ECS122A Final Review

Before you begin, find the following material:

- Lecture notes/slides
- 8 problem sets (*yes, including #8*)
- Solutions of problem sets
- Solution of midterm

ECS122A Final Review

Here is high-level organization of what we have learned:

I. Basics and tools of trade
II. Three algorithm design techniques
III. Graph algorithms
IV. NP-completeness – *a brief introduction*

I. Basics and tools of trade

1. Order of growth
   - O, Ω, Θ definitions
   - proof by definition
2. Recurrence relations
   - Linear recurrence relations
   - Divide and conquer recurrence relations
3. Solving the recurrence relations
   - Direct substitution
   - The master theorem/method for solving the DC recurrence relations

I. Basics and tools of trade

4. Graph terminology and representations
   - graph, path, connected graph, connected component, cycle, acyclic, tree, spanning tree, sink, ...
   - adjacency list, adjacency matrix, incidence matrix.
5. Data structures
   - FIFO queue:

     `enqueue`, `dequeue`
   - LIFO stack
   - Priority queue:

     `Insert`(S,x), `Extract-Min`(S), `Decrease-Key`(S,x,k), ...
   - Disjoint-set:

     `Make-set`(x), `Union`(x,y), `Find-set`(x)

II. Algorithm design techniques

Divide and Conquer algorithms

[1] If the subproblem sizes are small enough, however, just solve them in a straightforward manner.

II. Algorithm design techniques

Divide and Conquer algorithms

- Three steps:
  - **Divide** the problem into a number of *independent* subproblems;
  - **Conquer** subproblems by solving them *recursively*;[1]
  - **Combine** the solutions to the subproblems into the solution of the original problem

[1] If the subproblem sizes are small enough, however, just solve them in a straightforward manner.

II. Algorithm design techniques

Divide and Conquer algorithms

- Three steps:
  - **Divide** the problem into a number of *independent* subproblems;
  - **Conquer** subproblems by solving them *recursively*;[1]
  - **Combine** the solutions to the subproblems into the solution of the original problem

- Examples:
  1. Merge sort (vs. Insert sort)
  2. The maximum and minimum values
  3. The maximum subarray
  4. Strassen's algorithm for matrix-matrix multiplication
  5. the closest pair of points in one dimension.
  6. Searching for index i such that $A[i] = i$ in a sorted array A
  7. Integer multiplication
  8. k-way merge operation

[1] If the subproblem sizes are small enough, however, just solve them in a straightforward manner.

II. Algorithm design techniques

Greedy Algorithms

II. Algorithm design techniques

Greedy Algorithms

- Two key elements:
  - **The greedy-choice property:** a globally optimal solution can be arrived at by making a locally optimal (greedy) choice.
  - **The optimal substructure property:** an optimal solution to the problem contains within it optimal solution to subproblems.

II. Algorithm design techniques

Greedy Algorithms

- Two key elements:
  - **The greedy-choice property:** a globally optimal solution can be arrived at by making a locally optimal (greedy) choice.
  - **The optimal substructure property:** an optimal solution to the problem contains within it optimal solution to subproblems.

- Examples (greedy works)
  1. Activity selection
  2. Huffman coding (data compression)
  3. Job scheduling – minimizing the average completion time
  4. MST (a graph algorithm)

II. Algorithm design techniques

Greedy Algorithms

- Two key elements:
  - **The greedy-choice property:** a globally optimal solution can be arrived at by making a locally optimal (greedy) choice.
  - **The optimal substructure property:** an optimal solution to the problem contains within it optimal solution to subproblems.

- Examples (greedy works)
  1. Activity selection
  2. Huffman coding (data compression)
  3. Job scheduling – minimizing the average completion time
  4. MST (a graph algorithm)
- Examples that greedy does not works
  1. Knapsack problem
  2. Money changing

II. Algorithm design techniques

Dynamic Programming

[2] Unlike divide-and-conquer, where subproblems are independent.

II. Algorithm design techniques

Dynamic Programming

- Three key elements:
  - **The optimal substructure:** the optimal solution to the problem contains optimal solutions to subproblems $\Rightarrow$ "recursion".
  - **Overlapping subproblems:** There are few subproblems in total, and many recurring instances of each.[2]
  - **Memoization:** after computing solutions to subproblems, store in table, subsequent calls do table lookup.

[2] Unlike divide-and-conquer, where subproblems are independent.

II. Algorithm design techniques

Dynamic Programming

- Three key elements:
  - **The optimal substructure:** the optimal solution to the problem contains optimal solutions to subproblems $\Rightarrow$ "recursion".
  - **Overlapping subproblems:** There are few subproblems in total, and many recurring instances of each.[2]
  - **Memoization:** after computing solutions to subproblems, store in table, subsequent calls do table lookup.

- Examples:
  1. Rod cutting
  2. Matrix-chain multiplication
  3. Longest common subsequence/substring
  4. Edit distance
  5. Knapsack problem
  6. Change-making problem

[2] Unlike divide-and-conquer, where subproblems are independent.

III. Graph algorithms

- Elementrary graph algorithms

III. Graph algorithms

- Elementrary graph algorithms
  - Breadth-first search (BFS):
    I/O, FIFO queue, complexity
  - Depth-first search (DFS):
    I/O, LIFO stack, complexity

III. Graph algorithms

- Elementrary graph algorithms
  - Breadth-first search (BFS):
    I/O, FIFO queue, complexity
  - Depth-first search (DFS):
    I/O, LIFO stack, complexity

- Applications of BFS and DFS
  1. sorting a dag
  2. determining cycle
  3. finding a sink
  4. finding connected components

Make sure to know how to precisely (correctly) *illustrate BFS and DFS*

III Graph algorithms

- Minimum Spanning Tree (MST)
  - Prim's algorithm:
    priority queue, complexity
  - Kruskal's algorithm: disjoint-set, complexity
    priority queue, complexity

Make sure to know how to precisely (correctly) *illustrate Prim and Kruskal algorithms.*

III Graph algorithms

- Shortest paths (single-source)
  - Bellman-Ford algorithm
    dynamic programming-like, multiple passes
  - Dijkstra's algorithm
    greedy, priority queue
  - Bellman-Ford algorithm for DAG
    only need a single pass after TS

Make sure to know how to precisely (correctly) *illustrate these algorithms.*

IV. NP-completeness – *a brief introduction*

1. Tractable and intractable problems
2. Optimization problem versus decision problem
3. Polynomial transformation and reduction
4. Formal definitions: P, NP, NP-complete, NP-hard

IV. NP-completeness – *a brief introduction*

1. Tractable and intractable problems
2. Optimization problem versus decision problem
3. Polynomial transformation and reduction
4. Formal definitions: P, NP, NP-complete, NP-hard
5. Examples of NPC problems:
   - 5.1 Circuit-satisfiability (SAT),
   - 5.2 Graph-coloring,
   - 5.3 Hamiltonian-cycle (HC),
   - 5.4 Traveling-salesperson-problem (TSP),
   - 5.5 Knapsack-problem,
   - 5.6 Prime-testing,
   - 5.7 Subset-sum,
   - 5.8 Set-partition,
   - 5.9 Bin-packing,
   - 5.10 Vertex-cover,
   - 5.11 Clique problem.

IV. NP-completeness – brief introduction

6. How to prove a problem is NP-completeness
   - Proof structure and logic

     (1) ...

     (2) Step A: ...

     Step B: ...

IV. NP-completeness – brief introduction

6. How to prove a problem is NP-completeness
   - Proof structure and logic

     (1) ...
     (2) Step A: ...
     Step B: ...

   - Examples:

     6.1 Directed HC $\leq_T$ Undirected HC
     6.2 3-Color $\leq_T$ 4-Color

Good luck. Finish Strong