Do not open this quiz booklet until you are directed to do so. Read all the instructions first.

Write your name and user id (as indicated on T-square) on the top of every page, including the almost blank page at the end.

You have 170 minutes to earn up to $1 + 30 + 2 = 33$ points, the test is graded out of 30.

This booklet contains 7 questions on 15 pages, including this one. You can use the back of the pages for scratch work.

Do not spend too much time on any one problem. Generally, twice a problem’s point value is an indication of how many minutes to spend on it.

Write your solutions in the space provided. If you run out of space, continue your answer on the back of the same sheet and make a notation on the front of the sheet.

You may use a sheet with notes on both sides.

You may use a calculator, but not any device with transmitting functions, especially ones that can access the wireless or the Internet.

You may use any of the theorems/facts/lemmas/algorithms/home-works that we covered in class without re-proving them unless explicitly stated otherwise.

If necessary make reasonable assumptions but please be sure to state them clearly.

When we ask you to give an algorithm in this test, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.

Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Title</th>
<th>Points</th>
<th>Parts</th>
<th>Grade</th>
<th>Initial of Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Name &amp; ID on top of every page</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Recursion &amp; Master Theorem</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Word Ladder</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Shortest Path Linear Program</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NP-Hardness of Maximum Clique</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>NP-Completeness of Subgraph Isomorphism</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Approximation of 3-Matching</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Approximation of Edge Coloring</td>
<td>4+2</td>
<td>2+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
0 ( /1 point) Write your name and user id on top of every page.

1. ( /6 points) Recursion & Master Theorem

For the three recursive programs below, give their runtime recurrence, and calculate their running time using Master theorem.

(a) ( /2 points)

```
PARTA(n) {
    Steps that cost O(1)
    PARTA(n/4)
    Steps that cost O(1)
    PARTA(n/4)
    Steps that cost O(1)
    PARTA(n/4)
}
```

**SOLUTION:**

\[ T(n) = 3T(n/4) + O(1), \]

first case of Master theorem, \( O(n^{\log_4 3}) \).

(b) ( /2 points)

```
PARTB(n) {
    Steps that cost O(1)
    PARTB(n/2)
    Steps that cost O(n)
    PARTB(n/2)
    Steps that cost O(n^2)
    PARTB(n/2)
}
```

**SOLUTION:**

\[ T(n) = 3T(n/2) + O(n^2), \]

third case of Master theorem, \( O(n^2) \).

(c) ( /2 points)

```
PARTC(n) {
    Steps that cost O(1)
    PARTC(n/3)
    Steps that cost O(n)
    PARTC(n/3)
}
```
Steps that cost $O(n)$
\[
\text{PART C}(n/3)
\]
\}

**SOLUTION:**

\[
T(n) = 3T(n/3) + O(n),
\]

second case of Master theorem, $O(n \log n)$.

**GRADING:**

1 point each for recurrence and solution.

A maximum of $-1$ for wrong recurrence propagating into wrong outcome. Otherwise, use of Master theorem marked only based upon the recurrence produced.
2. ( 4 points) Word Ladder

Consider the following problem on a dictionary of \( n \) words, \( W_1 \ldots W_n \), each with exactly \( k \) characters.

You can transform a word \( W_i \) into word \( W_j \) if they differ in at most \( d \leq k \) characters. (Both \( d \) and \( k \) are specified as part of the input, along with \( n \) and the words)

For example, if the dictionary is

\[
W_1 = \text{hit}, W_2 = \text{cog}, W_3 = \text{hot}, W_4 = \text{dot}, W_5 = \text{dog}, W_6 = \text{lot}, W_7 = \text{log},
\]

and \( d = 1 \), one way to change \( \text{hit} \) to \( \text{cog} \) is:

\[
\text{hit} \rightarrow \text{hot} \rightarrow \text{dot} \rightarrow \text{dog} \rightarrow \text{cog}.
\]

We want to find the fewest number of steps to transform \( W_1 \) to \( W_2 \).

(a) ( /2 points) Formulate this problem as a shortest path problem.

**SOLUTION:**
Create one vertex \( v_i \) for each word \( W_i \).
Place an edge from \( i \) to \( j \) if \( W_i \) can be transformed into \( W_j \) by changing at most \( d \) characters.

**GRADING:**
1 point for identifying the words as vertices.
1 point for getting the edges right.

(b) ( /2 points) Show an \( O(n^2 k + n^3) \) time algorithm for finding the minimum number of steps (you only need to describe how to find the number of steps, not the sequence of transformations).

**SOLUTION:**
Run shortest path (Bellman-Ford algorithm) on the graph constructed in part a).
The graph in part a) has \( n \) vertices, \( n^2 \) edges, so Bellman-Ford algorithm takes \( O(n^3) \) time. Furthermore, the graph can be constructed by comparing every pair of words, giving \( O(n^2 k) \).

**GRADING:**
1 point for mentioning any shortest path algorithm.
1 point for correctly analyzing the running time of the shortest path part of the algorithm.

Due to an incorrect clarification given at the start of the CoB location of the exam, we are not grading for the running time of constructing the graph.
3. (4 points) Shortest Path Linear Program

Recall the problem of finding the shortest path from \( s \) to \( t \) in a directed graph. We want to formulate this problem on the following graph (with lengths written above the edges) as a linear program.

Note that the labels on edges are their lengths, not capacities. Formally the edges are:

- \( s \rightarrow a \) with length 2,
- \( s \rightarrow b \) with length 3,
- \( a \rightarrow t \) with length 4,
- \( b \rightarrow t \) with length 1,
- \( a \rightarrow b \) with length 1.

To formulate this linear program, we create one variable per edge representing whether it’s used in the path, leading to 5 variables: \( x_{s \rightarrow a} \), \( x_{s \rightarrow b} \), \( x_{a \rightarrow b} \), \( x_{a \rightarrow t} \), and \( x_{b \rightarrow t} \). Specifically, we want \( x_e \) to be 0 if \( e \) is not used in the path, and 1 if it is. So we put in the constraints

\[
0 \leq x_{s \rightarrow a} \leq 1,
0 \leq x_{s \rightarrow b} \leq 1,
0 \leq x_{a \rightarrow b} \leq 1,
0 \leq x_{a \rightarrow t} \leq 1,
0 \leq x_{b \rightarrow t} \leq 1.
\]

In the rest of this problem, we will complete this linear program for the graph above.

(a) (1 point) Give the objective function that’s the length of the path, using the lengths given in the graph above.

**SOLUTION:**

\[
\min 2x_{s \rightarrow a} + 3x_{s \rightarrow b} + 1x_{a \rightarrow b} + 4x_{a \rightarrow t} + 1x_{b \rightarrow t}
\]
GRADING:
1/2 point for a linear combination of the $x$s, 1/2 point for the right coefficients.
−1/2 for not having min or max.

(b) ( /1 point) Give the two constraints that specify we must use 1 edge leaving $s$, and 1 edge entering $t$.
SOLUTION:

\[
\begin{align*}
x_{s\rightarrow a} + x_{s\rightarrow b} &= 1 \\
x_{a\rightarrow t} + x_{b\rightarrow t} &= 1 
\end{align*}
\]

GRADING:
1/2 for each constraint on $s$ and $t$.

(c) ( /1 point) Give the two constraints that specify at both $a$ and $b$, the number of entering edges that we use must equal to the number of leaving edges that we use.
SOLUTION:

\[
\begin{align*}
x_{s\rightarrow a} &= x_{a\rightarrow t} + x_{a\rightarrow t} \\
x_{s\rightarrow b} + x_{a\rightarrow b} &= x_{b\rightarrow t} 
\end{align*}
\]

GRADING:
1/2 for each constraint on $a$ and $b$.

(d) ( /1 point) Give a fractional optimum solution for this linear program (discussed at the start of this problem, and in parts a, b, and c). That is, its cost (w.r.t. the cost function from part a) should be 4 (the length of the shortest $s \rightarrow t$ path here), but also have $x_e$ strictly between 0 and 1 for some edge $e$.
SOLUTION:
One solution is:

\[
\begin{align*}
x_{s\rightarrow a} &= 1/2, \\
x_{s\rightarrow b} &= 1/2, \\
x_{a\rightarrow b} &= 1/2, \\
x_{a\rightarrow t} &= 0, \\
x_{b\rightarrow t} &= 1.
\end{align*}
\]

GRADING:
1 point if this works. 1/2 if it’s a fractional solution and meets $0 \leq x_e \leq 1$ on all edges. 0 otherwise. (this is meant to be the hardest pre-NP-hardness question.)
4. ( 4 points) NP-Hardness of Maximum Clique

Recall the CLIQUE problem: given a graph $G$ and a value $k$, check whether $G$ has a set $S$ of $k$ vertices that’s a clique. A clique is a subset of vertices $S$ such that for all $u, v$ in $S$, $uv$ is an edge of $G$.

The goal of this problem is to establish the NP-hardness of CLIQUE by reducing VERTEXCOVER, which is itself an NP-hard problem, to CLIQUE. Recall that a vertex cover is a set of vertices $S$ such that every edge $uv$ has at least one endpoint ($u$ or $v$) in $S$, and the VERTEXCOVER problem is given a graph $H$ and a value $l$, check whether $H$ has a vertex cover of size at most $l$.

Note that all these problems are already phrased as decision problems, and you only need to show the NP-Hardness of CLIQUE. In other words, we will only solve the reduction part in this problem, and you DO NOT need to show that CLIQUE is in NP.

(a) (2 points) Let $S$ be a subset of vertices in $G$, and let $C$ be the complement graph of $G$ (where $uv$ is an edge in $C$ if and only if $uv$ is not an edge in $G$).

Prove that for any subset of vertices $S$, $S$ is a vertex cover in $G$ if and only if $V \setminus S$ is a clique in $C$.

Note that this is an if and only if proof, i.e. you need to show both directions for full credit. Each direction is worth 1 point.

SOLUTION:
If $S$ is a vertex cover in $G$, then for any $u, v \notin S$, the edge $uv$ is not in $E(G)$. This means $uv$ is an edge in $C$. So $V \setminus S$ is a clique.

If $V \setminus S$ is a clique in $C$, then for any $u, v \notin S$, $uv$ is an edge in $C$. So $uv$ is not an edge in $G$.

GRADING:
1 point for either direction.

(b) (2 points) Part a) implies the following result (which you may use without proof): $G$ has a vertex cover of size at most $k$ if and only if the complement of $G$ has a clique of size at least $n - k$.

Use this fact to give a reduction from VERTEXCOVER to CLIQUE. Your solution should have the following two steps, each of which is worth 1 point:

• First, show the reduction: specify how the inputs to VERTEXCOVER, $G$ and $k$, can be transformed to a valid input pair, $H$ and $l$, for CLIQUE. Make sure to explain why this takes polynomial time.

• Second, show that the answer to CLIQUE$(H, l)$ can be converted to the answer of VERTEXCOVER$(G, k)$. One possibility is to explain how a YES answer to CLIQUE$(H, l)$ must also mean YES to VERTEXCOVER$(G, k)$, AND a NO answer to VERTEXCOVER$(H, l)$ must also mean NO to CLIQUE$(G, k)$.

(Hint: The claim we proved in part a) is an if and only if statement.)
**SOLUTION:**
Let the conversion be:

\[ H \leftarrow \text{Complement} (G) \]
\[ l \leftarrow n - k \]

This takes \( O(n^2) \) time since we just go through all edges of \( G \) once.
Then by part a), \( \text{VERTEXCOVER}(G,k) \) is true iff \( \text{CLIQUE}(H,k) \) is true. So just returning the answer to \( \text{CLIQUE}(H,k) \) completes the reduction.

**GRADING:**
1/2 point for transformation, 1/2 point for its runtime. 1 point for completing the proof using part a), 1/2 point each for the \( YES \) and \( NO \) cases.
5. (4 points) NP-Completeness of Subgraph Isomorphism

Consider the subgraph isomorphism problem: given graphs $G$ and $H$, $\text{SUBGRAPHISO}(G, H)$ returns true if the vertices of $G$ can be mapped to distinct vertices so that $G$ becomes a subgraph of $H$. Formally, we want to look for a mapping

$$\pi : V(G) \rightarrow V(H)$$

so that:

(a) For two distinct vertices of $G$, $u \neq v$, $\pi(u) \neq \pi(v)$, and

(b) If $uv$ is an edge in $G$, $\pi(u)\pi(v)$ is an edge in $H$.

As an example, consider the graph $H$ on 4 vertices:

\[ \text{The length 4 path } G_1: \]

\[ \text{is subgraph isomorphic to } H \text{ via the mapping} \]

$$\pi(a) = w, \quad \pi(b) = u, \quad \pi(c) = v, \quad \pi(d) = x.$$  

Furthermore, since we only need to map the vertices of $G$ to $H$, and are ok with subgraphs, the length 3 path is also subgraph isomorphic to $H$.

On the other hand, the bipartite graph with 2 vertices on each side, $G_2$:

\[ \text{is not subgraph isomorphic to } H. \text{ In other words, } \text{SUBGRAPHISO}(G_2, H) \text{ should return ‘NO’.} \]

This is because each vertex in $G_2$ has degree at least 2, but $x$ in $H$ only has degree 1.
Note that the examples on the previous page are just examples. Your solutions need to work for any graphs $G$ and $H$, not just these particular examples.

(a) (\hspace{1cm}2 points) Show that $\text{SubgraphIsomorphism}(G, H)$ is in NP. **SOLUTION:**

We can build a poly-time verifier that takes a mapping $\pi$ as input.

We can check whether $\pi(u)$ is unique for all $u \in V(G)$ in $O(n)$ time.

**GRADING:**

1 point for specifying taking an assignment as input.

1 point for describing the verification. $-1/2$ point for not justifying that this verification takes poly time.

(b) (\hspace{1cm}2 points) We will show NP-hardness of $\text{SubgraphIso}(G, H)$ by reducing from the problem $\text{Clique}$ (described at the start of problem 4). That is, we want to give a way to solve $\text{Clique}$ using a routine that solves $\text{SubgraphIso}$. Your solution should have the following two steps, each of which is worth 1 point:

- First, show the reduction: make sure to specify how the inputs to $\text{Clique}$, $G$ and $k$, can be converted in polynomial time to a valid input for $\text{SubgraphIsomorphism}(G, H)$. A diagram would be helpful, although not necessary to get full credit.

- Second, show that the answer to $\text{SubgraphIso}(G, H)$ equals to the answer to $\text{Clique}(G, k)$. One way to do this is by showing that the YES answer to $\text{SubgraphIso}(G, H)$ can be used to answer YES to $\text{Clique}(G, k)$, AND that the NO answer to $\text{SubgraphIso}(G, H)$ can be used to answer NO to $\text{Clique}(G, k)$.

**SOLUTION:**

Let $G_k$ be the graph that’s a clique on $k$ vertices.

$\text{SubgraphIso}(G_k, G)$ returns true if and only if $G_k$ is a subgraph of $G$. This is the same as $G$ containing a clique of size $k$. So we have

$$\text{Clique}(G, k) = \text{SubgraphIso}(G_k, G).$$

**GRADING:**

1 point for putting $G$ as clique, and $H$ as $G$.

1/2 point for conversion of answer.

1/2 point for poly time of reduction.
6. ( /4 points) Approximation of 3-Matching

Consider the 3-matching problem, which is a generalization of matching where ‘edges’ involve three vertices each.

We have n vertices and a collection T of m triples, T₁, ..., Tₘ. Each triple has three (distinct) vertices

\[ T_i = \{u_i, v_i, w_i\} \, . \]

A 3-matching is a set of triples that contain each vertex at most once. We would like to find a 3-approximation to the maximum 3-matching. (Aside: finding the maximum 3-matching can be shown to be NP-Hard.)

Consider a maximal matching among these triples: This is a set of triples so that we cannot add another triple to it.

For example, if we have 6 vertices 1...6, and three triples are

\[ T_1 = \{2, 3, 4\} \]
\[ T_2 = \{1, 2, 3\} \]
\[ T_3 = \{4, 5, 6\} \]

then the set \{T₁\} is a maximal 3-matching because we cannot add either T₂ or T₃ to it. On the other hand, the maximum 3-matching \{T₂, T₃\} has size 2.

(a) ( /2 points) Give a set of triples with a maximum 3-matching of size 3, but has a maximal 3-matching of size 1.

**SOLUTION:**

There are many examples, but one such example is:

\[ T_1 = \{2, 5, 8\} \]
\[ T_2 = \{1, 2, 3\} \]
\[ T_3 = \{4, 5, 6\} \]
\[ T_4 = \{7, 8, 9\} \]

**Comment:** The solution must have 3 sets Tᵢ, Tⱼ, Tₖ such that, Tᵢ ∩ Tⱼ ∩ Tₖ = ∅, since this is the definition of a matching. There must also be some set Tᵢ which contains one element from each triple.

**GRADING:**

1 point for maximal matching having size 1.
1 point for maximum matching having size 3.

(b) ( /2 points) Show that the algorithm of returning a maximal 3-matching is a 3-approximation. That is, let M be a maximal 3-matching, show that no 3-matching can have size more than 3|M| (3 times the size of M).
**SOLUTION:**
Let $M^*$ be the maximum 3-matching.

Since $M$ is maximal, we cannot add any triple from $M^*$ to it. This means each triple in $M^*$ must have at least one vertex in common with some vertex in $M$.

The definition of 3-matchings means that all triples in $M^*$ contain disjoint vertices. So the size of $M^*$ is limited by the number of vertices in $M$, which is $3|M|$.

**GRADING:**
1 point for counting the number of vertices in $M$, or saying that each triple in $M$ excludes at most 3 triples in $M^*$.
1 point for relating that to the size of $M^*$. 
7. (4 + 2 points) Approximation of Edge Coloring

An edge-coloring of an undirected graph $G$ is an assignment of labels to the edges so that any two edges that share a vertex have different colors.

The edge-coloring problem is to color the edges of a graph with the fewest number of colors. Below is an example of an edge-coloring with 3 colors, which is optimum for this graph. The colors are labeled 1, 2, and 3.

(Note from TAs who have test-solved this question: parts a) and b) of this question are much simpler than they look.)

(a) (2 points) Show that if a graph $G$ has a vertex with degree $d$, then at least $d$ colors are necessary to edge-color $G$.

Note in particular this shows that 3 colors is optimum for the example above: the vertex $b$ (which is the vertex of maximum degree) has degree 3. However, your proof should work for any graph $G$.

**SOLUTION:**

Given a vertex $v$ in graph $G$ with incident edges $e_1, ..., e_d$, then each edge $e_1, ..., e_d$ must be a different color in any valid edge-coloring. Therefore, there must be at least $d$ colors used in any valid edge-coloring of $G$.

**GRADING:**

Points were given based on how along these lines the solutions are.

(b) (2 points)

Vizing’s theorem implies that there is a polynomial time routine, $\text{VISINGALGO}$, that colors all edges of a graph with maximum degree $\Delta$ with at most $\Delta + 1$ colors. You do not need to derive how such an algorithm works.

Instead, use $\text{VISINGALGO}$ to give a 2-approximation to the edge coloring problem. That is, give an algorithm $A$ such that the number of colors that it uses to edge-color a graph $G$, $A(G)$, always satisfies

$$A(G) \leq 2 \cdot \text{opt}(G),$$
where $opt(G)$ is the minimum number of colors needed to edge-color $G$.

You may assume that $\Delta \geq 1$, that is, the graph has at least 1 edge.

**SOLUTION:**

Run VIZINGALGO on the graph and output it's edge coloring.

From part a), we know that $OPT$ is at least as large as the degree of any vertex in the graph, so we must have $OPT \geq \Delta$. Our output from VIZINGALGO uses at most $\Delta + 1$ colors, and $\Delta + 1 \leq 2\Delta$ when $\Delta \geq 1$, which was assumed. Therefore $\Delta + 1 \leq 2 \cdot OPT$, and VIZINGALGO is a 2-approximation algorithm.

**GRADING:**

1 point for algorithm.

1 point for justification.

(c) (Bonus, /2 points)

**Note:** This is by far the hardest question on this test. Do not attempt it unless you have completed everything else, and checked your solutions once.

Give a polynomial time algorithm for edge-coloring and prove that it gives a 10-approximation to edge-coloring. Note that you cannot assume access to VIZINGALGO for this part unless you can derive it.

**Hint:** Vizing’s algorithm is far more complicated than the solutions that we have.

**SOLUTION:**

For $i = 1$ to $2\Delta$, find a maximal matching of the graph, color each edge of the matching with color $i$ and remove it from the graph.

Now we need to show that our algorithm gives a valid edge-coloring of the graph. By construction, the only edges that are colored the same color are those within the same matching, which implies by definition that none of them share an incident vertex. It then remains to show that our algorithm will color every edge. Suppose after $2\Delta$ steps that some edge $e = (u, v)$ still has not been colored. We know that the degree of $u$ and $v$ are both at most $\Delta$. Furthermore, for each maximal matching chosen, if edge $e$ is not in the matching then either $u$ or $v$ must have an incident edge in the matching because otherwise we could add $e$ to the matching, contradicting it’s maximality. Our algorithm then removes each edge in the matching from the graph, so at each step the degree of either $u$ or $v$ must be reduced by 1. Therefore, after $2\Delta - 2$ iterations, if edge $e$ is still remaining in our graph, it must be the only edge incident to $u$ or $v$, and must be chosen in the next maximal matching. This then contradicts $e$ not being colored after $2\Delta$ steps, and we conclude that all edges must be colored by our algorithm.
Finally, we show that this implies that our algorithm gives a 2-approximation. From part b) we know that $OPT \geq \Delta$, so if we can color the graph with $2\Delta$ colors, then our algorithm must use $\leq 2 \cdot OPT$ colors.

**GRADING:**
1 point for algorithm.
1 point for justification.