Monday, February 27

Huffman: Priority Queue->Huffman Tree->Code Table->Dump Tree->Emit Codes->Reconstruct Tree->Decode by bit

Lempel-Ziv Coding (LZ78)

- Dictionary of words
    - Best case: words with long, common prefixes
    - Store with a trie
    - Minimize redundancies
- LZ78 encoding
    - First node empty
    - Check if first symbol is in current node
        - Not: add child
        - There: step down so current node is that symbol
    - Back to root
    - Creates pair of symbol and the node before
    - Goes into a dictionary
- Lossless

Debugging and Testing

Bug: error/flaw that produces wrong output

Kinds of bugs

- Syntax errors: forgetting parentheses, missing semicolon
- Logical errors: off by one, operator precedence, times 2 instead of div 2 in binary
- Semantic errors: adding float to string, returning non void in void function

Easy bugs: syntax

- To fix
    - Examine most recent changes
    - Fix all occurrences
    - Fix crashes right away
    - Stack traces help debug var name errors

Hard bugs: location hard to find

- Finding them
    - assert() statements
    - Print statements
        - fflush() to flush buffered data
    - Play around with inputs/parameters
    - Write test harness
    - Debug tools

assert()

- Verify pre and postconditions
    - Precondition: must be true before execution
    - Postcondition: true after execution
- Can be turned off during compile time
    - #define NDEBUG or -DNDEBUG compile flag

- Boolean expression

Printing
- Variables
- Reach certain place

Scan-build
- Static analyzer
- Find bugs with running (works at compile time)
- Overrides CC environment var to build with fake compiler
- Static analyzer then executes to analyze
- Running it
    - Clean, scan-build make
    - Html bug repot with scan-view

Infer
- Easier with Makefile
- Two steps
    - Infer captures result of compiling
    - Infer analyze result of compiling

Valgrind
- Dynamic analyzer (works during runtime)
- Reports location and size of mem leaks
    - Also invalid reads and writes
- Invalid read: tried to read outside available mem address
- Invalid write: tried to write outside availnale mem address

Static vs dynamic
- Some bugs can only be caught by the diff kinds

lldb
- Higher performance debugger
- Set breakpoints and step through line by line
- Examine values and addresses of variables
- -g flag
- Not good for mem leaks
- Running it
    - Compile with flag, call lldb program
    - Print values with p

Lldb vs gdb
- Gdb is GNU, portable

Wednesday, March 1

Bottleneck
- Find what is taking the most time to then make it faster by fixing that one thing

Gprof
- Find that bottleneck
- Gathers timing info and stats about program
- Used to profile program performance and identity bottlenecks

- Req your program to be linked with -pg linker flag
    - Req each C-file compiled with -pg flags too

Profiling with gprof
- Creates gmon.out
- grof <program>
- No command line
- gprof prints the profile

What is the make program
- Automatically builds executables and libraries
- Gmake (GNU make, what we use)

What is a makefile
- Plaintext with syntax
- Like a script

Rules
- target-dependencies-commands

Phony target
- Doesn't produce file with the same name, but it does an action
- Makes it not part of the build process

Flags
- d: print debug info
- f: specify file to be read as Makefile
- I: specify dir

Variables
- = lazy assignment, thing on left is assigned the text, stored as-is
- := immediate assignment, assigns the value
- ?= conditional, assign if not already assigned
- += concatenation
- Use value of var $(var)

Dependency
- target/file name
- Ex) build: clean hello.o

Topological order
- Targets as vertices and dependencies as edge in a acyclic graph
- Depends on something before

Command
- Action to be executed
- Shell script commands
- More than one command each on own line
- Tab in front of command

Compilers and flags
- CC: c compiler to be used
    - cc, gcc, clang
- CFLAGS: compiler flags

        - Wall, Wextra, Wpedantic, Werror, g
    - OBJ: object files to build and link
    - SRC: .c source files

Automatic vars

- $@ name of target
- $^ list of all dependencies for target
- $? list dependencies more recent than target
    - Just recompiles the thing that was changed
    - foo: foo.c
        - gcc $? -o $@
- $< name of first dependency

Shell function

- Communicates with world outside of make
- Does shell command and evals the output
- SRC := $(shell ls *.c) → all c files in current directory
- This can also be done with wildcard

Wildcard

- * operator expanded by shell
- Source files with wildcard: SRC := $(wildcard *.c)
- Globbing

Patsubst

- Pattern substitution
- $(patsubt pattern, replacement, text)
- Ex OBJ := $(patsubt ……)

* vs %

- * expansion
- % pattern match placeholder
- Ex %.c=%.o

Pattern matching

- %.o: %.c
    - gcc -c $< -o $@

Recursive make

- Subsystem:
    - $ (MAKE) -C subsystem

Include common.mk

Friday, March 3

Graphs

Network routing

- The internet is a graph: nodes are computers and edges are connections
- Routers are nodes with many edges

A formal definition

- G = <V, E>
- V = {vertices}

- E = {edges}
- Each edge is a pair of vertices <vi, vj>

Directed and undirected

- Directed: edges have a direction in one way
- Undirected: both directions
- Edge weights: capacity, strength, cost

Representing a graph

- Adjacency matrix
    - nxn matrix
    - Binary: edges present or absent
    - Weighted: n != 0
    - Symmetric along diagonal → undirected graph
    - O(n^2) space
- Adjacent list
    - Column array for nodes
    - Linked list of edges from each node
    - May have weight
    - More useful/efficient for incomplete graph (sparse graph)
    - Each node represented as entry in col vector
        - Each entry is the head of a LL
        - List has destination node and weight of edge

Graph in c for matrix

- Adding edge is O(1)
- Checking existence of edge is O(1)

complete graph has n^2 edges

Graph in c for list

- Linked lists
- Add edge O(1)
- Checking for edge traverse entire list O(n)

Basic algorithms

- Searching
    - Breadth first search
        - Use a queue
        - Explore vertices immediately reachable, repeat each vertex
        - Level order traversal
    - Depth first search
        - Recursion or stack
        - Search as far as possible before backing up
        - Showcase iterative DFS using a stack

Trees

- Acyclic graph

        - Means you follow edges, no loops
    - DAG: directed acyclic graph

BFS and DFS (top of stack is rightmost element)

Topological sort

- Used for ordering of dependencies, Makefile
- Using DFS
    - Finished vertex is prepended to a list
- Ordering not unique → more than one valid ordering, partial ordering

Modified DFS to topological sort with recursion

Kahn's algorithm

- DAG input
- List with topological ordering output

Single-Source Shortest Paths (SSSP)

- Want shortest path from source vertex to any v
- SSSP algorithms
    - Bellman ford
    - Dijkstra's

Dijkstra's

Hamiltonian path

- Path is undirected or directed graph that visits each vertex once
- start and end at origin

Eulerian path

- Visits each edge exactly once
- Start and end at origin