CS 473 ✧ Fall 2022

Homework 0

Due Tuesday, August 30, 2022 at 9pm

- **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 PDF file for each numbered problem.
  - You can find a LaTeX solution template on the course web site; please use it if you plan to typeset your homework.
  - If you plan to submit scanned handwritten solutions, please use dark ink (not pencil) on blank white printer paper (not notebook or graph paper), and use a high-quality scanner or scanning app to create a high-quality PDF for submission (not a raw cell-phone photo). We reserve the right to reject submissions that are difficult to read.

☞ Some important course policies ☜

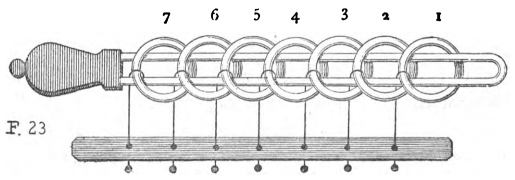
- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- **Avoid the Deadly Sins!** There are a few common writing (and thinking) practices that will be automatically penalized on every homework or exam problem. We're not just trying to be scary control freaks; history strongly suggests that people who commit these sins are more likely to make other serious mistakes as well. We're trying to break bad habits that seriously impede mastery of the course material.
  - Always give complete solutions, not just examples.
  - Every algorithm requires an English specification.
  - Never use weak induction. Weak induction should die in a fire.

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 online.

---

1. The Tower of Hanoi is a relatively recent descendant of a much older mechanical puzzle known as the Baguenaudier, Chinese rings, Cardano's rings, 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 around 1500CE;[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. Following tradition, we will index both the rings and the corresponding bits *from right to left*, as shown in the figure above. The puzzle allows two legal moves:

- Flip the rightmost bit.
- Flip the bit just to the left of the rightmost 1.

(The second move is impossible if the rightmost $n-1$ bits are all 0s.)

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

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

(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 each non-negative integer n, there is *exactly one* reduced sequence of moves that solves the n-ring Baguenaudier puzzle.

[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.

2. Prince Hutterdink and Princess Bumpercup are celebrating their recent marriage by inviting all the dukes and duchesses in the kingdom to sample the castle's celebrated wine cellars. Just before they drinking begins, the happy couple learns that exactly one of their n bottles of wine has been laced with iocaine powder, one of the deadliest poisons known to man.

   Hutterdink and Bumpercup employ several Royal Tasters who have built up an immunity to iocaine powder. (Strangely, every Royal Taster is named Roberts.) If a Taster consumes any amount of iocaine, no matter how small, they quickly become extremely ill, or as the Royal Miracle Workers optimistically put it, only *mostly* dead. Anyone else who consumes iocaine quickly becomes *all* dead.

   To test a set S of wine bottles, a Taster mixes one drop of wine from each bottle in S and consumes the resulting mixture. They will become mostly dead if and only if one of the bottles in S is poisoned. Each Taster must be paid 1000 guilders for each test they perform, so Hutterdink and Bumpercup want to use as few tests as possible. On the other hand, mostly dead Tasters require months to recover, and the party is tomorrow!

   (a) Suppose there is an unlimited supply of Tasters. Describe an algorithm to find the poisoned bottle using at most $O(\log n)$ tests. (This is best possible in the worst case.)
   (b) Now suppose there is only one Taster. Argue that $\Omega(n)$ tests are required in the worst case to find the poisoned bottle.
   (c) Now suppose there are two Tasters. Describe an algorithm that allows them to find the poisoned bottle using only $O(\sqrt{n})$ tests.
   (d) Finally, describe an algorithm to identify the poisoned bottle when there are k tasters. Report the number of tests that your algorithm uses as a function of both n and k.

3. At the start of the semester, the faculty and staff the See-Bull Center for Skeptical Media Consumption throw a welcome party for new and returning students. Every person who attended the party was given at least one rubber duck to keep in their office as a memento.

   When new graduate student Mariadne Inotaur arrives at the See-Bull Center the next day and asks for her rubber duck, she is told that all the rubber ducks are gone. Infuriated, Mariadne decides to return that night and steal as many ducks as she can.

   Some doors inside See-Bull are locked; each locked door requires a key card to open. There are six types of key cards, corresponding to different research groups that work in the building—Applesauce, Blarney, Claptrap, Drivel, Eyewash, and Flapdoodle. Each locked door can be unlocked by exactly one type of key card. Mariadne doesn't initially have any key cards, so if she wants to open a Flapdoodle door, for example, she must first find a Flapdoodle key card.

   Mariadne has a complete map of See-Bull, modeled as a sinple undirected graph $G = (V, E)$. The vertices V correspond to interior spaces (rooms, hallways, stairwells, and so on), plus one special vertex s for the outside world. Each vertex is labeled with the number of rubber ducks and the types of key cards (if any) in the corresponding space; there are no rubber ducks or key cards outside. The edges E correspond to doors between spaces. Each edge is labeled with a bit indicating whether the corresponding door is locked, and if so, the type of key card that unlocks it.

   Describe and analyze an algorithm to determine the maximum number of rubber ducks that Mariadne can steal, assuming she starts outside, with no rubber ducks or key cards.

CS 473 ✧ Fall 2022
Homework 1

Due Tuesday, September 6, 2022 at 9pm

- Starting with this homework, groups of up to three students can submit joint solutions for each problem. For each numbered problem, exactly one member of each homework group should submit a solution and identify the other group members (if any) on Gradescope. Please also list all group members at the top of the first page of every submission.

---

1. Suppose we are given a bit string $B[1..n]$. A triple of indices $1 \le i < j < k \le n$ is called a ***well-spaced triple*** if $B[i] = B[j] = B[k] = 1$ and $k - j = j - i$.

   (a) Describe a brute-force algorithm to determine whether B has a well-spaced triple in $O(n^2)$ time.

   (b) Describe an algorithm to determine whether B has a well-spaced triple in $O(n \log n)$ time. *[Hint: FFT!]*

   (c) Describe an algorithm to determine the *number* of well-spaced triples in B in $O(n \log n)$ time.

2. This problem explores different algorithms for computing the factorial function $n! = 1 \cdot 2 \cdot 3 \cdots n$. This may seem like a weird question; the obvious algorithm uses n multiplications, and thus runs in $O(n)$ time. Right?

   Well, actually, no. The inequalities $(n/2)^{n/2} < n! < n^n$ imply that the exact binary representation of $n!$ has length $\Theta(n \log n)$. So the *number* of multiplications is not a good estimate of the actual time required to compute $n!$; we also need to account for the *time* for those multiplications.

   (a) Recall that the standard lattice algorithm that you learned in elementary school multiplies any n-bit integer and any m-bit integer in $O(mn)$ time. Describe and analyze a variant of Karatsuba's algorithm that multiplies any n-bit integer and any m-bit integer, for any $n \ge m$, in $O(n \cdot m^{\lg 3 - 1}) = O(n \cdot m^{0.58496})$ time.

   (b) Consider the following classical algorithm for computing the factorial $n!$ of a non-negative integer n:

   ```
   FACTORIAL(n):
     fact ← 1
     for i ← 1 to n
         fact ← fact · i        (∗)
     return fact
   ```

   Analyze the running time of FACTORIAL(n) using different algorithms for the multiplication in line $(*)$:

   i. Lattice multiplication
   ii. Your variant of Karatsuba's algorithm from part (a)

(c) The following divide-and-conquer algorithm also computes the factorial function, but using a different grouping of the multiplications. The subroutine FALLING computes the *falling power* function $n^{\underline{m}} = n(n-1)(n-2)\cdots(n-m+1) = n!/(n-m)! = \binom{n}{m}\cdot m!$:

```
FALLING($n, m$):
  if $m = 0$
      return 1
  else if $m = 1$
      return $n$
  else
      return FALLING($n, \lfloor m/2 \rfloor$) · FALLING($n - \lfloor m/2 \rfloor, \lceil m/2 \rceil$)
```

```
FASTERFACTORIAL($n$):
  return FALLING($n, n$)
```

Analyze the running time of FASTERFACTORIAL(n) using different algorithms for the multiplication in the last line of FALLING:

i. Lattice multiplication

ii. Your variant of Karatsuba's algorithm from part (a)

(For simplicity, assume n is a power of 2 and ignore the floors and ceilings.)

3. Your new boss at the Dixon Ticonderoga Pencil Factory asks you to design an algorithm to solve the following problem. Suppose you are given N pencils, each with one of c different colors, and a non-negative integer k. ***How many different ways are there to choose a set of k pencils?*** Two pencil sets are considered identical if they contain the same number of pencils of each color.

For example, suppose you have two red pencils, four green pencils, and one blue pencil. Then you can form exactly five different two-pencil sets (RR, RG, RB, GG, GB), exactly six different four-pencil sets (RRGG, RRGB, RGGG, RGGB, GGGG, GGGB), and exactly three different six-pencil sets (RRGGGG, RRGGGB, RGGGGB).

Describe an algorithm to solve this problem, and analyze its running time. Your input is an array *Pencils*$[1..c]$ and an integer k, where *Pencils*$[i]$ stores the number of pencils with color i. Your output is a single non-negative integer. For example, given the input *Pencils* $= [2, 4, 1]$ and $k = 2$, your algorithm should return the integer 5.

For full credit, report the running time of your algorithm as a function of the parameters N (the total number of pencils), c (the number of colors), and k (the size of the target pencil sets). Assume that $k \ll c \ll N$, but do not assume that any of these parameters is a constant. *Assume for this problem that all arithmetic operations take $O(1)$ time.*

Hint:

$$\underbrace{(1+x+x^2)}_{\text{2 red pencils}}\cdot\underbrace{(1+x+x^2+x^3+x^4)}_{\text{4 green pencils}}\cdot\underbrace{(1+x)}_{\text{1 blue pencil}} = 1+3x+\underbrace{5x^2}_{\text{5 2-pencil sets}}+6x^3+\underbrace{6x^4}_{\text{6 4-pencil sets}}+5x^5+\underbrace{3x^6}_{\text{3 6-pencil sets}}+1$$

CS 473 ✧ Fall 2022
Homework 2
Due Tuesday, September 13, 2022 at 9pm

1. (a) Recall that a *palindrome* is any string that is exactly the same as its reversal, like I or DEED or RACECAR or AMANAPLANACATACANALPANAMA. Describe and analyze an algorithm to find the length of the *longest subsequence* of a given string that is also a palindrome.

For example, given the string MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM as input, your algorithm should return 11, which is the length of the longest palindrome subsequence MHYMRORMYHM.

(b) Similarly, a *repeater* is any string whose first half and second half are identical, like MEME or MURMUR or HOTSHOTS or SHABUSHABU. Describe and analyze an algorithm to find the length of the *longest subsequence* of a given string that is also a repeater.

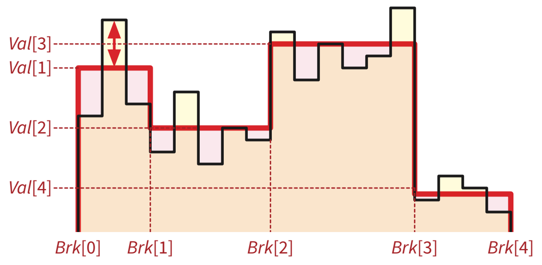
For example, given the string AINTNOPARTYLIKEMYNANASTEAPARTYHEYHO as input, your algorithm should return 20, which is the length of the longest repeater subsequence ANTPARTYEYANTPARTYEY.

2. Describe and analyze an efficient algorithm to solve the following one-dimensional clustering problem. Given an unsorted array $Data[1..n]$ of real numbers, we want to cluster this data into k clusters, each represented by an interval of indices and a real value, so that the maximum error between any data point and its cluster value is minimized.

More concretely, your algorithm should return an unsorted array $Val[1..k]$ of real numbers and a sorted array $Brk[0..k]$ of integer *breakpoints* such that $Brk[0] = 0$ and $Brk[k] = n$. For each index i, the ith interval covers items $Brk[i-1]+1$ through $Brk[i]$ in the input data, and the value of the ith interval is $Val[i]$. The output values $Val[i]$ are not necessarily elements of the input array. Your algorithm should compute arrays B and V that minimize the maximum absolute difference between any data point and the value of the unique interval that covers it:

$$Error(Data, Brk, Val) = \max\left\{ |Data[i] - Val[j]| \;\middle|\; Brk[j-1] < i \le Brk[j] \right\}$$

We can visualize both the input data and the output approximation using bar charts, as shown in the figure below; the double arrow shows the error for this approximation.



Approximating 18 data points with four weighted intervals

3. You've been hired to store a sequence of n books on shelves in a library, using as little *vertical* space as possible. The *horizontal* order of the books is fixed by the cataloging system and cannot be changed; each shelf must store a contiguous interval of the given sequence of books. You can adjust the height of each shelf to match the tallest book on that shelf; in particular, you can change the height of any empty shelf to zero.

   You are given two arrays $H[1 .. n]$ and $W[1 .. n]$, where $H[i]$ and $W[i]$ are respectively the height and width of the ith book. Each shelf has the same fixed length L. Each book as width at most L, so every book fits on a shelf. The total width of all books on each shelf cannot exceed L. Your task is to shelve the books so that the *sum of the heights* of the shelves is as small as possible.

   (a) There is a natural greedy algorithm, which actually yields an optimal solution when all books have the same height: If $n > 0$, pack as many books as possible onto the first shelf, and then recursively shelve the remaining books.

      Show that this greedy algorithm does *not* yield an optimal solution if the books can have different heights. *[Hint: There is a small counterexample.]*

   (b) Describe and analyze an efficient algorithm to assign books to shelves to minimize the total height of the shelves.

Standard dynamic programming rubric. 10 points =

- 3 points for a clear and correct English description of the recursive function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.)
  - −1 for naming the function "OPT" or "DP" or any single letter.
  - No credit if the description is inconsistent with the recurrence.
  - No credit if the description does not explicitly describe how the function value depends on the named input parameters.
  - No credit if the description refers to internal states of the eventual dynamic programming algorithm, like "the current index" or "the best score so far". The function must have a well-defined value that depends *only* on its input parameters (and constant global variables).
  - An English explanation of the *recurrence* or *algorithm* does not qualify. We want a description of *what* your function returns, not (here) an explanation of *how* that value is computed.
- 4 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - +1 for base case(s). −½ for one *minor* bug, like a typo or an off-by-one error.
  - +3 for recursive case(s). −1 for each *minor* bug, like a typo or an off-by-one error.
  - −2 for greedy optimizations without proof, even if they are correct.
  - **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 3 points for iterative details
  - +1 for describing (or sketching) an appropriate memoization data structure
  - +1 for describing (or sketching) a correct evaluation order; a clear picture is usually sufficient. If you use nested for loops, be sure to specify the nesting order.
  - +1 for correct 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 specifically says otherwise.
- ***Iterative pseudocode 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. However, you ***do*** still need and English description of the underlying recursive function (or equivalently, the contents of the memoization structure). ***Perfectly correct iterative pseudocode, with no explanation or time analysis, is worth at most 6 points out of 10.***
- Partial credit for incomplete solutions depends on the running time of the ***best possible*** completion (up to the target running time). For example, consider a solution that contains *only* a clear English description of a function, with no recurrence or iterative details.
  - If the described function *can* be developed into an algorithm with the target running time, the solution is worth 3 points.
  - If the described function leads to an algorithm that is slower than the target time by a factor of n, the solution could be worth only 2 points (= 70% of 3, rounded).
  - If the described function cannot lead to a polynomial-time algorithm, it could be worth 1 or even 0 points.

CS 473 ✧ Fall 2022
Homework 3
Due Tuesday, September 20, 2022 at 9pm

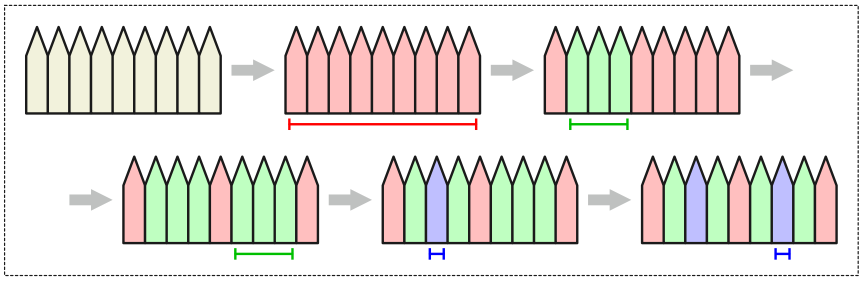
---

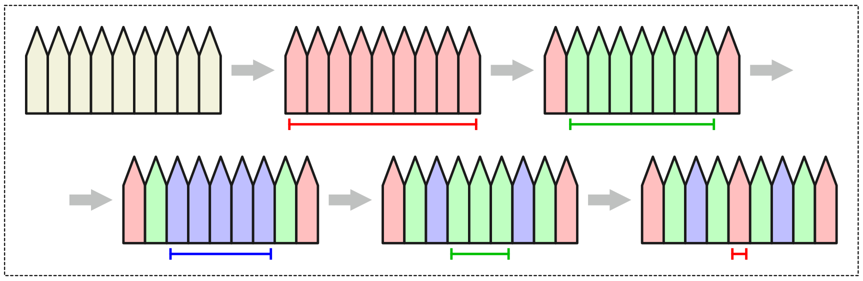
1. Huckleberry Sawyer needs to get his fence painted. His fence consists of a row of n wooden slats, all initially unpainted. He has been given a target color for each slat.

   Huck refuses to do any painting himself. Instead, each day he can hire one of his friends to paint any *contiguous* subset of slats a single color, for the uncomfortably high price of one nickel. Huck has really good paint, so he doesn't care if the same slat gets painted multiple times, as long as the last coat of paint on each slat matches its target color.

   Describe and analyze an algorithm that determines, given an array of target colors $C[1..n]$ as input, the minimum number of nickels that Huck must spend to have his fence completely painted.

   For example, if $n = 9$ and Huck's target color sequence is red, green, blue, green, red, green, blue, green, red, then your algorithm should return the integer 5. There are at least two different ways that Huck can get his fence painted by spending only five nickels:

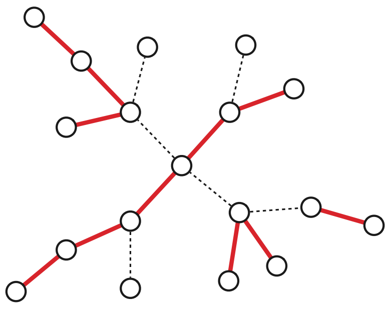




   [Hint: You need to prove that there is always an optimal plan with a certain structure.]

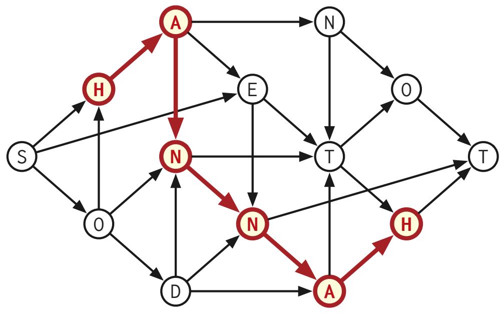
2. Let T be an arbitrary tree—a connected undirected graph with no cycles—each of whose edges has some positive weight. Describe and analyze an algorithm to cover the vertices of T with disjoint paths whose total length is as large as possible. (As usual, the length of a path is the sum of the weights of its edges.) Each vertex of T must lie on exactly one of the paths.

   The following figure shows a tree covered by seven disjoint paths, three of which have length zero.



3. 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 below, your algorithm should return the integer 6, which is the length of the palindrome HANNAH.



Standard dynamic programming rubric. 10 points =

- 3 points for a clear and correct English description of the recursive function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.)
  - − 1 for naming the function "OPT" or "DP" or any single letter.
  - No credit if the description is inconsistent with the recurrence.
  - No credit if the description does not explicitly describe how the function value depends on the named input parameters.
  - No credit if the description refers to internal states of the eventual dynamic programming algorithm, like "the current index" or "the best score so far". The function must have a well-defined value that depends *only* on its input parameters (and constant global variables).
  - An English explanation of the *recurrence* or *algorithm* does not qualify. We want a description of *what* your function returns, not (here) an explanation of *how* that value is computed.
- 4 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 for base case(s). −½ for one *minor* bug, like a typo or an off-by-one error.
  - + 3 for recursive case(s). −1 for each *minor* bug, like a typo or an off-by-one error.
  - − 2 for greedy optimizations without proof, even if they are correct.
  - **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 3 points for iterative details
  - + 1 for describing (or sketching) an appropriate memoization data structure
  - + 1 for describing (or sketching) a correct evaluation order; a clear picture is usually sufficient. If you use nested for loops, be sure to specify the nesting order.
  - + 1 for correct 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 specifically says otherwise.
- ***Iterative pseudocode 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. However, you ***do*** still need and English description of the underlying recursive function (or equivalently, the contents of the memoization structure). ***Perfectly correct iterative pseudocode, with no explanation or time analysis, is worth at most 6 points out of 10.***
- Partial credit for incomplete solutions depends on the running time of the ***best possible*** completion (up to the target running time). For example, consider a solution that contains *only* a clear English description of a function, with no recurrence or iterative details.
  - If the described function *can* be developed into an algorithm with the target running time, the solution is worth 3 points.
  - If the described function leads to an algorithm that is slower than the target time by a factor of n, the solution could be worth only 2 points (= 70% of 3, rounded).
  - If the described function cannot lead to a polynomial-time algorithm, it could be worth 1 or even 0 points.

CS 473 ✧ Fall 2022
Homework 4

Due Tuesday, October 4, 2022 at 9pm

Unless a problem specifically states otherwise, you may assume a function RANDOM that takes a positive integer k as input and returns an integer chosen uniformly and independently at random from $\{1, 2, \ldots, k\}$ in $O(1)$ time. For example, to flip a fair coin, you could call RANDOM(2).

---

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 number of heads and tails.
   (h) Hamlet flips the same positive number of heads and tails.
   (i) Hamlet flips more than twice as many heads as 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 randomly 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. 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.

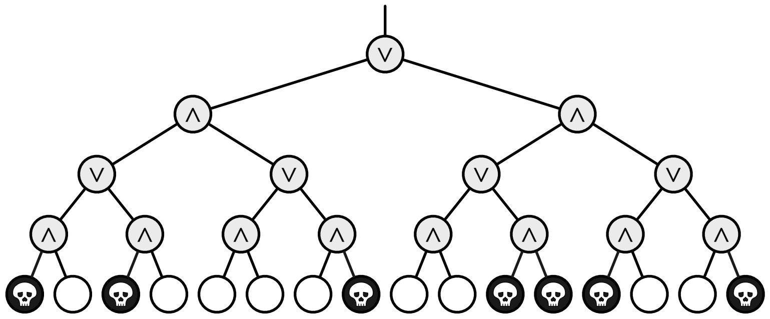
   (a) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, compute the number of elements of M larger than $M[i, j]$ and smaller than $M[i', j']$.

   (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 larger than $M[i, j]$ and smaller 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.

   (The algorithm for part (c) can be used as a subroutine to solve HW2.2 in $O(n \log^2 n)$ expected time!)

3. 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!]*

CS 473 ✧ Fall 2022
Homework 5

Due Tuesday, October 11 2022 at 9pm

Unless a problem specifically states otherwise, you may assume a function Random that takes a positive integer k as input and returns an integer chosen uniformly and independently at random from $\{1, 2, \ldots, k\}$ in $O(1)$ time. For example, to flip a fair coin, you could call Random(2).

---

1. Consider a random walk on a path with vertices numbered $1, 2, \ldots, n$ from left to right. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex n.

   (a) Prove that if we start at vertex 1, the probability that the walk ends by falling off the *right* end of the path is exactly $1/(n+1)$.

   (b) Prove that if we start at vertex k, the probability that the walk ends by falling off the *right* end of the path is exactly $k/(n+1)$.

   (c) Prove that if we start at vertex 1, the expected number of steps before the random walk ends is exactly n.

   (d) What is the *exact* expected length of the random walk if we start at vertex k, as a function of n and k? Prove your result is correct. (For partial credit, give a tight Θ-bound for the case $k = (n+1)/2$, assuming n is odd.)

   [Hint: Trust the recursion fairy. Yes, "see part (b)" is worth full credit for part (a), but only if your solution to part (b) is correct. Same for parts (c) and (d).]

2. ***Tabulation 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 $\mathcal{H}$ is 2-uniform.

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

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

   [Hint: Yes, "see part (b)" is worth full credit for (a), if your part (b) solution is correct.]

3. 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($\varepsilon$):
  $count \leftarrow 0$
  for $i \leftarrow 1$ to $N$
      if FLIP( ) = HEADS
          $count \leftarrow count + 1$
  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($\delta, \varepsilon$):
  for $j \leftarrow 1$ to $K$
      $estimate[j] \leftarrow$ MEANESTIMATE($\varepsilon$)
  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.]*

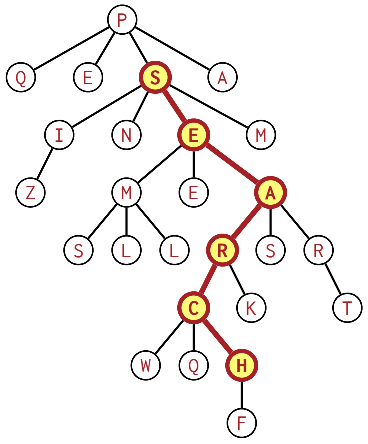
CS 473 ✧ Fall 2022
Homework 6

Due Tuesday, October 18, 2022 at 9pm

---

1. Describe and analyze an efficient algorithm to find strings in labeled rooted trees. Your input consists of a *pattern string* $P[1..m]$ and a rooted *text tree* T with n nodes, each labeled with a single character. Nodes in T can have any number of children. A path in T is called a *downward path* if every node on the path is a child (in T) of the previous node in the path. Your goal is to determine whether there is a downward path in T whose sequence of labels matches the string P.

   For example, the string SEARCH is the label of a downward path in the tree shown below, but the strings HCRAES and SMEAR is not.



2. A *fugue* (pronounced "fyoog") is a highly structured style of musical composition that was popular in the 17th and 18th centuries. A fugue begins with an initial melody, called the *subject*, that is repeated several times throughout the piece.

   Suppose we want to design an algorithm to detect the subject of a fugue. We will assume a *very* simple representation as an array $F[1..n]$ of integers, each representing a note in the fugue as the number of half-steps above or below middle C. (We are deliberately ignoring all other musical aspects of real-life fugues, like multiple voices, timing, rests, volume, and timbre.)

   (a) Describe an algorithm to find the length of the longest prefix of F that reappears later as a substring of F. The prefix and its later repetition must not overlap.

   (b) In many fugues, later occurrences of the subject are ***transposed***, meaning they are all shifted up or down by a common value. For example, the subject $(3, 1, 4, 1, 5, 9, 2)$ might be transposed transposed down two half-steps to $(1, -1, 2, -1, 3, 7, 0)$.

   Describe an algorithm to find the length of the longest prefix of F that reappears later, *possibly transposed*, as a substring of F. Again, the prefix and its later repetition must not overlap.

For example, if the input array is

$$\underbracket{\overbracket{3,1,4,1},5,9,2},6,5,\overbracket{3,1,4,\rlap{$\underbracket{1,-1,2,-1,3,7,0}$}},1,4,2$$

then your first algorithm should return 4, and your second algorithm should return 7.

3. There is no question 3!

CS 473 ✧ Fall 2022
Homework 7

Due Tuesday, October 25, 2022 at 9pm

---

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. 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. Let G be a flow network with integer edge capacities. An edge in G is *upper-binding* if increasing its capacity by 1 also increases the value of the maximum flow in G. Similarly, an edge is *lower-binding* if decreasing its capacity by 1 also decreases the value of the maximum flow in G.

   (a) Does every network G have at least one upper-binding edge? Prove your answer is correct.

   (b) Does every network G have at least one lower-binding edge? Prove your answer is correct.

   (c) Describe an algorithm to find all upper-binding edges in G, given both G and a maximum flow in G as input, in $O(E)$ time.

   (d) Describe an algorithm to find all lower-binding edges in G, given both G and a maximum flow in G as input, in $O(VE)$ time.

Standard graph-reduction rubric. For problems solved by reduction to a standard graph algorithm covered either in class or in a prerequisite class (for example: shortest paths, topological sort, minimum spanning trees, maximum flows, bipartite maximum matching, vertex-disjoint paths, ...).

Maximum 10 points =

+ 3 for constructing the correct graph.
    - +1 for correct vertices
    - +1 for correct edges
    - −½ for forgetting "directed" if the graph is directed
    - +1 for correct data associated with vertices and/or edges—for example, weights, lengths, capacities, costs, demands, and/or labels—if any
    - ∘ The vertices, edges, and associated data (if any) must be described as explicit functions of the input data.
- ∘ For most problems, the graph can be constructed in linear time by brute force; in this common case, no explicit description of the construction algorithm is required. If achieving the target running time requires a more complex algorithm, that algorithm will graded out of 5 points using the appropriate standard rubric, and all other points are cut in half.
+ 3 for explicitly relating the given problem to a specific **problem** involving the constructed graph. For example: "The minimum number of moves is equal to the length of the shortest path in G from $(0,0,0)$ to any vertex of the form $(k,\cdot,\cdot)$ or $(\cdot,k,\cdot)$ or $(\cdot,\cdot,k)$." or "Each path from s to t represents a valid choice of class, room, time, and proctor for one final exam; thus, we need to construct a path decomposition of a maximum (s,t)-flow in G."
    - – No points for just writing (for example) "shortest path" or "reachability" or "matching". Shortest path in which graph, from which vertex to which other vertex? How does that shortest path relate to the original problem?
    - – No points for only naming the algorithm, not the problem. "Breadth-first search" is not a problem!
+ 2 for correctly applying the correct black-box **algorithm** to solve the stated problem. (For example, "Perform a single breadth-first search in H from $(0,0,0)$ and then examine every target vertex." or "We compute the maximum flow using Ford-Fulkerson, and then decompose the flow into paths as described in the textbook.")
    - −1 for using a slower or more specific algorithm than necessary, for example, breadth- or depth-first search instead of whatever-first search, or Dijkstra's algorithm instead of breadth-first search.
    - −1 for *explaining* an algorithm from lecture or the textbook instead of just invoking it as a black box.
+ 2 for time analysis in terms of the *input* parameters (not just the number of vertices and edges of the constructed graph).

An extremely common mistake for this type of problem is to attempt to modify a standard algorithm and apply that modification to the input data, instead of modifying the input data and invoking a standard algorithm as a black box. This strategy can work in principle, but it is much harder to do it correctly, and it is terrible software engineering practice. ***Clearly correct*** solutions using this strategy will be given full credit, but partial credit will be given only sparingly.

CS 473 ✧ Fall 2022

Homework 8

Due **Wednesday**, November 9, 2022 at 9pm

---

1. The Autocratic Party is gearing up their fund-raising campaign for the 2024 election. Party leaders have already chosen their slate of candidates for president and vice-president, as well as various governors, senators, representatives, city council members, school board members, judges, and dog-catchers. For each candidate, the party leaders have determined how much money they must spend on that candidate's campaign to guarantee their election.

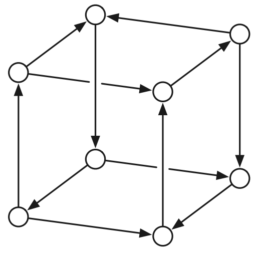
   The party is soliciting donations from each of its members. Each voter has declared the total amount of money they are willing to give each candidate between now and the election. (Each voter pledges different amounts to different candidates. For example, everyone is happy to donate to the presidential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

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

   Assume there are n candidates, p party members, and d days until the election. The input to your algorithm is a pair of arrays $Win[1..n]$ and $Limit[1..p, 1..n]$, where $Win[i]$ is the amount of money candidate i needs to win, and $Limit[i, j]$ is the total amount party member i is willing to donate to candidate j.

   Your algorithm should return an array $Donate[1..p, 1..n, 1..d]$, where $Donate[i, j, k]$ is the amount of money party member i should donate to candidate j on day k.

2. A ***k*-orientation** of an undirected graph G is an assignment of directions to the edges of G so that every vertex of G has at most k incoming edges. For example, the figure below shows a 2-orientation of the graph of the cube.



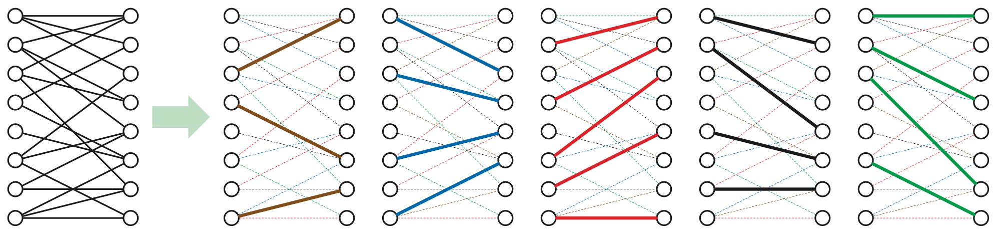
[1]or some nice men in suits will be visiting their home.
[2]See previous footnote.

Describe and analyze an algorithm that determines the smallest value of k such that G has a k-orientation, given the undirected graph G as input. Equivalently, your algorithm should find an orientation of the edges of G such that the maximum in-degree is as small as possible. For example, given the cube graph as input, your algorithm should return the integer 2.

3. Let $G = (L \sqcup R, E)$ be a bipartite graph, whose left vertices L are indexed $\ell_1, \ell_2, \ldots, \ell_n$ and whose right vertices are indexed $r_1, r_2, \ldots, r_n$. A matching M in G is ***non-crossing*** if, for every pair of edges $\ell_i r_j$ and $\ell_{i'} r_{j'}$ in M, we have $i < i'$ if and only if $j < j'$. If we place the vertices of G in index order along two vertical lines and draw the edges of G as straight line segments, a matching is non-crossing if its edges do not intersect.

   Describe and analyze an algorithm to find the smallest number of disjoint non-crossing matchings $M_1, M_2, \ldots, M_k$ such that each edge in G lies in exactly one matching M_i.

   [Hint: How would you compute the largest non-crossing matching in G?]


Decomposing a bipartite graph into non-crossing matchings.

4. **[just for practice, not for submission]** Suppose we are given a chessboard with certain squares removed, represented as a two-dimensional boolean array $A[1..n, 1..n]$. Describe and analyze efficient algorithms to place as many chess pieces of a given type onto the board as possible, so that no two pieces attack each other. A piece can be placed on the square in row i and column j if and only if $A[i, j] = \textsc{True}$. Specifically:

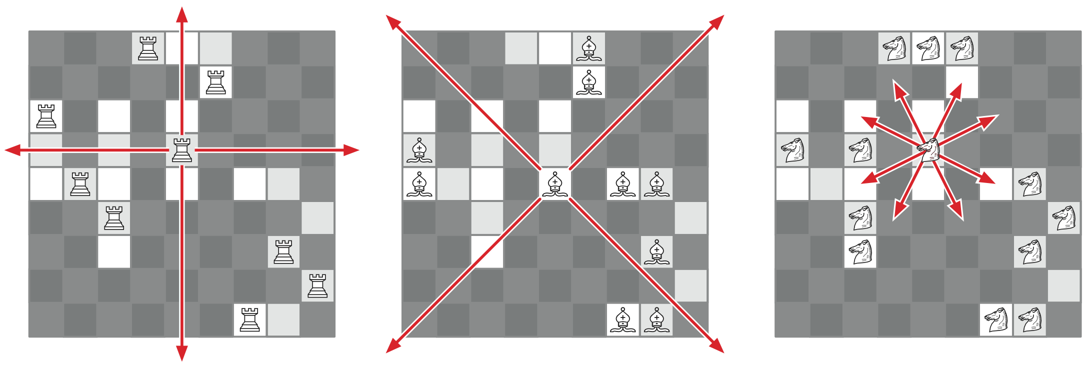
   (a) Describe an algorithm to places as many *rooks* on the board as possible. A rook on square (i, j) attacks every square in the same row or column; that is, every square of the form (i, k) or (k, j).

   (b) Describe an algorithm to places as many *bishops* on the board as possible. A bishop on square (i, j) attacks every square on the same diagonal or back-diagonal; that is, every square of the form $(i + k, j + k)$ or $(i + k, j - k)$.

   ⋆(c) Describe an algorithm to places as many *knights* on the board as possible. A knight on square (i, j) attacks the eight squares $(i \pm 1, j \pm 2)$ and $(i \pm 2, j \pm 1)$.

   ★(d) Prove that placing as many *queens* on the board as possible is NP-hard. A queen attacks like either a rook or a bishop; that is, it attacks every square on the same row, column, diagonal, or back-diagonal.

   Examples of (a), (b), and (c) are shown on the next page.


From left to right: Rooks, bishops, and knights.

CS 473 ✧ Fall 2022
Homework 9

Due **Thursday**, November 17, 2022 at 9pm

---

1. 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.

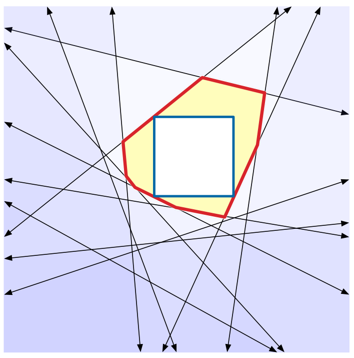
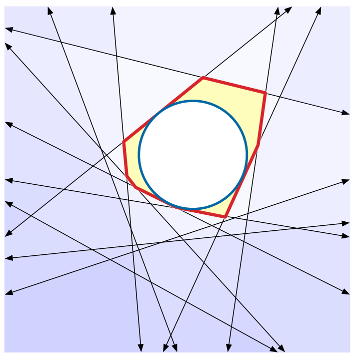
   Conversely, Dumbledore knows how many instructors are needed for each committee, and he has compiled a list of instructors who would be suitable members for each committee. (For example: "Dark Arts Revision: 5 members, anyone but Snape.") If Dumbledore assigns an instructor to a committee, he must pay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

   Dumbledore needs to assign instructors to committees so that (1) each committee is full, (2) only suitable and willing instructors are assigned to each committee, (3) no instructor is assigned to more than three committees, 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.

2. Suppose we are given a sequence of n linear inequalities of the form $a_i x + b_i y \le c_i$. Collectively, these n inequalities describe a convex polygon P in the plane.

   (a) Describe a linear program whose solution describes the largest square with horizontal and vertical sides that lies entirely inside P.

   (b) Describe a linear program whose solution describes the largest circle that lies entirely inside P.

3. Alex and Bo are playing ***Undercut***. Each player puts their right hand behind their back and raises some number of fingers; then both players reveal their right hands simultaneously. Thus, each player independently chooses an integer from 0 to 5.[1] If the two numbers do not differ by 1, each player adds their own number to their score. However, if the two numbers differ by 1, then the player with the lower number adds *both* numbers to their score, and the other player gets nothing. Both players want to maximize their score and minimize their opponent's score.

   Because Alex and Bo only care about the *difference* between their scores, we can reformulate the problem as follows. If Alex chooses the number i and Bo chooses the number j, then Alex gets M_{ij} points, where M is the following 6×6 matrix:

$$M = \begin{pmatrix} 0 & 1 & -2 & -3 & -4 & -5 \\ -1 & 0 & 3 & -2 & -3 & -4 \\ 2 & -3 & 0 & 5 & -2 & -3 \\ 3 & 2 & -5 & 0 & 7 & -2 \\ 4 & 3 & 2 & -7 & 0 & 9 \\ 5 & 4 & 3 & 2 & -9 & 0 \end{pmatrix}$$

   (In this formulation, Bo's score is always zero.) Alex wants to maximize Alex's score; Bo wants to minimize it.

   Neither player has a good *deterministic* strategy; for example, if Alex always plays 4, then Bo should always play 3. Exhausted from trying to out-double-think each other,[2] they both decide to adopt *randomized* strategies. These strategies can be described by two vectors $a = (a_0, a_1, a_2, a_3, a_4, a_5)^\top$ and $b = (b_0, b_1, b_2, b_3, b_4, b_5)^\top$, where a_i is the probability that Alex chooses i, and b_j is the probability that Bo chooses j. Because Alex and Bo's random choices are independent, Alex's expected score is $a^T M b = \sum_{i=0}^{5} \sum_{j=0}^{5} a_i M_{ij} b_j$.

   (a) Suppose Bo somehow learns Alex's strategy vector a. Describe a linear program whose solution is Bo's best possible strategy vector.

   (b) What is the dual of your linear program from part (b)?

   (c) So what *is* Bo's optimal strategy, as a function of the vector a? And what is Alex's resulting expected score? (You should be able to answer this part even without answering parts (a) and (b).)

   (d) Now suppose that Alex knows that Bo will discover Alex's strategy vector before they actually start playing. Describe a linear program whose solution is Alex's best possible strategy vector.

   (e) What is the dual of your linear program from part (d)?

   (f) **Extra credit:** So what *is* Alex's optimal Undercut strategy, if Alex knows that Bo will know that strategy?

   (g) **Extra credit:** If Bo knows that Alex is going to use their optimal strategy from part (f), what is Bo's optimal Undercut strategy?

   Please express your answers to parts (a)–(e) in terms of arbitrary $n \times n$ payoff matrices M, instead of this specific example. You may find a computer helpful for parts (f) and (g).

[1] In Hofstadter's original game, players were not allowed to choose 0 for some reason.

[2] "They were both poisoned. I've spent the last several years building up an immunity to iocaine powder."

CS 473 ✧ Fall 2022
"Homework" 10

Practice only — Solutions will be released on Monday, December 5, 2022

---

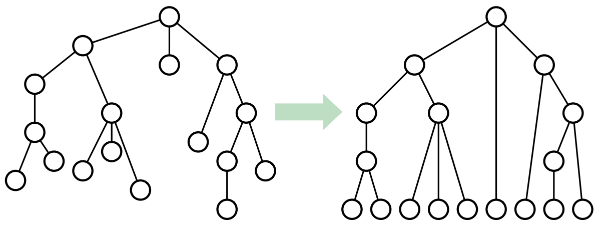
1. Alex and Bo are playing another game with even more complex rules. Each player independently chooses an integer between 0 and n, then both players simultaneously reveal their choices, and finally they get points based on those choices.

   Chris and Dylan are watching the game, but they don't really understand the scoring rules, so instead, they decide to place bets on the *sum* of Alex and Bo's choices. They both somehow know the probabilities that Alex and Bo use, and they want to figure out the probability of each possible sum.

   Suppose Chris and Dylan are given a pair of arrays $A[0..n]$ and $B[0..n]$, where $A[i]$ is the probability that Alex chooses i, and $B[j]$ is the probability that Bo chooses j. Describe and analyze an algorithm that computes an array $P[0..2n]$, where $P[k]$ is the probability that the sum of Alex and Bo's choices is equal to k.

2. Suppose you are given a rooted tree T, where every edge e has two associated values: a non-negative *length* $\ell(e)$ and a *cost* $\$(e)$ (which could be positive, negative, or zero). Your goal is to add a non-negative *stretch* $s(e) \geq 0$ to the length of every edge e in T, subject to the following conditions:

   - Every root-to-leaf path π in T has the same total stretched length $\sum_{e\in\pi}(\ell(e)+s(e))$
   - The total *weighted* stretch $\sum_e s(e)\cdot\$(e)$ is as small as possible.



   (a) Describe an instance of this problem with no optimal solution.

   (b) Give a concise linear programming formulation of this problem.

   (c) Suppose that for the given tree T and the given lengths and costs, the optimal solution to this problem is unique. Prove that in the optimal solution, $s(e)=0$ for every edge on some longest root-to-leaf path in T. In other words, prove that the optimally stretched tree has the same depth as the input tree. *[Hint: What is a basis in your linear program? When is a basis feasible?]*

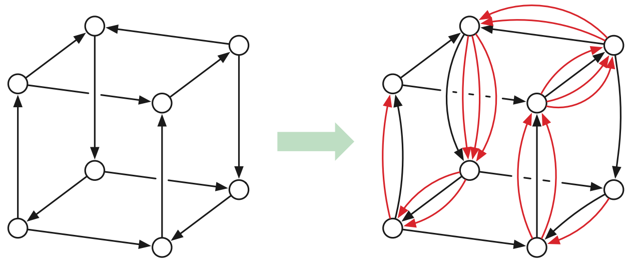
   (d) Describe and analyze a self-contained algorithm that solves this problem in (explicit) polynomial time. Your algorithm should either compute the minimum total weighted stretch, or report correctly that the total weighted stretch can be made arbitrarily negative.

3. An *Euler tour* in a directed graph G is a closed walk (starting and ending at the same vertex) that traverses every edge in G exactly once; a directed graph is *Eulerian* if it has an Euler tour. Euler tours are named after Leonhard Euler, who was the first person to systematically study them, starting with the Bridges of Königsberg puzzle.

   (a) Prove that a directed graph G with no isolated vertices is Eulerian if and only if (1) G is strongly connected—for any two vertices u and v, there is a directed walk in G from u to v and a directed walk in G from v to u—and (2) the in-degree of each vertex of G is equal to its out-degree. *[Hint: Flow decomposition!]*

   (b) Suppose that we are given a strongly connected directed graph G with no isolated vertices that is *not* Eulerian, and we want to make G Eulerian by duplicating existing edges. Each edge e has a duplication cost $€(e) \geq 0$. We are allowed to add as many copies of an existing edge e as we like, but we must pay $€(e)$ for each new copy. On the other hand, if G does not already have an edge from vertex u to vertex v, we cannot add a new edge from u to v.

   Describe an algorithm that computes the minimum-cost set of edge-duplications that makes G Eulerian.


Making a directed cube graph Eulerian.

4. *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 $S$):
  $\ell \leftarrow 0$
  while $S$ is not done
      $x \leftarrow$ next item in $S$
      $\ell \leftarrow \ell + 1$
      if Random($\ell$) $= 1$
          $\boldsymbol{sample \leftarrow x}$        $(\star)$
  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 problems, S denotes a stream of (unknown) length n.

(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?

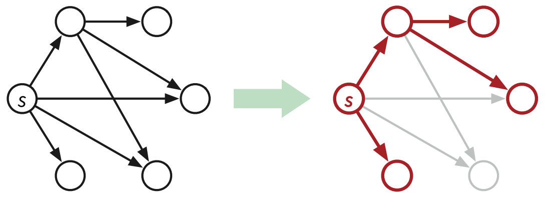
(d) What is the *exact* expected value of ℓ when either GetOneSample(S) executes line ($\star$) for the *second* time (or the algorithm ends, whichever happens first)?

(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 $1/6$.

5. Suppose you are given a directed acyclic graph G with a single source vertex s. Describe an algorithm to determine whether G contains a ***spanning binary tree***. Your algorithm is looking for a spanning tree T of G, such that every vertex in G has at most two outgoing edges in T and every vertex of G *except s* has exactly one incoming edge in T.

For example, given the dag on the left below as input, your algorithm should False, because the largest binary subtree excludes one of the vertices.



6. 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). Do not use the words “linear”, “program”, or “dual”. Yes, you have seen this problem before.

(d) Describe and analyze a self-contained 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”.