Quiz Review 2!!

or how I learned to stop worrying and love 6006LE

Outline

Every topic has a small review and some practice problems

- BFS
- DFS
- SSSP (relaxations, Dijkstra, Bellman-Ford)
- APSP (Johnson’s, Floyd-Warshall)
- Memoization
- Hashing

BFS and DFS

BFS

Bangkok Flight Services

DFS Deutsche Flugsicherung

(German air traffic control)

BFS and DFS

Breadth-first search and depth-first search

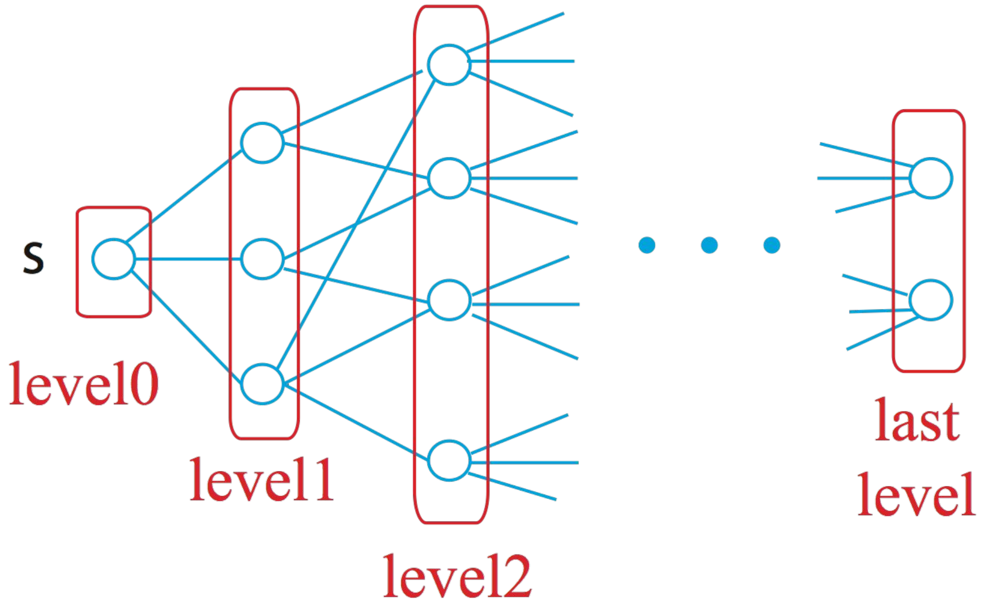
- Two basic ways of traversing a graph
- Both run in O(V + E) time

Why are they important?

- Give basic connectivity information
  - Is the graph connected?
  - Does the graph have a cycle?
  - ...
- Useful primitives / subroutines in more complicated algorithms

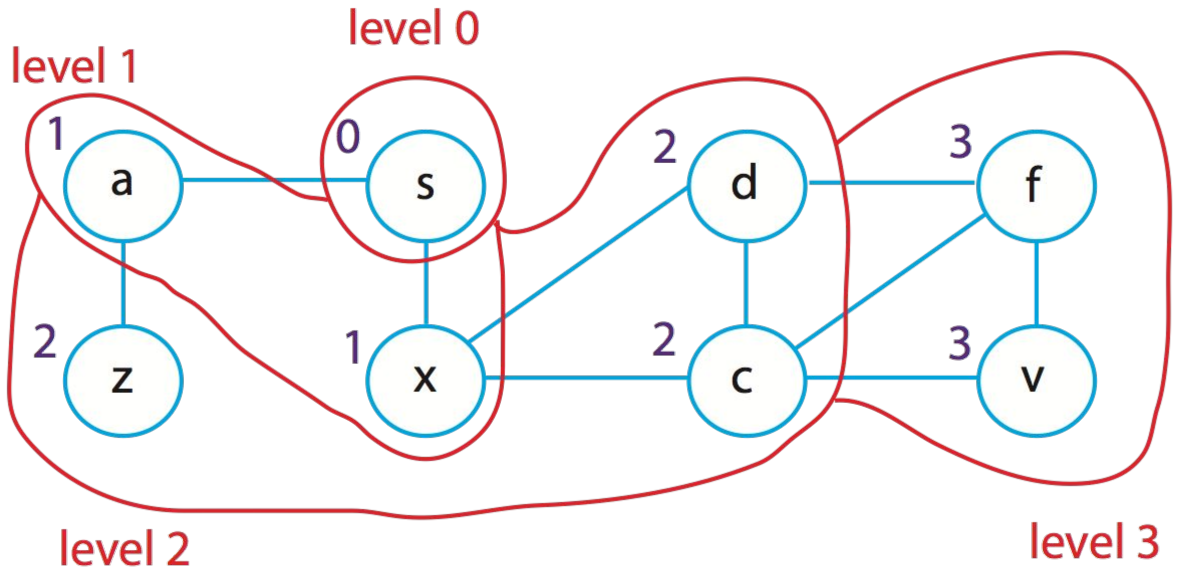
BFS

Main idea: traverse graph **layer-by-layer.**



BFS

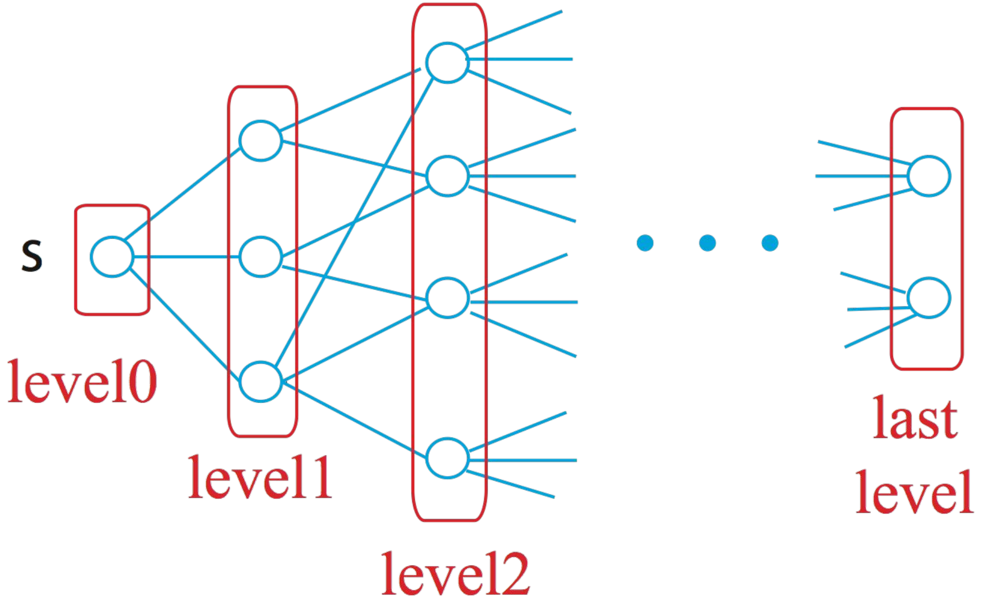
Layers are not always as clear, but BFS finds them.



$\text{frontier}_0 = \{s\}$
$\text{frontier}_1 = \{a, x\}$
$\text{frontier}_2 = \{z, d, c\}$
$\text{frontier}_3 = \{f, v\}$
(not x, c, d)

BFS application: shortest path

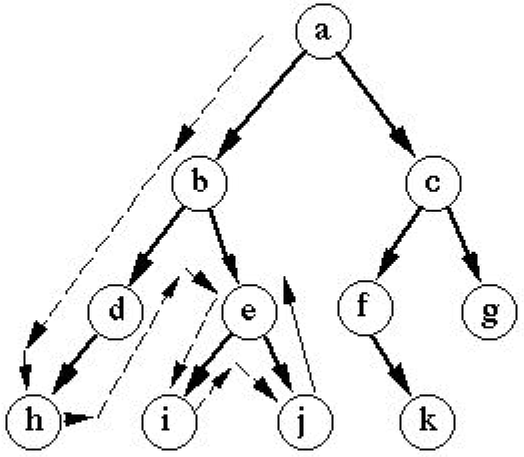
BFS computes the single-source shortest paths in an **unweighted** graph.



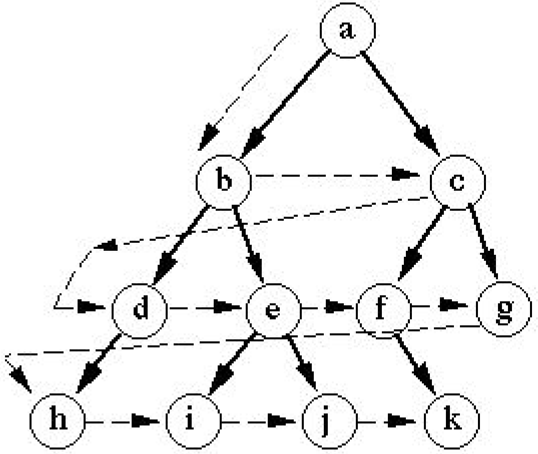
Level i contains the nodes at distance i from s.

DFS

Main idea: go as **deep** into the graph as possible.



Depth-first search



Breadth-first search

DFS

Remember: DFS is recursion + **outer loop**.

```
DFS-visit (Adj, s):
    for v in Adj [s]:
        if v not in parent:
            parent [v] = s
            DFS-visit (Adj, v)

DFS (V, Adj)
    parent = { }
    for s in V:
        if s not in parent:
            parent [s] = None
            DFS-visit (Adj, s)
```

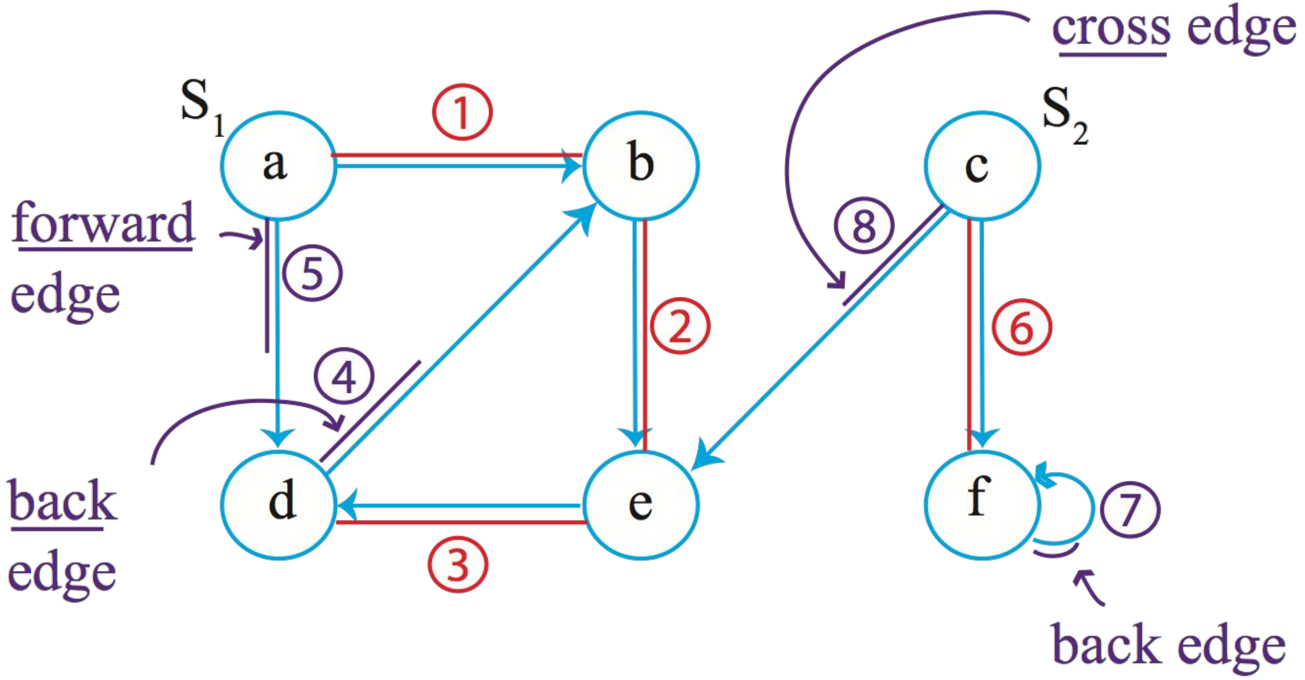
search from start vertex s (only see stuff reachable from s)

explore entire graph

(could do same to extend BFS)

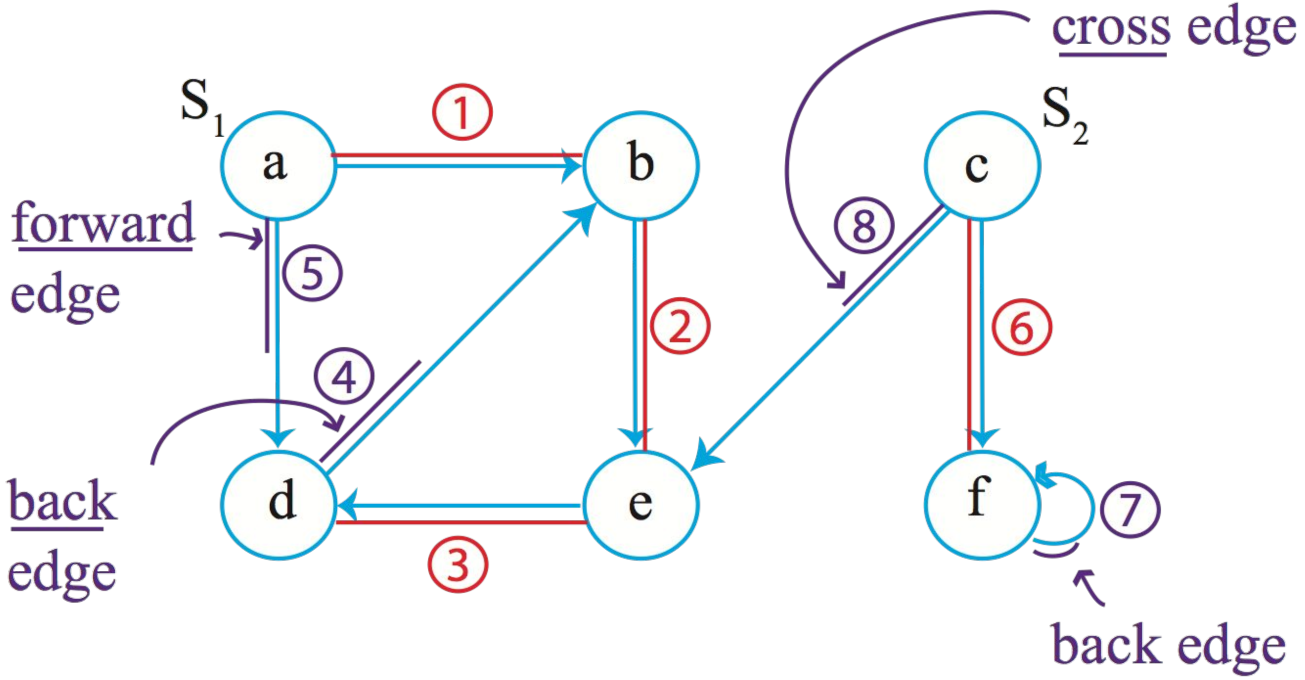
DFS edge classification

Four types of edges in a **directed** graph: forward, back, cross, and tree edge.



DFS edge classification

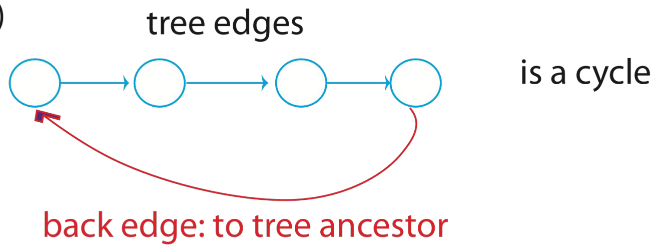
In an **undirected** graph: only tree and back edges.



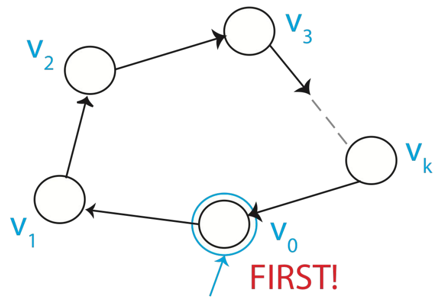
DFS application: cycle finding

A directed graph has a cycle if and only if DFS finds a back edge.

(<=)

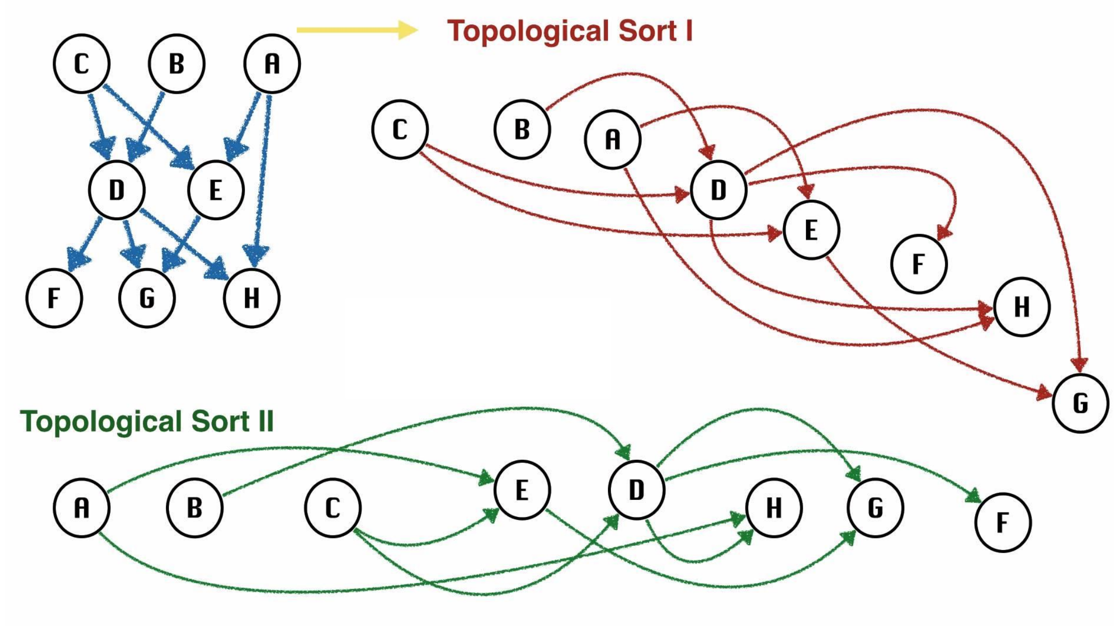


(=>) consider first visit to cycle:



DFS application: topological sort

Brings a DAG (directed acyclic graph) into a linear order.



Can do this by sorting according to DFS **finishing times**.

DFS start and finishing times

Record when we first visit a node and when we have visited all children.

```
DFS-visit (Adj, s):
start v →   for v in Adj [s]:
                if v not in parent:
                    parent [v] = s
finish v →          DFS-visit (Adj, v)

DFS (V, Adj)
    parent = { }
    for s in V:
        if s not in parent:
            parent [s] = None
            DFS-visit (Adj, s)
```

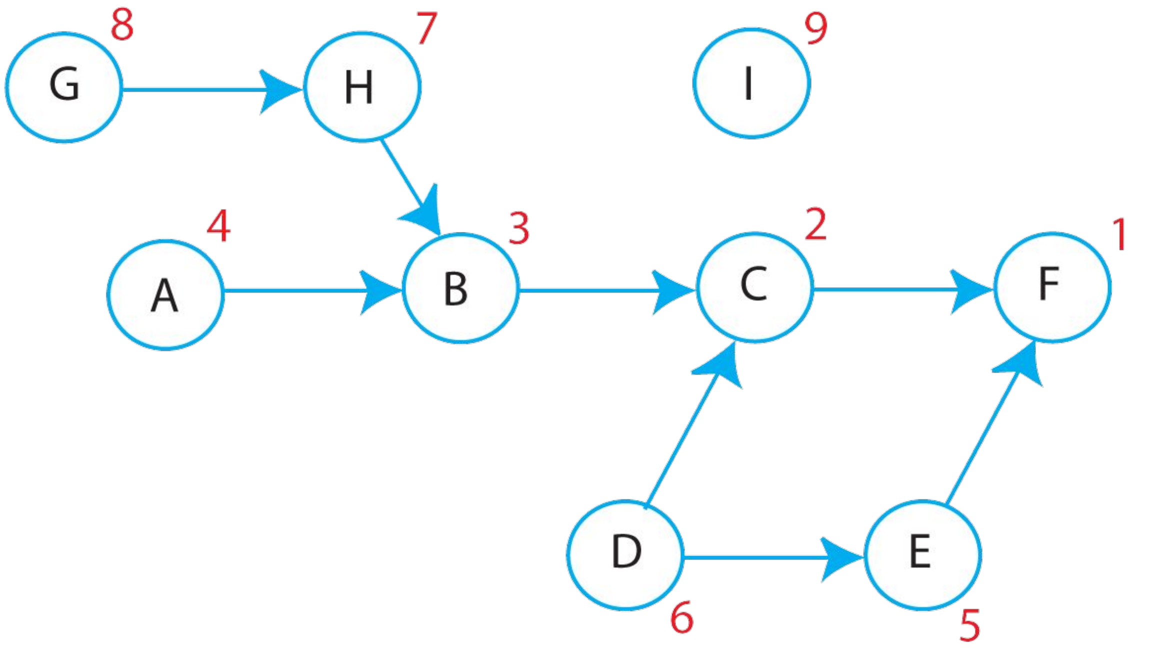
search from start vertex s (only see stuff reachable from s)

explore entire graph

(could do same to extend BFS)

DFS finishing times

Example where the DFS outer loop visits A, D, G, I.



Practice problems

True or False (6.006 Quiz 2 Spring 2015)

Suppose that we perform BFS on a directed graph G = (V, E) (starting from an arbitrary vertex). For each directed edge (u, v) in E, the two incident vertices u and v will lie on either the same level or two consecutive levels defined by the algorithm.

True or False (6.006 Quiz 2 Spring 2015)

If we run BFS and DFS starting from a source vertex s on a connected graph G = (V, E), then the BFS tree will have the same number of edges as the DFS tree.

True or False (6.006 Quiz 2 Spring 2014)

Let G be an undirected graph. If u is the root of some depth first search (DFS) tree and u's removal from G makes G a disconnected graph, then u has more than one child in the DFS tree.

Find the Root (6.006 Final Spring 2013)

A root in a directed graph is a vertex u such that there is a directed path starting at u to every other vertex v in the graph. Note that a graph may have more than one root vertex.

You are given a directed graph G with vertices V and edges E. G is an arbitrary directed graph which contains at least one cycle. Design and analyze an algorithm that does the following: If G has a root vertex u, the algorithm returns such a vertex. Otherwise, the algorithm returns FALSE.

For full credit, your algorithm should run in O(V + E) time.

Reachability warm-up (folklore)

Let G be an undirected graph with n vertices. We say that vertex t is reachable from vertex s if there is a path from s to t in G.

Propose a data structure that determines reachability between any pair of vertices. That is, given a query IS-REACHABLE(s, t), the data structure should answer TRUE if and only if there is a path from s to t in G. For full credit your data structure should be constructed in O(V + E) time, and it should answer each query in constant time.

Hint: Use DFS or BFS in the data structure construction phase.

Reachability on a Tree (6.006 Quiz 2 Fall 2015)

Let T be a rooted tree with n vertices. Note that T is a simple connected directed graph: each node has an outgoing edge to each of its children, and the root r is the only node without a parent. You may assume that we have a pointer to r.

Propose a data structure that determines reachability between any pair of vertices. That is, given a query IS-REACHABLE(s, t), the data structure should answer TRUE if and only if there is a directed path from s to t on this tree. For full credit your data structure should be constructed in O(V) time, and it should answer each query in constant time.

Hint: Use DFS in the data structure construction phase.

Crucial edges (6.006 Quiz 2 Fall 2014)

Given an undirected and connected graph G = (V, E), a crucial edge in the graph is an edge whose removal would break the graph into two pieces. In other words, an edge e ∈ E is crucial if G' = (V, E − {e}) is a disconnected graph.

(a) Show that an edge (u, v) is crucial only if it is a tree edge in every DFS tree.

(b) Design an algorithm that takes a graph G and returns some crucial edge in the graph, or returns None if no such edge exists. To receive full credit, your algorithm should run in time O(V + E).

Single-Source Shortest Path algorithms: framework

Keep track of shortest known distance d(v) from source s to each vertex v.

Basic operation: *Relax* an edge (u,v): d(v) = min(d(v), d(u) + w(u,v)).

If every shortest path has finite length (equivalently, if there are no reachable negative cycles), a finite (exponential) number of relaxations suffice.

Single-Source Shortest Paths: algorithms

Algorithm	Time	Cycles	Weights?	Summary
Brute force	O(n!)	Negative: Fail	Any	Relax every path.
BFS	O(V+E)		Unweighted	Visit closest vertices first.
Topo. Sort+1B-F	O(V+E)	DAG only	Any	Topological order of relaxation relaxes every path at once.
Bellman-Ford	O(VE)	Detect(+Mark)	Any	Relax every edge, V times.
Dijkstra’s	O(E+V log V)		Nonnegative	Grow sphere around source.

All-Pairs Shortest Paths

Algorithm	Time	Cycles	Weights?	Summary
Floyd-Warshall	O(V^3)	Negative: Detect	Any	Relax for every intermediate v.
Johnson's	O(V^2 log V+EV)	Negative: Detect(+Mark)	Any	Convert no negative cycles to no negative weights, then Dijkstra's from every source.

Multi-source shortest paths

How do you compute shortest paths when the source is not a single vertex s, but instead a set S of vertices?

More precisely, suppose you are given a weighted directed graph G=(V,E,w) with non-negative edge weights, you are given a source set S of vertices, and you are given a single target node t.

Your goal is to find the weight of the shortest path from *some* vertex s in S to t. Describe an O(V lg V + E) algorithm for computing this *multi-source shortest path distance*.

Faster Bellman-Ford

Is there a relaxation ordering of edges in G such that Bellman-Ford will find the minimum-weight path from X to every other vertex after only *one* iteration of Bellman-Ford? If so, provide justification. If not, provide a counterexample graph G and show why no such relaxation ordering can exist.

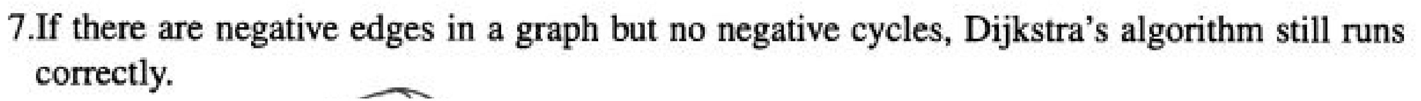
Fall 2013 quiz 2

True or false: Dijkstra's Algorithm can be implemented using an AVL tree instead of a heap for the Priority Queue and run in O((V+E) lg V) time.

Fall 2013 Quiz 2

True or false: Dijkstra's Algorithm can be implemented using a sorted array instead of a heap for the Priority Queue and run in $O((V+E) \lg V)$ time.

Spring 2011 Final



7.If there are negative edges in a graph but no negative cycles, Dijkstra's algorithm still runs correctly.

Spring 2011 Final

Problem 6. Does this path make me look fat? [10 points]

Consider a connected weighted directed graph $G = (V, E, w)$. Define the ***fatness*** of a path P to be the maximum weight of any edge in P. Give an efficient algorithm that, given such a graph and two vertices $u, v \in V$, finds the minimum possible fatness of a path from u to v in G.

What is hashing?

This is hashing!

Even better hashing...

ADT with `insert`, `delete`, and `search`

Motivation: *avoid maintaining structure of AVL/heaps -- see complexity gains!*

Hashing

ADT with `insert`, `delete`, and `search`

Motivation: *avoid maintaining structure of AVL/heaps -- see complexity gains!*

prehash(“ ”)

prehash(“

Katie Atkinson
@Katie_Atk

Following

I hope second semester isn't just one big metaphorical game of flappy bird.

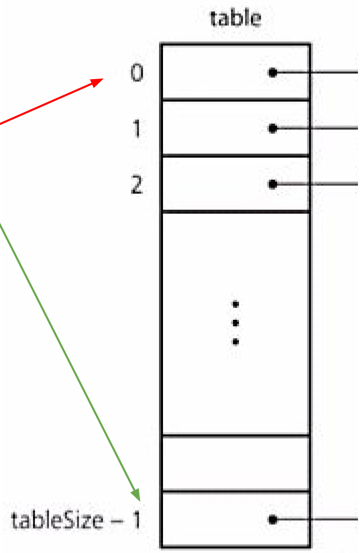
Reply Retweet Favorite More

RETWEETS 5 FAVORITES 4

6:46 PM - 4 Feb 2014

”)

```
hash function
U → 0,1,2,...tableSize
```



Collisions

But `U >> tableSize` in practice → collisions

1. Resolution by chaining!

```
hash function
U → 0,1,2,...tableSize
```

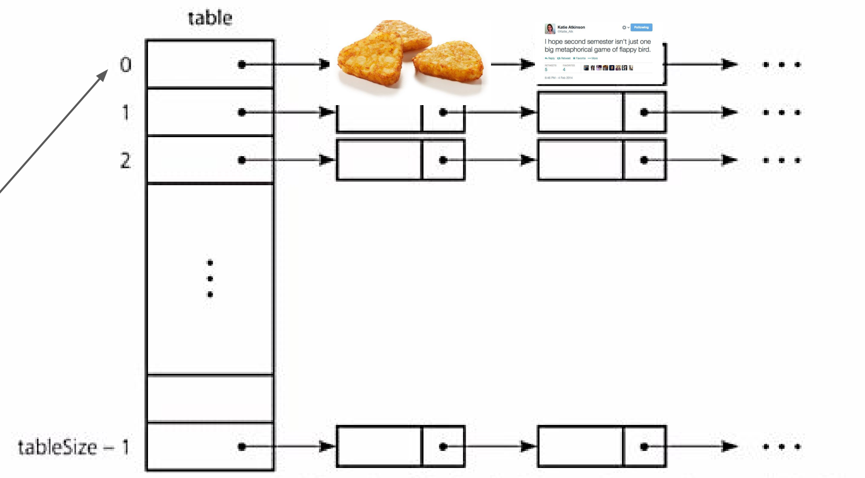


table
0
1
2
tableSize – 1
Katie Atkinson
I hope second semester isn't just one big metaphorical game of flappy bird.

Each location of the hash table contains a pointer to a linked list

Figure 12.49 Separate chaining

exp. complexity of ops = E[len of chain]

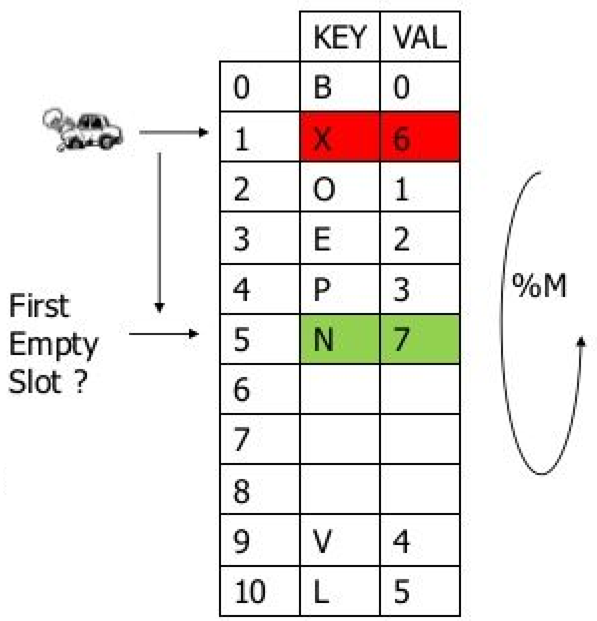
= O(1 + # items / table size)

Collisions

2. Resolution by linear probing

```
    trying to insert (N,7)
```

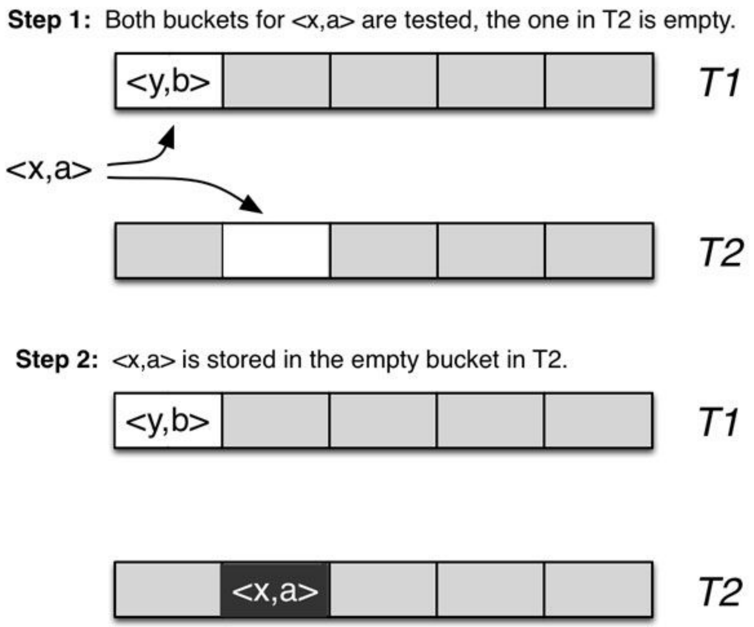
- `h(x,i) = x + i mod m`
- `Cache perf. Gain!`
- `be careful with DELETE!`



	KEY	VAL
0	B	0
1	X	6
2	O	1
3	E	2
4	P	3
5	N	7
6		
7		
8		
9	V	4
10	L	5

Collisions

3. Resolutions in cuckoo hashing


Step 1: Both buckets for <x,a> are tested, the one in T2 is empty.
<y,b>
T1
<x,a>
T2
Step 2: <x,a> is stored in the empty bucket in T2.
<y,b>
T1
<x,a>
T2

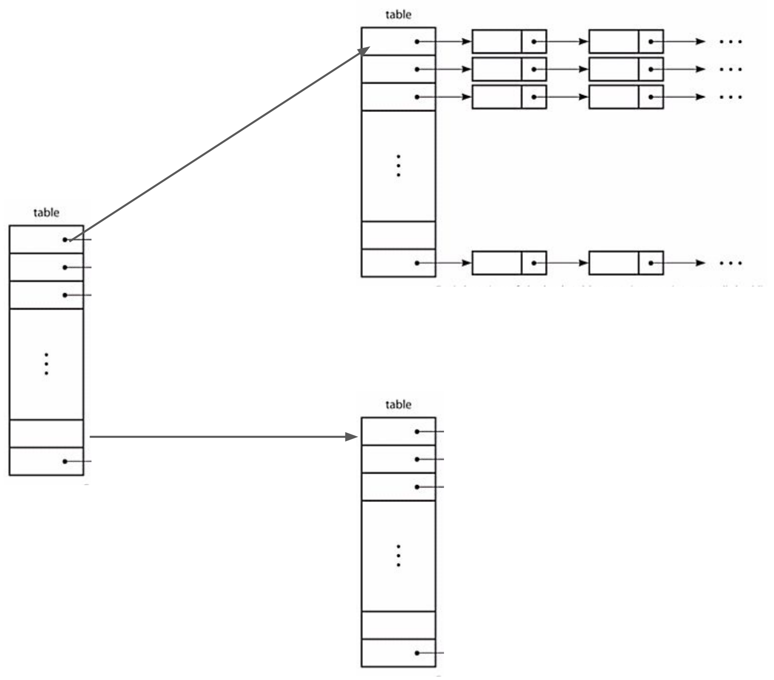
2 tables, 1 HF each!

Resolve cyclic insertions by choosing new table size and hash functions

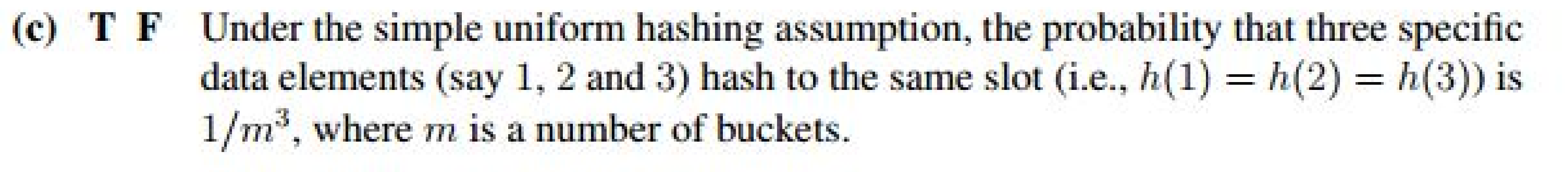
Collisions

3. Resolutions in Perfect Hashing

- double nested
- k collisions at slot
  - size k^2 inner table
  - birthday bound ->
  - Low collision prob.



Spring 2011 Final



(c) **T F** Under the simple uniform hashing assumption, the probability that three specific data elements (say 1, 2 and 3) hash to the same slot (i.e., $h(1) = h(2) = h(3)$) is $1/m^3$, where m is a number of buckets.

(non-open addressing scheme)

2. Under the uniform hashing assumption, if we use a hash table of size m with open addressing to hash 3 keys, the probability that the third inserted key needs exactly three probes before being inserted into the table is exactly $\frac{2}{m(m-1)}$.

(assume your open addressing hash never collides with itself as you increment the probe count)

Dynamic

Programming

Key: Optimal Substructure

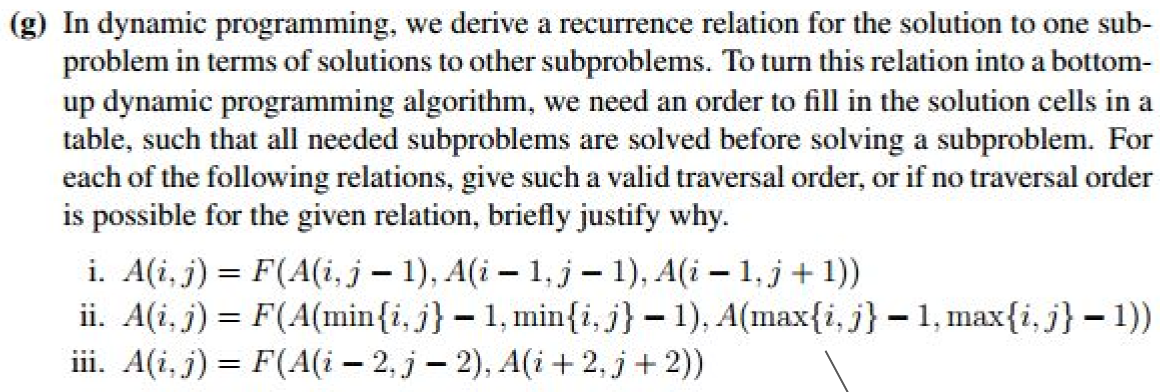
1. Solve problems recursively (similar to D & C)
    a. Piece together optimal solutions to sub-problems (“optimal SS”)
    b. (create a recurrence!)
2. Re-use solutions to existing sub-problems (“memoization”)
    a. Either “top-down” from recurrence
    b. Or build sub-problem solutions “bottom-up”

You have infinite number of bills (from your 6006LE salary)

The denominations in post-election USA are {1, 7, 20, 33, 39}

Algorithm to find the minimum number of bills required to create *n* dollars

Spring 2011 Final

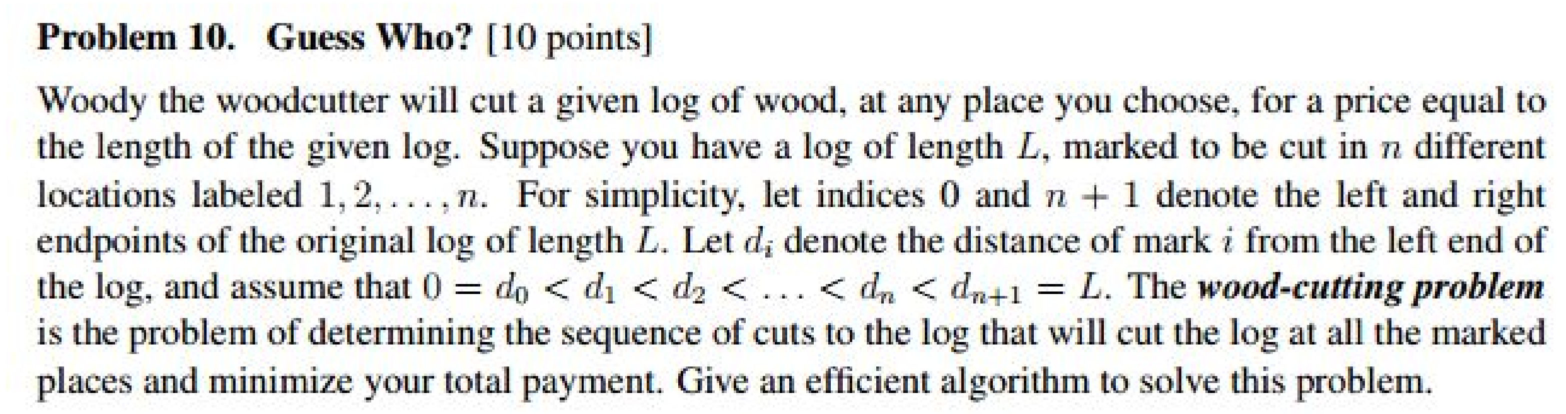


(g) In dynamic programming, we derive a recurrence relation for the solution to one subproblem in terms of solutions to other subproblems. To turn this relation into a bottom-up dynamic programming algorithm, we need an order to fill in the solution cells in a table, such that all needed subproblems are solved before solving a subproblem. For each of the following relations, give such a valid traversal order, or if no traversal order is possible for the given relation, briefly justify why.

i. $A(i, j) = F(A(i, j-1), A(i-1, j-1), A(i-1, j+1))$
ii. $A(i, j) = F(A(\min\{i, j\}-1, \min\{i, j\}-1), A(\max\{i, j\}-1, \max\{i, j\}-1))$
iii. $A(i, j) = F(A(i-2, j-2), A(i+2, j+2))$

Assume there exists a closed form solution for A(k,k)

Spring 2011 Final



Problem 10. Guess Who? [10 points]

Woody the woodcutter will cut a given log of wood, at any place you choose, for a price equal to the length of the given log. Suppose you have a log of length L, marked to be cut in n different locations labeled $1, 2, \ldots, n$. For simplicity, let indices 0 and $n+1$ denote the left and right endpoints of the original log of length L. Let d_i denote the distance of mark i from the left end of the log, and assume that $0 = d_0 < d_1 < d_2 < \ldots < d_n < d_{n+1} = L$. The ***wood-cutting problem*** is the problem of determining the sequence of cuts to the log that will cut the log at all the marked places and minimize your total payment. Give an efficient algorithm to solve this problem.

What is the recurrence for T(i,j), the minimum number of cuts bounded by mark i and j?

Backup slides

True or False (6.006 Quiz 2 Spring 2016)

When using an adjacency matrix graph representation, relaxing all the edges in a graph G(V, E) can be done in O(E) time.

True or False (6.006 Quiz 2 Spring 2015)

If we perform DFS on an undirected graph G = (V, E) (starting from an arbitrary vertex), there will not be any cross edges.

Closest Pair (6.006 Quiz 2 Spring 2014)

Let G = (V, E) be an unweighted undirected graph. Let A, B ⊆ V be two subsets of vertices,not necessarily disjoint. Define the closest pair of nodes between A and B to be the pair of nodes (a, b) such that a ∈ A, b ∈ B, and δ(a, b) is the smallest among all such nodes.

You are given G (as an adjacency list) and the two sets A and B (each set is given as a linked list of nodes). Clearly describe an algorithm to find the closest pair of nodes between A and B.

Analyze the complexity of your algorithm as a function of |V | and |E|.

(More efficient algorithms will receive greater credit.)

ET Goes Home (6.006 Quiz 2 Fall 2012)

Consider an n×n square grid with lower left corner at (0, 0). Suppose that ET the Extra-Terrestrial begins at (0, 0) and needs to get to the mothership located at (n, n) by walking horizontal and vertical unit steps along the grid.

Normally, ET can walk exactly one unit step per second. However, there are k teleporters located at various points (with integer coordinates) on the grid; when ET reaches a teleporter at (p, q), he can choose to enter it. If he does so, he must wait three seconds for the teleporter to warm up; then, he will be instantly transported to any location that would normally take him five seconds of walking to reach from (p, q).

Design an O(n^2) algorithm that minimizes the amount of time it takes ET to get from (0, 0) from (n, n). Prove that your algorithm is O(n^2).

True or False (6.006 Quiz 2 Spring 2015)

A directed, acyclic, and simple graph with n vertices and (n choose 2) edges has a unique topological order. (A simple graph has no self-loops or duplicate edges.)