Section 09: Finals review

1. Finding closed forms

Find closed-form formulas for the following recurrences. Use the Master Theorem when applicable, and show your work when it is not applicable. Assume that $T(1) = 1$ for all recurrences.

Note: the last page of this section has a list of formulas.

i) $T(n) = 2T(n/2) + n$ (Hint: $\sum_{i=0}^{\log_2(n)-1} 2^i = n - 1$)

ii) $T(n) = 3T(n/3) + 1$

iii) $T(n) = 2T(n/2) + \log_2(n)$ (Hint: $\sum_{i=0}^{\log_2(n)-1} 2^i \cdot i = 2n\log_2(n) - 2n + 2$)

iv) $T(n) = 3T(n/3) + n$

v) $T(n) = 2T(n/2) + n^2$

vi) $T(n) = 3T(n/3) + n^2$

vii) $T(n) = 2T(n/2) + n^3$

viii) $T(n) = 3T(n/3) + n^3$

ix) $T(n) = 2T(n/2) + n^4$

x) $T(n) = 3T(n/3) + n^4$

xi) $T(n) = 2T(n/2) + n^5$

xii) $T(n) = 3T(n/3) + n^5$

2. Memory, locality, and dynamarrays

i) Consider an dynamarray implementation that doubles the capacity every time capacity runs out, and halves the capacity when it drops below 1/4 full.

Suppose the following sequence of operations occurs: 1000 push_back() calls followed by 1000 pop_back() calls. Assume that the initial capacity is 1000.

Draw a plot of the capacity of the dynamarray over time, with the time on the x-axis and the capacity on the y-axis.

ii) Suppose we implement a dynamarray using a contiguous block of memory. When the array runs out of capacity, we allocate a new block of memory, copy over the old array, and free the old array.

Suppose we implement a dynamarray using a linked list instead. Each node in the linked list stores a single element, and we keep a pointer to the last node in the linked list. Adding a new element to the end of the array takes O(1) time, and finding the element at index i takes O(i) time.

Which implementation would you prefer for each of the following workloads? Briefly explain why.

a) Random access

b) Sequential access

3. Heaps

Consider the following list of numbers:

[9, 7, 8, 3, 5, 6, 1, 2, 4, 0]

i) Draw the binary heap that results from inserting these elements into an initially empty max heap.

ii) Draw the binary heap that results from inserting these elements into an initially empty min heap.

iii) Draw the binary heap that results from inserting these elements into an initially empty heap, then extract-max twice.

iv) Draw the binary heap that results from inserting these elements into an initially empty heap, then extract-min twice.

v) Draw the binary heap that results from inserting these elements into an initially empty heap, then extract-max once and extract-min once.

vi) Suppose we want to find the median of a stream of numbers. Which of the data structures described on this page would be useful for this?

Suppose we want to find the median of a stream of numbers, but we don't care about the order of the numbers that are smaller than the median. Which of the data structures described on this page would be useful for this?

4. Hashing

i) Suppose we want to hash the following strings into a hash table of size 10:

"Alice", "Bob", "Charlie", "Dan", "Eve", "Frank", "Gina", "Hal", "Ivan", "Jack"

Draw the hash table for each of the following hashing approaches:

- Chaining
- Linear probing
- Quadratic probing
- Double hashing (using the hash function $h_2(x) = 7 - (x \bmod 7)$)

ii) Suppose we want to store the following strings in a trie: "Alice", "Alicia", "Bob", "Bobby", "Bobbie", "Bobbie's", "Bobbie's dog", "Charlie", "Charlie's", "Charlie's dog"

Draw the trie.

iii) Suppose we want to store the following strings in a suffix trie: "Alice", "Alicia", "Bob", "Bobby", "Bobbie", "Bobbie's", "Bobbie's dog", "Charlie", "Charlie's", "Charlie's dog"

Draw the trie.

iv) Suppose we want to store the following strings in a suffix array: "Alice", "Alicia", "Bob", "Bobby", "Bobbie", "Bobbie's", "Bobbie's dog", "Charlie", "Charlie's", "Charlie's dog"

Draw the suffix array.

5. Graph basics

Consider the following graph:

i) Draw the graph's adjacency matrix.

ii) Draw the graph's adjacency list.

iii) Write down the order that the vertices would be visited in a DFS traversal starting from vertex 0 and visiting neighbors in ascending order.

iv) Write down the order that the vertices would be visited in a BFS traversal starting from vertex 0 and visiting neighbors in ascending order.

6. Dijkstra's algorithm

i) Consider the following graph:

a) Run Dijkstra's algorithm on the graph starting from vertex 0. Write down the order in which vertices are added to the set of vertices with finalized distances, and the distance to each vertex when it is added to the set.

b) Suppose we add a new edge from vertex 0 to vertex 4 with weight 1. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

c) Suppose we add a new edge from vertex 0 to vertex 4 with weight 10. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

d) Suppose we add a new edge from vertex 0 to vertex 4 with weight 100. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

e) Suppose we add a new edge from vertex 0 to vertex 4 with weight 1000. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

f) Suppose we add a new edge from vertex 0 to vertex 4 with weight 10000. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

g) Suppose we add a new edge from vertex 0 to vertex 4 with weight 100000. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

h) Suppose we add a new edge from vertex 0 to vertex 4 with weight 1000000. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

ii) Consider the following graph:

a) Run Dijkstra's algorithm on the graph starting from vertex 0. Write down the order in which vertices are added to the set of vertices with finalized distances, and the distance to each vertex when it is added to the set.

b) Suppose we add a new edge from vertex 0 to vertex 4 with weight 1. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

c) Suppose we add a new edge from vertex 0 to vertex 4 with weight 10. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

d) Suppose we add a new edge from vertex 0 to vertex 4 with weight 100. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

e) Suppose we add a new edge from vertex 0 to vertex 4 with weight 1000. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

f) Suppose we add a new edge from vertex 0 to vertex 4 with weight 10000. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

g) Suppose we add a new edge from vertex 0 to vertex 4 with weight 100000. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

h) Suppose we add a new edge from vertex 0 to vertex 4 with weight 1000000. Does the shortest path from vertex 0 to vertex 4 change? Why or why not?

7. Topological sort

i) Consider the following DAG:

a) Run Kahn's algorithm on the DAG to find a topological ordering of the vertices. Write down the order in which vertices are added to the topological ordering, and the number of incoming edges to each vertex when it is added to the topological ordering.

b) Run DFS-based topological sort on the DAG to find a topological ordering of the vertices. Write down the order in which vertices are added to the topological ordering, and the finishing time of each vertex.

c) Suppose we add an edge from vertex 4 to vertex 0. Does the topological ordering change? Why or why not?

d) Suppose we add an edge from vertex 0 to vertex 4. Does the topological ordering change? Why or why not?

e) Suppose we add an edge from vertex 3 to vertex 1. Does the topological ordering change? Why or why not?

f) Suppose we add an edge from vertex 1 to vertex 3. Does the topological ordering change? Why or why not?

ii) Consider the following DAG:

a) Run Kahn's algorithm on the DAG to find a topological ordering of the vertices. Write down the order in which vertices are added to the topological ordering, and the number of incoming edges to each vertex when it is added to the topological ordering.

b) Run DFS-based topological sort on the DAG to find a topological ordering of the vertices. Write down the order in which vertices are added to the topological ordering, and the finishing time of each vertex.

c) Suppose we add an edge from vertex 4 to vertex 0. Does the topological ordering change? Why or why not?

d) Suppose we add an edge from vertex 0 to vertex 4. Does the topological ordering change? Why or why not?

e) Suppose we add an edge from vertex 3 to vertex 1. Does the topological ordering change? Why or why not?

f) Suppose we add an edge from vertex 1 to vertex 3. Does the topological ordering change? Why or why not?

8. Minimum spanning trees

Consider the following graph:

i) Run Kruskal's algorithm on the graph to find a MST. Write down the order in which the edges are added to the MST, and the total weight of the MST.

ii) Run Prim's algorithm on the graph to find a MST. Write down the order in which the edges are added to the MST, and the total weight of the MST.

iii) Suppose we add an edge between vertex 0 and vertex 4 with weight 1. Does the MST change? Why or why not?

iv) Suppose we add an edge between vertex 0 and vertex 4 with weight 10. Does the MST change? Why or why not?

v) Suppose we add an edge between vertex 0 and vertex 4 with weight 100. Does the MST change? Why or why not?

vi) Suppose we add an edge between vertex 0 and vertex 4 with weight 1000. Does the MST change? Why or why not?

vii) Suppose we add an edge between vertex 0 and vertex 4 with weight 10000. Does the MST change? Why or why not?

viii) Suppose we add an edge between vertex 0 and vertex 4 with weight 100000. Does the MST change? Why or why not?

ix) Suppose we add an edge between vertex 0 and vertex 4 with weight 1000000. Does the MST change? Why or why not?

9. Debugging

Suppose you are the person responsible for writing a hash table that other people will be using to implement a spell-checker. Consider the following implementation:

```
class HashTable:
    def __init__(self, capacity):
        self.capacity = capacity
        self.size = 0
        self.table = [None] * capacity

    def __hash__(self, key):
        return hash(key) % self.capacity

    def __contains__(self, key):
        ...

    def __getitem__(self, key):
        ...

    def __setitem__(self, key, value):
        ...

    def __delitem__(self, key):
        ...

    def __iter__(self):
        ...

    def __str__(self):
        ...

    def __repr__(self):
        ...

    def __eq__(self, other):
        ...

    def __ne__(self, other):
        ...

    def __lt__(self, other):
        ...

    def __gt__(self, other):
        ...

    def __le__(self, other):
        ...

    def __ge__(self, other):
        ...
```

i) Consider the following implementation of the __hash__ function:

```
def __hash__(self, key):
    return 0
```

a) Suppose we insert the following strings into the hash table: "Alice", "Bob", "Charlie", "Dan", "Eve", "Frank", "Gina", "Hal", "Ivan", "Jack"

Draw the hash table.

b) Suppose we want to find the string "Alice" in the hash table. How many strings do we need to compare against "Alice" before we can determine that it is not in the hash table?

c) Suppose we want to find the string "Zoe" in the hash table. How many strings do we need to compare against "Zoe" before we can determine that it is not in the hash table?

d) Suppose we want to find the string "Alice" in the hash table. How many strings do we need to compare against "Alice" before we can determine that it is in the hash table?

ii) Consider the following implementation of the __hash__ function:

```
def __hash__(self, key):
    return len(key) % self.capacity
```

a) Suppose we insert the following strings into the hash table: "Alice", "Bob", "Charlie", "Dan", "Eve", "Frank", "Gina", "Hal", "Ivan", "Jack"

Draw the hash table.

b) Suppose we want to find the string "Alice" in the hash table. How many strings do we need to compare against "Alice" before we can determine that it is not in the hash table?

c) Suppose we want to find the string "Zoe" in the hash table. How many strings do we need to compare against "Zoe" before we can determine that it is not in the hash table?

d) Suppose we want to find the string "Alice" in the hash table. How many strings do we need to compare against "Alice" before we can determine that it is in the hash table?

iii) Suppose we want to store strings in a trie. When we insert a string into the trie, we create a node for each character in the string. When we search for a string in the trie, we traverse the trie character by character until we reach the end of the string. If we reach a node that does not have a child for the next character, we know that the string is not in the trie.

Suppose we want to find the longest common prefix of two strings. We can do this by traversing the trie character by character until we reach a node that does not have a child for the next character. The longest common prefix is the string that we have traversed so far.

Suppose we want to find the longest common prefix of two strings. We can do this by traversing the trie character by character until we reach a node that does not have a child for the next character. The longest common prefix is the string that we have traversed so far.

a) "BOTTOM" "BOTTLE"

b) "BOTTOM" "BOTTOM"

c) "BOTTOM" "BOTTOMS"

d) "BOTTOM" "TOP"

iv) Suppose we want to store strings in a suffix array. When we insert a string into the suffix array, we create a node for each character in the string. When we search for a string in the suffix array, we traverse the suffix array character by character until we reach a node that does not have a child for the next character. If we reach a node that does not have a child for the next character, we know that the string is not in the suffix array.

Suppose we want to find the longest common prefix of two strings. We can do this by traversing the suffix array character by character until we reach a node that does not have a child for the next character. The longest common prefix is the string that we have traversed so far.

Suppose we want to find the longest common prefix of two strings. We can do this by traversing the suffix array character by character until we reach a node that does not have a child for the next character. The longest common prefix is the string that we have traversed so far.

a) "BOTTOM" "BOTTLE"

b) "BOTTOM" "BOTTOM"

c) "BOTTOM" "BOTTOMS"

d) "BOTTOM" "TOP"

10. graphs and design

Consider the following problem:

You are planning a road trip from your house to your friend's house. You want to take the shortest possible route, but you also want to make sure that you will be able to find a place to stay in Kingston along the way.

You are given a map of the road network, represented as a weighted, undirected graph $G = (V, E)$. Some of the vertices in V are marked as "Kingston", meaning that there are hotels in Kingston at those locations. You are also given a number k, representing the number of miles you can drive before you need to find a place to stay. Your goal is to find the shortest path from your house to your friend's house, such that you never drive more than k miles without passing through Kingston.

Suppose you have a black box function `dijkstra(G, s)` that takes in a graph G and a vertex s, and returns a dictionary `d` such that `d[v]` is the shortest distance from s to v. For example, if `d[v] == 5`, then the shortest path from s to v is 5 miles long. If `d[v] == inf`, then there is no path from s to v.

i) Describe a way to use this black box to solve the problem. Your solution should be a single number, which represents the shortest path from your house to your friend's house, such that you never drive more than k miles without passing through Kingston. If there is no such path, your solution should be `inf`.

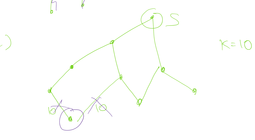
ii) Describe a way to use this black box to find the actual path from your house to your friend's house, such that you never drive more than k miles without passing through Kingston. If there is no such path, your solution should be `None`.

iii) Describe a way to use this black box to find the shortest path from your house to your friend's house, such that you never drive more than k miles without passing through Kingston, and you never drive more than k miles without passing through any other city. If there is no such path, your solution should be `None`.

iv) Describe a way to use this black box to find the shortest path from your house to your friend's house, such that you never drive more than k miles without passing through Kingston, and you never drive more than k miles without passing through any other city, and you never drive more than k miles without passing through any other city. If there is no such path, your solution should be `None`.

a)

b)



c)

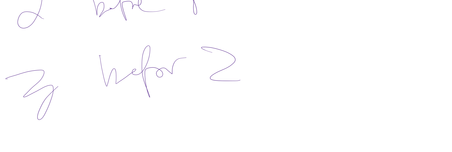
d)

$+ (V-2) + (V-3) + \dots$

$\frac{V(V-1)}{2}$

α before γ

β before ζ



11. X and Y#

Consider the following question:

Given two strings X and Y, is X a subsequence of Y?

For example, if X = "abc" and Y = "abcdefghijklmnopqrstuvwxyz", then X is a subsequence of Y because "abc" appears in Y in order.

Suppose we have a black box function `is_subsequence(X, Y)` that takes in two strings X and Y, and returns `True` if X is a subsequence of Y, and `False` otherwise.

i) Describe a way to use this black box to solve the original problem: given two strings X and Y, is X a subsequence of Y?

ii) Suppose instead of a single string Y, we have a collection of strings $Y_1, Y_2, \dots, Y_n$. Describe a way to use this black box to find the longest string in the collection that X is a subsequence of.

iii) Suppose instead of a single string X, we have a collection of strings $X_1, X_2, \dots, X_m$. Describe a way to use this black box to find the longest string in the collection that is a subsequence of Y.

iv) Suppose instead of single strings X and Y, we have collections of strings $X_1, X_2, \dots, X_m$ and $Y_1, Y_2, \dots, Y_n$. Describe a way to use this black box to find the longest string that is a subsequence of some string in the collection $X_1, X_2, \dots, X_m$, and also a subsequence of some string in the collection $Y_1, Y_2, \dots, Y_n$.

12. Mad science

Consider the following scenario: Dr. Frankenstein's lab has a collection of n body parts, numbered $1, 2, \dots, n$. Each body part i has a "quality" q_i. Dr. Frankenstein's goal is to create a new body by stitching together some of the body parts, such that the total quality of the new body is at least k. Stitching together body parts i and j costs $|q_i - q_j|$.

Dr. Frankenstein's lab also has a collection of m "stitch-ups", numbered $1, 2, \dots, m$. Each stitch-up i has a "quality" q_i. Dr. Frankenstein's goal is to create a new body by stitching together some of the body parts, such that the total quality of the new body is at least k. Stitching together body part i and stitch-up j costs $|q_i - q_j|$.

Dr. Frankenstein's lab also has a collection of m "stitch-ups", numbered $1, 2, \dots, m$. Each stitch-up i has a "quality" q_i. Dr. Frankenstein's goal is to create a new body by stitching together some of the body parts, such that the total quality of the new body is at least k. Stitching together stitch-up i and stitch-up j costs $|q_i - q_j|$.

i) Describe a way to use this black box to solve the original problem: given a collection of n body parts, is it possible to create a new body with total quality at least k?

ii) Describe a way to use this black box to find the actual path: what is the minimum cost way to create a new body with total quality at least k?

iii) Suppose instead of a single collection of n body parts, we have m collections of body parts. Describe a way to use this black box to find the minimum cost way to create a new body with total quality at least k, using one body part from each collection.

iv) Suppose instead of a single collection of n body parts, we have m collections of body parts. Describe a way to use this black box to find the minimum cost way to create a new body with total quality at least k, using one body part from each collection, and also one stitch-up.

v) Suppose instead of a single collection of n body parts, we have m collections of body parts. Describe a way to use this black box to find the minimum cost way to create a new body with total quality at least k, using one body part from each collection, and also one stitch-up from each collection.

vi) Suppose instead of a single collection of n body parts, we have m collections of body parts. Describe a way to use this black box to find the minimum cost way to create a new body with total quality at least k, using one body part from each collection, and also one stitch-up from each collection, and also one stitch-up.

Formulas

Arithmetic series

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

Geometric series

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1$$

Logarithm identity

$$\sum_{i=0}^{n-1} 2^i \cdot i = 2n\log_2(n) - 2n + 2$$

Exponentiation identity

$$(a^b)^c = a^{bc}$$

Multiplication identity

$$a \times b = a^{\log_2(b)}$$

Change of base formula

$$\log_2(n) = \frac{\log_3(n)}{\log_3(2)}$$

Master theorem

$$T(n) = aT(n/b) + f(n)$$

- If $f(n) = O(n^{\log_b(a) - \epsilon})$, then $T(n) = \Theta(n^{\log_b(a)})$
- If $f(n) = \Theta(n^{\log_b(a)})$, then $T(n) = \Theta(n^{\log_b(a)} \log(n))$
- If $f(n) = \Omega(n^{\log_b(a) + \epsilon})$, then $T(n) = \Theta(f(n))$