Listen

- *0 = Anker; *m->n = m+1; *m->n=NULL
- *0+*m = Anker; *m->n = m+1; *m->n=NULL
- ^^ auch mit Dummy für leichteres ändern von *head und *tail
- *0 = Anker; *m->n = m+1; *m->0 (ringförmig)
- ^^ auch doppeltverkettet (next+prev)
- Stack = LIFO; Queue = FIFO

Bäume

Binär Baum: Grad=2; leer o. rechter binTB+Wurzel+linker binTB
BB als Array: Vorgänger = i/2; Nachfolger = 2*i && 2*i+1
Löschen mit zwei Nachfolgern: max aus linkem/min aus rechtem TB
Baum durchlaufen (traversieren)

- Hauptreihenfolge = preorder/fix = WLR
- Nebenreihenfolge = postorder/fix LRW (m. Stack direkt rechenbar)
- Symmetrische Reihenfolge = inoder/fix = LWR

Vollst. BB: $h = log_2(ba) = log_2(ik+1) = log_2(gk+1) - 1$

Interne Pfadlänge: $\text{ipl} = \sum_i \text{tiefe}(i) + 1$ (Vgl. zum Finden aller Kn.)

Durchschn. Suchpfadlänge: $dpl = \frac{ipl}{ik}$; $ik = 2^h - 1$; $ba = 2^h$; $gk = 2^{h+1} - 1$

Vergleich zweier Bäume: Rekursiver Vgl. Wurzel → l. TB → r. TB

Sortieren

stabiles Sort. = Reihenfolge gleicher Werte bleibt erhalten
Speicherbedarf = insitu (Array)/ exsitu (Liste)

Verfahren	Vergleiche	Austausche	stabil	insitu
Bubble (sort)	O(N^2/2) O(N)	O(N^2/2) O(N)	ja	ja
Selection	O(N^2/2)	O(N)	nein	ja
Insert (sort.)	O(N^2/4) O(N)	O(N^2/8) O(N)	ja	ja
Shell (opt. Dist.)	O(N log (N)^2) O(N^1.2)	O(N log (N)^2) O(N^1.2)	nein	ja
Quick (worst-case)	O(N log N) O(N^2)	O(N log N) O(N^2)	nein	ja
Merge	O(N log N)	O(N log N)	ja	nein

Automaten und Sprachen

BNF: <XYZ> ::= <ABC> | | _<ABC>
nichtdeterministischer → deterministische Automaten

- Schritt 1: Aufspalten von Kanten mit Zeichenketten
- Schritt 2: Zusammenführen von Kanten und Zuständen mit gleichem Zeichen und Ausgangszustand (jeder Knoten, der einen Finalzustand enthält ist selbst Finalzustand; Fehlende Kanten führen zu einem Fehlerzustand)

Graphen

- Kantenzug: Folge von inzidenten (benachbarten) Kanten
- Weg: Kantenzug, ohne mehrfach vorkommende Knoten
- Kreis: geschlossener Weg (alle Knote → Hamilton Kreis)
- Eulerzug: Kantenzug, der alle Kanten genau einmal enthält
- Spannbaum: Alle Knoten, Kanten reduziert → Baum
- zusammenhäng. G: jeder Knoten mit jedem verbunden $\frac{(n-1)(n-2)}{2}$

Tiefensuche (für Zusammenhangsprobleme)
Start → Markieren → rekursiver Aufruf für alle nicht markierten Nachbarknoten
Breitensuche (für Distanzprobleme)
Start → Markieren → markieren aller Nachfolger, speichern dieser in einer Warteschlange → rekursiver Aufruf mit erstem Knoten aus der Warteschlange
Zusammenhangskomponenten
Durchlaufen aller Knoten → unmarkiert? → Tiefensuche (Knoten markieren) → Zahl der Komponenten erhöhen
Transitive Hülle (Erweit. um indirekt erreichbare Kanten)
Warshall: `k= i = j = 0..n;` (3x for)
`a[i][j]=a[i][j]||(a[i][k]&&a[k][j]);`
Alle kürzesten Pfade (zw. zwei Knoten)
Floyd: `k= i = j = 0..n;` (3x for)
`a[i][j]=min(a[i][j], a[i][k]+a[k][j]);`
Minimaler Spannbaum (Prim)
Start → kürzeste Kante eines zum Teilbaum adjazenten unmarkierten Knoten → als TB markieren → wiederholen
Travelling Salesman (Approximation mit MST)
MST → Kanten verdoppeln → Eulerzug → minimaler Hamilton-Kreis (besuchte Knoten überspringen)

Suchen

Auswahlproblem: i-größten/kleinsten Keys einer Sequenz

- mehrfaches Durchlaufen ø → jeweils min/max entfernen

Sequentielle Suche: alles einmal seq. Durchlaufen
Binäre Suche: Sequenz sortiert → mittlerer Key entscheidet, welche Hälfte weiter durchsucht wird (rekursiv)
Interpolationss.: bin. Such. mit geschätzt verkl/grö. Bereich
Selbstanord. List: verschieben zugegriffener Elem (v. Met.)
Hashverfahren: Hash als Arrayindex → kein Suchaufwand

- Divisions-Rest oder Multiplikative Methode
- Gleicher Hashwert → Kollision (Synonym) → Behandlung (Überläufer o. Sondieren)

Sondierungsfunktion S(j, k): (H(k)-S(s,k)) % m
Knuth-Morris-Pratt: Präfixtabelle → Offset des Mismatch wird Index des Zeichens darunter

0	1	2	3	4	5	
a	b	c	a	b	a	
-1	0	0	0	1	2	1