Day, Week, Month Overview

- Backtracking p/review
  - Boggle
  - APT

- Assignment review
  - Boggle
  - APT

- PQ Overview

Backtracking by image search

Searching with no guarantees

- Search for best move in automated game play
  - Can we explore every move?
  - Are there candidate moves ranked by “goodness”?
  - Can we explore entire tree of possible moves?

- Search with partial information
  - Predictive texting with T9 or iTap or …
  - Finding words on a Boggle board
  - What numbers fit in Sudoku square

- Try something, if at first you don’t succeed …

Search, Backtracking, Heuristics

- How do you find a needle in a haystack?
  - How does a computer play chess?
  - Why would you write that program?

- How does Bing/Googlemap find routes from one place to another?
  - Shortest path algorithms
  - Longest path algorithms

- Optimal algorithms and heuristic algorithms
  - When is close good enough? How do measure “closeness”?
  - When is optimality important, how much does it cost?
Exhaustive Search/Heuristics

- We use binary search trees to organize data, in searching we don’t need to examine all the data to find what we’re looking for
  - Where is the smallest item in a search tree? Largest?
  - How are trees balanced?

- What do we do when the search space is huge?
  - How many chess boards are there?
  - Count routes between my house and yours?

- Exhaustive search: look at everything!

Classic problem: N queens

- Can queens be placed on a chess board so that no queens attack each other?
  - Easily place two queens
  - What about 8 queens?

- Make the board NxN, this is the N queens problem
  - Place one queen/column
  - Horiz/Vert/Diag attacks

- Backtracking
  - Tentative placement
  - Recurse, if ok done!
  - If fail, undo tentative, retry

- wikipedia-n-queens

N queens backtracking: Queens.java

```java
public boolean solve(int col){
    if (col == mySize) return true;
    // try each row until all are tried
    for(int r=0; r < mySize; r++){
        if (myBoard.safeToPlace(r, col)){
            myBoard.setQueen(r, col, true);
            if (solve(col+1)){
                return true;
            }
            myBoard.setQueen(r, col, false);
        }
    }
    return false;
}
```

Backtracking idea with N queens

- For each column C, tentatively place a queen
  - Try first row in column C, if ok, move onto next column
    - Typically “move on” is recursive
  - If solved, done, otherwise try next row in column C
    - Must unplace queen when failing/unwind recursion

- Each column C “knows” what row R it’s on
  - If first time, that’s row zero, but might be an attack
  - Unwind recursion/backtrack, try “next” location

- Backtracking: record an attempt go forward
  - Move must be “undoable” on backtracking/unwinding
Basic ideas in backtracking search

- **Enumerate all possible choices/moves**
  - We try these choices in order, committing to a choice
  - If the choice doesn’t pan out we must undo the choice
    • Backtracking step, choices must be undoable

- **Inherently recursive, when to stop searching?**
  - When all columns tried in N queens
  - When we have found the exit in a maze
  - When every possible moved tried in Tic-tac-toe or chess?
    • Is there a difference between these games?

- **Summary:** enumerate choices, try a choice, undo a choice, this is *brute force* search: try everything

Pruning vs. Exhaustive Search

- **If we consider every possible placement of 4 queens on a 4x4 board, how many are there? (N queens)**
  - 4x4x4x4 if we don’t pay attention to any attacks
  - 4x3x2x1 if we avoid attacks in same row

- **What about if we avoid diagonal attacks?**
  - Pruning search space makes more search possible, still could be lots of searching to do!

- **Estimate how long to calculate # solutions to the N-queens problem with our Java code....**

Daphne Koller

- **2004, Macarthur**
- **2008, first ACM/Infosys**

“**The world is noisy and messy …You need to deal with the noise and uncertainty.**

“I find it distressing that the view of the field is that you sit in your office by yourself surrounded by old pizza boxes and cans of Coke, hacking away at the bowels of the Windows operating system,” she said. “I spend most of my time thinking about things like how does a cell work or how do we understand images in the world around us?”

http://tinyurl.com/3tdlug

Computer v. Human in Games

- **Computers can explore a large search space of moves quickly**
  - How many moves possible in chess, for example?

- **Computers cannot explore every move (why) so must use heuristics**
  - Rules of thumb about position, strategy, board evaluation
  - Try a move, undo it and try another, track the best move

- **What do humans do well in these games? What about computers?**
  - What about at Duke?
Games at Duke

- **Alan Biermann**
  - Natural language processing
  - Compsci 1: Great Ideas
  - Duchess, checkers, chess

- **Tom Truscott**
  - Duke undergraduate working with/for Biermann
  - Usenet: online community
  - Second EFF Pioneer Award (with Vint Cerf!)

Heuristics

- A heuristic is a rule of thumb, doesn’t always work, isn’t guaranteed to work, but useful in many/most cases
  - Search problems that are “big” often can be approximated or solved with the right heuristics

- What heuristic is good for Sudoku?
  - Is there always a no-reasoning move, e.g., 5 goes here?
  - What about “if I put a 5 here, then…”
  - Do something else?


- What other optimizations/improvements can we make?
  - For chess, checkers: good heuristics, good data structures

Boggle Program

OO Aside: Interfaces, Abstract Classes

- **Interface is a contract**
  - X implements Y provides method implementations
  - No code sharing among implementing classes
  - Characteristics of interface: signatures only
  - Why create an interface?

- **Abstract class leverages common code**
  - Often implements interface with common code
  - Consider Map/AbstractMap, List/AbstractList,…
  - One method labeled abstract, requires implementation!
Interfaces and Abstraction in Boggle

- General idea: create I Thing for interfaces
  - IPlayer, IAutoPlayer, ILexicon, IBoardMaker
  - Facilitates new implementations if Ixx fixed
    - Changing the interface creates cascade of changes

- AbstractPlayer and AbstractAutoPlayer
  - Factor out common code, differentiate in subclasses
  - Facilitate play by Controller/View, back-and-forth plays
  - Be careful, be wary of ignoring/not knowing parent code
    - How do you update score in an BoardFirstAutoPlayer?

- How can you create different boards?

YAQ, YAQ, haha! (Yet Another Queue)

- What is the dequeue policy for a Queue?
  - Why do we implement Queue with LinkedList
    - Interface and class in java.util
  - Can we remove an element other than first?

- How does queue help word-ladder/shortest path?
  - First item enqueued/added is the one we want
  - What if different element is “best”?

- PriorityQueue has a different dequeue policy
  - Best item is dequeued, queue manages itself to ensure operations are efficient

PriorityQueue raison d’être

- Algorithms Using PQ for efficiency
  - Shortest Path: Mapquest/Garmin to Internet Routing
    - How is this like word-ladder? How different?
  - Event based simulation
    - Coping with explosion in number of particles or things
  - Optimal A* search, game-playing, AI,
    - Can’t explore entire search space, can estimate good move

- Data compression facilitated by priority queue
  - Alltime best assignment in a Comp sci 100/100e course?
    - Subject to debate, of course
  - From A-Z, soup-to-nuts, bits to abstractions

PQ Application: Data Compression

- Compression is a high-profile application
  - .zip, .mp3, .jpg, .gif, .gz, ...
  - What property of MP3 was a significant factor in what made Napster work (why did Napster ultimately fail?)
  - Who invented Napster, how old, when?

- Why do we care?
  - Secondary storage capacity doubles every year
  - Disk space fills up quickly on every computer system
  - More data to compress than ever before
  - Will we ever need to stop worrying about storage?
More on Compression

- Different compression techniques
  - .mp3 files and .zip files?
  - .gif and .jpg?
  - Lossless and lossy

- Impossible to compress/lossless everything: Why?
  - Lossy methods
    - Good for pictures, video, and audio (JPEG, MPEG, etc.)
  - Lossless methods
    - Run-length encoding, Huffman, LZW, ...

Priority Queue

- Compression motivates ADT priority queue
  - Supports two basic operations
    - add/insert -- an element into the priority queue
    - remove/delete -- the minimal element from the priority queue
  - Implementations allow getmin/peek as well as delete
    - Analogous to top/pop, peek/dequeue in stacks, queues

- Think about implementing the ADT, choices?
  - Add compared to min/remove
  - Balanced search tree is ok, but can we do better?

Priority Queue sorting

- See PQDemo.java,
  - code below sorts, complexity?

```java
String[] array = {...}; // array filled with data
PriorityQueue<String> pq = new PriorityQueue<String>();
for(String s : array) pq.add(s);
for(int k=0; k < array.length; k++)
  array[k] = pq.remove();
```

- Bottlenecks, operations in code above
  - Add words one-at-a-time to PQ v. all-at-once
  - What if PQ is an array, add or remove fast/slow?
  - We’d like PQ to have tree characteristics, why?

Priority Queue top-M sorting

- What if we have lots and lots and lots of data
  - code below sorts top-M elements, complexity?

```java
Scanner s = ... // initialize;
PriorityQueue<String> pq = new PriorityQueue<String>();
while (s.hasNext())
  pq.add(s.next());
if (pq.size() > M) pq.remove();
```

- What’s advantageous about this code?
  - Store everything and sort everything?
  - Store everything, sort first M?
  - What is complexity of sort: O(n log n)
Priority Queue implementations

- Priority queues: average and worst case

<table>
<thead>
<tr>
<th></th>
<th>Insert average</th>
<th>Getmin (delete)</th>
<th>Insert worst</th>
<th>Getmin (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted list</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Sorted list</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Search tree</td>
<td>log n</td>
<td>log n</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Balanced tree</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td>Heap</td>
<td>O(1)</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
</tbody>
</table>

- Heap has $O(n)$ build heap from $n$ elements

PriorityQueue.java (Java 5+)

- What about objects inserted into pq?
  - Comparable, e.g., essentially sortable
  - How can we change what minimal means?
  - Implementation uses heap, tree stored in an array

- Use a Comparator for comparing entries we can make a min-heap act like a max-heap, see PQDemo
  - Where is class Comparator declaration? How used?
  - What if we didn’t know about Collections.reverseOrder?
    - How do we make this ourselves?

Big-Oh and a tighter look at inserts

- $\log(1) + \log(2) + \log(3) + \ldots + \log(n)$
  - Property of logs, $\log(a) + \log(b) = \log(a*b)$
  - $\log(1*2*3*\ldots*n) = \log(n!)$

- We can show using Sterling’s formula:
  - $n! \approx \sqrt{2\pi n} n^n e^{-n}$

- $\log(n!) = c_1 + \log(n) + n\log(n) - c_2 \cdot n$
- We can get $O(n \log n)$ easily, this goes tight, lower, $\Omega(n \log n)$ as well

Priority Queue implementation

- Heap data structure is fast and reasonably simple
  - Why not use inheritance hierarchy as was used with Map?
  - Trade-offs when using HashMap and TreeMap:
    - Time, space, ordering properties, TreeMap support?

- Changing comparison when calculating priority?
  - Create object to replace, or in lieu of compareTo
    - Comparable interface compares this to passed object
    - Comparator interface compares two passed objects
  - Both comparison methods: compareTo() and compare()
    - Compare two objects (parameters or self and parameter)
    - Returns -1, 0, +1 depending on $<$, $=$, $>$
Creating Heaps

- Heap: array-based implementation of binary tree used for implementing priority queues:
  - add/insert, peek/getmin, remove/deletemin, O(???)

- Array minimizes storage (no explicit pointers), faster too, contiguous (cache) and indexing

- Heap has shape property and heap/value property
  - shape: tree filled at all levels (except perhaps last) and filled left-to-right (complete binary tree)
  - each node has value smaller than both children

Array-based heap

- store "node values" in array beginning at index 1
- for node with index k
  - left child: index 2*k
  - right child: index 2*k+1

- why is this conducive for maintaining heap shape?
- what about heap property?
- is the heap a search tree?
- where is minimal node?
- where are nodes added? deleted?

Thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
- How many leaves are there in an N-node heap (big-Oh)?
  - O(n), but exact?
- What is complexity of find max in a minheap? Why?
  - O(n), but ½ N?
- Where is second smallest element? Why?
  - Near root?

Adding values to heap

- to maintain heap shape, must add new value in left-to-right order of last level
  - could violate heap property
  - move value “up” if too small

- change places with parent if heap property violated
  - stop when parent is smaller
  - stop when root is reached

- pull parent down, swapping isn’t necessary (optimization)
Adding values, details (pseudocode)

```java
void add(Object elt)
{
    // add elt to heap in myList
    myList.add(elt);
    int loc = myList.size() - 1;
    while (1 < loc &&
        elt < myList.get(loc/2)) {
        myList.set(loc, myList.get(loc/2));
        loc = loc/2; // go to parent
    }
    myList.set(loc, elt);
}
```

Removing minimal element

- Where is minimal element?
  - If we remove it, what changes, shape/property?
- How can we maintain shape?
  - “last” element moves to root
  - What property is violated?
- After moving last element, subtrees of root are heaps, why?
  - Move root down (pull child up) does it matter where?
- When can we stop “re-heaping”?
  - Less than both children
  - Reach a leaf

Anita Borg 1949-2003

- “Dr. Anita Borg tenaciously envisioned and set about to change the world for women and for technology. … she fought tirelessly for the development of technology with positive social and human impact.”
- “Anita Borg sought to revolutionize the world and the way we think about technology and its impact on our lives.”
- [http://www.youtube.com/watch?v=1yPxdsJazQ](http://www.youtube.com/watch?v=1yPxdsJazQ)