sqrt{n}) + 1$$

$T(n) \in T(n) \in O(\log n)$ or $T(n) \in O(2^{\log \log n})$

Show work:

Expanding out the recursion for a few steps, we see that the recursion only takes $\log \log n$ steps.

More detailed explanation: Below is a tree where each node is labelled with the size of the subproblem at that node. The work done at each node is constant, so the total work is equal to the number nodes.

$$\begin{tikzcd}
\text{Level 0} & & & & n \arrow[dll] \arrow[drr] & & & \\
\text{Level 1} & & n^{1/2} \arrow[dl] \arrow[dr] & & & & n^{1/2} \arrow[dl] \arrow[dr] & \\
\text{Level 2} & n^{1/4} & & n^{1/4} & & n^{1/4} & & n^{1/4} \\
 & \vdots & & \vdots & & \vdots & & \vdots \\
 & 2 & \cdots & 2 & \cdots & 2 & \cdots & 2
\end{tikzcd}$$

In general, at level k, we have 2^k problems of size $n^{1/2^k}$. The recursion stops when the size drops to a low enough value, at which point we have done $1+2+4+8+16+...+2^k = 2(2^k)-1$ work. We just need to solve for k to find the total work. If we say the recursion stops at when the size becomes 2, we need to solve $n^{1/2^k} = 2$.

$$n^{1/2^k} = 2$$
$$n = 2^{2k}$$
$$\log\log n = k$$

Therefore the total work done is $2(2^k) - 1 = 2(2^{\log\log n}) - 1 = O(2^{\log\log n}) = O(\log n)$

7. (a) (**5 points**) Suppose we want to multiply the following two polynomials using FFT:

$$A(x) = 3 - x + 4x^2 + 2x^3$$
$$B(x) = -2x^2 + x^3$$

What values would we evaluate the polynomials at to transform them?

8^{th} roots of unity

(b) (**5 points**) Suppose the FFT of a polynomial $A(x)$ of degree less than 3, at the 4-th roots of unity yields the vector $(1, 1, 1, 1)$. What is $A(x)$?

$A(x) = 1$. There is a unique degree 3 polynomial and $A(x) = 1$ satisfies the conditions.

8. (**5 points**) During the execution of BFS in an unweighted undirected graph G starting from a node S, two vertices A and B are both present in the queue at some point in time. What are the possible values of $dist[S, A] - dist[S, B]$?

-1, 0, 1
Justification: Two nodes presented at the same time in the queue must either be from the same layer or consecutive layers of BFS. Since $dist[S, U]$ is simply the layer that U is in the BFS tree, we can conclude that $dist[S, A] - dist[S, B]$ must be either 0 (if they are from the same layer), 1 (if A is one layer above B) or -1 (if B is one layer above A).

9. (**10 points**) Mark the following statements as True (**T**) or False (**F**). Scoring for this part is +2 for correct, 0 for blank, and **-2 for incorrect**.

(a) Suppose the edge weights are positive and distinct, if an edge e is not the lightest edge across a cut $(S, \overline{S})$, it cannot be part of any MST.

False
Justification: Consider a graph $G = (V, E)$ with three vertices A, B, C. If there are only two edges $\{A, B\}$ and $\{B, C\}$ of weights one and two respectively. Then, $\{B, C\}$ is not the lightest edge across the cut $(\{A\}, \{B, C\})$ but is contained in the MST.

(b) Suppose the edge weights are positive and distinct, if an edge e is the lightest edge across a cut $(S, \overline{S})$, it is part of some MST.

True
Justification: This follows from the Cut Property.

(c) Suppose the edge weights are positive and distinct, every edge in an MST is the lightest edge across some cut in the graph.

True
Justification: Consider an MST T and any edge e in T. Removing e must disconnects T into two components S and $V - S$. e must be the lightest edge across the cut $(S, V - S)$; if not, then adding the lightest edge across the cut and removing e would have resulted in a (strictly) lighter spanning tree, which would contradict the fact that T is a MST.

(d) Suppose the edge weights are positive and distinct, and e is the lightest edge incident at a vertex v, then e is part of every MST.

True
Justification: Consider an MST T. If e is not in T, then $T \cup \{e\}$ must have a cycle containing

e. This cycle must contain another edge e' incident to v. Since e is strictly lighter than e', $T - \{e'\} \cup \{e\}$ would be a strictly lighter spanning tree, which contradicts to the assumption that T is a MST.

(e) Suppose the edge weights are positive and distinct, and e is the lightest edge incident at a vertex v then e is part of the shortest path tree rooted at v.

True
Justification: Suppose that this edge is $\{v, u\}$. Since the edge weights are positive and all other edges incident to v has weights more than this edge, the shortest path from v to u must be this edge.

SID:

10. (**8 points**) Shown below are the pre- post values of vertices A, B, C, D, E in a DFS traversal of a graph G (there are other nodes in G that are not shown).

Node	Pre-value	Post value
A	100	200
B	180	240
C	160	300
D	120	140
E	110	150

(a) One of the vertices has incorrect pre/post values, can you guess which one?

A

Briefly, justify your answer.

Justification: Recall that in the correct DFS post- and pre- values, for every pairs of vertices u and v, either (1) one of the intervals $[\text{pre}(u), \text{post}(u)]$ or $[\text{pre}(v), \text{post}(v)]$ is contained in the other one, or (2) the two intervals do not intersect.

In the give values, $[\text{pre}(A), \text{post}(A)]$ insect with $[\text{pre}(B), \text{post}(B)]$ and $[\text{pre}(C), \text{post}(C)]$ without any containment. Thus, A must be the vertex with incorrect pre- or post- values.

(b) Ignore the vertex with the incorrect pre/post values. List all pairs among the remaining that cannot be edges in the graph G.

$E \to B$, $E \to C$, $D \to B$, $D \to C$

Justification: There can be no edges from u to v if $\text{post}(u) < \text{pre}(v)$; if there were an edge from u to v, then v must have been visited before explore(u) finishes, meaning that $\text{post}(v) \leq \text{pre}(u)$.

(c) Ignore the vertex with the incorrect pre/post values. Suppose G is a DAG, which additional pairs cannot be edges in G (apart from those already listed in the previous question)?

$D \to E$, $B \to C$

Justification: If $[\text{pre}(u), \text{post}(u)]$ contains $[\text{pre}(v), \text{post}(v)]$, then v is reachable from u. This implies that D is reachable from E and B is reachable from C. Since this graph is a DAG, there must be no edge from D to E and from B to C.

11. (**20 points**) Range Sum
Given integer array $A[1..n]$, design a $O(n \log n)$ algorithm to find the number of ranges $[a, b]$ such that summing the values of that range gives zero, i.e.,

$$\sum_{i=a}^{b} A[i] = 0$$

Describe the main idea behind the algorithm in a few sentences. (No need for proof of correctness or runtime analysis)

Algorithm 1: $\Theta(n)$

We calculate the partial sums for each index.
Partial sum for index j is defined as the cumulative sum from index i.

$$\sum_{i=1}^{j} A[i]$$

Invariant: If two indicies have the same partial sum, then the sum of values between the two indicies is 0.

$$\sum_{k=1}^{j} A[k] - \sum_{k=1}^{i} A[k] = \sum_{k=i+1}^{j-1} A[k] \quad i, j \in [1, n], i < j$$

We use a Hash Map to map partial sums to the number of indicies that have that partial sum. If we have as least 2 indicies that map to the same partial sum, let x be the number of indicies that has the same partial sum s. We add x choose 2 to our answer. Do this for each key value pair. Note that combinatorics is linear time in general. Since we are always choosing by 2, x choose 2 becomes $\frac{x(x-1)}{2}$. Another way us constant time is to do this in the main loop.

Our Hash Map will now map partial sums to pair of number of indicies and ongoing count. When we increment the number of indicies, we add the old number to ongoing count. After the first iteration, our answer will be the sum of each partial sum's ongoing count.

See Pseudocode for both versions below.

```
procedure RangeSumCount($A[1..n]$)
    if $A$ is empty then return 0
    partialSums ← HashMap with default value 0
    partialSums[0] ← 1                                ▷ partial sum to beginning is 0
    $s \leftarrow 0$
    for $i \in \{1..n\}$ do
        $s \leftarrow s + A[i]$
        partialSums[$s$] ← partialSums[$s$] + 1
    cnt ← 0
    for $v \in$ values(partialSums) do
        cnt ← cnt + $\frac{v \cdot (v-1)}{2}$
    return cnt
procedure RangeSumCountCombinatorics($A[1..n]$)
    if $A$ is empty then return 0
```

```
partialSums ← HashMap with default value $[0, 0]$
partialSums$[0]$ ← $[1, 0]$                                  ▷ partial sum to beginning is 0
$s \leftarrow 0$
for $i \in \{1..n\}$ do
    $s \leftarrow s + A[i]$
    if partialSums$[s][0] \geq 1$ then
        partialSums$[s][1]$ ← partialSums$[s][1]$ + partialSums$[s][0]$
    partialSums$[s][0]$ ← partialSums$[s][0] + 1$
cnt ← 0
for $v \in$ VALUES(partialSums) do
    cnt ← cnt $+ v[1]$
return cnt
```

Algorithm 2: Sorting, $\Theta(n \log n)$

We can first sort the partial sums. Partial sums will now be next to each other. For each contiguous chunk of the same value, let the number of values in that chunk to be x. This represents the number of indices with the same partial sum. Here we can perform the same x choose 2 operation as before in constant time. Adding these all up gives us our answer

Algorithm 3: Divide and Conquer, $\Theta(n \log n)$

We will still want to utilize the fact that indicies with the same partial sum means that sum of values of in-between indices is 0. Also we want to utilize sorting. It will be modification of merge sort. Our function will return the count of ranges that sum to 0 and the sorted partial sums for a range of indicies.

- First compute list of partial sums globally.
- In our count function.
  - Recursively call on left half and right half of partial sums. Let ongoing count be the sum of the counts from the recursive calls. The two halves are sorted by partial sums.
  - Merge the left half and right half partial sums in sorted order.
  - We want to find the values on the left that are equal to the values on the right.
  - When we see partial sums that are equal in the merge comparison, we find the chunks on both halves with that partial sum, and add number of value on left with that partial sum multiplied by number of values on right to ongoing count. This can be done in linear time with 4 indicies as the same values will be next to each other.
  - Return final ongoing count and sorted partial sums.

Note: FFT is incorrect. It does not guarantee continuous summation. Also it is inefficient as the largest range is n and integers are unbounded.

12. (**25 points**) The transportation network of Trigonoland consists of n cities (denoted by vertices V) and a network of bus routes represented by edges E_B, train routes represented by edges E_T and airplane routes represented by edges E_A between the cities.

For every edge $e \in E_B \cup E_T \cup E_A$, let $t(e)$ denote the length of the edge.

Devise an efficient algorithm to compute the length of the quickest route from a city s to a city t that never uses the same mode of transport in two consecutive edges along the route.

For example, if the path takes a train from city i to city j, then it can't take a train out of city j.

(a) **Main Idea**

Make three graphs that each have their own copies of V; call these graphs G_B, G_A, and G_T and denote G_i's version of the vertex u to be u_i.

For each edge (u, v) in E_A, make a copy between the u_B to v_A and between u_T and v_A. For each edge (u, v) in E_T, make a copy between the u_B to v_T and between u_A and v_T. For each edge (u, v) in E_B, make a copy between the u_A to v_B and between u_T and v_B.

Create a dummy node s_D with edges of weight 0 to s_A, s_B, and s_T. Also create a dummy node t_D with edges of weight 0 to t_A, t_B, and t_T. Run Dijkstra's Algorithm on this newly constructed graph starting at s_D, and return $dist(t_D)$. Intuitively, (WLOG) arriving at a node v_B in G_B means that you took a bus to get to get to city v, so you need to leave using either a train or a plane; this construction of the graph captures that information.

(b) **Runtime of algorithm** = $\Theta((E + V)\log(V))$

(c) **Proof of Correctness**

We construct a new graph made up of three different sub-graphs, each representing the mode of transportation taken to get to a certain point. No vertex in a sub-graph has an edge to a vertex in the same sub-graph, so any path taken must hop between different sub-graphs. Thus our path will not use the same mode of transportation twice or more in a row.

When constructing our three sub-graphs, every edge we add will have the same weight as its respective edge in Trigonoland. Therefore any path in our new graph will have the same cost as the path in Trigonoland visiting the same vertices in the same order and using the modes of transportation corresponding to the edges in our graph.

Thus this problem can be solved by simply finding the shortest path from any one of s_A, s_B, or s_T to any one of t_A, t_B, or t_T. We add the dummy nodes so we only have to run Dijkstra's once.