6.006 Cheat Sheet (shreyask)

Hash Tables

Pre-Hashing, whatever key we have, we convert to non-negative integer by just taking the binary representation of that object → integer.
Chaining if collision, store as a list. Worst case $O(n)$, any hashing. But randomized?
SUHA: each key is equally likely to be hashed to any slot of the table, independent of each other.
Proof of Constant Time: expected length of chain $n/m = \alpha$ load factor. n is keys, M slots.

$$collisions = \frac{N(N-1)}{2}\frac{1}{M}$$
$$P(collision) = 1/m$$
$$P(query\ correct) = (1-1/m)^{n-1}$$

It takes us $O(n+m+m')$ to grow table because we need to rehash. If we double the table when we hit the load factor, our insert time is **Amortized** $O(1)$. For table doubling to work our doubling factor can at least be 2 and at most be 3, anything above that isn't amortized constant anymore.

BFS

```
BFS (V,Adj,s):                          See CLRS for queue-based implementation
    level = { s: 0 }
    parent = {s : None }
    i = 1
    frontier = [s]                       # previous level, i − 1
    while frontier:
        next = [ ]                       # next level, i
        for u in frontier:
            for v in Adj [u]:
                if v not in level:       # not yet seen
                    level[v] = i         ♯ = level[u] + 1
                    parent[v] = u
                    next.append(v)
        frontier = next
        i + =1
```

DFS

```
visited = {}

def do_something(node):
    print node

def dfs_visit(node):
    visited[node] = True
    for child in graph[node]:
        if not child in visited:
            dfs_visit(child)
    do_something(node)

for node in graph.keys():
    if not node in visited:
        dfs_visit(node)
```

The output reversed is also toposorted.

SSSP

Graph Structure	Best Known SSSP Algorithm	Complexity
Unweighted	BFS	$O(E+V)$
General	Bellman-Ford	$O(EV)$
Nonnegative weights	Dijkstra + Fibonacci heap	$O(E + V\log V)$
DAG	DFS + Topological sort + BFS	$O(E+V)$

APSP

Graph Structure	Best Known APSP Algorithm	Complexity
Unweighted	$\lvert V\rvert \times$ BFS	$O(VE)$
General	$\lvert V\rvert \times$ Bellman-Ford*	$O(V^2E)$
General (dense)	Floyd-Warshall	$O(V^3)$
General (sparse)	Johnson's	$O(VE + V^2\log V)$
Nonnegative weights	$\lvert V\rvert \times$ Dijkstra	$O(VE + V^2\log V)$

*Not the best-known algorithm

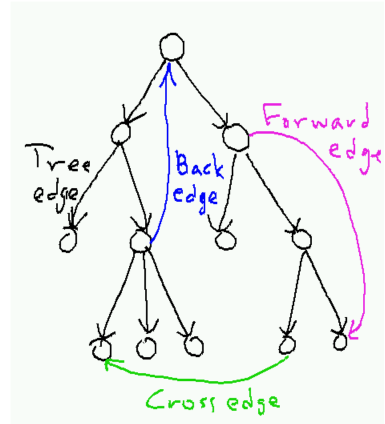
Dijkstra

RELAX(u, v, w)
  if $d[v] > d[u] + w(u,v)$
    then $d[v] \leftarrow d[u] + w(u,v)$
      $\Pi[v] \leftarrow u$

Lemma: The relaxation algorithm maintains the invariant that $d[v] \geq \delta(s,v)$ for all $v \in V$.
Proof: By induction on the number of steps.
Consider $RELAX(u,v,w)$. By induction $d[u] \geq \delta(s,u)$. By the triangle inequality, $\delta(s,v) \leq \delta(s,u) + \delta(u,v)$. This means that $\delta(s,v) \leq d[u] + w(u,v)$, since $d[u] \geq \delta(s,u)$ and $w(u,v) \geq \delta(u,v)$. So setting $d[v] = d[u] + w(u,v)$ is safe. □

Dijkstra (G, W, s) //uses priority queue Q
  Initialize (G, s)
  $S \leftarrow \phi$
  $Q \leftarrow V[G]$ //Insert into Q
  while $Q \neq \phi$
    do $u \leftarrow$ EXTRACT-MIN(Q) //deletes u from Q
    $S = S \cup \{u\}$
    for each vertex $v \in$ Adj$[u]$
      do RELAX (u, v, w) ← this is an implicit DECREASE_KEY operation

Types of Edges



Proof That Difference of 1 is balanced

N_h is the minimum number of nodes that's possible of height h. Since the two sub trees differ by height 1,

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

Open Addressing

Linear Probing

Let's say you have a table with a cluster. If $h(k,i)$ maps to this cluster. At the end, you just increase your cluster length by 1. If $0.01 < \alpha = n/m < 0.99$ cluster are of size $\Theta(\ln n)$. Dict is not constant time any more.

Double Hashing

$h(k,i) = (h_i(k) + ih_2(k)) mod m$ if $h_2(k)$ is relatively prime $\implies$ permutation. Number of expected probes on operation insert $\leq \frac{1}{1-\alpha}$