

---

# CS 573: Graduate Algorithms, Fall 2010

## Homework 0

Due Wednesday, September 1, 2010 in class

---

- This homework tests your familiarity with prerequisite material (<http://www.cs.uiuc.edu/class/fa10/cs573/stuff-you-already-know.html>) to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** For most topics, the early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
  - Each student must submit individual solutions for these homework problems. For all future homeworks, groups of up to three students may submit (or present) a single group solution for each problem.
  - Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. In particular:
    - Submit five separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page. Please do *not* staple everything together.
    - You may use any source at your disposal—paper, electronic, or human—but you **must** write your solutions in your own words, and you **must** cite every source that you use. In particular, each solution should include a list of *everyone* you worked with to solve that problem.
    - Unless explicitly stated otherwise, **every** homework problem requires a proof.
    - Answering “I don’t know” to any homework or exam problem (except for extra credit problems) is worth 25% partial credit.
    - Algorithms or proofs containing phrases like “and so on” or “repeat this process for all  $n$ ” instead of an explicit loop, recursion, or induction, will receive 0 points.
-

1. (•) **Write the sentence "I understand the course policies."**

Solutions that omit this sentence will not be graded.

- (a) [5 pts] Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . Assume reasonable but nontrivial base cases if none are given. **Do not submit proofs**—just a list of five functions—but you should do them anyway, just for practice.

- $A(n) = 4A(n - 1) + 1$

- $B(n) = B(n - 3) + n^2$

- $C(n) = 2C(n/2) + 3C(n/3) + n^2$

- $D(n) = 2D(n/3) + \sqrt{n}$

- $E(n) = \begin{cases} n & \text{if } n \leq 3, \\ \frac{E(n-1)E(n-2)}{E(n-3)} & \text{otherwise} \end{cases}$  [Hint: This is easier than it looks!]

- (b) [5 pts] Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do not submit proofs**—just a sorted list of 16 functions—but you should do them anyway, just for practice.

Write  $f(n) \ll g(n)$  to indicate that  $f(n) = o(g(n))$ , and write  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . We use the notation  $\lg n = \log_2 n$ .

$n$	$\lg n$	$\sqrt{n}$	$7^n$
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$7^{\sqrt{n}}$	$\sqrt{7^n}$
$7^{\lg n}$	$\lg(7^n)$	$7^{\lg \sqrt{n}}$	$7^{\sqrt{\lg n}}$
$\sqrt{7^{\lg n}}$	$\lg(7^{\sqrt{n}})$	$\lg \sqrt{7^n}$	$\sqrt{\lg(7^n)}$

2. Professore Giorgio della Giungla has a 23-node binary tree, in which every node is labeled with a unique letter of the Roman alphabet, which is just like the modern English alphabet, but without the letters **J**, **U**, and **W**. Inorder and postorder traversals of the tree visit the nodes in the following order:

- Inorder: **S V Z A T P R D B X O L F E H I Q M N G Y K C**

- Postorder: **A Z P T X B D L E F O H R I V N M K C Y G Q S**

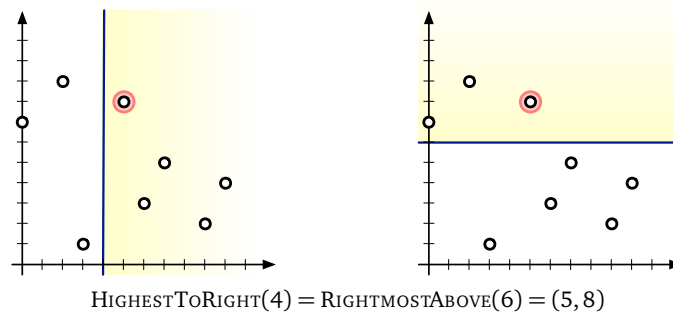
- (a) List the nodes in Prof. della Giungla's tree in the order visited by a *preorder* traversal.  
 (b) Draw Prof. della Giungla's tree.

3. The original version of this problem asked to support the mirror-image operations `LOWESTTORIGHT` and `LEFTMOSTABOVE`, which are *much* harder to support with a single data structure that stores each point at most once. We will accept  $O(n)$ -space data structures for either version of the problem for full credit.

Describe a data structure that stores a set  $S$  of  $n$  points in the plane, each represented by a pair  $(x, y)$  of coordinates, and supports the following queries.

- **HIGHESTTORIGHT( $\ell$ )**: Return the highest point in  $S$  whose  $x$ -coordinate is greater than or equal to  $\ell$ . If every point in  $S$  has  $x$ -coordinate less than  $\ell$ , return NONE.
- **RIGHTMOSTABOVE( $\ell$ )**: Return the rightmost point in  $S$  whose  $y$ -coordinate is greater than or equal to  $\ell$ . If every point in  $S$  has  $y$ -coordinate less than  $\ell$ , return NONE.

For example, if  $S = \{(3, 1), (1, 9), (9, 2), (6, 3), (5, 8), (7, 5), (10, 4), (0, 7)\}$ , then both `HIGHESTTORIGHT(4)` and `RIGHTMOSTABOVE(6)` should return the point  $(5, 8)$ , and `HIGHESTTORIGHT(15)` should return NONE.



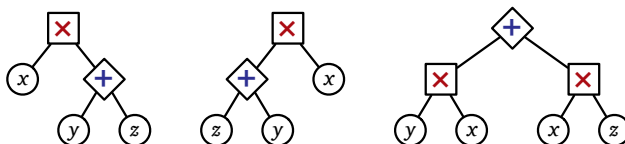
Analyze both the size of your data structure and the running times of your query algorithms. For full credit, your data structure should use  $O(n)$  space, and each query algorithm should run in  $O(\log n)$  time. **For 5 extra credit points, describe a data structure that stores each point at most once.** You may assume that no two points in  $S$  have equal  $x$ -coordinates or equal  $y$ -coordinates.

[Hint: Modify one of the standard data structures listed at <http://www.cs.uiuc.edu/class/fa10/cs573/stuff-you-already-know.html>, but just describe your changes; don't regurgitate the details of the standard data structure.]

4. An **arithmetic expression tree** is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are  $+$  and  $\times$ . Different leaves may or may not represent different variables.

Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any  $+$ -node is the sum of the values of its children. (2) The value of any  $\times$ -node is the product of the values of its children.

Two arithmetic expression trees are **equivalent** if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots.



Three equivalent expression trees. Only the third tree is in normal form.

An arithmetic expression tree is in **normal form** if the parent of every  $+$ -node (if any) is another  $+$ -node.

Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form. [Hint: Be careful. This is trickier than it looks.]

5. Recall that a standard (Anglo-American) deck of 52 playing cards contains 13 cards in each of four suits: spades ( $\spadesuit$ ), hearts ( $\heartsuit$ ), diamonds ( $\diamondsuit$ ), and clubs ( $\clubsuit$ ). Within each suit, the 13 cards have distinct ranks: 2, 3, 4, 5, 6, 7, 8, 9, 10, jack ( $J$ ), queen ( $Q$ ), king ( $K$ ), and ace ( $A$ ). The ranks are ordered  $2 < 3 < \dots < 9 < 10 < J < Q < K < A$ ; thus, for example, the jack of spades has higher rank than the eight of diamonds.

Professor Jay is about to perform a public demonstration with two decks of cards, one with red backs (“the red deck”) and one with blue backs (“the blue deck”). Both decks lie face-down on a table in front of Professor Jay, shuffled uniformly and independently. Thus, in each deck, every permutation of the 52 cards is equally likely.

To begin the demonstration, Professor Jay turns over the top card from each deck. Then, while he has not yet turned over a three of clubs ( $3\clubsuit$ ), the good Professor hurls the two cards he just turned over into the thick, pachydermatous outer melon layer of a nearby watermelon (that most prodigious of household fruits) and then turns over the next card from the top of each deck. Thus, if  $3\clubsuit$  is the last card in both decks, the demonstration ends with 102 cards embedded in the watermelon.

- What is the *exact* expected number of cards that Professor Jay hurls into the watermelon?
- For each of the statements below, give the *exact* probability that the statement is true of the **first** pair of cards Professor Jay turns over.
  - Both cards are threes.
  - One card is a three, and the other card is a club.
  - If (at least) one card is a heart, then (at least) one card is a diamond.
  - The card from the red deck has higher rank than the card from the blue deck.
- For each of the statements below, give the *exact* probability that the statement is true of the **last** pair of cards Professor Jay turns over.
  - Both cards are threes.
  - One card is a three, and the other card is a club.
  - If (at least) one card is a heart, then (at least) one card is a diamond.
  - The card from the red deck has higher rank than the card from the blue deck.

Express each of your answers as rational numbers in simplest form, like  $123/4567$ . **Do not submit proofs**—just a list of rational numbers—but you should do them anyway, just for practice.

# CS 573: Graduate Algorithms, Fall 2010

## Homework 1

~~Due Friday, September 10, 2010 at 1pm~~

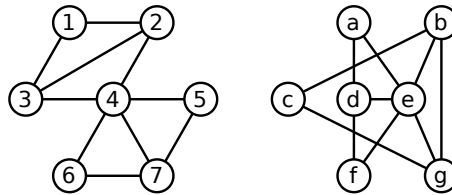
Due Monday, September 13, 2010 at 5pm  
(in the homework drop boxes in the basement of Siebel)

---

For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name and NetID on each page of your submission.

---

1. Two graphs are said to be *isomorphic* if one can be transformed into the other just by relabeling the vertices. For example, the graphs shown below are isomorphic; the left graph can be transformed into the right graph by the relabeling  $(1, 2, 3, 4, 5, 6, 7) \mapsto (c, g, b, e, a, f, d)$ .



Two isomorphic graphs.

Consider the following related decision problems:

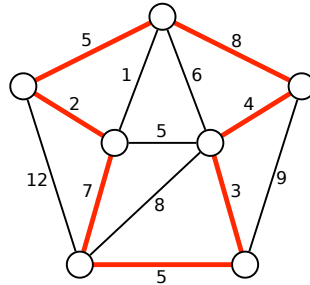
- **GRAPHISOMORPHISM**: Given two graphs  $G$  and  $H$ , determine whether  $G$  and  $H$  are isomorphic.
- **EVENGRAPHISOMORPHISM**: Given two graphs  $G$  and  $H$ , such that every vertex in  $G$  and  $H$  has even degree, determine whether  $G$  and  $H$  are isomorphic.
- **SUBGRAPHISOMORPHISM**: Given two graphs  $G$  and  $H$ , determine whether  $G$  is isomorphic to a subgraph of  $H$ .

- Describe a polynomial-time reduction from **EVENGRAPHISOMORPHISM** to **GRAPHISOMORPHISM**.
- Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **EVENGRAPHISOMORPHISM**.
- Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **SUBGRAPHISOMORPHISM**.
- Prove that **SUBGRAPHISOMORPHISM** is NP-complete.
- What can you conclude about the NP-hardness of **GRAPHISOMORPHISM**? Justify your answer.

[Hint: These are all easy!]

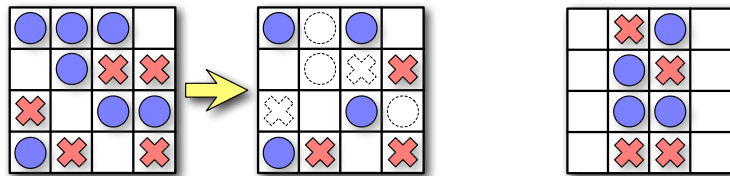
2. Suppose you are given a magic black box that can solve the **3COLORABLE** problem *in polynomial time*. That is, given an arbitrary graph  $G$  as input, the magic black box returns **TRUE** if  $G$  has a proper 3-coloring, and returns **FALSE** otherwise. Describe and analyze a *polynomial-time* algorithm that computes an actual proper 3-coloring of a given graph  $G$ , or correctly reports that no such coloring exists, using this magic black box as a subroutine. [Hint: The input to the black box is a graph. Just a graph. Nothing else.]

3. Let  $G$  be an undirected graph with weighted edges. A *heavy Hamiltonian cycle* is a cycle  $C$  that passes through each vertex of  $G$  exactly once, such that the total weight of the edges in  $C$  is at least half of the total weight of all edges in  $G$ . Prove that deciding whether a graph has a heavy Hamiltonian cycle is NP-complete.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

4. Consider the following solitaire game. The puzzle consists of an  $n \times m$  grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.

An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

5. A boolean formula in *exclusive-or conjunctive normal form* (XCNF) is a conjunction (AND) of several *clauses*, each of which is the *exclusive-or* of one or more literals. For example:

$$(u \oplus v \oplus \bar{w} \oplus x) \wedge (\bar{u} \oplus \bar{w} \oplus y) \wedge (\bar{v} \oplus y) \wedge (\bar{u} \oplus \bar{v} \oplus x \oplus y) \wedge (w \oplus x) \wedge y$$

The XCNF-SAT problem asks whether a given XCNF boolean formula is satisfiable. Either describe a polynomial-time algorithm for XCNF-SAT or prove that it is NP-complete.

# CS 573: Graduate Algorithms, Fall 2010

## Homework 2

Due Monday, September 27, 2010 at 5pm  
(in the homework drop boxes in the basement of Siebel)

---

- For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name and NetID of every group member on the first page of your submission.
- We will use the following rubric to grade all dynamic programming algorithms:
  - 60% for a correct recurrence (including base cases and a plain-English specification); no credit for anything else if this is wrong.
  - 10% for describing a suitable memoization data structure.
  - 20% for describing a correct evaluation order. (A clear picture is sufficient.)
  - 10% point for analyzing the running time of the resulting algorithm.

Official solutions will always include pseudocode for the final dynamic programming algorithm, but this is *not* required for full credit. However, if you do provide correct pseudocode for the dynamic programming algorithm, it is not necessary to separately describe the recurrence, the memoization data structure, or the evaluation order.

It is *not* necessary to state a space bound. There is no penalty for using more space than the official solution, but +1 extra credit for using less space with the same (or better) running time.

- The official solution for every problem will provide a target time bound. Algorithms faster than the official solution are worth more points (as extra credit); algorithms slower than the official solution are worth fewer points. For slower algorithms, partial credit is scaled to the lower maximum score. For example, if a full dynamic programming algorithm would be worth 5 points, just the recurrence is worth 3 points. However, incorrect algorithms are worth zero points, no matter how fast they are.
  - Greedy algorithms *must* be accompanied by proofs of correctness in order to receive *any* credit. Otherwise, **any correct algorithm, no matter how slow, is worth at least 2½ points**, assuming it is properly analyzed.
- 

1. Suppose you are given an array  $A[1..n]$  of positive integers. Describe and analyze an algorithm to find the *smallest* positive integer that is *not* an element of  $A$  in  $O(n)$  time.
2. Suppose you are given an  $m \times n$  bitmap, represented by an array  $M[1..m, 1..n]$  whose entries are all 0 or 1. A *solid block* is a subarray of the form  $M[i..i', j..j']$  in which every bit is equal to 1. Describe and analyze an efficient algorithm to find a solid block in  $M$  with maximum area.

3. Let  $T$  be a tree in which each edge  $e$  has a weight  $w(e)$ . A *matching*  $M$  in  $T$  is a subset of the edges such that each vertex of  $T$  is incident to at most one edge in  $M$ . The weight of a matching  $M$  is the sum of the weights of its edges. Describe and analyze an algorithm to compute a maximum weight matching, given the tree  $T$  as input.
4. For any string  $x$  and any non-negative integer  $k$ , let  $x^k$  denote the string obtained by concatenating  $k$  copies of  $x$ . For example,  $\text{STRING}^3 = \text{STRINGSTRINGSTRING}$  and  $\text{STRING}^0$  is the empty string.

A *repetition* of  $x$  is a prefix of  $x^k$  for some integer  $k$ . For example,  $\text{STRINGSTRINGSTRINGST}$  and  $\text{STR}$  are both repetitions of  $\text{STRING}$ , as is the empty string.

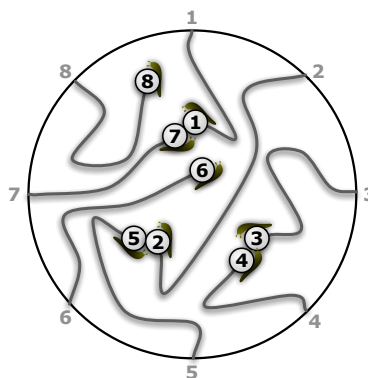
An *interleaving* of two strings  $x$  and  $y$  is any string obtained by shuffling a repetition of  $x$  with a repetition of  $y$ . For example,  $\text{STRWORINDGSTWIRNGDWSORR}$  is an interleaving of  $\text{STRING}$  and  $\text{WORD}$ , as is the empty string.

Describe and analyze an algorithm that accepts three strings  $x$ ,  $y$ , and  $z$  as input, and decides whether  $z$  is an interleaving of  $x$  and  $y$ .

5. Every year, as part of its annual meeting, the Antarctic Snail Lovers of Upper Glacierville hold a Round Table Mating Race. Several high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to  $n$ . During the race, each snail wanders around the table, leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail, even their own. If two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if the race goes on forever.

For every pair of snails, the Antarctic SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array  $M[1..n, 1..n]$  posted on the wall behind the Round Table, where  $M[i, j] = M[j, i]$  is the reward to be paid if snails  $i$  and  $j$  meet.

Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the array  $M$  as input.



The end of a typical Antarctic SLUG race. Snails 6 and 8 never find mates.

The organizers must pay  $M[3, 4] + M[2, 5] + M[1, 7]$ .

# CS 573: Graduate Algorithms, Fall 2010

## Homework 3

Due Monday, October 18, 2010 at 5pm  
(in the homework drop boxes in the basement of Siebel)

---

1. Suppose we are given two arrays  $C[1..n]$  and  $R[1..n]$  of positive integers. An  $n \times n$  matrix of 0s and 1s *agrees with  $R$  and  $C$*  if, for every index  $i$ , the  $i$ th row contains  $R[i]$  1s, and the  $i$ th column contains  $C[i]$  1s. Describe and analyze an algorithm that either constructs a matrix that agrees with  $R$  and  $C$ , or correctly reports that no such matrix exists.
2. Suppose we have  $n$  skiers with heights given in an array  $P[1..n]$ , and  $n$  skis with heights given in an array  $S[1..n]$ . Describe an efficient algorithm to assign a ski to each skier, so that the average difference between the height of a skier and her assigned ski is as small as possible. The algorithm should compute a permutation  $\sigma$  such that the expression

$$\frac{1}{n} \sum_{i=1}^n |P[i] - S[\sigma(i)]|$$

is as small as possible.

3. Alice wants to throw a party and she is trying to decide who to invite. She has  $n$  people to choose from, and she knows which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints:
  - For each guest, there should be at least five other guests that they already know.
  - For each guest, there should be at least five other guests that they *don't* already know.

Describe and analyze an algorithm that computes the largest possible number of guests Alice can invite, given a list of  $n$  people and the list of pairs who know each other.

4. Consider the following heuristic for constructing a vertex cover of a connected graph  $G$ : return the set of non-leaf nodes in any depth-first spanning tree of  $G$ .
  - (a) Prove that this heuristic returns a vertex cover of  $G$ .
  - (b) Prove that this heuristic returns a 2-approximation to the minimum vertex cover of  $G$ .
  - (c) Describe an infinite family of graphs for which this heuristic returns a vertex cover of size  $2 \cdot OPT$ .

5. Suppose we want to route a set of  $N$  calls on a telecommunications network that consist of a cycle on  $n$  nodes, indexed in order from 0 to  $n - 1$ . Each call has a source node and a destination node, and can be routed either clockwise or counterclockwise around the cycle. Our goal is to route the calls so as to minimize the overall load on the network. The load  $L_i$  on any edge  $(i, (i + 1) \bmod n)$  is the number of calls routed through that edge, and the overall load is  $\max_i L_i$ . Describe and analyze an efficient 2-approximation algorithm for this problem.

# CS 573: Graduate Algorithms, Fall 2010

## Homework 4

Due Monday, November 1, 2010 at 5pm  
(in the homework drop boxes in the basement of Siebel)

---

1. Consider an  $n$ -node treap  $T$ . As in the lecture notes, we identify nodes in  $T$  by the ranks of their search keys. Thus, 'node 5' means the node with the 5th smallest search key. Let  $i, j, k$  be integers such that  $1 \leq i \leq j \leq k \leq n$ .
  - (a) What is the *exact* probability that node  $j$  is a common ancestor of node  $i$  and node  $k$ ?
  - (b) What is the *exact* expected length of the unique path from node  $i$  to node  $k$  in  $T$ ?
2. Let  $M[1..n, 1..n]$  be an  $n \times n$  matrix in which every row and every column is sorted. Such an array is called *totally monotone*. No two elements of  $M$  are equal.
  - (a) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, compute the number of elements of  $M$  smaller than  $M[i, j]$  and larger than  $M[i', j']$ .
  - (b) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, return an element of  $M$  chosen uniformly at random from the elements smaller than  $M[i, j]$  and larger than  $M[i', j']$ . Assume the requested range is always non-empty.
  - (c) Describe and analyze a randomized algorithm to compute the median element of  $M$  in  $O(n \log n)$  expected time.
3. Suppose we are given a complete undirected graph  $G$ , in which each edge is assigned a weight chosen independently and uniformly at random from the real interval  $[0, 1]$ . Consider the following greedy algorithm to construct a Hamiltonian cycle in  $G$ . We start at an arbitrary vertex. While there is at least one unvisited vertex, we traverse the minimum-weight edge from the current vertex to an unvisited neighbor. After  $n - 1$  iterations, we have traversed a Hamiltonian path; to complete the Hamiltonian cycle, we traverse the edge from the last vertex back to the first vertex. What is the expected weight of the resulting Hamiltonian cycle? [*Hint: What is the expected weight of the first edge? Consider the case  $n = 3$ .*]

4. (a) Consider the following deterministic algorithm to construct a vertex cover  $C$  of a graph  $G$ .

```

VERTEXCOVER( $G$ ):
   $C \leftarrow \emptyset$ 
  while  $C$  is not a vertex cover
    pick an arbitrary edge  $uv$  that is not covered by  $C$ 
    add either  $u$  or  $v$  to  $C$ 
  return  $C$ 

```

Prove that VERTEXCOVER can return a vertex cover that is  $\Omega(n)$  times larger than the smallest vertex cover. You need to describe both an input graph with  $n$  vertices, for any integer  $n$ , and the sequence of edges and endpoints chosen by the algorithm.

- (b) Now consider the following randomized variant of the previous algorithm.

```

RANDOMVERTEXCOVER( $G$ ):
   $C \leftarrow \emptyset$ 
  while  $C$  is not a vertex cover
    pick an arbitrary edge  $uv$  that is not covered by  $C$ 
    with probability  $1/2$ 
      add  $u$  to  $C$ 
    else
      add  $v$  to  $C$ 
  return  $C$ 

```

Prove that the expected size of the vertex cover returned by RANDOMVERTEXCOVER is at most  $2 \cdot \text{OPT}$ , where OPT is the size of the smallest vertex cover.

- (c) Let  $G$  be a graph in which each vertex  $v$  has a weight  $w(v)$ . Now consider the following randomized algorithm that constructs a vertex cover.

```

RANDOMWEIGHTEDVERTEXCOVER( $G$ ):
   $C \leftarrow \emptyset$ 
  while  $C$  is not a vertex cover
    pick an arbitrary edge  $uv$  that is not covered by  $C$ 
    with probability  $w(v)/(w(u) + w(v))$ 
      add  $u$  to  $C$ 
    else
      add  $v$  to  $C$ 
  return  $C$ 

```

Prove that the expected weight of the vertex cover returned by RANDOMWEIGHTEDVERTEXCOVER is at most  $2 \cdot \text{OPT}$ , where OPT is the weight of the minimum-weight vertex cover. A correct answer to this part automatically earns full credit for part (b).

5. (a) Suppose  $n$  balls are thrown uniformly and independently at random into  $m$  bins. For any integer  $k$ , what is the *exact* expected number of bins that contain exactly  $k$  balls?
- (b) Consider the following balls and bins experiment, where we repeatedly throw a fixed number of balls randomly into a shrinking set of bins. The experiment starts with  $n$  balls and  $n$  bins. In each round  $i$ , we throw  $n$  balls into the remaining bins, and then discard any non-empty bins; thus, only bins that are empty at the end of round  $i$  survive to round  $i + 1$ .

BALLSDESTROYBINS( $n$ ):  
 start with  $n$  empty bins  
 while any bins remain  
   throw  $n$  balls randomly into the remaining bins  
   discard all bins that contain at least one ball

Suppose that in every round, *precisely* the expected number of bins are empty. Prove that under these conditions, the experiment ends after  $O(\log^* n)$  rounds.<sup>1</sup>

- \* (c) **[Extra credit]** Now assume that the balls are really thrown randomly into the bins in each round. Prove that with high probability, BALLSDESTROYBINS( $n$ ) ends after  $O(\log^* n)$  rounds.
- (d) Now consider a variant of the previous experiment in which we discard balls instead of bins. Again, the experiment  $n$  balls and  $n$  bins. In each round  $i$ , we throw the remaining balls into  $n$  bins, and then discard any ball that lies in a bin by itself; thus, only balls that collide in round  $i$  survive to round  $i + 1$ .

BINSDESTROYSINGLEBALLS( $n$ ):  
 start with  $n$  balls  
 while any balls remain  
   throw the remaining balls randomly into  $n$  bins  
   discard every ball that lies in a bin by itself  
   retrieve the remaining balls from the bins

Suppose that in every round, *precisely* the expected number of bins contain exactly one ball. Prove that under these conditions, the experiment ends after  $O(\log \log n)$  rounds.

- \* (e) **[Extra credit]** Now assume that the balls are really thrown randomly into the bins in each round. Prove that with high probability, BINSDESTROYSINGLEBALLS( $n$ ) ends after  $O(\log \log n)$  rounds.

---

<sup>1</sup>Recall that the iterated logarithm is defined as follows:  $\log^* n = 0$  if  $n \leq 1$ , and  $\log^* n = 1 + \log^*(\lg n)$  otherwise.

# CS 573: Graduate Algorithms, Fall 2010

## Homework 5

Due Friday, November 19, 2010 at 5pm  
(in the homework drop boxes in the basement of Siebel)

---

- Suppose we are given a set of boxes, each specified by their height, width, and depth in centimeters. All three side lengths of every box lie strictly between 10cm and 20cm. As you should expect, one box can be placed inside another if the smaller box can be rotated so that its height, width, and depth are respectively smaller than the height, width, and depth of the larger box. Boxes can be nested recursively. Call a box *visible* if it is not inside another box.

Describe and analyze an algorithm to nest the boxes so that the number of visible boxes is as small as possible.

- Suppose we are given an array  $A[1..m][1..n]$  of non-negative real numbers. We want to *round*  $A$  to an integer matrix, by replacing each entry  $x$  in  $A$  with either  $\lfloor x \rfloor$  or  $\lceil x \rceil$ , without changing the sum of entries in any row or column of  $A$ . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe an efficient algorithm that either rounds  $A$  in this fashion, or reports correctly that no such rounding is possible.

- The Autocratic Party is gearing up their fund-raising campaign for the 2012 election. Party leaders have already chosen their slate of candidates for president and vice-president, as well as various governors, senators, representatives, city council members, school board members, and dog-catchers. For each candidate, the party leaders have determined how much money they must spend on that candidate's campaign to guarantee their election.

The party is soliciting donations from each of its members. Each voter has declared the total amount of money they are willing to give each candidate between now and the election. (Each voter pledges different amounts to different candidates. For example, everyone is happy to donate to the presidential candidate,<sup>1</sup> but most voters in New York will not donate anything to the candidate for Trash Commissioner of Los Angeles.) Federal election law limits each person's total political contributions to \$100 per day.

Describe and analyze an algorithm to compute a donation schedule, describing how much money each voter should send to each candidate on each day, that guarantees that every candidate gets enough money to win their election. (Party members will of course follow their given schedule perfectly.<sup>2</sup>) The schedule must obey both Federal laws and individual voters' budget constraints. If no such schedule exists, your algorithm should report that fact.

---

<sup>1</sup>or some nice men in suits will be visiting their home.

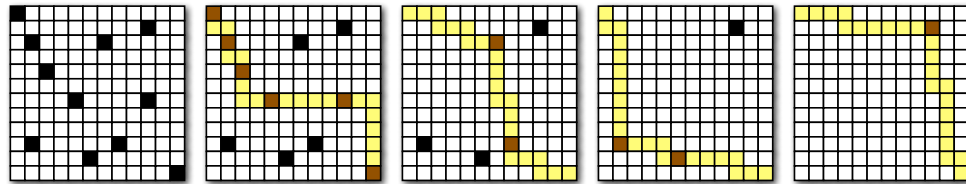
<sup>2</sup>It's a nice house you've got here. Shame if anything happened to it.

4. Consider an  $n \times n$  grid, some of whose cells are marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell. We want to compute the minimum number of monotone paths that cover all the marked cells.

(a) One of your friends suggests the following greedy strategy:

- Find (somehow) one “good” path  $\pi$  that covers the maximum number of marked cells.
- Unmark the cells covered by  $\pi$ .
- If any cells are still marked, recursively cover them.

Prove that this greedy strategy does *not* always compute an optimal solution.



Greeditly covering the marked cells in a grid with four monotone paths.

- (b) Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell. The input to your algorithm is an array  $M[1..n, 1..n]$  of booleans, where  $M[i, j] = \text{TRUE}$  if and only if cell  $(i, j)$  is marked.
5. Let  $G$  be a directed graph with two distinguished vertices  $s$  and  $t$ , and let  $r$  be a positive integer. Two players named Paul and Sally play the following game. Paul chooses a path  $P$  from  $s$  to  $t$ , and Sally chooses a subset  $S$  of **at most**  $r$  edges in  $G$ . The players reveal their chosen subgraphs simultaneously. If  $P \cap S = \emptyset$ , Paul wins; if  $P \cap S \neq \emptyset$ , then Sally wins. Both players want to maximize their chances of winning the game.
- (a) Prove that if Paul uses a deterministic strategy, and Sally knows his strategy, then Sally can guarantee that she wins.<sup>3</sup>
  - (b) Let  $M$  be the number of edges in a minimum  $(s, t)$ -cut. Describe a deterministic strategy for Sally that guarantees that she wins when  $r \geq M$ , no matter what strategy Paul uses.
  - (c) Prove that if Sally uses a deterministic strategy, and Paul knows her strategy then Paul can guarantee that he wins when  $r < M$ .
  - (d) Describe a randomized strategy for Sally that guarantees that she wins with probability at least  $\min\{r/M, 1\}$ , no matter what strategy Paul uses.
  - (e) Describe a randomized strategy for Paul that guarantees that he loses with probability at most  $\min\{r/M, 1\}$ , no matter what strategy Sally uses.

Paul and Sally’s strategies are, of course, algorithms. (For example, Paul’s strategy is an algorithm that takes the graph  $G$  and the integer  $r$  as input and produces a path  $P$  as output.) You do *not* need to analyze the running times of these algorithms, but you must prove all claims about their winning probabilities. Most of these questions are easy.

<sup>3</sup>“Good old rock. Nothing beats rock. . . . D’oh!”

# CS 573: Graduate Algorithms, Fall 2010

## Homework 5

Practice only — Do not submit solutions

---

1. (a) Describe how to transform any linear program written in general form into an equivalent linear program written in *slack* form.

$\begin{array}{ll} \text{maximize} & \sum_{j=1}^d c_j x_j \\ \text{subject to} & \sum_{j=1}^d a_{ij} x_j \leq b_i \quad \text{for each } i = 1 \dots p \\ & \sum_{j=1}^d a_{ij} x_j = b_i \quad \text{for each } i = p + 1 \dots p + q \\ & \sum_{j=1}^d a_{ij} x_j \geq b_i \quad \text{for each } i = p + q + 1 \dots n \end{array}$	$\implies$	$\begin{array}{l} \max \quad c \cdot x \\ \text{s.t. } Ax = b \\ \quad x \geq 0 \end{array}$
--	------------	--

- (b) Describe precisely how to dualize a linear program written in slack form.  
(c) Describe precisely how to dualize a linear program written in general form.

In all cases, keep the number of variables in the resulting linear program as small as possible.

2. Suppose you have a subroutine that can solve linear programs in polynomial time, but only if they are both feasible and bounded. Describe an algorithm that solves *arbitrary* linear programs in polynomial time. Your algorithm should return an optimal solution if one exists; if no optimum exists, your algorithm should report that the input instance is UNBOUNDED or INFEASIBLE, whichever is appropriate. *[Hint: Add one variable and one constraint.]*
3. An *integer program* is a linear program with the additional constraint that the variables must take only integer values.
- (a) Prove that deciding whether an integer program has a feasible solution is NP-complete.  
(b) Prove that finding the optimal feasible solution to an integer program is NP-hard.

*[Hint: Almost any NP-hard decision problem can be formulated as an integer program. Pick your favorite.]*

4. Give a linear-programming formulation of the *minimum-cost feasible circulation problem*. You are given a flow network whose edges have both capacities and costs, and your goal is to find a feasible circulation (flow with value 0) whose cost is as small as possible.

5. Given points  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  in the plane, the *linear regression problem* asks for real numbers  $a$  and  $b$  such that the line  $y = ax + b$  fits the points as closely as possible, according to some criterion. The most common fit criterion is minimizing the  $L_2$  error, defined as follows:<sup>1</sup>

$$\varepsilon_2(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2.$$

But there are several other fit criteria, some of which can be optimized via linear programming.

- (a) The  $L_1$  error (or *total absolute deviation*) of the line  $y = ax + b$  is defined as follows:

$$\varepsilon_1(a, b) = \sum_{i=1}^n |y_i - ax_i - b|.$$

Describe a linear program whose solution  $(a, b)$  describes the line with minimum  $L_1$  error.

- (b) The  $L_\infty$  error (or *maximum absolute deviation*) of the line  $y = ax + b$  is defined as follows:

$$\varepsilon_\infty(a, b) = \max_{i=1}^n |y_i - ax_i - b|.$$

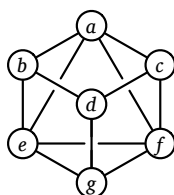
Describe a linear program whose solution  $(a, b)$  describes the line with minimum  $L_\infty$  error.

---

<sup>1</sup>This measure is also known as *sum of squared residuals*, and the algorithm to compute the best fit is normally called (*ordinary/linear*) *least squares fitting*.

This exam lasts 90 minutes.  
**Write your answers in the separate answer booklet.**  
 Please return this question sheet with your answers.

1. (a) Suppose  $A[1..n]$  is an array of  $n$  distinct integers, sorted so that  $A[1] < A[2] < \dots < A[n]$ . Each integer  $A[i]$  could be positive, negative, or zero. Describe and analyze an efficient algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists.
- (b) Now suppose  $A[1..n]$  is a sorted array of  $n$  distinct **positive** integers. Describe and analyze an even faster algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists.
2. A *double-Hamiltonian circuit* a closed walk in a graph that visits every vertex exactly *twice*. **Prove** that it is NP-hard to determine whether a given graph contains a double-Hamiltonian circuit.



This graph contains the double-Hamiltonian circuit  $a \rightarrow b \rightarrow d \rightarrow g \rightarrow e \rightarrow b \rightarrow d \rightarrow c \rightarrow f \rightarrow a \rightarrow c \rightarrow f \rightarrow g \rightarrow e \rightarrow a$ .

3. A palindrome is any string that is exactly the same as its reversal, like I, or DEED, or HANNAH, or AMANAPLANACATACANALPANAMA. Describe and analyze an algorithm to find the length of the longest subsequence of a given string that is also a palindrome.  
 For example, the longest palindrome subsequence of MAHDYNAMICPROGRAMZLETMESHOWOUTHEM is MHYMRORMYHM, so given that string as input, your algorithm should return the integer 11.
4. Suppose you are given a magic black box that can determine **in polynomial time**, given an arbitrary graph  $G$ , the number of vertices in the largest complete subgraph of  $G$ . Describe and analyze a **polynomial-time** algorithm that computes, given an arbitrary graph  $G$ , a complete subgraph of  $G$  of maximum size, using this magic black box as a subroutine.
5. Suppose we are given a  $4 \times n$  grid, where each grid cell has an integer value. Suppose we want to mark a subset of the grid cells, so that the total value of the marked cells is as large as possible. However, we are forbidden to mark any pair of grid cells that are immediate horizontal or vertical neighbors. (Marking diagonal neighbors is fine.) Describe and analyze an algorithm that computes the largest possible sum of marked cells, subject to this non-adjacency condition.

For example, given the grid on the left below, your algorithm should return the integer 36, which is the sum of the circled numbers on the right.

4	-5	1	6
2	6	-1	8
5	4	3	3
1	-1	7	4
-3	4	5	-2

⇒

④	-5	1	⑥
2	⑥	-1	8
⑤	4	3	③
1	-1	⑦	4
-3	⑤	4	-2

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output True?

**PLANARCIRCUITSAT:** Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAX2SAT:** Given a boolean formula in conjunctive normal form, with exactly two literals per clause, what is the largest number of clauses that can be satisfied by an assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINDOMINATINGSET:** Given an undirected graph  $G$ , what is the size of the smallest subset  $S$  of vertices such that every vertex in  $G$  is either in  $S$  or adjacent to a vertex in  $S$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**CHROMATICNUMBER:** Given an undirected graph  $G$ , what is the minimum number of colors needed to color its vertices, so that every edge touches vertices with two different colors?

**MAXCUT:** Given a graph  $G$ , what is the size (number of edges) of the largest bipartite subgraph of  $G$ ?

**HAMILTONIANCYCLE:** Given a graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**HAMILTONIANPATH:** Given a graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $n$  positive integers, can  $X$  be partitioned into  $n/3$  three-element subsets, all with the same sum?

**MINESWEEPER:** Given a Minesweeper configuration and a particular square  $x$ , is it safe to click on  $x$ ?

**TETRIS:** Given a sequence of  $N$  Tetris pieces and a partially filled  $n \times k$  board, is it possible to play every piece in the sequence without overflowing the board?

**SUDOKU:** Given an  $n \times n$  Sudoku puzzle, does it have a solution?

**KENKEN:** Given an  $n \times n$  Ken-Ken puzzle, does it have a solution?

This exam lasts 90 minutes.  
**Write your answers in the separate answer booklet.**  
 Please return this question sheet with your answers.

1. Assume we have access to a function  $\text{RANDOM}(k)$  that returns, given any positive integer  $k$ , an integer chosen independently and uniformly at random from the set  $\{1, 2, \dots, k\}$ , in  $O(1)$  time. For example, to perform a fair coin flip, we could call  $\text{RANDOM}(2)$ .

Now suppose we want to write an efficient function  $\text{RANDOMPERMUTATION}(n)$  that returns a permutation of the set  $\{1, 2, \dots, n\}$  chosen uniformly at random; that is, each permutation must be chosen with probability  $1/n!$ .

- (a) **Prove** that the following algorithm is **not** correct. [Hint: Consider the case  $n = 3$ .]

```

RANDOMPERMUTATION(n):
  for i ← 1 to n
    π[i] ← i
  for i ← 1 to n
    swap π[i] ↔ π[RANDOM(n)]
  return π
  
```

- (b) Describe and analyze a correct  $\text{RANDOMPERMUTATION}$  algorithm that runs in  $O(n)$  expected time. (In fact,  $O(n)$  worst-case time is possible.)
2. Suppose we have  $n$  pieces of candy with weights  $W[1..n]$  (in ounces) that we want to load into boxes. Our goal is to load the candy into as many boxes as possible, so that each box contains at least  $L$  ounces of candy. Describe an efficient 2-approximation algorithm for this problem. **Prove** that the approximation ratio of your algorithm is 2.
- (For 7 points partial credit, assume that every piece of candy weighs less than  $L$  ounces.)
3. The  $\text{MAXIMUM-}k\text{-CUT}$  problem is defined as follows. We are given a graph  $G$  with weighted edges and an integer  $k$ . Our goal is to partition the vertices of  $G$  into  $k$  subsets  $S_1, S_2, \dots, S_k$ , so that the sum of the weights of the edges that cross the partition (that is, with endpoints in different subsets) is as large as possible.
- (a) Describe an efficient randomized approximation algorithm for  $\text{MAXIMUM-}k\text{-CUT}$ , and **prove** that its expected approximation ratio is **at most**  $(k - 1)/k$ .
- (b) Now suppose we want to minimize the sum of the weights of edges that do *not* cross the partition. What expected approximation ratio does your algorithm from part (a) achieve for this new problem? **Prove** your answer is correct.

4. The citizens of Binaria use coins whose values are powers of two. That is, for any non-negative integer  $k$ , there are Binarian coins with value is  $2^k$  bits. Consider the natural greedy algorithm to make  $x$  bits in change: If  $x > 0$ , use one coin with the largest denomination  $d \leq x$  and then recursively make  $x - d$  bits in change. (Assume you have an unlimited supply of each denomination.)
- (a) **Prove** that this algorithm uses at most one coin of each denomination.
  - (b) **Prove** that this algorithm finds the minimum number of coins whose total value is  $x$ .

5. Any permutation  $\pi$  can be represented as a set of disjoint cycles, by considering the directed graph whose vertices are the integers between 1 and  $n$  and whose edges are  $i \rightarrow \pi(i)$  for each  $i$ . For example, the permutation  $\langle 5, 4, 2, 6, 7, 8, 1, 3, 9 \rangle$  has three cycles:  $(175)(24683)(9)$ .

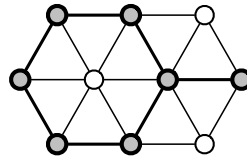
In the following questions, let  $\pi$  be a permutation of  $\{1, 2, \dots, n\}$  chosen uniformly at random, and let  $k$  be an arbitrary integer such that  $1 \leq k \leq n$ .

- (a) **Prove** that the probability that the number 1 lies in a cycle of length  $k$  in  $\pi$  is precisely  $1/n$ .  
[Hint: Consider the cases  $k = 1$  and  $k = 2$ .]
- (b) What is the *exact* expected length of the cycle in  $\pi$  that contains the number 1?
- (c) What is the *exact* expected number of cycles of length  $k$  in  $\pi$ ?
- (d) What is the *exact* expected number of cycles in  $\pi$ ?

You may assume part (a) in your solutions to parts (b), (c), and (d).

This exam lasts 180 minutes.  
**Write your answers in the separate answer booklet.**  
 Please return this question sheet with your answers.

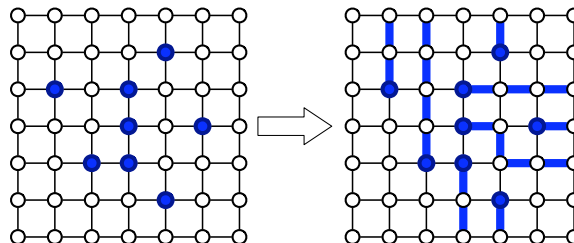
1. A subset  $S$  of vertices in an undirected graph  $G$  is called *triangle-free* if, for every triple of vertices  $u, v, w \in S$ , at least one of the three edges  $uv, uw, vw$  is *absent* from  $G$ . **Prove** that finding the size of the largest triangle-free subset of vertices in a given undirected graph is NP-hard.



A triangle-free subset of 7 vertices.  
 This is **not** the largest triangle-free subset in this graph.

2. An  $n \times n$  grid is an undirected graph with  $n^2$  vertices organized into  $n$  rows and  $n$  columns. We denote the vertex in the  $i$ th row and the  $j$ th column by  $(i, j)$ . Every vertex in the grid has exactly four neighbors, except for the *boundary* vertices, which are the vertices  $(i, j)$  such that  $i = 1$ ,  $i = n$ ,  $j = 1$ , or  $j = n$ .

Let  $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$  be distinct vertices, called *terminals*, in the  $n \times n$  grid. The **escape problem** is to determine whether there are  $m$  vertex-disjoint paths in the grid that connect the terminals to any  $m$  distinct boundary vertices. Describe and analyze an efficient algorithm to solve the escape problem.



A positive instance of the escape problem, and its solution.

3. Consider the following problem, called **UNIQUESETCOVER**. The input is an  $n$ -element set  $S$ , together with a collection of  $m$  subsets  $S_1, S_2, \dots, S_m \subseteq S$ , such that each element of  $S$  lies in exactly  $k$  subsets  $S_i$ . Our goal is to select some of the subsets so as to maximize the number of elements of  $S$  that lie in *exactly one* selected subset.

- (a) Fix a real number  $p$  between 0 and 1, and consider the following algorithm:

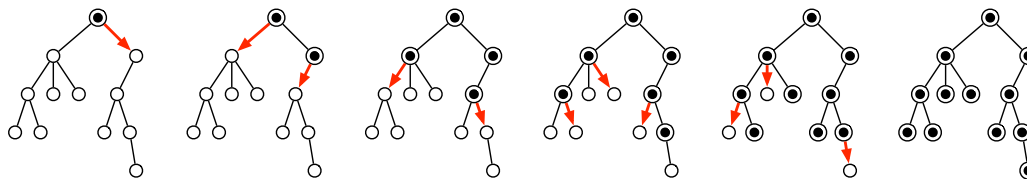
For each index  $i$ , select subset  $S_i$  independently with probability  $p$ .

What is the *exact* expected number of elements that are uniquely covered by the chosen subsets? (Express your answer as a function of the parameters  $p$  and  $k$ .)

- (b) What value of  $p$  maximizes this expectation?

- (c) Describe a polynomial-time randomized algorithm for **UNIQUESETCOVER** whose expected approximation ratio is  $O(1)$ .

4. Suppose we need to distribute a message to all the nodes in a rooted tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children. Describe and analyze an efficient algorithm to compute the minimum number of rounds required for the message to be delivered to every node.



A message being distributed through a tree in five rounds.

5. Every year, Professor Dumbledore assigns the instructors at Hogwarts to various faculty committees. There are  $n$  faculty members and  $c$  committees. Each committee member has submitted a list of their *prices* for serving on each committee; each price could be positive, negative, zero, or even infinite. For example, Professor Snape might declare that he would serve on the Student Recruiting Committee for 1000 Galleons, that he would *pay* 10000 Galleons to serve on the Defense Against the Dark Arts Course Revision Committee, and that he would not serve on the Muggle Relations committee for any price.

Conversely, Dumbledore knows how many instructors are needed for each committee, as well as a list of instructors who would be suitable members for each committee. (For example: “Dark Arts Revision: 5 members, anyone but Snape.”) If Dumbledore assigns an instructor to a committee, he must pay that instructor’s price from the Hogwarts treasury.

Dumbledore needs to assign instructors to committees so that (1) each committee is full, (2) no instructor is assigned to more than three committees, (3) only suitable and willing instructors are assigned to each committee, and (4) the total cost of the assignment is as small as possible. Describe and analyze an efficient algorithm that either solves Dumbledore’s problem, or correctly reports that there is no valid assignment whose total cost is finite.

6. Suppose we are given a rooted tree  $T$ , where every edge  $e$  has a non-negative *length*  $\ell(e)$ . Describe and analyze an efficient algorithm to assign a *stretched* length  $s\ell(e) \geq \ell(e)$  to every edge  $e$ , satisfying the following conditions:
- Every root-to-leaf path in  $T$  has the same total stretched length.
  - The total stretch  $\sum_e (s\ell(e) - \ell(e))$  is as small as possible.
7. Let  $G = (V, E)$  be a directed graph with edge capacities  $c : E \rightarrow \mathbb{R}^+$ , a source vertex  $s$ , and a target vertex  $t$ . Suppose someone hands you an *arbitrary* function  $f : E \rightarrow \mathbb{R}$ . Describe and analyze fast and *simple* algorithms to answer the following questions:
- (a) Is  $f$  a feasible  $(s, t)$ -flow in  $G$ ?
  - (b) Is  $f$  a *maximum*  $(s, t)$ -flow in  $G$ ?
  - (c) Is  $f$  the *unique* maximum  $(s, t)$ -flow in  $G$ ?

**Chernoff bounds:**

If  $X$  is the sum of independent indicator variables and  $\mu = E[X]$ , then

$$\Pr[X > (1 + \delta)\mu] \leq \left( \frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu \quad \text{for any } \delta > 0$$

$$\Pr[X < (1 - \delta)\mu] \leq \left( \frac{e^{-\delta}}{(1 - \delta)^{1-\delta}} \right)^\mu \quad \text{for any } 0 < \delta < 1$$

**You may assume the following running times:**

- Maximum flow or minimum cut:  $O(E|f^*|)$  or  $O(VE \log V)$
- Minimum-cost maximum flow:  $O(E^2 \log^2 V)$

(These are *not* the best time bounds known, but they're close enough for the final exam.)

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output True?

**PLANARCIRCUITSAT:** Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAX2SAT:** Given a boolean formula in conjunctive normal form, with exactly two literals per clause, what is the largest number of clauses that can be satisfied by an assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**MAXCUT:** Given a graph  $G$ , what is the size (number of edges) of the largest bipartite subgraph of  $G$ ?

**HAMILTONIANCYCLE:** Given a graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**HAMILTONIANPATH:** Given a graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $n$  positive integers, can  $X$  be partitioned into  $n/3$  three-element subsets, all with the same sum?