CS 341 Cheatsheet

Divide and Conquer

Asymptotic notation

Lower bound: $f(n) \in O(g(n))$

- $\exists c > 0, n_0, \forall n \geq n_0, f(n) \leq cg(n)$

Upper bound: $f(n) \in \Omega(g(n))$

- $\exists c > 0, n_0, \forall n \geq n_0, f(n) \geq cg(n)$

Equivalent: $f(n) \in \Theta(g(n))$

- $\exists c_1, c_2, n_0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$
- $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c; c \in (0, \infty)$

Master theorem

$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y) \Rightarrow$

$$T(n) \in \begin{cases} \Theta(n^x) & y < x \\ \Theta(n^y \log n) & y = x \\ \Theta(n^y) & y > x \end{cases}$$

where $x = \log_b a$

Graph Algorithms

BFS

- Traverse graph G level by level from source v_0

```
algorithm BFS(G, v_0)
  Q = [v_0]
  level(v_0) = 0
  while Q is non-empty
    v = Q.pop(0)
    for u ∈ N(v)
      if level(u) is undefined
        level(u) = level(v) + 1
        Q.append(u)
```

- Finds shortest paths tree from v_0
- Q contains vertices in non-decreasing level order
- level(v) is length of shortest path from v_0 to v
- Non-tree edges cross between branches and differ by at most one level
- Check if G is bipartite
  - If cross edge exists on the same level, G is bipartite
- $O(n + m)$ time

DFS

- Traverse graph G depth-first from v_0
- White path lemma: any vertex connected to v_0 is visited in DFS traversal
- Non-tree edges (undirected): back edges
- Non-tree edges (directed): forward edges, back edges, cross edges
- Produces topological sort of DAG
- Strongly connected
  - Kosaraju's algorithm
  - Starting from a vertex, run DFS forwards and backwards
- $O(n + m)$ time

Minimum Spanning Tree

- Find least weighted spanning tree

```
algorithm Kruskal-MST(G)
  sort E by weight
  for e ∈ E
    if e does not make a cycle with T
      add e to T
```

- Use union-find data structure to partition edges into sets of connected components
- $O(m \log n)$ time

Greedy Algorithms

- Algorithms with no backtracking and lookahead

Interval Scheduling

- Given intervals I with start times s and finish times f, find $S \subset I$ of pairwise disjoint intervals of maximum size
- Greedy solution: sort intervals by finish time and select valid intervals in order

```
algorithm Interval-Scheduling(I, s, f)
  sort I by finish time
  S ← {}
  for i = 1 to n
    if i is pairwise disjoint with all S
      S ← S ∪ {i}
```

- $O(n \log n)$ time

Interval Coloring

- Given intervals I with start times s and finish times f, find $S \subset I$ of pairwise disjoint intervals of maximum size
- Greedy solution: sort intervals by finish time and select valid intervals in order

Minimal Lateness Scheduling

- Given n jobs where job i requires t_i to complete and has a deadline at a_i.
- Greedy solution: do earliest deadline job first

Fractional Knapsack

- Given n items with weights w_i and values v_i, find $S \subset I$ of items of maximum value subject to weight constraint W
- Greedy solution: sort items by value-to-weight ratio and select items until weight limit is reached

Dynamic Programming

Edit Distance

- Given strings x and y, find minimum edit distance using replace, add, delete operations
- DP solution
  - Let $M[i, j]$ represent the minimum edit distance between $x[1..i]$ and $y[1..j]$

$$M[i, j] = \min \begin{cases} M[i-1, j-1] + c_r & \text{replace } x[i] \to y[j] \\ M[i-1, j] + c_i & \text{delete } x[i] \\ M[i, j-1] + c_d & \text{insert } y[j] \end{cases}$$

  - $M[0, j] = j$
  - $M[i, 0] = i$
  - $O(mn)$ space and time

Optimal BST

- Given probability distribution of accesses p, compute BST which minimizes expected search depth
- DP solution
  - Let $M[i, j]$ represent the optimal BST for $[i..j]$

$$M[i, j] = \min_{k \in [i, j]} (p_k + M[i, k-1] + M[k+1, j])$$

Independent Set