CS/ECE 374 A ✧ Fall 2023

Midterm 2 Study Questions

This is a "core dump" of potential questions for Midterm 2. This should give you a good idea of the *types* of questions that we will ask on the exam, but the actual exam questions may or may not appear in this list. This list intentionally includes a few questions that are too long or difficult for exam conditions; *most* of these are indicated with a *star.

Questions from Jeff's past exams are labeled with the semester they were used, for example, ***⟨⟨S18⟩⟩*** or ***⟨⟨F19⟩⟩***. Questions from this semester's homework are labeled ***⟨⟨HW⟩⟩***. Questions from this semester's labs are labeled ***⟨⟨Lab⟩⟩***. Some unflagged questions may have been used in exams by other instructors.

How to Use These Problems

Solving every problem in this handout is ***not*** the best way to study for the exam. Memorizing the solutions to every problem in this handout is the ***absolute worst*** way to study for the exam.

What we recommend instead is to work on a *sample* of the problems. Choose one or two problems at random from each section and try to solve them from scratch under exam conditions—by yourself, in a quiet room, with a 30-minute timer, *without* your notes, *without* the internet, and if possible, even without your cheat sheet. If you're comfortable solving a few problems in a particular section, you're probably ready for that type of problem on the exam. Move on to the next section.

Discussing problems with other people (in your study groups, in the review sessions, in office hours, or on Piazza) and/or looking up old solutions can be *extremely* helpful, but ***only after*** you have (1) made a good-faith effort to solve the problem on your own, and (2) you have either a candidate solution or some idea about where you're getting stuck.

If you find yourself getting stuck on a particular type of problem, try to figure out *why* you're stuck. Do you understand the problem statement? Have you tried several example inputs to see what the correct output *should* be? Are you stuck on choosing the right high-level approach, are you stuck on the details, or are you struggling to express your ideas clearly?

Similarly, if feedback suggests that your solutions to a particular type of problem are incorrect or incomplete, try to figure out what you missed. For recursion/dynamic programming: Are you solving the right recursive generalization of the stated problem? Are you having trouble writing a specification of the function, as opposed to a description of the algorithm? Are you struggling to find a good evaluation order? Are you trying to use a greedy algorithm? *[Hint: Don't.]* For graph algorithms: Are you aiming for the right problem? Are you having trouble figuring out the interesting states of the problem (otherwise known as vertices) and the transitions between them (otherwise known as edges)? Do you keep trying to modify the algorithm instead of modifying the graph? *[Hint: Don't.]*

Remember that your goal is *not* merely to "understand" the solution to any particular problem, but to become more comfortable with solving a certain *type* of problem on your own. **"Understanding" is a trap; aim for mastery.** If you can identify specific steps that you find problematic, read more *about those steps*, focus your practice *on those steps*, and try to find helpful information *about those steps* to write on your cheat sheet. Then work on the next problem!

Short answers

1. Solve the recurrence $T(n) = 3T(n/2) + O(n^2)$.
2. Solve the recurrence $T(n) = 7T(n/2) + O(n^2)$.
3. Solve the recurrence $T(n) = 4T(n/2) + O(n^2)$.
4. Solve the recurrence $T(n) = 2T(n/3) + O(\sqrt{n})$.
5. Solve the recurrence $T(n) = 2T(n/7) + O(\sqrt{n})$.
6. Solve the recurrence $T(n) = 2T(n/4) + O(\sqrt{n})$.
7. Solve the recurrence $T(n) = 16\,T(n/2) + O(n^3)$.
8. Solve the recurrence $T(n) = 16\,T(n/2) + O(n^7)$.
9. Solve the recurrence $T(n) = 16\,T(n/2) + O(n^4)$.

10. Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrence, and give the running time of the resulting iterative algorithm to compute $\mathit{Reeee}(1, n)$. (Assume all array accesses are legal.)

$$\mathit{Reeee}(i,k) = \begin{cases} 0 & \text{if } i > k \\ A[i] & \text{if } i = k \\ \mathit{Reeee}(i+1,k-1) + A[i] + A[k] & \text{if } A[i] = A[k] \\ \max \left\{ \begin{array}{c} \mathit{Reeee}(i+2,k), \\ \mathit{Reeee}(i+1,k-1), \\ \mathit{Reeee}(i,k-2) \end{array} \right\} & \text{otherwise} \end{cases}$$

11. Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrence, and give the running time of the resulting iterative algorithm to compute $\mathit{Huh}(1, n)$.

$$\mathit{Huh}(i,k) = \begin{cases} 0 & \text{if } i > n \text{ or } k < 0 \\ \min \left\{ \begin{array}{l} \mathit{Huh}(i+1,k-2), \\ \mathit{Huh}(i+2,k-1) \end{array} \right\} + A[i,k] & \text{if } A[i,k] \text{ is even} \\ \min \left\{ \begin{array}{l} \mathit{Huh}(i+1,k-2), \\ \mathit{Huh}(i+2,k-1) \end{array} \right\} - A[i,k] & \text{if } A[i,k] \text{ is odd} \end{cases}$$

12. Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrence, and give the running time of the resulting iterative algorithm to compute $\mathit{Spoopy}(1, 1)$.

$$\mathit{Spoopy}(i,k) = \begin{cases} 0 & \text{if } i > n \text{ or } k > n \\ \min \left\{ \begin{array}{l} \mathit{Spoopy}(i,k+j) \\ \quad + (k-i)\cdot A[j] \\ \quad + \mathit{Spoopy}(i+j,k) \end{array} \;\middle|\; 1 \le j \le n \right\} & \text{otherwise} \end{cases}$$

13. Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrence, and give the running time of the resulting iterative algorithm to compute $\mathit{What}(1, n)$.

$$\mathit{What}(i, j) = \begin{cases} 0 & \text{if } i > n \text{ or } j < 0 \\ \max \left\{ \begin{array}{c} \mathit{What}(i, j+1) \\ \mathit{What}(i-1, j) \\ A[i] \cdot A[j] + \mathit{What}(i-1, j+1) \end{array} \right\} & \text{otherwise} \end{cases}$$

Recursion and Dynamic Programming

Elementary Recursion/Divide and Conquer

1. ⟨⟨*Lab*⟩⟩

   (a) Suppose $A[1..n]$ is an array of n distinct integers, sorted so that $A[1] < A[2] < \cdots < A[n]$. Each integer $A[i]$ could be positive, negative, or zero. Describe a fast algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists..

   (b) Now suppose $A[1..n]$ is a sorted array of n distinct ***positive*** integers. Describe an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. *[Hint: This is* **really** *easy.]*

2. ⟨⟨*Lab*⟩⟩ Suppose we are given an array $A[1..n]$ such that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a ***local minimum*** if both $A[x-1] \geq A[x]$ and $A[x] \leq A[x+1]$. For example, there are exactly six local minima in the following array:

9	**7**	7	2	**1**	3	7	5	**4**	7	**3**	**3**	4	8	**6**	9
	▲			▲				▲		▲	▲			▲	

   Describe and analyze a fast algorithm that returns the index of one local minimum. For example, given the array above, your algorithm could return the integer 5, because $A[5]$ is a local minimum. *[Hint: With the given boundary conditions, any array* ***must*** *contain at least one local minimum. Why?]*

3. ⟨⟨*Lab*⟩⟩ Suppose you are given two sorted arrays $A[1..n]$ and $B[1..n]$ containing distinct integers. Describe a fast algorithm to find the median (meaning the nth smallest element) of the union $A \cup B$. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1..8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

   your algorithm should return the integer 9. *[Hint: What can you learn by comparing one element of A with one element of B?]*

4. ⟨⟨*F14, S14*⟩⟩ An array $A[0..n-1]$ of n distinct numbers is ***bitonic*** if there are unique indices i and j such that $A[(i-1) \bmod n] < A[i] > A[(i+1) \bmod n]$ and $A[(j-1) \bmod n] > A[j] < A[(j+1) \bmod n]$. In other words, a bitonic sequence either consists of an increasing sequence followed by a decreasing sequence, or can be circularly shifted to become so. For example,

4	6	9	8	7	5	1	2	3	is bitonic, but
3	6	9	8	7	5	1	2	4	is *not* bitonic.

   Describe and analyze an algorithm to find the index of the *smallest* element in a given bitonic array $A[0..n-1]$ in $O(\log n)$ time. You may assume that the numbers in the input array are distinct. For example, given the first array above, your algorithm should return 6, because $A[6] = 1$ is the smallest element in that array.

5. ⟨⟨*F16*⟩⟩ Suppose you are given a sorted array $A[1..n]$ of distinct numbers that has been *rotated* k steps, for some ***unknown*** integer k between 1 and $n-1$. That is, the prefix $A[1..k]$ is sorted in increasing order, the suffix $A[k+1..n]$ is sorted in increasing order, and $A[n] < A[1]$. For example, you might be given the following 16-element array (where $k = 10$):

9	13	16	18	19	23	28	31	37	42	−4	0	2	5	7	8

Describe and analyze an algorithm to determine if the given array contains a given number x. The input to your algorithm is the array $A[1..n]$ and the number x; your algorithm is ***not*** given the integer k.

6. ⟨⟨*S22*⟩⟩ Suppose you are given an array $A[1..n]$ of distinct numbers that contains exactly one local maximum. That is, for some ***unknown*** index k, the prefix $A[1..k]$ is sorted in increasing order, and the suffix $A[k..n]$ is sorted in decreasing order. You are also given a real number x. Describe and analyze an algorithm to determine whether $A[i] = x$ for any index i.

7. ⟨⟨*F16*⟩⟩ Suppose you are given two unsorted arrays $A[1..n]$ and $B[1..n]$ containing $2n$ distinct integers, such that $A[1] < B[1]$ and $A[n] > B[n]$. Describe and analyze an efficient algorithm to compute an index i such that $A[i] < B[i]$ and $A[i+1] > B[i+1]$. *[Hint: Why does such an index i always exist?]*

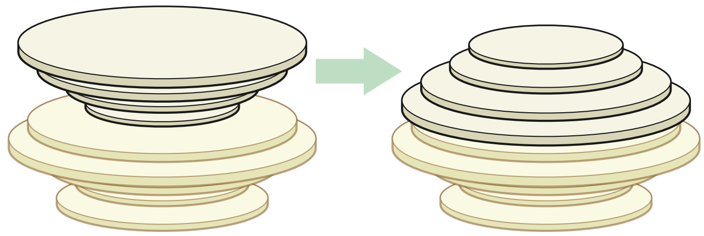
8. (a) Describe an algorithm to determine in $O(n)$ time whether an arbitrary array $A[1..n]$ contains more than $n/4$ copies of any value.

   (b) Describe and analyze an algorithm to determine, given an arbitrary array $A[1..n]$ and an integer k, whether A contains more than k copies of any value. Express the running time of your algorithm as a function of both n and k.

   Do not use hashing, or radix sort, or any other method that depends on the precise input values, as opposed to their order.

9. 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.

   (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?

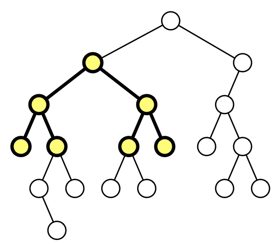


Flipping the top four pancakes.

(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!]*

10. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return both the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

*11. ⟨⟨*S18*⟩⟩ Suppose you have an integer array $A[1..n]$ that *used* to be sorted, but Swedish hackers have overwritten k entries of A with random numbers. Because you carefully monitor your system for intrusions, you know *how many* entries of A are corrupted, but not *which* entries or what the values are.

Describe an algorithm to determine whether your corrupted array A contains an integer x. Your input consists of the array A, the integer k, and the target integer x. For example, if A is the following array, $k = 4$, and $x = 17$, your algorithm should return TRUE. (The corrupted entries of the array are shaded.)

2	3	99	7	11	13	17	19	25	29	31	−5	41	43	47	53	8	61	67	71

Assume that x is not equal to any of the the corrupted values, and that all n array entries are distinct. Report the running time of your algorithm as a function of n and k. A solution only for the special case $k = 1$ is worth 5 points; a complete solution for arbitrary k is worth 10 points. *[Hint: First consider $k = 0$; then consider $k = 1$.]*

12. ⟨⟨*F21*⟩⟩ Your grandmother dies and leaves you her treasured collection of n radioactive Beanie Babies. Her will reveals that one of the Beanie Babies is a rare specimen worth 374 million dollars, but all the others are worthless. The valuable Beanie Baby is either slightly more or slightly less radioactive than the others, but you don't know which. Otherwise, as far as you can tell, they are all identical.

    You have access to a state-of-the-art Radiation Comparator at your job. The Comparator has two chambers. You can place any two disjoint sets of Beanie Babies in Comparator's two chambers; the Detector will then indicate which of the two subsets emits more radiation, or that the two subsets are equally radioactive. (The two subsets are equally radioactive if and only if they contain the same number of Beanie Babies, and they are all worthless.) The Comparator is slow and consumes a *lot* of power, and you really aren't supposed to use it for personal projects, so you *really* want to use it as few times as possible.

    Describe an efficient algorithm to identify the valuable Beanie Baby. How many times does your algorithm use the Comparator in the worst case, as a function of n?

Dynamic Programming

1. ⟨⟨*Lab*⟩⟩ Describe and analyze efficient algorithms for the following problems.

   (a) Given an array $A[1..n]$ of integers, compute the length of a longest ***increasing*** subsequence of A. A sequence $B[1..\ell]$ is *increasing* if $B[i] > B[i-1]$ for every index $i \geq 2$.

   (b) Given an array $A[1..n]$ of integers, compute the length of a longest ***decreasing*** subsequence of A. A sequence $B[1..\ell]$ is *decreasing* if $B[i] < B[i-1]$ for every index $i \geq 2$.

   (c) Given an array $A[1..n]$ of integers, compute the length of a longest ***alternating*** subsequence of A. A sequence $B[1..\ell]$ is *alternating* if $B[i] < B[i-1]$ for every even index $i \geq 2$, and $B[i] > B[i-1]$ for every odd index $i \geq 3$.

   (d) Given an array $A[1..n]$ of integers, compute the length of a longest ***convex*** subsequence of A. A sequence $B[1..\ell]$ is *convex* if $B[i]-B[i-1] > B[i-1]-B[i-2]$ for every index $i \geq 3$.

   (e) Given an array $A[1..n]$, compute the length of a longest ***palindrome*** subsequence of A. Recall that a sequence $B[1..\ell]$ is a *palindrome* if $B[i] = B[\ell-i+1]$ for every index i.

2. ⟨⟨*F19*⟩⟩

   (a) Recall that a *palindrome* is any string that is equal to its reversal, like `REDIVIDER` or `POOP`. Describe an algorithm to find the length of the longest subsequence of a given string that is a palindrome.

   (b) A *double palindrome* is the concatenation of two *non-empty* palindromes, like `AREDIVIDER` or `POOPPOOP`. Describe an algorithm to find the length of the longest subsequence of a given string that is a *double* palindrome. *[Hint: Use your algorithm from part (a).]*

   For both algorithms, the input is an array $A[1..n]$, and the output is an integer. For example, given the string `MAYBEDYNAMICPROGRAMMING` as input, your algorithm for part (a) should return 7 (for the palindrome subsequence `NMRORMN`), and your algorithm for part (b) should return 12 (for the double palindrome subsequence `MAYBYAMIRORI`).

3. Recall that a *palindrome* is any string that is the same as its reversal. For example, `I`, `DAD`, `HANNAH`, `AIBOHPHOBIA` (fear of palindromes), and the empty string are all palindromes.

   (a) ⟨⟨*S14*⟩⟩ Describe and analyze an algorithm to find the length of the longest substring (not *subsequence*!) of a given input string that is a palindrome. For example, **BASEESAB** is the longest palindrome substring of `BUB`**BASEESAB**`ANANA` ("Bubba sees a banana."). Thus, given the input string `BUBBASEESABANANA`, your algorithm should return the integer 8.

   (b) ⟨⟨*Lab, F16*⟩⟩ Describe and analyze an algorithm to find the length of the longest subsequence (not *substring*!) of a given input string that is a palindrome. For example, the longest palindrome subsequence of **M**A**H**D**Y**NA**M**ICP**RO**G**R**AMZLET**M**ESHOW**Y**OUT**H**E**M** is MHYMRORMYHM, so given that string as input, your algorithm should output the number 11.

(c) ⟨⟨*S14*⟩⟩ Any string can be decomposed into a sequence of palindrome substrings. For example, the string BUBBASEESABANANA 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 + A + N + A

Describe and analyze an algorithm to find the smallest number of palindromes that make up a given input string. For example:

- Given the string BUBBASEESABANANA, your algorithm should return the integer 3.
- Given the string PALINDROME, your algorithm should return the integer 10.
- Given the string RACECAR, your algorithm should return the integer 1.

(d) A ***metapalindrome*** is a decomposition of a string into a sequence of non-empty palindromes, such that the sequence of palindrome lengths is itself a palindrome. For example, the decomposition

BUB • B • ALA • SEES • ABA • N • ANA

is a metapalindrome for the string BUBBALASEESABANANA, with the palindromic length sequence $(3, 1, 3, 4, 3, 1, 3)$. Describe and analyze an efficient algorithm to find the length of the shortest metapalindrome for a given string. For example:

- Given the string BUBBALASEESABANANA, your algorithm should return the integer 7.
- Given the string PALINDROME, your algorithm should return the integer 10.
- Given the string DEPOPED, your algorithm should return the integer 1.

4. ⟨⟨***F16***⟩⟩ It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the the list of n songs that the judges will play during the contest, in chronological order.

   You know all the songs, all the judges, and your own dancing ability extremely well. For each integer k, you know that if you dance to the kth song on the schedule, you will be awarded exactly $Score[k]$ points, but then you will be physically unable to dance for the next $Wait[k]$ songs (that is, you cannot dance to songs $k+1$ through $k + Wait[k]$). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

   Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays $Score[1..n]$ and $Wait[1..n]$.

5. ⟨⟨*S16*⟩⟩ After the Revolutionary War, Alexander Hamilton's biggest rival as a lawyer was Aaron Burr. (Sir!) In fact, the two worked next door to each other. Unlike Hamilton, Burr cannot work non-stop; every case he tries exhausts him. The bigger the case, the longer he must rest before he is well enough to take the next case. (Of course, he is willing to wait for it.) If a case arrives while Burr is resting, Hamilton snatches it up instead.

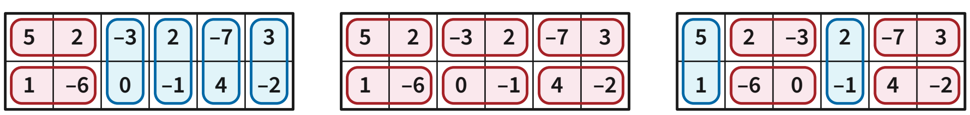
   Burr has been asked to consider a sequence of n upcoming cases. He quickly computes two arrays $\mathit{profit}[1..n]$ and $\mathit{skip}[1..n]$, where for each index i,

   - $\mathit{profit}[i]$ is the amount of money Burr would make by taking the ith case, and
   - $\mathit{skip}[i]$ is the number of consecutive cases Burr must skip if he accepts the ith case. That is, if Burr accepts the ith case, he cannot accept cases $i+1$ through $i+\mathit{skip}[i]$.

   Design and analyze an algorithm that determines the maximum total profit Burr can secure from these n cases, using his two arrays as input.

6. ⟨⟨*F21*⟩⟩ Suppose you are asked to tile a $2\times n$ grid of squares with dominos (1×2 rectangles). Each domino must cover exactly two grid squares, either horizontally or vertically, and each grid square must be covered by exactly one domino.

   Each grid square is worth some number of points, which could be positive, negative, or zero. The ***total value*** of a domino tiling is the sum of the points in squares covered by vertical dominos, *minus* the sum of the points in squares covered by horizontal dominos. The following figure shows three examples of domino tilings of the same 2×6 grid, along with their total values.



total value = –6 | total value = 2 | total value = 16

   Describe and analyze an efficient algorithm to compute the largest possible value of a domino tiling of a given $2\times n$ grid. Your input is an array $\mathit{Points}[1..2, 1..n]$ of point values. For example, given the grid shown above, your algorithm should return the integer 16.

7. ⟨⟨*F16*⟩⟩ 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. For example, the string BANANAANANAS is a shuffle of the strings BANANA and ANANAS in several different ways.

$$\texttt{BANANA}{}_{\texttt{ANANAS}} \qquad \texttt{BAN}{}_{\texttt{ANA}}{}^{\texttt{ANA}}{}_{\texttt{NAS}} \qquad \texttt{B}{}_{\texttt{AN}}{}^{\texttt{AN}}{}_{\texttt{A}}{}^{\texttt{A}}{}_{\texttt{NA}}{}^{\texttt{NA}}{}_{\texttt{S}}$$

   Similarly, the strings PRODGYRNAMAMMIINCG and DYPRONGARMAMMICING are both shuffles of DYNAMIC and PROGRAMMING:

$$\texttt{PRO}{}^{\texttt{D}}{}_{\texttt{G}}{}^{\texttt{Y}}{}_{\texttt{R}}{}^{\texttt{NAM}}{}_{\texttt{AMMI}}{}^{\texttt{I}}{}_{\texttt{N}}{}^{\texttt{C}}{}_{\texttt{G}} \qquad \texttt{DY}{}_{\texttt{PRO}}{}^{\texttt{N}}{}_{\texttt{G}}{}^{\texttt{A}}{}_{\texttt{R}}{}^{\texttt{M}}{}_{\texttt{AMM}}{}^{\texttt{IC}}{}_{\texttt{ING}}$$

   Describe and analyze an efficient algorithm to determine, given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, whether C is a shuffle of A and B.

8. ⟨⟨*F21*⟩⟩ Suppose we are given an n-digit integer X. Repeatedly remove one digit from either end of X (your choice) until no digits are left. The *square-depth* of X is the maximum number of perfect squares that you can see during this process. For example, the number 32492 has square-depth 3, by the following sequence of removals:

$$32492 \to \overset{57^2}{3249} \to \overset{18^2}{324} \to 24 \to \overset{2^2}{4} \to \varepsilon.$$

Describe and analyze an algorithm to compute the square-depth of a given integer X, represented as an array $X[1..n]$ of n decimal digits. Assume you have access to a subroutine IsSquare that determines whether a given k-digit number (represented by an array of digits) is a perfect square ***in $O(k^2)$ time***.

9. Suppose you are given a sequence of non-negative integers separated by $+$ and $\times$ signs; for example:

$$2 \times 3 + 0 \times 6 \times 1 + 4 \times 2$$

You can change the value of this expression by adding parentheses in different places. For example:

$$2 \times (3 + (0 \times (6 \times (1 + (4 \times 2))))) = 6$$
$$(((((2 \times 3) + 0) \times 6) \times 1) + 4) \times 2 = 80$$
$$((2 \times 3) + (0 \times 6)) \times (1 + (4 \times 2)) = 108$$
$$(((2 \times 3) + 0) \times 6) \times ((1 + 4) \times 2) = 360$$

Describe and analyze an algorithm to compute, given a list of integers separated by $+$ and $\times$ signs, the smallest possible value we can obtain by inserting parentheses.

Your input is an array $A[0..2n]$ where each $A[i]$ is an integer if i is even and $+$ or $\times$ if i is odd. Assume any arithmetic operation in your algorithm takes $O(1)$ time.

10. Suppose you are given three strings $A[1..n]$, $B[1..n]$, and $C[1..n]$.

    (a) Describe and analyze an algorithm to find the length of the longest common subsequence of all three strings. For example, given the input strings

    $$A = \texttt{AxxBxxCDxEF}, \qquad B = \texttt{yyABCDyEyFy}, \qquad C = \texttt{zAzzBCDzEFz},$$

    your algorithm should output the number 6, which is the length of the longest common subsequence `ABCDEF`.

    (b) Describe and analyze an algorithm to find the length of the shortest common supersequence of all three strings. For example, given the input strings

    $$A = \texttt{AxxBxxCDxEF}, \qquad B = \texttt{yyABCDyEyFy}, \qquad C = \texttt{zAzzBCDzEFz},$$

    your algorithm should output the number 21, which is the length of the shortest common supersequence `yzyAxzzxBxxCDxyzEyFzy`.

11. (a) Suppose we are given a set L of n line segments in the plane, where each segment has one endpoint on the line $y = 0$ and one endpoint on the line $y = 1$, 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.

    (b) Suppose we are given a set L of n line segments in the plane, where each segment has one endpoint on the line $y = 0$ and one endpoint on the line $y = 1$, and all $2n$ endpoints are distinct. Describe and analyze an algorithm to compute the largest subset of L in which ***every*** pair of segments intersects.

12. ⟨⟨***S18***⟩⟩ Suppose we want to split an array $A[1..n]$ of integers into k contiguous intervals that partition the sum of the values as evenly as possible. Specifically, define the *cost* of such a partition as the maximum, over all k intervals, of the sum of the values in that interval; our goal is to minimize this cost. Describe and analyze an algorithm to compute the minimum cost of a partition of A into k intervals, given the array A and the integer k as input.

    For example, given the array $A = [1, 6, -1, 8, 0, 3, 3, 9, 8, 8, 7, 4, 9, 8, 9, 4, 8, 4, 8, 2]$ and the integer $k = 3$ as input, your algorithm should return the integer 37, which is the cost of the following partition:

$$\Big[\ \overbrace{1, 6, -1, 8, 0, 3, 3, 9, 8}^{37}\ \Big|\ \overbrace{8, 7, 4, 9, 8}^{36}\ \Big|\ \overbrace{9, 4, 8, 4, 8, 2}^{35}\ \Big]$$

    The numbers above each interval show the sum of the values in that interval.

13. ⟨⟨***S18,HW***⟩⟩ The City Council of Sham-Poobanana needs to partition Purple Street into voting districts. A total of n people live on Purple Street, at consecutive addresses $1, 2, \ldots, n$. Each voting district must be a contiguous interval of addresses $i, i+1, \ldots, j$ for some $1 \le i < j \le n$. By law, each Purple Street address must lie in exactly one district, and the number of addresses in each district must be between k and $2k$, where k is some positive integer parameter.

    Every election in Sham-Poobanana is between two rival factions: Oceania and Eurasia. A majority of the City Council are from Oceania, so they consider a district to be *good* if more than half the residents of that district voted for Oceania in the previous election. Naturally, the City Council has complete voting records for all n residents.

    For example, the figure below shows a legal partition of 22 addresses into 4 good districts and 3 bad districts, where $k = 2$ (so each district contains either 2, 3, or 4 addresses). Each `O` indicates a vote for Oceania, and each `X` indicates a vote for Eurasia.

    Describe an algorithm to find the largest possible number of *good* districts in a legal partition. Your input consists of the integer k and a boolean array $GoodVote[1..n]$ indicating which residents previously voted for Oceania (True) or Eurasia (False). You can assume that a legal partition exists. Analyze the running time of your algorithm in terms of the parameters n and k.

14. Suppose you are given an $m \times n$ bitmap, represented by an array $M[1..n, 1..n]$ of 0s and 1s. A *solid square block* in M is a subarray of the form $M[i..i+w, j..j+w]$ containing only 1-bits. Describe and analyze an algorithm to find the largest solid square block in M.

15. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, the cards are dealt face up in a long row. Each card is worth a different number of points. After all the cards are dealt, you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends.

    Having never taken an algorithms class, Elmo follows the obvious greedy strategy—when it's his turn, Elmo *always* takes the card with the higher point value. Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely *hates* it when grown-ups let him win.)

    (a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do *not* follow the same greedy strategy as Elmo.

    (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.

    (c) Five years later, thirteen-year-old Elmo has become a *much* stronger player. Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against a *perfect* opponent.

16. ***⟨⟨S16⟩⟩*** Your nephew Elmo is visiting you for Christmas, and he's brought a different card game. Like your previous game with Elmo, this game is played with a row of n cards, each labeled with an integer (which could be positive, negative, or zero). Both players can see all n card values. Otherwise, the game is almost completely different.

    On each turn, the current player must take the leftmost card. The player can either keep the card or give it to their opponent. If they keep the card, their turn ends and their opponent takes the next card; however, if they give the card to their opponent, the current player's turn continues with the next card. In short, the player that does *not* get the ith card decides who gets the $(i+1)$th card. The game ends when all cards have been played. Each player adds up their card values, and whoever has the higher total wins.

    For example, suppose the initial cards are $[3, -1, 4, 1, 5, 9]$, and Elmo plays first. Then the game might proceed as follows:

    - Elmo keeps the 3, ending his turn.
    - You give Elmo the -1.
    - You keep the 4, ending your turn.
    - Elmo gives you the 1.
    - Elmo gives you the 5.

- Elmo keeps the 9, ending his turn. All cards are gone, so the game is over.
- Your score is $1+4+5=10$ and Elmo's score is $3-1+9=11$, so Elmo wins.

Describe an algorithm to compute the highest possible score you can earn from a given row of cards, assuming Elmo plays first and plays perfectly. Your input is the array $C[1..n]$ of card values. For example, if the input is $[3,-1,4,1,5,9]$, your algorithm should return the integer 10.

17. ⟨⟨*F14*⟩⟩ The new mobile game *Candy Swap Saga XIII* involves n cute animals numbered 1 through n. Each animal holds one of three types of candy: circus peanuts, Heath bars, and Cioccolateria Gardini chocolate truffles. You also have a candy in your hand; at the start of the game, you have a circus peanut.

    To earn points, you visit each of the animals in order from 1 to n. For each animal, you can either keep the candy in your hand or exchange it with the candy the animal is holding.

    - If you swap your candy for another candy of the *same* type, you earn one point.
    - If you swap your candy for a candy of a *different* type, you lose one point. (Yes, your score can be negative.)
    - If you visit an animal and decide not to swap candy, your score does not change.

    You *must* visit the animals in order, and once you visit an animal, you can never visit it again.

    Describe and analyze an efficient algorithm to compute your maximum possible score. Your input is an array $C[1..n]$, where $C[i]$ is the type of candy that the ith animal is holding.

18. ⟨⟨*F14*⟩⟩ Farmers Boggis, Bunce, and Bean have set up an obstacle course for Mr. Fox. The course consists of a row of n booths, each with an integer painted on the front with bright red paint, which could be positive, negative, or zero. Let $A[i]$ denote the number painted on the front of the ith booth. Everyone has agreed to the following rules:

    - At each booth, Mr. Fox ***must*** say either "Ring!" or "Ding!".
    - If Mr. Fox says "Ring!" at the ith booth, he earns a reward of $A[i]$ chickens. (If $A[i]<0$, Mr. Fox pays a penalty of $-A[i]$ chickens.)
    - If Mr. Fox says "Ding!" at the ith booth, he pays a penalty of $A[i]$ chickens. (If $A[i]<0$, Mr. Fox earns a reward of $-A[i]$ chickens.)
    - Mr. Fox is forbidden to say the same word more than three times in a row. For example, if he says "Ring!" at booths 6, 7, and 8, then he *must* say "Ding!" at booth 9.
    - All accounts will be settled at the end; Mr. Fox does not actually have to carry chickens through the obstacle course.
    - If Mr. Fox violates any of the rules, or if he ends the obstacle course owing the farmers chickens, the farmers will shoot him.

Describe and analyze an algorithm to compute the largest number of chickens that Mr. Fox can earn by running the obstacle course, given the array $A[1..n]$ of booth numbers as input.

19. ⟨⟨*F19*⟩⟩ Satya is in charge of establishing a new testing center for the Standardized Awesomeness Test (SAT), and he found an old conference hall that is perfect. The conference hall has n rooms of various sizes along a single long hallway, numbered in order from 1 through n. Satya knows exactly how many students fit into each room, and he wants to use a subset of the rooms to host as many students as possible for testing.

    Unfortunately, there have been several incidents of students cheating at other testing centers by tapping secret codes through walls. To prevent this type of cheating, Satya can use two adjacent rooms only if he demolishes the wall between them. For example, if Satya wants to use rooms 1, 3, 4, 5, 7, 8, and 10, he must demolish three walls: between rooms 3 and 4, between rooms 4 and 5, and between rooms 7 and 8.

    (a) ⟨⟨*HW*⟩⟩ The city's chief architect has determined that demolishing the walls on both sides of the same room would threaten the building's structural integrity. For this reason, Satya can never host students in three consecutive rooms.

    Describe an efficient algorithm that computes the largest number of students that Satya can host for testing without using three consecutive rooms.

    The input to your algorithm is an array $S[1..n]$, where each $S[i]$ is the (non-negative integer) number of students that can fit in room i.

    (b) The city's chief architect has determined that demolishing more than k walls would threaten the structural integrity of the building.

    Describe an efficient algorithm that computes the largest number of students that Satya can host for testing without demolishing more than k walls.

    The input to your algorithm is the integer k and an array $S[1..n]$, where each $S[i]$ is the (non-negative integer) number of students that can fit in room i. Analyze your algorithm as a function of both n and k.

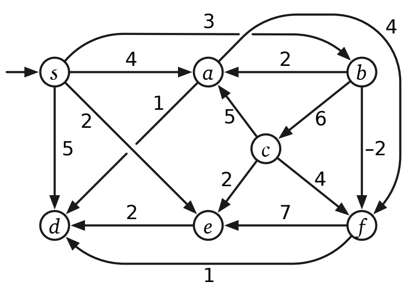
    Parts (a) and (b) appeared as complete problems in different versions of the same exam.

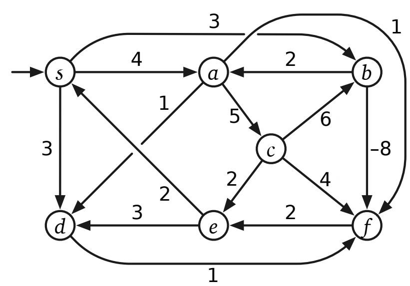
Graph Algorithms

Sanity Check

1. *⟨⟨S14, F14, F16, F19⟩⟩* Indicate the following structures in the example graphs below.

   - To indicate a subgraph (such as a path, a spanning tree, or a cycle), draw over every edge in the subgraph with a **heavy black line**. Your subgraph should be visible from across the room.
   - To indicate a subset of vertices, either draw a heavy black line around the entire subset, completely blacken the vertices in the subset, or list the vertex labels.
   - If the requested structure does not exist, just write the word NONE.





(a) A depth-first spanning tree rooted at node s.

(b) A breadth-first spanning tree rooted at node s.

(c) A shortest-path tree rooted at node s.

(d) The set of all vertices reachable from node c.

(e) The set of all vertices that can reach node c.

(f) The strong components. (Circle each strong component.)

(g) A simple cycle containing vertex s.

(h) A directed cycle with the minimum number of edges.

(i) A directed cycle with the smallest total weight.

(j) A walk from s to d with the maximum number of edges.

(k) A walk from s to d with the largest total weight.

(l) A depth-first pre-ordering of the vertices. (List the vertices in order.)

(m) A depth-first post-ordering of the vertices. (List the vertices in order.)

(n) A topological ordering of the vertices. (List the vertices in order.)

(o) A breadth-first ordering of the vertices. (List the vertices in order.)

(p) Draw the strong-component graph.

[On an actual exam, we would only ask about one graph, we would give you several copies of the graph in the answer booklet, and we would ask for only a few of these structures.]

2. ⟨⟨*F21*⟩⟩ Draw one example of each of the following graphs.

   (a) A connected undirected graph G with at most ten vertices, such that every vertex has degree at least 2, and *no* depth-first spanning tree of G is a path.

   (b) A directed acyclic graph with one source, and one sink, and more than one topological order.

   (c) A strongly connected directed graph with at least four vertices that contains no directed cycle with more than three edges.

   (d) A directed graph whose edges have distinct weights, but that has more than one shortest path from some vertex s to some other vertex t.

   (e) A strongly connected directed graph, with more than three but less than ten vertices, that contains no directed cycle with *exactly* three edges.

   (f) A strongly connected directed graph, with more than three but less than ten vertices, that contains no directed cycle with *more than* three edges.

[On an actual exam, we would only ask for a small number of these graphs.]

Reachability/Connectivity/Traversal

1. Describe and analyze algorithms for the following problems; in each problem, you are given a graph $G = (V, E)$ with unweighted edges, which may be directed or undirected. You may or may not need different algorithms for directed and undirected graphs.

   (a) Find two vertices that are (strongly) connected.

   (b) Find two vertices that are not (strongly) connected.

   (c) Find two vertices, such that neither vertex can reach the other.

   (d) Find all vertices reachable from a given vertex s.

   (e) Find all vertices that can reach a given vertex s.

   (f) Find all vertices that are strongly connected to a given vertex s.

   (g) Find a simple cycle, or correctly report that the graph has no cycles. (A simple cycle is a closed walk that visits each vertex at most once.)

   (h) Find the *shortest* simple cycle, or correctly report that the graph has no cycles.

   (i) Determine whether deleting a given vertex v would disconnect the graph.

   (j) ***⟨⟨F22⟩⟩*** Find a vertex that can reach the smallest number of other vertices.

   [On an actual exam, we would ask for at most a few of these structures, and we would specify whether the input graph is directed or undirected.]

2. ***⟨⟨F21⟩⟩*** Suppose you are given a directed graph $G = (V, E)$, each of whose vertices is either red, green, or blue. Edges in G do not have weights. G is not necessarily a dag.

   (a) Describe and analyze an algorithm to find every blue vertex b in G such that (1) at least one red vertex can reach b, and (2) b can reach at least one green vertex.

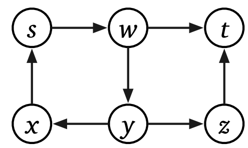
   (b) A vertex of G is *good* if it can reach vertices of all three colors in G. Describe and analyze an algorithm to find every good vertex in G.

   (c) Describe and analyze an algorithm to find a shortest path in G from any red vertex to any green vertex. (In particular, your algorithm must choose the best start and end vertices for the path.)

   (d) Describe and analyze an algorithm to find a shortest path in G that contains at least one vertex of each color. (In particular, your algorithm must choose the best start and end vertices for the path.)

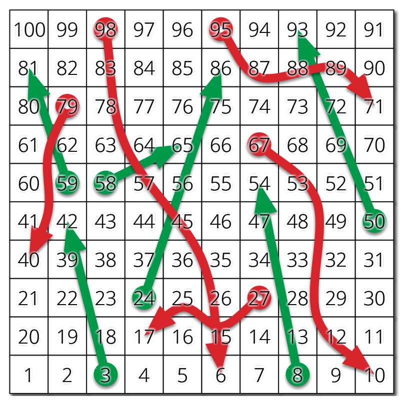
3. ***⟨⟨F14⟩⟩*** Suppose you are given a directed graph $G = (V, E)$ and two vertices s and t. Describe and analyze an algorithm to determine if there is a walk in G from s to t (possibly repeating vertices and/or edges) whose length is divisible by 3.

   For example, given the graph 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.]

4. ***Snakes and Ladders*** is a classic board game, originating in India no later than the 16th century. The board consists of an $n \times n$ grid of squares, numbered consecutively from 1 to n^2, starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares, always in different rows, are connected by either "snakes" (leading down) or "ladders" (leading up). Each square can be an endpoint of at most one snake or ladder.



A typical Snakes and Ladders board.
Upward straight arrows are ladders; downward wavy arrows are snakes.

You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to k positions, for some fixed constant k (typically 6). If the token ends the move at the *top* end of a snake, you ***must*** slide the token down to the bottom of that snake. If the token ends the move at the *bottom* end of a ladder, you ***may*** move the token up to the top of that ladder.

Describe and analyze an algorithm to compute the smallest number of moves required for the token to reach the last square of the grid.

5. Let G be a connected undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of G, and we want to move the coins so that they lie on the same vertex using as few moves as possible. At every step, each coin *must* move to an adjacent vertex.

   (a) Describe and analyze an algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of the graph $G = (V, E)$ and two vertices $u, v \in V$ (which may or may not be distinct).

(b) Now suppose there are *forty-two* coins. Describe and analyze an algorithm to determine whether it is possible to move all 42 coins to the same vertex. Again, *every* coin must move at *every* step. The input to your algorithm consists of the graph $G = (V, E)$ and an array of 42 vertices (which may or may not be distinct). For full credit, your algorithm should run in $O(V + E)$ time.

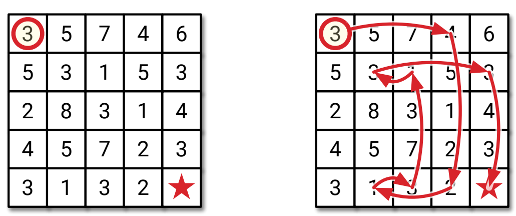
6. A graph (V, E) is bipartite if the vertices V can be partitioned into two subsets L and R, such that every edge has one vertex in L and the other in R.

   (a) Prove that every tree is a bipartite graph.

   (b) Describe and analyze an efficient algorithm that determines whether a given undirected graph is bipartite.

7. ***⟨⟨F14, S18, Lab⟩⟩*** A ***number maze*** is an $n \times n$ grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner. On each turn, you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right. However, you are never allowed to move the token off the edge of the board.

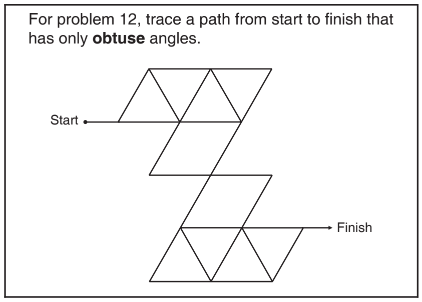
   Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution. For example, given the maze shown below, your algorithm would return the number 8.

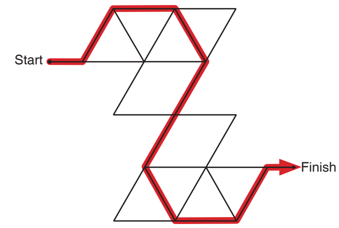


A 5 × 5 number maze that can be solved in eight moves.

8. ***⟨⟨F16⟩⟩*** The following puzzle appeared in my daughter's math workbook several years ago.[1] (I've put the solution on the right so you don't waste time solving it during the exam.)

[1]Jason Batterson and Shannon Rogers, *Beast Academy Math: Practice 3A*, 2012. See https://www.beastacademy.com/resources/printables.php for more examples.



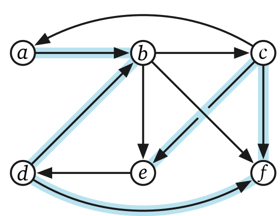


Describe and analyze an algorithm to solve arbitrary obtuse-angle mazes.

You are given a connected undirected graph G, whose vertices are points in the plane and whose edges are line segments. Edges do not intersect, except at their endpoints. For example, a drawing of the letter X would have five vertices and four edges; the maze above has 17 vertices and 26 edges. You are also given two vertices Start and Finish.

Your algorithm should return TRUE if G contains a walk from Start to Finish that has only obtuse angles, and FALSE otherwise. Formally, a walk through G is valid if $\pi/2 < \angle uvw \leq \pi$ for every pair of consecutive edges $u\to v\to w$ in the walk. Assume you have a subroutine that can determine whether the angle between any two segments is acute, right, obtuse, or straight in $O(1)$ time.

9. A *zigzag walk* in in a directed graph G is a sequence of vertices connected by edges in G, but the edges alternately point forward and backward along the sequence. For example, the following graph contains the zigzag walk $a\to b\leftarrow d\to f\leftarrow c\to e$:



If the edges of G have weights, the *length* of a zigzag walk is the sum of the weights of its edges (both forward edges and backward edges).

(a) Suppose you are given a directed graph G with non-negatively weighted edges, along with two vertices s and t. Describe and analyze an algorithm to find the shortest zigzag walk from s to t in G.

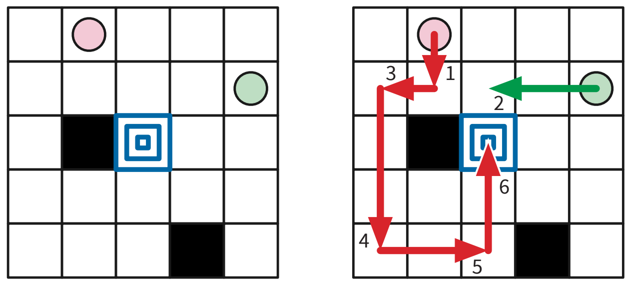
(b) Give an example where the shortest zigzag walk from s to t must visit a vertex more than once.

*(c) ***Prove*** that if all edge weights are non-negative, the shortest zigzag walk never traverses the same edge in both directions.

10. The famous puzzle-maker Kaniel the Dane invented a solitaire game played with two tokens on an $n \times n$ square grid. Some squares of the grid are marked as *obstacles*, and one

grid square is marked as the *target*. In each turn, the player must move one of the tokens from is current position *as far as possible* upward, downward, right, or left, stopping just before the token hits (1) the edge of the board, (2) an obstacle square, or (3) the other token. The goal is to move either of the tokens onto the target square.

For example, in the instance below, we move the red token down until it hits the obstacle, then move the green token left until it hits the red token, and then move the red token left, down, right, and up. In the last move, the red token stops at the target *because* the green token is on the next square above.



An instance of the Kaniel Dane puzzle that can be solved in six moves.
Circles indicate the initial token positions; black squares are obstacles; the center square is the target.

Describe and analyze an algorithm to determine whether an instance of this puzzle is solvable. Your input consist of the integer n, a list of obstacle locations, the target location, and the initial locations of the tokens. The output of your algorithm is a single boolean: TRUE if the given puzzle is solvable and FALSE otherwise. The running time of your algorithm should be a small polynomial in n. *[Hint: Don't forget about the time required to construct the graph!]*

Depth-First Search, Dags, Strong Connectivity

1. ⟨⟨*Lab*⟩⟩ Inspired by an earlier question, you decided to organize a Snakes and Ladders competition with n participants. In this competition, each game of Snakes and Ladders involves three players. After the game is finished, they are ranked first, second and third. Each player may be involved in any (non-negative) number of games, and the number needs not be equal among players.

   At the end of the competition, m games have been played. You realized that you had forgotten to implement a proper rating system, and therefore decided to produce the overall ranking of all n players as you see fit. However, to avoid being too suspicious, if player A ranked better than player B in any game, then A must rank better than B in the overall ranking.

   You are given the list of players involved and the ranking in each of the m games. Describe and analyze an algorithm to produce an overall ranking of the n players that satisfies the condition, or correctly reports that it is impossible.

2. Let G be a directed acyclic graph with a unique source s and a unique sink t.

   (a) A *Hamiltonian path* in G is a directed path in G that contains every vertex in G. Describe an algorithm to determine whether G has a Hamiltonian path.

   (b) Suppose the vertices of G have weights. Describe an efficient algorithm to find the path from s to t with maximum total weight.

   (c) Suppose we are also given an integer ℓ. Describe an efficient algorithm to find the maximum-weight path from s to t, such that the path contains at most ℓ edges. (Assume there is at least one such path.)

   (d) Suppose several vertices in G are marked *essential*, and we are given an integer k. Design an efficient algorithm to determine whether there is a path from s to t that passes through at least k essential vertices.

   (e) Suppose the vertices of G have integer labels, where $label(s) = -\infty$ and $label(t) = \infty$. Describe an algorithm to find the path from s to t with the maximum number of edges, such that the vertex labels define an increasing sequence.

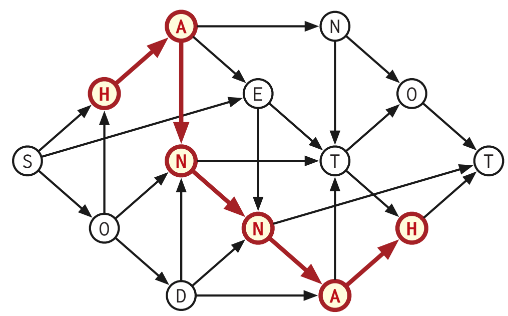
   (f) Describe an algorithm to compute the number of distinct paths from s to t in G. (Assume that you can add arbitrarily large integers in $O(1)$ time.)

3. Suppose you are given a directed acyclic graph G whose nodes represent jobs and whose edges represent *precedence constraints*: Each edge $u \rightarrow v$ indicates that job u must be completed before job v begins. Each node v stores a non-negative number $v.duration$ indicating the time required to execute job v. All jobs are executed in parallel; any job can start or end while any number of other jobs are executing, provided all the precedence constraints are satisfied. You'd like to get all these jobs done as quickly as possible.

   Describe an algorithm to determine, for every vertex v in G, the earliest time that job v can ***begin***, assuming the first job starts at time 0 and no precedence constraints are violated. Your algorithm should record the answer for each vertex v in a new field $v.earliest$.

4. Let G be a directed acyclic graph whose vertices have labels from some fixed alphabet. Any directed path in G has a label, which is a string obtained by concatenating the labels of its vertices. Recall that a *palindrome* is a string that is equal to its reversal.

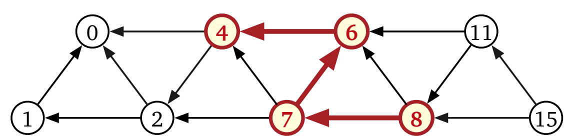
   Describe and analyze an algorithm to find the length of the longest palindrome that is the label of a path in G. For example, given the dag below, your algorithm should return the integer 6, which is the length of the palindrome HANNAH.



5. ⟨⟨*S18*⟩⟩ Let G be a ***directed*** graph, where every vertex v has an associated height $h(v)$, and for every edge $u \to v$ we have the inequality $h(u) > h(v)$. Assume all heights are distinct. The *span* of a path from u to v is the height difference $h(u) - h(v)$.

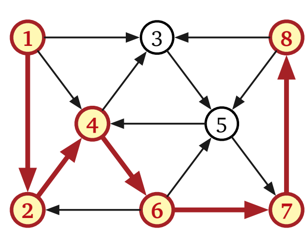
   Describe and analyze an algorithm to find the ***minimum span*** of a path in G with ***at least*** k edges. Your input consists of the graph G, the vertex heights $h(\cdot)$, and the integer k. Report the running time of your algorithm as a function of V, E, and k.

   For example, given the following labeled graph and the integer $k = 3$ as input, your algorithm should return the integer 4, which is the span of the path $8 \to 7 \to 6 \to 4$.



6. ⟨⟨*S18*⟩⟩ Let G be an arbitrary (*not* necessarily acyclic) directed graph in which every vertex v has an integer label $\ell(v)$. Describe an algorithm to find the longest directed path in G whose vertex labels define an increasing sequence. Assume all labels are distinct.

   For example, given the following graph as input, your algorithm should return the integer 5, which is the length of the increasing path $1 \to 2 \to 4 \to 6 \to 7 \to 8$.



7. ⟨⟨***F16***⟩⟩ Suppose you have a collection of n lockboxes and m gold keys. Each key unlocks *at most* one box; however, each box might be unlocked by one key, by multiple keys, or by no keys at all. There are only two ways to open each box once it is locked: Unlock it properly (which requires having a matching key in your hand), or smash it to bits with a hammer.

   Your baby brother, who loves playing with shiny objects, has somehow managed to lock all your keys inside the boxes! Luckily, your home security system recorded everything, so you know exactly which keys (if any) are inside each box. You need to get all the keys back out of the boxes, because they are made of gold. Clearly you have to smash at least one box.

   (a) Your baby brother has found the hammer and is eagerly eyeing one of the boxes. Describe and analyze an algorithm to determine if smashing the box your brother has chosen would allow you to retrieve all m keys.

   (b) Describe and analyze an algorithm to compute the minimum number of boxes that must be smashed to retrieve all the keys.

8. ⟨⟨***F21***⟩⟩ Suppose you are given a height map of a mountain, in the form of an $n \times n$ grid of evenly spaced points, each labeled with an elevation value. You can safely hike directly from any point to any neighbor immediately north, south, east, or west, but only if the elevations of those two points differ by at most Δ. (The value of Δ depends on your hiking experience and your physical condition.)

   Describe and analyze an algorithm to determine the longest hike from some point s to some other point t, where the hike consists of an uphill climb (where elevations must increase at each step) followed by a downhill climb (where elevations must decrease at each step). Your input consists of an array $Elevation[1..n, 1..n]$ of elevation values, the starting point s, the target point t, and the parameter Δ.

9. ⟨⟨***HW***⟩⟩ Ronnie and Hyde are a professional robber duo who plan to rob exactly *one* house in the Leverwood neighborhood of Sham-Boobanana. They have managed to obtain a map of the neighborhood in the form of a directed graph G, whose vertices represent houses and whose edges represent one-way streets.

   - One vertex s represents the house that Ronnie and Hyde plan to rob.
   - A set X of special vertices designate eXits from the neighborhood.
   - For each directed edge $u{\rightarrow}v$, Ronnie can drive directly from house u to house v in $w(u{\rightarrow}v)$ minutes.
   - Driving backwards along any street immediately triggers traffic drones.

   Describe and analyze an algorithm to compute the shortest time needed to exit the neighborhood, starting at house s. The input to your algorithm is the directed graph $G = (V, E)$, with clearly marked subset of exit vertices $X \subseteq V$, and non-negative weights $w(u{\rightarrow}v)$ for every edge $u{\rightarrow}v$.

10. ⟨⟨*F21*⟩⟩ Aladdin and Badroulbadour are playing a cooperative game. Each player has an array of positive integers, arranged in a row of squares from left to right. Each player has a token, which starts at the leftmost square of their row; their goal is to move *both* tokens to the rightmost squares.

    On each turn, *both* players move their tokens *in the same direction*, either left or right. The distance each token travels is equal to the number under that token at the beginning of the turn. For example, if a token starts on a square labeled 5, then it moves either five squares to the right or five squares to the left. If *either* token moves past either end of its row, then both players immediately lose.

    For example, if Aladdin and Badroulbadour are given the arrays

A:	7	5	4	1	2	3	3	2	3	1	4	2
B:	5	1	2	4	7	3	5	2	4	6	3	1

    they can win the game by moving right, left, left, right, right, left, right. On the other hand, if they are given the arrays

A:	2	3	5	1	3
B:	3	4	1	2	1

    they cannot win the game. (The first move must be to the right; then Aladdin's token moves out of bounds on the second turn.)

    Describe and analyze an algorithm to determine whether Aladdin and Badroulbadour can solve their puzzle, given the input arrays $A[1..n]$ and $B[1..n]$.

11. ⟨⟨*S23*⟩⟩ Let $G = (V, E)$ be a directed graph, where every edge $e \in E$ has a non-negative *width* $w(e)$. The *bottleneck width* of a directed cycle C in G is the *minimum* width among all edges in C.

    (a) Describe and analyze an algorithm that, given a graph G, a vertex s, and a real number ω, determines whether there is a cycle in G containing s with bottleneck width at least ω.

    (b) Describe and analyze an algorithm that, given a graph G and a vertex s, either finds the minimum value ω such that there is a cycle in G containing s with bottleneck width ω, or reports correctly that no cycle in G contains s. (You can skip part (a) if you can answer this part directly.)

Shortest Paths

1. Suppose you are given a directed graph G with weighted edges and a vertex s of G.

   (a) ⟨⟨***F14***⟩⟩ Suppose every vertex $v \neq s$ stores a pointer $pred(v)$ to another vertex in G. Describe and analyze an algorithm to determine whether these predecessor pointers correctly define a single-source shortest path tree rooted at s.

   (b) Suppose every vertex v stores a finite real value $dist(v)$. (In particular, $dist(v)$ is never equal to ∞ or $-\infty$.) Describe and analyze an algorithm to determine whether these real values are correct shortest-path distances from s.

   Do ***not*** assume that G has no negative cycles.

2. ⟨⟨***F14***⟩⟩ Suppose we are given an undirected graph G in which every *vertex* has a positive weight.

   (a) Describe and analyze an algorithm to find a *spanning tree* of G with minimum total weight. (The total weight of a spanning tree is the sum of the weights of its vertices.)

   (b) Describe and analyze an algorithm to find a *path* in G from one given vertex s to another given vertex t with minimum total weight. (The total weight of a path is the sum of the weights of its vertices.)

3. ⟨⟨***S14, S18, Lab***⟩⟩ You just discovered your best friend from elementary school on Twitbook. You both want to meet as soon as possible, but you live in two different cites that are far apart. To minimize travel time, you agree to meet at an intermediate city, and then you simultaneously hop in your cars and start driving toward each other. But where *exactly* should you meet?

   You are given a weighted graph $G = (V, E)$, where the vertices V represent cities and the edges E represent roads that directly connect cities. Each edge e has a weight $w(e)$ equal to the time required to travel between the two cities. You are also given a vertex p, representing your starting location, and a vertex q, representing your friend's starting location.

   Describe and analyze an algorithm to find the target vertex t that allows you and your friend to meet as quickly as possible.

4. ⟨⟨***F16***⟩⟩ There are n galaxies connected by m intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. Also, each teleport-way uv has an associated cost of $c(uv)$ galactic credits, for some positive integer $c(uv)$. The same teleport-way can be used multiple times in either direction, but the same toll must be paid every time it is used.

   Judy wants to travel from galaxy s to galaxy t, but teleportation is rather unpleasant, so she wants to minimize the number of times she has to teleport. However, she also wants the total cost to be a multiple of 10 galactic credits, because carrying small change is annoying.

Describe and analyze an algorithm to compute the minimum number of times Judy must teleport to travel from galaxy s to galaxy t so that the total cost of all teleports is an integer multiple of 10 galactic credits. Your input is a graph $G = (V, E)$ whose vertices are galaxies and whose edges are teleport-ways; every edge uv in G stores the corresponding cost $c(uv)$.

*[Hint: This is **not** the same Intergalactic Judy problem that you saw in lab.]*

5. ⟨⟨*Lab*⟩⟩ A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has non-negative weight.

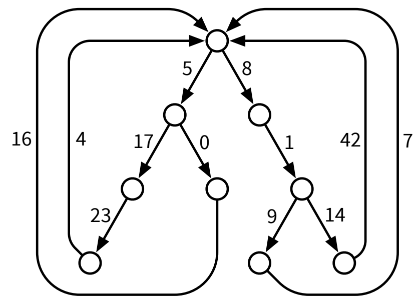


Figure 1. A looped tree.

(a) How much time would Dijkstra's algorithm require to compute the shortest path from one vertex s to another vertex t in a looped tree with n nodes?

(b) Describe and analyze a faster algorithm.

6. ⟨⟨*F17, Lab*⟩⟩ Suppose you are given a directed graph G with weighted edges, where *exactly one* edge has negative weight and all other edge weights are positive, along with two vertices s and t. Describe and analyze an algorithm that either computes a shortest path in G from s to t, or reports correctly that the G contains a negative cycle. (As always, faster algorithms are worth more points.)

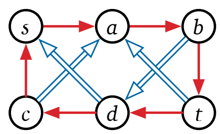
7. ⟨⟨*HW*⟩⟩ When there is more than one shortest path from one node s to another node t, it is often convenient to choose a shortest path with the fewest edges; call this the ***best*** path from s to t. Suppose we are given a directed graph G with positive edge weights and a source vertex s in G. Describe and analyze an algorithm to compute best paths in G from s to every other vertex.

8. After graduating you accept a job with Aerophobes-Я-Us, the leading traveling agency for people who hate to fly. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying (and by extension, airports), so any trip you plan needs to be as short as possible. You know all the departure and arrival times of all the flights on the planet.

Suppose one of your customers wants to fly from city X to city Y. Describe an algorithm to find a sequence of flights that minimizes the *total time in transit*—the length of time from the initial departure to the final arrival, including time at intermediate airports waiting for connecting flights.

9. ⟨⟨*S18*⟩⟩ Suppose you are given a directed graph G where some edges are red and the remaining edges are blue. Describe an algorithm to find the shortest walk in G from one vertex s to another vertex t in which no three consecutive edges have the same color. That is, if the walk contains two red edges in a row, the next edge must be blue, and if the walk contains two blue edges in a row, the next edge must be red.

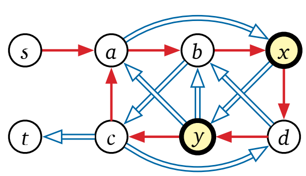
   For example, if you are given the graph below (where single arrows are red and double arrows are blue), your algorithm should return the integer 7, because the shortest legal walk from s to t is $s{\to}a{\to}b{\Rightarrow}d{\to}c{\Rightarrow}a{\to}b{\to}c$.



10. ⟨⟨*S18*⟩⟩ Suppose you are given a directed graph G in which every edge is either red or blue, and a subset of the vertices are marked as *special*. A walk in G is *legal* if color changes happen only at special vertices. That is, for any two consecutive edges $u{\to}v{\to}w$ in a legal walk, if the edges $u{\to}v$ and $v{\to}w$ have different colors, the intermediate vertex v must be special.

    Describe and analyze an algorithm that either returns the length of the shortest legal walk from vertex s to vertex t, or correctly reports that no such walk exists.[2]

    For example, if you are given the following graph below as input (where single arrows are red, double arrows are blue), with special vertices x and y, your algorithm should return the integer 8, which is the length of the shortest legal walk $s{\to}x{\to}a{\to}b{\to}x{\Rightarrow}y{\Rightarrow}b{\Rightarrow}c{\Rightarrow}t$. The shorter walk $s{\to}a{\to}b{\Rightarrow}c{\Rightarrow}t$ is not legal, because vertex b is not special.

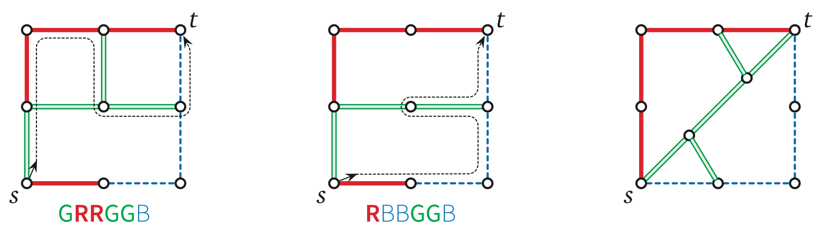


11. ⟨⟨*S18*⟩⟩ Let G be a directed graph with weighted edges, in which every vertex is colored either red, green, or blue. Describe and analyze an algorithm to compute the length of the shortest walk in G that starts at a red vertex, then visits any number of vertices of any color, then visits a green vertex, then visits any number of vertices of any color, then visits a blue vertex, then visits any number of vertices of any color, and finally ends at a red vertex. Assume all edge weights are positive.

[2] If you've read China Miéville's excellent novel *The City & the City*, this problem should look familiar. If you haven't read *The City & the City*, I can't tell you why this problem should look familiar without spoiling the book.

12. ⟨⟨*F19*⟩⟩ Suppose you are given an undirected graph G in which every edge is either red, green, or blue, along with two vertices s and t. Call a walk from s to t *awesome* if the walk does not contain three consecutive edges with the same color.

    Describe and analyze an algorithm to find the length of the shortest awesome walk from s to t. For example, given either the left or middle input below, your algorithm should return the integer 6, and given the input on the right, your algorithm should return ∞.



13. ⟨⟨*F19*⟩⟩ During her walk to work every morning, Rachel likes to buy a cappuccino at a local coffee shop, and a croissant at a local bakery. Her home town has *lots* of coffee shops and lots of bakeries, but strangely never in the same building. Punctuality is not Rachel's strongest trait, so to avoid losing her job, she wants to follow the shortest possible route.

    Rachel has a map of her home town in the form of an undirected graph G, whose vertices represent intersections and whose edges represent roads between them. A subset of the vertices are marked as bakeries; another disjoint subset of vertices are marked as coffee shops. The graph has two special nodes s and t, which represent Rachel's home and work, respectively.

    Describe an algorithm that computes the shortest path in G from s to t that visits both a bakery and a coffee shop, or correctly reports that no such path exists.

14. ⟨⟨*F19*⟩⟩ As the days get shorter in winter, Eggsy Hutmacher is increasingly worried about his walk home from work. The city has recently been invaded by the notorious Antimilliner gang, whose members hang out on dark street corners and steal hats from unwary passers-by, and a gentleman is simply *not* seen out in public without a hat. The city council is slowly installing street lamps at intersections to deter the Antimilliners, whose uncovered faces can be easily identified in the light. Eggsy keeps k extra hats in his briefcase in case of theft or other millinery emergencies.

    Eggsy has a map of the city in the form of an undirected graph G, whose vertices represent intersections and whose edges represent streets between them. A subset of the vertices are marked to indicate that the corresponding intersections are lit. Every edge e has a non-negative length $\ell(e)$. The graph has two special nodes s and t, which represent Eggsy's work and home, respectively.

    Describe an algorithm that computes the shortest path in G from s to t that visits at most k unlit vertices, or correctly reports that no such path exists. Analyze your algorithm as a function of the parameters V, E, and k.

15. *⟨⟨F19, HW⟩⟩* You and your friends are planning a hiking trip in Jellystone National Park over winter break. You have a map of the park's trails that lists all the scenic views in the park but also warns that certain trail segments have a high risk of bear encounters. To make the hike worthwhile, you want to see at least three scenic views. You also don't want to get eaten by a bear, so you are willing to hike at most one high-bear-risk segment. Because the trails are narrow, each trail segment allows traffic in only one direction.

    Your friend has converted the map into a directed graph $G = (V, E)$, where V is the set of intersections and E is the set of trail segments. A subset S of the edges are marked as *Scenic*; another subset B of the edges are marked as *high-Bear-risk*. You may assume that $S \cap B = \varnothing$. Each segment $e \in E$ is also labeled with a positive length $\ell(e)$ in miles. Your campsite appears on the map as a particular vertex $s \in V$, and the visitor center is another vertex $t \in V$.

    Describe and analyze an algorithm to compute the shortest hike from your campsite s to the visitor center t that includes *at least* three scenic trail segments and *at most* one high-bear-risk trail segment. You may assume such a hike exists.