Theory

Summations

$$\sum_{k=m}^{n} c = (n-m+1)c \qquad \sum_{k=1}^{n} k = \frac{n(n+1)}{2} \qquad \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=0}^{n} a^k = \frac{a^{n+1}-1}{a-1} \qquad \sum_{k=1}^{n} ka^k = \frac{a-(n+1)a^{n+1}+na^{n+2}}{(a-1)^2}$$

Master Theorem

Let $a \geq 1$ and $b > 1$ be constants $f(n)$ a function and $T(n)$ be a recurrence defined by $T(n) = aT(\frac{n}{b}) + f(n)$. Then $T(n)$ has the following asymptotic bounds.

1. $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for $\epsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$.
2. $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$
3. $f(n) = \Omega(n^{\log_b a + \epsilon})$, $\epsilon > 0$, $af(\frac{n}{b}) \leq cf(n)$ for $c < 1$ and all large n, then $T(n) = \Theta(f(n))$.

Runtime classes

For constants c and n_0 find values that make the following statements true.

Little O (o): Strictly upper bounded by. $f(n) : c > 0 \exists n_0 > 0 : 0 \leq f(n) < cg(n) \forall n \geq n_0$.

Big O ($\mathcal{O}$): Upper bounded by. $f(n) : c > 0 \exists n_0 > 0 : 0 \leq f(n) \leq cg(n) \forall n \geq n_0$.

Big theta (Θ): Both $\mathcal{O}$ and Ω.

Big omega (Ω): Lower bounded by. $f(n) : c > 0 \exists n_0 > 0 : 0 \leq cg(n) \leq f(n) \forall n \geq n_0$.

Little omega (ω): Strictly lower bounded by. $f(n) : c > 0 \exists n_0 > 0 : 0 \leq cg(n) < f(n) \forall n \geq n_0$.

Divide and Conquer

Closest Points in 2-d space

Sort point in one dimension

Recurse to split in half finding closest points in each half

Check within min_distance of center for points that are closer.

Longest subvector

Recursively split in half and find the biggest vector in the left and right.

Start at the middle and expand back to start of left and end of right.

Return the biggest subvector in left, right, or straddling left and right.

Dynamic Programming

Max Subsequence

$MS(i) = \max(MS(i-1) + A[i], A[i])$

Walk through starting at $i = 0$. Store the index where you start the sequence.

Rod cutting

p is the price matrix.

$q(i) = \max(q, p[i])$

Store index of cut when we choose to cut.

Longest common subsequence

x, y character sequences, $c[\text{len}(x)][\text{len}(y)]$. Build matrix using:

$$c[i,j] = \left\{ \begin{array}{ll} c[i-1,j-1]+1 & \text{if } x[i] = y[j] \\ \max(c[i,j-1], c[i-1,j]) & \text{else} \end{array} \right\}$$

Traverse backward following maximum path. A diagonal jump means the characters match.

Greedy Algorithms

Prim's Algorithm

Remove a node from the queue and travel down its shortest edge.

Kruskal's Algorithm

Sort edges by weight.

From low to high, add each edge to the graph if it doesn't create a cycle.

Activity Selection

Sort based on end times, choose the event that has the soonest end time.

Huffman Codes

While there are at least two things in the min heap.

Make a tree out of the two things and give this node the weight of the sum of its left and right children. Put this node back in the tree.

Repeat until only one node is left and return it.

Minimum Spanning Trees

Prim's and Kruskal's algorithm

Graph Algorithms

Dijkstra

Start with source node

Update weights of all connected vertices, travel to vertex that hasn't been visited and has minimal score.

Bellman-Ford

Start with source node

Travel down every edge, updating weights and adding newly found edges to the set of edges.

Return false if there is a negative weight cycle.

Floyd-Warshall

Start with the matrix containing the weights between directly connected vertices.

For each vertex (indexed by i), update the matrix with the shortest distance between x and y given that the path is through i.

Ford-Fulkerson Algorithm

Calculates maximum flow through a graph.

Start with 0 flow.

while there exists an augmenting path find an augmenting path compute bottleneck capacity increase flow on path by bottleneck capacity

Johnson's Algorithm

Insert phantom node and connect it to all vertices with weight of zero.

Run Bellman-Ford algorithm to compute optimal distances between phantom node and all other nodes in the graph.

Create new weights for edges using $\hat{W}(i,j) = W(i,j) + h(s,i) - h(s,j)$

Remove phantom node and run Dijkstra's on each node in the graph. Store these results.

Compute original lengths by taking $W(i,j) = \hat{W}(i,j) - h(s,i) + h(s,j)$.

String Matching

Rabin-Karp algorithm

Compute hash.

Remove leftmost digit.

Make right digit the next hash value.

Check to make sure matches are real matches and not just a coincidence.

Finite Automaton

Build finite automaton using transition table for the alphabet.

Knuth-Morris-Pratt Algorithm

π table is the longest prefix that is a proper suffix for some substring in the pattern starting at index 0.

If q characters have matched at shift s the next shift to check is $s\prime = s + (q - \pi[q])$.