CS3230: D&A of Algo

Reasoning Algorithms

- **Loop Invariant: true** at the beginning of iteration and **remains true** at the beginning of next iteration
- **Correctness using invariant:**
  - **Initialization:** Invariant true before first iteration of loop
  - **Maintenance:** If invariant is **true** before iteration, it **remains true before next** iteration
  - **Termination:** When algo terminates, invariant provides a useful property for showing correctness
- **Recursive Algorithms:** use **induction** on size of problem

Asymptotic Notation

- **O-notation (upper):** $f(n) = O(g(n))$ if $\exists c > 0, n_0 > 0$ s.t. $0 \le f(n) \le cg(n), \forall n \ge n_0$
- **Ω-notation (lower):** $f(n) = \Omega(g(n))$ if $\exists c > 0, n_0 > 0$ s.t. $0 \le cg(n) \le f(n), \forall n \ge n_0$
- **Θ-notation (tight):** $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
- **o and ω-notation:** no equal sign

Solving recurrences

- **Substitution Method**
  [Ex: $T(n) = 4T(n/2) + n$]
  - **Guess** the solution $[O(n^3)]$
  - **Verify** by induction [Prove $T(n) \le cn^3$, assuming $T(k) \le ck^3$ for $k < n$] (Modify to desired - residual form)
  - **Solve** for c and n_0
- **Recursion-tree method**
  [Ex: $T(n) = 2T(n/2) + n^2$]
  Start with n, child will have $(n/2)^2$. Sum up rows in tree and sum the sum.
- **Master Method**
  [$T(n) = aT(n/b) + f(n)$, $a \ge 1, b > 1$, f asymptotically +ve]
  (Cmp $f(n)$ & $n^{\log_b a}$)
  - **Case 1** $f(n) = O(n^{\log_b a - \epsilon})$ for const $\epsilon > 0 \to T(n) = \Theta(n^{\log_b a})$
  - **Case 2** $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for const $k \ge 0$
    $\to T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
  - **Case 3** $f(n) = \Omega(n^{\log_b a + \epsilon})$ for const $\epsilon > 0$ & $f(n)$ satisfies **regularity cond** that $af(n/b) \le cf(n)$ for const $c < 1$
    $\to T(n) = \Theta(f(n))$

Properties of Functions

- $e^x \ge 1 + x$
- Exp $f(x)$ with base $a > 1$ grows faster than any poly, $n^k = O(a^n)$
- $\lg n = \log_2 n$ — $\ln n = \log_e n$
- $a^{\log_b c} = c^{\log_b a}$
- $\lg n = \Theta(\ln n) = \Theta(\log_{10} n)$
- **Stirling's approximation**
  $$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$
  $\log(n!) = \Theta(n \lg n)$
- $\sum_{k=1}^{n} k = \frac{n}{2}(n+1) = \Theta(n^2)$
- $\sum_{k=1}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}$
  $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}, |x| < 1$
- **Harmonic:** $H_n = \sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$
- **Telescoping** $\sum_{k=0}^{n-1}(a_k - a_{k+1}) = a_0 - a_n$
- **Limits**
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \implies f(n) \in O(g(n))$
  - $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} \implies f(n) \in \Omega(g(n))$
  - $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \implies f(n) \in \Theta(g(n))$
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \implies f(n) \in o(g(n))$
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \implies f(n) \in \omega(g(n))$
- **L'Hôpital:** $\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)}$

Probabilistic Analysis

- **Conditional Probability**
  $$Pr\{A|B\} = \frac{Pr\{A \cap B\}}{Pr\{B\}}$$
- **Independent Events** if $Pr\{A \cap B\} = Pr\{A\}Pr\{B\}$
- **Bayes's Theorem**
  $$Pr\{A|B\} = \frac{Pr\{A\}Pr\{B|A\}}{Pr\{B\}}$$
- **Expectation**
  - $E[X] = \sum(x \cdot Pr\{X = x\})$
  - **Linearity of Expectation**
    $E[X + Y] = E[X] + E[Y]$
    $E[aX] = aE[X]$
  - $E[XY] = E[X]E[Y]$ (if **independent**)
- p: prob of success & $q = 1 - p$
- **Geometric Distribution**
  - Let X be no of trials to obtain success
  - Then, $Pr\{X = k\} = q^{k-1}p$ & $E[X] = 1/p$
- **Binomial Distribution**
  - Let X be the number of successes in n Bernoulli trials.
  - Then, $Pr\{X = k\} = \binom{n}{k}p^k q^{n-k}$ & $E[X] = np$
- **Indicator Random Variable**
  $$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{if } A \text{ does not occur} \end{cases}$$
  $X_A = I\{A\} \implies E[X_A] = Pr\{A\}$

Probabilistic Algorithms

- **Las Vegas Algo** – **always** correct answer, but random runtime
- **Monte Carlo Algo** – constant runtime but answer **not guaranteed** to be correct
- Usually determine **average case runtime**

Lower Bound of Sorting

- Any decision tree that can sort n elements must have height $\Omega(n \lg n)$
- **Corollary** Heapsort and merge sort are asymptotically optimal comparison sorting algorithms.

Radix Sort

- $\Theta(nw)$ to sort n items with w size.
- Sort on **least-significant digit first** with aux **stable** sort (usually counting sort)
- For b-bit word broken into r-bit pieces, each counting sort pass takes $\Theta(n + 2^r)$
- Thus, $T(n, b) = \Theta\left(\frac{b}{r}(n + 2^r)\right)$

Order-Statistic Selection

- All are comparison-based algorithms
- **Randomized (QuickSelect)** $\Theta(n)$ (expected), but $\Theta(n^2)$ (worse case). Practical.
- **Worst-case linear-time order statistics:** $\Theta(n)$ worse case, but c may be too big.
  Recurrence:
  $$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{3}{4}n\right) + \Theta(n)$$
  Running Time Analysis:
  $$T(n) \le cn - \left(\frac{1}{20}cn - \Theta(n)\right)$$

Amortized Analysis

- **Aggregate** method: find cost for n operation and calculate average cost of each operation
- **Accounting** method
  - Charge **amortized cost**, $\hat{c}_i$, where \$1 pays for 1 unit of work
  - Ensure that: $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i$ (bank balance never go negative)
  - Thus, amortized cost is the **upper bound** of actual cost
- **Potential** method
  - D_i is structure after i-th oper
  - Define **potential method** s.t. $\Phi(D_0) = 0$ & $\Phi(D_i) \geq 0, \forall i$
  - Thus, $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Dynamic Programming

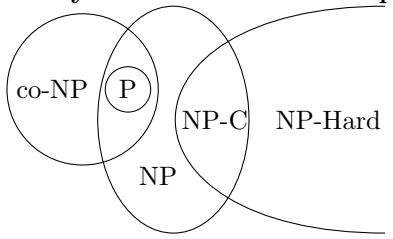
- **Memoization** – top-down approach, start from desired position and work down, caching previously computed results
- **Dyanmic Programming** – can be implemented through memoization or tabulation (i.e. calculate all values, without regard). However, idea is to start from small sizes/cases and work your way up.
- **Optimal substructure** – An optimal solution to a problem (instance) contains optimal solutions to subproblems.
- **Overlapping subproblems** A recursive solution contains a "small" number of distinct subproblems repeated many times

Greedy Algorithms

- Prove using **optimal substructures** and **greedy-choice**
- **Greedy-choice property** – A locally optimal choice is globally optimal. To prove (cut and paste),
  - Find the optimal choice
  - Replace the optimal choice by the greedy choice
  - Show that the greedy choice is as good as the optimal choice

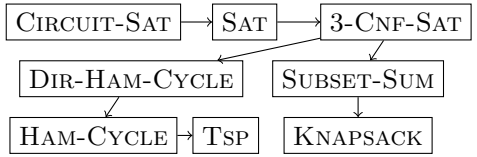
Computational Complexity

- A **Polynomial Time** algorithm has a worse case time complexity of $O(n^c)$ for some constant c
- A **Superpolynomial Time** algorithm has worse case complexity of $\omega(n^c)$ for **all** constants c.
- A **pseudo-polynomial time** algorithm runs in poly time in the numeric value of the input but is exponential in the len of the input
- **Decision problem**: answer either YES or NO (e.g. Subset-Sum)
- **Optimization problem**: output optimal value (and solution)
- **Classes of Problems**
  - **P** – Problems solvable in polynomial time
  - **NP** – Problems with polynomial time verifiable solutions
  - **co-NP** – Problems with poly time verifiable counterexamples
- **Likely Class Relationships**

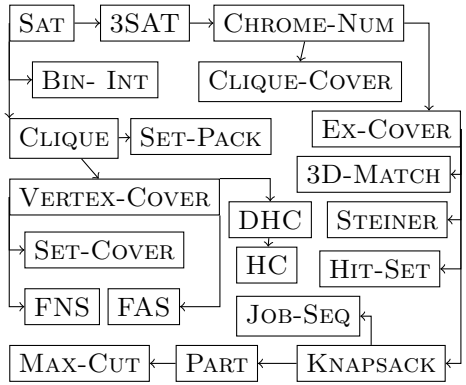


- **Polynomial Time Reduction** ($A \leq_P B$, A to B) is transformation with the following properties:
  - Trans takes **polynomial time**
  - Answer for α is YES $\iff$ answer for β is YES
- **NP-Hard**: Problem $L \in$ NP-Hard if any other problem $L' \in$ NP can be reduced to L in poly time (i.e. $L' \leq_P L, \forall L' \in$ NP)
  - If L solvable in poly time, then **ALL** NP problems can be solved in poly time
- **NP-Complete**: Problem L is both NP-Hard and NP

NP-Complete Problems



Karp's 21 NP-C Problems



Approximation Algorithms

- C^* denotes optimal cost and C is the cost of approx algo, A
- **Approximation Ratio**
  A has approx ratio of $\rho(n)$ if
  – $C/C^* \leq \rho(n)$ (minimization) or
  – $C/C^* \geq \rho(n)$ (maximization)
  A is a $\rho(n)$ approx algo (or ρ-approximation if independent of n)
- **Proving non-approximability**
  Reduce NP-complete (decision) problem X to (decision version of) Approx-Y
  – "yes" instances of X correspond to instances of Y with value $\leq k$
  – "no" instances of X corr. to instances of Y with value $> \rho k$
  Unless $P = NP$, there's no poly time ρ-approximate algo for Y
- **Polynomial-time approximation scheme (PTAS)** is approx algo whose approx ratio is $1 + \epsilon$ ($1 - \epsilon$ for max) for any $\epsilon > 0$
  – As ϵ decreases, runtime for PTAS increases, potentially exponential in $1/\epsilon$ (e.g. $O(n^{1/\epsilon})$)
  – If runtime of PTAS is poly in both $1/\epsilon$ and n (e.g. $O((1/\epsilon)^2 n^3)$), it is a **fully polynomial-time approximation scheme (FPTAS)**
- **Common techniques** to find approx algos – greedy, relaxation, DP

CS3230: NP-Complete Problems

Circuit-Sat (Circuit Satisfiability)

- **Input**
  A Boolean circuit composed of logic gates **AND**, **OR** and **NOT** with single output
- **Output**
  Does there exist an assignment of input variables such that the circuit is **satisfiable** (i.e. output 1)?

Sat (Satisfiability)

- **Input**
  - n boolean variables: $x_1, x_2, ..., x_n$
  - Boolean formula ϕ formed by Boolean variables and operators ($\wedge, \vee, \neg, \rightarrow, \leftrightarrow$ (equality))
- **Output**
  Yes if ϕ is satisfiable or No

3-Cnf-Sat (3-satisfiability)

- **CNF (Conjunctive Normal Form)** is a Boolean formula expressed as
  - AND of clauses, where
  - Each clause is the OR of one or more literals
- **Input:** 3-CNF formula ϕ
- **Output:** If 3-CNF formula ϕ is satisfiable?

Dir-Ham-Cycle (Dir Hamiltonian Cycle)

- **Input:** Given a digraph $G = (V, E)$
- **Output:** Doesz there exist a simple directed cycle, Γ, that contains every node in V **exactly once**?

Ham-Cycle (Hamiltonian Cycle)

- **Input:** An undirected graph $G = (V, E)$
- **Output:** Decide if G contains Hamiltonian cycle

TSP (Travelling Salesman Problem)

- No poly time constant-approx algo for general form (unless $P = NP$)!
- **Input:**
  - Digraph $G = (V, E)$
  - Cost f(x): $c : E \rightarrow \mathbb{Z}$
  - Cost threshold k
- **Output:** Decide if there exists a **tour** that visits all vertices in V with cost at most k.
- **2-approx** Metric-TSP – Construct min spanning tree ($\Theta(V^2)$), Compute L, list of vertices visited in preorder tree walk of MST ($\Theta(V)$) & return Ham cycle, skipping repeated vertices

Subset-Sum

- **Input:** A set of integers S and an integer t
- **Output:** Decide if there exists a **non-empty** subset $S' \subseteq S$ s.t. $\sum_{x \in S'} x = t$

Knapsack (Knapsack Problem)

- **Input:**
  - (w_n, v_n)-s where w_i & v_i are non-negative $\mathbb{Z}$
  - Capacity W
  - Value threshold V
- **Output:** Is there a subset of items with total weight at most W and value at least V?
- **0.5-approximation:** Select greedily based on v_x/w_x and compare with item with largest value (& weight less than W)

Vertex-Cover (Vertex Cover)

- **Vertex Cover** is the set of vertices where all edges in G are incident to a vertex in the set
- **Input:** Graph G & $k \in \mathbb{Z}^+$
- **Output:** Is there a vertex cover of size at most k in G?

Set-Cover

- **Input:** (X, F), where X is a finite set and F is a family of subsets of X s.t. every elem of X belongs to at least one subset in F (i.e. $X = \bigcup_{S \in F} S$) and $k \in \mathbb{Z}$
- **Output:** Is there a set cover of size $\leq k$?
- $\log |X|$**-approximation** – greedy select set, S s.t. maximize $|S \cap U|$ (U remaining elements)

Chrome-Num (Chromatic Number)

- Also known as **graph colouring problem**
- **Input:** Graph G
- **Output:** Find the smallest number of colours needed to colour the vertices s.t. no two adjacent vertices share same colour

Ex-Cover (Exact Cover)

- **Input:** Collection of subsets S
- **Output:** Is there a $S^* \subseteq S$ s.t. $\forall x \in S_i \in S^*$, x is only in S_i

Clique

- Also known as **complete subgraphs**
- **Input:** Graph G and clique size k (# of vertices)
- **Output:** Is there a clique of size k in G?

Clique-Cover

- **Input:** Graph G and clique size k
- **Output:** Is there a clique of size $< k$ in G?

Steiner

- Tree that allows use of other vertices, if it costs less
- **Input:** $G = (V, E)$ be an undirected graph with non-negative edge weights c, $S \subseteq V$ and $k \in \mathbb{N}$
- **Output:** A Steiner tree exists whose total weight $< k$ and passes all vertex in S

Part (Partitioning)

- **Input:** A set of $\mathbb{Z}^+$ $S = \{s_1, \ldots, s_n\}$
- **Output:** Is there a partition of S into two disjoint subsets S_1 & S_2 s.t. $S = S_1 \cap S_2$ & $\sum S_1 = \sum S_2$

Max-Cut (Maximum Cut)

- A k size cut is a cut that has k edges (or edges with total weight k)
- **Karp's** reduction uses weighted edges
- **Input:** Graph G and size k
- **Output:** Is there a cut of size at least k in G?

CS3230: Useful Stuff

Useful Facts

- **Triangle Inequality** – $z \leq a + b$
- **Unstable** – quick, selection, heap & cycle sorts

Heaps

- **Create Heap (Heapify)** – $O(n)$ (insert to last, bubble costs $O(\lg n)$)
- **Insert** – $O(\lg n)$ (insert to last place and bubble) $O(1)$ (binomial, amortized)
- **Delete-Max/Min** – $O(\lg n)$ (all heap types)
- **Find-Max/Min** – $O(1)$ (all except binomial) $O(\lg n)$ (binomial, unoptimized)
- **Array representation** – $i/2$ (parent), $2i$ (left), $2i + 1$ (right)

Union Find

- **QuickFind** – $O(1), O(n)$ (F,U) – root parent (flat)
- **QuickUnion** – $O(n)$ (F,U) – arbitrary parent (tall)
- **Weighted/Ranked** – $O(\lg n)$ (F,U) – connect smallest to largest
- **Weighted & Path Compress** – $O(\alpha(n))$ (F,U)
- $O(1)$ for both **impossible**

Graphs

- **Complete** – every pair of distinct vertices is connected by a unique edge ($O(V) = O(E^2)$, $n = n(n-1)/2$)
- **Strongly Connected** – there's path between all pairs of vertices
- **TopoSort** – sort vertices in DAG s.t. u appears before v for edge (u, v) ($O(V+E)$, post-order DFS)
  - Use to find shortest & longest path in DAG (set vertex to ∞ or $-\infty$)
- **DFS/BFS** – $O(V + E)$
- **Prim's** – start from arbitrary vertex and greedily choose cheapest vertex, $O(E \log V)$ (binheap)
- **Kruskal's** – choose cheapest global edge greedily (while avoiding cycles), $O(E \log V)$ (binheap)
- **Boruvka's** – start with single vertex components, join components with cheapest edge $O(E \log V)$ (binheap)
- **Bellman-Ford** – Single-source shortest path problem for weighted digraph ($\Theta(VE)$ (worst), $\Theta(V)$ (space))
  Relax all edges $V-1$ times. To find negative weight cycles, run again and see if estimates change
- **Dijsktra's** (Not for negative weights!)
  - **Worst Running Time:**
    $O(|E| \log |V|)$ (Binary Heap, List)
    $O(|V|^2 \log |V|)$ (Bin Heap, Matrix)
    $O(|E| + |V| \log |V|)$ (Fib Heap, List)
    $O(|V|^2)$ (Fibby Heap, Matrix)
    * Maintain distance **estimate** for every node
    * Begin with empty shortest-path-tree
    * Repeat: Consider v_i with minimum **estimate**, add v_i to shortest-path-tree, relax all outgoing edges from v_i! $O(E)$ relax
- **Floyd-Warshall** (APSP)
  - **Complexity:** $\Theta(V^3)$ (runtime) & $\Theta(V^2)$ (space for adj matrix)
  - shortest(start, end, first x vertices)
    * **Base Case:** $\text{shortest}(i, j, 0) = w(i, j)$
    * **Recursive:** $\text{shortest}(i, j, k + 1) = \min(\text{shortest}(i, j, k), \text{shortest}(i, k + 1, k) + \text{shortest}(k + 1, j, k))$
- **Closest Pair of Points** ($O(n \lg n)$)
  - Sort coords based on x coords
  - Divide to two halves and recursively find smallest dist in both subarrays
  - Take min of the two distances & create array to store points that are at most d from middle line dividing the two sets
  - Return min of d and smallest dist in array

Full Algorithms

- Worse-case linear time select
  1. Div n elements into groups of 5. Find the median of each 5-element group by rote. – $\Theta(n)$
  2. SELECT median x of $\lfloor n/5 \rfloor$ grp medians to be pivot – $T(n/5)$
  3. Part around pivot x. Let $k = \text{rank}(x)$ – $\Theta(n)$
  4. if $i = k$ then return x
  5. else if $i < k$ SELECT i-th element in lower part
  6. else SELECT $(i - k)$-th elem in upper part