CSE 101
Discussion Section
Week 3

January 22 - January 29

Important

Homework solutions:

- Any solution to a problem must include (if not stated otherwise) high-level description, pseudo-code, formal proof of time complexity and formal proof of correctness
- Any used resources must be properly cited
- Please refer to model solutions at the end of the website for examples

Introduction

- Master Theorem
- The lower-bound on searching an ordered list
- Constructive induction
- BFS and Dijkstra applications

Master Theorem

Let for some $a \geq 1$ and $b > 1$:

$$T(1) = \theta(1)$$
$$T(n) = aT(\frac{n}{b}) + \theta(n^d) \text{ for } n > 1$$

Then:

- If $a < b^d$, then $T(n) = \theta(n^d)$
- If $a = b^d$, then $T(n) = \theta(n^d \log(n))$
- If $a > b^d$, then $T(n) = \theta(n^{\log_b(a)})$

In the lectures you had $\mathcal{O}$ instead of θ. Both variations are true.

Master Theorem. Example

The binary search algorithm compares the element x with the middle element a_m of a given ordered list. If $x = a_m$, we return index m. If $x < a_m$, we look for x in the first half of the list, otherwise in the second half. In the worst case we will have to divide the list till its size becomes one.

Thus, we have:

$$T(1) = \theta(1)$$
$$T(n) = T(\frac{n}{2}) + \theta(1) \text{ for } n > 1$$

Here we have $a = 1, b = 2, d = 0$. Thus, $a = b^d$ $(1 = 2^0)$

We conclude that $T(n) = \theta(n^0 \log(n)) = \theta(\log(n))$.

Thus, the worst case for the binary search is $\theta(\log(n))$.

The lower-bound on searching an ordered list

- We are given some element x and an ordered list of n elements $A = (a_1, a_2, ..., a_n)$, such that $a_1 < a_2 < ... < a_n$. We need to output the index i of the element a_i, if $a_i = x$, for some i.
- Example: $x = 7$, $A = (2, 3, 5, 7, 11, 13, 17)$. The output is 4.

The lower-bound on searching an ordered list

Let's find the lower-bound on searching an ordered list in the same way as we did for sorting:

Let's construct a decision tree. For each element a_i there is an internal node which compares it to the element x:

- if $x = a_i$, there is an edge to a leaf with label i.
- if $x < a_i$, then there is an edge to the "left" child
- if $x > a_i$, then there is an edge to the "right" child

Each internal node can lead to at most 2 other internal nodes. Thus, at level k of this decision tree, there will be no more than 2^k internal nodes.

The lower-bound on searching an ordered list

The number of internal nodes of a such decision tree of height k is:

$$2^0 + 2^1 + ... + 2^{k-1} = 2^k - 1$$

We must have at least n internal nodes in the decision tree (for every a_i). Otherwise, we can come up with a list, such that $a_i = x$ and there is no internal node for a_i. The algorithm will return a wrong answer.

Thus, we have:

$$2^k - 1 \geq n$$

$$k \geq log(n+1)$$

The lower-bound on searching an ordered list

We have shown that the number of comparisons should be at least $\log(n+1)$.

Thus, any searching algorithm over an order least must perform at least $\log(n+1)$ operations for some input.

We conclude that the lower-bound on searching an ordered list is $\Omega(\log(n))$.

Therefore, the binary search algorithm is optimal for this task.

Constructive induction

We can solve recurrence relations by guessing their closed forms and then proving their correctness using mathematical induction.

For example, let's say we have the following recurrence relation:

$$f(0) = 0$$
$$f(n) = f(n-1) + n \text{ for } n > 0$$

We can guess its closed form as $\frac{n(n+1)}{2}$ (the sum of the first n positive integers) and use mathematical induction (week) to prove it.

Constructive induction

But sometimes guessing the exact closed form may be hard.

In this case we may try to guess *the general form* of the answer and then find its exact form using mathematical induction.

Constructive induction

Let's consider the same example again:

$$f(0) = 0$$
$$f(n) = f(n-1) + n \text{ for } n > 0$$

At each step we add a number of an order of $O(n)$. There are $\theta(n)$ steps. The closed form should have an order of $\theta(n^2)$.

We may assume that the general form of the closed form will be quadratic, i.e. $an^2 + bn + c$, where a, b, c are some real constants.

Constructive induction

We will reproduce the steps of the proof with mathematical induction as if the exact form of the closed form was known to us. Then we will find the constants a, b, c.

Construtive induction

- *Base case:* $f(0) = 0$, thus $a \cdot 0 + b \cdot 0 + c = 0$. For this equation to be true we must have $c = 0$. We have found the value for constant c.
- *Inductive hypothesis:* For some $n - 1 \geq 0$, let $f(n-1) = a(n-1)^2 + b(n-1) + c = a(n-1)^2 + b(n-1)$
- *Inductive Step:*

$$f(n) = f(n-1) + n$$
$$an^2 + bn + 0 = a(n-1)^2 + b(n-1) + n$$
$$an^2 + bn + 0 = an^2 + n(-2a + b + 1) + a - b$$

Construtive induction

As the result we have:

$$a - b = 0 \text{ and } b = -2a + b + 1$$

$$a = b \text{ and } b = -2b + b + 1$$

$$a = b = \frac{1}{2}$$

If we want the above proof to be true our constants a, b, c must have values $\frac{1}{2}, \frac{1}{2}, 0$ respectively. Thus, the closed form for our recurrence relation is: $an^2 + bn + c = \frac{1}{2}n^2 + \frac{1}{2}n = \frac{n(n+1)}{2}$.

Construtive induction

During the lectures we have covered DQ Selection algorithm and supposed that its time complexity satisfies: $T(n) \leq dn + T(\frac{3n}{4})$ for some constant d (Refer to slide no. 14 of Lecture 5 (ABK) or slide no. 50 of Lecture 5 (Miles) on the course website).

To prove this statement we guessed that $T(n) \leq kn$ and used strong mathematical induction. We assumed that $T(m) \leq km$ for all $m \leq n-1$ and then showed that:

$$T(n) \leq dn + T(\frac{3n}{4}) \leq dn + k\frac{3n}{4} \leq kn$$

for all $k \geq 4d$. Thus, $T(n) \leq 4dn$, which implies $T(n) = O(n)$.

Construtive induction

It may be impossible to apply Master Theorem or guess the exact form of the recurrence relation. For example:

$$T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + \theta(n)$$

It is possible to prove that $T(n) = \theta(n)$ using constructive induction.

BFS and Dijkstra applications. Problem 1

Statement. You are given an undirected unweighted graph $G = (V, E)$ and two nodes v and u in this graph. Output all nodes in V that belong to some shortest path from u to v.

Observations.

- The graph is unweighted, thus, the weights of all edges are equal to one.
- There may exist several shortest paths from u to v.
- We can find the shortest distances from u and v to all other nodes with BFS.

BFS and Dijkstra applications. Problem 1

Let $d(x, y)$ be the shortest distance from node x to node y. It equals ∞, if there is no path between them.

Key observation. The node x belongs to some shortest path from u to v if and only if $dist(v, x) + dist(x, u) = dist(v, u)$. (Why?)

Solution. Find the shortest distance from u and v to all other nodes with two lunches of BFS. Then for every node x in V, output x, if $dist(v, x) + dist(x, u) = dist(v, u)$.

Time complexity. $\mathcal{O}(|V| + |E|)$

BFS and Dijkstra applications. Problem 2

Statement. You are given a directed weighted graph $G = (V, E)$ with positive weights and a node t in this graph. Find node s in V, such that the shortest distance from s to t is as large as possible. If there is no such, node output -1.

Observations.

- The graph has positive weights, thus, we can use Dijkstra's algorithm to find the shortest paths from a single source.
- The naive solution is to lunch Dijkstra's algorithm from each node s in V and choose that node s, such that $d(s, t)$ is the largest.
- The naive solution would take $\mathcal{O}(|\mathcal{V}|^{\ni})$ time in the worst case. Can we do better?

BFS and Dijkstra applications. Problem 2

Key observation. Construct a reversed graph G^R. The shortest distance from t to s in G equals the shortest distance from s to t in G^R. (Why?)

Solution. Find the shortest distance from t to all other nodes in G^R. Choose the node s, such that $d(t, s)$ is the largest. If there is no such node, output -1.

Time complexity. $\mathcal{O}(|V|^2)$