Final

Name: Targaryen

SID: 0123456789

Name and SID of student to your left: Lannister

Name and SID of student to your right: Stark

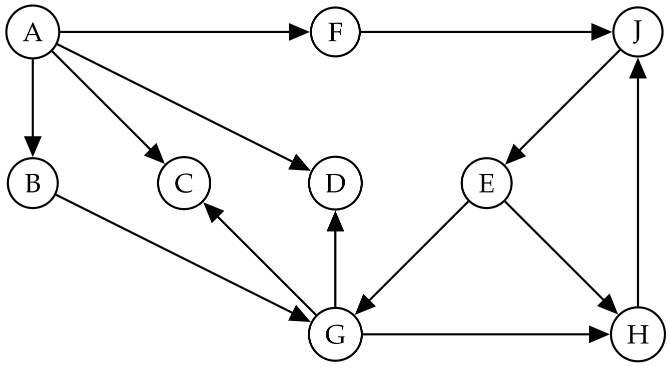
Exam Room:

Rules and Guidelines

- **The exam will last 170 minutes.**
- The exam has 170 points in total.
- 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.
- You may assume that comparison of integers or real numbers, and addition, subtraction, multiplication and division of integers or real or complex numbers, require $O(1)$ time.
- If multiple differing solutions are provided for a given question, only the first solution we read will be graded.
- Good luck!

1 SCCs (8 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		

Solution:

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

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
$A \to C$	F
$F \to J$	C
$H \to J$	T
$E \to H$	B

3. List the strongly connected components of the above graph in some 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	A
2	B
3	F
4	G,E,H,J
5	C
6	D
7	
8	
9	
10	

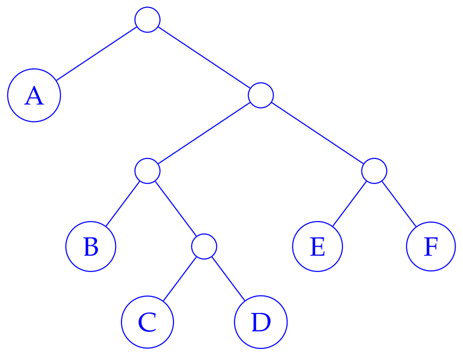
2 Huffman Encoding (3 points)

Construct the Huffman code for the alphabet A, B, C, D, E, F, with the following frequencies.

Letter	A	B	C	D	E	F
Frequency	100	30	1	29	40	50

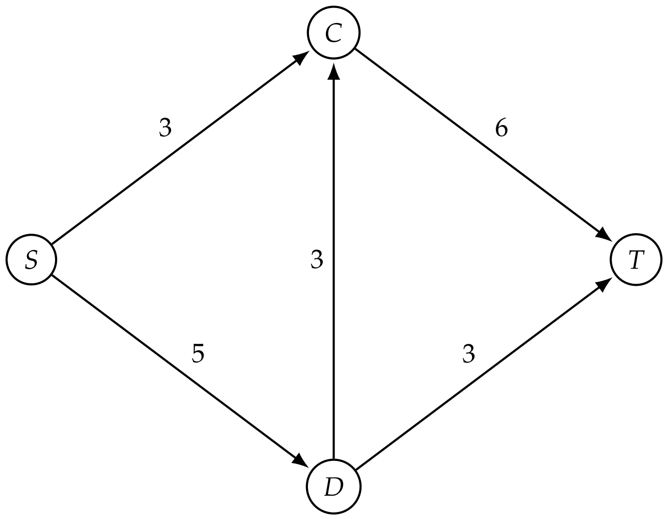
Draw the tree corresponding to the Huffman code.

Solution:



3 Max Flow via Linear Programming (12 points)

Consider the following directed capacitated graph.



1. Write the linear program corresponding to the maximum flow problem from S to T.

Solution:

$$\begin{aligned}
\max \quad & x_{sc} + x_{sd} \\
\text{s.t.} \quad & x_{sc} + x_{dc} - x_{ct} = 0 \\
& x_{dc} + x_{dt} - x_{sd} = 0 \\
& x_{sc} \leq 3, x_{sd} \leq 5, x_{dc} \leq 3, x_{ct} \leq 6, x_{dt} \leq 3, \\
& x \geq 0
\end{aligned}$$

There's an alternate solution which is the path formulation:

$$\begin{aligned}
\max \quad & x_{sct} + x_{sdt} + x_{sdct} \\
\text{s.t.} \quad & x_{sct} \leq 3 \\
& x_{sct} + x_{sdct} \leq 6 \qquad \text{(redundant, can do without this one)} \\
& x_{sdct} + x_{sdt} \leq 5 \\
& x_{sdct} \leq 3 \\
& x_{sdt} \leq 3 \\
& x \geq 0
\end{aligned}$$

2. Write the dual to the above linear program. (The dual linear program corresponds to the minimum cut problem.)

Solution: There are multiple correct solutions.

$$\begin{aligned} \min \quad & 3y_{sc} + 5y_{sd} + 3y_{dc} + 3y_{dt} + 6y_{ct} \\ \text{s.t.} \quad & y_{dc} \geq z_c - z_d \\ & y_{sc} \geq z_c \\ & y_{sd} \geq z_d \\ & y_{ct} \geq 1 - z_c \\ & y_{dt} \geq 1 - z_d \\ & y \geq 0 \end{aligned}$$

The LP above is the "standard" min s-t cut LP. But if we directly turn the primal LP above into its dual, we get the same thing but with all the z's replaced by $1 - z$, which is also valid and gives the same output:

$$\begin{aligned} \min \quad & 3y_{sc} + 5y_{sd} + 3y_{dc} + 3y_{dt} + 6y_{ct} \\ \text{s.t.} \quad & y_{dc} \geq z_d - z_c \\ & y_{sc} \geq 1 - z_c \\ & y_{sd} \geq 1 - z_d \\ & y_{ct} \geq z_c \\ & y_{dt} \geq z_d \\ & y \geq 0 \end{aligned}$$

The dual of the path formulation of the max flow LP is the following:

$$\begin{aligned} \min \quad & 3y_{sc} + 5y_{sd} + 3y_{dc} + 3y_{dt} + 6y_{ct} \\ \text{s.t.} \quad & y_{sc} + y_{ct} \geq 1 \\ & y_{sd} + y_{dc} + y_{ct} \geq 1 \\ & y_{sd} + y_{dt} \geq 1 \\ & y \geq 0 \end{aligned}$$

4 Zero-Sum Games (6 points)

Consider a zero-sum game given by the following payoff matrix.

	C_1	C_2	C_3
R_1	1	2	3
R_2	2	3	1
R_3	3	1	2

The row-player goes first, and the column player goes second. The row player is trying to maximize their payoff, which is specified by the entries of the matrix.

The *value* of the game is the expected payoff of the row-player when both players use optimal mixed strategies.

Mark the following statements as true or false (each subpart is independent of the others).

1. If the column player is restricted to the three pure strategies $\{(1,0,0),(0,1,0),(0,0,1)\}$, row player's expected payoff doesn't change.

   ◯ True ◯ False

   True

2. If the row player is restricted to the three pure strategies $\{(1,0,0),(0,1,0),(0,0,1)\}$, column player's payoff doesn't change.

   ◯ True ◯ False

   False

3. Suppose we modify the game to include an additional row (so the resulting matrix is a 4×3 payoff matrix). Then the value of the game can possibly both increase or decrease depending on the entries of the new row.

   ◯ True ◯ False

   False. The row player has more moves in the new game, but the column player has the same set of moves. Therefore, the value of the row player cannot decrease.

Write a linear program to find the optimal mixed strategy for the row-player.

Solution:

$$\begin{aligned} \max \quad & z \\ \text{s.t.} \quad & z \leq x_1 + 2x_2 + 3x_3 \\ & z \leq 2x_1 + 3x_2 + x_3 \\ & z \leq 3x_1 + x_2 + 2x_3 \\ & x_1 + x_2 + x_3 = 1 \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \end{aligned}$$

5 Median (4 points)

Suppose we ran quickselect (the randomized divide and conquer algorithm for finding the k-th smallest element of an array) on the following array to find it's median.

3	8	4	12	1	9	4	6	5

1. Specify which elements should be picked as pivots at each step in order to **maximize** the runtime of this algorithm. Write the numerical value of the elements, not their indices. You might not need to fill all boxes provided.

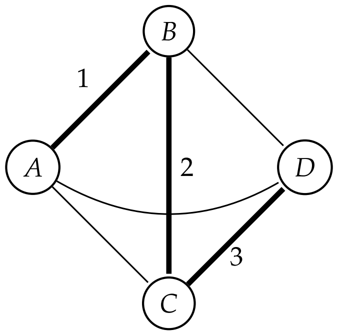
1	3	4	12	9	8	6	5		

2. Specify which elements should be picked as pivots at each step in order to **minimize** the runtime of this algorithm. Write the numerical value of the elements, not their indices. You might not need to fill all boxes provided.

5									

6 MST weights via linear program (8 points)

Consider the following graph on 4 vertices A, B, C, D with edge weights $\{w_{AB}, w_{BC}, w_{BD}, w_{AC}, w_{CD}, w_{AD}\}$ on the edges.



We are given the weights $w_{AB} = 1$, $w_{BC} = 2$ and $w_{CD} = 3$.

The other three weights w_{AC}, w_{AD}, w_{BD} are unknown, but they are such that $AB - BC - CD$ is a minimum spanning tree of the graph. Write a linear program to determine the smallest possible value of $2w_{AC} + 3w_{AD} + 4w_{BD}$.

1. What are the constraints on the unknowns w_{AC}, w_{AD}, w_{BD}?

Solution: For each edge AB, BC, CD, it must be the lightest edge across the cut it connects.
For example, AB must be the lightest edge across the cut $\{A\} \cup \{B, C, D\}$. So we get,

$$w_{AC} \geq 1 \text{ and } w_{AD} \geq 1$$

Similarly, we get the constraints

$$w_{AC} \geq 2, \text{ and } w_{BD} \geq 2 \text{ and } w_{AD} \geq 2$$

from the cut $\{A, B\} \cup \{C, D\}$ that edge BD connects.
Finally, the constraint

$$w_{BD} \geq 3 \text{ and } w_{AD} \geq 3$$

from the cut $\{A, B, C\} \cup \{D\}$ that the edge CD crosses.

What is the value of the optimum? $\boxed{}$

$2 \cdot 2 + 3 \cdot 3 + 4 \cdot 3 = 25$

7 Short Answer (24 points)

1. Suppose $T(n) = 2T(n/4) + n^2$ then $T(n) =$ $\boxed{}$ (write the tightest possible bound)

   $\Theta(n^2)$

2. If $T(n) = 2T(n/4) + n^2 \log^8 n$ then $T(n) =$ $\boxed{}$ (write the tightest possible bound)

   $\Theta(n^2 \log^8 n)$

3. An n-vertex directed acyclic graph can have as many as $\boxed{}$ $\binom{n}{2}$ directed edges, while an n-vertex undirected graph with no cycles can have as many as $\boxed{}$ $n-1$ edges.

4. Suppose A, B, C, D, E are five vertices that are part of a large directed graph. Here are the pre and post values of these vertices in a DFS traversal. The traversal was carried out in an unknown order, not necessarily lexicographical.

Node	pre	post
A	20	30
B	10	80
C	50	54
D	90	100
E	93	97

Then,

(a) There is necessarily no directed path from A to C in the graph.

○ True ○ False

Solution: False, there may be a directed path through a parent of A in the DFS tree

(b) The directed edge $A \to C$ is not part of the graph.

○ True ○ False

Solution: True, because DFS returned from vertex A before visiting C.

(c) If the directed edge $D \to A$ exists, then it is a cross-edge the graph.

○ True ○ False

Solution: True.

(d) D and E are necessarily part of the same SCC.

○ True ○ False

Solution: False. D is an ancestor of E, so there is a path from D to E in the graph, but there may be no path from E to D.

(e) C and E are part of different SCCs.

○ True ○ False

Solution: This question is ambiguous since we don't specify they are necessary in different SCCs. Thus, both true and false are technically correct answers.

5. Let $\omega = e^{2\pi i/16}$ be the 16th root of unity. Suppose the Fourier transform of a vector $(p_0, p_1, \ldots, p_{15})$ is $(1, \omega, \omega^2, \ldots, \omega^{15})$.

   What is the value of $\sum_{i=0}^{15} p_i \cdot 2^i$? $\boxed{}$

   Solution: From the Fourier transform it is clear that the polynomial $p(x) = \sum_i p_i x^i$ is given by $p(x) = x$. Therefore, $p(2) = 2$

6. The following points are some of the feasible solutions of a linear program.

$$(0,0), (0,10), (5,8), (10,10), (10,0), (5,2)$$

   For each of these points indicate whether they are necessarily feasible, necessarily infeasible or neither.

   (a) $(0,5)$

   ○ necessarily feasible ○ necessarily infeasible ○ neither **Solution:** Necessarily Feasible.

   (b) $(15,18)$

   ○ necessarily feasible ○ necessarily infeasible ○ neither **Solution:** Neither

   (c) $(5,10)$

   ○ necessarily feasible ○ necessarily infeasible ○ neither **Solution:** Necessarily feasible.

7. The dynamic programming based algorithm for edit distance between two strings $x[1,\ldots,m]$ and $y[1,\ldots,n]$ takes $\boxed{}$ $O(mn)$ time, because the algorithm uses $\boxed{}$ $O(mn)$ many subproblems computing each of which takes $\boxed{}$ $O(1)$ time.

8. Suppose we run the distinct elements algorithm on a stream $s_1, \ldots, s_m$. Assume that the algorithm uses a hash-function h that is perfectly random.

   If there is only one distinct element in the stream, what is the probability that the algorithm outputs an estimate > 99?

   $\boxed{}$

   1/100.

   If the smallest hash element is α, the algorithm outputs $\lceil \frac{1}{\alpha} \rceil - 1$. Therefore, for it to output an estimate > 99, there must be a hash value smaller than $1/100$. Since there is only one distinct element, the probability of this event is at most $\frac{1}{100}$

9. Suppose we run the distinct elements algorithm on a stream $s_1, \ldots, s_m$. Assume that the algorithm uses a hash-function h that is perfectly random.

   If there are 100 distinct elements in the stream, what is the probability that the algorithm outputs an estimate < 9?

$$(1-1/10)^{100}$$

The smallest hash value needs to be larger than $1/10$. For each element, the probability that its hash value is larger than $1/10$ is $(1-1/10)$. Hence, the probability that all hash values are larger than $1/10$ is at most $(1-1/10)^{100}$

10. Suppose we pick a hash function $h : \{0, 1. \ldots, p-1\} \to \{0, 1, \ldots, p-1\}$ from a pairwise independent hash family. What is the probability that $h(2) = (h(1)+2) mod p$?

$1/p$

Since the hash family is pairwise independent, $h(1)$ and $h(2)$ are independent random values in $\{0, 1, \ldots, p-1\}$. Therefore, the probability that $h(2) = h(1) + 1$ is exactly $1/p$.

8 Set Cover (6 points)

Let S_1, S_2, S_3 denote the first 3 sets picked during an execution of the greedy algorithm for Set Cover problem.

Suppose the size of the universe is 24 and the sizes of the unions of sets are as follows

$$\begin{aligned} |S_1| &= 12 \\ |S_1 \cup S_2| &= 15 \\ |S_1 \cup S_2 \cup S_3| &= 17 \end{aligned}$$

For each subpart, provide the tightest possible bound.

1. The size of optimal set cover is at least $\boxed{}$

   Solution: Ans: 5

   The number of elements still uncovered after each iteration are respectively $N_0 = 24, N_1 = 12, N_2 = 9, N_3 = 7$. If there are k sets in optimal set cover then, $N_{t+1} \leq N_t(1 - \frac{1}{k})$ for each t. Smallest possible value of k for which the above sequence satisfies the inequality is $k \geq 5$.

2. The greedy algorithm will necessarily pick at least $\boxed{}$ sets before termination.

   Solution: The greedy algorithm has 7 more elements to cover, and in any future iteration, it will cover at most 2 elements, since $N_2 - N_3 = 2$.

   So greedy algorithm will take at least 4 more iterations, i.e., 7 sets before termination.

3. The greedy algorithm will pick at most $\boxed{}$ sets before termination.

   Solution: Greedy algorithm needs to cover 7 more elements, and it might use 7 sets for it. Thus, in total, the greedy algorithm could use $7 + 3 = 10$ sets.

9 NP-completness true/false (9 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."

1. Rudrata Cycle reduces to Maximum Flow.

   ○ T ○ F ○ **P** = **NP** ○ **P** ≠ **NP**

   Solution: P = NP. If such a reduction existed, it would give a polynomial-time algorithm for Independent Set. Because Independent Set is NP-complete, this means that there's a polynomial time for every NP problem. If P=NP, then there's a trivial reduction from Independent Set to LIS using the polynomial-time solution algorithm for Independent Set.

2. Every problem in NP reduces to the independent set problem.

   ○ T ○ F ○ **P** = **NP** ○ **P** ≠ **NP**

   Solution: True. Independent Set problem is NP-hard, so every problem in NP reduces to Independent Set.

3. Every problem in P reduces to the Maximum Flow problem.

   ○ T ○ F ○ **P** = **NP** ○ **P** ≠ **NP**

   Solution: True. Since a polynomial-time reduction can essentially solve the problem by itself, without really using the call to Maximum Flow problem.

4. 3-SAT can be reduced to independent set, but not vice-versa.

   ○ T ○ F ○ **P** = **NP** ○ **P** ≠ **NP**

   Solution: If we consider the decision version of Independent Set, then the problem is NP-complete, and all NP-complete problems are reducible to one another. So statement is false.

   However, "Independent Set" could also be interpreted as its optimization version, namely "Minimum Independent Set". Minimum Independent Set is NP-hard, but not necessarily in NP. So, the statement would be true if $P = NP$.

   Since there are two different interpretations of "Independent Set", both answers are accepted.

5. Knapsack problem can be solved in polynomial-time.

   ○ T ○ F ○ **P** = **NP** ○ **P** ≠ **NP**

Solution: P=NP or False. Similar to the previous subpart, it depends on whether or not you're assuming the decision version of Knapsack or the optimization version. Hence, both solutions are accepted.

6. All NP-Hard problems can reduce to 3 SAT in polynomial time.

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

Solution: False, the halting problem is NP-Hard, but does not reduce to 3-SAT.

10 Non-MST Edge (10 points)

Devise a $O(|V|)$-time algorithm for the following problem:

Input: Graph $G = (V, E)$ with distinct edge weights w_{ij} for each edge $(i, j) \in E$.

Solution: Find some edge $(i, j) \in E$ such that (i, j) is NOT part of any MST.

1. Give a succinct and precise description of the algorithm:

2. Briefly justify the run-time of the algorithm.

Solution:

1. The idea is to find the heaviest edge in some cycle of the graph.

   Run DFS on the graph until we encounter a back-edge $u \to v$, discovering a cycle. We will find the heaviest edge on this cycle and return it. To find the heaviest edge, check all edges along the path from v to u in the tree. This can be done efficiently by popping the call stack for Explore to retrace the path up the tree.

2. Each edge encountered during DFS in an undirected graph is either a tree edge or a back-edge. Since there are only $|V| - 1$ tree edges, DFS will encounter a back-edge in $O(|V|)$ time.

11 Discount Path (15 points)

Consider a directed graph $G = (V, E)$ with positive weights $c(i, j)$ on its edges.

Given a path $P = v_0 \to v_1 \ldots \to v_k$, the *discounted cost* of the path is the sum of lengths of all but the the longest edge. Formally,

$$cost(P) = \left(\sum_{i=0}^{k-1} c(v_i, v_{i+1}) \right) - \max_i c(v_i, v_{i+1})$$

(It is the sum of cost of all the edges - maximum cost edge)

Devise an algorithm to compute the smallest discounted cost path from a vertex s to vertex t. Give a succinct and precise description of your algorithm. Proof of correctness and run-time analysis are not necessary. The runtime of your algorithm should be $O((|V| + |E|) \log |V|)$.

Solution: Minimizing the discounted cost of a path is the same as minimizing the cost of a path wherein one edge can be skipped at zero cost.

Since we are looking for paths with exactly one "skipped edge", we can create a layered graph G' as follows:

For each vertex $v \in V$, there would be two copies v_0, v_1. For each edge $u \to v$, add edges $u_0 \to v_0$, $u_0 \to v_1$ and $u_1 \to v_1$.

Weights of the edges are

$$c(u_0 \to v_0) = c(u_1 \to v_1) = c(u \to v)$$

and

$$c(u_0, v_1) = 0$$

Essentially G' consists of two separate copies of G, with zero-weight ”skip” edges from one copy to other. We construct the graph G' and use Djikstra's algorithm to find the shortest path from s_0 to t_1.

12 NP-Completeness Reductions (30 points)

You may assume that the following problems are NP-complete: Rudrata (Hamiltonian) Path, Rudrata (Hamiltonian) Cycle, Vertex Cover, Independent Set, 3-Coloring, 3-SAT and Integer Programming.

For each of the following problems, fill in the details of the proof that they are NP-complete.

1. **TriClique:** Given a graph $G = (V, E)$, a subset $S \subset V$ of vertices is called a **clique**, if every pair of vertices in S are connected by an edge, i.e., for every $u, v \in S$ we have $(u, v) \in E$.

   Here is the **TriClique** problem.

   Input: A graph $G = (V, E)$

   Solution: A partition of the vertices into three disjoint sets $S_1 \cup S_2 \cup S_3 = V$ such that, each of the sets S_1, S_2, S_3 is a *clique*.

   Proof. *It is clear that the TriClique 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 3-Coloring *we will construct an instance Ψ of the problem* TriClique *as follows.*

   Given an instance $G = (V, E)$ of $3 - Coloring$, define an instance $G' = (V', E')$ of TriClique as follows: Set $V' = V$. For each pair (u, v) define,

$$(u, v) \in E' \leftrightarrow (u, v) \notin E$$

   (In other words, edge (u, v) exists in E' if and only if it does not exist in E.

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

Given a 3-coloring of the graph $G = (V, E)$, let S_1, S_2, S_3 be the set of vertices colored red, blue and green respectively. Within each color class, there are no edges in E. Therefore, the sets S_1, S_2, S_3 are cliques in the graph $G' = (V', E')$.

Conversly, suppose S_1, S_2, S_3 are cliques in $G' = (V', E')$. Then, consider the coloring that assigns vertices in these sets red,blue and green respectively. Since each S_i is a clique in G', there are no edges within any S_i. Hence this is a valid 3 coloring of graph G.

2. **Set Ordering:**

Input: A family of subsets $S_1, \ldots, S_n$ of $\{1, \ldots, m\}$.

Solution: An ordering of the sets so that consecutive sets in the ordering have intersection exactly one. Formally, an ordering of the sets is given by a permutation $\pi : [n] \to [n]$, so that for each $i \in \{1, \ldots, n-1\}$, $|S_{\pi(i)} \cap S_{\pi(i+1)}| = 1$.

Proof. It is clear that the Set-Ordering 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 *undirected Rudrata Path* *we will construct an instance Ψ of the problem* *Set Ordering* *as follows.*

Given an instance $G = (V, E)$ of Rudrata Path, define an instance of Set Ordering as follows: Let the universe be the set of edges E and we will have each S_i be a subset of this univers. For each vertex

$i \in V$, define

$$S_i = \{ \text{ edges incident at vertex } i\}$$

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

Suppose there is a Rudrata path in the original graph, then it gives an ordering $\pi : [n] \to [n]$ of the vertices of the graph. The corresponding ordering of sets satisfies the condition, since $S_{\pi(i)} \cap S_{\pi(i+1)}$ contains precisely the edge from the vertex $\pi(i)$ to vertex $\pi(i+1)$.

Conversly, suppose there is an ordering $\pi : [n] \to [n]$ of sets. Since $S_{\pi(i)} \cap S_{\pi(i+1)} = 1$ then, it means that the set of edges incident at vertices $\pi(i)$ and $\pi(i+1)$, have exactly one common edge. This implies that $\pi(i) - -\pi(i+1)$ is an edge of the graph. Hence $\pi(1) - -\pi(2) - - \ldots \pi(n)$ is a Rudrata path in the original graph.

13 All Pairs Shortest Paths (15 points)

In the lecture, we have seen the Floyd-Warshall algorithm for computing distances between every pair of vertices in an input graph. The Floyd-Warshall algorithm is a dynamic programming based algorithm with runtime $O(n^3)$ In this question, we will devise a simpler dynamic programming based algorithm for all pairs shortest paths, with a slightly worse runtime.

Given an input graph $G = (V, E)$ with positive edge weights w_{ij} for each edge (i, j). Consider the following sub-problem:

$$D[i, j, t] = \textit{length of the shortest path from } i \textit{ to } j \textit{ with at most } 2^t \textit{ hops}$$

(Here *"hop"* refers to the number of edges on the path).

1. Write the recurrence relation for $D[i, j, t]$.

Solution:

$$D[i, j, t] = \min_{k \in V} \{D[i, k, t-1] + D[k, j, t-1]\}$$

2. Write pseudocode for the algorithm to compute all-pair distances, including the initialization.

Solution:

```
For $i, j \in V$
$D[i,j] = \begin{cases} w_{ij} & \text{if } i \to j \in E \\ \infty & \text{otherwise} \end{cases}$
For $t = 1$ to $\lceil \log n \rceil$
For $i$ in $V$
For $j$ in $V$
For $k$ in $V$
```

$$D[i,j,t] = \min_{k \in V} \{D[i,k,t-1] + D[k,j,t-1]\}$$

return $D[i,j,\lceil \log n \rceil]$ for all i, j.

3. What is the runtime of your algorithm? $\boxed{}$ **Solution:** $\Theta(n^3 \log n)$ where $n = |V|$

14 Triangle Removal (20 points)

In the triangle removal problem, the input is a graph $G = (V, E)$. The goal is to remove the smallest number of edges from E, so as to delete all triangles in the graph.

Formally, here is the definition of triangle-freeness.

Definition: A graph is *triangle-free* if there are NO three vertices $i, j, k \in V$ such that (i, j), (j, k) and (k, i) are all edges.

Here is the formal definition of triangle removal problem:

Input: An unweighted undirected graph $G = (V, E)$

Solution: Find the smallest set of edges E' such that deleting E' makes the graph become *triangle-free*

We will now develop an approximation algorithm for the problem. First, write a linear programming relaxation for the triangle-removal problem.

1. What are the variables of the linear program? (Write out succinctly and precisely, the intended meaning of the variables in your linear program)

   Solution: x_e for each edge $e \in E$. The intended meaning is that x_e is the amount of edge e that is deleted.

2. What is the objective function?

   Solution: Minimize $\sum_{e \in E} x_e$.

3. What are the constraints of the linear program?

   Solution: $0 \leq x_e \leq 1$ for all $e \in E$. $x_a + x_b + x_c \geq 1$ for all triangles a, b, c.

4. Describe a rounding scheme to obtain a $\{0, 1\}$-solution from the optimal solution to the above linear program.

   Solution: If $x_e \geq 1/3$, round it to 1. Otherwise round it to 0.

   E' is the set of edges e with $x_e = 1$.

5. Briefly justify why the rounding scheme produces a solution to the Triangle Removal problem.

   Solution: For any triangle (a, b, c) in G, since $x_a + x_b + x_c \geq 1$, at least one of the edges $e \in \{a, b, c\}$ has $x_e \geq 1/3$, so we delete at least one of the edges in the triangle. So there cannot be any triangle in E'.

6. What is the approximation ratio of your algorithm?

   Solution: 3, because x_e is scaled up by a factor of at most 3 during rounding.

Use this page for rough work

Use this page for rough work

Use this page for rough work