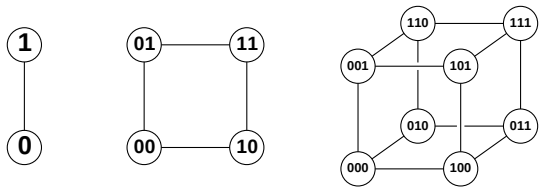
Write your answers in the separate answer booklet.
This is a 180-minute exam. The clock started when you got the questions.

1. The *d-dimensional hypercube* is the graph defined as follows. There are 2^d vertices, each labeled with a different string of d bits. Two vertices are joined by an edge if their labels differ in exactly one bit.

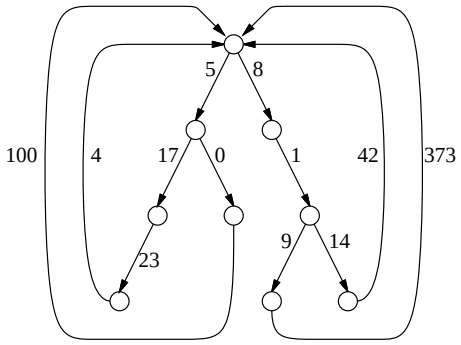


The 1-dimensional, 2-dimensional, and 3-dimensional hypercubes.

   (a) **[8 pts]** Recall that a Hamiltonian cycle passes through every vertex in a graph exactly once. ***Prove*** that for all $d \geq 2$, the d-dimensional hypercube has a Hamiltonian cycle.

   (b) **[2 pts]** Which hypercubes have an Eulerian circuit (a closed walk that visits every edge exactly once)? *[Hint: This is very easy.]*

2. 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 a non-negative weight. The number of nodes in the graph is n.



   (a) How long would it take Dijkstra's algorithm to compute the shortest path between two vertices u and v in a looped tree?

   (b) Describe and analyze a faster algorithm.

3. Prove that $(x + y)^p \equiv x^p + y^p \pmod{p}$ for any prime number p.

4. A *palindrome* is a string that reads the same forwards and backwards, like X, 373, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. *Any* string can be written as a sequence of palindromes. For example, the string bubbaseesabanana ('Bubba sees a banana.') can be decomposed in several ways; for example:

$$\begin{array}{c} \mathsf{bub} + \mathsf{baseesab} + \mathsf{anana} \\ \mathsf{b} + \mathsf{u} + \mathsf{bb} + \mathsf{a} + \mathsf{sees} + \mathsf{aba} + \mathsf{nan} + \mathsf{a} \\ \mathsf{b} + \mathsf{u} + \mathsf{bb} + \mathsf{a} + \mathsf{sees} + \mathsf{a} + \mathsf{b} + \mathsf{anana} \\ \mathsf{b} + \mathsf{u} + \mathsf{b} + \mathsf{b} + \mathsf{a} + \mathsf{s} + \mathsf{e} + \mathsf{e} + \mathsf{s} + \mathsf{a} + \mathsf{b} + \mathsf{a} + \mathsf{n} + \mathsf{a} + \mathsf{n} + \mathsf{a} \end{array}$$

   Describe an efficient algorithm to find the *minimum* number of palindromes that make up a given input string. For example, given the input string bubbaseesabanana, your algorithm would return the number 3.

5. Your boss wants you to find a *perfect* hash function for mapping a known set of n items into a table of size m. A hash function is perfect if there are *no* collisions; each of the n items is mapped to a different slot in the hash table. Of course, this requires that $m \geq n$.

   After cursing your 373 instructor for not teaching you about perfect hashing, you decide to try something simple: repeatedly pick *random* hash functions until you find one that happens to be perfect. A random hash function h satisfies two properties:

   - $\Pr\big[h(x) = h(y)\big] = \frac{1}{m}$ for any pair of items $x \neq y$.
   - $\Pr\big[h(x) = i\big] = \frac{1}{m}$ for any item x and any integer $1 \leq i \leq m$.

   (a) **[2 pts]** Suppose you pick a random hash function h. What is the *exact* expected number of collisions, as a function of n (the number of items) and m (the size of the table)? Don't worry about how to *resolve* collisions; just count them.

   (b) **[2 pts]** What is the *exact* probability that a random hash function is perfect?

   (c) **[2 pts]** What is the *exact* expected number of different random hash functions you have to test before you find a perfect hash function?

   (d) **[2 pts]** What is the *exact* probability that none of the first N random hash functions you try is perfect?

   (e) **[2 pts]** How many random hash functions do you have to test to find a perfect hash function *with high probability*?

   To get full credit for parts (a)–(d), give *exact* closed-form solutions; correct $\Theta(\cdot)$ bounds are worth significant partial credit. Part (e) requires only a $\Theta(\cdot)$ bound; an exact answer is worth extra credit.

6. Your friend Toidi is planning to hold a Christmas party. He wants to take a picture of all the participants, including himself, but he is quite shy and thus cannot take a picture of a person whom he does not know very well. Since he has only shy friends[1], everyone at the party is also shy. After thinking hard for a long time, he came up with a seemingly good idea:

   - Toidi brings a disposable camera to the party.
   - Anyone holding the camera can take a picture of someone they know very well, and then pass the camera to that person.
   - In order not to waste any film, every person must have their picture taken exactly once.

   Although there can be some people Toidi does not know very well, he knows completely who knows whom well. Thus, *in principle*, given a list of all the participants, he can determine whether it is possible to take all the pictures using this idea. But how quickly?

   Either describe an efficient algorithm to solve Toidi's problem, or show that the problem is NP-complete.

7. The recursion fairy's cousin, the reduction genie, shows up one day with a magical gift for you: a box that can solve the NP-complete PARTITION problem in constant time! Given a set of positive integers as input, the magic box can tell you in constant time it can be split into two subsets whose total weights are equal.

   For example, given the set $\{1, 4, 5, 7, 9\}$ as input, the magic box cheerily yells "YES!", because that set can be split into $\{1, 5, 7\}$ and $\{4, 9\}$, which both add up to 13. Given the set $\{1, 4, 5, 7, 8\}$, however, the magic box mutters a sad "Sorry, no."

   The magic box does not tell you *how* to partition the set, only whether or not it can be done. Describe an algorithm to actually split a set of numbers into two subsets whose sums are equal, **in polynomial time**, using this magic box.[2]

[1] Except you, of course. Unfortunately, you can't go to the party because you're taking a final exam. Sorry!

[2] Your solution to problem 4 in homework 1 does *not* solve this problem in polynomial time.