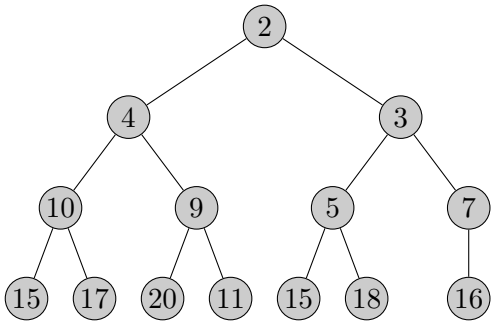
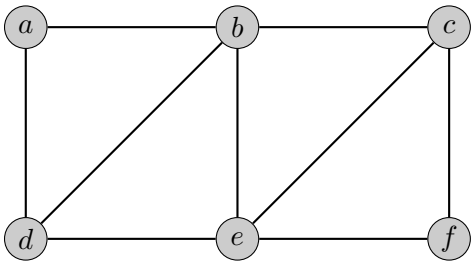
Written Problems. due Feb 18 (Thu), 6.30 pm.

1. **(Stable Marriage Problem)** Textbook Chapter 1, Exercises 1.

2. **(Extract-Min)** Draw the tree you obtain when you execute EXTRACTMIN on the heap represented below:



3. **(Breadth-First Search)** Compute a breadth-first search tree on the following undirected graph, starting from the node a. Assume that we always process the neighbors of a vertex in alphabetical order. Indicate with dotted lines edges that are in the graph but not in the BFS tree.



Programming Problems. due Feb 18 (Thu), 11.59 pm.

1. **(Longest Common Substring)** Humans have 23 pairs of chromosomes, while other primates like chimpanzees have 24 pairs. Biologists claim that human chromosome #2 is a fusion of two primate chromosomes that they call 2a and 2b. We wish to verify this claim by locating long nucleotide chains shared between the human and primate chromosomes.

   We define the *longest common substring* of two strings to be the longest contiguous string that is a substring of both strings e.g. the longest common substring of DEADBEEF and EA7BEEF is BEEF.[1] If there is a tie for longest common substring, we just want to find one of them.

   Download `ps2-dna.zip` from the class website.

   (a) Ben Bitdiddle wrote `substring1.py`. What is the asymptotic running time of his code? Assume $|s| = |t| = n$.
   (NOTE: If s is a length n string, then the operation `s[0:n/2]` takes $\Theta(n)$ time.)

[1] Do not confuse this with the *longest common subsequence*, in which the characters do not need to be contiguous. The longest common subsequence of DEADBEEF and EA7BEEF is EABEEF.

(b) Alyssa P Hacker realized that by only comparing substrings of the same length, and by saving substrings in a hash table (in this case, a Python set), she could vastly speed up Ben's code.

Alyssa wrote `substring2.py`. What is the asymptotic running time of her code?

(HINT: Assume that the operations `add`, `remove`, and `__contains__` and `__len__` on a Python set run in constant time.)

(c) Using binary search on the length of the string, implement an $O(n^2 \log n)$ solution. You should be able to copy Alyssa's `k_substring` code without changing it, and just rewrite the outer loop `longest_substring`.

Check that your code is faster than `substring2.py` for `chr2_first_10000` and `chr2a_first_10000`.

Put your solution in `substring3.py`, and copy the file to your `submit` directory.

2. **(Heap Delete)** In this problem you will implement the heap data structure.

Download `ps2_heap.zip`. Implement `heapify_down(i)` and fix the buggy implementations of `insert(v)` and `delete(i)`. Your code for `heapify_down` should run in $O(\log n)$ time where n is the number of nodes currently in the heap.

Run `test_heap.py` to help determine if your new delete method works, and submit `heap.py` by copying the file to your `submit` directory. The entire test suite in `test_heap.py` must complete running in under 12 secs on `owl.cs.qc.edu`.