Graphs
Assuming no self-loops, no multiple edges.
Undir: $m \leq \binom{n}{2}$, $\sum_v deg(v) = 2m$
Dir: $m \leq n*(n-1)$, $\sum_v indeg(v) = \sum_v outdeg(v) = m$
Subgraph: result of removing an edge, $V' \subseteq V, E' \subset E$
Induced SG: result of removing node. Is subgraph, $e \in E' \leftrightarrow (e \in E \wedge (u,v)) \in V$.
u connected to v: $(u \sim v) \rightarrow \exists$ path from u to v
G connected if $(u \sim v)\ \forall (u,v)$
Representations
Adj. Matrix: row 1 = outgoing edge, col 1 = incoming edge
- Undir: $G = G^T$
- Find a neighbour: $O(n)$
- Access$(v) = O(1)$, traverse $V = O(n)$
- Traverse all edges of $v = O(n)$
- Traverse all edges of $G = O(n^2)$
- $O(V^2) = O(n^2)$ space
Adj. List: $\sum_v outdeg(v) = E$
- $O(V+E) = O(n+m)$ space
- Find a neighbour: $O(1)$
- Traverse all edges of $v = O(|neighbours(v)|)$
- Traverse all edges of $G = O(m)$
- Access$(v) = O(1)$, traverse $V = O(n)$
Trees
Tree: connected, acyclic graph
- Add edge $\rightarrow$ cycle; Remove edge $\rightarrow$ not connected
- n - 1 edges
Forest: graph with trees as connected components
Spanning tree: $T \subseteq G$ S.T $V(T) = V(G)$
Undir. is a tree $\leftrightarrow$ connected and E = V - 1
BFS
- $O(n+m)$ time (list), $O(n^2)$ (matrix)
- $\Theta(n)$ space for both list and matrix
- Finds all nodes, finds shortest path from s to all others
DFS
- $O(n+m)$ time (list), $O(n^2)$ (matrix)
- $O(n+m)$ space
- Finds all nodes, finds shortest path from s to all others
Edge Classes
The edge (u,v) refers to the FOREST, not the graph.
- Tree: $(u,v) \in$ Forest
- Forward: v is descendant of u
- Back: v is ancestory of u (back = fwd in undirs)
- Cross: none of the above are true.
- Und: DFS tree/fwd only, BFS tree/cross only
- Dir: DFS: all edges, BFS: no fwd edges
DAGs
- Source: no incoming edges
- Sink: no outgoing edges
- Multiple sources/sinks are possible. At least one of each in every DAG.
- Source & Sink $\rightarrow$ no cycles
- A Digraph is a DAG $\leftrightarrow$ DFS has no back edges
Toposort
- Produces ordering s.t $(u,v) \in E \rightarrow u$ appears before v in the ordering
- Digraph has a Toposort $\leftrightarrow$ it is a DAG
- Run DFS, order in decreasing order of finish time
- $O(n+m)$
- Orders are not unique.
SCCs
- Run DFS. Run DFS on G^T, in decreasing order of ftime. The forests of the transpose DFS are the SCCs.
- Independent of toposort ordering

Minimum Spanning Trees
- Spanning tree w/ minumum weight
- DFS and BFS build spanning trees, but *not* necessarily the MST.
- Optimal Substructure: if $T =$ MST of $G \rightarrow T[U]$, where $U \subset V$, if $T[U]$ is connected it is an MST.
Prim: add best vertex
Kruskal's Algorithm $(O((m+n)\log(n)))$
- Only consider edges that do NOT create a cycle (safe edges)
- Add lowest-cost edge on each iteration
- Has greedy-choice property
Edge Switching
- Let $e' \notin T$, where $T \cup \{e'\}$ has a cycle. For any $e \in E$, where e is in the cycle, $T \cup \{e'\} - \{e\}$ is a tree.
Single-Source Shortest Paths (SSSPs)
- The problem: find the min-cost path from source s to each vertex v.
- Shortest path is at most of length $n-1$.
Relax
If $d(v) > d(u) + w(u,v)$, *then* $d(v) \leftarrow d(u) + w(u,v)$
Dijkstra
It's kind of like A^* search!
Algo: *Init.* $S = \emptyset$; *Init.* $d(v) = \infty$, $\forall v \neq s$, $d(s) = 0$; *Init.* Min-Queue, keyed by $d(v)$. *While* $Q \neq \emptyset$, *extract* $u = \min(Q)$, *Add* u to S; *Relax* all edges (u,v).
- List + binary heap (as a min-PQ): $O((m+n)\log(n))$
- Matrix: $O(n^2 + m\log(n))$
- Fib heap: $\Theta(m + n\log(n))$
- No negative cycles (undir), no negative edges (dir)
Properties
Subpath Optimality: If $P = (s, ...u, ...v, ...t)$, then each subpath is the shortest corresponding path.
Triangle Inequality: $d(u,w) \leq d(u,v) + d(v,w)$.
Relaxing an Edge
- $d(v)$ only updated w/ relax $\rightarrow d(v) \geq d(s,v)$
relax(u,v): if $d(v) > d(u) + w(u,v)$:
    $d(v) \leftarrow d(u) + w(u,v)$
Bellman-Ford $O(VE)$
- Can handle negative edge weights (but not negative cycles!)
- Updates distances OFFLINE.
Algo: *Init.* $d(v) = \infty$, $\forall v \neq s$, $d(s) = 0$; *For* $i \in \{1, ..., n-1\}$, *For* $(u,v) \in E$, *Relax* edges (u,v). *Endfor. For* $(u,v) \in E$; *If* $d(u) + w(u,v) \leq d(v)$, *Return* FALSE. *Else* return TRUE.
- Use a DP table, dim $= 0 : (n-1) \times n$
- $d_i[v] \leftarrow \min(d_{i-1}[v], \min_{u \in Neigh(v)}[d_{i-1}[u] + w(u,v)])$
- Order irrelevant to final product.
Floyd-Warshall $O(n^3)$
- $V \times V$ table, $v_1, ...v_n$.
- Allows negative weights, no negative cycles.
- All-pairs shortest path (APSP)
Suppose $V = \{1, 2, \ldots, n\}$.
Then, $d[i,j,k]$ = distance of shortest path from i to j, such that all intermediate vertices $\in \{1, 2, \ldots, k\}$.
- Case 1: don't need vertex k: $d[i,j,k] = d[i,j,(k-1)]$.
- Case 2: need vertex k: $d[i,j,k] = d[i,k,(k-1)] + d[k,j,(k-1)]$.
- So, $d[i,j,k] = \min($Case 1, Case 2$)$
- Base cases: $d[i,j,0] = w(i,j)$, $d[i,i,k] = 0$.

Greedy Algorithms
Optimal Substructure: an optimal choice must be included in *an* optimal solution, but not necessarily all of them.
Fractional Knapsack
v_i = profit per unit weight b_i/w_i
$x_i = \min(w_i, W_{\text{remaining}})$
add x_i amount of item i
$W_{\text{remaining}} \leftarrow W_{\text{remaining}} - x_i$
Choose item with highest v_i
X^* is unique $\rightarrow$ all $x^* \in X^*$ are saturated.
Job Scheduling
- Earliest-Start-Time-First (ESTF)
Sort the n jobs by start time $O(n\log(n))$; iterate n times, and on each iteration find the right machine, schedule the job, and update the times. Total runtime: $O(n\log(n))$.
Alternatively use a min-heap: also $O(n\log(n))$.
Activity Selection
- Earliest-Finish-Time-First (EFTF); $\Theta(n\log(n))$
Dynamic Programming
Integral Knapsack
- n items: table with $0 : n-1$ rows, $0 : W$ columns. D is remaining capacity, W is total capacity.
- $\forall D,\ A[0, D] = 0$
- $\forall i,\ A[i, 0] = 0$
- Else, $A[i, D] = \max(A[i-1, D], v_i + A[i-1, D-w_i])$
Rod Cutting
- $r_0 = 0$
- Else, $r_n = \max_{i\in 1:n}(p_i + r_{n-i})$
Longest Common Subsequence
- $O(nm)$
- $len(X) = n,\ len(Y) = m.$
- Table: $0 : n \times 0 : m.$
- $A[i, j] = \max(D[i-1, j], D[i, j-1], 1 + D[i-1, j-1])$
- Last one only if $X[i] = Y[j]$.
MCM
- n matrices
- $d \in \{0, \ldots, n+1\}$
- $A[i, j] = \min_{i\leq k<j}(A[i, k] + A[k+1, j], + d_{i-1}d_k d_j)$

Asymptotic Cheatsheet
For some set S,
If $f(n) \in S(g(n))$ and $g(n) \in S(h(n))$ then $f(n) \in S(h(n))$.
$\sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r} = \frac{r^{n+1}-1}{r-1}$
$n! = \sqrt{2\pi n}(\frac{n}{e})^n$
Limits:
$\lim_{n\to\infty} \frac{h(n)}{f(n)} = 0 \rightarrow h(n) \in o(f(n))$
$\lim_{n\to\infty} \frac{h(n)}{f(n)} = k > 0 \rightarrow h(n) \in \Theta(f(n))$
$\log^k(n) \in o(n^\varepsilon),\ \forall k, \varepsilon > 0$
$n\log(n) \in \Theta(\log(n!))$
Master Theorem
Let $T(n) = aT(\frac{n}{b}) + f(n)$
Bottom-Heavy: Leaves dominate runtime
- $f(n) \in O(n^{\log_b(a)-\varepsilon}) \rightarrow T(n) \in \Theta(n^{\log_b(a)})$
Balanced: Leaves and internal nodes do equal work
- $f(n) \in \Theta(n^{\log_b(a)} \log_n^k) \rightarrow T(n) \in \Theta(n^{\log_b(a)} \log^{k+1}(n)),\ k \geq 0$
Top-Heavy: Eariler nodes dominate runtime
- $f(n) \in \Omega(n^{\log_b(a)+\varepsilon})$, and $af(\frac{n}{b}) \leq \delta f(n)$, $\delta < 1 \rightarrow T(n) \in \Theta(f(n))$
Heaps
Max-Heap Property: $A[\text{Parent}(i)] \geq A[i]$
Heap of n keys has height $\lfloor \log(n) \rfloor$
buildMaxHeap: $O(n)$
extractMax: $O(\log(n))$
heapSort: $\Theta(n\log(n))$
QuickSort
- Partition: $O(n)$, returns index of pivot.
- Worst case: $O(n^2)$
- Average and best case: $O(n\log(n))$
- For any split of constant ratio, $\Theta(n\log(n))$
Randomized QuickSort
- Pivot selected uniformly at random
- Average, best, and expected worst-case runtime: $\Theta(n\log(n))$
Sorting Lower Bound
- Any comparison-based sorting algorithm requires $\Omega(n\log(n))$ comparisons in the worst case.
- A binary tree with t leaves has at least $1+\log(t)$, or eqiuvalently, a height of at least $\log(t)$.
- So in other words, a decision tree has at least $n!$ leaves. It follows that the height of the tree is at least $\log(n!)$.
- Height = edges
- levels = edges + 1
Binary Search Trees
- Right-(Left)-Rotate: x becomes the right (left) child of new root, usually its left (right) child.
AVL Trees
- Height-balanced: $|h_l - h_r| \leq 1$
- $h = O(\log(n))$
-