ALGORITHMS AND DATA STRUCTURES

CSCI 382

Author
Who?
Where?
When?

Fall 2023

Contents

1 Algorithms

1.1 How to write

1.2 How to solve

1. reduce to
   (a) sorting
   (b) searching
2. design new algo
   (a) brute force
   (b) divide and conquer

1.3 shortest path algos

1. graph, find shortest ways to get there

2 Running time

"How to count"

2.1 Big O

2.2 Big Ω

2.3 Big Θ

3 recurrences

4 models of computation

1. which operations are O(1)
2. word RAM models
3. comparison model

5 Data Structures

5.1 Vocabulary

1. interface:
2. implementation
3. array-based (example: array/list)
4. static: region of memory stays static
5. pointer-based (ex: linked list)
6. dynamic

5.2 Static Sequence

1. Stores elements $x_0, \ldots, x_{n-1}$

5.2.1 Operations

1. `build(X)` builds the sequence from collection of elements `X`.
2. `iterseq()` output items in order
3. `code`
4. `len()`, returns `n`
5. get at index, returns thing at index
6. set at i, val, sets val at location I

5.3 Dynamic Sequence

5.3.1 Operations

All carry over from static sequences, but we include

1. `insertAt(i, x)` which inserts item x at location I
2. `deleteAt(i)` deletes item at location i and returns it

5.4 Sets

5.4.1 Operations

Static:

1. `build(X)`

5.5 Direct access table

5.5.1 Example

items: Reed students
keyspace: 8 digit reed id
make table: 0 - $10^8 - 1$
index = Reed ID #
10 to the 8th -1 is way too big for Reed, wasting space
but like that findk is O(1)

5.6 Hash tables

Before: sorting and searching using only comparisons on items.
Now: use keys in more complex ways arrow direct access table

```
h : {ids} -> {0, ... , m-1}
# assume m = Theta(m)
```

store items as determined by h(key)
cannot be a bijection
collision: things in ids that map to same idem in codomain
solution: instead of storing item in table, store pointer to another data structure
“Chaining” to some other data structure

```
def hash.find(k):
digest = h(k)
c = get(h.direst) -> go find k?
```

5.6.1 Running time

ideally: all chains are constant size so then all operations on chains are constant size

chose m,h such that all chains end up with about n/m approx O(1) things in them

5.6.2 Example

modular division: `h(k) = k mod(m)`.
Problem: only as even as keys themselves
if too even, might get mapped to same thing

5.6.3 Better example: universal family of hash functions

example of one family:
defien $h_{ab}(k) = (((ak + b) \mod p) \mod m)$ where p is a large prime and a, b chosen randomly from 0 to p-1
Formally, define family $\mathcal{H}(p, m) = \{h_{ab}(k) \mid a, b \in [0 \dots p-1] \text{ and } a \neq 0\}$ desired property: for some $h \in \mathcal{H}$, Pr that any pairs of keys collide $[h(k_1) = h(k_2)] \leq \frac{1}{m}$ for all k1, k2 such that k1 not equal k2
parallel concept for sorting comparison model
lower bound Omega (n log n)

5.7 Binary Trees

6 Running Time

7 Correctness

7.1 Loop invariants

8 Recursion

9 Limits

9.1 Limit Laws

10 Sorting

10.1 Vocab

10.2 Permutation Sort

sorting

10.3 Selection Sort

10.4 Comparison

10.5 Merge Sort

```
if n = 0, 1: done
otherwise:
    split array into Left, Right
    MergeSort(Left), MergeSort(Right)
    Merge(Left, Right)
```

1. specificity: array A, p starting point, r ending point
2. if p geq r (if p equal to r, then only one element and return)
3. convention: A[a:b] = [] if b less than A
4. a:b means including both end points
5. define midpoint $q = (p + r)/2$ (floor of the averge)
6. Left = A[p:q]
7. Right A[q+1:r]
8. MergeSort(A, p, q)
9. MergeSort(A, q+1, r)
10. Merge(A, p, q, r)

1. comparisons in constant time?
   (a) mostly comparing integers

(b) assume $i \leq j$ in constant time

(c) might be more complex

(d)

11 Sept 25th

11.1 review

1. direct access Arrays

2. hash tables

12 Binary Search Trees

O(h) time where h is height
usually use these for sets
example: set us cs profs
name/id
adam/560
charlie/703
erica/998
greg/997
jim/100
include bst from notes
properties: for any X, keys in x.left leq x.key leq keys x.right
can also store a sequence in this way
ex 2: grocery list
apples, bananas, cereal, dish soap, eggs
order doesn't matter
bst property: for any L[i], all items in L[i].left appear before L[i]
all items in right subtree appear after L[i] is balanced, height is at most O(log n)
when balanced, called red-black trees

12.1 non-modifying operations

1. find min - all the down left

2. find max - all the way down right

3. kind k - compare at every node

4.

12.2 modifying properties

1. insert item: traverse, then insert as a leaf

2. delete: find successor and swap 16 with 17, then remove old leaf (find a safe leaf to remove)

3.

balanced if even under dynamic operations it maintains height O log N
red-black tree

1. dummy nodes
2. imagine single dummy node ”tree ends”
3. stop drawing them

loop invariants
different cases for red black trees

13 priority queues and heaps

13.1 Priority Queue Interface

new: delete max - remove item with highest priority
find max = find highest priority item
supports subset of set interface operations
optemized for finding min or max, we will focus on max priority
also has insert and build

13.2 priority queue sort

algo in slides
simple because you make the ds do it

13.3 binary heaps

13.3.1 infer tree from aray

require: completely full in upper levels (until last level)
binary trees with n things -¿ arrays of length n
bottom level should be left justified
diferent than traversal ordering function
root is index 0
finding the left child of i is just at index 2i+1
right(i) = 2i+2
parent(i) = floor((i-1)/2)
max heap properties:

1. at node l, the value of thing at position l is going to be at least the value its children

2.

tree does not live in memory, just array
we want higher importance nodes to be closer to the top
bottom layer is less than all elements above it
example 1 from slides: not a max heap: 15 is a node that is is not greater than or equal to both of its children

inserting(x)

-append x to array: whole function is in O(logn)
-fix problems in heap
fixing: swap up with parents until max heap rule is followed:
check that val of parent geq val node, if no, swap and keep going
delete-max():
this deletes the root!
1. swap root with item at node number n-1, then delete thing at n-1
the fix downwards:
- start at root
- swap it with the greater key in its children, then continue
this is o of log n time

14 Cheat Sheets

14.1 Running Times for DS

Operations $O(\cdot)$				
Set Data Structure	Container	Static	Dynamic	Order
array	AX	ALA	248	
sorted array	AL	ALB	008	

14.2 orders

15 servers

1. servers and clients
2. each server: some capacity
3. each client: some workload
4. assign so that no server goes over its capacity

implementation

1. initialize(X) whre X is set of servers
2. connect(client, workload)
3. disconnect(client)
4. clients-of(server): return list of clients connected to that server

how to do:
idea
use priority queue for clients and prioritize biggest
priority queue for servers so that you can find best fit the best

notes on actual:
server has set of clients where key is client id and value is workload
priority queue storing the servers
store based on priorty, which is available capacity

structures:
pq P:(server.id, available capacity)
set S: (maps server id to set of clients and pointer to si in P) set C: maps client id to connected server and workload (if not stored as client object)

16 Graph algorithms

later: get chapters

16.1 Definition

graph $G = (V, E)$
can use for state transition diagram
in this class, simple graph, so for all u,v in edges, u not equal to v
no duplicate edges. each edge in E is unique
common to talk about outgoing and incoming neighbor set for directed and regular neighbor set for undirected.
outgoing edge neighbor of u Adj+(u) = set(v in V such that (u, v) in E)
incoming neighbord of u Adj - (u) = set(v in V st (v, u) in E) out degree(u) = size of adj+ in degree(u) = size adj -

16.2 Running time

1. linear means O(V + E) (size of those sets)

    doing something with each edge and each vertex

2. $|E| = O(V^2)$ for simple graphs

    most edges you can have is fully connected graph

3. undirected: —E— leq size of V choose 2 (proved on homework)

4. directed : —E— leq 2 times (V size chose 2)

16.3 how to represent

1. adj list

2. adj matrix

abstract:
top level set adj of vertices
lower level: adj lists Adj(u) either set or sequence (default to outgoing
can be direct access array
)

16.4 breadth-first search

16.5 path problems

17 Feedback

17.1 Problem set 1

1. Missing justification of $n \leq (logn)^(logn)$

2. Give a specific expression for the new n_0, c_1, c_2 in terms of the corresponding constants for f and g. Also, note that the definition of f(n) in O(g) (or Theta(g), or Omega(g)) only tells us that there exist constants such that the inequality holds, not necessarily equality. So in the case of O(g), we know there exist c, n_0 s.t. $0 <= f(n) <= cg(n)$ for all $n >= n_0$, but it is not necessarily true that there exists c, n_0 s.t. $f(n) = cg(n)$ for all $n >= n_0 (<= vs =)$.

17.2 Problem set 2

17.3 Problem set 3c