CMSC 420 Problem List

1 Warm-Up Problems

Q1.1. It is the case that $\mathcal{O}(\sqrt{n})$ is slower than $\mathcal{O}(\lg n)$. Formally this means that we have $\lg n = \mathcal{O}(\sqrt{n})$ but $\sqrt{n} \neq \mathcal{O}(\lg n)$.

(a) State the limit theorem (from CMSC351) which applies to $\mathcal{O}$.

(b) Prove that $\lg n = \mathcal{O}(\sqrt{n})$ using the $\mathcal{O}$ limit theorem.

(c) Prove that $\sqrt{n} \neq \mathcal{O}(\lg n)$ using the definition of $\mathcal{O}$.

Note: This is a short but fairly challenging mathematical argument.

Q1.2. Suppose h is a nonnegative integer and we are given that:

$$n \leq \sum_{i=0}^{h} 3^i$$

Find an integer lower bound for h which does not include a sum.

Q1.3. Suppose m and h are positive integers. Simplify the following sum:

$$1 + \sum_{i=0}^{h-1} 2 \left\lceil m/2 \right\rceil^i \left(\left\lceil m/2 \right\rceil - 1\right)$$

Q1.4. When we talk about *the Ackermann function* in the context that we do, we generally mean the function defined first using:

$$\begin{aligned} A(0,n) &= n+1 \\ A(m+1,0) &= A(m,1) \\ A(m+1,n+1) &= A(m, A(m+1,n)) \end{aligned}$$

Then via:

$$Ack(n) = A(n,n)$$

Calculate $Ack(n)$ for $n = 0, 1, 2$. Show intermediate steps, don't just give the answer.

Q1.5. Suppose algorithm A has time complexity $T_A(n) = \Theta(n)$ and algorithm B has time complexity $T_B(n) = \Theta(n^2)$.

(a) Which algorithm is generally accepted as "faster" and why?

(b) Give an example of two specific running times $T_A(n)$ and $T_B(n)$ such that algorithm B is faster for $0 \leq n < 10000$ and algorithm A is faster for $n > 10000$ and show why this is the case. Along with your example explain how you came up with it, so don't simply give the example. Both should be valid for all $n \geq 0$.

Q1.6. We wish to store a collection of n key-value pairs where the keys are integers and the values are strings. We put the key-value pairs in a 0-indexed list of objects where each object represents a pair and such that the keys are in increasing order. For example the following key-value pairs:

$$[3, apple],\ [10, cat],\ [5, math],\ [-2, hello]$$

Would be stored in a list A with:

$$A[0] = [-2, hello],\ A[1] = [3, apple],\ A[2] = [5, math],\ A[3] = [10, cat].$$

(a) To search for a particular key we will use a binary search. What is the best and worst case $\mathcal{O}$ time complexity of this? Explain.

(b) To insert a key-value pair we use a binary search to find the next smallest key (do you know how to do this?) and shift to make space accordingly. What is the best and worst case $\mathcal{O}$ time complexity of this? Explain.

(c) To delete a key-value pair we use a binary search to find the key and then splice to remove the key-value pair accordingly. What is the best and worst case $\mathcal{O}$ time complexity of this? Explain.

Q1.7. Suppose we construct a data structure to store distinct nonnegative integers as follows:

We have a standard list (1-d array) A for which each entry has the form $[a, ptr]$ where a is a nonnegative integer and ptr is a pointer to a sorted linked list. The list A is sorted by the integer values.

Given an integer k which we wish to store, Let α be the leftmost half (rounding down) of the digits in k and let β be the rightmost remaining digits. (For example if $k = 1234$ then $\alpha = 12$ and $\beta = 34$ and if $k = 12345$ then $\alpha = 12$ and $\beta = 345$.)

We first see if $[\alpha, ptr]$ (for some pointer ptr) exists in A and if it does then we insert β into the linked list which ptr points to, otherwise we create a linked list pointed to by ptr consisting of the value β and insert $[\alpha, ptr]$ into A.

(a) Illustrate the data structure look like after the following are inserted: $k = 46, 456, 1234, 409$

(b) As a function of n, the number of elements in the data structure, What is the worst-case $\mathcal{O}$ time complexity of inserting some integer k? Assume that we can use a binary search on A (but not on the linked lists) and that extracting α and β from k are $\mathcal{O}(1)$. Explain.

Q1.8. One of the most frustrating things when it comes to proofs related to algorithms is the fact that floors and ceilings show up and get in the way. Prove that for all $x \geq 2$ we have:

$$\left\lfloor \log_{3/2}(x+1) \right\rfloor - \left\lfloor \log_{3/2} x \right\rfloor \leq 1$$

Note 1: This is not particularly difficult if you're comfortable with some basic calculus so if you find your solution getting a little out of hand you're probably doing more than you need to!

Q1.9. If algorithm A has time complexity $\mathcal{O}(n \lg n)$ and algorithm B has time complexity $\mathcal{O}(n^2)$ does this mean we should always use algorithm A? Explain in no more than one paragraph!

Q1.10. Prove that for any positive integer $x \geq 1$ we have:

$$\lceil \log_3 x \rceil \leq \lfloor \log_2 x \rfloor$$

Q1.11. Suppose A is a 0-indexed list of integers which has the property that if A has length j then inserting an integer at index $0 \leq k \leq j$ (where $k = j$ is an append) takes time $j - k + 1$ seconds.

(a) Starting with a list of length 0, how many seconds will it take to append n integers?

(b) Starting with a list of length n, if an integer is inserted into a random index $0 \leq k \leq n$ (with each index equally likely) what is the expected amount of time required? Simplify your answer.

2 Amortized Analysis

Q2.1. Consider amortized cost for the scenario where we have a stack such that:

- A pop costs 1.
- A push costs 1, not counting any allocation that may be necessary.
- Re-allocation costs the new size. This cost includes copying the contents of the old stack.

Assume we start with 0 bytes allocated and when we re-allocate we multiply the length by $\alpha \in \mathbb{Z}$ where $\alpha \geq 2$ except for the very first re-allocation which goes from 0 to 1.

Define:

$$f(\alpha) = \frac{\alpha^2 + \alpha - 1}{\alpha - 1}$$

(a) Use the aggregate method to prove that in the worst case with n operations the amortized cost $AC(n)$ of each stack operation satisfies $AC(n) \leq f(\alpha)$.

(b) Prove mathematically that this expression is minimum when $\alpha = 2$.

(c) Use the token method to prove that collecting $f(\alpha)$ tokens per push/pop is sufficient. As in class you can ignore the first and last runs.

Q2.2. Suppose we execute a sequence of n operations on a data structure. Each operation has a base cost of 2 but the ninth and tenth operations have supplemental costs of 4 each. Use the token method to calculate the amortized cost. You can focus on just one run.

Q2.3. Consider amortized cost for the scenario where:

- A pop costs 1.
- A push costs 1, not counting any allocation that may be necessary.
- When reallocation is necessary we multiply the length by 3.
- Rellocation costs the size of the new list and includes copying the old elements over.

Suppose that we do some sequence of n pushes and pops. Use the aggregate method to prove that the amortized cost is less than 5.5.

Q2.4. Suppose we initially allocate 1 byte of contiguous memory to store a stack, which begins empty. Each time we overflow the memory we reallocate to twice the previous length plus 1 extra byte and copy over the original stack. The reallocation cost in tokens equals the new size and this includes the copying. Each push costs 1 token. Use the token method of amortized analysis to show that charging 6 tokens per operation is sufficient to cover the costs. You only need to show this is true for one full run.

Q2.5. Suppose you are working on a list structure onto which you will append $i = 1, 2, ..., n$.

You start with zero entries allocated and decide that when an append will overflow the list you will reallocate to the next highest perfect square.

For example the $i = 1$ append overflows and hence requires a reallocation to length 1. The $i = 2$ append overflows and hence requires a reallocation to length 4. The $i = 3, 4$ appends require no reallocation but the $i = 5$ append overflows and hence requires a reallocation to length 9. And so on.

Assuming that each append cost 1 and reallocating costs the size of the new list, including the copying, prove that the amortized cost is $\mathcal{O}(\sqrt{n})$.

Use the aggregate method.

Q2.6. Suppose we have an list A which represents a binary string, so $A[0]$ represents the $2^0 = 1$s digit, $A[1]$ represents the $2^1 = 2$s digit, $A[2]$ represents the $2^2 = 4$s digit, and so on.

Suppose A starts at all 0s and we then implement n increment operations. Every time a bit flips the cost is 2^k where 2^k is the index of the bit.

Show that $AC(n) = \mathcal{O}(\lg n)$ using the aggregate method.

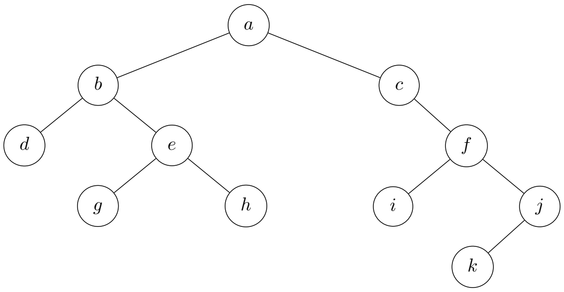
Q2.7. Revisiting the stack allocation problem suppose that when reallocation is necessary we multiply the length by some constant $C \in \mathbb{Z}$ with $C > 1$. Each push costs 1 and a reallocation cost equals the number of bytes being allocated and this includes copying the old data.

(a) Use the token method to find an upper bound on the amortized cost. Show the calculation for one full run.

(b) As $C \to \infty$, what happens to $AC(n)$?

3 Binary Trees and Modifications

Q3.1. Consider the following tree:



(a) List the nodes according to a pre-order traversal.

(b) List the nodes according to an in-order traversal.

(c) List the nodes according to a post-order traversal.

(d) Redraw the tree and include in-order threads.

Q3.2. Define b_h to be the number of distinct binary tree structures with height h. We don't care about the contents, just the overall structure.

(a) What is b_0 and why?

(b) What is b_1 and why?

(c) Prove that for $h \geq 2$ we have:

$$b_h = b_{h-1}^2 + 2b_{h-1}\left(1 + b_0 + b_1 + ... + b_{h-2}\right)$$

Q3.3. A trinary tree is a tree in which each node has between 0 and 3 children (call these the left child, middle child, and right child). Define $T(h)$ for $h \geq -1$ to be the number of distinct trinary tree structures with height h. We don't care about the contents, just the overall structure.

(a) What is $T(-1)$ and why?

(b) What is $T(0)$ and why?

(c) What is $T(1)$ and why?

(d) Prove that for $h \geq 1$ we have:

$$T(h) = 3T(h-1)(T(-1)+T(0)+...+T(h-2))^2 + 3T(h-1)^2(T(-1)+T(0)+...+T(h-2)) + T(h-1)^3$$

(e) Use this to calculate $T(2)$ for $n = 2$.

Q3.4. For a binary tree T with n nodes define $d(x, T)$ to be the depth of the node x (the root is depth 0) and define:

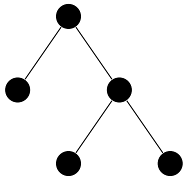
$$P(T) = \sum_{x \in T} d(x, T)$$

If T_L and T_R are the two subtrees of the root of a tree T, prove that:

$$P(T) = P(T_L) + P(T_R) + n - 1$$

Q3.5. Arrange the numbers 1 through 10 such that when inserted into a binary search tree (which is originally empty) the tree is a complete binary tree after each and every insertion. Briefly explain the logic you used to come up with your answer.

Q3.6. (a) Consider the binary tree structure:



Assign the list $A = [1, 2, 3, 4, 5]$ as node keys so the result is a binary search tree.

(b) Given an arbitrary binary tree structure with n nodes explain algorithmically (how would you do it?) why it is always possible to select n distinct integers and assign them as node keys within the structure so that the result is a binary search tree.

Q3.7. Here is the pseudocode for an inorder traverse:

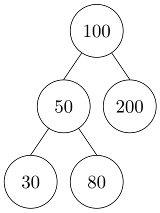
```
function inorder(x):
    if x != NULL:
        inorder(x.left)
        print(x.key)
        inorder(x.right)
    end if
end function
```

Prove by strong induction on $n \geq 0$, the number of nodes in a tree, that if `inorder(x)` is called where `x` is the root that the output is the keys in increasing order. Assume the keys are all distinct.

4 AVL Trees

Q4.1. In class we drew pictures of the effect of a right rotation on a left-left heavy AVL tree after insertion and the effect of a left-right rotation on a left-right heavy AVL tree after insertion. Draw appropriate pictures of the effect of a left rotation on a right-right heavy AVL tree after insertion and the effect of a right-left rotation on a right-left heavy AVL tree after insertion.

Q4.2. Starting with the following AVL tree:



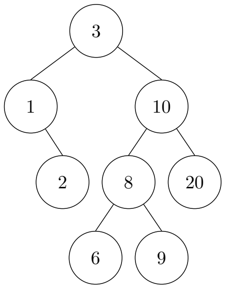
Perform the following operations cumulatively in the order given. For each operation draw:

- The tree after the operation but before any required rebalancing.
- The tree after each step of any required rebalancing operations. For example if a single rotation is required, just draw the result, but if a double-rotation is required draw the result after each component rotation.

(a) Insert 90.

(b) Insert 10.

Q4.3. Given the following AVL tree:



For the following questions start with the above tree. In other words the questions do not build from one another!

(a) Show the process (each step) through which the key 5 is inserted into the tree.

(b) Show the process (each step) through which the key 2 is deleted from the tree.

Q4.4. In class we showed that for an AVL tree with height h and n nodes that when h is even then $N(h) > 2N(h-2)$ implies that $h = \mathcal{O}(\lg n)$, where $N(h)$ is the minimum number of nodes in an AVL tree of height h. Prove that the statement is also true when h is odd.

Q4.5. Here is the code (in Python) for the left rotation in an AVL tree:

```
def rotateLeft(root: Node) -> Node:
    RightChild = root.rightchild
    RightLeftSubtree = RightChild.leftchild
    RightChild.leftchild = root
    root.rightchild = RightleftSubtree
    return RightChild
```

Suppose that the `Node` class also contains variables `Node.leftheight` and `Node.rightheight` which store the heights of the left and right subtrees.

Write down the two (and only two!) necessary lines which will go immediately before the `return` statement so that these two variables are updated wherever necessary. Your result does not need to be proper Python, pseudocode will suffice.

Q4.6. Suppose b_h equals the number of different AVL tree structures of height h. We don't care about the contents, just the overall structure.

(a) Calculate b_0 and b_1 via illustrations.

(b) Prove that for $h \geq 2$ we have:

$$b_h = 2b_{h-1}b_{h-2} + b_{h-1}^2$$

(c) Use this to calculate b_h for $h = 2, 3, 4$.

(d) Verify your calculation for $h = 2$ by illustrating all possible structures.

Q4.7. The following statement is either true or false. State which you believe it to be and justify. Note: The depth of a node is the distance (number of edges) from the node to the root.

Statement: For any AVL tree if L_i and L_j are leaves, then:

$$\text{depth}(L_i) - \text{depth}(L_j) \in \{-1, 0, 1\}$$

Q4.8. Define an AVL-2 tree to be a BST where each node v satisfies balance$(v) \in \{-2, -1, 0, 1, 2\}$. Let $N(h)$ be the minimum number of nodes in an AVL-2 tree of height h. Prove that $h = \mathcal{O}(\lg n)$ for the case where h is a multiple of 3.

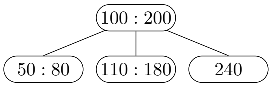
Q4.9. Define an AVL-2 tree to be a BST where each node v satisfies balance$(v) \in \{-2, -1, 0, 1, 2\}$. Let $N(h)$ be the minimum number of nodes in an AVL-2 tree of height h. Prove that $h = \mathcal{O}(\lg n)$ for the case where h is a multiple of 3 plus 1.

Q4.10. Suppose an AVL tree with height h has n nodes. Let F_i be the i^{th} Fibonacci number, so $F_0 = 0$, $F_1 = 1$, and $F_j = F_{j-1} + F_{j-2}$ for $j \geq 2$. Prove that:

$$n \geq F_{h+3} - 1$$

2-3 Trees

Q4.1. Given the following 2-3 tree:



For the following questions start with the above tree. In other words the questions do not build from one another!

(a) Show the process (each step) through which the key 190 is inserted into the tree.

(b) Show the process (each step) through which the key 240 is deleted from the tree.

(c) Insert 31.

(d) Delete 28.

Q4.2. will contain between $2^{h+1} - 1$ and $3^{h+1} - 1$ keys. Suppose you wished to store exactly 10000 keys. Which tree heights would suffice? Justify.

Note: This question is about keys, not nodes!

Q4.3. For an integer $k \geq 3$ a 2-k tree is the obvious generalization of a 2-3 tree in which each node may have between 2 and k children and a node with $2 \leq i \leq k$ children contains $i - 1$ keys. As with a 2-3 tree all leaf nodes are at the same height. Find, with proof, the minimum and maximum number of keys possible in all of the nodes of a 2-k tree of height h.

Q4.4. Suppose b_h equals the number of different 2-3 tree structures with height h. We don't care about the contents, just the overall structure.

(a) It's true that $b_0 = 2$. Draw the two possible trees.

(b) Prove that for $h \geq 1$ we have:

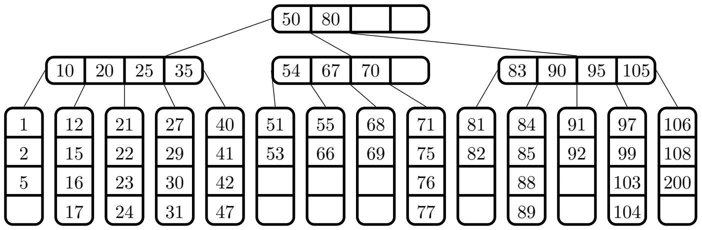
$$b_h = b_{h-1}^2 \left(b_{h-1} + 1\right)$$

(c) Use this to calculate b_h for $h = 1, 2, 3$. Only simplify b_1 and b_2.

Q4.5. What is the minimum and maximum number of levels in a 2-3 tree with n nodes? Justify.

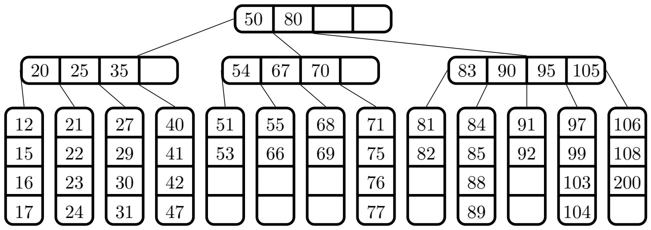
5 B-Trees

Q5.1. Consider this B-tree with $m = 5$:



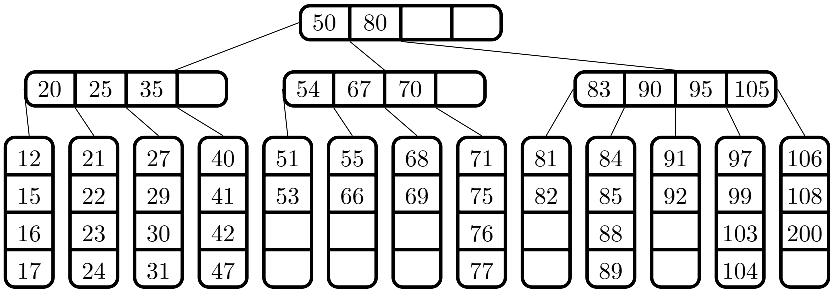
Explain in words how the tree will be rebalanced if 13 is inserted. Use words and perhaps small sub-trees of the picture above. Don't redraw the entire tree, that would be insane.

Q5.2. Consider this B-tree with $m = 5$:



Explain in words how the tree will be rebalanced if 28 is inserted. Use words and perhaps small sub-trees of the picture above. Don't redraw the entire tree, that would be insane.

Q5.3. Consider this B-tree with $m = 5$:



Explain in words how the tree will be rebalanced if 51 is deleted. Use words and perhaps small sub-trees of the picture above. Don't redraw the entire tree, that would be insane.

Q5.4. Suppose a B-tree of order $m = 10$ has 10^j keys. For each of $j = 1, 3, 6, 9$ calculate the maximum and minimum possible heights. Together on one jh-axis (j is horizontal) plot the maximum and minimum heights as a function of j. Connect the maximum heights with a smooth line and connect the minimum heights with a smooth line. What do you notice?

Q5.5. Prove the unproven theorem from class: A B-tree of order m with k keys and height h satisfies:

(a) The densest possible such tree has:

$$k = m^{h+1} - 1$$

(b) Consequently any such tree has:

$$k \leq m^{h+1} - 1$$

(c) And it then follows that:

$$h \geq \log_m(k+1) - 1$$

(d) Thus:

$$h = \Omega(\lg k)$$

Q5.6. Technically speaking when searching in a B-tree we need to search the keys of each node in order to figure out which child branch to follow.

(a) Why is this not included into our asymptotic search time complexity?

(b) Suppose we wanted to compute the asymptotic search time complexity not just as a function of k but as a function of both k and m. First do so, then use some basic precalculus to simplify this to a somewhat surprising result. If you're not sure when you should be surprised, ask Justin or a TA and we can tell you whether you need to simplify further.

Q5.7. Why is it not sensible to define a B-tree of order $m = 2$?

Q5.8. Suppose we have a B-tree with order $m \geq 3$. Suppose a non-root node is underfull and neither of its siblings is available for rotation. Prove that merging with either sibling does not result in an overfull node.

Q5.9. Generally when we deal with an overfull or underfull node in a B-tree we attempt to rotate before splitting (for insertion) or merging (for deletion). Suppose we accidentally write our code so the splitting or merging is attempted first. What long-term effects might this have on the structure of the B-tree? Explain. This does not need to be justified rigorously, just explain what you think as best you can.

Q5.10. Suppose we have a B-tree of even order $m \geq 4$. Suppose $b > 1$ and the number of keys at time t is $k(t) = b^t$. Prove mathematically (think derivative) that as $t \to \infty$ the maximum height approaches a line and find the slope of that line.

Q5.11. Suppose a B-tree of order 10 contains 1999 keys. What are its maximum and minimum possible heights? You should be able to give exact integer answers (without a calculator).

Q5.12. Suppose a B-tree of order $m = 10$ contains exactly 9999 keys. For each of the following you should be able to get an exact answer without a calculator.

(a) What is the minimum possible height?

(b) What is the maximum possible height?

Q5.13. Suppose we have a B-tree with order $m \geq 3$. You may assume that m is odd. Prove that when a node is split that the two resulting new nodes are not underfull.

Q5.14. Suppose we have a B-tree of order $m \geq 4$. Suppose $b > 1$ and the number of keys at time t is $k(t) = b^t$. Prove mathematically (think derivative) that as $t \to \infty$ the minimum height approaches a line and find the slope of that line.

Q5.15. We know that for a B-tree of order $m \geq 3$, height h, and k keys, that:

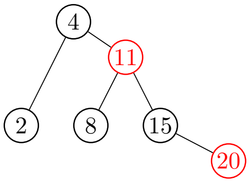
$$2\lceil m/2 \rceil^h - 1 \leq k \leq m^{h+1} - 1$$

Suppose m and h are fixed. Prove that for every k in that range, we can actually construct a B-tree with k keys.

Note: The theorems which give us the inequalities above simply give us upper and lower bounds on k and say that those upper and lower bounds are possible, they don't say that all k in that range are possible - that's what you're proving here.

6 AA Trees

Q6.1. Given the following AA tree:



The following questions are not cumulative.

(a) Show the steps involved when 17 is inserted into the tree.

(b) Show the steps involved when 2 is deleted from the tree.

Q6.2. Suppose an AA tree has R red nodes. Find a lower bound on the number of levels that the tree may have. Explain.

Q6.3. What is the maximum number of red nodes possible in an AA tree with L levels? Explain.

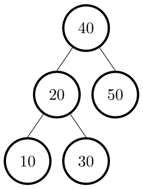
Q6.4. Suppose an AA tree has the maximum number of nodes possible at each level. For each level $L \geq 0$ (with $L = 0$ being the root level) define:

$$B(L) = \text{the number of black nodes at level } L$$
$$R(L) = \text{the number of red nodes at level } L$$

Explain why $B(L-1) = B(L) + 2R(L)$.

7 Scapegoat Trees

Q7.1. Given the following scapegoat tree:



Insert the keys $31, 32, 33, 34, 35$ in order. Restructure as needed. Explain all relevant computations. You don't need to show every intermediate tree but show enough to be clear.

Q7.2. Regarding scapegoat trees:

(a) Suppose n is a positive integer. Prove that if n is not a power of 2 then $\lfloor \lg(n-1) \rfloor = \lfloor \lg n \rfloor$.

(b) Use strong induction to prove the theorem from class that states that a restructured tree with n nodes will have height $\lfloor \lg n \rfloor$. Part (a) will probably be useful.

Q7.3. Suppose that at some instant we know that the height of a scapegoat tree satisfies the following with $n \geq 2$:

$$h \leq \left\lfloor \log_{3/2} n \right\rfloor + 1$$

(a) Suppose we insert a node and do not need to rebuild the tree. Thus we now have $n+1$ nodes. Denote the new height by h'.

Prove that the height condition is still valid. That is, prove:

$$h' \leq \left\lfloor \log_{3/2}(n+1) \right\rfloor + 1$$

(b) Suppose we delete a node and do need to rebuild the tree. Thus we now have $n-1$ nodes. Denote the new height by h'.

Prove that the height condition is still valid. That is, prove:

$$h' \leq \left\lfloor \log_{3/2}(n-1) \right\rfloor + 1$$

Q7.4. Suppose we insert keys $1, 2, 3, ...$ in that order into an initially empty generalized scapegoat tree with parameter α. Suppose we fix a key k and insist that α be chosen so that k will be the first inserted key which triggers a scapegoat search.

(a) Find an inequality for α in terms of k.

(b) Take your inequality and make it an equality and then plot this function for $1 \leq k \leq 100$ and explain why its shape makes sense. You should comment both on the general shape (increasing? decreasing?) nature of the graph as well as its behavior as $k \to \infty$.

Q7.5. Regarding Scapegoat trees, we proved a lemma in class that stated:

If a node p has $\text{depth}(p) > \log_{3/2} n$ then p has an ancestor u for which:

$$\frac{\text{size}(u.child)}{\text{size}(u)} > \frac{2}{3}$$

Suppose we replace $\frac{3}{2}$ by $\beta > 1$ and we want generalize the above to:

If a node p has $\text{depth}(p) > \log_{\beta} n$ then p has an ancestor u for which:

$$\frac{\text{size}(u.child)}{\text{size}(u)} > f(\beta)$$

(a) Following the proof from class what would be the obvious choice for $f(\beta)$? Explain.

(b) Prove that the statement is true with this $f(\beta)$.

Q7.6. Suppose we want to reduce the frequency of rebuilding a scapegoat tree on insertion. To do this, should we increase or decrease the value of α? Explain.

Q7.7. Insert the keys $3, 2, 1, 4, 5, 6, 7, ...$ in that order into an empty scapegoat tree until the first insertion violates the scapegoat height condition. Identify the scapegoat and rebuild. You only need to show:

- The tree when the violation occurs.
- Justification of the violation.
- Justification of the scapegoat.
- The tree after the rebuild.

Q7.8. Prove that for a node u in a scapegoat tree, if u violates the scapegoat condition with $\alpha = 2/3$ and with its left child then we must have:

$$\frac{1}{2}\text{size}(u.left) > 1 + \text{size}(u.right)$$

Q7.9. This problem works through a specific example of the token calculations associated with the amortized analysis of a Scapegoat Tree. Starting with an empty Scapegoat Tree, Suppose the following sequence of operations is performed where I means insert and D means delete:

{ I:10, I:3, I:1, I:4, I:20, I:30, I:40, I:50, I:60, D:60, I:60, I:70 }

The final insert will trigger a scapegoat hunt and rebuild but for now don't worry about that.

(a) Draw a table showing the progress of the ID account balance during the sequence of operations.

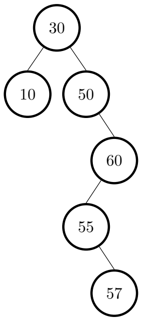
(b) How many tokens are in the DR account at the end? Explain.

(c) Draw a picture of the tree and next to each node show the totals for the IR account at the end.

(d) Find the scapegoat and explain how there will be enough tokens to cover the rebuild.

8 Splay Trees

Q8.1. Show the result of inserting 52 into the splay tree shown below. Show enough detail and provide enough explanation as to clarify your steps.



Q8.2. In a splay tree with height h when we search for a key in a node p, when p is in the tree we know p ends up at the root. It is possible that during this process that some other node gets a free ride closer to the root than it was. What is the maximum number of levels above its original level could some other node end up at? Justify.

Q8.3. (a) Explain why there are exactly six possible operations we might implement during each step of a splay process.

(b) Find the *inverse* of each. In other words for each operation in (a) which operation would undo it?

Q8.4. Suppose a node p is inserted into a splay tree. We know that p ends up at the root. If r was the original root (before insertion) What is the shallowest and deepest level that r could end up at after insertion of p? Assume the root is level 0. Justify.

Q8.5. Suppose x and y are keys/nodes in a splay tree with $x \neq y$. Is it possible that splay(x) could bring y closer to the root than it was? Justify.

Q8.6. Suppose a splay tree is undergoing a sequence of operations (deletions, insertions, and searches). The order is unclear except every other operation is a search for a specific key k_0.

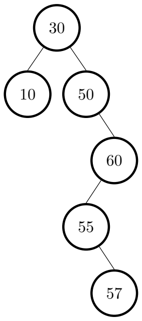
(a) What does this guarantee about the location of k_0? Justify.

(b) What does this say about the time complexity of searching for k_0? Justify.

Q8.7. In a splay tree suppose x is the key at the root and $splay(y)$ is called where y is a key in the tree. It's possible that $y = x$. What are the possibilities for how far x could end up from the root? Provide evidence and/or examples.

Note that your answer should not just say where it could end up but it should justify why it could end up in those places and why those are the only possibilities.

Q8.8. Show the result of deleting 55 from the splay tree shown below. Show enough detail and provide enough explanation as to clarify your steps. For the secondary search use the left subtree.

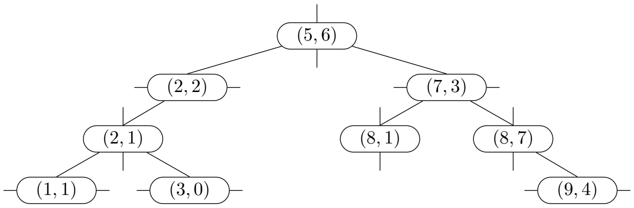


9 KD-Trees

Q9.1. Insert the following points in order into a k-d tree of dimension 2. Starting the cutting at the root node in the x-direction.

$$(60, 90), (20, 70), (70, 50), (30, 60), (40, 10), (50, 20), (80, 80), (10, 30), (90, 40)$$

Q9.2. Consider the following kd-tree of dimension $k = 2$:



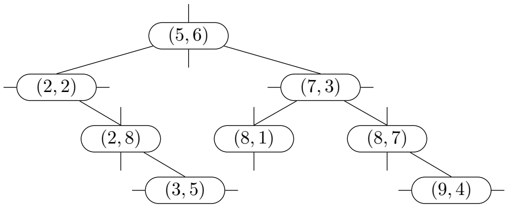
Each of (a) and (b) should start with this tree - they are not cumulative!

(a) Show the resulting tree when $(7, 3)$ is deleted. Just state briefly what happens (a sentence or two) and show the final tree.

(b) Show the resulting tree when $(2, 2)$ is deleted. Just state briefly what happens (a sentence or two) and show the final tree.

Q9.3. Suppose we are working with a kd-tree of dimension $k = 2$. For a node p define the *bounding box* for p to be the smallest axes-aligned (not on an angle) rectangle in the xy-plane which contains all of the points in the subtree rooted at p. The bounding box has the form of a list of two lists $[[x_{min}, x_{max}], [y_{min}, y_{max}]]$.

Given the following kd-tree:



(a) Calculate the bounding boxes for $(2, 2)$ and for $(7, 3)$.

(b) Suppose you have pre-emptively calculated the answer to (a) and you are looking for the point closest to the origin. Why would it make more sense to check the $(2, 2)$ bounding box first? You may not say anything about the points in the bounding boxes, just about the bounding boxes themselves. This is just a one-sentence answer with minimal calculation.

Q9.4. Suppose we have a perfect kd-tree with $k = 2$ (so storing points (x, y)) using alternate splitting where the root splits by x and with L levels where L is even.

Moreover suppose there are no repeated x- or y-coordinates at all.

(a) How many nodes could potentially contain the point with minimum x-value? Write this in terms of L. Justify.

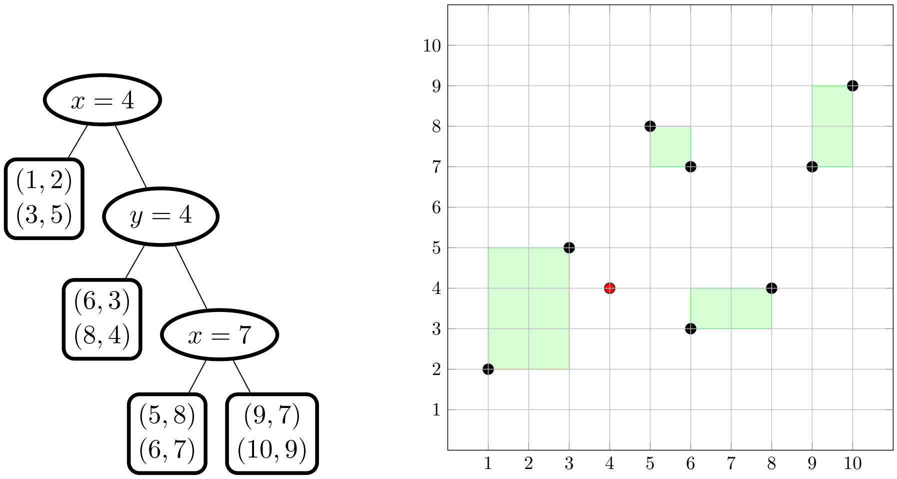
(b) We know that if n is the number of nodes then $n = 2^{L+1} - 1$ so rewrite your answer to (a) in terms of n.

Q9.5. Give an example of a kd tree such that using the inorder predecessor with respect to the splitting coordinate as a replacement node during deletion will ruin our requirement that a point with equal coordinate always branches right. Demonstrate why your example works.

10 Extended KD-trees

Q10.1. In an extended KD-tree when we split according to the largest spread we use the median of the node's data. What is a good reason not to use the mean?

Q10.2. Consider the following extended kd-tree. The tree is on the left and the plane diagram is on the right. The shaded boxes are the bounding boxes for the leaf nodes.



(a) Draw separate plots of the bounding boxes for each of the non-leaf nodes.

(b) Suppose we are looking for the two nearest neighbors to $(4, 4)$. We are following the algorithm discussed in class and used on the coding project. Walk through this algorithm, explaining each step briefly, and write down what the list looks like after each change

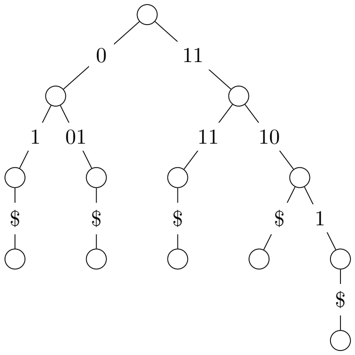
Q10.3. Suppose in an extended KD-tree with n nodes for each node p the bounding box for the subtree rooted at p is stored as part of p's data. Explain how updating this information in a KD-tree during insertion is average case $\mathcal{O}(\sqrt{n})$.

Q10.4. Give a graphical example in 2D of a point P and two bounding boxes B_1 and B_2 such that all of the following are satisfied:

- P is not in either bounding box.
- Each bounding box contains exactly two points.
- P is closer to B_1 than it is to B_2.
- P is closer to both points in B_2 than it is to both points in B_1.

11 Tries

Q11.1. Why is the following not a valid compressed trie?



Q11.2. Draw the compressed trie containing the binary strings:

11010, 1110, 11110, 0101, 01011, 01000

Q11.3. Is it true that insertion into a compressed trie will only increase the height of the trie if the inserted string has an entire existing string as a prefix? Justify your answer.

Q11.4. Prove that the height of a compressed trie T with n strings satisfies $h \leq n + 1$. This includes final branches to "$". The outline of this proof is in the PDF notes but the details are missing from there. This can be done by weak or structural induction. Strong induction works too, but isn't necessary.

12 Skip Lists

Q12.1. Suppose the keys 30,10,80,50,20 are inserted into a randomized skip list in that order.

We know each element exists in level 0. When determine whether each element extends to the next higher level we use a random number generator which outputs either 0, which means the element extends no further, or 1, which means the element extends another level and we must check again.

Draw the resulting randomized skip list under the assumption that the random number generator outputs the following:

1,1,1,1,0,1,0,1,1,0,0,1,0

Q12.2. Suppose a skip list has 100 nodes. What is the probability that at least one node will reach a level of 5 or beyond?

Q12.3. This problem digs deeper into the probabilities associated with a skip list with n nodes reaching a certain number of levels.

(a) For each $k = 1, 2, 3, ...$ what is the probability that a skip list does not reach level k? Justify.

(b) For each $k = 0, 1, 2, 3, 4, ...$ what is the probability that a skip list does reach level greater than or equal to k? Justify.

(c) For each $k = 0, 1, 2, 3, 4, ...$ what is the probability that the skip list reaches exactly level k? Justify.

Q12.4. When building a skip list recall that there is a 0.5 probability that a node which reaches level i also reaches level $i + 1$. Suppose we change this probability to some $0 < p < 1$.

If $n = 80$ nodes are inserted into such a skip list what is the maximum value of p which ensures that the probability that at least one node reaches level 10 or beyond is 0.4 or below?

Q12.5. Suppose we build a skip list but we modify the probability such that each node has a p probability of extending to the next level.

(a) If the skip list has exactly one node, what do we expect the maximum level to be? Justify.

Note: It may be useful to know that:

$$\sum_{i=0}^{\infty} ix^i = \frac{x}{(1-x)^2}$$

(b) Suppose $p = 1/k$ for some integer k and the skip list has k^2 nodes what do we expect the maximum level to be, in terms of p? Justify.

Q12.6. Suppose a skip list contains n keys/nodes. Recall that the expected number of levels is $2+\lg n$. Assuming for simplicity of computation that $\lg n$ is an even integer, find an expression for the probability that at least one of the keys/nodes has more than half the expected number of levels.

13 Treaps

Q13.1. Suppose you have a treap in which all keys and all priorities are distinct. Write pseudocode for a function `findmaxpriority(root,min,max)` which takes two keys `min` and `max` (which may or may not be in the treap) and returns the maximum priority of all the nodes whose keys are between `min` and `max` (inclusive). Assume that `root` points to the root and that each node `n` has `n.key`, `n.priority`, `n.left`, and `n.right`. Assume there are nodes which satisfy the condition.

Q13.2. Consider the keys and random priorities:

$$[(10, 22), (20, 63), (30, 98), (40, 91), (50, 28), (60, 48), (70, 77), (80, 1), (90, 14), (100, 94)]$$

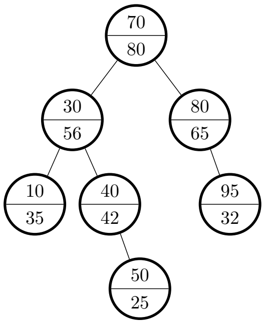
Fun fact: I had Wolfram Alpha generate these random priorities.

(a) Write down the (unique) treap which contains the above.

(b) Treating the above as a list of points in the xy-plane, plot them.

(c) What do you notice about your answers to (a) and (b)? This is a qualitative/investigative question.

Q13.3. Show the result of inserting the key 42 into the following treap assuming it is assigned a random priority of 60. Show each step in the progression back to a valid treap.



14 Hash Functions

Q14.1. Suppose $K = \mathbb{Z}$ and T has $m = 7$ entries (hence indexed by $\mathbb{Z}_7$). Define the hash function $h : K \to \mathbb{Z}_7$ by $h(x) = ax \mod m$ for $a = 2$. Suppose we are using closed addressing with linked lists and we set $\lambda_{min} = 0.2$ and $\lambda_{max} = 0.8$. We insert the following integers into the hash table:

$$4, 101, 10, 18, 17, 13, 2, 37$$

This should require exactly one rebuild. When the rebuild happens set the new m accordingly and set a to be the smallest integer greater than 1 which is coprime to the new m.

(a) Show the hash table (and its linked lists) after the last insert which does not violate the load factor maximum. Explain why there is no violation.

(b) Show the hash table (and its linked lists) after the last insert which does violate the load factor maximum. Explain why there is a violation.

(c) Show the hash table after the final insert. Explain why there is no violation.

Q14.2. Suppose $p > 3$ is a prime and we define our hash function $h : \mathbb{Z} \to \mathbb{Z}_p$. The explicit definition of h is not relevant here. To resolve collisions we use cubic probing, meaning if $T[h(x)]$ is full we check $T[h(x) + 1^3]$, $T[h(x) + 2^3]$, $T[h(x) + 3^3]$, and so on with additions taken mod p. We claim that cubic probing will probe at least $\left\lfloor \sqrt[3]{p/3} \right\rfloor + 1$ indices.

(a) Just to make sure you understand what this means, how many indices are guaranteed to be probed for each of $p = 11, 29, 59, 1009$?

(b) Prove the claim.

Hint: There is a similar proof in the PDF notes but you'll need to adjust it a bit. Beyond basic algebra and modular arithmetic all you need to know is that if a prime divides a product of two terms then it divides one of the terms.

Q14.3. Suppose $h : K \to \mathbb{Z}_m$ is a hash function which has the property that each index in $\mathbb{Z}_m$ is equally likely to be hit. Assume we are using closed addressing. Suppose n items have been hashed.

(a) How many empty indices are expected in T? Justify.

(b) Suppose $m = 100$. Use your answer to (a) to determine the largest n value such that you would expect at least half the indices to be empty.

Q14.4. Suppose for load-balancing our hash table we use $\lambda_{min} = 0$ and $\lambda_{max} = 0.5$. We start our hash table with length 1. Suppose we do a sequence of n inserts and each takes time 3. Prove via the aggregate method of amortized analysis that insertion is average case $\mathcal{O}(1)$. Assume that an entire rebuild takes time equal to length of the new table.

Q14.5. Suppose you decide to store your data in a hash table with only three entries using closed addressing using standard binary search trees. The keys are integers and the hash function you choose is:

$$h : \mathbb{Z} \to \mathbb{Z}_3 \text{ given by } h(x) = x + 1 \mod 3$$

You insert the following keys in the order given: $1, 5, 3, 7, 10, 9, 4, 2$

Show what the resulting structure looks like. You can illustrate this however you like as long as it is clear.

15 Bloom Filters

Q15.1. Suppose we define a Bloom filter using $S = \mathbb{Z}_{12}$, with B indexed by $\mathbb{Z}_{11}$, and with two hash functions:

$$\begin{aligned} h_1(x) &= 2x \mod 11 \\ h_2(x) &= 3x \mod 11 \end{aligned}$$

(a) Suppose you insert the keys $x = 1, 6, 9$. Show the state of the Bloom filter after each insertion.

(b) Identify all false positives.

Q15.2. Suppose we define a very small Bloom filter with $m = 3$ and $k = 2$.

(a) We insert $n = 1$ keys. Write down all the possible resulting bit arrays and the probability of each.

(b) We insert $n = 2$ keys. What is the probability that the resulting bit array is `100`?

(c) We insert $n = 2$ keys. What is the probability that the resulting bit array is `110`?

Q15.3. Suppose we define a Bloom filter with $m \geq 2$ and $k = 2$. We insert $n = 2$ keys.

(a) What is the probability (in terms of m) that the resulting bit array has `1`s in exactly one index and `0`s elsewhere?

(b) What is the probability (in terms of m) that the resulting bit array has `1`s in exactly two indices and `0`s elsewhere?

Q15.4. Using the inaccurate (but not terrible) bloom filter false positive expression:

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k}$$

(a) Suppose we wish to store $n = 70$ items with $k = 3$ hash functions and a bit array of size $m = 50$. What would the corresponding false positive probability be? Give an exact answer (unsimplified) and an approximation to four decimal digits.

(b) Suppose we wish to store $n = 100$ items and we want a false positive rate no higher than 2%. Find a restriction on m (as a function of k) which will guarantee this.

Q15.5. Suppose the state of a particular count array for a Counting Bloom Filter after n insertions with k hash functions is:

```
B = [3,1,2,1,0,2,2,0,0]
```

Suppose you don't know k but you know that n is a single-digit number. Moreover suppose that for any given key there are at least two hash functions which produce different outputs. Prove that there are at least six distinct hash functions.

Q15.6. Suppose the state of a particular count array for a Counting Bloom Filter after n insertions with k hash functions is:

```
B = [2,1,2,2,1,5,1,0,0]
```

Suppose you know both k and n are single-digit numbers and you know that for any given key no more than two of the hash functions will give identical results.

What are n and k? Justify.

Q15.7. Suppose we define a Bloom filter with $m = 4$ and $k = 3$. We insert $n = 2$ keys.

What is the probability that the resulting bit array has `1`s in exactly two indices and `0`s elsewhere?

16 Disjoint Set Data Structures

Q16.1. Draw a disjoint data forest corresponding to the following disjoint subsets. There are many answers.

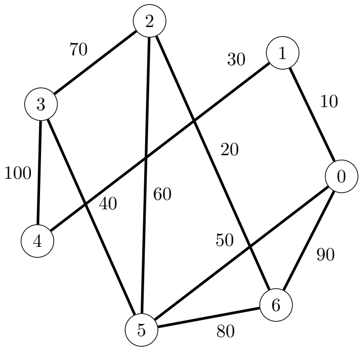
$$\{1, 5, 8, 10, 3\}, \{4, 6, 9\}, \{2, 7\}$$

Q16.2. Suppose a graph G starts with n nodes and no edges. We represent it by a disjoint set forest. Assume we are using path compression and weighted union.

(a) Assume $n = 2^k$ for some positive integer k. Prove by induction on k that it is possible to find a sequence of edges which could be added to the graph and which would result in the disjoint set forest finally ending up as a single tree of height exactly k.

(b) Is it possible to find a sequence of edges which could be added to the graph and which would result in the disjoint set forest ending up as a single tree of height exactly $n - 1$? Why or why not? Your answer does not need to be particularly formal, a general outline will suffice.

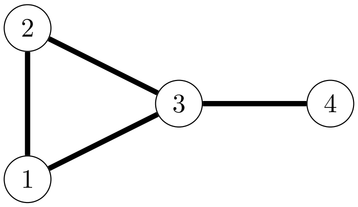
Q16.3. Suppose Kruskal's Algorithm (see the notes) is run on the following graph:



Draw the disjoint data forest at the start and then after each edge has been added. Use path compression and weighted union. Your pictures should be after these have occurred.

17 Storing Graphs

Q17.1. Consider the following graph:



(a) Write down the adjacency matrix for the graph.

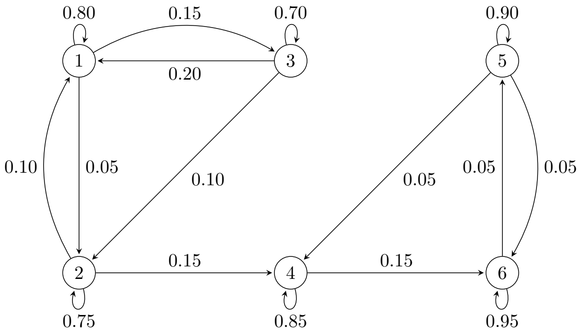
(b) Find the number of walks of length 3 from vertex 1 to vertex 3.

(c) Find the number of walks of length 20 from vertex 2 to vertex 4.

(d) There are no walks of length 3 from vertex 4 to itself. Rather than using A^3, explain intuitively why this is.

18 Markov Chains

Q18.1. Given the following transition diagram:



Find each of the following without actually finding the transition matrix T.

(a) The $(3, 2)$ entry of T.

(b) The $(3, 2)$ entry of T^2.

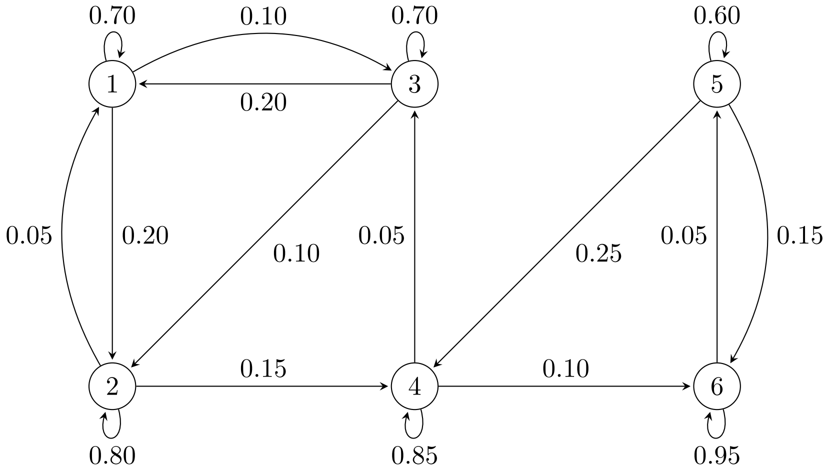
(c) The $(1, 3)$ entry of T^2.

(d) The smallest k such that the $(5, 3)$ entry of T^k is nonzeo and what that value is.

(e) All (i, j) such that the (i, j) entry of T^k is zero for all k.

(f) Do you think the Markov chain has a steady state to which all other states converge? Justify intuitively.

Q18.2. Given the following transition diagram:



(a) Write down the corresponding transformation matrix.

(b) Show that T is regular.

(c) Determine the corresponding limiting steady state to four decimal places using eigen methods.

Q18.3. Give an example of a 5×5 transition matrix such that $T \neq I$, $T^2 \neq I$, ..., $T^5 \neq I$, but $T^6 = I$.

Q18.4. The following problem is a very simple example that looks at whether it's better to have lots of low-ranked pages pointing to a page or just one high-ranked page pointing to a page:

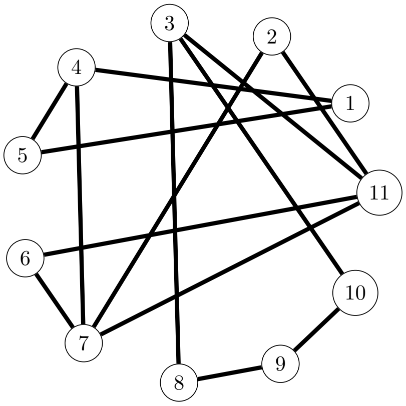
Consider an internet with $n \geq 3$ pages where pages P_3,...,P_n all point to P_2 and P_2 points to P_1.

(a) Show that for $n = 3$ the page ranking of P_1 is higher than that of P_2.

(b) Experiment to see whether, as n increases, this remains true.

19 Graph Partitioning

Q19.1. Consider the following graph:



(a) Find a Fiedler vector.

(b) The values in this Fiedler Vector, sorted, can be grouped into three separated subsets. Do so.

(c) Use this grouping to partition the graph into three subgraphs.

(d) Draw and label the separated components neatly and individually and then indicate with dashed lines the edges that go between them.

(e) From the previous step are there any insights you gain about the structure of the graph?

20 Blockchains

Q20.1. Consider the following simple hash function which works on messages of length equal to a multiple of four bits.

```
(i) Break $M$ up into $N$ sub-messages of length 4 bits: $M^{(1)}$, $M^{(2)}$, ... , $M^{(N)}$.
(ii) Initialize $H$ = 1010 and $K$ = 0101.
(iii) For each sub-message $M^{(i)}$ do the following:
        i. Update $H$ by XORing it with $M^{(i)}$.
        ii. Update $H$ by XORing it with $K$.
        iii. Update $K$ by left-rotating it by 1 bit.
(iv) The message digest is the resulting $H$, so $h(M) = H$.
```

Suppose this hash function is used to create a blockchain as follows.

(a) The $i = 0$ block contains four bits of data followed by `0000`.

(b) Each $i = 1, 2, ...$ block contains four bits of data followed by a hash of the entire previous block.

If the data for blocks $i = 0$ and $i = 1$ are `0111` and `1010` respectively, what do the corresponding blocks look like?