

# 1 Asymptotic analysis

- Notations

- $O(f(x))$ : upper bound.
- $\Omega(f(x))$ : lower bound.
- $\Theta(f(x))$ : both upper and lower bound.
- Not every  $g(x)$  has  $\Theta(f(x))$ .

- \*  $g(x) = (1 + \sin x)x^3 + x^2$

- Master theorem

- $T(n) = aT(n/b) + f(n)$ ,  $a \geq 1, b > 1$
- Case 1:  $f(n) = O(n^c)$ ,  $c < \log_b a$ .  
Then  $T(n) = \Theta(n^{\log_b a})$ .
  - \* Example:  $T(n) = 8T(n/2) + 10n^2$ .  
 $a = 8, b = 2, c = 2, \log_b a = 3 > c$ .  
Thus  $T(n) = \Theta(n^3)$
- Case 2:  $f(n) = \Theta(n^c \log^k n)$ ,  $c = \log_b a$ .  
Then  $T(n) = \Theta(n^c \log^{k+1} n)$ .
  - \* Example:  $T(n) = 2T(n/2) + 10n$ .  
 $a = 2, b = 2, c = 1, k = 0, \log_b a = 1 = c$ .  
Thus  $T(n) = \Theta(n \log n)$
- Case 3:  $f(n) = \Omega(n^c)$ ,  $c > \log_b a$ .  
Then  $T(n) = \Theta(f(n))$ .
  - \* Example:  $T(n) = 2T(n/2) + n^2$ .  
 $a = 2, b = 2, c = 2, \log_b a = 1 < c$ .  
Thus  $T(n) = \Theta(n^2)$

# 2 Divide and conquer

- Example: peak finding

- 1-d  $[O(\lg n)]$ :
  - \*  $a[m-1] > a[m]$ : search left.
  - \*  $a[m+1] > a[m]$ : search right.
- 2-d  $[O(n \lg n)]$ :
  - \* Given  $m$ th col, find max in  $i$ th row.
  - \*  $a[i, m-1] > a[i, m]$ : search left.
  - \*  $a[i, m+1] > a[i, m]$ : search right.

# 3 Heap

- Priority queue

- insert(S, x)
- max(S)
- extract\_max(S)
- increase\_key(S, x, k)

## 3.1 Max heap

- Parent is greater than children (and descendants).
- build\_max\_heap[ $O(n)$ ]: bottom up max\_heapify.  
Denote  $h$  as height.  $\frac{n}{4}$ th node has height 1.

$$T(n) = \sum_{h=1}^H \frac{n}{2^{h+1}} h = \frac{n}{2} \sum_{h=1}^H \frac{h}{2^h} = O(n)$$

$$\sum_{l=0}^{\infty} hx^h = \frac{x}{(1-x)^2} \text{ when } x < 1. \text{ Here } x = 1/2.$$

$$s = \sum_{h=0}^{\infty} hx^h$$
$$t = \int \frac{s}{x} dx = x \sum_h x^h = \frac{x}{1-x}$$
$$\frac{s}{x} = \frac{dt}{dx} = \frac{1}{(1-x)^2}$$

- max\_heapify[ $O(\log h)$ ]: sink down a node, provided this node is not max heap, but both children are max heap. This process makes the node a max heap.
- insert[ $O(\log H)$ ]: insert to the end and bubble up.
- pop[ $O(\log H)$ ]: replace root with last and heapify.

## 3.2 Median heap

- Maintain MaxHeap on the left and MinHeap on the right.
- Insert:
  - Insert into MaxHeap, if key  $<$  root of MaxHeap.
  - Insert into MinHeap, otherwise.
  - Balance if  $size_{max} - size_{min} > 1$  by pop-and-insert.

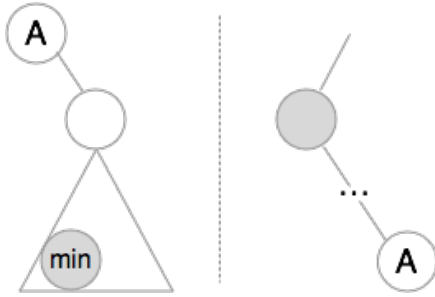
# 4 Tree

- Height of tree: length of longest path from root to leaf.
- Height of node: length of longest path from node to leaf.
- Depth of node: length from root to node.
- Augmented tree: augment nodes with some property (size, height, etc.).
- Balanced tree:  $h = O(\lg n)$ .

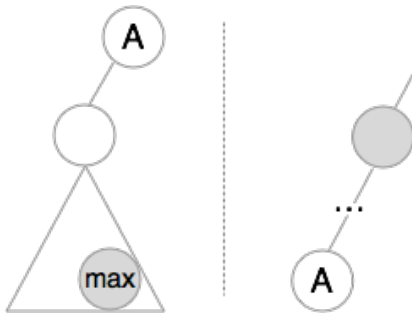
- AVL tree
- Red-black (2-3-4) tree
- Skip list
- Treap
- 2-3 tree
- B-tree

## 4.1 Binary search tree

- Example: lane reservation.
- Def: left is smaller, right is larger.
- search, min, max  $[O(h)]$
- succ  $[O(h)]$ 
  - Case 1: has right child.
  - Case 2: doesn't have right child.



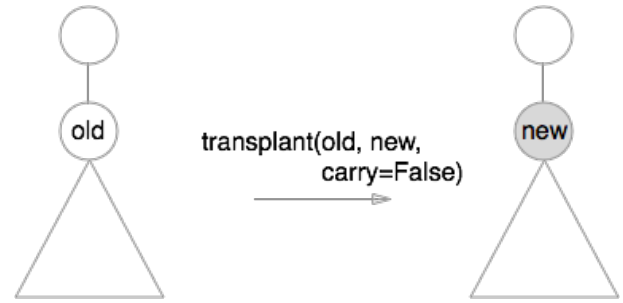
- pred  $[O(h)]$



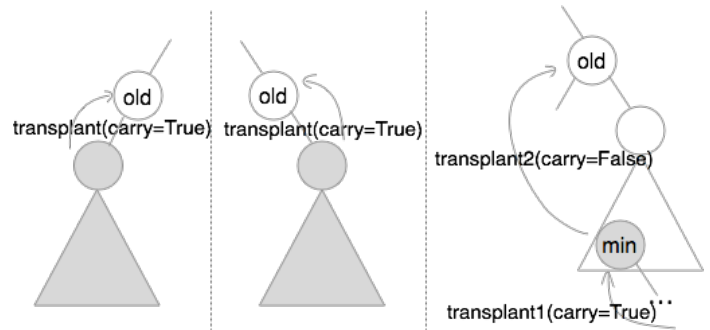
- insert
  - Probe parent.
  - Insert node as leaf.
- delete
  - Transplant
    - \* Carry



\* Non-carry



- Case 1: node with single left (or null).
- Case 2: node with single right (or null).
- Case 3: node with both children.
- Candidate strategy:
  - \* Case 1/2: transplant its only child.
  - \* Case 3, strategy 1: transplant min of right subtree. Min of right subtree falls into case 1 or 2, and needs one more transplant.
  - \* Case 3, strategy 2: transplant max of left subtree. Max of left subtree falls into case 1 or 2, and needs one more transplant.



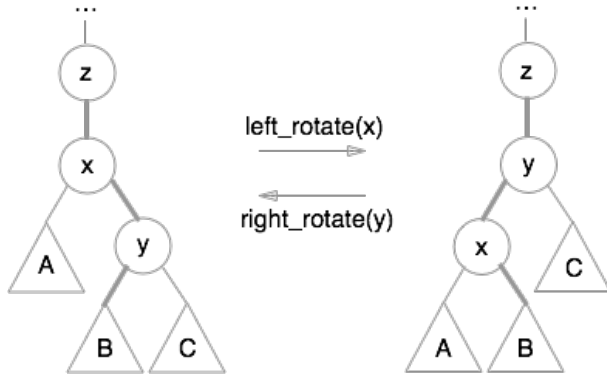
## 4.2 AVL tree

- Def:  $|h_{left} - h_{right}| \leq 1$ , ( $h_{null} = -1$ ).
- AVL tree is balanced tree. Proof: Denote  $N_h$  as min # of nodes to form an AVL tree, then

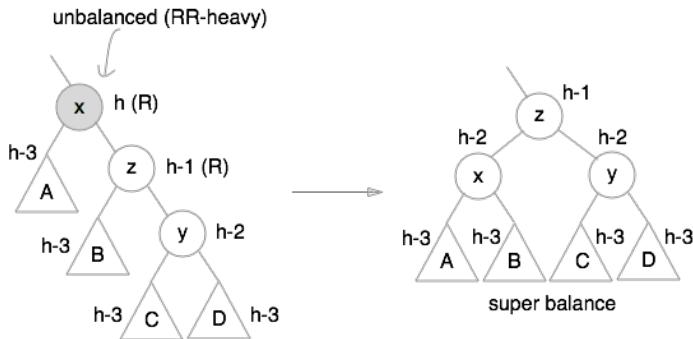
$$\begin{aligned} N_h &= 1 + N_{h-1} + N_{h-2} \\ &> 2N_{h-2} = \Theta(2^{h/2}) \\ h &= \Theta(\lg N_h) = O(\lg n) \end{aligned}$$

- insert
  - Intuitions:
    - \* left\_rotate on the right-heavy node.
    - \* right\_rotate on the left-heavy node.
    - \* Only ancestors of the inserted node can become unbalanced after insertion.
    - \* Ensure only ancestors need rotation, and keep subtrees untainted.
  - left\_rotate: rotate node from parent to left child.

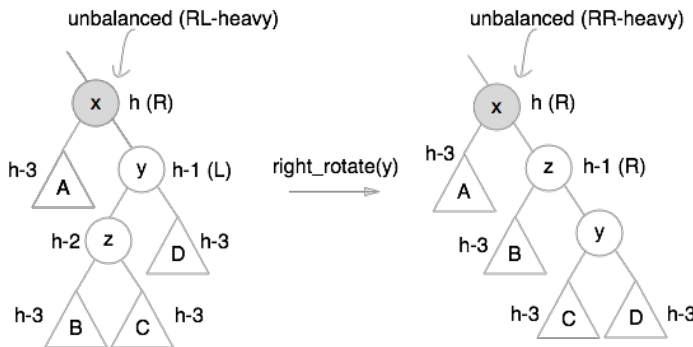
- right\_rotate: rotate node from parent to right child.



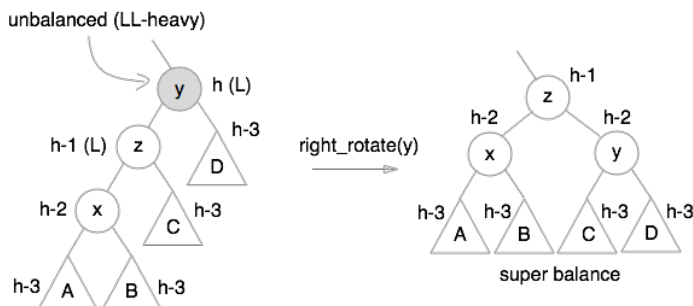
- Case 1: RR-heavy.
- Strategy 1: left rotate the second right-heavy node.



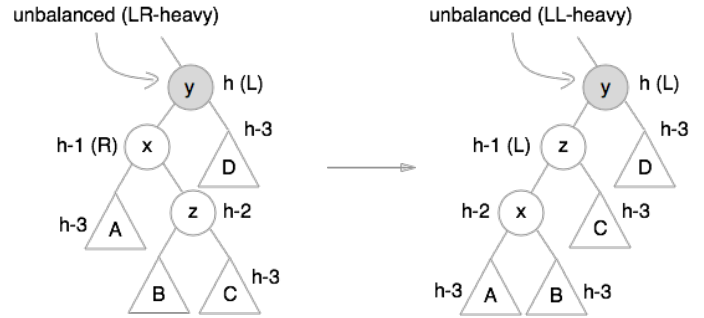
- Case 2: RL-heavy.
- Strategy 2: right rotate the left-heavy node and reduce to case 1. Brief: right\_rotate then left\_rotate.



- Case 3: LL-heavy.
- Strategy 3: right rotate the second left-heavy node.

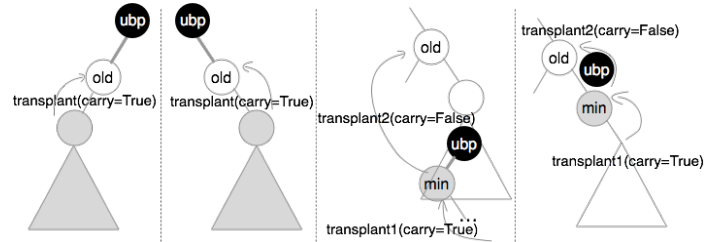


- Case 4: LR-heavy.
- Strategy 4: left rotate the right-heavy node and reduce to case 3. Brief: left\_rotate then right\_rotate.



#### • delete

- Delete as binary search tree.
- Unbalance point
  - \* Case 1: the deleted has single left (or nil).
  - \* Case 2: the deleted has single right (or nil).
  - \* Case 3: the deleted has both children, and the candidate (successor or predecessor depending on the strategy) is not son of it.
  - \* Case 4: the deleted has both children, and the candidate is son of it.



## 4.3 Red-black tree

#### • Properties

- All nodes are either red or black.
- End nodes are extended with nil leaves.
- Root and leaves (nil's) are black.
- Red node has black parent.
- All simple paths from node  $x$  to a leaf have same # of black nodes.
  - \* black\_height: # of black's to leaf, excluding start, including leaf.

#### • Claim: $h \leq 2 \log(n + 1)$ . Proof:

- Merge red's into black's (2-3-4 tree with height  $h'$ ).
- # leaves =  $n+1$
- $2^{h'} \leq \# \text{ leaves} \leq 4^{h'}$
- $h \leq 2h'$

#### • insert

- Insert as red node.

- Resolve the nodes down to the root.
- TODO: cases to switch color or rotate.
- Versus AVL tree
  - More cases to consider.
  - Asymptotically the same complexity.
  - Faster insert/delete (fewer rotations).
  - Slower lookup (greater height).

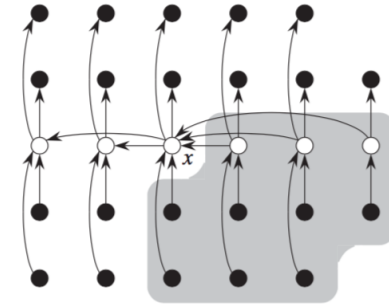
## 5 Sort

- Goal: output  $n$  sorted items.
- Comparison model
  - Decision tree: every comparison model can be represented as a decision tree.
  - $\# \text{ leaves} \geq \# \text{ possible outcomes} = n!$
  - $\text{height} \geq \log n! = n \log n - O(n)$  (Sterling).
  - Lower bound:  $\Omega(n \log n)$
- $O(n \log n)$  sorting
  - Quick sort: Pivot. Divide-and-conquer.
  - Merge sort: Divide-and-conquer.
  - Heap/BST.
- Linear-time (integer) sorting
  - Counting sort
    - \*  $L[i]$ : counter or list of items for integer  $i$ .
    - \*  $O(n + k)$ , where  $n$  is number of items,  $k$  is the largest integer.
  - Radix sort
    - \* Base  $b$  repr:  $(\overline{d_{D-1} \dots d_1 d_0})_b$ , where  $D = \log_b k$ .
    - \* Sort each digit by counting sort.
    - \*  $O((n + b) \log_b k)$
    - \*  $O(cn)$  if  $k = n^c$ ,  $b = \Theta(n)$ .

## 6 Order statistics

- Find  $k$ th smallest element.
- $\text{pivot}(p, q, k)$ : find  $k$ th element within  $[p, q]$  recursively.
  - Worst case:  $T(n) = T(n - 1) + \Theta(n) = \Theta(n^2)$
  - Expectation:  $E[T(n)] = \Theta(n)$ 
    - \*  $X_i$ : indicates if split into  $(i, n - i - 1)$
    - \*  $T_i = T(\max\{i, n - i - 1\}) + \Theta(n)$
    - \*  $T(n) = \sum X_i T_i$
    - \*  $E[T(n)] = \sum_{i=0}^{n-1} T_i / n \leq \frac{2}{n} \sum_{i=0}^{\lceil n/2 \rceil} T_i$
    - \*  $E[T(n)] \leq cn$  by mathematical induction.
- Median pivot
  - Recursively find median of median.

- Choose median of median as pivot.
- Partition like above.
- $T(n) \leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n)$
- $T(n) \leq \Theta(n)$  by induction.



## 7 Hash

- Prehash: map keys to non-negative integers.
- Hash: map the large key space to smaller space.
- Chaining: chain up the collided items.
- Simple uniform hashing
  - Assumption: a key is equally likely to be mapped into any slot. The hash value is  $r.v.$ . This assumption generally does NOT hold true, especially when hash function is determinantal.
  - Load factor  $\alpha = n/m$ : expected length of chain, where  $n$  is  $\#$  keys,  $m$  is  $\#$  slots.
  - Complexity:  $O(1 + \alpha)$ .
- Hash functions:
  - Mod:  $h(k) = k \% m$ , where  $m$  is prime.
  - Mul:  $h(k) = [(a \cdot k) \% 2^w] \gg (w - r)$ , where  $w$  is  $\#$  bits in  $k$  (key), and  $r$  is  $\#$  bits in  $m$  ( $\#$  slots).



- Universal:  $h_{a,b}(k) = [(ak + b) \% p] \% m$ , where  $k \in \mathcal{K}$ ,  $p > |\mathcal{K}|$  is picked as prime, and  $a, b$  are randomly picked within  $\{0, \dots, p - 1\}$ .
  - \*  $Pr\{h_1(k_1) = h_2(k_2)\} = 1/m$ , given  $k_1, k_2$ .

- Applications

- Dict: random hash + dynamic table
- String match: Rabin-Karp
  - \*  $h(s) = (\sum ord(s_i) \cdot base^i) \% m$
  - \*  $h(s_{(i+1):(i+l)}) = \{[h(s_{i:(i+l-1)}) - ord(s_i)] \cdot base + ord(s_{i+l})\} \% m$

- \*  $k$  insertions involve  $\log k$  doublings
- \* Doubling cost:  $\Theta(2^1 + 2^2 + \dots + 2^{\log k}) = \Theta(k)$
- \* Insertion cost:  $\Theta(k)$
- \* Amortized cost:  $\frac{1}{k}(\Theta(k) + \Theta(k)) = \Theta(1)$

## 7.1 Universal hashing

- Adversary model: given hash function is not random, adversary can always find a set of collided keys.
- Example:  $h_{a,b}(k) = [(ak + b) \% p] \% m$ . Each dict is instantiated with different  $a$  and  $b$ .
- Universal:  $|\{h \in \mathcal{H} : h(x) = h(y), x \neq y\}| = |\mathcal{H}|/m$ .
  - Collision:  $Pr\{h_1(x) = h_2(y)\} = 1/m$ , given  $x, y$ .
  - Theorem: given  $x$ ,  $E[\#collisions] < n/m = \alpha$ .  
Proof: given  $h$  is random, denote  $r.v.$   $C_{x,y} = \mathbb{1}\{h(x) = h(y)\}$ , and  $C_x = \sum_{y \in \mathcal{K} \setminus x} C_{x,y}$ , then
    - \*  $E[C_{x,y}] = 1/m$
    - \*  $E[C_x] = \sum_{y \in \mathcal{K} \setminus x} E[C_{x,y}] = \frac{n-1}{m}$
- Example  $h_a(k) = (\sum a_i k_i) \% m$ , where  $a = (\overline{a_r \dots a_1 a_0})_m$  and  $k = (\overline{k_r \dots k_1 k_0})_m$ . Then  $|\mathcal{H}| = |\mathcal{K}| = m^{r+1}$ .
  - Assume  $x, y$  differ in 0th bit, i.e.  $x_0 \neq y_0$ .
  - $\sum_{i=0}^r a_i(x_i - y_i) \equiv 0 \pmod{m}$
  - $a_0(x_0 - y_0) \equiv -\sum_{i=1}^r a_i(x_i - y_i) \pmod{m}$
  - A theory of Finite (Galois) Field:  $\forall z \neq 0 \in \mathbb{Z}_m, m \in P$ , then  $\exists z^{-1}$ , s.t.  $z z^{-1} \equiv 1 \pmod{m}$ .
  - $a_0 \equiv [-\sum_{i=1}^r a_i(x_i - y_i)](x_0 - y_0)^{-1} \pmod{m}$
  - # choices to collide at  $a_0$ :  $m^r$  ( $a_1 \dots a_r$  are free).

## 7.2 Perfect hashing

- Two-level hashing
  - Allow  $n_i$  collisions in  $i$ th slot at 1st level.
  - Nearly no collision at 2nd level if # slots  $m_i = n_i^2$ .
  - Worst complexity:  $O(1)$ .
  - Theorem: Hash  $n$  keys into  $m = n^2$  slots using random  $h$  in universal hashing would lead to  $E[\#collisions] < 1/2$ .
    - \*  $E[C_{x,y}] = \frac{1}{m} = \frac{1}{n^2}$
    - \*  $E[C] = \binom{n}{2} \frac{1}{m} = \frac{n-1}{2n} < 1/2$

## 8 Amortization

- Dynamic table
  - Grow/shrink the table when necessary.
  - Load factor  $\alpha = n/m$ ,  $n$  is # items,  $m$  is # slots.
  - Double the table when  $\alpha > \bar{\alpha}$
  - Shrink the table when  $\alpha < \underline{\alpha}$
  - Aggregate analysis on insertion

- Aggregate method
- Accounting method
  - Charge  $i$ th op amortized cost  $\hat{c}_i$ .
  - Overcharged are stored to bank.
  - undercharged are taken from bank.
  - Design charge scheme such that  $balance \geq 0$ .
- Potential method
  - Design  $\Phi_i$  bound with data structure  $D_i$ .
  - Requirements:
    - \*  $\Phi_0 = 0$
    - \*  $\Phi_i \geq 0, \forall i$
    - \*  $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} \geq 0$
  - Analyze  $\hat{c}_i$  for different cases according to  $\Phi_i$ .
  - Any  $\Phi_i$  satisfies requirements will do. Better  $\Phi_i$  yields tighter upper bound.
- Example:  $\bar{\alpha} = 1, \underline{\alpha} = 1/4$ 
  - Potential

$$\Phi_i = \begin{cases} 2n_i - m_i & , \alpha_i \geq 1/2 \\ n_i/2 - m_i & , \alpha_i < 1/2 \end{cases}$$

## 9 Numbers

### 9.1 Catalan numbers

- Def
  - $P$ : set of balanced parentheses.
  - $\perp \in P$  ( $\perp$  means empty).
  - If  $\alpha, \beta \in P$ , then  $(\alpha)\beta \in P$ .
- $C_n = |P_n|$  with  $n$  pairs of parentheses.
  - $C_0 = C_1 = 1$ .
  - $C_{n+1} = \sum_{k=0}^n C_k C_{n-k}$

### 9.2 High precision

- Newton's method
  - Tangent on  $x_i$ :  $y = f(x_i) + f'(x_i)(x - x_i)$
- Square root
  - $x = \sqrt{a} \Rightarrow f(x) = x^2 - a = 0$
  - $x_{i+1} = x_i - f(x_i)/f'(x_i) = (x_i + a/x_i)/2$
  - Quadratic convergence: #  $\checkmark$  digits  $\propto$  (# iters)<sup>2</sup>
- Multiplication (Karatsuba)

- $z = x y$
- $x = x_1 r^{n/2} + x_0$
- $z = z_0 + z_2 + z_1$ 
  - \*  $z_0 = x_0 y_0$
  - \*  $z_2 = x_2 y_2$
  - \*  $z_1 = (x_0 + x_1)(y_0 + y_1) - z_0 - z_2$
- $T(n) = 3T(n/2) + \Theta(n) = \Theta(n^{\log 3}) \approx \Theta(n^{1.58})$

- Advanced multiplication

- gmpy2
- Schönhage-Strassen: FFT,  $\Theta(n \lg n \lg \lg n)$ .
- Furer:  $\Theta(n \log n 2^{O(\log^* n)})$

- Division

- $x = R/b \Rightarrow f(x) = 1/x - b/R = 0$
- $x_{i+1} = 2x_i - bx_i^2/R$
- $R = (\overline{r_{d-1}..r_1 r_0})_2$ , power of 2 is easy to divide.
- $T(n) = O(\lg n n^\alpha)$ , break down to mul's.

- When precision digits grow large, complexity of mul, div, sqrt becomes nearly the same as  $O(n^\alpha)$ .

## 10 Graph

- Edge

- Tree edge:  $old \rightarrow new$
- Forward edge: given path  $x_0 \rightarrow \dots \rightarrow x_n$ , any edge not in the path that points to descendant.
- Backward edge:  $\dots$ , to ascendant.
- Cross edge: Between subtrees.
- Undirected G can have tree edges and back edges.

- Search

- Breadth first search (BFS)
- Depth first search (DFS)

- Min span tree (DFS/BFS): search and save edges.

- Cyclic detection (DFS): find back edges.

- Back edge: Edge( $v_2 \rightarrow v_1$ ),  $v_1$  is ascendant.
- $v_1$  is ascendant of  $v_2$  if  $(t_1^{in}, t_1^{out}) \supset (t_2^{in}, t_2^{out})$ .
- Practically we also need to check if  $v_1$  and  $v_2$  have the same root. It is possible that we first DFS  $v_1$ , and then  $v_2$ .

- Topological sort (DFS):

- Append the out nodes.
- Reverse the order.

- Dependency management (DFS):

- Reverse edges and run topological sort.

## 10.1 Dijkstra

- Single source shortest path

- Do NOT allow negative weights

- Outputs

- $d[v]$ : shortest distance from source to  $v$ .
- $p[v]$ : parent of  $v$  in the shortest path.

- Dynamic programming

- $S$  is safe set, within which have been reached by shortest path from source.
- $R$  is the remaining set.

- Optimize

- Maintain  $d$  as priority queue, so that each time *EXTRACT-MIN* only takes  $O(\log |V|)$
- Total complexity:  $O(|E| + |V| \log |V|)$

- Single source single destination

- $S_f$  forward safe set.
- $S_b$  backward safe set.
- Stop when two sets intersect.

## 10.2 Bellman-Ford

- Single source shortest path that allows negative weights.

- Mark  $d[v] = -\text{inf}$  to indicate negative cycle.

- Algorithm

- Relax ALL edges for  $|V| - 1$  passes.
- Relax one more time to check negative cycle.

- Complexity  $O(|V||E|)$ .

## 11 Misc