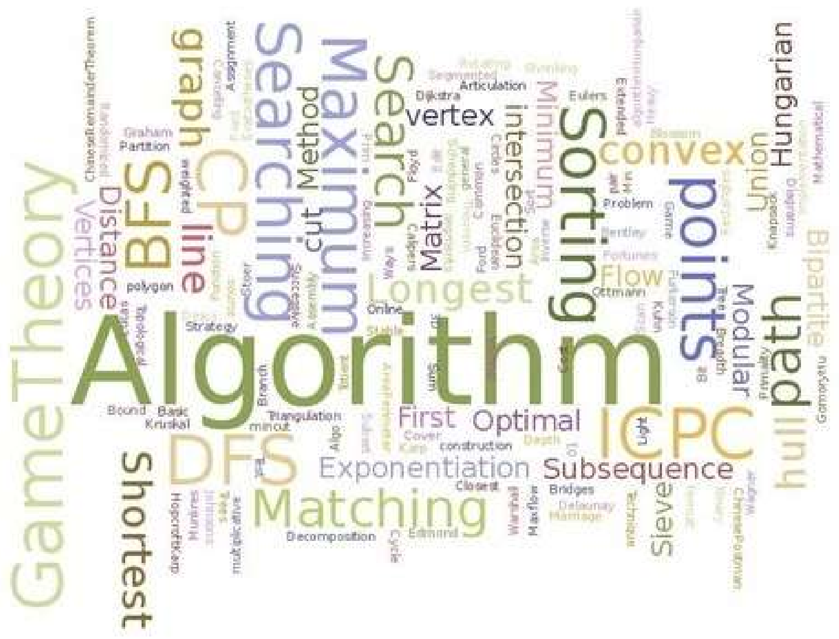
CSE 2201

COMPUTER ALGORITHMS



rmShoeb

RUET_CSE_16

Asymptotic Notations

Asymptotic analysis

- https://www.geeksforgeeks.org/analysis-of-algorithms-set-1-asymptotic-analysis/

Asymptotic notations

- https://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/

Complexity Analysis of Algorithms

- https://www.geeksforgeeks.org/analysis-of-algorithms-set-2-asymptotic-analysis/
- Worst case
- Best case
- Average case

Solving recurrences

- https://www.geeksforgeeks.org/analysis-algorithm-set-4-master-method-solving-recurrences/
- Substitution method
- Recurrence tree method
- Master method:

  $Let, T(n) = aT\left(\frac{n}{b}\right) + f(n)$

  $a = relative\ subproblem\ size\ \geq 1$

  $b = subproblems > 1$

  $c = power\ of\ f(n)$

  - $If\ c < \log_b a, then\ T(n) = O(n^{\log_b a})$
  - $If\ c = \log_b a, then\ T(n) = O(n^c logn)$
  - $If\ c > \log_b a, then\ T(n) = O\big(f(n)\big)$

Sorting Algorithms

Divide and Conquer Approach

- https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/
- Merge sort: https://www.geeksforgeeks.org/merge-sort/
- Quick sort: https://www.geeksforgeeks.org/quick-sort/
- https://www.geeksforgeeks.org/quick-sort-vs-merge-sort/

Heap

- https://medium.com/basecs/learning-to-love-heaps-cef2b273a238

Heap Construction Algorithm

- https://medium.com/@randerson112358/lets-build-a-min-heap-4d863cac6521
- Time complexity: https://www.geeksforgeeks.org/time-complexity-of-building-a-heap/
- Visualization: https://www.cs.usfca.edu/~galles/visualization/Heap.html

Heap sort

- https://www.programiz.com/dsa/heap-sort
- https://medium.com/basecs/heapify-all-the-things-with-heap-sort-55ee1c93af82
- https://www.hackerearth.com/practice/algorithms/sorting/heap-sort/tutorial/
- Visualization: https://www.cs.usfca.edu/~galles/visualization/HeapSort.html

Application of Heap

- Priority queue
- Decision tree model
    - https://www.geeksforgeeks.org/decision-tree/
    - https://economictimes.indiatimes.com/definition/decision-tree-model
- Worst case lower bound on sorting
    - https://www.geeksforgeeks.org/lower-bound-on-comparison-based-sorting-algorithms/
    - http://www.quretec.com/u/vilo/edu/2003-04/ATABI_2004k/BigOh/sort_lower.html

Sorting in Linear time

- **Counting sort**
    - https://www.geeksforgeeks.org/counting-sort/
    - https://medium.com/basecs/counting-linearly-with-counting-sort-cd8516ae09b3
    - Visualization: https://www.cs.usfca.edu/~galles/visualization/CountingSort.html
- **Radix sort**
    - https://www.geeksforgeeks.org/radix-sort/
    - https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_radix_sort.htm
    - https://medium.com/basecs/getting-to-the-root-of-sorting-with-radix-sort-f8e9240d4224
    - http://btechsmartclass.com/data_structures/radix-sort.html
    - Visualization: https://www.cs.usfca.edu/~galles/visualization/RadixSort.html
- **Bucket sort**
    - https://medium.com/karuna-sehgal/an-introduction-to-bucket-sort-62aa5325d124
    - https://www.oreilly.com/library/view/algorithms-in-a/9780596516246/ch04s08.html
    - Visualization: https://www.cs.usfca.edu/~galles/visualization/BucketSort.html

Comparison

- https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/
- https://www.geeksforgeeks.org/stability-in-sorting-algorithms/

Graph Algorithms

Representation of Graph

- Adjacency matrix
- Adjacency list
- https://medium.com/basecs/from-theory-to-practice-representing-graphs-cfd782c5be38
- https://www.hackerearth.com/practice/algorithms/graphs/graph-representation/tutorial/

Breadth First Search:

- https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/
- Visualization: https://www.cs.usfca.edu/~galles/visualization/BFS.html
- Application: https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/

Depth First Search:

- https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/
- Visualization: https://www.cs.usfca.edu/~galles/visualization/DFS.html
- Application: https://www.geeksforgeeks.org/applications-of-depth-first-search/

Disjoint set

- https://www.hackerearth.com/practice/data-structures/disjoint-data-strutures/basics-of-disjoint-data-structures/tutorial/
- https://www.topcoder.com/community/competitive-programming/tutorials/disjoint-set-data-structures/
- Visualization: https://www.cs.usfca.edu/~galles/visualization/DisjointSets.html

Minimum Spanning Tree

- It is the minimum cost to traverse all the nodes
- https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/
- **Prim's algorithm**
    - http://scanftree.com/Data_Structure/prim's-algorithm
    - Visualization: https://www.cs.usfca.edu/~galles/visualization/Prim.html
    - Proof of correctness: http://math.wikia.com/wiki/Proof_of_Prim%27s_algorithm
- **Kruskal's algorithm**
    - http://scanftree.com/Data_Structure/kruskal's-algorithm
    - Visualization: https://www.cs.usfca.edu/~galles/visualization/Kruskal.html
    - Proof of correctness: https://ebrary.net/25909/computer_science/proof_correctness

- **Applications of MST:** https://www.geeksforgeeks.org/applications-of-minimum-spanning-tree/

Applications of Graph: https://www.geeksforgeeks.org/applications-of-graph-data-structure/

Shortest Path

Shortest path problem:

- The shortest path problem is about finding a path between 2 vertices in a graph such that the total sum of the edges weights is minimum. This problem could be solved easily using BFS if all edge weights were 1, but here weights can take any value.
- https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/
- https://algs4.cs.princeton.edu/44sp/

Dijkstra's Algorithm

- Dijkstra's Algorithm allows you to calculate the shortest path between one node (you pick which one) and every other node in the graph. This is a greedy algorithm.
- https://www.codingame.com/playgrounds/1608/shortest-paths-with-dijkstras-algorithm/dijkstras-algorithm
- $Complexity: O(V^2);$
  $using\ min-priority\ queue: O(V + E\log V);$
  $using\ Fibonacci\ heap: O(V\log V)$
- Visualization: https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html
- Proof of correctness: https://slideplayer.com/slide/10592953/

Bellmen-Ford Algorithm

- Bellman Ford algorithm helps us find the shortest path from a vertex to all other vertices of a weighted graph. It is similar to Dijkstra's algorithm but it can work with graphs in which edges can have negative weights. It is a dynamic programing algorithm.
- https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/
- https://www.programiz.com/dsa/bellman-ford-algorithm
- $Complexity: O(V.E)$

Floyd Warshall Algorithm

- Floyd-Warshall Algorithm is best suited for dense graphs since its complexity depends only on the number of vertices in the graph. It is a dynamic programing algorithm.
- Visualization: https://www.cs.usfca.edu/~galles/visualization/Floyd.html
- Difference between Dijkstra's algorithm and Floyd-Warshall algorithm: https://www.geeksforgeeks.org/comparison-dijkstras-floyd-warshall-algorithms/
- $Complexity: O(V^3)$

Searching Algorithms

Traditional search:

Heuristic search: Heuristic search is an AI search technique that employs heuristic for its moves. Heuristic is a rule of thumb that probably leads to a solution. Heuristics play a major role in search strategies because of exponential nature of the most problems. In computer science, artificial intelligence and mathematical optimization, a heuristic is a technique designed for solving a problem more quickly when classic methods are too slow or for finding an approximate solution when classic methods fail to find any exact solution.

Binary Search Tree

- https://www.hackerearth.com/practice/data-structures/trees/binary-search-tree/tutorial/
- https://www.programiz.com/dsa/breadth-first-search-tree
- https://www.programiz.com/dsa/tree-traversal
- https://medium.freecodecamp.org/data-structures-101-binary-search-tree-398267b6bff0
- Visualization: https://www.cs.usfca.edu/~galles/visualization/BST.html
- Number of possible BST- (Catalan Number): https://www.geeksforgeeks.org/total-number-of-possible-binary-search-trees-with-n-keys/

Balanced Binary Search Tree

- http://www.cs.ecu.edu/karl/3300/spr16/Notes/DataStructure/Tree/balance.html
- **AVL tree**
    - https://medium.com/basecs/the-little-avl-tree-that-could-86a3cae410c7
    - http://btechsmartclass.com/data_structures/avl-trees.html
    - Visualization: https://www.cs.usfca.edu/~galles/visualization/AVLtree.html
- **Red-Black tree**
    - https://medium.com/basecs/painting-nodes-black-with-red-black-trees-60eacb2be9a5
    - https://towardsdatascience.com/red-black-binary-tree-maintaining-balance-e342f5aa6f5
    - Visualization: https://www.cs.usfca.edu/~galles/visualization/RedBlack.html
- **Interval trees**

B-trees

Skip Lists

Hashing

Priority Queues

Dynamic Programming

Properties of Dynamic programming

- https://www.geeksforgeeks.org/overlapping-subproblems-property-in-dynamic-programming-dp-1/
- https://www.geeksforgeeks.org/optimal-substructure-property-in-dynamic-programming-dp-2/

Longest Common Subsequence (LCS)

- https://www.dyclassroom.com/dynamic-programming/longest-common-subsequence
- https://www.geeksforgeeks.org/printing-longest-common-subsequence/
- Visualization: https://www.cs.usfca.edu/~galles/visualization/DPLCS.html
- Application: DNA matching

Matrix Chain Multiplication/Matrix Chain Ordering Problem

- https://www.geeksforgeeks.org/matrix-chain-multiplication-dp-8/
- https://www.techiedelight.com/matrix-chain-multiplication/

Knapsack Problem

- https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/
- https://www.hackerearth.com/practice/notes/the-knapsack-problem/
- Space optimized: https://www.geeksforgeeks.org/space-optimized-dp-solution-0-1-knapsack-problem/

Multistage Graphs

Dynamic programing vs. Divide and conquer:

- https://www.geeksforgeeks.org/dynamic-programming-vs-divide-and-conquer/

Greedy Algorithm

Greedy algorithm: Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy. For example, consider the Fractional Knapsack Problem. The local optimal strategy is to choose the item that has maximum value vs. weight ratio. This strategy also leads to global optimal solution because we allowed to take fractions of an item.

Activity Selection Problem

- https://www.dyclassroom.com/greedy-algorithm/activity-selection-problem
- https://algoskills.com/activityselection.php

Huffman coding

- https://www2.cs.duke.edu/csed/poop/huff/info/
- https://www.techiedelight.com/huffman-coding/

- **Time complexity:** O(nlogn) where n is the number of unique characters. If the inputs are sorted, then complexity becomes O(n).

Knapsack problem

-

Tree vertex splitting

-

Job sequencing problem

- https://www.includehelp.com/operating-systems/job-sequencing.aspx
- https://www.techiedelight.com/job-sequencing-problem-deadlines/
- Time complexity is $O(n^2)$ but using disjoint set can optimize it further.

Correctness in greedy technique

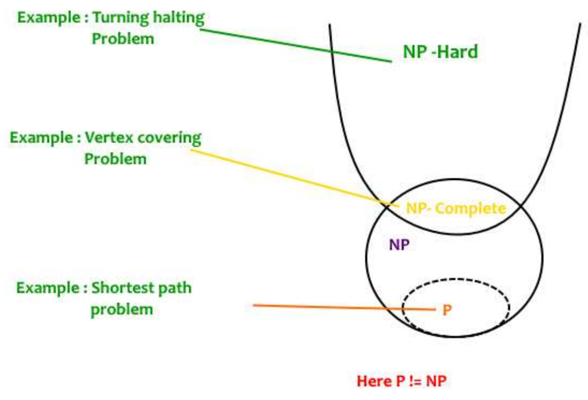
- https://www.geeksforgeeks.org/correctness-greedy-algorithms/

Recurrences & Backtracking

Recurrences

- A recurrence is a recursive description of a function or in other words, a description of a function in terms of itself. Like all recursive structures, a recurrence consists of one or more base cases and one or more recursive cases.

NP-hard and NP-complete problems


Example : Turning halting Problem
NP -Hard
Example : Vertex covering Problem
NP- Complete
NP
Example : Shortest path problem
P
Here P != NP

1. **NP-Hard:** A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder.
2. **NP-Complete**
    - https://www.geeksforgeeks.org/np-completeness-set-1/
    - https://www.britannica.com/science/NP-complete-problem
    - https://www.ics.uci.edu/~eppstein/161/960312.html
    - https://www.mathsisfun.com/sets/np-complete.html
    - https://xlinux.nist.gov/dads/HTML/npcomplete.html

3. **NP:** Non-deterministic polynomial time. Actual complexity is exponential but can be solved in polynomial complexity (in non-deterministic way). Exponential complexity is the most risky.
4. **P:** Algorithms that are solvable in deterministic polynomial time.

Intractable problem: From a computational complexity stance, intractable problems are problems for which there exist no efficient algorithms to solve them. Most intractable problems have an algorithm – the same algorithm – that provides a solution, and that algorithm is the brute-force search.

Tractable problem: So-called easy, or tractable, problems can be solved by computer algorithms that run in polynomial time; i.e., for a problem of size n, the time or number of steps needed to find the solution is a polynomial function of n.

Optimization problem: In mathematics and computer science, an optimization problem is the problem of finding the best solution from all feasible solutions. Optimization problems can be divided into two categories depending on whether the variables are continuous or discrete.

Decision problem: In computability theory and computational complexity theory, a decision problem is a problem that can be posed as a yes-no question of the input values. An example of a decision problem is deciding whether a given natural number is prime. Another is the problem "given two numbers x and y, does x evenly divide y?"

Reduction

π_2 can be reduced to π_1 if π_1 has a polynomial time solution and that solution can lead to polynomial time complexity of π_2.

Backtracking

- https://www.geeksforgeeks.org/backtracking-introduction/
- Backtracking is a general algorithm for finding all solutions to some computational problems, notably constraint satisfaction problems that incrementally builds candidates to the solutions and abandons a candidate as soon as it determines that the candidate cannot possibly be completed to a valid solution.
- **E-node**
  - The nodes that are being explored.
- **Live node**
  - The nodes that are to be explored.
- **Dead node**
  - The nodes that has already been explored.
- **Branch and bounds**
  - A branch and bound algorithm is an optimization technique to get an optimal solution to the problem. It looks for the best solution for a given problem in the entire space of the solution. The bounds in the function to be optimized are merged with the value of the latest best solution. It allows the algorithm to find parts of the solution space completely. The purpose of a branch and bound search is to maintain the lowest-cost path to a target. Once a solution is found, it can keep improving the solution.

- **Application**
  - Puzzle: n-Queens problem, crosswords, verbal arithmetic, Sudoku, peg solitaire etc.
  - Game development.
- **n-Queens problem**
  - https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/
  - In O(n) space: https://www.geeksforgeeks.org/n-queen-in-on-space/
  - Printing all solutions: https://www.geeksforgeeks.org/printing-solutions-n-queen-problem/
  - Using branch and bound: https://www.geeksforgeeks.org/n-queen-problem-using-branch-and-bound/
- **Knapsack problem**
  - http://condor.depaul.edu/ichu/csc491/notes/wk8/knapsack.html
  - Using branch and bound: https://www.geeksforgeeks.org/0-1-knapsack-using-branch-and-bound/
  - Implement branch and bound: https://www.geeksforgeeks.org/implementation-of-0-1-knapsack-using-branch-and-bound/
- **Graph coloring**
  - https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/
- **Generating all permutation of a string**
  - Print: https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/
  - With duplications: https://www.geeksforgeeks.org/print-all-permutations-of-a-string-with-duplicates-allowed-in-input-string/
  - Using STL: https://www.geeksforgeeks.org/permutations-of-a-given-string-using-stl/
  - Time complexity: https://www.geeksforgeeks.org/time-complexity-permutations-string/
- **Find all subsets**
  - https://www.geeksforgeeks.org/backtracking-to-find-all-subsets/
- **8-puzzle**
  - B
  - Using branch and bound: https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/
- **Traveling Salesman Problem**
  - B
  - Using branch and bound: https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/
- **Job Assignment Problem**
  - B
  - Using branch and bound: https://www.geeksforgeeks.org/job-assignment-problem-using-branch-and-bound/
- **Recursion and Backtracking:** https://www.hackerearth.com/practice/basic-programming/recursion/recursion-and-backtracking/tutorial/

Reducibility between Problems & NP-Completeness

Lower bound theory

Discussion of different NP-complete problems

- **Satisfiability**
  - In mathematical logic, satisfiability and validity are elementary concepts of semantics. A formula is satisfiable if it is possible to find an interpretation that makes the formula true. A formula is valid if all interpretations make the formula true.
  - Boolean satisfiability
- **Clique**
  - Clique finding
  - Minimum clique finding
- **Independent set**
  - Minimum independent set
- Relation between independent set and clique finding
- **Vertex cover**
  - Minimum vertex cover
- Relation between Independent set and Vertex cover
  - If one is solved then can the other be solved too?
- **Hamilton cycle**
- **Knapsack**
  - Can knapsack problem really be solved in polynomial time?
- **Set cover**
- **Bin packing**

Computational geometry

Line segment Properties

Convex hull

- Given a set of points in the plane, the convex hull of the set is the smallest convex polygon that contains all the points of it.
- Divide and conquer
  - https://www.geeksforgeeks.org/convex-hull-using-divide-and-conquer-algorithm/
  - https://iq.opengenus.org/divide-and-conquer-convex-hull/
- Graham scan
  - https://www.cs.auckland.ac.nz/software/AlgAnim/convex_hull.html

Amortized analysis

- https://www.geeksforgeeks.org/analysis-algorithm-set-5-amortized-analysis-introduction/
- https://www.geeksforgeeks.org/amortized-analysis-increment-counter/

- http://www.cs.cornell.edu/courses/cs3110/2011sp/Lectures/lec20-amortized/amortized.htm

Pseudo-polynomial complexity

- https://www.geeksforgeeks.org/pseudo-polynomial-in-algorithms/
- https://en.wikipedia.org/wiki/Pseudo-polynomial_time

Fibonacci heap

- B
- How can Prim's algorithm be improved by Fibonacci heap?

Guess and verifying

Approximation algorithm

Others

Integer multiplication

- http://www.cs.cmu.edu/~cburch/pgss99/lecture/0721-divide.html

Median

- https://www.geeksforgeeks.org/median/
- Median finding:

Strassen's matrix multiplication

- https://www.geeksforgeeks.org/strassens-matrix-multiplication/
- https://en.wikipedia.org/wiki/Strassen_algorithm

Hope is the only way you will survive this. Good luck ☺