statisches Wörterbuch:
Hashing ? O(1) für lookup
sortiertes Array + Binärsuche $O(\log_2 n)$ für lookup

geordnetes Wörterbuch:
alle möglichen Baumstrukturen

Binärer Suchbaum
$Wert(v_{links})<Wert(v)<Wert(v_{rechts})$
lookup in O(Tiefe)=O(log n)
remove(v): falls v 2 Kinder hat, ersetzen durch linkestes Kind im rechtesten Teilbaum
Insert, Remove, Lookup O(Tiefe) ? erwartet O(log n), worst-case O(n)

AVL-Baum
$|Tiefe(T_L(v) - T(T_R(v)| = 1$,
$Tiefe(\emptyset)=-1$, $Tiefe(n)<= 2 \log_2 n$, $n>=2^{T/2}$
Lookup wie bei Binärem Suchbaum, Lazy-Delete, Insert wie bei Binärem Suchbaum + Reparatur am Suchpfad, alles in O(log n)
ROTATIONEN

Splay-Bäume
einzelne Operationen vielleicht in O(n), aber insgesamt O(n log n)
Lookup/Insert/Remove: betrachte Wurzel y nach splay(x), einzige Operation splay(x)
ROTATIONEN

(a,b)-Bäume
a>=2, b>=2a-1, alle Blätter haben die gleiche Tiefe, alle Knoten haben <= b Kinder, die Wurzel hat >= 2 Kinder – alle anderen >= a Kinder, jeder Knoten speichert seine Schlüssel in aufsteigender Reihenfolge, jeder innere Knoten mit k Kindern hat genau k-1 Schlüssel, ein Blatt speichert >= a-1 und <= b-1 Schlüssel, unterstützt Binärsuche
falls b=2a-1 ? B-Bäume?
geringe Tiefe
Tiefe a,b-Baum B mit n Knoten

$$\lceil \log_b \frac{n-1}{2} \rceil$$

? $\log_b n-1<=Tiefe(B)<=+1$
lookup Tiefe(T)+1, insert 2(Tiefe(T)+1), remove 4(Tiefe(T)+1)

Hashing
lookup, insert, remove in O(1), aber dafür nicht injektiv ?
Auflösung von Kollisionen
Hashing mit Verkettung:
Listenlänge <= n/m = ?
Hashing mit offener Adressierung:
lineares Austesten: $h_i(x)=(x+i \bmod m)$? Klumpungseffekt
doppeltes Hashing:
$h_i(x)=(f(x)+ig(x)) \bmod m$, mit $f(x)=x \bmod m$ und $g(x)=m' -(x \bmod m')$, sowie m Primzahl und $m'<m$
erwartete Laufzeit 1/(1-?)universelles Hashing : Eine Menge H heißt c-universell, falls für x!=y gilt |h in H | h(x)=h(y)}| <= c(|H|/m)
U={0, .., p-1} mit p prim, dann $H=\{h_{a,b} | 0<=<a,b<p$, $h_{a,b}(x)=((ax+b) \bmod p) \bmod m)$ c-universell mit |U|=p

$$c = \left(\frac{\frac{\lceil |U| \rceil}{m}}{\frac{|U|}{m}} \right)^2 \approx 1$$

Hashing zerstört Ordnungsinformationen

Lookup:
Bäume:
Binäre Suchbäume(O(log n) erwartet),
AVL (O(log n) worst-case,
Splay-Bäume (O(log n) amortisiert),
(a,b)-Bäume
Hashing:
mit Verkettung ?+n /|U|+c
doppeltes Hashing: 1/(1-?)
mit offener Adressierung

Tiefensuche
Preorder + besuchte Knoten markieren O(|V| +|E|)
Baumkanten, Rückwärtskanten (Querkanten, Vorwärtskanten)
Zusammenhangskomponenten für Startknoten s, Zyklenfreiheit ermitteln - alles in O(|V| +|E|)
Alle Querkanten verlaufen von rechs nach links, da Tiefensuchbaum von links nach rechts aufgebaut wird
Zyklenfreiheit gilt, wenn keine Rückwärtskanten existieren
Entscheiden, ob Graph stark zusammenhängend ist über Tiefensuche von s über G und über $G^{reverse}$
Datenstruktur Stack

Breitensuche O(|V| +|E|)
nur Baum und Rückwärtskanten
Bestimmung der kürzesten Wege für ungewichtete Graphen
Datenstruktur Queue

Djkstra O((|V| +|E|) log |V|)
S={s}; distanz[v]={länge(s,v), fallst (s,v) in E, ? sonst}
Solange S != V: {wähle Knoten w in V-S mit kleinstem Distanz-Wert; Füge w in S ein; Berechne neue Distanzwerte aller Nachfolger von w;}
// c=distanz[w] + laenge[w,u]; // distanz[u]= (distanz[u]>c)? c: distanz[u];
Datenstruktur Heap
Korrektheit zeigen über Wiederspruch: distanz[w]=Länge des kürzesten Weges von s nach w; es gibt einen kürzeren Weg p=(s,v_1, ..., v_i, v_{i+1}, ...w) und v_{i+1} ist erster Knoten der nicht zu S gehört:
distanz[w]<=distanz[v_{i+1}]
distanz[v_{i+1}]<=länge(p)
--> distanz[w]<=länge(p)
Ein Teilstück eines kürzesten Weges ist selbst ein kürzester Weg

Minimale Spannbäume
G besitzt Spannbaum ?G zusammenhängend
minimaler Spannbaum hat kleinste Kantensumme unter allen Spannbäumen
Beweis zu Prim: kürzeste kreuzende Kante e' zu S hinzufügen und e" entfernen, falls e' einen Kreis schließt

Prims Algorithmus
S={1}
V != S : u in S, v in V-S, e={u,v} kürzeste kreuzende Kante?S=S vereinigt {v}
Implementierung mit Heap ? O(|V| log |V| + |E| log |V|) = O(|E| log |V|)

Kruskals Algorithmus
Sortiere Kanten nach aufsteigender Länge W=(V,?)
Solange W kein Spannbaum:
bestimme die gegenwärtig kürzeste Kante e={u,v} und entferne e aus sortierter Folge
verwerfe e, falls e in W einen Kreis schließt, ansonsten füge e zu W hinzu
Implementierung UNION-FIND (Vater-Array)? O(|E| log |V| + |E| log |V| + |V|) = O(|E| log |V|)

Divide & Conquer
Bestimme Teilprobleme
Löse die Teilprobleme in der durch den Rekursionsbaum diktierten Reihenfolge (Knoten ist lösbar, wenn seine Kinder bekannt, also Blätter sofort lösbar)

Dynamisches Programmieren
Bestimme Teilprobleme
Löse Teilprobleme wie durch einen gerichteten, azyklischen Graphen diktiert (Knoten ist lösbar, wenn seine direkten Vorgänger bekannt, also Quellen sofort lösbar)

Warshalls Algorithmus zur Bestimmung der transitiven Hülle ($O(|V|^3)$), aber Tiefensuche für jeden Knoten $O(|V|(|V|+|E|))$ besser, falls $|E| << n^2$
Initialisierung des Hüllenarrays
for(k=1; k<n;k++)
for(i=1; i<n; i++)
for(j=1; j<n; j++){H[i][j]=H[i][j] || (H[i][k] && H[k][j]);}

Floyds Algorithmus für APSP-Problem:
for(k=0; k<n;k++)
for(u=0; u<n; u++)
for(v=0; v<n; v++){temp=A[u][k]+A[k][v]; if(A[u][v]>temp) A[u][v]=temp;}
($O(|V|^3)$), aber iterativ Dijkstra in $O(n(|V|+|E|) \log |V|)$ besser für

$$|E| < \frac{n^2}{\log n}$$

Floyds Algorithmus funktioniert, wenn alle Kreise nicht negative Länge haben, Djkstra nicht!

Greedy-Algorithmen
Tue das, was zum gegenwärtigen Zeitpunkt optimal ist - Beispiel:
Nicht-preemptives Scheduling
Sortiere Jobs nach Terminierungszeit t_i

Solange noch Jobs vorhanden:
wähle Job i mit kleinstem t_i und entferne aus sortierter Folge
Führe Job i aus
Entferne alle Jobs j mit $s_j <= t_i$

Huffman-Code: Initialisiere Heap; Füge a aus A in den Heap ein; Heap nach kleinster Häufigkeit geordnet;
Solange Heap mind. 2 Buchstaben enthält: {Entferne Buchstaben a und b gerinster Häufigkeit; Füge neuen Buchstaben c in den Heap ein, mit H(c)=H(a)+H(b); Schaffe neuen Knoten v(c) mit Kindern v(a) und v(b) mit Kante {v(a),v(c)}=1, {v(b),v(c)}=0
O(nlogn)

Binpacking-Problem? Best-Fit: Das i-te Objekt wird in den Behälter gepackt, der am vollsten unter allen Behältern mit freier Kapazität mind. w(i) ist.

Gegebsp.: (n/2 mal 2/5, dann n/2 mal 3/5)
Rucksackproblem
TSP-Problem
wege(u,s){nr[u]=nr++; if (nr==s) gib das nr-Array aus; else{ for(v=0; v<n; v++) if((nr[v]==-1) && (A[u][v])) wege(v,s) /* alle unmarkierten Nachfolger besuchen */ nr--; nr[u]=-1; } }
oder Bestimme minimalen Spannbaum T für G und durchlaufe T in Preorder
Branch & Bound
Backtracking
Simulated Annealing