ECE 590.01
C++ Programming, Data structures, and Algorithms

Algorithm Classes, Other DSes, etc..
Admin

• Reading
  • Chapter 10
  • Last Chapter to read!
    • Covers a bunch of other good-to-know stuff

• Homework 4
  • Due April 5

• Project
  • Due April 12
  • In class: 10 min demos to everyone last week
    • Show off what you did
    • “Normal” demos w/ Tas: same week
Talk next week: you should go!

- Todd Austin is giving a talk next week
  - Distinguished lecture series
  - Thurs 3/4 @ 3:15
  - Hudson 125

- Attend if you can
What have we been talking about?

• What did we talk about last time?
What have we been talking about?

- What did we talk about last time?
  - Graphs
    - MSTs
      - Prim’s
      - Kruskal’s
    - SCCs
      - Good use of DFS
Now, Classes of Algorithms

- We have seen a lot of algorithms and data structures
  - Can group some of these into general classes
  - Useful to help think of design of new algorithms

- Also, a few techniques we haven’t seen before
Brute Force

• One class of algorithms: Brute Force
  • Try all possibilities
  • Generally quite slow (exponential time)
  • Not NP complete? Can probably do better

• Examples:
  • NP complete problems (TSP, Graph coloring)
  • Simplistic Password Cracking
    • Note: realistically might do something else if we have hashes
Greedy Algorithms

• Another class: greedy algorithms
  • Take the best choice right now
  • Commit to that choice (no backtracking)

• One example:

• Want to reach a specified amount with fewest coins
  • Quarter (25 cents)
  • Dime (10 cents)
  • Nickel (5 cents)
  • Penny (1 cent)

• How do you get 66 cents?
Greedy Algorithms

- Want to reach a specified amount with fewest coins
  - Quarter (25 cents)
  - Dime (10 cents)
  - Nickel (5 cents)
  - Penny (1 cent)
- How do you get 66 cents? Choose biggest coin that fits.
  - $25 \leq 66 \rightarrow \text{Quarter + 41 cents}$
  - $25 \leq 41 \rightarrow \text{Quarter + Quarter + 16}$
  - $10 \leq 16 < 25 \rightarrow \text{Quarter + Quarter + Dime + 6 cents}$
  - $5 \leq 6 < 10 \rightarrow \text{Quarter + Quarter + Dime + Nickel + 1 cent}$
  - $1 \leq 1 < 5 \rightarrow \text{Quarter + Quarter + Dime + Nickel + Penny}$
Greedy Algorithms

- Some problems: greedy produces optimal solution
  - Making change with 25/10/5/1
  - A bunch of other algorithms we have seen (think...)

- Some problems: greedy produces solution, may not be optimal
  - TSP: Can do greedy, may get very bad results

- Some problems: greedy may not produce any solution, even when one exists
Greedy Algorithms

- What other problems have we seen with an optimal greedy solution?
Greedy Algorithms

• What other problems have we seen with an optimal greedy solution?
  • MST: Prim’s + Kruskal’s
  • Shortest Path: Dijkstra’s
  • Compression: Huffman Coding
  • Find min/max of BST: always go left/right
  • ...

ECE 590.01 (Hilton): Algs II, Other DSees, etc
Closely related: Best so far, trade up

- Closely related to greedy: remember the best so far, trade up when you see better
  - Find min/max of list
  - Find largest not larger than X on a BST

- Generally “find most extreme thing meeting some condition”
Greedy Algorithms: Local Optima

- Greedy algorithms that don’t produce the best solution may fall into a “local optima”
  - Looks best locally, but not globally

I’m on top of the world!
Genetic Algorithms

- Genetic Algorithms try to evolve solutions
  - Random mutations perturb solution out of local maxima

- Gist of genetic algorithms
  - Describe fitness function for solution
    - How good is it?
    - Solutions that don’t meet constraints should get negative value
  - Create random initial population
  - Cull the heard:
    - Keep fittest solutions
  - Breed them:
    - Mix pieces from best solutions
    - Introduce some small random mutations
      - Insert Zerg joke here
  - New population: repeat
Genetic Algorithms

• Good for large solution spaces
  • Generally many variables
  • Too big to explore by brute force
  • Too many local maxima much worse than global to trust greedy
  • Too complex to have clear cut algorithm to find winner

• We aren’t going to do any, but good for you to know they exist
  • And what they are good for
Divide and Conquer

- Split problem into smaller sub-problems (e.g., half)
  - Solve sub-problems
  - Combine results

- Where have we seen this?
Divide and Conquer

• Split problem into smaller sub-problems (e.g., half)
  • Solve sub-problems
  • Combine results

• Where have we seen this?
  • Mergesort

• Note that just because you split the problem in half does not mean you are more efficient
  • Find max of array?
    • Cut in half..
    • Find max of each half (recursively)
    • Select max of two halves?
  • Doesn’t help
Divide and Conquer

- Your book has some more complex examples... and a lot more math
  - Good to read and understand the problems and how D+C works
  - Math: less important for this class... Good to know overall (esp if you take an alg class)
Dynamic Programming

• Important class of algorithms: dynamic programming
  • Have not seen this one yet

• Sometimes sub-problems overlap
  • Remember solution to sub-problems to avoid re-computing that
  • Saves a lot of time
  • Example of memoization [not memorization]
    • Remember (cache/store) answers to problems you have seen

• Example: Diagonal Line Problem
Diagonal Line Problem

- Have MxN matrix of booleans (black = true, white = false)
  - Want to find length of longest diagonal (top left to lower right) which is all “trues”
  - Diagonal must most strictly right/down, but does not need to be contiguous
Diagonal Line Problem

- This path is valid (and length 5)
Diagonal Line Problem

- This path is invalid
  - One red x is down, but not right of the previous
  - The other is right, but not down from the previous
Diagonal Line Problem

- By the way, this problem has applications in similarity analysis
  - The rows represent pieces of one thing
  - The columns represent pieces of the other
  - 1 = same, 0 = different (could also use scale of “how different”)
Diagonal Line Problem

- Dynamic programming
  - Work from top left
  - Keep result of all sub-problems in a matrix
  - Consider each sub problem solution
Diagonal Line Problem

- Smallest problem
  - Just the top left corner
  - It to itself trivially has a path of length 1
Diagonal Line Problem

- Now work right and down
  - Next square right: no sub-problems either: 0
  - Next square down: no sub-problems either: 0
  - Next square right + down: one sub-problem: top left square
Diagonal Line Problem

- Keep expanding the frontier
  - New “2” square considers 3 possibilities:
    - 0 to get to the left 2\text{nd} row, then to here
    - 0 to get to top 2\text{nd} column, then here
    - 2 to get to diag adjacent square, then here
Diagonal Line Problem

- Keep expanding the frontier
Diagonal Line Problem

- Keep expanding the frontier
Diagonal Line Problem

• Keep expanding the frontier
Diagonal Line Problem

- When we finish, we take the max of the bottom row and right column
  - 7

- What is the running time on an MxN matrix?
Running time

- Examine each square in matrix (MxN of them)
  - For each, look at sub-problem solutions (at most M+N)

- So \((M \times N) \times (M + N)\)
  - Roughly \(N^3\) if \(M \approx N\)

- What would happen if we tried all possible paths without remembering sub-problem solutions?
Minimax Search

- Want to play an adversarial turn based game
  - Example: connect 4
    - 4 in a row, column, or diagonal of one color wins
  - We are playing black, our move
Minimax Search

- Minimax:
  - Make the best move (for you) on your turn
  - Assuming adversary will make best move (for him/her) on his/her turn
    - = worst move for you

If we don’t play there, we lose
Minimax Search

• Step 0:
  • Form an evaluation function (difficult, affects how well you play)
    • Big numbers = good for us
      • Winning position = big positive
      • Losing position = very negative

Value: -75
Minimax Search

- Step 0:
  - Form an evaluation function (difficult, affects how well you play)
    - Big numbers = good for us
      - Winning position = big positive
      - Losing position = very negative

Value: +90
Minimax Search

Step 0:
- Form an evaluation function (difficult, affects how well you play)
  - Big numbers = good for us
    - Winning position = big positive
    - Losing position = very negative
Minimax Search

- Don’t care quite so much about value of the board now
- As much as value several moves in the future
  - Our moves: best = max
  - Their moves: worst = min
Minimax

- Moves (logically) form a tree
  - Evaluate board state at the bottom: where will I end up in X moves
- Blue arrows = my move
- Red arrows = opponent’s move
- Reality: would search deeper, but can’t draw that
Minimax

- Opponent’s move:
  - Pick min
Minimax

- Our move:
  - Pick max
- More move’s ahead you can examine, better you will do
  - Why not look 100 moves ahead?
Minimax Difficulties

- Our simple game has 6 possible moves per turn
  - One move: 6 choices
  - Two moves: 36 choices
  - Three moves: 216 choices
  - N moves: ??
Minimax Difficulties

- Our simple game has 6 possible moves per turn
  - One move: 6 choices
  - Two moves: 36 choices
  - Three moves: 216 choices
  - N moves: $6^N$ choices
Minimax Difficulties

- $N$ moves: $6^N$ choices
  - How many moves can we examine if
    - Want to spend max 3 seconds
    - Can consider 1M moves per second?
Minimax Difficulties

- \( N \) moves: \( 6^N \) choices
  - How many moves can we examine if
    - Want to spend max 3 seconds
    - Can consider 1M moves per second?
      \[ \log_6(3M) = 8 \text{ moves} \]
      4 of mine
      4 of my opponent’s
8 moves: not bad?

• 4 moves for each side: probably good enough to beat most people

• But our game was pretty simple...
  • What if, instead of 6 possible moves, we had 20?

• $\log_{20}(3M) = 4.97$ moves (let’s call it 5)
  • 3 of mine
  • 2 of my opponent’s

• Decent, but not great
  • (Adding in the next move of the opponent takes 20x as long)
Two improvements

• Minimax can reduce the search space in two way
  • Memoization: may end up in same board position by different sequence of moves
  • Alpha-beta pruning
    • Alpha: best score we can get—start at +infinity
    • Beta: worst (lowest) score they can get—start at –infinity
    • As we go, narrow down the range based on what we see
      • Alpha < Beta?
        • No point in exploring: better options elsewhere
  
• Also, works well with **iterative deepening**
  • Limit depth of search to N (keep all results)
  • Time remaining? Expand depth to N+1
  • Time remaining? Expand depth to N+2
  • Out of time: give answer
Problem: Auto-complete?

• Suppose I want to implement an “auto-complete” functionality
  • Start typing a word, have it complete based on valid words
    • E.g., comp -> computer, complete, complacent, complementary,...
  • What data structure would I use to hold my valid words?
One last DS: Trie

• A trie:
  • Take first piece of input, index into table
  • Get second table, index into that with second piece of input
  • Third table with third piece..
  • Etc...

• Those of you who took 590.03 with me have seen this, and just didn’t know it— sound familiar?
One last DS: Trie

- A trie:
  - Take first piece of input, index into table
  - Get second table, index into that with second piece of input
  - Third table with third piece..
  - Etc...

- Those of you who took 590.03 with me have seen this, and just didn’t know it— sound familiar?

Page tables
One last DS: Trie

- A trie:
  - Take first piece of input, index into table
  - Get second table, index into that with second piece of input
  - Third table with third piece..
  - Etc...

- For our autocomplete problem, we can have one level of trie per letter
- Go through as many letters as we have had actual input, then either
  - Compute the set of possible words from what exists down the trie
  - Have a set of most likely completions at that point in the trie
Next week: Advanced topics in OO

• That wraps up data structures and algorithms

• Next week: Advanced topics in OO
  • Object layout (how does dynamic dispatch work?)
  • Multiple inheritance (and its variations)
  • Mixins
  • (None of this is in your book—no text reading, may find a paper)

• After that: concurrency

• Then: final exam and done