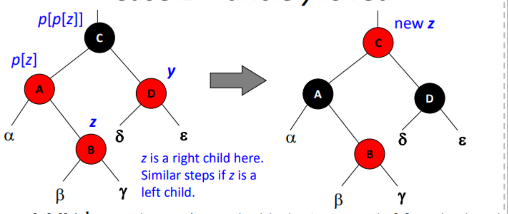
Heaps: height = lg n. Maintain property by downheaping: swap with larger of 2 children. BuildMaxHeap: MaxHeapify from node A.length/2 to 1.

Trees:

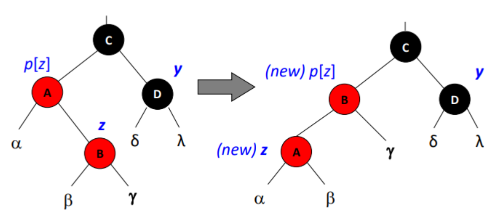
RB trees: bh(x)≤h(x)≤2bh(x). Subtree root x has ≥$2^{bh(x)}$-1 internal nodes. RB tree w n internal nodes has height ≤2lg(n+1)

Case 1 – uncle *y* is red



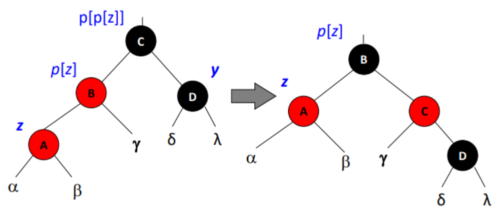
- *p*[*p*[*z*]] (*z*' s grandparent) must be black, since *z* and *p*[*z*] are both red and there are no other violations of property 4.
- Make *p*[*z*] and *y* black ⇒ now *z* and *p*[*z*] are not both red. But property 5 might now be violated.
- Make *p*[*p*[*z*]] red ⇒ restores property 5.
- The next iteration has *p*[*p*[*z*]] as the new *z* (i.e., *z* moves up 2 levels).

Case 2 – *y* is black, *z* is a right child



- Left rotate around *p*[*z*], *p*[*z*] and *z* switch roles ⇒ now *z* is a left child, and both *z* and *p*[*z*] are red.
- Takes us immediately to case 3.

Case 3 – *y* is black, *z* is a left child



- Make *p*[*z*] black and *p*[*p*[*z*]] red.
- Then right rotate on *p*[*p*[*z*]] (in order to maintain property 4).
- No longer have 2 reds in a row.
- *p*[*z*] is now black ⇒ no more iterations.

AVL:
Insert: x lowest node violating AVL
If x right heavy:
  If x's right child right-heavy or balanced: **<- rotation**
  Else: **->** then **<- rotation**
If x left heavy:
  If x's left child left-heavy or balanced: **->rotation**
  Else: **<-** then **-> rotation**
Repeat with x's ancestors

Disjoint sets:

Union by size: Depth of any node ≤ log n
Proof: Union causes depth of node to increase⇒node belongs to smaller tree. Size of smaller tree at least doubles. Can only double at most log n times.■
Union by height: Tree obt. from ubh has height ≤ log n, and # nodes ≥ 2^h.
Proof: Case h = 0, # nodes = 1, ≥ 2^0.
Case ubh tree of height h+1. Then the 2 unioned trees both had height h, and # nodes ≥ 2^h. 2^h + 2^h = 2^(h+1).■

Greedy algorithms:

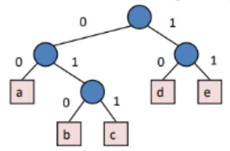
Huffman encoding: Compute freq f(c) of each char c. Assign short code words for high freq.

A **code** is a mapping of each character of an alphabet to a binary code-word

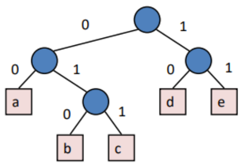
A **prefix code** is a binary code such that no code-word is the prefix of another code-word

An **encoding tree** represents a prefix code
- Each external node (leaf) stores a character
- The code word of a character is given by the path from the root to the external node storing the character (0 for a left child and 1 for a right child)

00	010	011	10	11
a	b	c	d	e



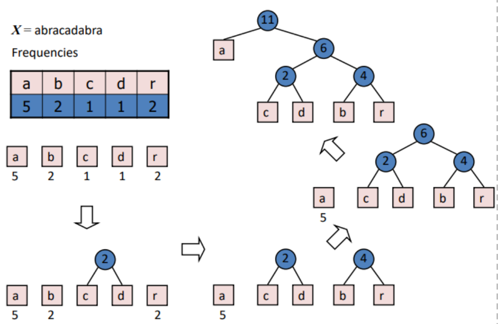
Encoding Example



Initial string: X = acda

Encoded string: Y = 00 011 10 00

Example



Graph algorithms:

DFS: # back edges = # cycles, 0 back edges⇒is DAG.
Parenthesis th: d[u]<f[u]<d[v]<f[v]⇒u and v not descendant of another. d[u]<d[v]<f[v]<f[u]⇒v descendant of u.

Topological sorting and strongly connected components:

White path th: v descendant of u iff ∃ white verts path u→v.
Tree edge: in depth first forest. Back e (u,v): u des of v in dfforest. Foward e (u,v): v des of u, not tree edge. Cross e: other edges, in or b/w diff dftrees.
Top sort: sort nodes from high f time to low f time.
Strongly connected component (SCC): ∀u,v, ∃ paths u→v&v→u.
G^{SCC} is DAG. Determine SCCs of G, SCC(G): 1. Compute f times with DFS(G). 2. Call DFS(G^T) (G^T = G with edges reversed), considering verts in order of decreasing f times. 3. Each DFS tree is a SCC.
If C, C' distinct SCCs in G, (u,v) edge with u in C and v in C'⇒f(C)>f(C')

MST:

Prim's proof: If T≠T′, let ek=(u,v) 1st edge chosen by Prim's not in T′, chosen on the kth iteration of Prim's. Let P path u->v in T′, e* edge in P s.t. 1 endpoint is in the tree generated at the k−1'th iteration of Prim's and the other is not, i.e., one endpoint of e* is u or one endpoint is v, but the endpoints are not u and v.
If weight e*<weight ek, Prim's would have chosen it on its kth iteration, so it's certain w(e*)≥w(ek). In particular, when w(e*)=w(ek), the choice between the two is arbitrary. Whether w(e*) > or =w(ek), e* can be substituted with ek while preserving minimal total weight of T′. This process can be repeated indefinitely, until T′ is equal to T, and it is shown that the tree generated by any instance of Prim's is a MST.■
Unique MST if unique weights: T1/=T2. Let e* min cost edge in T1 not in T2. Removing e* disconnects T1 into 2 comps. e* must be min crossing edge of the 2 comps. By cut property e* then must be in all MSTs, and thus in T2. T1=T2.■

Single source shortest path:

Dijkstra's:
create a heap or priority queue
place the starting node in the heap
dist[2...n] = {∞}
dist[1] = 0
while the heap contains items:
  vertex v = top of heap
  pop top of heap
  for each vertex u connected to v:
    if dist[u] > dist[v] + weight of v-->u:
      dist[u] = dist[v] + weight of edge v-->u
      place u on the heap with weight dist[u]

Dijkstra's proof:
Convergence property: If s→..→u→v is a shortest path from s to v, then after u is added to S and relax(u,v,w) called, then d[v]=δ(s,v) and d[v] remains unchanged.
Loop invariant: at start of each while loop iteration, d[v]=δ(s,v)∀v∈S
Initialization: initially, S=∅, so trivially true
Termination: at end, Q=∅⇒S=V⇒d[v]=δ(s,v)∀v∈V
Maintenance: show d[u]=δ(s,u) when u is added to S in each iteration
Let u first vertex st d[u]≠δ(s,u) when u is added to S. u≠s, since d[s]=0=δ(s,u). There must be some path u→v, thus a shortest path p u→v. Before u added to S, p connects s in S to u in V-S. Let y first vertex on p in V-S, and x predecessor of y. x∈S & u 1st vertex st d[u]≠δ(s,u)⇒d[x]=δ(s,x) when x added to S. Relax(x,y). By convergence prop, d[y]=δ(s,y). y on shortest path s→u&non neg weights⇒d[y]=δ(s,y)≤δ(s,u)≤d[u]. y & u were in Q when u was chosen⇒d[u]≤d[y].d[y]≤d[u] & d[u]≤d[y]⇒d[u]=d[y]⇒d[u]=δ(s,y)=δ(s,u). Contradiction.■

Bipartite graphs:

No odd cycles proof: if G bipartite with vertex sets V1 & V2, every step in a walk takes you either from V1 to V2 or from V2 to V1. To end up where you started, therefore, must take an even number of steps.
Conversely, suppose that every cycle of G is even. Let v0 be any vertex. For each vertex v in the same component C0 as v0 let d(v) be the length of the shortest path from v0 to v. Color red every vertex in C0 whose distance from v0 is even, and color the other vertices of C0 blue. Do the same for each component of G. Check that if G had any edge between two red vertices or between two blue vertices, it would have an odd cycle. Thus, G is bipartite, the red vertices and the blue vertices being the two parts.■
Gale-Shapley:
matching←∅
while there is α∈A not yet matched:
  β←pref[α].removeFirst()
  if β not yet matched:
    matching←matching∪{(α,β)}
  else:
    γ←β's current match
    if β prefers α over γ:

```
    matching ← matching-{(γ,β)} ∪ {(α,β)}
return matching
```

Flow network:
Compute min cut:
1. Run Ford-Fulkerson to compute max flow
2. Run BFS or DFS from s in G_f (residual graph)
3. The reachable verts define set A of cut

Dynamic programming:
Coin change: $f(n) = \min_{j\in\{0:m\}}(1+f(n-c_j))$
Bellman-Ford: negative cycle exist ⇒ shortest path ill defined. At most |V|-1 iterations, else negative cycle.
$d(i,j)=0$ if $i=s, j=0$; ∞ if $i\neq s, j=0$;
$\{d(k,j-1)+w(k,i):i\in Adj(k)\}\cup\{d(i,j-1)\}$ if $j>0$
Knapsack: OPT(i,w)=0 if i=0; OPT(i-1,w) if wi>w; max{OPT(i-1,w), vi+OPT(i-1,w-wi)} otherwise

```
Knapsack(n,W,w1,...,wn,v1,...,vn)
  for w = 0 to W:
    M[0,w] ← 0
  for i = 1 to n:
    for w = 1 to W:
      if (wi > w) { M[i ,w] ← M[i - 1, w] }
      else { M[i, w] ← max{ M[i - 1, W], vi + M[i - 1, w - wi]
} }
  return M[n, W]
```

Pairwise sequence alignment:
c(m,n)=c(m-1,n)+c(m-1,n-1)+c(m,n-1)
(deletion+sub/match+insertion)

Divide and conquer:
Master theorem:
$T(n)=aT(n/b)+f(n)$, $a\geq 1$, $b>1$, $f(n)=n^d\log^p n$, $f(n)>0$, $k=\log_b a$
$T(n)=\Theta(n^d\log^p n)$ if $d>k, p\geq 0$; $\Theta(n^d)$ if $d>k, p<0$; $\Theta(n^d\log^{p+1} n)$ if $d=k, p>-1$; $\Theta(n^d\log\log n)$ if $d=k, p=-1$; $\Theta(n^d)$ if $d=k, p<-1$; $\Theta(n^k)$ if $d<k$

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 1. If $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$.

Case 2. If $f(n) = \Theta(n^k \log^p n)$, then $T(n) = \Theta(n^k \log^{p+1} n)$.

Case 3. If $f(n) = \Omega(n^{k+\varepsilon})$ for some constant $\varepsilon > 0$ and if $a\,f(n/b) \leq c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Strassen's matrix mult: $T(n)=7T(n/2)+\Theta(n^2)$. 7 rec calls each round, combining solutions cost=$\Theta(n^2)$, case 1 Master theorem
Int multiplication: d&c **not** always more efficient (faster) than brute force.
Karatsuba: **not** asymptotically optimal

Amortized analysis:
Cost of ith insert = ci: i if i power of 2, 1 otherwise
Cost of n inserts = $\sum_i c_i \leq n+\sum^{\log_2 n} 2^j \leq n+2n \leq 3n = O(n)$

1 multipush(k)=$\Theta(k)$. n multipush(k)=$\Theta(nk)$. Amort. time=$\Theta(nk)/n=\Theta(k)$

Randomized algorithm:
Karger's contraction:
Proof: return min cut w prob≥$2/n^2$.
k=|# edges in min cut|.
∀verts deg≥k & sum deg = 2|E| ⇒ 2|E|≥kn ⇒ |E|≥kn/2
Let E_i event where alg doesn't contract edge from min cut.
$P(E_1)=1-k/|E|\geq 1-2/n$; $P(E_2|E_1)\geq 1-2/(n-1)$;
$P(E_i|E_1\cap E_2\cap\ldots)\geq 1-2/(n-i+1)$
$P(\text{success})=P(E_1)P(E_2|E_1)\ldots P(E_{n-2}|E_1\cap\ldots\cap E_{n-3})=2(n-2)!/n!=2/(n(n-1))\geq 2/n^2$■
Early iterations lower failure rate than later.
Proof: repeat $n^2\ln n$ times ⇒ p(failure)≤$1/n^2$.
$(1-2/n^2)^{n^2\ln n}=[(1-2/n^2)^{1/2\,n^2}]^{2\ln n}\leq(e^{-1})^{2\ln n}\leq 1/n^2$■

Maximum 3-satisfiability:
Proof: k clauses 3-sat formula, expected # clauses satisfied by random assignment = 7k/8.
$Z_j=1$ if clause C_j is satisfied; 0 otherwise
$E[Z]=\sum E[Z_j]=\sum P(\text{clause } C_j \text{ satisfied})=7/8\,k$■
Collorary: ∀3-sat instance, ∃truth ass satisfying≥7/8 of all clauses.
$7k/8=E[Z]=\sum jp_j=\sum_{j<7k/8}jp_j+\sum_{j\geq 7k/8}=(7k/8-1/8)\sum p_j+k\sum p_j=(7k/8-1/8)(1)+kp \Rightarrow p=1/(8k)$■

Johnson's algorithm: Repeatedly gen rand assignments until one satisfies ≥7k/8 clauses.
Is a 7/8 approx alg. By prev lemma, each iteration succeeds w prob ≥1/(8k). By waiting time bound, expected # trials to find satisfying ass ≤8k■
Monte Carlo: guaranteed to run in poly time, likely to find right answ. E.g. contraction alg for global min cut. Success prob ↗ as # iterations ↗.
Las Vegas: guar. right answ, likely poly time. E.g. randomized quicksort, Johnson's max 3-sat alg.

Probabilistic analysis:
Randomized quicksort: Proof of complexity.
Items $z_1,\ldots,z_n$. $Z_{ij}=\{z_i,z_{i+1},\ldots,z_j\}$. $X_{ij}=1$ if z_i is compared to z_j; 0 otherwise (indicator RV). Then, $X=\sum_{i=1 \text{ to } n-1}\sum_{j=i+1 \text{ to } n}X_{ij}$.
$E[X]=\sum_{i=1 \text{ to } n-1}\sum_{j=i+1 \text{ to } n}E[X_{ij}]=\sum\sum 2/(j-i+1)=\sum\sum_{k=1 \text{ to } n-i}2/(k+1)<\sum\sum_{k=1 \text{ to } n}2/k<O(n\log n)$■

Complexities:
MaxHeapify: O(lg n)
Heapsort: O(nlg n)
Red black trees, AVL: O(lg n)
Union by size & path compression: O(lg n)
Huffman encoding: O(n+d lg d), n size of word, d # distinct chars in word
DFS, top sort: Θ(V+E)
SSSP DAG: O(V+E)
Dijkstra's bin heap: O(E logV)
Dijkstra's fib heap: O(V logV + E)
Gale-Shapley: $O(n^2)$
Ford-Fulkerson: O(CE), C sum cap of edges outgoing from s
Bellman-Ford: O(VE)
Knapsack: O(nW), W max weight of knapsack
Merge sort: $T(n)=2T(n/2)+n=\Theta(n\log n)$
Binary search: $T(n)=T(n/2)+1$
Karatsuba: $T(n)=3T(n/2)+n=3T(n/2)+O(n)=\Theta(n^{\log 3})=\Theta(n^{1.585})$
Quick sort worst case: $T(n)=T(n-1)+T(0)+\Theta(n)=\sum_{k \text{ to } n}\Theta(k)=\Theta(\sum k)=\Theta(n^2)$■
Karger's contraction: $\Theta(n^2\log n)$ iterations, $\Omega(|E|)$ time ⇒ $O(n^2|E|\log n)$ complexity. When $n/\sqrt{2}$ verts left, 50% of failure. Run contraction **once** until $n/\sqrt{2}$ remain, **twice** on resulting graph, pick **best of 2 cuts:** $O(n^2\log^3 n)$
Best known improvement: $O(|E|\log^3 n)$
Randomized quicksort: O(nlog n)