CS483 Spring 2009 Review 02

Instructor: Jyh-Ming Lien

Algorithms by Dasgupta *et al.*: Chapter 1 $\sim$ Chapter 8

1 Overview

This document is written to help you prepare the coming exam. Do not use this document as your cheat sheet for the the final exam.

So far we have covered Chapters 1 $\sim$ 8 of our textbook. In this final exam, we will focus on the second half of the book, i.e., Chapters 6 to 8. However you should also review Chapters 1 $\sim$ 5. (One or two questions may be from the first half of the textbook.)

1.1 Algorithm Design Strategies

- Theoretical analysis of algorithm: To analyze an algorithm, we should ask:
  - What is the input size?
  - What is the basic operation?
  - What is the efficiency class when the input size is infinitely large?
- Asymptotic notations for a function $f(n)$ (which normally represents the number of basic step of an algorithm with input size n)
  - $O(f(n))$ means $f(n)$ is an upper bound
  - $\Omega(f(n))$ means $f(n)$ is a lower bound
  - $\Theta(f(n))$ means $f(n)$ is a tight bound
- Compare order of growth: $\lim_{n\to\infty} \frac{f(n)}{g(n)}$
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ means $f(n) = O(g(n))$
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} =$ constant means $f(n) = \Theta(g(n))$
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ means $f(n) = \Omega(g(n))$
- Numerical algorithms
  - Integer addition, multiplication, division
  - Modular arithmetic
    - addition, multiplication
    - compute $x^y \% n$ efficiently
    - greatest common divisor
    - division is computed using Euclid's GCD method
  - Prime number testing (Little theorem)
  - Prime number generation (Brute force)
  - RSA
- Divide-n-Conquer (divide into sub-problems and solve all sub-problems)
  - quicksort, merge sort, integer and matrix multiplications,

- – Selection (Medians, k-the smallest element, binary search)
- – FFT

- Graph algorithms
  - Depth-/Breadth-first search and their applications
    - what data structures are used?
  - Directed graphs (cycles, types of edges)
  - Directed acyclic graph (dag) and linearization (topological sort)
  - (Strongly) connected components
    - undirected graph
    - directed graph
  - Single-source shortest paths
    - graphs without weight
    - weight graphs with positive weights
      - what data structures are used?
    - weight graphs with negative weights
    - DAG
- Greedy algorithm
  - Decision is made based on current (or local) state only
  - usually done iteratively (with partial solution)
  - used for solving optimization problems
  - Example: making change, minimum spanning tree, single-source shortest paths, Huffman coding
- Dynamic programming
  - solve and **store** results from sub-problems
  - Reuse the results from the sub-problems
  - Do not confuse dynamic programming with divide and conquer
  - "programming" means optimization. Therefore, dynamic programming is usually used to solve optimization problems (problems that ask you to find maximum value or minimum cost).
  - When you are asked to solve to problem using dynamic programming, the first thing you do is to think and define how such a problem can be solved using **sub-problems**. The second thing you should think is how to re-use the solutions of the sub-problems.
  - Example: Fibonacci number, transitive closure, all-pairs shortest-paths, knapsack, matrix sequence multiplication
- Linear programming
  - Representing problems using linear programs
  - Network flow
    - Max flow problem
    - Min cut problem
    - Bipartite matching
  - Duality
    - Game theory (min-max theorem)
  - Simplex

    - Start with a (compete) solution
    - Improve the quality of the solution in each iteration
    - Until no better solution can be obtained
    - used for solving optimization problems

- NP completeness
  - Definition of search problems
  - Classical search problems (some hard, some easy)
  - Complexity classes
    - P (solvable in polynomial time), NP (checkable in polynomial time), NP-hard (all problems in NP can be reduced to any problem in NP hard), NP-complete (NP and NP-hard)
    - Problems in P (MST), in NP (SAT), in NP hard (halting problem), in NP-complete (SAT), in NP but neither NP-hard nor P (graph isomorphism)
  - Reduction
    - Used in solving problem, establishing lower bound, proving NP-hardness
    - Problem $A \rightarrow$ Problem B (reduce A to B), the complexity of problem A is known, we want to show the complexity of B
    - Given an instance I_A of problem A modify I_A as an input of problem S
  - Example: Vertex Cover, Independent Set, Clique, TSP, Knapsack, Rudrata cycle, SAT, ...

2 Problems and Strategies We Learned

The following table listed a set of problems that we have learned from Chapter 6 to Chapter 8.

Category	Problem	Complexity	Strategy
Graph	transitive closure		
	all-pairs shortest-paths		
	maximum flow		
	bipartite matching		
	Rudrata circuit/path		
	traveling salesmen		
	vertex cover		
	clique		
	independent set		
Other optimization	knapsack		
	edit distance		
	longest increasing sequence		
	matrix sequence mult		
	zero-sum games		
	bandwidth allocation		
	task assignment		
search	SAT		
	3SAT		
	subset sum		
	Integer LP		
	ZOE		
other	Halting problem		
	Factoring		

3 Sample Questions

1. **(***10 points***)** Answer and provide explanations to the following questions.

a. (*5 pts***)** Compare Dijsktra's algorithm and Prim's algorithm.

- What are the problems that these two algorithms try to solve?

- What kind of strategies are used by these two algorithms (e.g., divide&conquer, space-time, iterative, or ...)?

- What functions are used to order unvisited vertiecs?

- What kind of data structure can be used to maintain unvisited vertiecs?

- What are their time complexities?

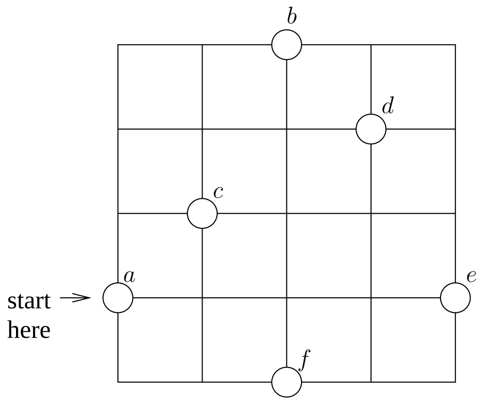
2. (*10 points*) Problem reduction.

a. (*7 pts*) Reduce the following 3 SAT problem to a clique problem.

$$(x \vee y \vee \bar{z}) \wedge (\bar{w} \vee z) \wedge (\bar{y} \vee w) \wedge (\bar{x} \vee \bar{z})$$

b. (*3 pts*) Solve the 3 SAT problem above using the solution of the clique problem. (**Hint**: Find a clique with 4 vertices.)

3. (*15 points*) Given a robot arm and a 4×4 regular grid marked with five circles as shown in the figure below. The robot arm can only move on the lines of the grid. Find (or approximate) the shortest path for the robot arm to drill holes on these five circles using:



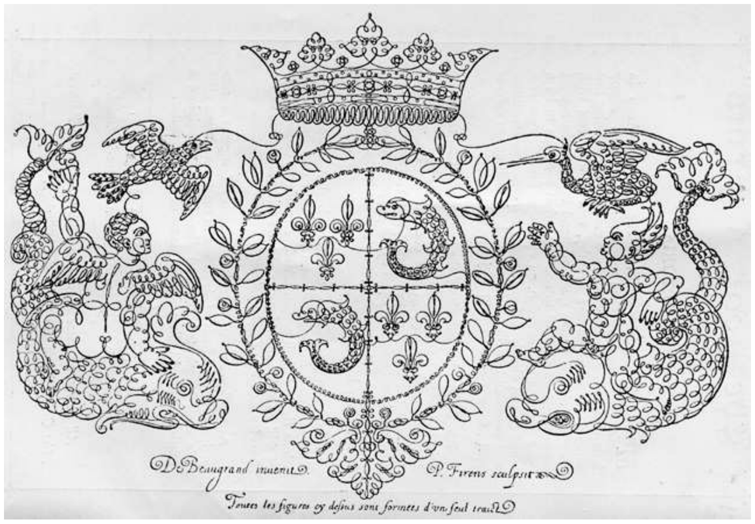
a. (*5 pts*) Greedy algorithm

b. (*5 pts*) Kruskal's algorithm

c. (*5 pts*) Dynamic programming

4. (*15 points*) Answer the following questions.

a. (*3 pts*) If a stranger gives you a figure (such as the one shown below) and asks you to trace the lines of the figure without repeating any lines or lifting your pen, after taking CS483, you can simple say: This is just a ______________________ problem and it has been solved more than two hundred years ago!

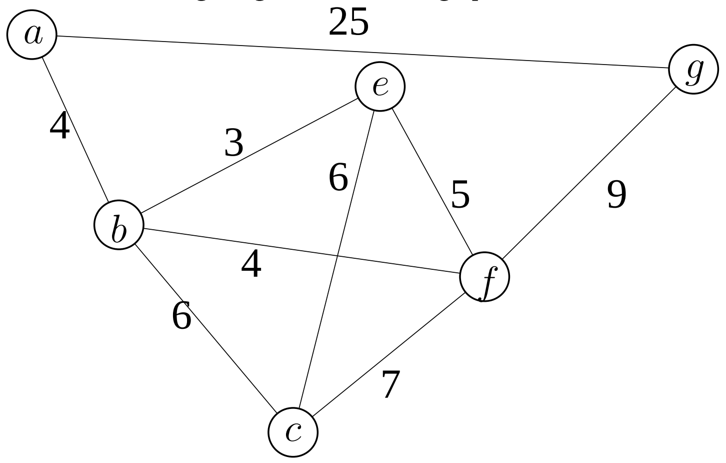


(Lineography created by Jean de Beaugrand 1584-1640)

b. (*9 pts***)** Sketch an algorithm to find the shortest path (cycle) of such paths (cycles) from a graph.

c. (*3 pts***)** What is the time complexity of your algorithm?

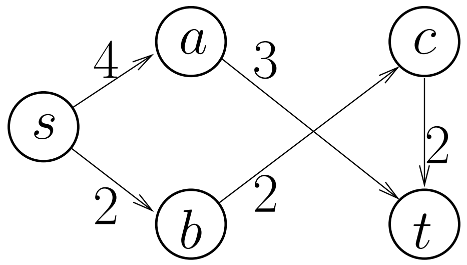
5. (*10 points***)** Consider the following weighted undirected graph.



a. (*5 pts***)** Compute the shortest paths from the node a using Dijkstra's algorithm. You must show steps to earn points.

b. (*5 pts***)** Compute the all-pairs shortest paths using (Warshall-)Floyd's algorithm. You must show steps to earn points.

6. (*15 points***)** Consider the following graph.



a. (*8 pts***)** Show that you can reduce the max-flow problem with the source s and the sink t to a linear programming problem.

b. (*7 pts***)** Solve your linear programming problem using the Simplex algorithm.