Late-Term Exam Review

PPT by Brandon Fain
Course Evaluations

• I want to stress that we take this seriously.

• Part of the reason this lab exists is in response to student feedback.

• Please let us know what you liked and did not like, what we should keep and what we can make better.
Course Evaluations

• Go to https://dukehub.duke.edu/

2. Click on the Evaluation icon (see image below) to begin the evaluation process. A course evaluation form will open up.
Late-Term Exam Format

• 4 problems, each will ask you to write and analyze an algorithm

• 2 from graph theory
  • Connectivity: DFS, connected components, cycles, topological sort
  • Short paths: BFS, Dijkstra’s, Bellman-Ford
  • Spanning Trees: Greedy, Prim, Kruskal

• 2 from other topics from lecture
  • Polynomial multiplication and FFT
  • Number theory algorithms and RSA
  • Pattern matching
  • Computational Geometry
  • Dynamic Programming
Chapter 30  Polynomials and the FFT

Polynomial Multiplication and FFT

\[
\begin{align*}
A(\omega_{2n}^0, B(\omega_{2n}^0) \\
A(\omega_{2n}^1, B(\omega_{2n}^1) \\
\vdots \\
A(\omega_{2n}^{2n-1}, B(\omega_{2n}^{2n-1})
\end{align*}
\]

\[
\begin{align*}
C(\omega_{2n}^0) \\
C(\omega_{2n}^1) \\
\vdots \\
C(\omega_{2n}^{2n-1})
\end{align*}
\]

Evaluation
Time \(\Theta(n \lg n)\)

Ordinary multiplication
Time \(\Theta(n^2)\)

Interpolation
Time \(\Theta(n \lg n)\)

Pointwise multiplication
Time \(\Theta(n)\)

Coefficient representations

Point-value representations
Public Key Cryptography

**Working of RSA**
1. Select at random two LARGE prime numbers $p$ and $q$ (100-200 decimal digits).

2. Compute $n = pq$.

3. Select a small odd integer $e$ relatively prime to $\phi(n) = (p - 1)(q - 1)$= number nontrivial factors of $n$

4. Compute $d$ such that $ed = 1 \mod \phi(n)$ (d exists and is unique!!!).

5. Publish the public key function $P_A(M) = M^e \mod n$ (the pair $(e, n)$).

6. Keep secret the secret key function $S_A(C) = C^d \mod n$. 

Pattern Matching

- **Input:** Two strings $T[1\ldots n]$ and $P[1\ldots m]$, containing symbols from alphabet $\Sigma$.
  
  E.g.:
  - $\Sigma=\{a,b,\ldots,z\}$
  - $T[1\ldots 18]=\text{“to be or not to be”}$
  - $P[1..2]=\text{“be”}$

- **Goal:** find all “shifts” $0 \leq s \leq n-m$ such that $T[s+1\ldots s+m]=P$
  
  E.g. 3, 16
Computational Geometry

- Algorithms for geometric problems
- Applications: CAD, GIS, computer vision, ....
- E.g., the closest pair problem:
  - Given: a set of points $P = \{p_1 \ldots p_n\}$ in the plane, such that $p_i = (x_i, y_i)$
  - Goal: find a pair $p_i \neq p_j$ that minimizes $\|p_i - p_j\|$ for $\|p-q\| = [(p_x - q_x)^2 + (p_y - q_y)^2]^{1/2}$
Computational Geometry

• Divide:
  – Compute the median of x-coordinates
  – Split the points into $P_L$ and $P_R$, each of size $n/2$

• Conquer: compute the closest pairs for $P_L$ and $P_R$

• Combine the results (the hard part)
Dynamic Programming

**Optimal substructure**
An optimal solution to a problem (instance) contains optimal solutions to subproblems.

**Overlapping subproblems**
A recursive solution contains a “small” number of distinct subproblems repeated many times.
Dynamic Programming

**Example: Longest Common Subsequence (LCS)**

- Given two sequences $x[1 \ldots m]$ and $y[1 \ldots n]$, find a longest subsequence common to them both.

  “a” not “the”

$x$: A B C B D A B  

$y$: B D C A B A

$BCBA = \text{LCS}(x, y)$

functional notation, but not a function
Dynamic Programming

**Strategy:** Consider *prefixes* of $x$ and $y$.
- Define $c[i, j] = |LCS(x[1 \ldots i], y[1 \ldots j])|$.
- Then, $c[m, n] = |LCS(x, y)|$.

**Theorem.**

$$c[i, j] = \begin{cases} 
  c[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\
  \max \{c[i-1, j], c[i, j-1]\} & \text{otherwise.}
\end{cases}$$
Dynamic Programming

\[ m = 3, \ n = 4: \]

\[ 3,4 \]

\[ 2,4 \]

\[ 1,4 \]

\[ 2,3 \]

\[ 1,3 \]

\[ 2,2 \]

\[ 3,3 \]

\[ 2,3 \]

\[ 1,3 \]

\[ 2,2 \]

\[ 3,2 \]

same subproblem

\[ m + n \]
Graph Connectivity – Depth First Search
Graph Connectivity – Depth First Search

• **Runtime.**
  • $O(|V| + |E|)$ using adjacency lists.
  • $O(|V|^2)$ using adjacency matrix
  • In a dense graph, both are the same.

• **Applications**
  • Connectivity - “Does there exist a path from $u$ to $v$?” Also, discovering connected components.
  • Cycle Detection – Just look for a “back” edge.
  • Topological Sort – Find a directed acyclic graph such that all edges are left to right (to do this, sort decreasing by finish time).
Short Paths – Breadth First Search

Runtime
• $O(|V|+|E|)$

Applications
• Shortest path in an unweighted graph
• Graph coloring / Testing for bipartite graph
Short Paths – Dijkstra’s Algorithm

• Exchange a standard queue in breadth first search for a priority queue maintained on minimum distance so far.

• Runtime
  • $O(|E|\log(|V|))$ with a binary heap

• Application
  • Shortest paths in weighted graphs with no negative edges
Short Paths – Bellman Ford

• Rather than making a clever exploration of the graph...
• Repeat $|V|-1$ times:
  • For every edge:
    • If that gives you a shorter path to some vertex, update.

• **Runtime**
  • $O(|V||E|)$

• **Application**
  • Shortest path in weighted graphs with negative edges but no negative cycles
  • Detecting negative cycles
Greedy Algorithm – Spanning Trees

• A tree is a connected graph with no cycles.
• A spanning tree is a tree with every vertex in the graph.
• Minimum spanning trees have a greedy choice property.

**Greedy-choice property**
A locally optimal choice is globally optimal.
Greedy Algorithm – Spanning Trees

Optimal substructure

MST $T$: (Other edges of $G$ are not shown.)

- The MST of $T_1 \cup T_2$ just takes the “cheapest” edge between the two components.
Greedy Algorithm – Spanning Trees

• This intuition yields two algorithms for minimum spanning trees.

• **Prim’s Algorithm** – Maintain a single tree / connected component. At each step include the vertex outside the current tree with the cheapest edge to the current tree.

• **Kruskal’s Algorithm** – Maintain many different trees / connected components. At each step, merge any two components using the cheapest edge possible.

• **Runtime**
  • $O(|E| \log(|V|))$ for both, but...
  • Prim’s algorithm just needs a priority queue, Kruskal’s algorithm needs a new disjoint set data structure for maintaining and merging components.
Questions?