∽ New CS 473: Algorithms, Spring 2015 ∾
Homework 0

Due Tuesday, January 27, 2015 at 5pm

- **This homework tests your familiarity with prerequisite material:** designing, describing, and analyzing elementary algorithms (at the level of CS 225); fundamental graph problems and algorithms (again, at the level of CS 225); and especially facility with recursion and induction. Notes on most of this prerequisite material are available on the course web page.
- **Each student must submit individual solutions for this homework.** For all future homeworks, groups of up to three students will be allowed to submit joint solutions.

☞ Some important course policies ☜

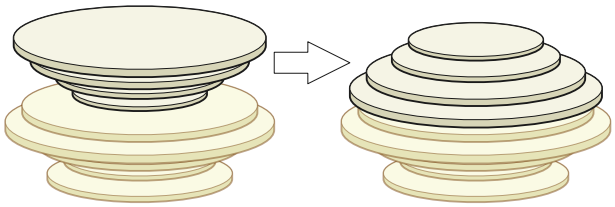
- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- **Submit your solutions on standard printer/copier paper.** Please use both sides of the paper. Please clearly print your name and NetID at the top of each page. If you plan to write your solutions by hand, please print the last four pages of this homework as templates. If you plan to typeset your homework, you can find a LaTeX template on the course web site; well-typeset homework will get a small amount of extra credit.
- **Start your solution to each numbered problem on a new sheet of paper.** Do not staple your entire homework together.
- **Submit your solutions in the drop boxes outside 1404 Siebel labeled "New CS 473".** There is a separate drop box for each numbered problem; if you put your solution in the wrong drop box, we won't grade it. Don't give your homework to Jeff in class; he is fond of losing important pieces of paper.
- **Avoid the Three Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an automatic zero on any homework or exam problem, unless your solution is nearly perfect otherwise. Yes, we are completely serious.
  - Always give complete solutions, not just examples.
  - Always declare all your variables.
  - Never use weak induction.
- Answering any homework or exam problem (or subproblem) in this course with "I don't know" *and nothing else* is worth 25% partial credit. We will accept synonyms like "No idea" or "WTF", but you must write *something*.

See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

---

1. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a ***flip***—insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over.



Flipping the top four pancakes.

   (a) Describe an algorithm to sort an arbitrary stack of n pancakes, which uses as few flips as possible in the worst case. *Exactly* how many flips does your algorithm perform in the worst case?

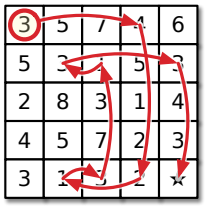
   (b) Now suppose one side of each pancake is burned. Describe an algorithm to sort an arbitrary stack of n pancakes, so that the burned side of every pancake is facing down, using as few flips as possible in the worst case. *Exactly* how many flips does your algorithm perform in the worst case?

2. *[From last semester's CS 374 final exam]* A ***number maze*** is an $n \times n$ grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner.

   - On each turn, you are allowed to move the token up, down, left, or right.
   - The distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right.
   - However, you are never allowed to move the token off the edge of the board. In particular, if the current number is too large, you may not be able to move at all.

   Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution. For example, given the maze shown below, your algorithm would return the number 8.

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



A 5×5 number maze that can be solved in eight moves.

[Hint: Build a graph. What are the vertices? What are the edges? Is the graph directed or undirected? Do the vertices or edges have weights? If so, what are they? What textbook problem do you need to solve on this graph? What textbook algorithm should you use to solve that problem? What is the running time of that algorithm as a function of n?]

3. (a) The ***Fibonacci numbers F_n*** are defined by the recurrence $F_n = F_{n-1} + F_{n-2}$, with base cases $F_0 = 0$ and $F_1 = 1$. Here are the first several Fibonacci numbers:

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
0	1	1	2	3	5	8	13	21	34	55

Prove that every non-negative integer can be written as the sum of distinct, non-consecutive Fibonacci numbers. That is, if the Fibonacci number F_i appears in the sum, it appears exactly once, and its neighbors F_{i-1} and F_{i+1} do not appear at all. For example:

$$17 = F_7 + F_4 + F_2, \qquad 42 = F_9 + F_6, \qquad 54 = F_9 + F_7 + F_5 + F_3 + F_1.$$

(b) The Fibonacci sequence can be extended backward to negative indices by rearranging the defining recurrence: $F_n = F_{n+2} - F_{n+1}$. Here are the first several negative-index Fibonacci numbers:

F_{-10}	F_{-9}	F_{-8}	F_{-7}	F_{-6}	F_{-5}	F_{-4}	F_{-3}	F_{-2}	F_{-1}
−55	34	−21	13	−8	5	−3	2	−1	1

Prove that $F_{-n} = -F_n$ if and only if n is even.

(c) Prove that *every* integer—positive, negative, or zero—can be written as the sum of distinct, non-consecutive Fibonacci numbers *with negative indices*. For example:

$$17 = F_{-7} + F_{-5} + F_{-2}, \quad -42 = F_{-10} + F_{-7}, \quad 54 = F_{-9} + F_{-7} + F_{-5} + F_{-3} + F_{-1}.$$

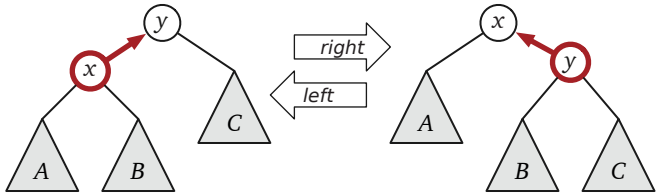
[Hint: Zero is both non-negative and even. Don't even think about using weak induction!]

4. ***[Extra credit]*** Let T be a binary tree whose nodes store distinct numerical values. Recall that T is a ***binary search tree*** if and only if either (1) T is empty, or (2) T satisfies the following recursive conditions:

- The left subtree of T is a binary search tree.
- All values in the left subtree of T are smaller than the value at the root of T.
- The right subtree of T is a binary search tree.
- All values in the right subtree of T are larger than the value at the root of T.

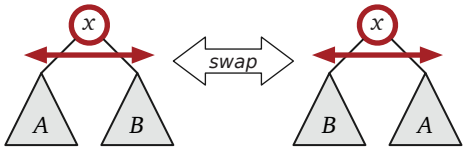
Describe and analyze an algorithm to transform an *arbitrary* binary tree T with distinct node values into a binary search tree, using ***only*** the following operations:

- Rotate an arbitrary node. Rotation is a local operation that decreases the depth of a node by one and increases the depth of its parent by one.



Left to right: right rotation at x. Right to left: left rotation at y.

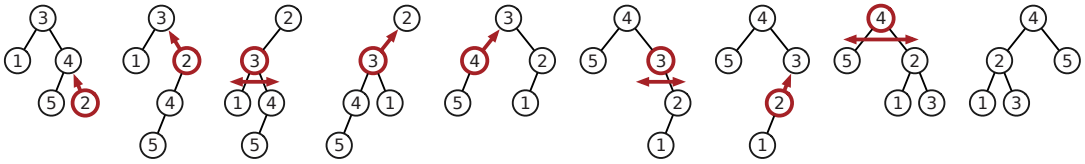
- Swap the left and right subtrees of an arbitrary node.



Swapping the subtrees of x

For both of these operations, some, all, or none of the subtrees A, B, and C may be empty.

The following example shows a five-node binary tree transforming into a binary search tree in eight operations:



"Sorting" a binary tree in eight steps: rotate 2, rotate 2, swap 3, rotate 3, rotate 4, swap 3, rotate 2, swap 4.

Your algorithm cannot directly modify parent or child pointers, and it cannot allocate new nodes or delete old nodes; the ***only*** way it can modify T is using rotations and swaps. On the other hand, you may *compute* anything you like for free, as long as that computation does not modify T. In other words, the running time of your algorithm is *defined* to be the number of rotations and swaps that it performs.

For full credit, your algorithm should use as few rotations and swaps as possible in the worst case. *[Hint: $O(n^2)$ operations is not too difficult, but we can do better.]*

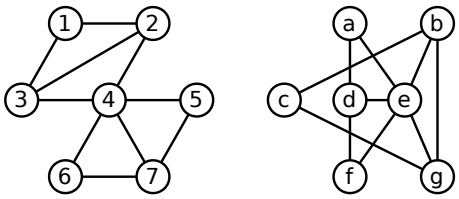
∽ New CS 473: Algorithms, Spring 2015 ∾
Homework 1

Due Tuesday, February 3, 2015 at 5pm

For this and all future homeworks, groups of up to three students can submit joint solutions. Please print (or typeset) the name and NetID of every group member on the first page of each submission.

---

1. Two graphs are ***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 every vertex in 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.

(a) Describe a polynomial-time reduction from EvenGraphIsomorphism to GraphIsomorphism.

(b) Describe a polynomial-time reduction from GraphIsomorphism to EvenGraphIsomorphism.

(c) Describe a polynomial-time reduction from GraphIsomorphism to SubgraphIsomorphism.

(d) Prove that SubgraphIsomorphism is NP-complete.

(e) What can you conclude about the NP-hardness of GraphIsomorphism? Justify your answer.

[Hint: These are all easy!]

2. Prove that the following problems are NP-hard.

   (a) Given an undirected graph G, does G have a spanning tree with at most 473 leaves?

   (b) Given an undirected graph $G = (V, E)$, what is the size of the largest subset of vertices $S \subseteq V$ such that at most 2015 edges in E have both endpoints in S?

3. The Hamiltonian cycle problem has two closely related variants:

   - UNDIRECTEDHAMCYCLE: Given an *undirected* graph G, does G contain an *undirected* Hamiltonian cycle?
   - DIRECTEDHAMCYCLE: Given an *directed* graph G, does G contain a *directed* Hamiltonian cycle?

   This question asks you to prove that these two problems are essentially equivalent.

   (a) Describe a polynomial-time reduction from UNDIRECTEDHAMCYCLE to DIRECTEDHAMCYCLE.

   (b) Describe a polynomial-time reduction from DIRECTEDHAMCYCLE to UNDIRECTEDHAMCYCLE.

*4. ***[Extra Credit]*** Describe a direct polynomial-time reduction from 4COLOR to 3COLOR. (This is a lot harder than the opposite direction!)

New CS 473: Algorithms, Spring 2015
Homework 2

Due Tuesday, February 10, 2015 at 5pm

---

1. The ***maximum k-cut problem*** asks, given a graph G with edge weights and an integer k as input, to compute a partition of the vertices of G into k disjoint subsets $S_1, S_2, \ldots, S_k$ such that the sum of the weights of the edges that cross the partition (that is, have endpoints in different subsets) is as large as possible.

   (a) Describe an efficient $(1 - 1/k)$-approximation algorithm for this problem.

   (b) Now suppose we wish to minimize the sum of the weights of edges that do *not* cross the partition. What approximation ratio does your algorithm from part (a) achieve for the new problem? Justify your answer.

2. In the ***bin packing*** problem, we are given a set of n items, each with weight between 0 and 1, and we are asked to load the items into as few bins as possible, such that the total weight in each bin is at most 1. It's not hard to show that this problem is NP-Hard; this question asks you to analyze a few common approximation algorithms. In each case, the input is an array $W[1..n]$ of weights, and the output is the number of bins used.

```
NextFit(W[1..n]):
  bins ← 0
  Total[0] ← ∞
  for i ← 1 to n
      if Total[bins] + W[i] > 1
          bins ← bins + 1
          Total[bins] ← W[i]
      else
          Total[bins] ← Total[bins] + W[i]
  return bins
```

```
FirstFit(W[1..n]):
  bins ← 0
  for i ← 1 to n
      j ← 1; found ← False
      while j ≤ bins and ¬found
          if Total[j] + W[i] ≤ 1
              Total[j] ← Total[j] + W[i]
              found ← True
          j ← j + 1
      if ¬found
          bins ← bins + 1
          Total[bins] = W[i]
  return bins
```

   (a) Prove that NextFit uses at most twice the optimal number of bins.

   (b) Prove that FirstFit uses at most twice the optimal number of bins.

   *(c) ***[Extra Credit]*** Prove that if the weight array W is initially sorted in decreasing order, then FirstFit uses at most $(4 \cdot OPT + 1)/3$ bins, where OPT is the optimal number of bins. The following facts may be useful (but you need to prove them if your proof uses them):

   - In the packing computed by FirstFit, every item with weight more than $1/3$ is placed in one of the first OPT bins.
   - FirstFit places at most $OPT - 1$ items outside the first OPT bins.

3. Consider the following greedy algorithm for the metric traveling salesman problem: Start at an arbitrary vertex, and then repeatedly travel to the closest unvisited vertex, until every vertex has been visited.

   (a) Prove that the approximation ratio for this algorithm is $O(\log n)$, where n is the number of vertices. *[Hint: Argue that the kth least expensive edge in the tour output by the greedy algorithm has weight at most* $\mathrm{OPT}/(n-k+1)$*; try* $k=1$ *and* $k=2$ *first.]*

   *(b) ***[Extra Credit]*** Prove that the approximation ratio for this algorithm is $\Omega(\log n)$. That is, describe an infinite family of weighted graphs such that the greedy algorithm returns a Hamiltonian cycle whose weight is $\Omega(\log n)$ times the weight of the optimal TSP tour.

∽ New CS 473: Algorithms, Spring 2015 ∾
Homework 3

Due Tuesday, February 17, 2015 at 5pm

---

1. A string w of parentheses **(** and **)** and brackets **[** and **]** is ***balanced*** if it satisfies one of the following conditions:

   - w is the empty string.
   - $w =$ **(**x**)** for some balanced string x
   - $w =$ **[**x**]** for some balanced string x
   - $w = xy$ for some balanced strings x and y

   For example, the string

   $$w = \texttt{([()][]())[()()]()}$$

   is balanced, because $w = xy$, where

   $$x = \texttt{([()] [] ())} \qquad \text{and} \qquad y = \texttt{[() ()] ()}.$$

   Describe and analyze an algorithm to compute the length of a longest balanced subsequence of a given string of parentheses and brackets. Your input is an array $w[1..n]$, where $w[i] \in \{$**(**, **)**, **[**, **]**$\}$ for every index i. (You may prefer to use different symbols instead of parentheses and brackets—for example, L, R, l, r or ◁, ▷, ◀, ▶—but please tell us what symbols you're using!)

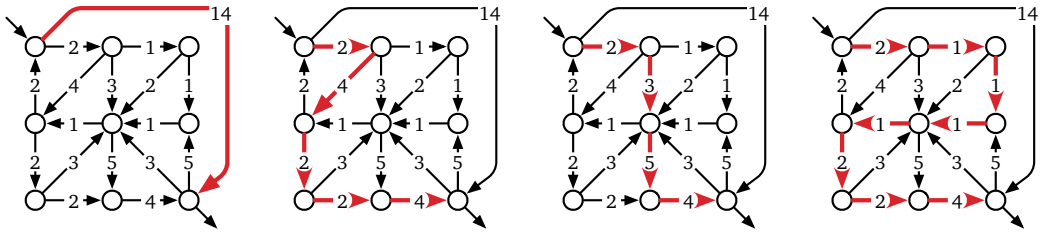
2. Congratulations! You've just been hired at internet giant Yeehaw! as the new party czar. The president of the company has asked you to plan the annual holiday party. Your task is to find exactly k employees to invite, including the president herself. Employees at Yeehaw! are organized into a strict hierarchy—a tree with the president at the root. The all-knowing oracles in Human Resources have determined two numerical values for every employee:

   - $With[i]$ measures much fun employee i would have at the party if their immediate supervisor is also invited.
   - $Without[i]$ measures how much fun employee i would have at the party if their immediate supervisor is *not* also invited.

   These values could be positive, negative, or zero, and $With[i]$ could be greater than, less than, or equal to $Without[i]$.

   Describe an algorithm that finds the set of k employees to invite that maximizes the sum of the k resulting "fun" values. The input to your algorithm is the tree T, the integer k, and the *With* and *Without* values for each employee. Assume that everyone invited to the party actually attends. Do *not* assume that T is a *binary* tree.

3. Although we typically speak of "the" shortest path between two nodes, a single graph could contain several minimum-length paths with the same endpoints.



Four (of many) equal-length shortest paths.

Describe and analyze an algorithm to determine the *number* of shortest paths from a source vertex s to a target vertex t in an arbitrary directed graph G with weighted edges. You may assume that all edge weights are positive and that all necessary arithmetic operations can be performed in $O(1)$ time.

[Hint: Compute shortest path distances from s to every other vertex. Throw away all edges that cannot be part of a shortest path from s to another vertex. What's left?]

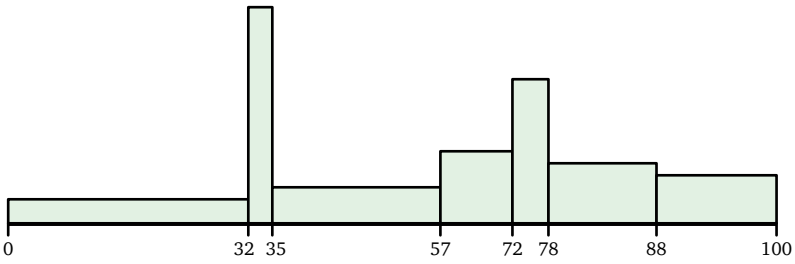
New CS 473: Algorithms, Spring 2015
Homework 4

Due Tuesday, February 24, 2015 at 5pm

Starting with this assignment, all homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. Suppose we want to summarize a large set S of values—for example, course averages for students in CS 105—using a variable-width histogram. To construct a histogram, we choose a sorted sequence of ***breakpoints*** $b_0 < b_1 < \cdots < b_k$, such that every element of S lies between b_0 and b_k. Then for each interval $[b_{i-1}, b_i]$, the histogram includes a rectangle whose height is the number of elements of S that lie inside that interval.



A variable-width histogram with seven bins.

Unlike a standard histogram, which requires the intervals to have equal width, we are free to choose the breakpoints arbitrarily. For statistical purposes, it is useful for the *areas* of the rectangles to be as close to equal as possible. To that end, define the ***cost*** of a histogram to be the sum of the *squares* of the rectangle areas; we want to compute the histogram with minimum cost.

More formally, suppose we fix a sequence of breakpoints $b_0 < b_1 < \cdots < b_k$. For each index i, let n_i denote the number of input values in the ith interval:

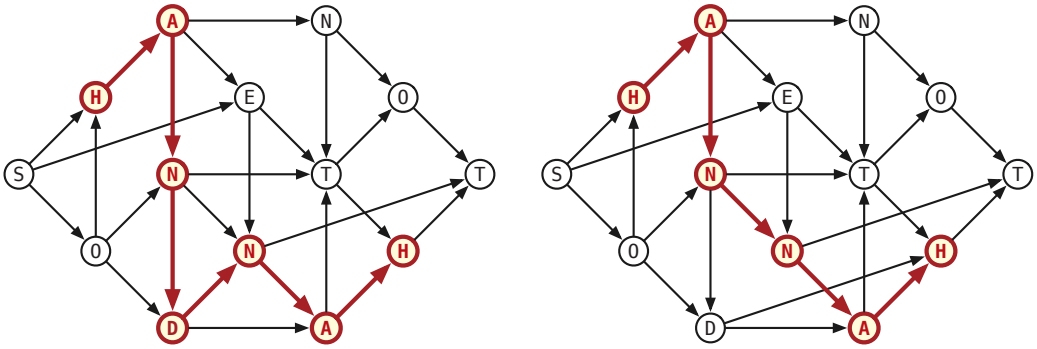
$$n_i := \#\left\{x \in S \;\middle|\; b_{i-1} \le x < b_i\right\}.$$

Then the ***cost*** of the resulting histogram is $\sum_{i=1}^{k} (n_i(b_i - b_{i-1}))^2$.

Describe and analyze an algorithm to compute a variable-width histogram with minimum cost for a given set of data values. Your input is an unsorted array $S[1..n]$ of ***distinct*** real numbers, all strictly between 0 and 1, and an integer k. Your algorithm should return a sorted array $B[0..k]$ of breakpoints that minimizes the cost of the resulting histogram, where $B[0] = 0$ and $B[k] = 1$***, and every other breakpoint $B[i]$ is equal to some input value $S[j]$***.

2. Suppose we are given a directed acyclic graph G with labeled vertices. Every path in G has a label, which is a string obtained by concatenating the labels of its vertices in order. Recall that a *palindrome* is a string that is equal to its reversal.

   Describe and analyze an algorithm to find the length of the longest palindrome that is the label of a path in G. For example, given the graph on the left below, your algorithm should return the integer 7, which is the length of the palindrome `HANDNAH`; given the graph on the right, your algorithm should return the integer 6, which is the length of the palindrome `HANNAH`.



∽ New CS 473: Algorithms, Spring 2015 ∾
Homework 5

Due Tuesday, March 10, 2015 at 5pm

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. On their long journey from Denmark to England, Rosencrantz and Guildenstern amuse themselves by playing the following game with a fair coin. First Rosencrantz flips the coin over and over until it comes up tails. Then Guildenstern flips the coin over and over until he gets as many heads in a row as Rosencrantz got on his turn. Here are three typical games:

   Rosencrantz: **H H** T
   Guildenstern: H T **H H**

   Rosencrantz: T
   Guildenstern: (no flips)

   Rosencrantz: **H H H** T
   Guildenstern: T H H T H H T H T T **H H H**

   (a) What is the *exact* expected number of flips in one of Rosencrantz's turns?

   (b) Suppose Rosencrantz happens to flip k heads in a row on his turn. What is the *exact* expected number of flips in Guildenstern's next turn?

   (c) What is the *exact* expected total number of flips (by both Rosencrantz and Guildenstern) in a single game?

   Include formal proofs that your answers are correct. If you have to appeal to "intuition" or "common sense", your answer is almost certainly wrong!

2. Recall from class that a ***priority search tree*** is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A ***heater*** is a priority search tree in which the *priorities* are given by the user, and the *search keys* are distributed uniformly and independently at random in the real interval $[0, 1]$. Intuitively, a heater is an "anti-treap".

   The following questions consider an n-node heater T whose priorities are the integers from 1 to n. Here we identify each node in T by its ***priority rank***, rather than by the rank of its search keys; for example, "node 5" means the node in T with the 5th smallest *priority*. In particular, the min-heap property implies that node 1 is the root of T. Finally, let i and j be arbitrary integers such that $1 \le i < j \le n$.

(a) What is the *exact* expected depth of node j in an n-node heater? Answering the following subproblems will help you:

i. Prove that in a uniformly random permutation of the $(i+1)$-element set $\{1, 2, \ldots, i, j\}$, elements i and j are adjacent with probability $2/(i+1)$.

ii. Prove that node i is an ancestor of node j with probability $2/(i+1)$. *[Hint: Use the previous question!]*

iii. What is the probability that node i is a *descendant* of node j? *[Hint: Do **not** use the previous question!]*

(b) Describe and analyze an algorithm to insert a new item into a heater. Express the expected running time of the algorithm in terms of the priority rank of the newly inserted item.

(c) Describe an algorithm to delete the minimum-priority item (the root) from an n-node heater. What is the expected running time of your algorithm?

3. Suppose we are given a two-dimensional array $M[1..n, 1..n]$ in which every row and every column is sorted in increasing order and no two elements 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.

Assume you have access to a subroutine RANDOM(k) that returns an integer chosen independently and uniformly at random from the set $\{1\,2, \ldots, k\}$, given an arbitrary positive integer k as input.

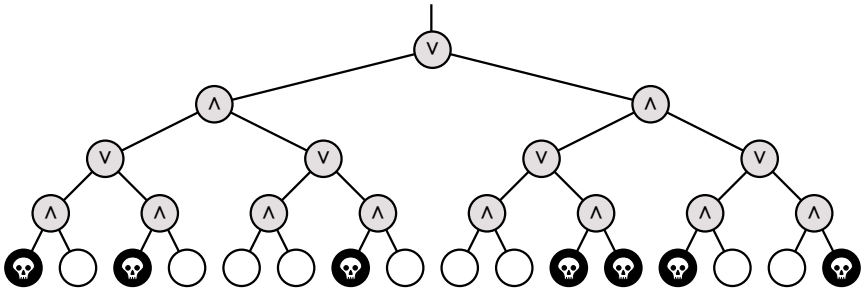
New CS 473: Algorithms, Spring 2015
Homework 6

Due Tuesday, March 17, 2015 at 5pm

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

(a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy, but be specific!]*

*(b) ***[Extra credit]*** Prove that *any* deterministic algorithm that correctly determines whether you can win *must* examine every leaf in the tree. It follows that any correct algorithm for part (a) *must* take $\Omega(4^n)$ time. *[Hint: Let Death cheat, but not in a way that the algorithm can detect.]*

(c) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $O(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*

*(d) ***[Extra credit]*** Describe and analyze a randomized algorithm that determines whether you can win in $O(c^n)$ expected time, for some explicit constant $c < 3$. Your analysis should yield an exact value for the constant c. *[Hint: You may not need to change your algorithm from part (b) at all!]*

2. A ***meldable priority queue*** stores a set of priorities from some totally-ordered universe (such as the integers) and supports the following operations:

   - MAKEQUEUE: Return a new priority queue containing the empty set.
   - FINDMIN(Q): Return the smallest element of Q (if any).
   - DELETEMIN(Q): Remove the smallest element in Q (if any).
   - INSERT(Q, x): Insert element x into Q, if it is not already there.
   - DECREASEPRIORITY(Q, x, y): Replace an element $x \in Q$ with a new element $y < x$. (If $y \geq x$, the operation fails.) The input includes a pointer directly to the node in Q containing x.
   - DELETE(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q containing x.
   - MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2. The elements of Q_1 and Q_2 could be arbitrarily intermixed; we do *not* assume, for example, that every element of Q_1 is smaller than every element of Q_2.

   A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a priority, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm. The input consists of pointers to the roots of the two trees.

```
MELD(Q1, Q2):
  if Q1 = NULL then return Q2
  if Q2 = NULL then return Q1

  if priority(Q1) > priority(Q2)
      swap Q1 ↔ Q2

  with probability 1/2
      left(Q1) ← MELD(left(Q1), Q2)
  else
      right(Q1) ← MELD(right(Q1), Q2)

  return Q1
```

   (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (not just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: What is the expected length of a random root-to-leaf path in an n-node binary tree, where each left/right choice is made with equal probability?]*

   (b) Prove that MELD(Q_1, Q_2) runs in $O(\log n)$ time with high probability. *[Hint: You can use Chernoff bounds, but the simpler identity $\binom{c}{k} \leq (ce)^k$ is actually sufficient.]*

   (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (It follows that every operation takes $O(\log n)$ time with high probability.)

∽ New CS 473: Algorithms, Spring 2015 ∾
Homework 7

Due Tuesday, March 31, 2015 at 5pm

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. ***Reservoir sampling*** is a method for choosing an item uniformly at random from an arbitrarily long stream of data; for example, the sequence of packets that pass through a router, or the sequence of IP addresses that access a given web page. Like all data stream algorithms, this algorithm must process each item in the stream quickly, using very little memory.

```
GetOneSample(stream S):
  ℓ ← 0
  while S is not done
      x ← next item in S
      ℓ ← ℓ + 1
      if Random(ℓ) = 1
          sample ← x        (⋆)
  return sample
```

   At the end of the algorithm, the variable ℓ stores the length of the input stream S; this number is *not* known to the algorithm in advance. If S is empty, the output of the algorithm is (correctly!) undefined.

   Consider an arbitrary non-empty input stream S, and let n denote the (unknown) length of S.

   (a) Prove that the item returned by GetOneSample(S) is chosen uniformly at random from S.

   (b) What is the *exact* expected number of times that GetOneSample(S) executes line ($\star$)?

   (c) What is the *exact* expected value of ℓ when GetOneSample(S) executes line ($\star$) for the *last* time?

   (d) What is the *exact* expected value of ℓ when either GetOneSample(S) executes line ($\star$) for the *second* time (or the algorithm ends, whichever happens first)?

   (e) Describe and analyze an algorithm that returns a subset of k distinct items chosen uniformly at random from a data stream of length at least k. The integer k is given as part of the input to your algorithm. Prove that your algorithm is correct.

   For example, if $k = 2$ and the stream contains the sequence $\langle ♠, ♥, ♦, ♣ \rangle$, the algorithm would return the subset $\{♦, ♠\}$ with probability $1/6$.

∽ New CS 473: Algorithms, Spring 2015 ∾
Homework 8

Due Tuesday, April 6, 2015 at 5pm

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. Suppose you are given a directed graph $G = (V, E)$, two vertices s and t, a capacity function $c : E \to \mathbb{R}^+$, and a second function $f : E \to \mathbb{R}$.

   (a) Describe and analyze an efficient algorithm to determine whether f is a maximum (s, t)-flow in G.

   (b) Describe and analyze an efficient algorithm to determine whether f is the *unique* maximum (s, t)-flow in G.

   Do not assume anything about the function f.

2. A new assistant professor, teaching maximum flows for the first time, suggested the following greedy modification to the generic Ford-Fulkerson augmenting path algorithm. Instead of maintaining a residual graph, the greedy algorithm just reduces the capacity of edges along the augmenting path. In particular, whenever the algorithm saturates an edge, that edge is simply removed from the graph.

```
GreedyFlow(G, c, s, t):
  for every edge e in G
      f(e) ← 0

  while there is a path from s to t in G
      π ← arbitrary path from s to t in G
      F ← minimum capacity of any edge in π
      for every edge e in π
          f(e) ← f(e) + F
          if c(e) = F
              remove e from G
          else
              c(e) ← c(e) − F

  return f
```

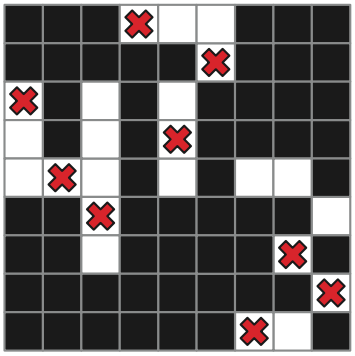
   (a) Prove that GreedyFlow does not always compute a maximum flow.

   (b) Prove that GreedyFlow is not even guaranteed to compute a good approximation to the maximum flow. That is, for any constant $\alpha > 1$, describe a flow network G such that the value of the maximum flow is more than α times the value of the flow computed by GreedyFlow. *[Hint: Assume that GreedyFlow chooses the worst possible path π at each iteration.]*

(c) Prove that for any flow network, if the Greedy Path Fairy tells you precisely which path π to use at each iteration, then GreedyFlow does compute a maximum flow. (Sadly, the Greedy Path Fairy does not actually exist.)

3. Suppose we are given an $n \times n$ square grid, some of whose squares are colored black and the rest white. Describe and analyze an algorithm to determine whether tokens can be placed on the grid so that

   - every token is on a white square;
   - every row of the grid contains exactly one token; and
   - every column of the grid contains exactly one token.



Your input is a two dimensional array $IsWhite[1..n, 1..n]$ of booleans, indicating which squares are white. Your output is a single boolean. For example, given the grid above as input, your algorithm should return True.

∽ New CS 473: Algorithms, Spring 2015 ∾
Homework 9

Due Tuesday, April 21, 2015 at 5pm

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. Suppose we are given an array $A[1..m][1..n]$ of real numbers. We want to *round* A to an integer array, 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:

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

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

2. 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.

   (a) Describe and analyse an algorithm to find the largest subset of the given boxes that can be nested so that only one box is visible.

   (b) Describe and analyze an algorithm to nest *all* the given boxes so that the number of visible boxes is as small as possible. *[Hint: Do **not** use part (a).]*

3. Describe and analyze an algorithm for the following problem, first posed and solved by the German mathematician Carl Jacobi in the early 1800s.

   > *Disponantur nn quantitates $h_k^{(i)}$ quaecunque in schema Quadrati, ita ut k habeantur n series horizontales et n series verticales, quarum quaeque est n terminorum. Ex illis quantitatibus eligantur n transversales, i.e. in seriebus horizontalibus simul atque verticalibus diversis positae, quod fieri potest* $1.2\ldots n$ *modis; ex omnibus illis modis quaerendum est is, qui summam n numerorum electorum suppeditet maximam.*

Carl Gustav Jacob Jacobi. De investigando ordine systematis aequationum differentialum vulgarium cujuscunque. *J. Reine Angew. Math.* 64(4):297–320, 1865. Posthumously published by Carl Borchardt.

For the few students who are not fluent in mid-19th century academic Latin, here is a modern English translation of Jacobi's problem. Suppose we are given an $n \times n$ matrix M. Describe and analyze an algorithm that computes a permutation σ that maximizes the sum $\sum_i M_{i,\sigma(i)}$, or equivalently, permutes the columns of M so that the sum of the elements along the diagonal is as large as possible.

Please do not submit your solution in mid-19th century academic Latin.

New CS 473: Algorithms, Spring 2015
Homework 10

Due Tuesday, April 28, 2015 at 5pm

This is the last graded homework.

1. Given points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ 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 the L_2 ***error***, defined as follows:

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

(This is the error metric *(ordinary/linear) least squares*.)

But there are several other ways of measuring how well a line fits a set of points, 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 the sum of the vertical distances from the given points to the line:

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

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

(b) The L_∞ ***error*** (or *maximum absolute deviation*) of the line $y = ax + b$ is the maximum vertical distance from any given point to the line::

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

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

2. (a) Give a linear-programming formulation of the maximum-cardinality bipartite matching problem. The input is a bipartite graph $G = (L \cup R, E)$, where every edge connects a vertex in L ("on the left") with a vertex in R ("on the right"). The output is the largest matching in G. Your linear program should have one variable for each edge.

(b) Now dualize the linear program from part (a). What do the dual variables represent? What does the objective function represent? What problem is this!?

3. An ***integer program*** is a linear program with the additional constraint that the variables must take only integer values. Prove that deciding whether a given integer program has a feasible solution is NP-hard. *[Hint: Any NP-complete decision problem can be formulated as an integer program. Choose your favorite!]*

New CS 473: Algorithms, Spring 2015
Homework 11

This homework will not be graded.

1. In Homework 10, we considered several different problems that can be solved by reducing them to a linear programming problem:

   - Finding a line that fits a given set of n points in the plane with minimum L_1 error.
   - Finding a line that fits a given set of n points in the plane with minimum L_∞ error.
   - Finding the largest matching in a bipartite graph.
   - Finding the smallest vertex cover in a bipartite graph.

   The specific linear programs are described in the homework solutions. For each of these linear programs, answer the following questions in the language of the original problem:

   (a) What is a basis?

   (b) (For the line-fitting problems only:) How many different bases are there?

   (c) What is a *feasible* basis?

   (d) What is a *locally optimal* basis?

   (e) What is a pivot?

2. Let $G = (V, E)$ be an arbitrary directed graph with weighted vertices; vertex weights may be positive, negative, or zero. A ***prefix*** of G is a subset $P \subseteq V$, such that there is no edge $u \to v$ where $u \notin P$ but $v \in P$. A ***suffix*** of G is the complement of a prefix. Finally, an ***interval*** of G is the intersection of a prefix of G and a suffix of G. The weight of a prefix, suffix, or interval is the sum of the weights of its vertices.

   (a) Describe a linear program that characterizes the maximum-weight prefix of G. Your linear program should have one variable per vertex, indicating whether that vertex is or is not in the chosen prefix.

   (b) Describe a linear program that characterizes the maximum-weight interval of G.

   [Hint: Don't worry about the solutions to your linear programs being integral; they will be. If all vertex weights are negative, the maximum-weight interval is empty; if all vertex weights are positive, the maximum-weight interval contains every vertex.]

Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheet with your answers.

1. Recall that a boolean formula is in *conjunctive normal form* if it is the conjunction (AND) of a series of *clauses*, each of which is a disjunction (OR) of a series of literals, each of which is either a variable or the negation of a variable. Consider the following variants of SAT:

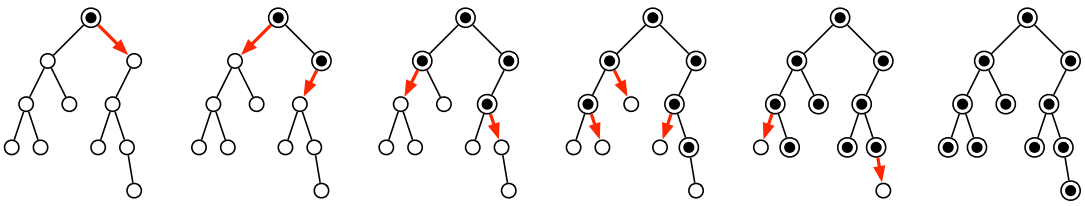
   - ***3SAT***: Given a boolean formula Φ in conjunctive normal form, such that every clause in Φ contains exactly *three* literals, is Φ satisfiable?
   - ***4SAT***: Given a boolean formula Φ in conjunctive normal form, such that every clause in Φ contains exactly *four* literals, is Φ satisfiable?

   (a) Describe a polynomial-time reduction from 3SAT to 4SAT.

   (b) Describe a polynomial-time reduction from 4SAT to 3SAT.

   Don't forget to ***prove*** that your reductions are correct!

2. Suppose we need to distribute a message to all the nodes in a given binary tree. Initially, only the root node knows the message. In a single round, each node that knows the message is allowed (but not required) forward it to at most one of its children. Describe and analyze an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in the tree.



A message being distributed through a binary tree in five rounds.

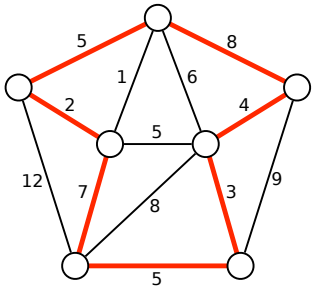
3. The ***maximum acyclic subgraph*** problem is defined as follows: The input is a directed graph $G = (V, E)$ with n vertices. Our task is to label the vertices from 1 to n so that the number of edges $u \to v$ with $label(u) < label(v)$ is as large as possible. Solving this problem exactly is NP-hard.

   (a) Describe and analyze an efficient 2-approximation algorithm for this problem.

   (b) ***Prove*** that the approximation ratio of your algorithm is at most 2.

   [Hint: Find an ordering of the vertices such that at least half of the edges point forward. Why is that enough?]

4. 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-hard.



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

5. Lenny Rutenbar, founding dean of the new Maximilian Q. Levchin College of Computer Science, has commissioned a series of snow ramps on the south slope of the Orchard Downs sledding hill and challenged William (Bill) Sanders, head of the Department of Electrical and Computer Engineering, to a sledding contest. Bill and Lenny will both sled down the hill, each trying to maximize their air time. The winner gets to expand their department/college into both Siebel Center and the new ECE Building; the loser has to move their entire department/college under the Boneyard bridge next to Everitt Lab.

   Whenever Lenny or Bill reaches a ramp *while on the ground*, they can either use that ramp to jump through the air, possibly flying over one or more ramps, or sled past that ramp and stay on the ground. Obviously, if someone flies over a ramp, they cannot use that ramp to extend their jump.

   Suppose you are given a pair of arrays $Ramp[1..n]$ and $Length[1..n]$, where $Ramp[i]$ is the distance from the top of the hill to the ith ramp, and $Length[i]$ is the distance that a sledder who takes the ith ramp will travel through the air. Describe and analyze an algorithm to determine the maximum total distance that Lenny or Bill can spend in the air.

The north slope is faster, but too short for an interesting contest.

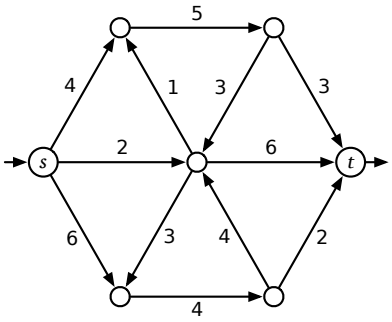
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheet with your answers.

1. Clearly indicate the following structures in the directed graph on the right. Some of these subproblems may have more than one correct answer.

   (a) A maximum (s,t)-flow f.

   (b) The residual graph of f.

   (c) A minimum (s,t)-cut.



2. Recall that a family $\mathcal{H}$ of hash functions is ***universal*** if $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] \le 1/m$ for all distinct items $x \neq y$, where m is the size of the hash table. For any fixed hash function h, a ***collision*** is an unordered pair of distinct items $x \neq y$ such that $h(x) = h(y)$.

   Suppose we hash a set of n items into a table of size $m = 2n$, using a hash function h chosen uniformly at random from some universal family. Assume $\sqrt{n}$ is an integer.

   (a) ***Prove*** that the expected number of collisions is at most $n/4$.

   (b) ***Prove*** that the probability that there are at least $n/2$ collisions is at most $1/2$.

   (c) ***Prove*** that the probability that any subset of more than $\sqrt{n}$ items all hash to the same address is at most $1/2$. *[Hint: Use part (b).]*

   (d) ***[The actual exam question assumed only pairwise independence of hash values; under this weaker assumption, the claimed result is actually false. Everybody got extra credit for this part.]***

   Now suppose we choose h at random from a ***strongly 4-universal*** family of hash functions, which means for all distinct items w,x,y,z and all addresses i,j,k,l, we have

$$\Pr_{h\in\mathcal{H}}\big[h(w) = i \ \wedge\ h(x) = j \ \wedge\ h(y) = k \ \wedge\ h(z) = \ell\big] = \frac{1}{m^4}.$$

   Prove that the probability that any subset of more than $\sqrt{n}$ items all hash to the same address is at most $O(1/n)$.

[Hint: Use Markov's and Chebyshev's inequalities. All four statements have short elementary proofs.]

3. Suppose we have already computed a maximum flow f^* in a flow network G with *integer* capacities. Assume all flow values $f^*(e)$ are integers.

   (a) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is *increased* by 1.

   (b) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is *decreased* by 1.

   Your algorithms should be significantly faster than recomputing the maximum flow from scratch.

4. Let T be a treap with n vertices.

   (a) What is the *exact* expected number of leaves in T?

   (b) What is the *exact* expected number of nodes in T that have two children?

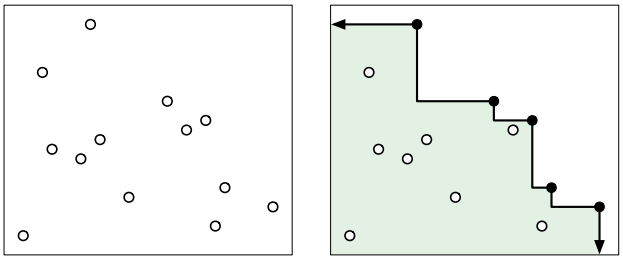
   (c) What is the *exact* expected number of nodes in T that have exactly one child?

   You do *not* need to prove that your answers are correct. *[Hint: What is the probability that the node with the kth smallest search key has no children, one child, or two children?]*

5. There is no problem 5.

Write your answers in the separate answer booklet.
You may take this question sheet with you when you leave.

1. Let S be an arbitrary set of n points in the plane. A point p in S is ***Pareto-optimal*** if no other point in S is both above and to the right. The ***staircase*** of S is the set of all points in the plane (not just in S) that have at least one point in S both above and to the right. All Pareto-optimal points lie on the boundary of the staircase.



A set of points in the plane and its staircase (shaded), with Pareto-optimal points in black.

   (a) Describe and analyze an algorithm that computes the staircase of S in $O(n \log n)$ time.

   (b) Suppose each point in S is chosen independently and uniformly at random from the unit square $[0,1] \times [0,1]$. What is the ***exact*** expected number of Pareto-optimal points in S?

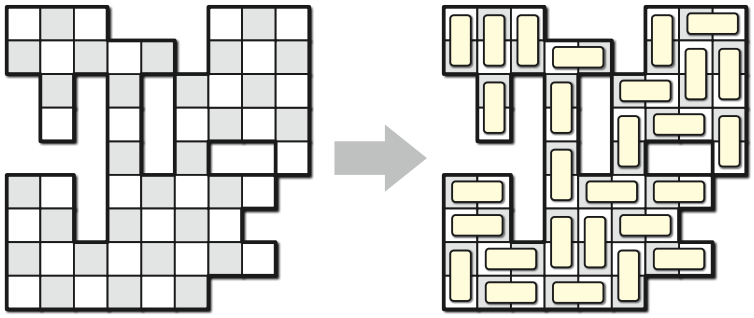
2. Let $G = (V, E)$ be a directed graph, in which every edge has capacity equal to 1 and some arbitrary cost. Edge costs could be positive, negative, or zero. Suppose you have just finished computing the minimum-cost circulation in this graph. Unfortunately, after all that work, now you realize that you recorded the direction of one of the edges incorrectly!

   Describe and analyze an algorithm to update the minimum-cost circulation in G when the direction of an arbitrary edge in G is reversed. The input to your algorithm consists of the directed graph G, the costs of edges in G, the minimum-cost circulation in G, and the edge to be reversed. Your algorithm should be faster than recomputing the minimum-cost circulation from scratch.

3. The ***chromatic number*** $\chi(G)$ of an undirected graph G is the minimum number of colors required to color the vertices, so that every edge has endpoints with different colors. Computing the chromatic number exactly is NP-hard, because 3Color is NP-hard.

   Prove that the following problem is also NP-hard: Given an arbitrary undirected graph G, return any integer between $\chi(G)$ and $\chi(G) + 473$.

4. Suppose you are given an $n \times n$ checkerboard with some of the squares deleted. You have a large set of dominos, just the right size to cover two squares of the checkerboard. Describe and analyze an algorithm to determine whether it is possible to tile the remaining squares with dominos—each domino must cover exactly two undeleted squares, and each undeleted square must be covered by exactly one domino.



Your input is a two-dimensional array $Deleted[1..n, 1..n]$ of bits, where $Deleted[i, j] = \text{TRUE}$ if and only if the square in row i and column j has been deleted. Your output is a single bit; you do ***not*** have to compute the actual placement of dominos. For example, for the board shown above, your algorithm should return TRUE.

5. Recall from Homework 11 that a ***prefix*** of a directed graph $G = (V, E)$ is a subset $P \subseteq V$ of the vertices such that no edge $u \to v \in E$ has $u \notin P$ and $v \in P$.

   Suppose you are given a rooted tree T, with all edges directed away from the root; every vertex in T has a weight, which could be positive, negative, or zero. Describe and analyze a *self-contained* algorithm to compute the prefix of T with maximum total weight. *[Hint: Don't use linear programming.]*

6. Suppose we are given a sequence of n linear inequalities of the form $a_i x + b_i y \le c_i$; the set of all points (x, y) that satisfy these inequalities is a convex polygon P in the (x, y)-plane. Describe a linear program whose solution describes the largest square with horizontal and vertical sides that lies inside P. (You can assume that P is non-empty.)