CS 473 ✧ Spring 2017

ᔕ Homework 0 ᔓ

Due Wednesday, January 25, 2017 at 8pm

- **This homework tests your familiarity with prerequisite material:** designing, describing, and analyzing elementary algorithms; fundamental graph problems and algorithms; 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.
- **Submit your solutions electronically on Gradescope as PDF files.**
  - Submit a separate file for each numbered problem.
  - You can find a LaTeX solution template on the course web site (soon); please use it if you plan to typeset your homework.
  - If you must submit scanned handwritten solutions, please use dark ink (not pencil) on blank white printer paper (not notebook or graph paper), use a high-quality scanner (not a phone camera), and print the resulting PDF file on a black-and-white printer to verify readability before you submit.

☞ 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.
- The answer ***"I don't know"*** (and *nothing* else) is worth 25% partial credit on any problem or subproblem, on any homework or exam, except for extra-credit problems. We will accept synonyms like "No idea" or "WTF" or "¯\(•_•)/¯", but you must write *something*.
- **Avoid the Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an ***automatic zero*** on any homework or exam problem. Yes, really.
  - Always give complete solutions, not just examples.
  - Every algorithm requires an English specification.
  - Greedy algorithms require formal correctness proofs.
  - Never use weak induction.

See the course web site for more information.

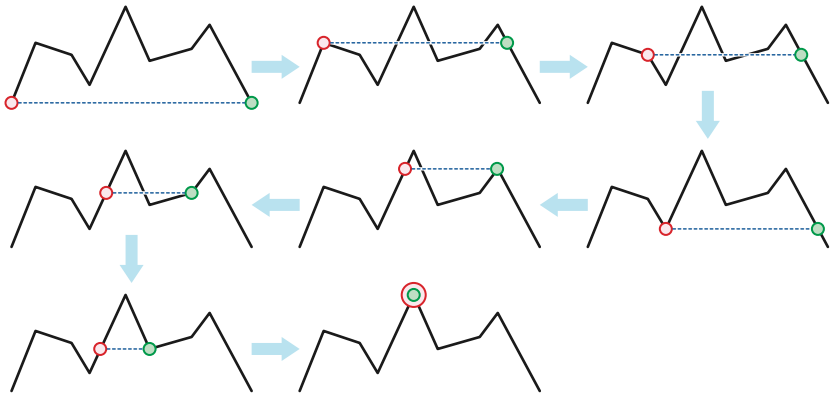
If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

---

1. Every cheesy romance movie has a scene where the romantic couple, after a long and frustrating separation, suddenly see each other across a long distance, and then slowly approach one another with unwavering eye contact as the music rolls in and the rain lifts and the sun shines through the clouds and the music swells and everyone starts dancing with rainbows and kittens and chocolate unicorns and. . . .[1]

   Suppose a romantic couple—in grand computer science tradition, named Alice and Bob—enters their favorite park at the east and west entrances and immediately establish eye-contact. They can't just run directly to each other; instead, they must stay on the path that zig-zags through the part between the east and west entrances. To maintain the proper dramatic tension, Alice and Bob must traverse the path so that they always lie on a direct east-west line.

   We can describe the zigzag path as two arrays $X[0..n]$ and $Y[0..n]$, containing the x- and y-coordinates of the corners of the path, in order from the southwest endpoint to the southeast endpoint. The X array is sorted in increasing order, and $Y[0] = Y[n]$. The path is a sequence of straight line segments connecting these corners.



Alice and Bob meet. Alice walks backward in step 2, and Bob walks backward in steps 5 and 6.

   (a) Suppose $Y[0] = Y[n] = 0$ and $Y[i] > 0$ for every other index i; that is, the endpoints of the path are strictly below every other point on the path. Prove that under these conditions, Alice and Bob can meet.

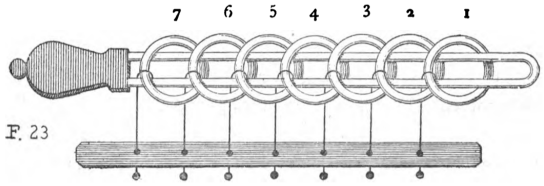
   [Hint: Describe a graph that models all possible locations and transitions of the couple along the path. What are the vertices of this graph? What are the edges? What can you say about the degrees of the vertices?]

   (b) If the endpoints of the path are *not* below every other vertex, Alice and Bob might still be able to meet, or they might not. Describe an algorithm to decide whether Alice and Bob can meet, without either breaking east-west eye contact or stepping off the path, given the arrays $X[0..n]$ and $Y[0..n]$ as input.

   *[Hint: Build the graph from part (a). (How?) What problem do you need to solve on this graph? Call a textbook algorithm to solve that problem. (Do **not** regurgitate the textbook algorithm.) What is your overall running time as a function of n?]*

[1] Fun fact: Damien Chazelle, the director of *Whiplash* and *La La Land*, is the son of Princeton computer science professor Bernard Chazelle.

2. The Tower of Hanoi is a relatively recent descendant of a much older mechanical puzzle known as the Chinese rings, Baguenaudier (a French word meaning "to wander about aimlessly"), Meleda, Patience, Tiring Irons, Prisoner's Lock, Spin-Out, and many other names. This puzzle was already well known in both China and Europe by the 16th century. The Italian mathematician Luca Pacioli described the 7-ring puzzle and its solution in his unpublished treatise *De Viribus Quantitatis*, written between 1498 and 1506;[1] only a few years later, the Ming-dynasty poet Yang Shen described the 9-ring puzzle as "a toy for women and children".



A drawing of a 7-ring Baguenaudier, from *Récréations Mathématiques* by Édouard Lucas (1891)

The Baguenaudier puzzle has many physical forms, but it typically consists of a long metal loop and several rings, which are connected to a solid base by movable rods. The loop is initially threaded through the rings as shown in the figure above; the goal of the puzzle is to remove the loop.

More abstractly, we can model the puzzle as a sequence of bits, one for each ring, where the ith bit is `1` if the loop passes through the ith ring and `0` otherwise. (Here we index the rings from right to left, as shown in the figure.) The puzzle allows two legal moves:

- You can always flip the 1st (= rightmost) bit.
- If the bit string ends with exactly i `0`s, you can flip the $(i+2)$th bit.

The goal of the puzzle is to transform a string of n `1`s into a string of n `0`s. For example, the following sequence of 21 moves solve the 5-ring puzzle:

$$\texttt{11111} \xrightarrow{1} \texttt{11110} \xrightarrow{3} \texttt{11010} \xrightarrow{1} \texttt{11011} \xrightarrow{2} \texttt{11001} \xrightarrow{1} \texttt{11000} \xrightarrow{5} \texttt{01000}$$
$$\xrightarrow{1} \texttt{01001} \xrightarrow{2} \texttt{01011} \xrightarrow{1} \texttt{01010} \xrightarrow{3} \texttt{01110} \xrightarrow{1} \texttt{01111} \xrightarrow{2} \texttt{01101} \xrightarrow{1} \texttt{01100} \xrightarrow{4} \texttt{00100}$$
$$\xrightarrow{1} \texttt{00101} \xrightarrow{2} \texttt{00111} \xrightarrow{1} \texttt{00110} \xrightarrow{3} \texttt{00010} \xrightarrow{1} \texttt{00011} \xrightarrow{2} \texttt{00001} \xrightarrow{1} \texttt{00000}$$

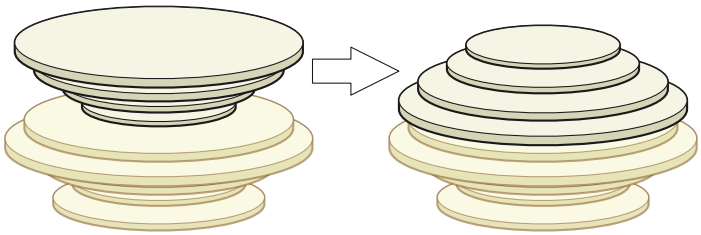
(a) Describe an algorithm to solve the Baguenaudier puzzle. Your input is the number of rings n; your algorithm should print a sequence of moves that solves the n-ring puzzle. For example, given the integer 5 as input, your algorithm should print the sequence $1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1$.

(b) *Exactly* how many moves does your algorithm perform, as a function of n? Prove your answer is correct.

(c) ***[Extra credit]*** Call a sequence of moves *reduced* if no move is the inverse of the previous move. Prove that for any non-negative integer n, there is *exactly one* reduced sequence of moves that solves the n-ring Baguenaudier puzzle. *[Hint: See problem 1!]*

[1] *De Viribus Quantitatis* [*On the Powers of Numbers*] is an important early work on recreational mathematics and perhaps the oldest surviving treatise on magic. Pacioli is better known for *Summa de Aritmetica*, a near-complete encyclopedia of late 15th-century mathematics, which included the first description of double-entry bookkeeping.

3. 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 using as few flips as possible. *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. *Exactly* how many flips does your algorithm perform in the worst case?

*[Hint: This problem has **nothing** to do with the Tower of Hanoi!]*

CS 473 ✧ Spring 2017

Homework 1

Due Wednesday, February 1, 2017 at 8pm

Starting with this homework, groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

---

1. A palindrome is any string that is exactly the same as its reversal, like `I`, or `DEED`, or `RACECAR`, or `AMANAPLANACATACANALPANAMA`.

   Any string can be decomposed into a sequence of palindromes. For example, the string `BUBBASEESABANANA` ("Bubba sees a banana.") can be broken into palindromes in the following ways (and many others):

   `BUB` • `BASEESAB` • `ANANA`

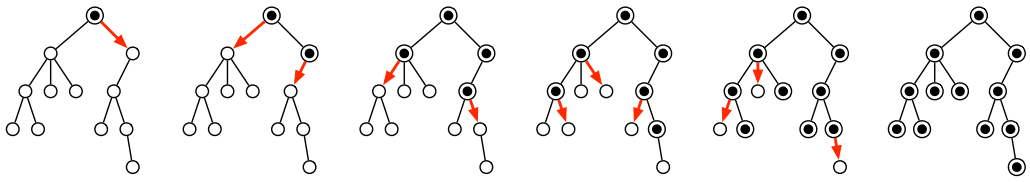
   `B` • `U` • `BB` • `A` • `SEES` • `ABA` • `NAN` • `A`

   `B` • `U` • `BB` • `A` • `SEES` • `A` • `B` • `ANANA`

   `B` • `U` • `B` • `B` • `A` • `S` • `E` • `E` • `S` • `A` • `B` • `A` • `N` • `ANA`

   Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string `BUBBASEESABANANA`, your algorithm would return the integer 3.

2. 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.



A message being distributed through a tree in five rounds.

   (a) Describe an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in a ***binary*** tree.

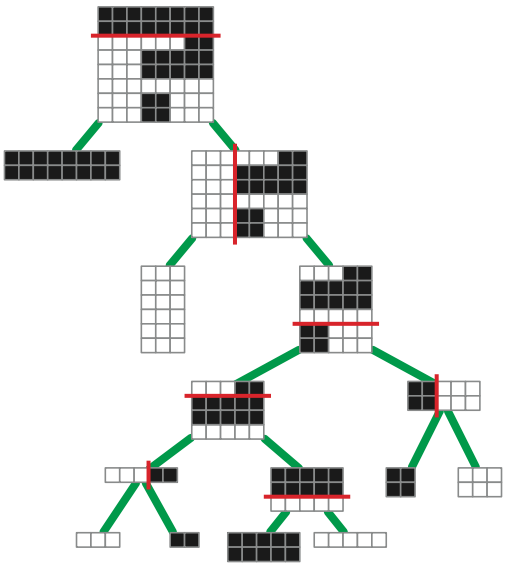
   (b) Describe an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in an ***arbitrary rooted*** tree.

   [Hint: Don't forget to justify your algorithm's correctness; you may find the lecture notes on greedy algorithms helpful. Any algorithm for this part also solves part (a).]

3. Suppose you are given an $m \times n$ bitmap, represented by an array $M[1..m, 1..n]$ of 0s and 1s. A *solid block* in M is a contiguous subarray $M[i..i', j..i']$ in which all bits are equal.

   A *guillotine subdivision* is a compact data structure to represent bitmaps as a recursive decomposition into solid blocks. If the entire bitmap M is a solid block, there is nothing to do. Otherwise, we cut M into two smaller bitmaps along a horizontal or vertical line, and then decompose the two smaller bitmaps recursively.□

   Any guillotine subdivision can be represented as a binary tree, where each internal node stores the position and orientation of a cut, and each leaf stores a single bit 0 or 1 indicting the contents of the corresponding block. The *size* of a guillotine subdivision is the number of leaves in the corresponding binary tree (that is, the final number of solid blocks), and the *depth* of a guillotine subdivision is the depth of the corresponding binary tree.



A guillotine subdivision with size 8 and depth 5.

   (a) Describe and analyze an algorithm to compute a guillotine subdivision of M of minimum *size*.

   (b) Describe and analyze an algorithm to compute a guillotine subdivision of M of minimum *depth*.

□Guillotine subdivisions are similar to kd-trees, except that the cuts in a guillotine subdivision are *not* required to alternate between horizontal and vertical.

Standard dynamic programming rubric. For problems worth 10 poins:

- 3 points for a clear **English** specification of the recursive function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.)
  - + 2 points for describing the function itself. (For example: "$OPT(i, j)$ is the edit distance between $A[1..i]$ and $B[1..i]$.")
  - + 1 point for stating how to call your recursive function to get the final answer. (For example: "We need to compute $OPT(m, n)$.")
  - + An English description of the ***algorithm*** is not sufficient. We want an English description of the underlying recursive ***problem***. In particular, the description should specify precisely the role of each input parameter.
  - + No credit for the rest of the problem if the English description is is missing. (This is a Deadly Sin.)
- 4 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 point for base case(s). −½ for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for recursive case(s). −1 for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 3 points for details of the iterative dynamic programming algorithm
  - + 1 point for describing the memoization data structure
  - + 1 point for describing a correct evaluation order; a clear picture is usually sufficient. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for time analysis
- It is *not* necessary to state a space bound.
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem says otherwise.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, ***but iterative psuedocode is not required for full credit***. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. (But you still need to specify the underlying recursive function in English.)
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of n. Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.

  We rarely include these target time bounds in the actual questions, because when we have included them, significantly more students turned in algorithms that meet the target time bound but didn't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).

CS 473 ✧ Spring 2017

Homework 2

Due Wednesday, February 8, 2017 at 8pm

There are only two problems, but the first one counts double.

---

1. Suppose you are given a two-dimensional array $M[1..n, 1..n]$ of numbers, which could be positive, negative, or zero, and which are *not* necessarily integers. The ***maximum subarray problem*** asks to find the largest sub of elements in any contiguous subarray of the form $M[i..i', j..j']$. In this problem we'll develop an algorithm for the maximum subarray problem that runs in $O(n^3)$ time.

   The algorithm is a combination of divide and conquer and dynamic programming. Let L be a horizontal line through M that splits the rows (roughly) in half. After some preprocessing, the algorithm finds the maximum-sum subarray that crosses L, the maximum-sum subarray above L, and the maximum-sum subarray below L. The first subarray is found by dynamic programming; the last two subarrays are found recursively.

   (a) For any indices i and j, let $Sum(i, j)$ denote the sum of all elements in the subarray $M[1..i, 1..j]$. Describe an algorithm to compute $Sum(i, j)$ for all indices i and j in $O(n^2)$ time.

   (b) Describe a simple(!!) algorithm to solve the maximum subarray problem in $O(n^4)$ time, using the output of your algorithm for part (a).

   (c) Describe an algorithm to find the maximum-sum subarray ***that crosses L*** in $O(n^3)$ time, using the output of your algorithm for part (a). *[Hint: Consider the top half and the bottom half of M separately.]*

   (d) Describe a divide-and-conquer algorithm to find the maximum-sum subarray in M in $O(n^3)$ time, using your algorithm for part (c) as a subroutine. *[Hint: Why is the running time $O(n^3)$ and not $O(n^3 \log n)$?]*

   In fact, the subproblem in part (c) — and thus the entire maximum subarray problem — can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ time using a recent algorithm of Ryan Williams. Williams' algorithm can also be used to compute all-pairs shortest paths in the same slightly subcubic running time. The divide-ando-conquer strategy itself is due to Tadao Takaoka.

   There is a simpler $O(n^3)$-time algorithm for the maximum subarray problem, based on Kadane's $O(n)$-time algorithm for the one-dimensional problem. (For every pair of indices i and i', find the best subarray of the form $M[i..i', j..j']$ in $O(n)$ time.) It's unclear whether this approach can be sped up using Williams' algorithm (or its predecessors) without the divide-and-conquer layer.

   An algorithm for the maximum subarray problem (or all-pairs shortest paths) that runs in $O(n^{2.99999999})$ time would be a major breakthrough.

2. The Doctor and River Song decide to play a game on a directed acyclic graph G, which has one source s and one sink t.[1]

   Each player has a token on one of the vertices of G. At the start of the game, The Doctor's token is on the source vertex s, and River's token is on the sink vertex t. The players alternate turns, with The Doctor moving first. On each of his turns, the Doctor moves his token forward along a directed edge; on each of her turns, River moves her token *backward* along a directed edge.

   If the two tokens ever meet on the same vertex, River wins the game. ("Hello, Sweetie!") If the Doctor's token reaches t or River's token reaches s before the two tokens meet, then the Doctor wins the game.

   Describe and analyze an algorithm to determine who wins this game, assuming both players play perfectly. That is, if the Doctor can win *no matter how River moves*, then your algorithm should output "Doctor", and if River can win *no matter how the Doctor moves*, your algorithm should output "River". (Why are these the only two possibilities?) The input to your algorithm is the graph G.

[1] The labels s and t may be abbreviations for the Untempered **S**chism and the **T**ime Vortex, or the Shining World of the Seven Systems (otherwise known as Gallifrey) and Trenzalore, or Skaro and Telos, or Something else Timey-wimey.

CS 473 ✧ Spring 2017

Homework 5

Due Wednesday, February 15, 2017 at 8pm

---

0. **[Warmup only; do not submit solutions]**

   After sending his loyal friends Rosencrantz and Guildenstern off to Norway, Hamlet decides to amuse himself by repeatedly flipping a fair coin until the sequence of flips satisfies some condition. For each of the following conditions, compute the *exact* expected number of flips until that condition is met.

   (a) Hamlet flips heads.

   (b) Hamlet flips both heads and tails (in different flips, of course).

   (c) Hamlet flips heads twice.

   (d) Hamlet flips heads twice in a row.

   (e) Hamlet flips heads followed immediately by tails.

   (f) Hamlet flips more heads than tails.

   (g) Hamlet flips the same positive number of heads and tails.

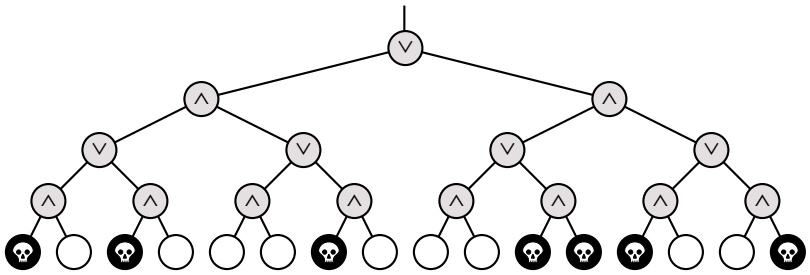
   [Hint: Be careful! If you're relying on intuition instead of a proof, you're probably wrong.]

1. Consider the following non-standard algorithm for shuffling a deck of n cards, initially numbered in order from 1 on the top to n on the bottom. At each step, we remove the top card from the deck and ***insert*** it randomly back into in the deck, choosing one of the n possible positions uniformly at random. The algorithm ends immediately after we pick up card $n-1$ and insert it randomly into the deck.

   (a) Prove that this algorithm uniformly shuffles the deck, meaning each permutation of the deck has equal probability. *[Hint: Prove that at all times, the cards below card $n-1$ are uniformly shuffled.]*

   (b) What is the *exact* expected number of steps executed by the algorithm? *[Hint: Split the algorithm into phases that end when card $n-1$ changes position.]*

2. 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!]*

(b) 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 $n = 1$.]*

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

3. The following randomized variant of "one-armed quicksort" selects the kth smallest element in an unsorted array $A[1..n]$. As usual, PARTITION$(A[1..n], p)$ partitions the array A into three parts by comparing the pivot element $A[p]$ to every other element, using $n-1$ comparisons, and returns the new index of the pivot element.

```
QUICKSELECT(A[1..n], k):
  r ← PARTITION(A[1..n], RANDOM(n))
  if k < r
      return QUICKSELECT(A[1..r−1], k)
  else if k > r
      return QUICKSELECT(A[r+1..n], k−r)
  else
      return A[k]
```

(a) State a recurrence for the expected running time of QUICKSELECT, as a function of n and k.

(b) What is the *exact* probability that QUICKSELECT compares the ith smallest and jth smallest elements in the input array? The correct answer is a simple function of i, j, and k. *[Hint: Check your answer by trying a few small examples.]*

(c) What is the *exact* probability that in one of the recursive calls to QUICKSELECT, the first argument is the subarray $A[i..j]$? The correct answer is a simple function of i, j, and k. *[Hint: Check your answer by trying a few small examples.]*

(d) Show that for any n and k, the expected running time of QUICKSELECT is $\Theta(n)$. You can use either the recurrence from part (a) or the probabilities from part (b) or (c).

CS 473 ✧ Spring 2017

Homework 4

Due Wednesday, March 1, 2017 at 8pm

---

1. Recall 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 a sort of anti-treap.

   The following problems consider an n-node heater T whose priorities are the integers from 1 to n. We identify nodes in T by their *priorities*; thus, "node 5" means the node in T with *priority* 5. For example, the min-heap property implies that node 1 is the root of T. Finally, let i and j be integers with $1 \le i < j \le n$.

   (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 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. Analyze the expected running time as a function of the number of nodes.

   (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?

2. Suppose we are given a coin that may or may not be biased, and we would like to compute an accurate *estimate* of the probability of heads. Specifically, if the actual unknown probability of heads is p, we would like to compute an estimate $\tilde{p}$ such that

$$\Pr[|\tilde{p} - p| > \varepsilon] < \delta$$

   where ε is a given ***accuracy*** or ***error*** parameter, and δ is a given ***confidence*** parameter.

   The following algorithm is a natural first attempt; here FLIP( ) returns the result of an independent flip of the unknown coin.

```
MEANESTIMATE(ε):
  count ← 0
  for i ← 1 to N
      if FLIP( ) = HEADS
          count ← count + 1
  return count/N
```

   (a) Let $\tilde{p}$ denote the estimate returned by MEANESTIMATE(ε). Prove that $\mathrm{E}[\tilde{p}] = p$.

(b) Prove that if we set $N = \lceil \alpha/\varepsilon^2 \rceil$ for some appropriate constant α, then we have $\Pr[|\tilde{p} - p| > \varepsilon] < 1/4$. *[Hint: Use Chebyshev's inequality.]*

(c) We can increase the previous estimator's confidence by running it multiple times, independently, and returning the *median* of the resulting estimates.

```
MedianOfMeansEstimate(δ, ε):
  for j ← 1 to K
      estimate[j] ← MeanEstimate(ε)
  return Median(estimate[1..K])
```

Let p^* denote the estimate returned by MedianOfMeansEstimate(δ, ε). Prove that if we set $N = \lceil \alpha/\varepsilon^2 \rceil$ (inside MeanEstimate) and $K = \lceil \beta \ln(1/\delta) \rceil$, for some appropriate constants α and β, then $\Pr[|p^* - p| > \varepsilon] < \delta$. *[Hint: Use Chernoff bounds.]*

CS 473 ✧ Spring 2017

Homework 5

Due Wednesday, March 8, 2017 at 8pm

---

1. ***Reservoir sampling*** is a method for choosing an item uniformly at random from an arbitrarily long stream of data.

```
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.

   In the following, 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 should return the subset $\{♦, ♠\}$ with probability $1/6$.

2. ***Tabulated hashing*** uses tables of random numbers to compute hash values. Suppose $|\mathcal{U}| = 2^w \times 2^w$ and $m = 2^\ell$, so the items being hashed are pairs of w-bit strings (or $2w$-bit strings broken in half) and hash values are ℓ-bit strings.

   Let $A[0..2^w - 1]$ and $B[0..2^w - 1]$ be arrays of independent random ℓ-bit strings, and define the hash function $h_{A,B} : \mathcal{U} \to [m]$ by setting

$$h_{A,B}(x, y) := A[x] \oplus B[y]$$

   where $\oplus$ denotes bit-wise exclusive-or. Let $\mathcal{H}$ denote the set of all possible functions $h_{A,B}$. Filling the arrays A and B with independent random bits is equivalent to choosing a hash function $h_{A,B} \in \mathcal{H}$ uniformly at random.

(a) Prove that $\mathscr{H}$ is 2-uniform.

(b) Prove that $\mathscr{H}$ is 3-uniform. *[Hint: Solve part (a) first.]*

(c) Prove that $\mathscr{H}$ is ***not*** 4-uniform.

Yes, "see part (b)" is worth full credit for part (a), but only if your solution to part (b) is correct.

CS 473 ✧ Spring 2017

Homework 6

Due Wednesday, March 16, 2017 at 8pm

---

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 $f : E \to \mathbb{R}$. Describe and analyze an algorithm to determine whether f is a maximum (s, t)-flow in G. *[Hint: Don't make any "obvious" assumptions!]*

2. Suppose you are given a flow network G with ***integer*** edge capacities and an ***integer*** maximum flow f^* in G. Describe algorithms for the following operations:

   (a) INCREMENT(e): Increase the capacity of edge e by 1 and update the maximum flow.

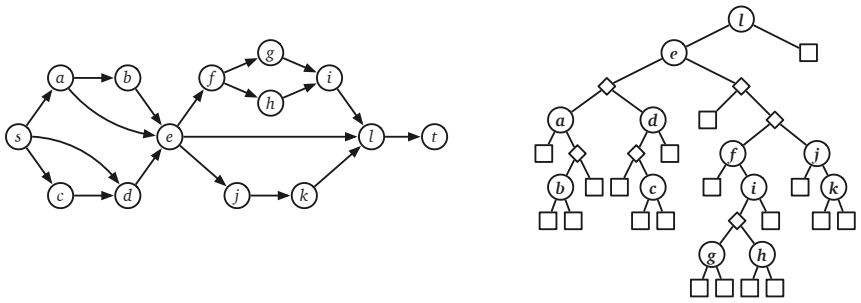
   (b) DECREMENT(e): Decrease the capacity of edge e by 1 and update the maximum flow.

   Both algorithms should modify f^* so that it is still a maximum flow, but more quickly than recomputing a maximum flow from scratch.

3. An **(s, t)-*series-parallel*** graph is a directed acyclic graph with two designated vertices s (the *source*) and t (the *target* or *sink*) and with one of the following structures:

   - **Base case:** A single directed edge from s to t.
   - **Series:** The union of an (s, u)-series-parallel graph and a (u, t)-series-parallel graph that share a common vertex u but no other vertices or edges.
   - **Parallel:** The union of two smaller (s, t)-series-parallel graphs with the same source s and target t, but with no other vertices or edges in common.

   Every (s, t)-series-parallel graph G can be represented by a ***decomposition tree***, which is a binary tree with three types of nodes: leaves corresponding to single edges in G, series nodes (each labeled by some vertex), and parallel nodes (unlabeled).



An series-parallel graph and its decomposition tree.

   (a) Suppose you are given a directed graph G with two special vertices s and t. Describe and analyze an algorithm that either builds a decomposition tree for G or correctly reports that G is not (s, t)-series-parallel. *[Hint: Build the tree from the bottom up.]*

   (b) Describe and analyze an algorithm to compute a maximum (s, t)-flow in a given (s, t)-series-parallel flow network with arbitrary edge capacities. *[Hint: In light of part (a), you can assume that you are actually given the decomposition tree.]*

CS 473 ✧ Spring 2017

Homework 7

Due Wednesday, March 29, 2017 at 8pm

---

1. 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:

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

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

   (b) Prove that a legal rounding is possible *if and only if* the sum of entries in each row is an integer, and the sum of entries in each column is an integer. In other words, prove that either your algorithm from part (a) returns a legal rounding, or a legal rounding is *obviously* impossible.

2. Quentin, Alice, and the other Brakebills Physical Kids are planning an excursion through the Neitherlands to Fillory. The Neitherlands is a vast, deserted city composed of several plazas, each containing a single fountain that can magically transport people to a different world. Adjacent plazas are connected by gates, which have been cursed by the Beast. The gates between plazas are open only for five minutes every hour, all simultaneously—from 12:00 to 12:05, then from 1:00 to 1:05, and so on—and are otherwise locked. During those five minutes, if more than one person passes through any single gate, the Beast will detect their presence.[1] Moreover, anyone attempting to open a locked gate, or attempting to pass through more than one gate within the same five-minute period will turn into a niffin.[2] However, any number of people can safely pass through *different* gates at the same time and/or pass through the same gate at *different* times.

   You are given a map of the Neitherlands, which is a graph G with a vertex for each fountain and an edge for each gate, with the fountains to Earth and Fillory clearly marked. Suppose you are also given a positive integer h. Describe and analyze an algorithm to compute the maximum number of people that can walk from the Earth fountain to the Fillory fountain in at most h hours—that is, after the gates have opened at most h times—without anyone alerting the Beast or turning into a niffin. *[Hint: Build a different graph.]*

---

[1] This is very bad.

[2] This is very very bad.

Rubric (graph reductions): For a problem worth 10 points, solved by reduction to maximum flow:

- 2 points for a complete description of the relevant flow network, specifying the set of vertices, the set of edges (being careful about direction), the source and target vertices s and t, and the capacity of every edge. (If the flow network is part of the original input, just say that.)
- 1 point for a description of the algorithm to construct this flow network from the stated input. This could be as simple as "We can construct the flow network in $O(n^3)$ time by brute force."
- 1 point for precisely specifying the problem to be solved on the flow network (for example: "maximum flow from x to y") and the algorithm (For example: "Ford-Fulkerson" or "Orlin") to solve that problem. Do *not* regurgitate the details of the maximum-flow algorithm itself.
- 1 point for a description of the algorithm to extract the answer to the stated problem from the maximum flow. This could be as simple as "Return TRUE if the maximum flow value is at least 42 and FALSE otherwise."
- **4 points for a proof that your reduction is correct.** This proof will almost always have two components (worth 2 points each). For example, if your algorithm returns a boolean, you should prove that its True answers are correct and that its False answers are correct. If your algorithm returns a number, you should prove that number is neither too large nor too small.
- 1 point for the running time of the overall algorithm, expressed as a function of the original input parameters, *not* just the number of vertices and edges in your flow network. You may assume that maximum flows can be computed in $O(VE)$ time.

Reductions to other flow-based problems described in class or in the notes (for example: edge-disjoint paths, maximum bipartite matching, minimum-cost circulation) or to other standard graph problems (for example: reachability, topological sort, minimum spanning tree, all-pairs shortest paths) have similar requirements.

CS 473 ✧ Spring 2017

Homework 8

Due Wednesday, April 12, 2017 at 8pm

---

1. Recall that a ***path cover*** of a directed acyclic graph is a collection of directed paths, such that every vertex in G appears in at least one path. We previously saw how to compute *disjoint* path covers (where each vertex lies on *exactly* one path) by reduction to maximum bipartite matching. Your task in this problem is to compute path covers *without* the disjointness constraint.

   (a) Suppose you are given a dag G with a unique source s and a unique sink t. Describe an algorithm to find the smallest path cover of G in which every path starts at s and ends at t.

   (b) Describe an algorithm to find the smallest path cover of an arbitrary dag G, with no additional restrictions on the paths. *[Hint: Use part (a).]*

2. Recall that an ***(s, t)-series-parallel*** graph is an directed acyclic graph with two designated vertices s (the *source*) and t (the *target* or *sink*) and with one of the following structures:

   - **Base case:** A single directed edge from s to t.
   - **Series:** The union of an (s,u)-series-parallel graph and a (u,t)-series-parallel graph that share a common vertex u but no other vertices or edges.
   - **Parallel:** The union of two smaller (s,t)-series-parallel graphs with the same source s and target t, but with no other vertices or edges in common.

   Any series-parallel graph can be represented by a binary *decomposition tree*, whose interior nodes correspond to series compositions and parallel compositions, and whose leaves correspond to individual edges. In a previous homework, we saw how to construct the decomposition tree for any series-parallel graph in $O(V+E)$ time, and then how to compute a maximum (s,t)-flow in $O(V+E)$ time.

   Describe an efficient algorithm to compute a *minimum-cost* maximum flow from s to t in an (s,t)-series-parallel graph G in which every edge has capacity 1 and arbitrary cost. *[Hint: First consider the special case where G has only two vertices but lots of edges.]*

3. 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, (3) no instructor is assigned to more than three committees, (2) 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.

CS 473 ✧ Spring 2017

Homework 9

Due Wednesday, April 19, 2017 at 8pm

---

1. Suppose you are given an arbitrary directed graph $G = (V, E)$ with arbitrary edge weights $\ell : E \to \mathbb{R}$. Each edge in G is colored either red, white, or blue to indicate how you are permitted to modify its weight:

   - You may increase, but not decrease, the length of any red edge.
   - You may decrease, but not increase, the length of any blue edge.
   - You may not change the length of any black edge.

   The ***cycle nullification*** problem asks whether it is possible to modify the edge weights—subject to these color constraints—so that *every cycle in G has length* 0. Both the given weights and the new weights of the individual edges can be positive, negative, or zero. To keep the following problems simple, assume that G is strongly connected.

   (a) Describe a linear program that is feasible if and only if it is possible to make every cycle in G have length 0. *[Hint: Pick an arbitrary vertex s, and let* $\text{dist}(v)$ *denote the length of every walk from s to v.]*

   (b) Construct the dual of the linear program from part (a). *[Hint: Choose a convenient objective function for your primal LP.]*

   (c) Give a self-contained description of the combinatorial problem encoded by the dual linear program from part (b), and prove *directly* that it is equivalent to the original cycle nullification problem. Do not use the words "linear", "program", or "dual". Yes, you have seen this problem before.

   (d) Describe and analyze an algorithm to determine *in $O(EV)$ time* whether it is possible to make every cycle in G have length 0, using your dual formulation from part (c). Do not use the words "linear", "program", or "dual".

2. *There is no problem 2.*

CS 473 ✧ Spring 2017

Homework 10

Due Wednesday, April 26, 2017 at 8pm

---

1. An ***integer linear program*** is a linear program with the additional explicit constraint that the variables must take *only* integer values. The ILP-Feasibility problem asks whether there is an integer vector that satisfies a given system of linear inequalities—or more concisely, whether a given integer linear program is feasible.

   Describe a polynomial-time reduction from 3Sat to ILP-Feasibility. Your reduction implies that ILP-Feasibility is NP-hard.

2. There are two different versions of the Hamiltonian cycle problem, one for directed graphs and one for undirected graphs. We saw a proof in class (and there are two proofs in the notes) that the *directed* Hamiltonian cycle problem is NP-hard.

   (a) Describe a polynomial-time reduction from the *undirected* Hamiltonian cycle problem to the *directed* Hamiltonian cycle problem. Prove your reduction is correct.

   (b) Describe a polynomial-time reduction from the *directed* Hamiltonian cycle problem to the *undirected* Hamiltonian cycle problem. Prove your reduction is correct.

   (c) Which of these two reductions implies that the *undirected* Hamiltonian cycle problem is NP-hard?

3. Recall that a 3CNF formula is a conjunction (And) of several distinct clauses, each of which is a disjunction (Or) of exactly three distinct literals, where each literal is either a variable or its negation.

   Suppose you are given a magic black box that can determine **in polynomial time**, whether an arbitrary given 3CNF formula is satisfiable. Describe and analyze a **polynomial-time** algorithm that either computes a satisfying assignment for a given 3CNF formula or correctly reports that no such assignment exists, using the magic black box as a subroutine. *[Hint: Call the magic black box more than once. First imagine an even more magical black box that can decide Sat for arbitrary boolean formulas, not just 3CNF formulas.]*

Rubric (for all polynomial-time reductions): 10 points =

- \+ 3 points for the reduction itself
  - – For an NP-hardness proof, the reduction must be from a known NP-hard problem. You can use any of the NP-hard problems listed in the lecture notes (except the one you are trying to prove NP-hard, of course).
- \+ 3 points for the "if" proof of correctness
- \+ 3 points for the "only if" proof of correctness
- \+ 1 point for writing "polynomial time"

- An incorrect polynomial-time reduction that still satisfies half of the correctness proof is worth at most 4/10.
- A reduction in the wrong direction is worth 0/10.

CS 473 ✧ Spring 2017

Homework 11

"Due" Wednesday, May 3, 2017 at 8pm

This homework will not be graded.
However, material covered by this homework *may* appear on the final exam.

---

1. Let Φ be a boolean formula in conjunctive normal form, with exactly three literals in each clause. Recall that an assignment of boolean values to the variables in Φ ***satisfies*** a clause if at least one of its literals is TRUE. The ***maximum satisfiability problem*** for 3CNF formulas, usually called MAX3SAT, asks for the maximum number of clauses that can be simultaneously satisfied by a single assignment.

   Solving MAX3SAT exactly is clearly also NP-hard; this question asks about approximation algorithms. Let $\mathit{Max3Sat}(\Phi)$ denote the maximum number of clauses in Φ that can be simultaneously satisfied by one variable assignment.

   (a) Suppose we assign variables in Φ to be TRUE or FALSE using independent fair coin flips. Prove that the expected number of satisfied clauses is at least $\frac{7}{8}\mathit{Max3Sat}(\Phi)$.

   (b) Let k^+ denote the number of clauses satisfied by setting every variable in Φ to TRUE, and let k^- denote the number of clauses satisfied by setting every variable in Φ to FALSE. Prove that $\max\{k^+, k^-\} \geq \mathit{Max3Sat}(\Phi)/2$.

   (c) Let $\mathit{Min3Unsat}(\Phi)$ denote the *minimum* number of clauses that can be simultaneously left *unsatisfied* by a single assignment. Prove that it is NP-hard to approximate $\mathit{Min3Unsat}(\Phi)$ within a factor of $10^{10^{100}}$.

2. Consider the following algorithm for approximating the minimum vertex cover of a connected graph G: ***Return the set of all non-leaf nodes of an arbitrary depth-first spanning tree***. (Recall that a depth-first spanning tree is a rooted tree; the root is not considered a leaf, even if it has only one neighbor in the tree.)

   (a) Prove that this algorithm returns a vertex cover of G.

   (b) Prove that this algorithm returns a 2-approximation to the smallest vertex cover of G.

   (c) Describe an infinite family of connected graphs for which this algorithm returns a vertex cover of size *exactly* $2 \cdot \text{OPT}$. This family implies that the analysis in part (b) is tight. *[Hint: First find just **one** such graph, with few vertices.]*

3. Consider the following modification of the "dumb" 2-approximation algorithm for minimum vertex cover that we saw in class. The only change is that we return a set of edges instead of a set of vertices.

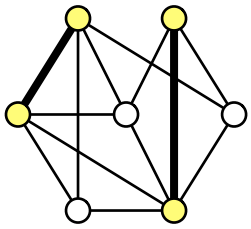
```
ApproxMinMaxMatching(G):
  M ← ∅
  while G has at least one edge
      uv ← any edge in G
      G ← G \ {u, v}
      M ← M ∪ {uv}
  return M
```

   (a) Prove that the output subgraph M is a *matching*—no pair of edges in M share a common vertex.

   (b) Prove that M is a *maximal* matching—M is not a proper subgraph of another matching in G.

   (c) Prove that M contains at most twice as many edges as the *smallest* maximal matching in G.

   (d) Describe an infinite family of graphs G such that the matching returned by ApproxMinMaxMatching(G) contains exactly twice as many edges as the smallest maximum matching in G. This family implies that the analysis in part (c) is tight. *[Hint: First find just **one** such graph, with few vertices.]*



The smallest maximal matching in a graph.

CS 473 ✧ Spring 2017

Midterm 1

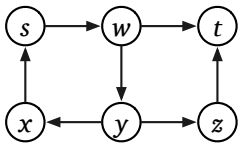
February 21, 2016

---

1. Recall that a ***walk*** in a directed graph G is an arbitrary sequence of vertices $v_0 \to v_1 \to \cdots \to v_k$, such that $v_{i-1} \to v_i$ is an edge in G for every index i. A ***path*** is a walk in which no vertex appears more than once.

   Suppose you are given a directed graph $G = (V, E)$ and two vertices s and t. Describe and analyze an algorithm to determine if there is a walk in G from s to t whose length is a multiple of 3.

   For example, given the graph shown below, with the indicated vertices s and t, your algorithm should return TRUE, because the walk $s \to w \to y \to x \to s \to w \to t$ has length 6.



   [Hint: Build a (different) graph.]

2. A ***shuffle*** of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. A ***smooth*** shuffle of X and Y is a shuffle of X and Y that never uses more than two consecutive symbols of either string. For example,

   - `prDoYgNAraMmmIiCng` is a smooth shuffle of `DYNAMIC` and `programming`.
   - `DYprNogrAaMmmICing` is a shuffle of `DYNAMIC` and `programming`, but it is not a smooth shuffle (because of the substrings `ogr` and `ing`).

   Describe and analyze an algorithm to decide, given three strings X, Y, and Z, whether Z is a smooth shuffle of X and Y.

3. (a) Describe an algorithm that simulates a fair coin, using independent rolls of a fair three-sided die as your only source of randomness. Your algorithm should return either HEADS or TAILS, each with probability $1/2$.

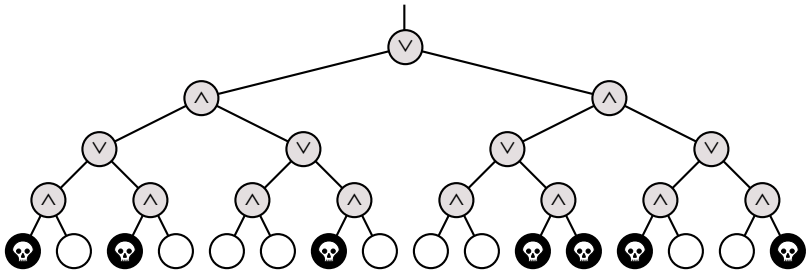
   (b) What is the expected number of die rolls performed by your algorithm in part (a)?

   (c) Describe an algorithm that simulates a fair three-sided die, using independent fair coin flips as your only source of randomness. Your algorithm should return either 1, 2, or 3, each with probability $1/3$.

   (d) What is the expected number of coin flips performed by your algorithm in part (c)?

4. Death knocks on Dirk Gently's door one cold blustery morning and challenges him to a game. Emboldened by his experience with algorithms students, Death presents Dirk 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, Dirk 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, Dirk dies; if it's white, Dirk lives forever. Dirk moves first, so Death gets the last turn.

   (Yes, this is precisely the same game from Homework 3.)



Unfortunately, Dirk slept through Death's explanation of the rules, so he decides to just play randomly. Whenever it's Dirk's turn, he flips a fair coin and moves left on heads, or right on tails, confident that the Fundamental Interconnectedness of All Things will keep him alive, unless it doesn't. Death plays much more purposefully, of course, always choosing the move that maximizes the probability that Dirk loses the game.

(a) Describe an algorithm that computes *the probability* that Dirk wins the game against Death.

(b) Realizing that Dirk is not taking the game seriously, Death gives up in desperation and decides to also play randomly! Describe an algorithm that computes *the probability* that Dirk wins the game again Death, assuming *both* players flip fair coins to decide their moves.

For both algorithms, the input consists of the integer n (specifying the depth of the tree) and an array of 4^n bits specifying the colors of leaves in left-to-right order.

CS 473 ✧ Spring 2017

Midterm 2

April 4, 2016

---

1. Let $G = (V, E)$ be an arbitrary undirected graph. Suppose we color each vertex of G uniformly and independently at random from a set of three colors: red, green, or blue. An edge of G is *monochromatic* if both of its endpoints have the same color.

   (a) What is the *exact* expected number of monochromatic edges? (Your answer should be a simple function of V and E.)

   (b) For each edge $e \in E$, define an indicator variable X_e that equals 1 if e is monochromatic and 0 otherwise. ***Prove*** that

   $$\Pr[(X_a = 1) \wedge (X_b = 1)] \;=\; \Pr[X_a = 1] \cdot \Pr[X_b = 1]$$

   for every pair of edges $a \neq b$. This claim implies that the random variables X_e are pairwise independent.

   (c) ***Prove*** that there is a graph G such that

   $$\Pr[(X_a = 1) \wedge (X_b = 1) \wedge (X_c = 1)] \;\neq\; \Pr[X_a = 1] \cdot \Pr[X_b = 1] \cdot \Pr[X_c = 1]$$

   for some triple of distinct edges a, b, c in G. This claim implies that the random variables X_e are *not necessarily* 3-wise independent.

2. The White Rabbit has a very poor memory, and so he is constantly forgetting his regularly scheduled appointments with the Queen of Hearts. In an effort to avoid further beheadings of court officials, The King of Hearts has installed an app on Rabbit's pocket watch to automatically remind Rabbit of any upcoming appointments. For each reminder Rabbit receives, Rabbit has a 50% chance of actually remembering his appointment (decided by an independent fair coin flip).

   First, suppose the King of Hearts sends Rabbit k separate reminders for a *single* appointment.

   (a) What is the *exact* probability that Rabbit will remember his appointment? Your answer should be a simple function of k.

   (b) What value of k should the King choose so that the probability that Rabbit will remember this appointment is at least $1 - 1/n^\alpha$? Your answer should be a simple function of n and α.

   Now suppose the King of Hearts sends Rabbit k separate reminders for each of n different appointments. (That's nk reminders altogether.)

   (c) What is the *exact* expected number of appointments that Rabbit will remember? Your answer should be a simple function of n and k.

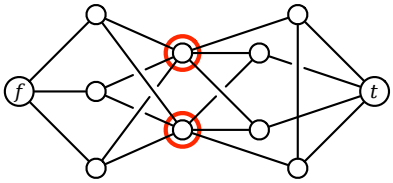
   (d) What value of k should the King choose so that the probability that Rabbit remembers *every* appointment is at least $1 - 1/n^\alpha$? Again, your answer should be a simple function of n and α.

   [Hint: There is a simple solution that does not use tail inequalities.]

3. The Island of Sodor is home to an extensive rail network. Recently, several cases of a deadly contagious disease (either swine flu or zombies; reports are unclear) have been reported in the village of Ffarquhar. The controller of the Sodor railway plans to close certain railway stations to prevent the disease from spreading to Tidmouth, his home town. No trains can pass through a closed station. To minimize expense (and public notice), he wants to close as few stations as possible. However, he cannot close the Ffarquhar station, because that would expose him to the disease, and he cannot close the Tidmouth station, because then he couldn't visit his favorite pub.

   Describe and analyze an algorithm to find the minimum number of stations that must be closed to block all rail travel from Ffarquhar to Tidmouth. The Sodor rail network is represented by an undirected graph, with a vertex for each station and an edge for each rail connection between two stations. Two special vertices f and t represent the stations in Ffarquhar and Tidmouth.

   For example, given the following input graph, your algorithm should return the integer 2.



4. The Department of Commuter Silence at Shampoo-Banana University has a flexible curriculum with a complex set of graduation requirements. The department offers n different courses, and there are m different requirements. Each requirement specifies a subset of the n courses and the number of courses that must be taken from that subset. The subsets for different requirements may overlap, but each course can only be used to satisfy *at most one* requirement.

   For example, suppose there are $n = 5$ courses A, B, C, D, E and $m = 2$ graduation requirements:

   - You must take at least 2 courses from the subset $\{A, B, C\}$.
   - You must take at least 2 courses from the subset $\{C, D, E\}$.

   Then a student who has only taken courses B, C, D cannot graduate, but a student who has taken either A, B, C, D or B, C, D, E can graduate.

   Describe and analyze an algorithm to determine whether a given student can graduate. The input to your algorithm is the list of m requirements (each specifying a subset of the n courses and the number of courses that must be taken from that subset) and the list of courses the student has taken.

CS 473 ✧ Spring 2017

Final Exam

May 10, 2017

---

1. A ***three-dimensional matching*** in an undirected graph G is a collection of vertex-disjoint triangles. A three-dimensional matching is *maximal* if it is not a proper subgraph of a larger three-dimensional matching in the same graph.

   (a) Let M and M' be two arbitrary maximal three-dimensional matchings in the same underlying graph G. ***Prove*** that $|M| \leq 3 \cdot |M'|$.

   (b) Finding the *largest* three-dimensional matching in a given graph is NP-hard. Describe and analyze a fast 3-approximation algorithm for this problem.

   (c) Finding the *smallest maximal* three-dimensional matching in a given graph is NP-hard. Describe and analyze a fast 3-approximation algorithm for this problem.

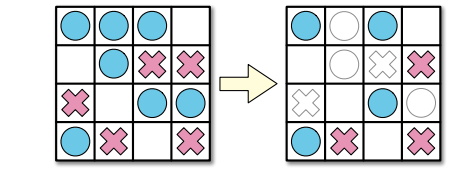
2. Let $G = (V, E)$ be an arbitrary dag with a unique source s and a unique sink t. Suppose we compute a random walk from s to t, where at each node v (except t), we choose an outgoing edge $v \to w$ uniformly at random to determine the successor of v.

   (a) Describe and analyze an algorithm to compute, for every vertex v, the probability that the random walk visits v.

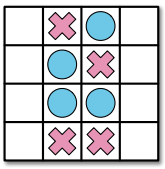
   (b) Describe and analyze an algorithm to compute the expected number of edges in the random walk.

   Assume all relevant arithmetic operations can be performed exactly in $O(1)$ time.

3. 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.

4. Suppose you are given a bipartite graph $G = (L \sqcup R, E)$ and a maximum matching M in G. Describe and analyze fast algorithms for the following problems:

   (a) INSERT(e): Insert a new edge e into G and update the maximum matching. (You can assume that e is not already an edge in G, and that $G + e$ is still bipartite.)

   (b) DELETE(e): Delete the existing edge e from G and update the maximum matching. (You can assume that e is in fact an edge in G.)

   Your algorithms should modify M so that it is still a maximum matching, faster than recomputing a maximum matching from scratch.

5. You are applying to participate in this year's Trial of the Pyx, the annual ceremony where samples of all British coinage are tested, to ensure that they conform as strictly as possible to legal standards. As a test of your qualifications, your interviewer at the Worshipful Company of Goldsmiths has given you a bag of n commemorative Alan Turing half-guinea coins, exactly two of which are counterfeit. One counterfeit coin is very slightly lighter than a genuine Turing; the other is very slightly heavier. Together, the two counterfeit coins have *exactly* the same weight as two genuine coins. Your task is to identify the two counterfeit coins.

   The weight difference between the real and fake coins is too small to be detected by anything other than the Royal Pyx Coin Balance. You can place any two disjoint sets of coins in each of the Balance's two pans; the Balance will then indicate which of the two subsets has larger total weight, or that the two subsets have the same total weight. Unfortunately, each use of the Balance requires completing a complicated authorization form (in triplicate), submitting a blood sample, and scheduling the Royal Bugle Corps, so you *really* want to use the Balance as few times as possible.

   (a) Suppose you *randomly* choose $n/2$ of your n coins to put on one pan of the Balance, and put the remaining $n/2$ coins on the other pan. What is the probability that the two subsets have equal weight?

   (b) Describe and analyze a randomized algorithm to identify the two fake coins. What is the expected number of times your algorithm uses the Balance? To simplify the algorithm, you may assume that n is a power of 2.

6. Suppose you are given a set L of n line segments in the plane, where each segment has one endpoint on the vertical line $x = 0$ and one endpoint on the vertical line $x = 1$, and all $2n$ endpoints are distinct. Describe and analyze an algorithm to compute the largest subset of L in which no pair of segments intersects.

Some Useful Inequalities

Let $X = \sum_{i=1}^{n} X_i$, where each X_i is a 0/1 random variable, and let $\mu = \mathrm{E}[X]$.

- **Markov's Inequality:** $\Pr[X \geq x] \leq \mu/x$ for all $x > 0$.
- **Chebyshev's Inequality:** If $X_1, X_2, \ldots, X_n$ are pairwise independent, then for all $\delta > 0$:

$$\Pr[X \geq (1+\delta)\mu] < \frac{1}{\delta^2 \mu} \quad \text{and} \quad \Pr[X \leq (1-\delta)\mu] < \frac{1}{\delta^2 \mu}$$

- **Chernoff Bounds:** If $X_1, X_2, \ldots, X_n$ are fully independent, then for all $0 < \delta \leq 1$:

$$\Pr[X \geq (1+\delta)\mu] \leq \exp\left(-\delta^2 \mu/3\right) \quad \text{and} \quad \Pr[X \leq (1-\delta)\mu] \leq \exp\left(-\delta^2 \mu/2\right)$$

Some Useful Algorithms

- **Random(k):** Returns an element of $\{1, 2, \ldots, k\}$, chosen independently and uniformly at random, in $O(1)$ time. For example, Random(2) can be used for a fair coin flip.
- **Ford and Fulkerson's maximum flow algorithm:** Returns a maximum (s,t)-flow f^* in a given flow network in $\boldsymbol{O(E \cdot |f^*|)}$ ***time***. If all input capacities are integers, then all output flow values are also integers.
- **Orlin's maximum flow algorithm:** Returns a maximum (s,t)-flow in a given flow network in $\boldsymbol{O(VE)}$ ***time***. If all input capacities are integers, then all output flow values are also integers.
- **Orlin's minimum-cost flow algorithm:** Returns a minimum-cost flow in a given flow network in $\boldsymbol{O(E^2 \log^2 V)}$ ***time***. If all input capacities, costs, and balances are integers, then all output flow values are also integers.

Some Useful NP-hard Problems:

3Sat: Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying 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 $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MinHittingSet: Given a collection of subsets $S_1, S_2, \ldots, S_m$ 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?

HamiltonianCycle: Given a graph G (either directed or undirected), is there a cycle in G that visits every vertex exactly once?

FeasibleILP: Given a matrix $A \in \mathbb{Z}^{n \times d}$ and a vector $b \in \mathbb{Z}^n$, determine whether the set of feasible integer points $\max\{x \in \mathbb{Z}^d \mid Ax \leq b, x \geq 0\}$ is empty.

HydraulicPress: And here ve go!

Common Grading Rubrics

(For problems out of 10 points)

General Principles:

- Faster algorithms are worth more points, and slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of n. Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.
- A clear, correct, and correctly analyzed algorithm, no matter how slow, is always worth more than "I don't know". An incorrect algorithm, no matter how fast, may be worth nothing.
- Proofs of correctness are required on exams if and only if we explicitly ask for them.

Dynamic Programming:

- 3 points for a **clear English specification** of the underlying recursive function = 2 for describing the function itself + 1 for describing how to call the function to get your final answer. We want an English description of the underlying recursive *problem*, not just the algorithm/recurrence. In particular, your description should specify precisely the role of each input parameter. **No credit for the rest of the problem if the English description is is missing; this is a Deadly Sin.**
- 4 points for correct recurrence = 1 for base case(s) + 3 for recursive case(s). **No credit for iterative details if the recursive case(s) are incorrect.**
- 3 points for iterative details = 1 for memoization structure + 1 for evaluation order + 1 for time analysis. Complete iterative pseudocode is *not* required for full credit.

Graph Reductions:

- 4 points for a complete description of the relevant graph, including vertices, edges (including whether directed or undirected), numerical data (weights, lengths, capacities, costs, balances, and the like), source and target vertices, and so on. If the graph is part of the original input, just say that.
- 4 points for other details of the reduction, including how to build the graph from the original input, the precise problem to be solved on the graph, the precise algorithm used to solve that problem, and how to extract your final answer from the output of that algorithm.
- 2 points for running time of the overall algorithm, expressed as a function of the original input parameters, *not* just the number of vertices and edges in the graph.

NP-hardness Proofs:

- 3 points for a complete description of the reduction, including an appropriate NP-hard problem to reduce from, how to transform the input, and how to transform the output.
- 6 points for the proof of correctness = 3 for the "if" part + 3 for the "only if" part.
- 1 points for "polynomial time".