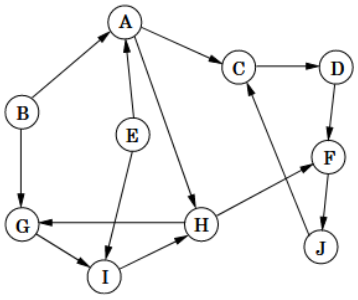
0.1. In each of the following situations, indicate whether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$).

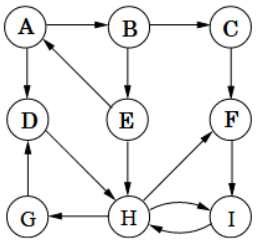
	$f(n)$	$g(n)$
(a)	$n - 100$	$n - 200$
(b)	$n^{1/2}$	$n^{2/3}$
(c)	$100n + \log n$	$n + (\log n)^2$
(d)	$n \log n$	$10n \log 10n$
(e)	$\log 2n$	$\log 3n$
(f)	$10 \log n$	$\log(n^2)$
(g)	$n^{1.01}$	$n \log^2 n$
(h)	$n^2 / \log n$	$n(\log n)^2$
(i)	$n^{0.1}$	$(\log n)^{10}$
(j)	$(\log n)^{\log n}$	$n / \log n$
(k)	$\sqrt{n}$	$(\log n)^3$
(l)	$n^{1/2}$	$5^{\log_2 n}$
(m)	$n2^n$	3^n
(n)	2^n	2^{n+1}
(o)	$n!$	2^n
(p)	$(\log n)^{\log n}$	$2^{(\log_2 n)^2}$
(q)	$\sum_{i=1}^{n} i^k$	n^{k+1}

3.4. Run the strongly connected components algorithm on the following directed graphs G. When doing DFS on G^R: whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

(i)



(ii)



In each case answer the following questions.

(a) In what order are the strongly connected components (SCCs) found?

(b) Which are source SCCs and which are sink SCCs?

(c) Draw the "metagraph" (each meta-node is an SCC of G).

(d) What is the minimum number of edges you must add to this graph to make it strongly connected?

3.8. *Pouring water.* We have three containers whose sizes are 10 pints, 7 pints, and 4 pints, respectively. The 7-pint and 4-pint containers start out full of water, but the 10-pint container is initially empty. We are allowed one type of operation: pouring the contents of one container into another, stopping only when the source container is empty or the destination container is full. We want to know if there is a sequence of pourings that leaves exactly 2 pints in the 7- or 4-pint container.

(a) Model this as a graph problem: give a precise definition of the graph involved and state the specific question about this graph that needs to be answered.

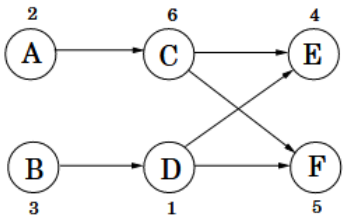
(b) What algorithm should be applied to solve the problem?

(c) Find the answer by applying the algorithm.

3.25. You are given a directed graph in which each node $u \in V$ has an associated *price* p_u which is a positive integer. Define the array `cost` as follows: for each $u \in V$,

$$\texttt{cost}[u] = \text{price of the cheapest node reachable from } u \text{ (including } u \text{ itself).}$$

For instance, in the graph below (with prices shown for each vertex), the `cost` values of the nodes A, B, C, D, E, F are $2, 1, 4, 1, 4, 5$, respectively.



Your goal is to design an algorithm that fills in the *entire* `cost` array (i.e., for all vertices).

(a) Give a linear-time algorithm that works for directed *acyclic* graphs. (*Hint:* Handle the vertices in a particular *order*.)

(b) Extend this to a linear-time algorithm that works for all directed graphs. (*Hint:* Recall the "two-tiered" structure of directed graphs.)

If graph G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree.

If G has a cycle with a unique heaviest edge e, then e cannot be part of any MST.

If e is part of some MST of G, then it must be a lightest edge across some cut of G.

If G has a cycle with a unique lightest edge e, then e must be part of every MST.

2.4. Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n - 1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

2.23. An array $A[1 \ldots n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as GIF files, say.) However you *can* answer questions of the form: "is $A[i] = A[j]$?" in constant time.

(a) Show how to solve this problem in $O(n \log n)$ time. (*Hint:* Split the array A into two arrays A_1 and A_2 of half the size. Does knowing the majority elements of A_1 and A_2 help you figure out the majority element of A? If so, you can use a divide-and-conquer approach.)

(b) Can you give a linear-time algorithm? (*Hint:* Here's another divide-and-conquer approach:

- Pair up the elements of A arbitrarily, to get $n/2$ pairs
- Look at each pair: if the two elements are different, discard both of them; if they are the same, keep just one of them

Show that after this procedure there are at most $n/2$ elements left, and that they have a majority element if and only if A does.)

Longest palindromic subsequence A palindromic sequence reads the same from left to right as from right to left. More precisely, $B[1, ..., t]$ is palindromic if and only if for all $1 \leq i \leq t$, $B[i] = B[t + 1 - i]$. For example $[3, 5, 5, 2, 5, 5, 3]$ and $[3, 5, 5, 3]$ are palindromic. Given an input array, you are asked to design a dynamic programming algorithm to compute one of its longest palindromic subsequences.

- Input: an unordered array $A[1, ..., n]$
- Output: one of its longest palindromic subsequences.
- For example, the output of $[9, 14, 9, 5, 10, 6, 15, 6, 13, 9]$ should be $[9, 6, 15, 6, 9]$

Use Dynamic Programming to compute the answers to the following subproblems: for each pair of $1 \leq i \leq j \leq n$, let $S[i][j]$ denote the longest palindromic subsequence of $A[i, ..., j]$. Then the solution we want is simply $S[1][n]$.
Hints:

- When $j \geq i + 2$, $S[i][j]$ is closely related to the following three cases
  1. $S[i + 1][j]$
  2. $S[i][j - 1]$
  3. $A[i] + S[i + 1][j - 1] + A[j]$, if $A[i] = A[j]$
- What happens for $S[i][i]$ and $S[i][i + 1]$?

Try your solution on $[7, 2, 4, 6, 9, 11, 2, 6, 10, 6, 15, 6, 14, 2, 7, 5, 13, 9, 12, 15]$. Next, generate an array of size 1000 with each entry being a random integer between 1 and 100. About how long are the longest palindromic subsequences in this case?

Given two strings $x = x_1x_2 \cdots x_n$ and $y = y_1y_2 \cdots y_m$, we wish to find the length of their *longest common substring*, that is, the largest k for which there are indices i and j with $x_ix_{i+1} \cdots x_{i+k-1} = y_jy_{j+1} \cdots y_{j+k-1}$. Show how to do this in time $O(mn)$.

Give an $O(nt)$ algorithm for the following task.

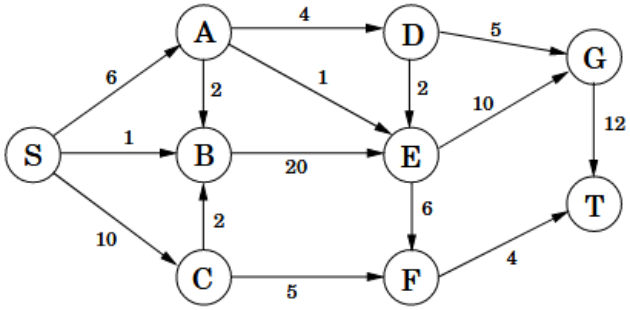
Input: **A list of n positive integers $a_1, a_2, \ldots, a_n$; a positive integer t.**
Question: **Does some subset of the a_i's add up to t? (You can use each a_i at most once.)**

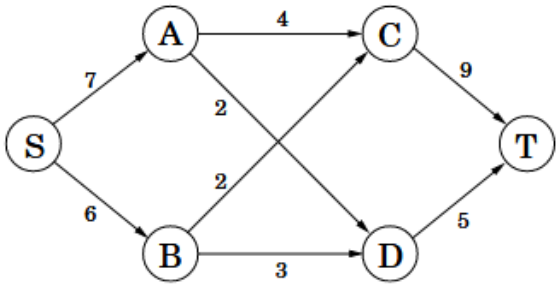
(*Hint:* Look at subproblems of the form "does a subset of $\{a_1, a_2, \ldots, a_i\}$ add up to s?")

Duckwheat is produced in Kansas and Mexico and consumed in New York and California. Kansas produces 15 shnupells of duckwheat and Mexico 8. Meanwhile, New York consumes 10 shnupells and California 13. The transportation costs per shnupell are $4 from Mexico to New York, $1 from Mexico to California, $2 from Kansas to New York, and $3 and from Kansas to California.

Write a linear program that decides the amounts of duckwheat (in shnupells and fractions of a shnupell) to be transported from each producer to each consumer, so as to minimize the overall transportation cost.

For the following network, with edge capacities as shown, find the maximum flow from S to T, along with a matching cut.





Find the maximum flow f and a minimum cut.

In the HITTING SET problem, we are given a family of sets $\{S_1, S_2, \ldots, S_n\}$ and a budget b, and we wish to find a set H of size $\leq b$ which intersects every S_i, if such an H exists. In other words, we want $H \cap S_i \neq \emptyset$ for all i.

Show that HITTING SET is **NP**-complete.

Determine which of the following problems are **NP**-complete and which are solvable in polynomial time. In each problem you are given an undirected graph $G = (V, E)$, along with:

(a) A set of nodes $L \subseteq V$, and you must find a spanning tree such that its set of leaves includes the set L.

(b) A set of nodes $L \subseteq V$, and you must find a spanning tree such that its set of leaves is precisely the set L.

(c) A set of nodes $L \subseteq V$, and you must find a spanning tree such that its set of leaves is included in the set L.

Prove that the following problem is **NP**-complete: given an undirected graph $G = (V, E)$ and an integer k, return a clique of size k *as well as* an independent set of size k, provided both exist.