CS3230 Final Cheat sheet

Week 1: Asymptotic Analysis

Week 2: Recurrences and Master's Theorem

Example of Runtime for Decreasing Functions

T(n) $= T(n-1) + 1 \hookrightarrow O(n)$
T(n) $= T(n-1) + n \hookrightarrow O(n^2)$
T(n) $= T(n-1) + log(n) \hookrightarrow O(nlogn)$
T(n) $= 2T(n-1) + 1 \hookrightarrow O(2^n)$
T(n) $= 3T(n-1) + 1 \hookrightarrow O(3^n)$
T(n) $= 2T(n-1) + n \hookrightarrow O(n2^n)$

Rules

$T(n) = aT(n/b) + f(n)$

- $a \geq 1$
- $b > 1$
- $f(n) = \theta(n^k log^p n)$

case 1: if $log_b^a > k$ then $\theta(n^{log_b^a})$
case 2: if $log_b^a = k$
if $p > -1$ then $\theta(n^k log^{p+1} n)$
if $p = 1$ then $\theta(n^k loglogn)$
if $p < -1$ then $\theta(n^k)$
case 3: if $log_b^a < k$
if $p \geq 0$ then $\theta(n^k log^p n)$
if $p < 0$ then $\theta(n^k)$
Regularity Condition: $af(n/b) \leq cf(n)$

Week 6: Randomized Algorithms and Order Statistics

Randomized Las Vegas Algorithm

- Output is always correct
- Running time is a random variable

Randomized Monte Carlo Algorithm

- Output may be incorrect with some small probability
- Running time is deterministic

Week 7: Amortized Analysis

- **Amortized Analysis**: Strategy for analyzing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive
- Actual Cost $\leq$ Amortized Cost

Accounting Method

- Charge ith operation a fictitious amortized cost c(i)
- This fee is used to perform the operation
- Any amount not immediately consumed is stored in the bank for use by subsequent operations
- The idea is to impose an extra charge on inexpensive operations and use it to pay for expensive operations later on
- The bank balance must not go negative
- We must ensure that $\Sigma_{i=1}^n t(i) \leq \Sigma_{i=1}^n c(i)$ for all n
- Thus, the total amortized cost provides an upper bound on the total true costs

Potential Method

θ: Potential function associated with the algorithm/data-structure
$\theta(i)$: Potential at the end of the ith operation

Important conditions to be fulfilled by θ

- $\theta(0) = 0$
- $\theta(i) \geq 0$ for all i
- $\Delta\theta_i = \theta(i) - \theta(i-1)$
- Amortized cost of the ith operation = Actual cost of the ith operation + $\Delta\theta_i$
- Amortized cost of n operation $\geq$ Actual cost of n operations
- Actual Cost $\leq$ Amortized Cost

If we want to show that actual cost of n operations is O(g(n)) then it suffices to show that amortized cost of n operations is O(g(n))

Week 8: Dynamic Programming

Optimal Substructure: An optimal solution to a problem (instance) contains optimal solutions to sub-problems

Longest Common Subsequence

Definition: C is said to be a subsequence of A if we can obtain C by removing zero or more elements from A

Recursive Formula for finding LCS

Base Case:
$LCS(i, 0) = \emptyset$ for all i
$LCS(0, j) = \emptyset$ for all j
General Case:
If $a_n = b_m$ then $LCS(n, m) = LCS(n-1, m-1) :: a_n$
If $a_n \neq b_m$ then
$LCS(n, m) = max(LCS(n-1, m), LCS(n, m-1))$

0/1 Knapsack Problem

- Utilize the bottom-up dynamic programming method
- Input: n (number of items in the bag), m (bag capacity), w (array describing weight of each item), p (array describing profit of each item)
- Output: bit vector indicating whether an item should be added to the bag or not
- **FORMULA:**
  $V[i, w] = maxV[i-1, w], V[i-1, w-w[i]] + p[i]$

Week 9: Greedy Algorithm

1. Cast the problem where we have to **make a choice and are left with one sub problem** to solve
2. Prove that there is always an **optimal solution to the original problem that makes the greedy choice,** so the greedy choice is safe
3. Use **optimal substructure** to show that we can combine an optimal solution to the sub problem with the greedy choice to get an optimal solution to the original problem

Fractional Knapsack Problem

Greedy choice you want to make: Let j^* be the item with the **maximum value / kg**, v_i/w_j. Then, there exists an optimal knapsack containing $min(w_j, W)$ **kgs** of item j^* Strategy for Greedy Algorithm

- Use greedy-choice property to put $min(w_{j^*}, W)$ kgs of item j^* in knapsack
- If knapsack weighs W kgs, we are done
- Otherwise, use optimal substructure to solve subproblem where all of item j^* is removed and knapsack weight limit is $W - w_{j^*}$

Week 10: Reductions and Intractability

Polynomial Time Reduction

- Definition: A $\leq$ B
- If there is a p(n)-time reduction from A to B for some polynomial function $p(n) = O(n^c)$ for some constant c.

Decision vs. Optimization

- Decision Problems: Given a directed graph G with two given vertices u and v, is there a path from u to v of length $\leq$ k?
- Optimization Problems: Given a directed graph G with two given vertices u and v, what is the length of the shortest path from u to v?

Reductions between Decision Problems (Karp- Reduction)

- Given two decision problems A and B, a polynomial time reduction from A to B, denoted A $\leq$ B, is a transformation from instances α of A to instances β of B such that:
    1. α is a YES-instance for A if and only if β is a YES-instance for B
    2. The transformation takes polynomial time int he size of α
- Suppose that A $\leq$ B. We can infer: **If A cannot be solved in polynomial time, then neither can B**

Week 11: NP-Completeness

NP: The set of all decision problems which have **efficient certifier**
P: The set of all decision problems which have **efficient algorithm** (polynomial time)
NP-Complete: A problem is considered NP-Complete if it is at least as hard as every other NP problem

Proving NP-Completeness

1. Show that $X \in NP$
2. Pick a problem A which is already know to be NP-complete
3. Show that $A \leq_P X$

Circuit Satisfiability Problem

A directed acyclic graph (DAG) with nodes corresponding to **AND, NOT, OR** gates can n binary inputs

CNF-SAT

Given a CND formula Φ, does it have a satisfyig truth assignment?

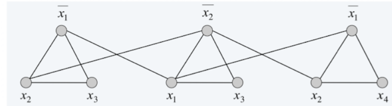
3-SAT

SAT where **each clause contains exactly 3 literals** corresponding to different variables.
$\Phi = (x_1' \vee x_2 \vee x_3) \wedge (x_1 \vee x_2' \vee x_3) \wedge (x_1' \vee x_2 \vee x_4)$
Circuit Satisfiability $\leq_P$ CNF-SAT $\leq_P$ 3-SAT

3-SAT $\leq_P$ Independent Set

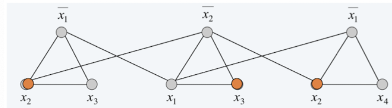
Given an instance Φ of 3-SAT, goal is to construct an instance (G, k) of INDEPENDENT-SET so that G has an independent set of size k **if and only if** Φ is satisfiable
Reduction

- G contains 3 vertices for each clause, one for each literal
- Connect 3 literals in clause in a triangle
- Connect literal to each of its negations
- Set k = number of clauses



$(\overline{x_1} \vee x_2 \vee x_3)$
$\wedge (x_1 \vee \overline{x_2} \vee x_3)$
$\wedge (\overline{x_1} \vee x_2 \vee x_4)$

Suppose Φ is a YES-instance. Take any satisfying assignment for Φ and select a true literal from each class. Corresponding k vertices form an independent set in G



$(\overline{x_1} \vee x_2 \vee x_3)$
$\wedge (x_1 \vee \overline{x_2} \vee x_3)$
$\wedge (\overline{x_1} \vee x_2 \vee x_4)$

Suppose (G, k) is a YES-instance. Let S be the independent set of size k. Each of the k triangles must contain exactly one vertex in S. Set these literals to true, so all clauses satisfied
This proof shows that <u>some instances</u> of INDEPENDENT-SET are as hard to solve as the 3-SAT problem. This <u>DOES NOT MEAN</u> that all instances of the INDEPENDENT-SET problem are hard
Therefore, if there isn't a poly-time algorithm that solves ALL 3-SAT instances, there is no poly-time algorithm that solves ALL INDEPENDENT-SET instances.

Max-Clique

- Given an undirected graph G and an integer k, whether there exists a clique of size at least k or not G?
- Show Max-Clique is NP-Complete
- Must prove that Independent Set $\leq_P$ Max-Clique

Vertex Cover Problem

- Optimization version: Find the vertex cover of the smallest size
- Decision version: Does a vertex cover of size $\leq$ k exist?
- Can be reduced to the Independent Set Problem

VC $\leq_p$ IS

Theorem: If $X \subseteq V$ is a vertex cover of G **then** $V \setminus X$ is an independent set of G
Proof: Let $Y = V \setminus X$
Consider any two vertices $u, v, \in Y$
Is it possible that $(u, v) \in E$? **NO!**
Reason: $u \notin X$ and $v \notin X$ so if $(u, v) \notin E$, then X is not a vertex cover.
Hence, for each $u, v \in Y, (u, v) \notin E$
$\rightarrow Y$ is an independent set of G ———————— **Theorem:**
If X is an independent set of G, then $V \setminus X$ is a vertex cover of G
Proof: Let $Y = V \setminus X$
Consider any edge $(u, v) \in E$
Since X is an independent set, so at most one of u, v can be in X
So at least one of u, v must be in Y
$\rightarrow Y$ is a vertex cover of G

NP-Complete Problems

- Circuit-SAT
- CNF-SAT
- 3-SAT
- Independent Set
- Vertex Cover
- Max-Clique
- Hamiltonian Cycle (directed and undirected)
- Traveling Sales Person
- Subset Sum
- Knapsack
- Hitting Set

Sorting Algorithms

Sorting Algorithm	Time Complexity - Best	Time Complexity - Worst	Time Complexity - Average	Space Complexity
Bubble Sort	n	n^2	n^2	1
Selection Sort	n^2	n^2	n^2	1
Insertion Sort	n	n^2	n^2	1
Merge Sort	nlog n	nlog n	nlog n	n
Quicksort	nlog n	n^2	nlog n	log n
Counting Sort	n+k	n+k	n+k	max
Radix Sort	n+k	n+k	n+k	max
Bucket Sort	n+k	n^2	n	n+k
Heap Sort	nlog n	nlog n	nlog n	1
Shell Sort	nlog n	n^2	nlog n	1

The End