

COS 226
Final Exam Review
Spring 2016

Ananda Gunawardena
guna@cs.Princeton.edu

Logistics

- The final exam **time and location**
 - **Thursday May 19th from 9AM-12PM.**
 - Location
 - McCosh Hall 46: **Friday Precepts** P06, P07, P07A.
 - McCosh Hall 50: **Thursday Precepts** P01, P02, P02A, P03, P03A, P04, P05 and P99(flipped).
 - .
- **Exam Format**
 - Closed book, closed note.
 - You may bring one 8.5-by-11 sheet (both sides) with notes in your own handwriting to the exam.
 - No electronic devices (e.g., calculators, laptops, and cell phones).

Material covered

- The exam will *stress* material covered since the midterm, including the following components.
 - Lectures 13–23.
 - *Algorithms in Java, 4th edition*, Chapters 4–6.
 - Exercises 12–22.
 - Programming assignments 6–8
 - Wordnet, seam-carving, burrows-wheeler

Topics covered

Depth-first search

Kruskal's algorithm

Key-indexed counting

Knuth-Morris-Pratt substring search

RE to NFA

Run-length coding

Topological sort

Bellman-Ford algorithm

MSD radix sort

Rabin-Karp substring search

Ternary search tries

LZW compression

Breadth-first search

Dijkstra's algorithm

LSD radix sort

Boyer-Moore substring search

R-way tries

Huffman coding

Prim's algorithm

Ford-Fulkerson algorithm

3-way radix quicksort

Reductions

Burrows-Wheeler

Algorithms

- focus on understanding fundamentals, not memorizing details (eg: code)
- *Write down as **many algorithms** as you can recall*
- ***For each algorithm***
 - understand how it works on typical, worst case, best case input
 - How is it different from other algorithms for the same class of problems?
 - When is it effective?

Algorithms	Application Area
KMP, Boyer-Moore, Rabin-Karp	String search
Insertion, selection, mergesort, quicksort, 3-way quicksort	General Sorting
LSD, MSD, 3-way MSD	String/Radix Sorting
Linear search, binary search	General Search
Kosaraju-Sharir	Strong components in a directed graph
Topological Sort	Vertex ordering
DFS, BFS	Graph search, path detection, cycle detection
Dijkstra's, Bellman-Ford	Single-source shortest path
Ford-Fulkerson	MaxFlow-MinCut, Matching Algorithms
Huffman, run-length encoding, LZW, Burrows-Wheeler	Data compression
Prims (eager, lazy), Kruskal's	Minimum Spanning Tree (MST)

Data Structures

- *Write down as many data structures as you can*
- ***For each data structure***
 - How is it implemented?
 - Alternate implementations?
 - What type of problems are good for this data structure?
 - What are the memory requirements?

Data Structure	Properties
Arrays	Random access, contiguous memory, static
Resizable arrays	Random access, contiguous memory, dynamic with constant amortized cost per insert/delete
Linked Lists, Doubly Linked Lists	Sequential access, flexible memory allocation/deallocation
Priority Queues	Binary heap implementation, delMax, insert in worst case $O(\log N)$ time
Binary Search Trees	At most two children per node, worst case linear access, ordered
Red-Black Trees	Balanced BST, guaranteed $\log N$ operations
Kd-Trees	Efficient organization of multi-dimensional data
Directed Graphs	Set of vertices, set of directed edges
Undirected Graphs	Set of vertices, set of undirected edges
Tries	Supports efficient prefix lookup, store strings with common prefixes efficiently
Union-Find	Support two operations, union and find, implementations of quick union, quick find, weighted UF
Symbol tables	A key-value mapping, multiple implementations with red-black trees(ordered ST), arrays
Hash table	Constant time insert/search with uniform hashing, linear time worst case

Past Exams Problem Clusters

Identify the sort (Sorting invariants)	Maxflow-mincut (application of the algorithm)
Tilde Notation (counting comparisons, exchanges) Order of growth Analysis (recurrences)	Design Problems (data structure, algorithm, performance requirements)
Memory Calculation (size of data)	Graph Algorithms Trace (DFS, BFS, Kruskals, Prim's, Dijkstra's, SCA)
String Sort/Search (LSD, MSD, 3-way quicksort, Key-indexed counting, KMP, DFA construction, Boyer-Moore, Rabin-Karp)	RegEx/NFA (construction, tracing)
Tries and TST (construction, operations, memory)	Compression (Huffman, LZW compress/expand, Burrows-Wheeler)
Reductions (3sum → 3sumvariant, longest path → longest cycle)	Miscellaneous (Matching, True/False, Possible/impossible)

Analysis of Algorithms

```
public static int f2(int N, int R) {  
    int x = 0;  
    for (int i = 0; i < R; i++)  
        x += f1(i);  
    return x;  
}
```

Assume $f1(N)$ is of $O(N)$

$$\begin{aligned} f2(N,R) &= f1(0) + f1(1) + \dots + f1(R-1) \\ &= 0 + 1 + \dots + R-1 \sim O(R^2) \end{aligned}$$

```
public static int f5(int N, int R) {  
    int x = 0;  
    for (int i = 0; i < N; i++)  
        for (int j = 1; j <= R; j += j)  
            x += f1(j);  
    return x;  
}
```

$$\begin{aligned} f5(N,R) &= N (1 + 2 + 4 + \dots + R) \\ &\sim N (2R) \\ &\sim O(NR) \end{aligned}$$

Analysis of Algorithms ctd..

Assume f1 is of O(N)

```
public static int f4(int N) {  
    if (N == 0) return 0;  
    return f4(N/2) + f1(N) + f1(N) + f1(N) + f4(N/2);  
}
```

$$\begin{aligned}f4(N) &= 2 f4(N/2) + 3 f1(N) \\ &= 2 f4(N/2) + 3 N \quad (\text{like mergesort}) \\ &\sim O(N \log N)\end{aligned}$$

```
public static int f3(int N) {  
    if (N == 0) return 1;  
    int x = 0;  
    for (int i = 0; i < N; i++)  
        x += f3(N-1);  
    return x;  
}
```

$$\begin{aligned}f3(N) &= N * f3(N-1) \\ &\sim O(N!)\end{aligned}$$

Counting Memory

```
public class Bag<Item> implements Iterable<Item> {  
    private Node<Item> first;    // beginning of bag  
    private int N;                // number of elements in bag  
  
    // helper linked list class  
    private static class Node<Item> {  
        private Item item;  
        private Node<Item> next;  
    }  
}
```

Node = 16 + 8 + 8 = 32 (no inner class overhead)

Bag of N Nodes = 32N + 16 + 8 + 4 + 4 (padding)
~ 32N

Graphs

problem	description
s-t path	<i>Is there a path between s and t ?</i>
shortest s-t path	<i>What is the shortest path between s and t ?</i>
cycle	<i>Is there a cycle in the graph ?</i>
Euler cycle	<i>Is there a cycle that uses each edge exactly once ?</i>
Hamilton cycle	<i>Is there a cycle that uses each vertex exactly once ?</i>
connectivity	<i>Is there a path between every pair of vertices ?</i>
biconnectivity	<i>Is there a vertex whose removal disconnects the graph ?</i>
planarity	<i>Can the graph be drawn in the plane with no crossing edges ?</i>
graph isomorphism	<i>Are two graphs isomorphic?</i>

Graph Order Traversals

- **Preorder:** order in which dfs() is called.

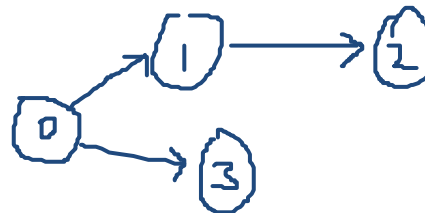
0 1 2 3

- **Postorder:** order in which dfs() returns.

2 1 3 0

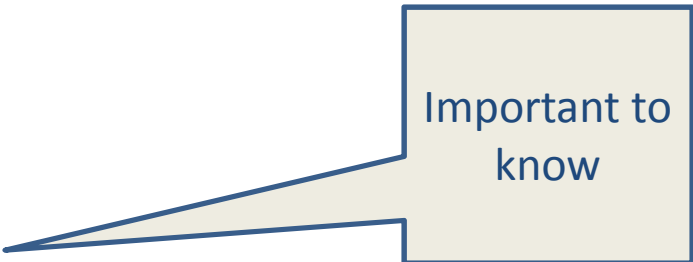
- **Reverse postorder:** reverse order in which dfs() returns.

0 3 1 2



DFS vs BFS

- DFS enables
 - Reachability.
 - Path finding.
 - Topological sort.
 - Directed cycle detection.
- BFS enables
 - Single source shortest path



Important to know

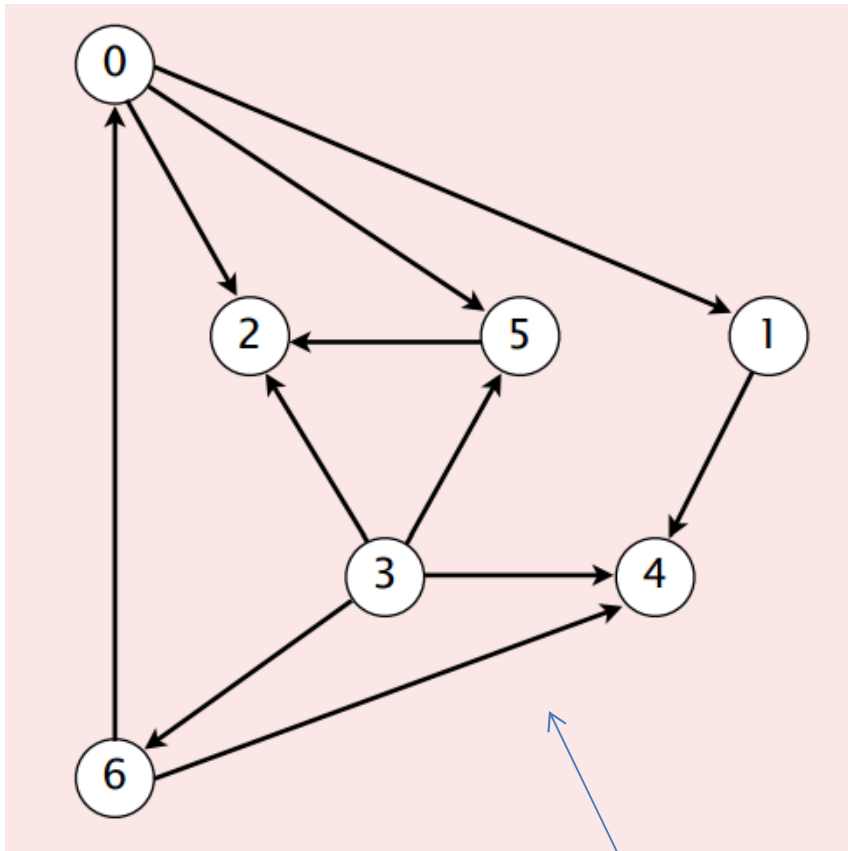
Mystery Code

```
for (Edge e : G.adj(v))
{
    int w = e.to();
    if (dist[w] > dist[v] + e.weight())
    {
        dist[w] = dist[v] + e.weight();
        pred[w] = e;
        pq.insert(dist[w], w);
    }
}
```

This partial code belongs to one of the graph algorithms. Which one(s)?

This is piece of a shortest path algorithm

Finding SCC's



v and w are **connected** if there is a path between v and w

v and w are **strongly connected** if there is both a directed path from v to w and a directed path from w to v

Finding the SCC

1. Find the reverse Post order of G^R
4 2 5 1 0 6 3
2. Traverse the original graph G in the order found in 1

The outcome is each component is its own strong component. $\{0\}$ $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$

Find the Strongly Connected Components of this Graph

Hashing

- When implementing a ST with hashing, what operations are not allowed in the ST?

Ordered operations are not allowed (like max, min, range)

- What is a collision in a hashtable?

Given two keys $K1$ and $K2$, $H(K1) = H(K2)$

- How can we minimize collisions?

Use a uniform hashing function

- With uniform hashing assumption, can we assure that hashtable operations always perform at $O(1)$

No. There can be bad input, where all keys anyway map to the same place.

Hashing

- True/False

- A linear probing hash table always finds a place

Yes, if the number of keys M is less than or equal to the table size N

–

- A separate chaining hash table always finds a place

Yes

- The load factor of a hash table is always ≤ 1

**The load factor is M/N and if we are using a linear probing table this is true.
Otherwise**

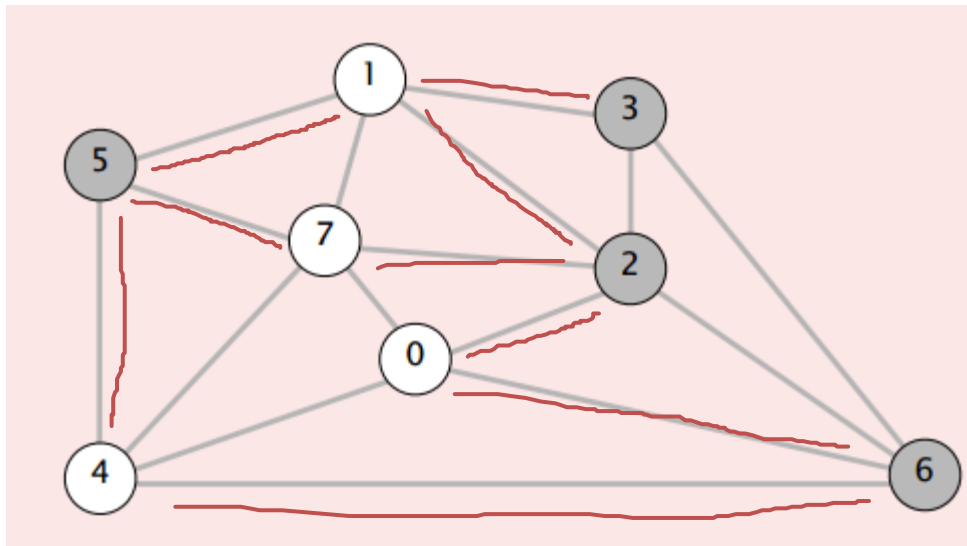
- A linear probing hash table must be rehashed if load factor is over 0.7

Yes, otherwise the performance can degrade

- A rehashed entry will be at the same location as the original

**Not necessarily as the table size will change and hence we will have a
different % value for each hash code**

MST

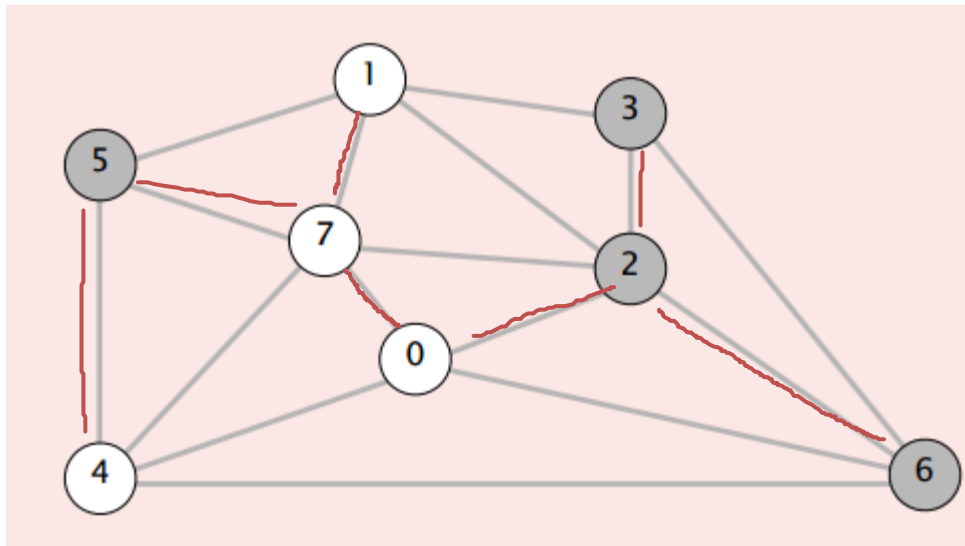


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Which is the min weight edge crossing the cut $\{2, 3, 5, 6\}$?

In this case draw the edges that crosses the cut $\{2,3,5,6\}$ (as shown in red)
And find the one with the lowest edge cost (0, 2, 0.26)

Kruskal's



operation	frequency	time per op
build pq	1	E
delete-min	E	$\log E$
union	V	$\log V$
connected	E	$\log V$

0-7	0.16	✓	Kruskals algorithm
2-3	0.17	✓	
1-7	0.19	✓	
0-2	0.26	✓	1. build a PQ of edges.
5-7	0.28	✓	
1-3	0.29	✗	2. Delete min and add the edge and vertices to MST as long as no cycles are created
1-5	0.32	✗	
2-7	0.34	✗	
4-5	0.35	✓	
1-2	0.36	✗	
4-7	0.37	✗	
0-4	0.38	✗	
6-2	0.40	✓	
3-6	0.52		
6-0	0.58		
6-4	0.93		

We are done now,
since we have added
 $V-1$ edges to MST

E

Prim's (Lazy)

```
pq = new MinPQ<Edge>();
mst = new Queue<Edge>();
marked = new boolean[G.V()];
visit(G, 0);

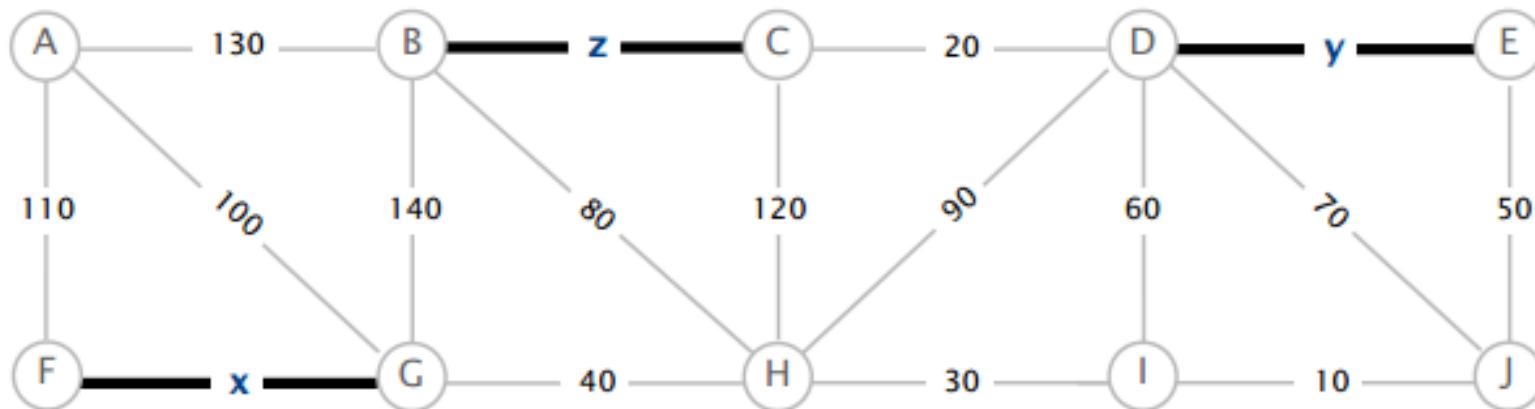
while (!pq.isEmpty() && mst.size() < G.V() - 1)
{
    Edge e = pq.delMin();
    int v = e.either(), w = e.other(v);
    if (marked[v] && marked[w]) continue;
    mst.enqueue(e);
    if (!marked[v]) visit(G, v);
    if (!marked[w]) visit(G, w);
}
```

```
private void visit(WeightedGraph G, int v)
{
    marked[v] = true;
    for (Edge e : G.adj(v))
        if (!marked[e.other(v)])
            pq.insert(e);
}
```

This is the code for Prim's Lazy algorithm

1. It maintains a PQ of edges (space E)
2. It starts to build MST from node 0
3. Find the min edge weight that connects 0 to other nodes. Add that to MST
4. Continue this until we have found $V-1$ edges
5. Notes
 1. The complexity of this algorithm is also $E \log E$
 2. It uses extra space proportional to E
6. Another version of this is Prim's eager algorithm. In that case, you maintain a PQ of vertices instead of edges. $E \log V$ order of growth

MST



If edges X, Y and Z are in the MST

1. Find the other edges that are in MST

Choose the min edges such that they do not form a cycle. They are
10, 20, 30, 40, 50 and 100

2. find upper bounds for edge costs of X, Y and Z?

1. $X \leq 110$, otherwise we can replace X with edge A-F
2. $Y \leq 60$, otherwise we can replace Y with edge D-I
3. $Z \leq 80$, otherwise we can replace Z with edge B-H

Shortest Paths

Generic algorithm (to compute a SPT from s)

Initialize $\text{distTo}[s] = 0$ and $\text{distTo}[v] = \infty$ for all other vertices.

Repeat until optimality conditions are satisfied:

- Relax any edge.**
-

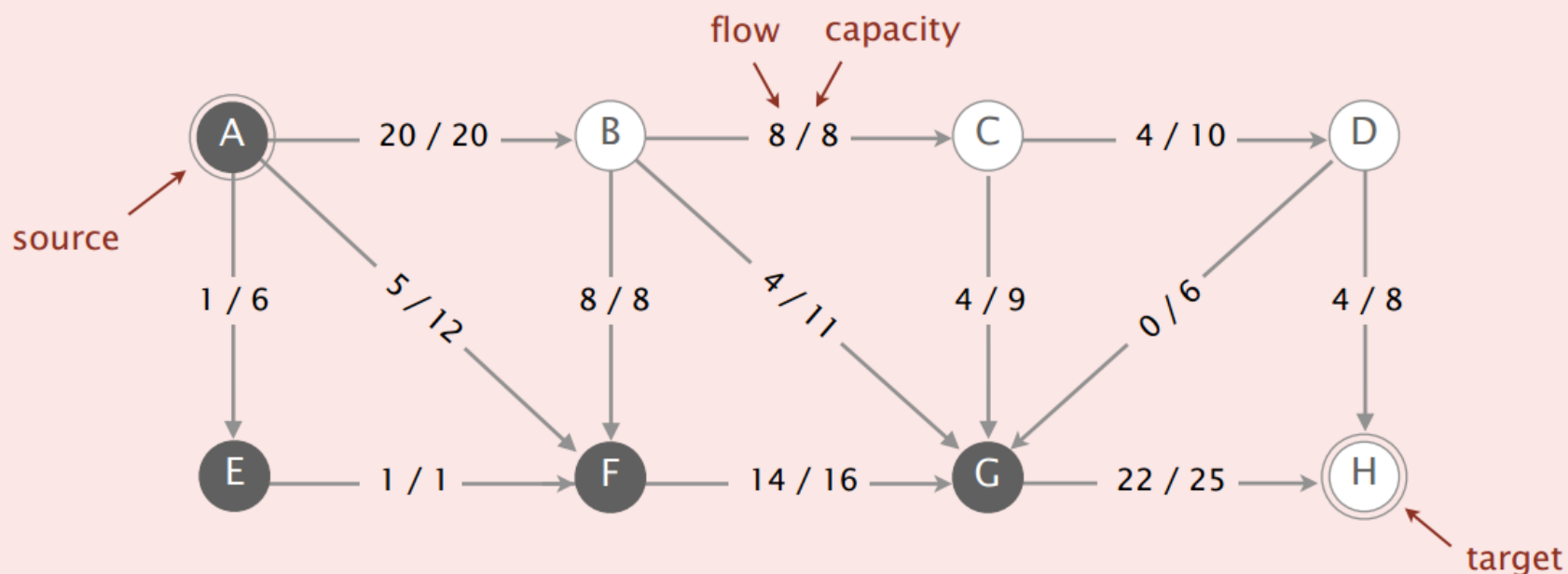
Specific Algorithms

Dijkstra's algorithm (nonnegative weights).

Topological sort algorithm (no directed cycles).

Bellman–Ford algorithm (no negative cycles).

Max Flow



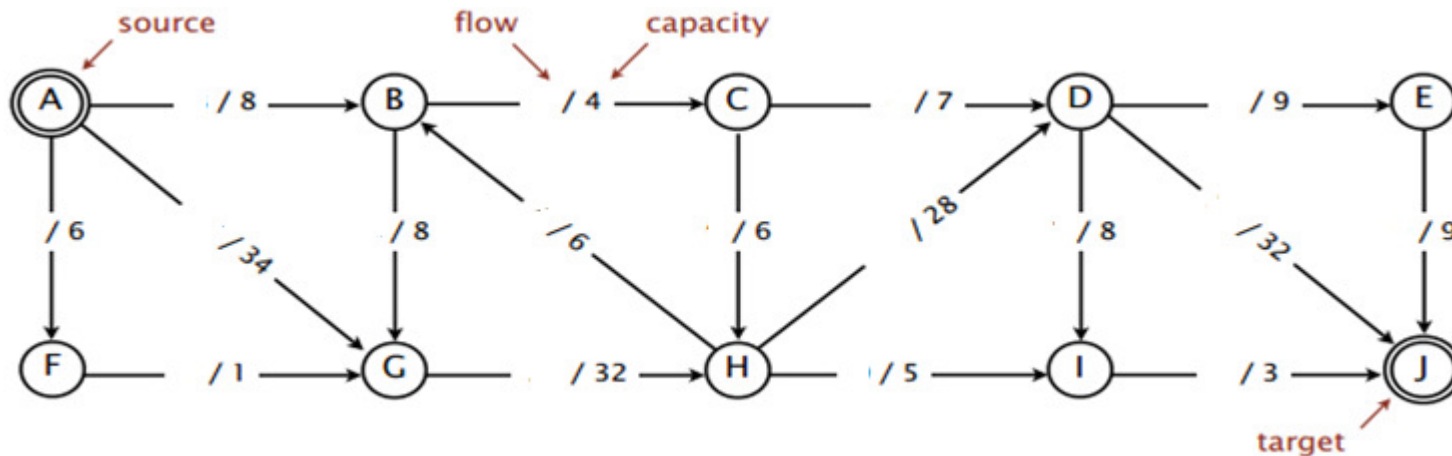
Fundamental questions.

- How to compute a mincut? **Easy. ✓**
- How to find an augmenting path? **BFS works well.**
- If FF terminates, does it always compute a maxflow? **Yes. ✓**
- Does FF always terminate? If so, after how many augmentations?

yes, provided edge capacities are integers
(or augmenting paths are chosen carefully)

requires clever analysis

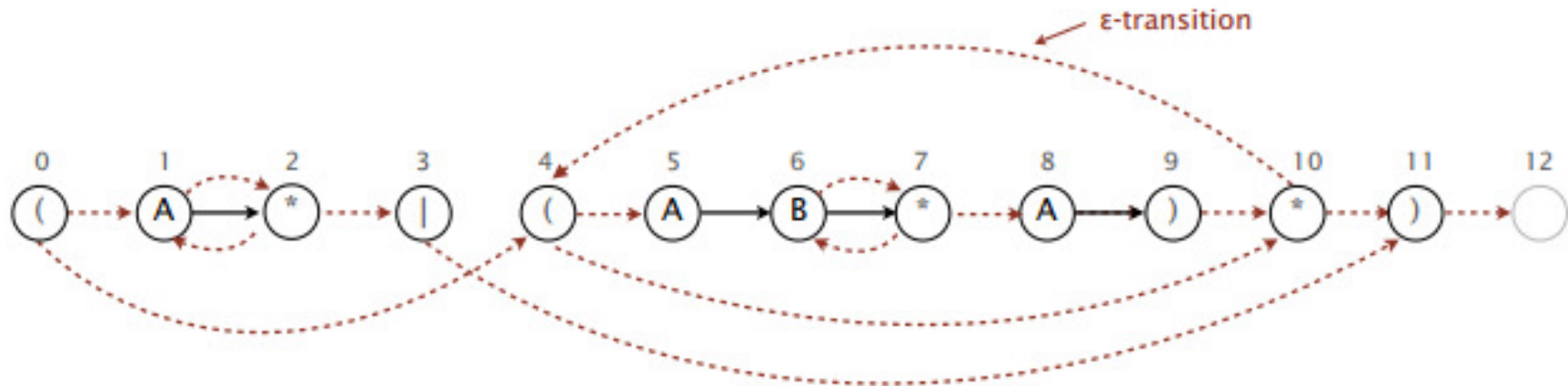
Ford-Fulkerson



1. What is the possible max flow of the network?
2. Mark an augmenting path and increase flow
3. Find all augmenting paths and increase flow
4. What is the actual max flow?
5. What is a min-cut?
6. Min-cut can only be calculated when a certain condition is true. What is it? How do we find out?

Find a max flow and min cut for the flow diagram as a homework problem

Regular Expressions and NFA



1. What is the regular expression?

$(A^* \mid (A B^* A)^*)$

2. Suppose that you simulate the following sequence of characters on the NFA above: **A A A A A A A** . In which one or more states could the NFA be?

Empty = {1, 2, 3, 11, 12, 4, 5, 10}

A = {1, 2, 3, 6, 7, 8, 10, 11, 12}

AA = {1, 2, 3, 11, 12, 9, 10}

3. Suppose that you want to construct an NFA for the regular expression $(A^* \mid (A B^+ A)^+)$ where the operator $+$ means one or more copies. What minimal change(s) would you make to the NFA above?

Remove the epsilon edge from state 4 to state 10

Compression Algorithms

Run-length encoding

- Describe the algorithm

If we have long runs of say 0's and 1's they can be encoded by an integer. For example, if we have 200 0's followed by 300 1's, we can represent them by two integers 200 100
If the integers take on 16 bits each, we have encoded 500 original bits with 32 bits

- Under what circumstances would you use this algorithm?

For example, an image with many white spaces like a fax machine output

- If 8-bit words are used to store counts, what is the length of the maximum run that can be stored?

The max number we can represent with 8-bits is $2^8 - 1 = 255$

- What can we do if the length of the run cannot be accommodated by n-bit word?

Encode with 0's. For example if the compressed file is : 255 0 54 0 etc.. What we have is a file that has 255 0's followed by 54 1's. In the decompression step we ignore the runs of length 0

- What is the best case input for run-length encoding (8-bit code words)

When we can encode 255 bits with just 8 bits. A compression ratio of 8:255

Compression Algorithms

Huffman encoding

- Describe the algorithm

Given a file, find the frequencies of each character. Build a code (pre-fix free) such that characters with high frequencies have lower length codes

- What is the preprocessing step of the algorithm?

Using a PQ of counts, pick the smallest two counts first, combine them. Add that to the PQ. Continue this until there is one tree. The codes can be obtained from this Trie

- What data structure(s) is/are used in the preprocessing step?

Need a PQ and some way to maintain a Trie structure

- If you compress a file with all characters the same (eg: 10000 A's) what is the compression ratio?

All characters can be represented with 1 or 0. So we can compress one character (8-bits) to 1 bit. So we have 1:8 ratio

- Describe a situation where no compression is obtained

Suppose there is a file with all 256 characters with the same frequency. Then we will create a nice looking huffman tree whose height is $\log(256) = 8$. So each character would require 8 bits to compress. No advantage. In fact the file size can increase since we need to transmit the code table

Compression Algorithms

LZW encoding

- Describe the algorithm
The algorithm stores codes for strings of words it has seen before. As algorithm progresses, it finds longer and longer strings and encode them with some value
- What data structure is used to store code words?
Code words are stored in a Trie
- Is it possible to run out of code words to store new words? How?
Yes, if we use 16-bit code words, then we can have a max of $2^{16} - \{\text{alphabet size}\}$ of codes. But that is not a big problem, as encoding longer strings with code words, probably won't be useful any way
- Should we send the code words with the compressed file?
No unlike the huffman code (where we have to send the code table), in LZW, we can reconstruct the code table as we decompress
- How can we decompress a file?
Most likely the first few numbers are alphabet codes like 41, 42. We can decompress them with A B etc and then add AB to the table as next code word. Continue this process.
- What is the tricky case and how do we overcome that?
Tricky case is when a code appears when there is no matching string in the table. This is usually caused when a code is immediately used by the the compression algorithm. The way to deal with this is to know that each new code is previous code + some other character. So if the last encoded message was: abb, then the missing code should have been : abb + a

LZW

97	a	ab = 128
98	b	ba = 129
128	ab	abb = 130
129	ba	we don't know what to add
131	ba + b	bab = 131 (previous code: ba + first character b)
132	bab + b	babb = 132 (sane as above)
130	abb	

Decode the message a=97,
b=98, and start next token
from 128

KMP Algorithm

- Briefly describe the algorithm

This is one of the very clever algorithms that was ever invented. The key idea is to remember the strings we have seen before, so we do not need to backup the text

- What is the order of growth of building the DFA? Typical algorithm? Best algorithm?
If the string is size M , then it would cost $M R$ to build the DFA where R is the alphabet size. As an order of growth we would need $O(M)$. The typical case is M^2 , but in the best case this can be done in linear time

- What is the order of growth of the algorithm for searching for a pattern of length m in a text of length n ?

$M + N$

- Can KMP be adjusted to find all occurrences of a pattern in a text? What is the order of growth?

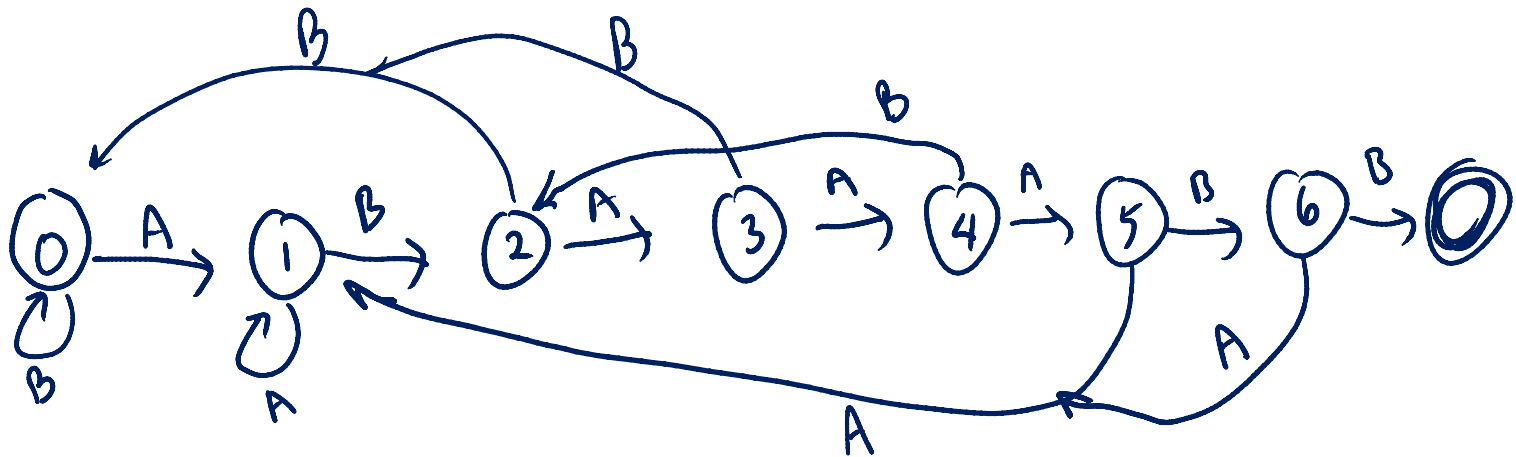
Sure, reset the DFA and keep looking for the next pattern. The order of growth is $M + N$

- Is KMP the algorithm of choice for any substring search application?

Not necessarily. But is the only one with guaranteed linear performance.

Exercise

- Build the DFA for : ABAAABB



Exercise

	0	1	2	3	4	5	6	7	8	9	10
A	0	0	3				7			10	11
B	1	2		4	5	6	2	8	9		4
s	B	B	A	B	B	B	A	B	B	A	A

Complete the table and find the search string

same

Boyer Moore (BM)

- Briefly describe the algorithm

Start comparing characters from right to left. You can do bigger jumps when they don't match.

- What is the pre-processing step of Boyer-Moore?

Create a table of jumps we need to make when it fails. Align the text with the last occurrence of the failed character in the pattern

- What is the order of growth of the algorithm for searching for a pattern of length m in a text of length n ? Best case? Worst case?

Best case, it can be sub-linear making jumps in the order of m . So we can do this in n/m time. The worst case still can be $m n$ (consider text=aaaaaaaaabaa and pattern = baa)

- Can BW be adjusted to find all occurrences of a pattern in a text? What is the order of growth?

Yes, Just keep looking as the substring after the matched pattern is a new string. Order of growth would be the same.

Rabin-Karp (RK)

- Briefly describe the algorithm

Given a text T and pattern M , find all substrings of T of the form $T(i \dots i+M-1)$ and compute hash codes. Compute the hash code of M and see if there is any substring with the same hashcode and then compare the characters

- What is the pre-processing step of RK algorithm?

Finding the hashes and this can be done efficiently

- What is the order of growth of the algorithm for searching for a pattern of length m in a text of length n ? Best case? Worst case?

To compute N hashes it cost N time. So it can be done in $M + N$

- Can RK be adjusted to find all occurrences of a pattern in a text? What is the order of growth?

Yes, this is really efficient since all occurrences of the string will have same hash codes

Challenge Questions

- **Answer each question with possible, impossible, unknown**

- There exist an algorithm where duplicity of elements in a set can be determined in sub-linear time

Not possible w/o examining at least all of the elements

- The convex hull problem (i.e. finding a set of points that encloses a given set of n points) can be solved in linearithmic time

Possible, since we know a way to do this

- Inserting n comparable keys into a BST in time proportional to n

Not possible. If it is, then we can sort in linear time and we know that is not true

Match Algorithms

- | | |
|---|---------------------------|
| A ___ T9 texting in a cell phone | A. Trie |
| D ___ 1D range search | B. Hashing |
| E ___ 2D range search | C. 3-way radix quicksort |
| B ___ Document similarity | D. Binary search tree |
| K ___ Traveling salesperson problem | E. Kd tree |
| G ___ Web crawler | F. Depth-first search |
| H ___ Google maps | G. Breadth-first search |
| I ___ PERT/CPM (Program Evaluation and Review Technique / Critical Path Method). | H. Dijkstra's algorithm |
| | I. Topological sort |
| | J. Bellman-Ford |
| | K. Enumerate permutations |

Classifying Algorithms

- (a) Which of the following can be performed in *linear time* in the *worst case*?
Write *P* (possible), *I* (impossible), or *U* (unknown).

P
___ Printing the keys in a binary search tree in ascending order.

P
___ Finding a minimum spanning tree in a weighted graph.

P
___ Finding all vertices reachable from a given source vertex in a graph.

P
___ Checking whether a digraph has a directed cycle.

p
___ Building the Knuth-Morris-Pratt DFA for a given string.

p
___ Sorting an array of strings, accessing the data solely via calls to `charAt()`.

i
___ Sorting an array of strings, accessing the data solely via calls to `compareTo()`.

i
___ Finding the closest pair of points among a set of points in the plane, accessing the data solely via calls to `distanceTo()`.

Wordnet

1. What data structures are used to store wordnet?

DiGraphs, HashMaps

2. What data structures are used to store SCA?

DiGraphs,

3. Is there a reason that we used BFS not DFS?

DFS can fail to find shortest path as shown in precepts

4. What is the order of the best algorithm that can find the length of the common ancestor?

$V + E$

5. What is the order of the best algorithm that can find the common ancestor?

$V + E$

6. What is a rooted DAG and how do we determine that? Order or growth of your algorithm?

A Digraph with one node, whose outdegree is 0. Graph must also be acyclic

7. If the wordnet is NOT a rooted DAG, will answers to 3 and 4 will hold?

No since there may not be a SCA (nodes in two disjoint subsets)

8. Given a list of n nouns, What is the order of growth of the outcast algorithm?

$n^2 (V + E)$

Seam-Carving

- What is the purpose of the seamcarving assignment?

To resize an image by removing seams that are low energy

- How does it relate to shortest path?

The problem can be reduced to finding the shortest path in a DAG. An image with $N \times N$ dimension can be represented as a graph with at most $3 \cdot N^2$ edges. So we have an algorithm of order: $3 \cdot N^2$

- How to find Vertical and Horizontal seams?

Find a column of pixels with the lowest energy total. Do the same for horizontal seams

- Why do a defensive copy of Picture?

To return to client a new copy of the changed image

- What is the order of growth for the two methods, removeHorizontalSeam and removeVerticalSeam?

$W \cdot H$

Burrows-Wheeler

- What libraries were used to read and write input/output to the program?
binaryStdIn and BinaryStdOut
- What method in the output library that was required to print the output correctly?
HexDump, Close, Flush
- What data structures were used to implement BW, CSA, MoveToFront?
BW – Circular Suffix Array
CSA uses Array and MoveToFront uses an Array
- What is the order of growth to form circular suffixes of a given string?
Can be done in linear time. One trick is to form a new string $S + S$ and then use start and end to keep track of the suffixes
- What is wrong with using `LSD.sort()` in the program?
Can be quadratic in performance
- Could we have used quicksort to sort suffixes? How? If so how can you avoid quadratic performance?
We could have used quicksort. However, the compareTo should be designed carefully to avoid suffixes comparing to itself leading to quadratic performance

Burrows-Wheeler ctd...

- What are the 3-steps to burrows-wheeler transform?

BW, move to front, huffman

- How would you sort strings w/o forming them explicitly?

You can use suffixes using the references to first and last characters of the suffix

- Is it necessary to do move to front? If not, why did we do it?

Not necessary. But it helps with the huffman by creating only a few characters with many duplicates

- How did we do the inverse transform?

We did the next array to figure out how to get string from the suffix sort

Good Luck with All Finals