

Final Examination

- Do not open this exam booklet until you are directed to do so.
- This exam ends at 4:30 P.M. It contains 8 problems, some with several parts. You have 180 minutes to earn 160 points.
- This exam is closed book, but you may use two double-sided $8\frac{1}{2}'' \times 11''$ or A4 crib sheets.
- When the exam begins, write your name in the space below and on the top of every page in this exam. Circle your recitation instructor.
- Write your solutions in the space provided. If you need more space, use the scratch paper at the end of the exam booklet. Please write your name on any extra pages that you use.
- **Do not spend too much time on any one problem.** Read them all through first, and attack them in the order that allows you to make the most progress.
- Do not waste time rederiving algorithms and facts that we have studied. It suffices to cite known results.
- Show your work, as partial credit will be given. You will be graded on the correctness and efficiency of your answers and also on your clarity. Please be neat.
- When giving an algorithm, sketch a proof of its correctness and analyze its running time using an appropriate measure.
- Good luck!

Problem	Points	Grade	Initials
1	48		
2	33		
3	15		
4	10		

Problem	Points	Grade	Initials
5	10		
6	15		
7	10		
8	19		

Total	160		
--------------	-----	--	--

Name: _____

Circle your recitation instructor:

TB (F10, F11) Ammar (F12,F1) Angelina (F2,F3)

Problem 1. True/False and Justify [48 points] (12 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why. Your justification is worth more than your true or false designation.

- (a) **T F** [4 points] If a sequence of n operations on a data structure cost $T(n)$, then the amortized runtime of each operation in this sequence is $T(n)/n$.

Solution: True. This is the aggregate method.

- (b) **T F** [4 points] Fix some integers $m \gg n > 0$. For every function $h : [m] \rightarrow [n]$ in a universal hashing family \mathcal{H} , there exists an integer $0 \leq i \leq m - 1$ such that $h(i) \neq 0$.

Solution: If \mathcal{H} is the set of all functions (which is a universal hashing family), then h can be a zero function.

- (c) **T F** [4 points] If a problem in NP can be solved in polynomial time, then it is known that all problems in NP can be solved in polynomial time.

Solution: False. Any problem in P is also in NP and can be solved in polynomial time. However, it is not known that all problems in NP can be solved in polynomial time.

- (d) **T F** [4 points] If $P = NP$, then every nontrivial decision problem $L \in P$ is NP-complete. (A decision problem L is *nontrivial* if there exist some x, y such that $x \in L$ and $y \notin L$.)

Solution: True. If $P = NP$, then every NP-complete problem reduces in polynomial time from any other problem in NP, by solving the problem and then outputting a canonical yes or no instance.

- (e) **T F** [4 points] A spanning tree of a given undirected, connected graph $G = (V, E)$ can be found in $O(E)$ time.

Solution: True. We can simply use BFS (breath-first search) to achieve this. Note that the question asks about finding a spanning tree, not a minimum spanning tree.

- (f) **T F** [4 points] Consider the following algorithm for computing the square root of an n -bit integer x :

```
SQUARE-ROOT( $x$ )
  For  $i = 1, 2, \dots, \lfloor x/2 \rfloor$ :
    If  $i^2 = x$ , then output  $i$ .
```

This algorithm runs in polynomial time.

Solution: False. To run in polynomial-time, this algorithm would have to run in time polynomial in $\lg x$ (input size). This algorithm uses $\Theta(x)$ multiplications in the worst case.

- (g) **T F** [4 points] If all edge capacities in a flow network are integer multiples of 3, then the maximum flow value is a multiple of 3.

Solution: True. Consider the minimum cut. It is made up of edges with capacities that are multiples of 3, so the capacity of the cut (sum of capacities of edges in the cut) must be a multiple of 3. By the Maxflow-Mincut theorem, the maximum flow has the same value.

- (h) **T F** [4 points] Given a connected directed graph $G = (V, E)$ and a source vertex $s \in V$ such that each every $e \in E$ has an integer weight $w(e) \in \{0, 1, \dots, V^3\}$, there is an algorithm to compute single-source shortest-path weights $\delta(s, v)$ for all $v \in V$ in $O(E \lg \lg V)$ time.

Solution: True. We can achieve this by modifying the Dijkstra algorithm to use Van Emde Boas data structure in place of the priority queue.

- (i) **T F** [4 points] Given a constant $\varepsilon > 0$, probabilistic property testing whether a sequence is ε -close (as defined in the lecture) to monotone requires $\Omega(n)$ queries.

Solution: False. It can be done in $\Theta(\log n)$ queries.

- (j) **T F** There is a sublinear-time algorithm that decides whether a given undirected graph is connected.

Solution: False; in order to decide whether a undirected graph is connected, all nodes in the graph must be checked.

- (k) **T F** [4 points] An adversary can force a skip-list insertion to take $\Omega(n)$ time.

Solution: False. An adversary cannot influence the outcome of coin tosses.

- (l) **T F** [4 points] Assume $P \neq NP$. The Traveling Salesman Problem has a polynomial-time α -approximation algorithm for some constant $\alpha > 1$.

Solution: False. There is a polynomial-time approximation algorithm for metric TSP, but no approximation algorithm for the general case (consider the hard special case where there is a 0-weight cycle) assuming $P \neq NP$.

Problem 2. Short answer [33 points] (5 parts)

Give brief answers to the following problems.

- (a) [9 points] Match up each application with an algorithm or data structure that we used to solve it in this course. Use each answer exactly once.

<input type="checkbox"/>	Map folding	A. Polynomial reduction
<input type="checkbox"/>	Integer multiplication	B. Ford-Fulkerson algorithm
<input type="checkbox"/>	Finding a minimum spanning tree	C. Dynamic programming
<input type="checkbox"/>	All-pairs shortest paths	D. General matching
<input type="checkbox"/>	Polynomial identity testing	E. Divide and conquer
<input type="checkbox"/>	SubsetSum is NP-hard	F. Johnson algorithm
<input type="checkbox"/>	Chinese postman tour	G. Dijkstra
<input type="checkbox"/>	Finding a minimum cut	H. Greedy algorithm
<input type="checkbox"/>	Single-source shortest paths	I. Monte Carlo algorithm

Solution:

<i>C.</i> Map folding	A. Polynomial reduction
<i>E.</i> Integer Multiplication	B. Ford-Fulkerson algorithm
<i>H.</i> Finding a minimum spanning tree	C. Dynamic programming
<i>F.</i> All pair shortest path	D. General Matching
<i>I.</i> Polynomial identity testing	E. Divide and Conquer
<i>A.</i> SubsetSum is NP-hard	F. Johnson Algorithm
<i>D.</i> Chinese postman tour	G. Dijkstra
<i>B.</i> Finding Mincut	H. Greedy Algorithm
<i>G.</i> Single source shortest path	I. Monte Carlo algorithm

- (b) [6 points] Suppose that you are given an unsorted array A of n integers, some of which may be duplicates. Explain how you could “uniquify” the array (that is, output another array containing each unique element of A exactly once) in $O(n)$ expected time.

Solution: We use universal hashing to solve this problem. Create a hash table of $2n$ elements, and for each element x in A , search for x in the table and insert it only if the search fails. Then walk down the slots of the table and output every element. The searches and insertions each take $O(1)$ expected time, and walking down the table takes $O(n)$ time, for a total expected runtime of $O(n)$.

- (c) [6 points] Prove that there is no polynomial-time $(1 + \frac{1}{2n})$ -approximation algorithm for Vertex Cover (unless $P = NP$).

Solution: Prove by contradiction. Assume that there exists an $(1 + \frac{1}{2n})$ -approximation algorithm \mathcal{A} for Vertex Cover. For a given graph $G = (V, E)$, let S be a minimum vertex cover for G . Then \mathcal{A} can decide if G has a vertex cover of size at least

$$|S|/(1 + \frac{1}{2n}) \geq |S| \times (1 - \frac{1}{2n}) > |S| - 1$$

. Therefore, \mathcal{A} can solve Vertex Cover in polynomial time. Contradiction.

(d) [6 points] The following table gives the frequencies of the characters of an alphabet.

Character	Frequency
<i>A</i>	$1/20$
<i>B</i>	$2/20$
<i>C</i>	$2/20$
<i>D</i>	$4/20$
<i>E</i>	$4/20$
<i>F</i>	$7/20$

Show a tree that Huffman's algorithm could produce for these characters and frequencies, and fill in the table below with the codeword for each character in the alphabet produced by this tree.

Character	Codeword
<i>A</i>	
<i>B</i>	
<i>C</i>	
<i>D</i>	
<i>E</i>	
<i>F</i>	

Solution:

- (e) [6 points] A nonvertical line L in the plane can be represented by an equation $y = m_L x + b_L$ for real numbers m_L, b_L . A point $P = (x_P, y_P)$ is **above** a line L if $y_P \geq m_L x_P + b_L$.

Given n nonvertical lines L_1, L_2, \dots, L_n in the plane, describe how to find in linear time the point P of minimum y coordinate that is above all n lines.

Solution: LP.

Problem 3. Hadamard chronicles IV: Divide and conquer [15 points]

For each nonnegative integer k , the **Hadamard matrix** H_k is the $2^k \times 2^k$ matrix defined recursively as follows:

- $H_0 = [1]$.
- For $k > 0$, $H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array} \right]$.

Let \vec{v} be a column vector of length $n = 2^k$. Describe an algorithm that computes the product $H_k \vec{v}$ in $O(n \log n)$ arithmetic operations (additions, subtractions, multiplications, or divisions). Show that your algorithm achieves the stated complexity.

Solution:

HADAMARD-MULT(k, v)

```

1  if  $k = 0$ 
2    then
3      return  $v$ 
4    else
5      Let  $v_1, v_2$  be the first and second half of  $v$ . (each  $v_i$  has length  $n_i = 2^{k-1}$ ).
6       $a \leftarrow$  HADAMARD-MULT( $k - 1, v_1$ )
7       $b \leftarrow$  HADAMARD-MULT( $k - 1, v_2$ )
8      return  $((a + b)^T, (a - b)^T)^T$ 
```

Let $T(n)$ be the running time.

$$T(n) = 2T(n/2) + O(n)$$

By the Master Theorem case 2,

$$T(n) = O(n \log n)$$

Problem 4. Biggest. Cut. Ever. [10 points]

Given an undirected graph $G = (V, E)$ and two vertices $s, t \in V$, a **maximum s - t cut** is a cut (S, T) satisfying the following conditions:

- i. (S, T) is a cut: $S, T \subset V$, $S \cap T = \emptyset$, and $S \cup T = V$.
- ii. $s \in S$ and $t \in T$.
- iii. The number of edges $(u, v) \in E$ with $u \in S$ and $v \in V \setminus S$ is the maximum possible.

The MAXIMUM- s - t -CUT problem is to find a maximum cut for a given pair of vertices. Unlike its counterpart, the MINIMUM- s - t -CUT problem, MAXIMUM- s - t -CUT is NP-hard. Analyze the following algorithm and show that it is a 2-approximation algorithm for MAXIMUM- s - t -CUT.

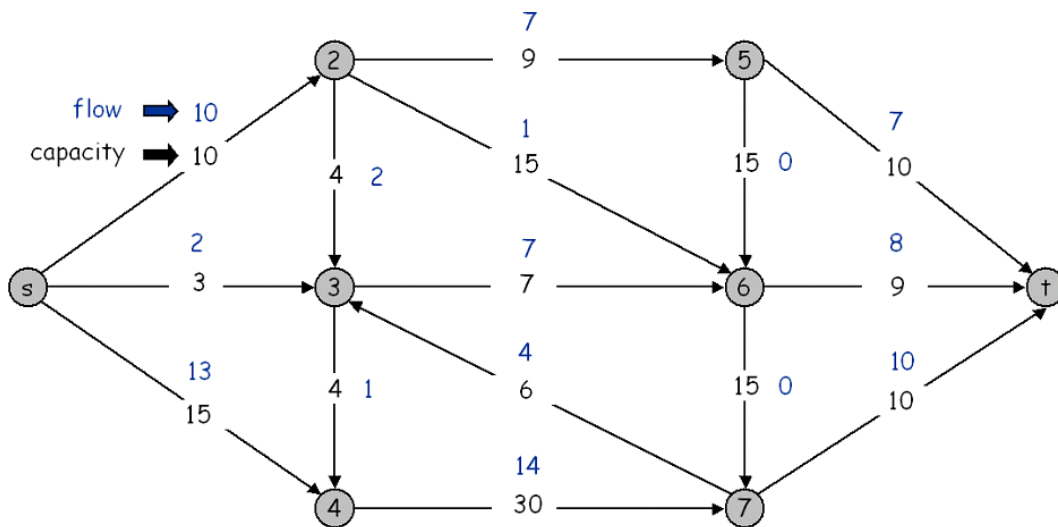
MAX-CUT(G, s, t)

```
1   $S \leftarrow \{s\}$ 
2   $T \leftarrow \{t\}$ 
3  for each vertex  $v \in V - \{s, t\}$ 
4      do
5           $a \leftarrow$  the number of edges  $(u, v)$  with  $u \in S$ .
6           $b \leftarrow$  the number of edges  $(v, w)$  with  $w \in T$ .
7          if  $a > b$ 
8              then
9                   $T \leftarrow T \cup \{v\}$ 
10             else
11                  $S \leftarrow S \cup \{v\}$ 
12 return  $(S, T)$  as the approximation for the MAXIMUM- $s$ - $t$ -CUT of  $s$  and  $t$ .
```

Solution: Each time we add a new vertex to the cut, the number of edges removed is smaller than the number of edges added. Therefore, the size of the cut is at least half the total number of edges.

Problem 5. Be the computer [10 points]

Starting from the following flow (printed above or to the right of the capacities), perform one iteration of the Edmonds-Karp algorithm.



- (a) [4 points] Write down your shortest augmenting path, that is, the augmenting path with the fewest possible edges.

Solution:

$$s \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow t$$

- (b) [3 points] Perform the augmentation. What is the value of the resulting flow?

Solution: 26.

- (c) [3 points] Is the resulting flow optimal? If so, give a cut whose capacity is equal to the value of the flow. If not, give a shortest augmenting path.

Solution: No.

$$s \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow t$$

Problem 6. Graphs and paths and cycles, oh my! [15 points]

Given a directed graph $G = (V, E)$, a **Hamiltonian path** is a path that visits each vertex in G exactly once. Consider the following properties for a directed graph G :

- $P_1(G)$: G contains either a cycle (not necessarily Hamiltonian) *or* a Hamiltonian path (or both).
- $P_2(G)$: G contains both a cycle (not necessarily Hamiltonian) *and* a Hamiltonian path.

Given that the problem HAM-PATH (which decides whether a graph G has a Hamiltonian path) is NP-complete, prove that one of the two properties above is decidable in polynomial time, while the other property is NP-complete.

Solution: Easy to check that both problems are in NP.

P_2 is NP-hard. Let us construct a graph $G' = (V', E')$ from G as follow.

$$V' = \{u_1, u_2, u_3\} \cup V$$

$$E' = \{(u_1, u_2), (u_2, u_3), (u_3, u_1)\} \cup \{(u_1, v) : v \in V\} \cup E$$

G' always has a cycle of length 3 - (u_1, u_2, u_3) . Also, if there exists any Hamiltonian path P in G , then (u_3, u_2, u_1, P) is also a Hamiltonian path in G' . Conversely, if P' is a Hamiltonian path in G' . Then P' must be has one the following patterns

$$(u_2, u_3, u_1, P), (u_3, u_2, u_1, P), (P, u_1, u_2, u_3), (P, u_1, u_3, u_2)$$

In any case, P is a Hamiltonian path in G . Therefore, solving HAM-PATH for G' is equivalent to solving HAM-PATH for G .

P_1 is decidable. Checking whether a graph has cycle can be done in polynomial time using DFS. If the graph has a cycle, accept and return. In case the graph does not have a cycle, checking if it has a Hamiltonian path is easy since the graph is acyclic.

Problem 7. If I only had a black box... [10 points]

Suppose you are given a magic black box that, in polynomial time, determines the *number of vertices* in the largest complete subgraph of a given undirected graph G . Describe and analyze a polynomial-time algorithm that, given an undirected graph G , computes a complete subgraph of G of maximum size, using this magic black box as a subroutine.

Solution: Let $A(G)$ be output of the black box on input G . The algorithm works as follow.

```
1   $S \leftarrow \emptyset$ 
2   $k = A(G)$ 
3  for each vertex  $v \in V$ 
4      do
5           $G' \leftarrow G \setminus \{v\}$ 
6          if  $A(G') < k$ 
7              then
8                   $S \leftarrow S \cup \{v\}$ 
9                   $k \leftarrow k - 1$ 
10         else
11              $G \leftarrow G'$ 
12 return the subgraph of  $G$  induced by  $S$ .
```

Problem 8. Hero Training [19 points]

You are training for the World Championship of Guitar Hero World Tour, whose first prize is a *real guitar*. You decide to use algorithms to find the optimal way to place your fingers on the keys of the guitar controller to maximize the ease by which you can play the 86 songs.

Formally, a **note** is an element of $\{A, B, C, D, E\}$ (representing the green, red, yellow, blue, and orange keys on the guitar). A **chord** is a nonempty set of notes, that is, a nonempty subset of $\{A, B, C, D, E\}$. A **song** is a sequence of chords: c_1, c_2, \dots, c_n . A **pose** is a function from $\{1, 2, 3, 4\}$ to $\{A, B, C, D, E, \emptyset\}$, that is, a mapping of each finger on your left hand (excluding thumb) either to a note or to the special value \emptyset meaning that the finger is not on a key. A **fingering** for a song c_1, c_2, \dots, c_n is a sequence of n poses p_1, p_2, \dots, p_n such that pose p_i places exactly one finger on each note in c_i , for all $1 \leq i \leq n$,

You have carefully defined a real number $D[p, q]$ measuring the difficulty of transitioning your fingers from pose p to q , for all poses p and q . The **difficulty** of a fingering p_1, p_2, \dots, p_n is the sum $\sum_{i=2}^n D[p_{i-1}, p_i]$. Give an $O(n)$ -time algorithm that, given a song c_1, c_2, \dots, c_n , finds a fingering p_1, p_2, \dots, p_n of the song with minimum possible difficulty.

Solution:

Dynamic programming. Subproblems $\text{OPT}(i, p_i)$ = the minimum possible difficulty for the song c_1, c_2, \dots, c_i and the i^{th} pose is p_i . There are $O(n) \cdot 6^4 = O(n)$ such subproblems.

Base case: $\text{OPT}(1, p_1) = 0$. For $i > 1$, in order to compute $\text{OPT}(i, p_i)$, we guess pose p_{i-1} (there are $6^4 = O(1)$ choices):

$$\text{OPT}(i, p_i) = \min\{\text{OPT}(i-1, p_{i-1}) + D[p_{i-1}, p_i] \mid p_{i-1} \text{ is a valid pose for } c_{i-1}\}$$

Running time is $O(n)$ since there are $O(n)$ subproblems and it takes $O(1)$ to compute the solution for each of them.

Other solution : You could define an n -stage graph with 6^4 nodes in each stage, and find a shortest path. Since this graph is acyclic, we can find the shortest path in linear time (using topological sort).

SCRATCH PAPER

SCRATCH PAPER

SCRATCH PAPER

SCRATCH PAPER

SCRATCH PAPER