**Persistent BST:** copy whole tree = O(N) time/space; reuse nodes = O(H) time/space where H = height of path traversed. If balanced BST, H = lgN. Need 2 nodes for rotation. No failfast iterators, no parent links

**Adjacency list of undirected edges:** E+V space

Bipartite:verts dived into 2 disjoint sets (DFS/BFS in E+V time)

Path between v and w? DFS and BFS in E+V time

Shortest Path b/t v and w:BFS in E+V time

Bridge:edge that would disconnect endpoints if removed; **2ec comp:** # comp if bridge removed

Search in Digraph: DFS and BFS in O(V+E) time and O(V) space

Multi-source shortest paths: use BFS and enqueue all source verts

**Topological sort:** all DEdges point same way, reverse post order (opp recursion fin); if cycle, no topo

**Kosarsaju-Sharir:** Kernal DAG (strong comp in 1 vert), finds strong comp in E+V time/space, DFSx2

Greedy: makes local-looking decisions and never looks back (not always optimal: min weight vert cover)

**Greedy:** any cut, min weight crossing edge is in MST, correct if equal weights

**Kruskal’s:** add edges in ascending weight (PQ) unless cycle. Sort in O(ElgV) time, O(E) space; check if cycle using UF in O(Elg\*V) time, O(V) space; finds MST in O(ElogE) time

**Prim’s:** lazy-keep every frontier edge on PQ, insert/delMin=O(ElgE) time, PQ size = O(E).

eager-find lightest frontier edge, add to tree, if find better, update item in indexed PQ. PQ size = O(V)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PQ type** | **insert** | **delMin** | **Δ entry** | **total** | **pros/cons** |
| unordered arr | 1 | V | 1 | V^2 | dense graphs |
| binary heap | logV | logV | logV | ElogV | fast for sparse |
| d-ary heap | log(b.d)V | dlog(b.d)V | log(b.d)V | Elog(b.E/V)V | best d = E/V |
| Fibonacci heap | 1 | logV | 1 | E+VlogV | best in theory |

**Euclidean:** MST of points in plane, all distances = 0.5N^2, use geometry = cNlogN

**Boruvka’s:** each comp chooses lightest edge, k passes = at most V/2^k verts, at most lgV phases, use UF to keep track of comp/cycles. O(ElgV) time and O(V) space

Single source (1:all); single sink (all:1); source sink (1:1); all pairs

Relax: distTo[w] <= distTo[v] + e.weight()

**Dijkstra’s:** (no neg edges) indexed PQ of verts by dist, relax all edges from that vert, lazy space = O(E), eager space = O(V). O(ElogV) time

**Topo Sort:** (no cycles) verts in topo order, relax edges from vert. O(E+V) time, seam carving

Longest paths: negate weights, use Topo Sort (parallel job scheduling: critical path)

**Bellman-Ford:** (no neg cycles) relax each edge, repeat V times; after k times found all paths using at most k edges. Computes SPT in O(EV) time. Can use FIFO to only relax changed verts

Dynamic programming: compute value of distTo(v) after i iterations

Arbitrage detection: weight of edge = -ln(exch rate), find negative cycle

**Repeated Dijkstra’s:** run from every vert; single pass \* V for time and space bounds

**Floyd-Warshall:** allows neg edges, matrix describes graph, simple code V^3 time, V^2 space

* K+1 not used:
* K+1 is used:

**Johnson’s:** good for sparse, run BF to get rid of neg edges: newHeight = oldHeight + h(u) – h(v), then use repeated Dijkstra’s

Reverse string: O(N^2) time; Compare strings: O(shorter string) but often sublinear

**Key-Indexed Counting:** freq array of size R+1, compute cumulative, traverse input and increment location. O(N+R) time/space, beats sorting lower bound of ~NlgN comparisons, stable

**LSD Radix:** same length, do key-indexed from right to left. 2W(N+R) time and N+R space, stable

**MSD Radix:** do key-indexed from left to right, recursion, each func call needs own helper array. Slow for small subarrays, can be sublinear. w.c. time = 2W(N+R), typical = Nlog(b.R)N. N+DR space where D = length of longest prefix match/func-call stack depth, stable

**3-Way Radix QS:** in place, recursive 3 way partition, random = ~2NlnN comps, w.c. time = 1.39WNlgR, typical = 1.39NlgN, space = logN + W, good for var len and big R

**Suffix arrays:** sort in O(N^2logN) time

Trie: search tree, keys are strings, store chars in nodes, char-based ops, logN chars examined

Prefix match, wildcard match, longest prefix, compressed trie (child w/o sib merges w parent)

**R-Way:** miss = log(b.R)N, hit/insert = L, space = N(R+1), fast, wastes space

**Ternary:** miss = lnN, hit/insert = L+lnN, space = 4N, balanced = L+logN, naïve

**Choi:** use hashtable for each node, char is in table, fast and low space but no ordered ops

**Suffix tree:** O(N) space/construction, suffix link=internal node – first char appears elsewhere

**Brute force:** slow, little space, w.c. = MN char compares, need backup for every mismatch or buffer

**KMP:** avoids backup, use DFS; if in state k, found match of first k chars. If don’t get right letter do what trailing state would do. Time = O(MR+N), space = O(MR). if use NFA, correct char go right, wrong char go left time = O(N), space = O(M)

**Boyer-Moore:** scan pattern from right to left, skip up to M chars, mis not in pat: on char after mis. Mis in pat, alight text w rightmost in pat OR i++. O(N/M) time on rand, O(MN) wc, O(M) space

**Rabin-Karp:** hash matching, Rolling hash in O(1) bc related by algebra. If Q~MN^2, prob of false collision~1/N. w.c. time = O(MN), typical = O(N), space = O(1). MC: just return ; LV: check first

**Run-Length:** exploits runs, use 4 bits to indicate how many zeros/ones alternating

**Huffman:** exploits frequencies, no code is prefix of another. Make freq table, put nodes in PQ, merge lightest. Left = 0, right = 1; Expansion: pre-order, 0=node, 1=leaf, N+RlogR time

**LZW:** adaptive model, start at 81, trie with first layer given for compression. If use value immediately after creating it, last char is first char

**QF**: u=O(N) f=O(1). **QU**: O(N) u/f. **WQU**: O(lgN) u/f, O(N) const/sp. **WQUPC**: wct O(N+Mlg\*N)

**BST**: w.c. sid: O(N). avg sid: rad(N), expected height: O(rad(N)), 2NlgN + O(N)

**Binary Heap**: insert/delMax O(lgN), ht = lgN bc balanced. Insert=lgN comps, delete=2lgN comp

**d-ary Heap**: height = log(b.d)N, swim is faster O(log(b.d)N), sink is slower O(dlog(b.d)N)

**Treaps**: w.c. height sid O(N), expected heigh sid lgN, expected depth lgN,

**2-3**: wc sid O(lgN) all 2 nodes, bc 0.63lgN all 3 nodes

**LLRB**: wc height sid O(2lgN), avg heigh sid O(lgN)

**intIterator**: O(N^2) time and O(N) space. **stackIterator**: O(N) time and O(H) space

**Heapsort**: O(N) construction, O(1) space, 2NlgN comps, delAllMax in O(NlgN) time

**Mergesort**: w.c. time O(NlgN), O(N) space, NlgN array accesses

**Quicksort**: w.c. 0.5N^2 comps, NlgN avg comps, P[i and j compared] = 2NlgN + O(N)

Every bin at least one ball after ~MlnM throws. After M balls, max in bin is logM/loglogM

**Separate chaining**: wc sid N, avg sid 3-5. **2probe hashing**: E[len of longest chain] = loglogN

**Linear probing**: wc O(N), avg 3-5. **Double hashing**: eliminates clustering