Final

Name: Targaryen

SID: 0123456789

Name and SID of student to your left: Lannister

Name and SID of student to your right: Stark

Exam Room:

☐ RSF Fieldhouse ☐ Soda 405 ☐ Wozniak Lounge ☐ Other

Please color the checkbox completely. Do not just tick or cross the box.

Rules and Guidelines

- **The exam is out of 194 points and will last 170 minutes.** Roughly, one should expect to spend just less than a minute for a point.
- 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.
- The problems may **not** necessarily follow the order of increasing difficulty. ***Avoid getting stuck on a problem.***
- Any algorithm covered in lecture can be used as a blackbox. Algorithms from homework need to be accompanied by a proof or justification as specified in the problem.
- 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.**

☐ Arpita, Thursday 9 - 10 am, Dwinelle 223

☐ Avni, Thursday 10 - 11 am, Dwinelle 215

☐ Emaan, Thursday 10 am - 11 pm, Etcheverry 3107

☐ Lynn, Thursday 11 am - 12 pm, Barrows 118

☐ Dee, Thursday 11 am - 12 pm, Wheeler 30

☐ Max, Thursday 12 - 1 pm, Wheeler 220

☐ Sean, Thursday 1 - 2 pm, Etcheverry 3105

☐ Neha, Thursday 2 - 3 pm, Dwinelle 242

☐ Julia, Thursday 2 - 3 pm, Etcheverry 3105

☐ Henry, Thursday 3 - 4 pm, Haviland 12

☐ Kedar, Thursday 3 - 4 pm, Barrows 104

☐ Ajay, Thursday 4-5 pm, Barrows 136

☐ Varun, Thursday 4-5 pm, Dwinelle 242

☐ Gillian, Friday 9 - 10 am, Dwinelle 79

☐ Hermish, Friday 10 - 11 am, Evans 9

☐ Vishnu, Friday 11 am - 12 pm, Wheeler 222

☐ Carlo, Friday 11 am - 12 pm, Dwinelle 109

☐ Tarun, Friday 12 - 1 pm, Hildebrand B56

☐ Noah, Friday 12 - 1 pm, Hildebrand B51

☐ Jiazheng, Friday 1 - 2 pm, Wheeler 30

☐ Claire, Friday 2 - 3 pm, Barrows 155

☐ Jialin, Friday 1 - 2 pm, Wheeler 30

☐ Teddy, Friday 1 - 2 pm, Dwinelle 105

☐ Jierui, Friday 2 - 3 pm, Wheeler 202

☐ David, Friday 2 - 3 pm, Wheeler 130

☐ Nate, Friday 2 - 3 pm, Evans 9

☐ Rishi, Friday 3 - 4 pm, Dwinelle 243

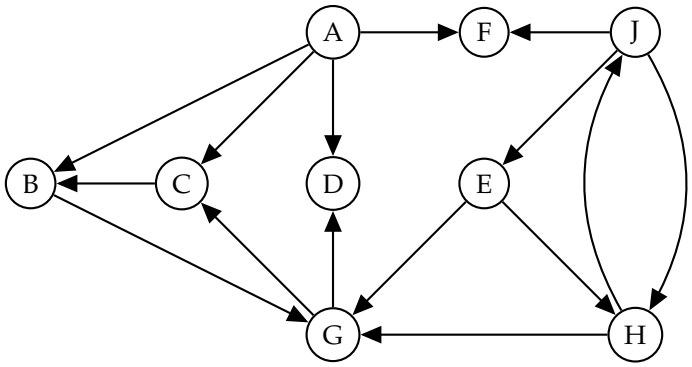
☐ Tiffany, Friday 3 - 4 pm, LeConte 385

☐ Ida, Friday 3-4 pm, Dwinelle 109

☐ Don't attend section.

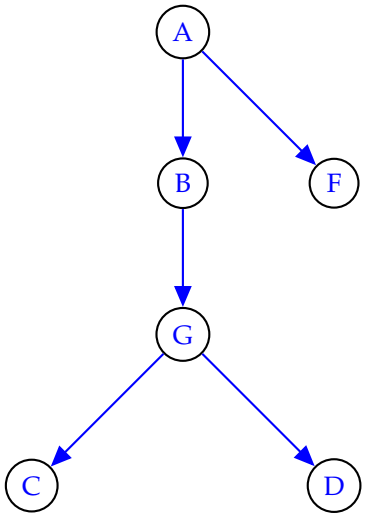
1 SCCs (10 points)

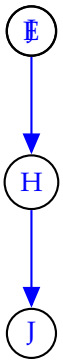
Execute a DFS on the directed graph shown below starting at node A and breaking ties alphabetically.



1. Fill in the table of pre and post values.

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





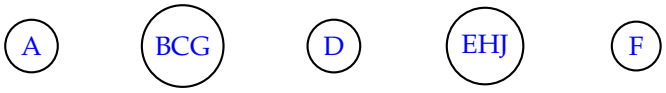
Node	pre	post
A	1	12
B	2	9
C	4	5
D	6	7
E	13	18
F	10	11
G	3	8
H	14	17
J	15	16

2. 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
$E \to G$	C
$C \to B$	B
$A \to D$	F
$H \to J$	T

3. List the strongly connected components of the above graph in a linearized order. (Note that the number of SCCs is possibly smaller than the number of rows provided in the table below)

SCC no.	Vertices in the component
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	



2 Runtime Analysis (5 points)

Consider the following snippet of code.

```
Function what(n) {
      what(√n)
      what(√n)
      for k = 1 to n {
            print BLAH
      }
}
```

Write a recurrence relation for the runtime.

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

Write the tightest upper bound for runtime.

$T(n) = O(n)$

3 Linear-Time Selection for the Median (5 points)

Consider the execution of the linear-time selection algorithm to find the median element of the following array:

1	3	5	7	9	2	4	6	8

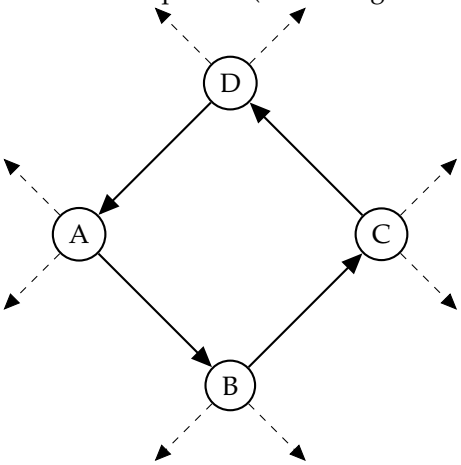
1. What is the best possible value of the first pivot element (one that would make the algorithm terminate the fastest)? 5
2. Give an example of the worst possible sequence of choices of pivot elements for finding median using the algorithm (one that would make the algorithm terminate the slowest)? $1, 2, 3, 4, 9, 8, 7, 6, 5$
3. What is best possible value of the first pivot if the input was the following array?

9	8	7	6	5	4	3	2	1

5

4 Post Values (4 points)

A **part of a directed graph** is shown below. There are other vertices and edges in the graph that are not shown in the picture (but all edges **between** vertices A, B, C, D are shown).



Consider a DFS run on the entire graph. Indicate whether each of the following subsets of post-values is possible or impossible.

Vertex v	$post[v]$
A	22
B	100
C	64
D	36

Table 1

○ possible
○ impossible

Vertex v	$post[v]$
A	22
B	36
C	64
D	100

Table 2

○ possible
○ impossible

Solution:

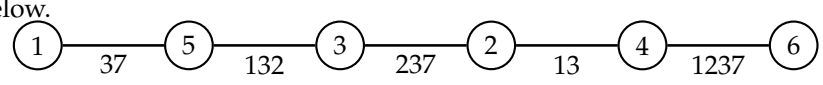
The text of the question suggests that there are other vertices in the graph with incoming and outgoing edges to this cycle. Both Table 1 and Table 2 are possible, as per the text of the question.

However, unfortunately, the picture is a little misleading in that all the arrows seems to be pointed away from the cycle, with no dashed edges shown coming in. With this interpretation, Table 1 is possible, but Table 2 is impossible.

For grading, we have given full credit to "possible" for table 1 and either possible/impossible for table 2.

5 Floyd-Warshall algorithm (4 points)

Consider the following undirected graph G on vertices numbered $\{1, 2, 3, 4, 5, 6\}$, with edge lengths shown below.



Consider the execution of the Floyd-Warshall algorithm for all-pairs shortest paths on graph G. The dis-

tance values between pairs of vertices converge to their true value as the algorithm executes. Order the following distance values from the earliest to converge, to the last to converge.

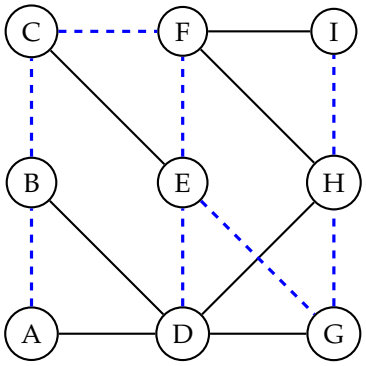
$$d(1,3), d(3,6), d(4,6), d(2,5)$$

Assume that the Floyd-Warshall algorithm uses the same ordering of vertices as the given names of the vertices.

First	Second	Third	Fourth
$d(4,6)$	$d(2,5)$	$d(3,6)$	$d(1,3)$

6 Unique MST (5 points)

Consider the undirected graph below with edge weights omitted. Suppose the dashed edges represent the unique minimum spanning tree (MST) in the graph.



1. How many edges are necessarily heavier than the edge FE? [ ]

   AD, BD, CE, FH, FI, all the other edges across the cut

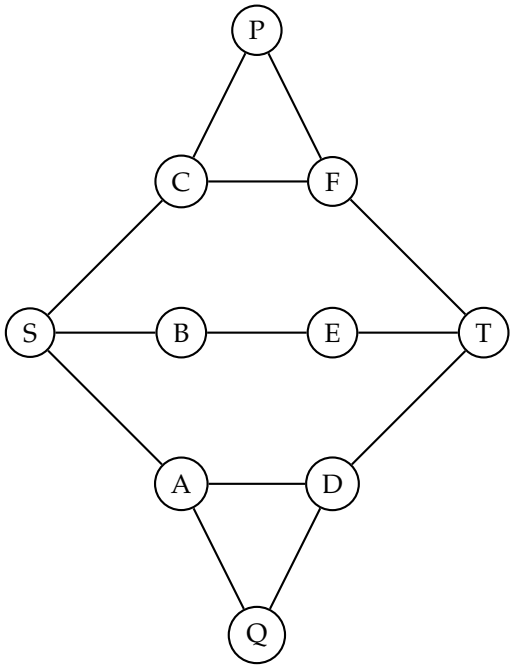
2. How many edges are necessarily lighter than the edge DH? [ ] DE, EG, GH, all the other edges along the cycle

3. Suppose k is the number of distinct edge weights in the graph, what is the smallest possible value of k? [ ]

   2

7 Min Cuts (4 points)

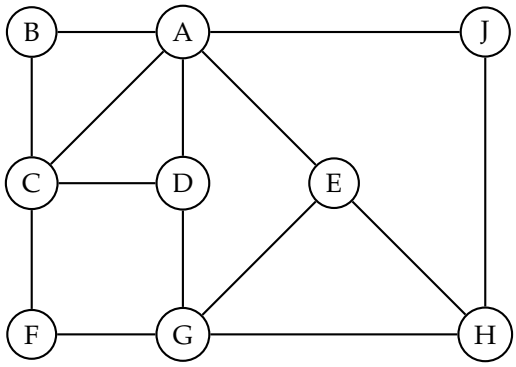
Assume that the capacity of every edge in the graph below is equal to 1. How many distinct minimum cuts are there in the following graph with source S and sink T?



Solution: Minimum cut value is 3, by cutting SA,SB,SC. Any minimum cut will have to cut one edge along each path from S to T.

Along SBET, this gives 3 choices. Along the other two branches, there are two choices each – SC or FT and SA or DT. So total number of minimum cuts is $3 \times 2 \times 2 = 12$

8 Kruskal and Prim's Algorithm (5 points)



The ordering of the weights of the edges in the above graph is as follows.

$$AB < EH < AD = FG < GD < BC < CD = AE < CF < AC < EG < AJ < GH < JH$$

(Assume all ties between edge-lengths are broken alphabetically)

1. What are the first 4 edges that Kruskal's algorithm includes in the tree? Write the edges in the order they are included.

First edge	Second edge	Third edge	Fourth edge
AB	EH	AD	FG

2. What are the first 4 edges that Prim's algorithm starting at A includes in the tree? Write the edges in the order they are included.

First edge	Second edge	Third edge	Fourth edge
AB	AD	GD	FG

9 Two-Player Game (5 points)

There are n^2 soldiers standing on a $n \times n$-grid (in n rows of n soldiers each).

Pick the tallest soldier from each row and then the shortest among them, call him A. Alternately, pick the shortest soldier from each column and then the tallest among them, call him B.

Which of the following statements are true? (If there are multiple true statements, mark all that are true. 5 points for marking all true statements, -3 for each false statement marked true. **Negative points on this question will apply to the exam total score.**) (**Hint:** Is there a two-player game lurking beneath?)

- ◯ Depending on the heights of soldiers, A can be taller than B or vice versa.
- ◯ A is always at least as tall as B.
- ◯ B is always at least as tall as A.
- ◯ A and B are always the same height.

◯ A and B are always the same person.

Solution: Suppose H is the $n \times n$ matrix of heights of soldiers then,

$$\text{height of } A = \min_i \left(\max_j H[i,j] \right)$$

$$\text{height of } B = \max_j \left(\min_i H[i,j] \right)$$

This is just the values of the two player game with only pure strategies allowed for each player. In one case, the min player goes first, and in other the min player goes second. The min player is at a disdavantage when going first, so height of $A \geq$ height of B.

10 NP-completeness true/false (10 points)

For each of the following questions, there are four options:

(1) True (T); (2) False (F); (3) True if and only if **P** = **NP**; (4) True if and only if **P** ≠ **NP**.

Circle one for each question.

Note: By "reduction" in this exam it is always meant "polynomial-time reduction with one call to the problem being reduced to."

If a statement is True or False without any assumptions on **P** and **NP**, pick that option over the other three.

1. Minimum Spanning Tree problem is not NP-complete

   ◯ T ◯ F ◯ **P** = **NP** ◯ **P** ≠ **NP**

   $P \neq NP$

2. CircuitSAT reduces to 3-SAT.

   ◯ T ◯ F ◯ **P** = **NP** ◯ **P** ≠ **NP**

   T

3. 3-SAT reduces to CircuitSAT.

   ◯ T ◯ F ◯ **P** = **NP** ◯ **P** ≠ **NP**

   T

4. Every NP problem reduces to integer programming.

   ◯ T ◯ F ◯ **P** = **NP** ◯ **P** ≠ **NP**

   T

5. 3-SAT reduces to HornSAT.

◯ T ◯ F ◯ $\mathbf{P = NP}$ ◯ $\mathbf{P \neq NP}$

$P = NP$

11 Buggy Distinct Elements (6 points)

The following code was intended to estimate the number of distinct elements in a stream of integers in the range $\{1,\ldots,n\}$. But the code is buggy.

$h \leftarrow$ `a pairwise independent hash function from` $\{1,\ldots,n\} \to (0,1)$

1. $s \leftarrow$ `first element of stream`
2. $\alpha = h(s)$
3. `For each upcoming stream element` r `do`
4. `If` $(\alpha < h(r))$ $\alpha \leftarrow h(r)$
5. `Return` $\lceil 1/\alpha \rceil$.

1. What is the first line number for which the code behaves differently from the distinct elements algorithm given in class?

   [ ] 4

2. Re-write only line 5 of the buggy code so that the estimate for the number of distinct elements is correct. [ ] $\frac{1}{(1-\alpha)}$ or $\frac{1}{1-\alpha} - 1$

   Note: a slightly more accurate estimate is $1/\alpha - 1$. Recall that minhash maps the k elements into the interval of length 1. Thus, the expected length of the first interval is $\frac{1}{k+1}$, or $E[\alpha] = \frac{1}{k+1}$. Solving this for k, gives an estimate of $k = \frac{1}{\alpha} - 1$. Either solution was fine.

12 True/False (14 points)

1. Suppose all the capacities in a graph are even numbers then the value of Maximum Flow is an even number.

   ◯ True ◯ False True

2. Suppose all the capacities in a graph are odd numbers then the value of Maximum Flow is an odd number

   ◯ True ◯ False True

3. The knapsack problem can be approximated to a factor of $1 + 10^{-6}$ in polynomial time.

   ◯ True ◯ False

   True

4. Let $G = (V, E)$ be a graph with edge weights w_e for each edge $e \in E$, and a unique minimum spanning tree T. Here the edge weights can be positive or negative. Suppose we change the edge weights to $w'_e = w_e^2$ for each edge $e \in E$, then T is also a minimum spanning tree for the new weights w'_e.

   ◯ True ◯ False False

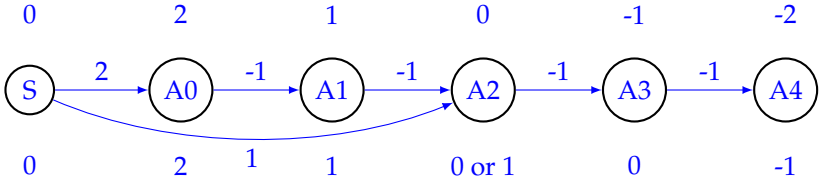
5. Let $G = (V, E)$ be a graph with edge weights w_e for each edge $e \in E$, and a unique minimum spanning tree T. Here the edge weights can be positive or negative. Suppose we change the edge weights to $w'_e = 2^{w_e}$ for each edge $e \in E$, then T is also a minimum spanning tree for the new weights w'_e.

   ◯ True ◯ False True

6. Consider a graph G all of whose edge weights are either positive integers or are equal to -1. Dijkstra's algorithm on G will always return the correct shortest path distances while updating each edge once.

   ◯ True ◯ False

   Solution: False. Consider that contains the a path of length k of negative edges, a source vertex with an edge of length two to the begining of the path, and an edge of length 1 to the middle of the path, then the labels of the last half of the path will be $0, -1, \ldots$ and they should be $2 - k/2 - 1, 2 - k/2 - 2, \ldots$. Below is a figure for $k = 4$. The correct distances are above the vertices and the ones produced by Dijkstra's algorithm are below. A2 is labelled 0 or 2 depending on whether the implementation allows for updates of visited vertices.



7. Suppose a graph G with edge weights contains a cycle C, then the edge with the largest weight on C cannot be in any minimum spanning tree.

◯ True ◯ False

Solution: True or False.

True. Consider a tree T containing the heaviest edge e on cycle C. Removing the heaviest edge in T produces two connected components in the graph with edges $T - e$. The cycle contains at least one edge $e' \neq e$ between the two components as there is a path between the endpoints of e in the cycle that does not use e. This edge is lighter than e' and adding it produces a spanning tree of lower costs. This contradicts the existence of T.

False. If there are ties, then the above argument is false.

13 Fill in the Blanks (24 points)

When asked for a bound, always give the tightest bound possible. In particular, if an exact bound is possible, then give it instead of an asymptotic bound. Some questions have choices in parentheses after the answer box.

1. The node that receives the highest [post] (pre/post) number in a depth-first search must lie in a [source] (source/sink) strongly connected component.

2. For a degree $n-1$ polynomial $p(x)$, its evaluation $p(0)$ can be computed in time [ ] from its coefficient representation. However, it takes [ ] time to compute $p(0)$ from its value representation given by $(p(1), p(\omega), \ldots, p(\omega^{n-1})$ where ω is a primitive n^{th} root of unity. (Assume n is a power of 2).

   $O(1), O(n \log n)$

3. Recall the linear programming relaxation for Vertex Cover problem. Suppose the optimal vertex cover in a graph G is of size α, while the optimal solution to the LP relaxation has value β then,

   [ ] $\times \alpha \quad \leq \quad \beta \quad \leq$ [ ] $\times \alpha$

   .

   $\frac{1}{2}, 1$

4. Let $h : \{1, \ldots, n\} \to \{1, \ldots, 1000\}$ be a hash function chosen uniformly from a pairwise independent hash family $\mathcal{H}$. Then for a pair $i \neq j$,

   $\Pr[h(i) \leq 300 \text{ and } h(j) > 700] =$ [ ]

   $300/1000 \times 300/1000 = 0.09$

5. Suppose an element occurs f times in a stream, then the count-min algorithm's estimate for its frequency is [ ] (write one of $\geq f$ or $\leq f$ or $= f$).

   $\geq f$

6. In union/find data structure of n items, if we use union by size without path compression then any combination of m unions and/or find operations takes at most [$O(m \log n)$] time.

7. Recall that Dijkstra's algorithm is run using a priority queue (whose implementation is a heap) where one uses the operations DeleteMin which returns and deletes the minimum valued entry in the queue and and DecreaseKey which decreases the value of the key for an item in the queue. Consider a graph G with n vertices and m edges and non-negative edge weights.

   (a) Give an asymptotic upper bound on the number of DeleteMin operations when Dijkstra's algorithm is run on G. $O(n)$

   (b) Give an asymptotic upper bound on the number of DecreaseKey operations when Dijkstra's algorithm is run on G. $O(m)$

8. Any acyclic undirected graph with n vertices and exactly $n-1$ edges is a tree.

   Solution: This is a definition of a tree, which we proved to be equivalent to others such as being acyclic and connected.

9. What is the minimum number of connected components in an undirected graph with n vertices and m edges?

   $\max(1, n-m)$

10. A vertex of the feasible region for a linear program with m constraints and n variables has at least n tight constraints. (A constraint of the form $a \cdot x \leq b$ is tight if $a \cdot x = b$.)

11. For the Knapsack Problem without repetition for items with values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$, where $K(w,i)$ is the maximum value subset of the first i elements of weight w. The recurrence is $K(w,i) =$ $\max(K(w-w_i, i-1) + v_i, K(w, i-1)$.

12. For the multiplicative weights algorithm with $\epsilon > 0$ on n experts whose losses in the range $(0,1]$ (each expert always has a non-zero loss), the potential function which is the sum of the weights of all the experts is strictly decreasing. (Answer should be increasing or decreasing.))

14 3-SAT for Powers of Two (8 points)

Prof. Bluffman has discovered an algorithm *PowerSAT* that solves 3-SAT instances in m clauses only when m is a power of two, in time $O(m^3)$. Complete the following argument that Prof. Bluffman has proved $NP = P$.

Proof: We use *PowerSAT* algorithm to solve the [ ] 3-SAT problem on all instances. Since the [ ] 3-SAT problem is [ ] NP-complete, this implies that $NP = P$.

Given an instance Φ of [ ] $3 - SAT$ problem, we construct an instance Φ' as follows, [ ]

We then run the PowerSAT algorithm on Φ'.

Let us suppose Φ has n clauses. Suppose $2^{k-1} < n \leq 2^k$, add $2^k - n$ clauses of the form $x_i \vee y_i \vee z_i$ for $i - 1, \ldots, 2^k - n$, where the variables x_i, y_i, z_i are entirely new variables. The resulting instance has a number of clauses which is a power of two, and so we can run PowerSAT algorithm on it

15 Happiest Increasing Subsequence (15 points)

You are given an array of positive numbers $a[1], \ldots, a[n]$. For an increasing subsequence of numbers, $a[i_1] < a[i_2] < \cdots < a[i_t]$, its happiness score is given by

$$\sum_{\ell=1}^{t} \ell * a[i_\ell]$$

For example for the input $a = [22, 44, 33, 66, 55]$, the increasing subsequence $[22, 44, 55]$ has happiness score $(1) * (22) + (2) * (44) + (3) * (55) = 275$ while the increasing subsequence $[22, 33, 55]$ has happiness score $(1) * (22) + (2) * (33) + (3) * (55) = 253$.

Devise a dynamic programming algorithm to compute the happiest increasing subsequence.

1. Subproblems

2. Recurrence Relation

3. Runtime

Solution:

1. $L(i, \ell)$ is the happiest subsequence that ends at the i items and using the ith element for a subsequence of length ℓ.

2. $L(i,\ell) = \max_{j<i, a[j]<a[i]} L(j,\ell-1) + \ell * a[i]$
3. $O(n^3)$ since there are at most $O(n^2)$ entries and each entry take $O(n)$ time to compute.

16 Weakest Link

The country of Snowland has n cities which are connected by a road network, specified by a graph $G = (\{1, \ldots, n\}, E)$.

Excessive snowfall often disrupts the edges in the network. Every edge $(u, v) \in E$ has an associated *strength* $s[u, v]$ – a positive number. The strength of an edge indicates how much snowfall it can withstand.

For a path P in the graph, its strength $s(P)$ is the minimum strength of an edge on the path P. (*A path is only as strong as the weakest edge on it*).

For a pair of vertices i, j, let $D[i, j]$ denote the strength of the strongest path from i to j. Formally,

$$D[i, j] = \max_{\text{paths } P \text{ from } i \text{ to } j} s(P)$$

1. (10 points) Recall the dynamic programming based algorithm for all-pairs shortest path (Floyd-Warshall algorithm). We will (slightly) modify it to compute $D[i, j]$ for all pairs of vertices i, j.

   (a) The definition of subproblems $d[i, j, k]$ is,

   Solution: $d[i, j, k]$ = strength of the strongest path from i to j that uses only vertices $\{1, \ldots, k\}$ as intermediaries.

   (b) The base cases are given by,

   Solution:

   $$d[i, j, 0] = \begin{cases} s(i, j) & (i, j) \in E \\ -\infty & \text{otherwise} \end{cases}$$

   (c) The recurrence relation would be

   Solution:

   $$d[i, j, k] = \max\left(\min(d[i, k, k-1], d[k, j, k-1]), d[i, j, k-1]\right)$$

2. (**6 points**) We will now modify Djikstra's algorithm so as to compute strongest paths from a single vertex. Here is the pseudocode for Djikstra's algorithm on a graph G with edge lengths $\ell[u,v]$.

```
procedure Djikstra(G, edge lengths ℓ)
1       for i = 1 to n
2           dist[i] = ∞
3           prev[i] = null
4       dist[1] = 0

5       H =makequeue(V)
6       while H is not empty
7           u =deletemin(H).
8           for all edges (u,v) ∈ E
9               if dist[v] ≥ dist[u] + ℓ[u,v]:
10                  dist[v] = dist[u] + ℓ[u,v]
11                  prev(v) = u
12                  decreasekey(H,v)
```

How would you modify the following line so as to compute strongest paths instead of shortest paths?

$$\texttt{If } (dist[v] \geq dist[u] + \ell[u,v]): \quad dist[v] = dist[u] + \ell[u,v]$$

Your modification will appropriately use $s[u,v]$ – the strengths of roads in the graph G.

(Hint: the $dist[u]$ values in the modified code do not need to exactly equal the strengths of the paths, but *prev* pointers would yield the strongest path)

Solution: If $(dist[v] \geq \max(dist[u], 1/s[u,v]))$ set $dist[v] = \max(dist[u], 1/s[u,v])$.

To see why this works, first let us formally define the distance values.

$$dist[v] = \max_{\text{paths } P \text{ from 1 to } v} \left(\min_{\text{edge } e \in P} s(e) \right)$$

and the strongest path is nothing but

$$P^* = \underset{\text{paths } P \text{ from 1 to } v}{\arg\max} \left(\min_{\text{edge } e \in P} s(e) \right)$$

Note that the above path can also be obtained as,

$$P^* = \underset{\text{paths } P \text{ from 1 to } v}{\arg\min} \left(\max_{\text{edge } e \in P} \frac{1}{s(e)} \right) \tag{1}$$

To see this, observe that for every path

$$\frac{1}{s(P)} = \max_{e \in P} \frac{1}{s(e)}$$

and that,

$$\arg\min_P \frac{1}{s(P)} = \arg\max_P s(P)$$

Equation (1) expresses strongest path in manner similar to shortest paths. Recall that shortest paths are given by,

$$P' = \underset{\text{paths } P \text{ from 1 to } v}{\arg\min} \left(\sum_{\text{edge } e \in P} \ell(e) \right)$$

where $\ell(e)$ is the length of the edge e. If we consider $1/s(e)$ to be the length of edge $\ell(e)$ then only difference between shortest paths and strongest paths now, is that $\sum_{e \in P}$ is replaced by $\max_{e \in P}$. The modification in the code changes the values of $dist[v]$ so that they reflect $\max_{e \in P}$ instead of $\sum_{e \in P}$.

17 NP-Completeness Reductions (10 points)

Show that the following problems are NP-complete by providing a polynomial-time reduction. You may assume that the following problems are NP-complete: Rudrata (Hamiltonian) Path, Rudrata (Hamiltonian) Cycle, Vertex Cover, Independent Set, 3-SAT and Integer Programming.

1. **Far-Away Points**

   Input: Distances $\{d[i,j]\}$ between n cities numbered $\{1,\ldots,n\}$ such that they satisfy the triangle inequality:

   $$d[i,j] + d[j,k] \geq d[i,k] \qquad \forall i,j,k$$

   and positive numbers k and D.

   Solution: A set S of k cities that are at least D away from each other, i.e.,

   $$d[i,j] \geq D \qquad \forall i,j \in S$$

   Proof. It is clear that Far-Away Points problem is in NP. Now we will use a polynomial time reduction to show that the problem is indeed NP-complete.

   Given an instance Φ of the problem [ ] *we will construct an instance Ψ of the problem* [ ] as follows.

   [ ]

   The proof that this is a valid reduction is as follows:

   [ ]

Solution: Given an instance Φ of the Independent Set problem we will construct an instance Ψ of the problem FarAway points problem as follows.

Suppose the instance consists of $\Phi = ((V, E), \text{ size } s)$, we will set $d[i, j]$ to be the shortest path distance between i and j in the unweighted undirected graph Φ.

Set $k = s$ and $D = 2$.

By running all pairs shortest paths, one can construct Ψ from Φ in polynomial time. Since shortest path distances satisfy the triangle inequality, our instance is a valid instance of Φ^*.

Suppose $A \subseteq V$ is an independent set of size s, then for every pair of vertices $i, j \in A$, we will have $d[i, j] \geq 2$. So $S = A$ yields a set of $k = s$ points that are $D = 2$ away from each other.

Conversly, if the set S of k points are distance 2 away, then no pair of them are connected by an edge in Φ. So $A = S$ is an independent set of size k in Φ.

Alternatively, one can set $d[i, j] = 1$ when $(i, j) \in E$ and $d(i, j) \in [1, 2]$ for $(i, j) \notin E$. Other constructions where $d[i, j] = s$ for $(i, j) \in E$, and $d[i, j] = 2s$ for $(i, j) \notin E$, and use $D = 2s$. This obeys the triangle inequality and the previous argument works for arguing that a solution here corresponds to an independent set solution.

18 Usage Logs (15 points)

A network router is monitoring login/logout requests to a social media platform. Each request that the router observes is of the form below.

`Request` $r\{$

- $r.Time$ `# a number in` $\{1,\ldots,m\}$ `indicating the time (in minutes from start of the day) at which request is received.`
- $r.Id$ `# a number in` $\{1,\ldots,n\}$ `that is the identity of the user.`
- $r.Type$ `# either LOGIN/LOGOUT indicating the type of the request.`

$\}$

The router is trying to estimate the average time spent by a user on the website using a streaming algorithm. Assume that the stream of requests is a valid in that,

1. There is never a logout request for a user who is not currently logged in.
2. The system begins with no user logged in, and all the users logout by the end of the stream.

Note that the same user can have multiple login/logout sessions in the stream. Fill in the pseudocode on the next page.

1 $h \leftarrow$ a random hash function from $\{1,\ldots,n\} \to (0,1)$

2 Define your variables here (indicate the type of each variable, and its initial value).

3 for each request r in stream do

4 if (r is a LOGIN request) {

5

6 }

7 if (r is a LOGOUT request) {

8

9 }

10 end For loop

11 Return

Solution: Here are three solutions to the problem.

Solution 1:

Estimate number of distinct users logging in using minHash algorithm, and compute the total time spent, and then use the two to compute the average.

total = 0 : integer

minHashValue = 1 : real number in $(0,1)$

1. If (r is a LOGIN request){

2. $total = total - r.Time$

```
3.          minHash = min(minHash, h(r.Id))
4. }
5. If (r is a LOGOUT request){
6.          total = total + r.Time
7.          minHash = min(minHash, h(r.Id))  }
8. return minHash * total
```

Solution 2:

Estimate number of distinct users logging in using minHash algorithm, and compute the total time spent (in a different way than solution 1), and then use the two to compute the average. (solution assumes that the time on the very first request is 0, one would have to slightly modify the initialization otherwise)

```
currentNumUsersLoggedIn = 0 :  integer
previousUpdateTime = 0 :  integer
totalTime = 0:  integer
minHashValue = 1 :  real number in (0,1)
```

```
1. If (r is a LOGIN request){
2.          total = (r.Time − previousUpdateTime) * currentNumUsersLoggedIn
3.          currentNumUsersLoggedIn + +
4.          previousUpdateTime = r.Time
5.          minHash = min(minHash, h(r.Id))
6. }
7. If (r is a LOGOUT request){
8.          total = (r.Time − previousUpdateTime) * currentNumUsersLoggedIn
9.          currentNumUsersLoggedIn − −
10.         previousUpdateTime = r.Time
11.         minHash = min(minHash, h(r.Id))
12. }
13. return minHash * total
```

Solution 3

Here we sample a uniformly random user who has logged in using the minHash algorithm, and output the total time spent by that user. The answer is equal to the average in expectation. Solution assumes no two hash values are exactly equal, which is reasonable since the range is a real number in $(0,1)$, or in general a large range.

totalTimeOfMinHashUser = 0: integer
minHashValue= 1 : real number in $(0,1)$

1. If (r is a LOGIN request){
2. if($h(r.Id) < minHashValue$) then {
3. $minHashValue = h(r.Id)$
4. $totalTimeOfMinHashUser = -r.Time.$
5. }
6. if($h(r.Id) == minHashValue$) then {
7. totalTimeOfMinHashUser = total - r.Time.
8. }
9. }
10. If (r is a LOGOUT request){
11. if($h(r.Id) == minHashValue$) then {
12. $totalTimeOfMinHashUser = total + r.Time$
13. }
14. }
15. return $totalTimeOfMinHashUser$

19 Three-Tour (15 points)

Recall that a TSP tour is a cycle that visits all the vertices of a graph exactly once.

Analogously, a three-tour consists of three **disjoint** cycles C_1, C_2, C_3 such that each vertex appears in exactly one of the cycles exactly once. The cost of a 3-tour is the sum of the lengths of the edges on the three cycles. (Here a single vertex is considered a cycle of length 1, so one or more of the cycles C_1, C_2, C_3 can consist of a single vertex.)

Suppose you are given a complete graph on n vertices with distances $dist(\cdot, \cdot)$ between pairs of vertices. Assume that the distances satisfy the triangle inequality in that for each triple of vertices i, j, k,

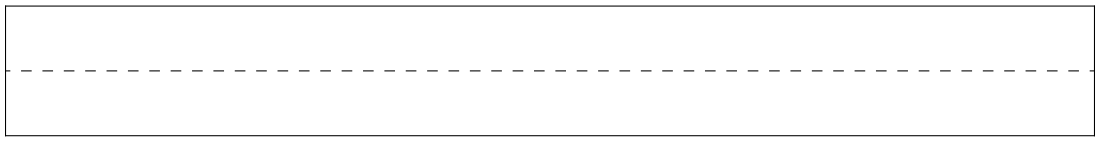
$$dist(i, j) + dist(j, k) \geq dist(i, k)$$

Describe a 2-approximation algorithm for the minimum cost three-tour problem. (Give a succinct but clear description of your algorithm in at most four sentences)

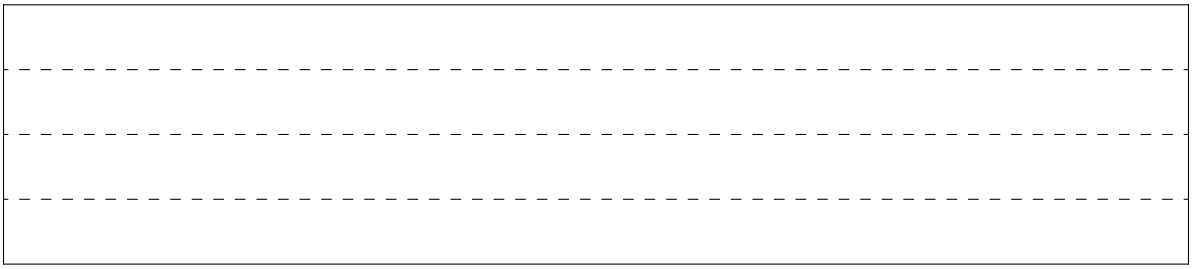
We will now sketch a proof that the algorithm yields a 2-approximation to the Three-Tour problem (on next page).

Proof of approximation factor: *We will split the proof into the following two claims.*

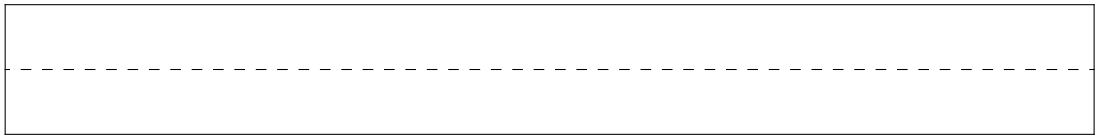
Claim 1: *Cost of the optimal Three-Tour is at least the cost of ..*



Proof of Claim1: *The claim holds because,*



Claim 2: *Cost of Three-Tour output by the algorithm is at most twice the cost of..*



Proof of Claim 2: *We will omit the proof of claim 2 here.*

Solution: Compute the Minimum Spanning Tree (MST) and delete the two heaviest edges to recover a forest with three trees T_1, T_2, T_3. The rest of the algorithm is analogous to the TSP approximation algorithm, but on three trees T_1, T_2, T_3. In words, perform a DFS on each of the three trees T_1, T_2, T_3 to obtain a traversal $\Gamma_1, \Gamma_2, \Gamma_3$. Shortcut the traversals to not have any vertex appearing more than once. This yields a Three-Tour solution.

Claim 1: Cost of the optimal Three-Tour is at least the cost of the (Minimum spanning 3-forest), which is the minimum cost graph that connects the vertices into three connected components.

Proof of Claim1: Take a 3-Tour consisting of cycles C_1, C_2, C_3 and delete one edge from each cycle to get a forest with three paths P_1, P_2, P_3. The cost of the three cycles $\geq$ cost of three paths. Since the three paths is a forest that connects the graph into three components, its cost is at least the cost of Minimum spanning 3 forest.

Claim 2: Cost of Three-Tour output by the algorithm is at most twice the cost of Minimum spanning 3 forest.

Proof of Claim 2: We will omit the proof of claim 2 here.

Remark: we only gave any credit when one computed the minimum spanning forest with three components as above. Partial credit was only awarded if one got this far.

20 Multiplicative weights intuition.(10 points)

Recall that the multiplicative weights method starts with a weight w_i of 1 for each expert i. Furthermore, on a day where an expert i suffers a loss of ℓ_i, the weights are updated by multiplying the weight of an expert by $(1-\epsilon)^{\ell_i}$ if the loss for expert i is ℓ_i. The algorithm chooses an expert with probability proportional to w_i and suffers the loss of the chosen expert.

1. Consider a simple situation with two experts A and B where expert A loses $1/3$ **every** day and expert B loses $2/3$ **every** day. If one runs the multiplicative weights framework on this situation, with $\epsilon = 1/2$.

   (a) What is the weight of expert A on day d?

   $2^{-d/3}$

   (b) What is the probability of choosing expert A on day d? $\frac{2^{-d/3}}{2^{-2d/3}+2^{-d/3}}$

   (c) How many days does it take so that one chooses expert A with probability at least .99?

   Solution: Let d be the days, we want $\frac{2^{-d/3}}{2^{-2d/3}+2^{-d/3}} \geq .99$.

   We can set $x = 2^{d/3}$, and obtain the equation $\frac{x}{x^2+x} \geq .99$, or $x \geq .99(x^2+x)$.

   Solving one get $x = 2^{-d/3} \geq 1/99$, and then solving for d, we get $d \geq 3\log 99$.

2. What if there are $n+1$ experts and the first expert loses $1/3$ and all other experts lose $2/3$ every day, how many days does one need for the probability of choosing first expert to be least .99?

   Solution:

   The ratio is now $\frac{2^{-d/3}}{n2^{-2d/3}+2^{d/3}} \geq .99$. Again, we substitute $x = 2^{-d/3}$ into the equation getting $x \geq .99(nx^2+x)$ or $x \leq \frac{1}{99n}$.

   Thus, we get $d \geq 3\log(99n) = 3(\log 99 + \log n)$.

   Notice that the logarithm one pays for the number of experts, n, is only additive with the logarithm that one pays to drive down the probability of choosing the wrong expert to .01.

21 Linear Programming relaxation of Travelling Salesman (10 points)

For the Traveling Salesman problem, one is given a set of cities $V = [1, \ldots, n]$ and distances $d(i, j)$ between pairs of cities. A tour is a circular ordering of the cities.

Thus, we formulate an (integer) linear programming relaxation where a variable $x_{i,j}$ is 1 if the pair i, j are adjacent in a tour and zero otherwise: in some sense these are the edges in the cycle that form a tour. Also, we note that $x_{i,j}$ is unordered in that there is only one variable for $x_{i,j}$ and $x_{j,i}$.

The linear program's objective function is

$$\min \sum_{i,j \in V} d(i,j) x_{i,j}$$

The notion that an edge cannot be used more than once can be enforced by equations of the form:

$$\forall i, j \in V, x_{i,j} \leq 1.$$

We call these the packing constraints.

Also, notice for a tour, each vertex is incident to exactly 2 edges.

$$\forall i \sum_{j} x_{i,j} = 2$$

which we call the degree constraint.

Moreover, the number of edges between the two sides of a partition $V = S \cup \overline{S}$ of the cities, must be at least two as a tour should be connected and enter and leave any subset.

$$\forall S \subset V, \sum_{i \in S, j \in V - S} x_{i,j} \geq 2$$

These constraints are the subtour elimination constraints. And finally, we have that $x_{i,j} \geq 0$. for all i, j.

The dual linear program will have dual variables $y_{i,j}$ for each packing constraint for pair of vertices i, j, and y_i for each degree constraint for each vertex i, and y_S for each subtour elimination constraint for each set $S \subset V$. Write the dual linear program using these variables.

1. What is the objective function?

2. What are the constraints?

Solution:

$\max \sum_{i \in V} 2y_i + \sum_{S \subset V} 2y_S - \sum_{i,j \in V} y_{i,j}$

$\forall i, j \in V \quad y_i + y_j - y_{i,j} + \sum_{S \ni i, V - S \ni j} y_S \leq d(i,j)$

$y_{i,j} \geq 0$

y_i are unrestricted