CS340 Analysis of Algorithms

Introduction

Dianna Xu

1

Prerequisites

- Basic programming
  - pointers, structures/classes, recursion
- Discrete Math
  - induction, recurrence relations, counting, probability and graph theory
- Understanding of basic data structures
  - lists, stacks, queues, trees, graphs and heaps
- Knowledge of basic sorting algorithms
- Ability to do a time analysis (loops and DS operations)

2

Learning Resources

- Course website
  - www.cs.brynmawr.edu/cs340
- Moodle
- CS data server
  - account: ddiaz1@brynmawr.edu
  - goldengate.cs.brynmawr.edu
- Slack
  - If I am near a computer: near-instant responses
  - otherwise: slower responses, but still faster than email

3

Policies

- Be on time and come to all classes and labs
- Actively participate in the discussions
- Expect 24-hour response time for emails. Longer on weekends
- No late work, no late projects
- Extensions should be requested at least 24-hours ahead of deadline

5

Logistics

- Syllabus
- Lab: Wednesdays
  - 2:40pm-4:00pm
- Textbook is required (and useful!)
- Learn LaTeX

6

Study Groups and Discussions

- Discussions are vital
- Study groups
  - formation is required

7

Collaboration policy

- Allowed sources:
  - your classmates in this class – work together, then write it up separately
  - the textbook
  - me
  - the TAs
  - your notes or textbooks from previous classes
- Disallowed sources for solution lookup:
  - the internet/AI tools
  - students in this class from previous years
  - anything else
- Cite anything that contributed to your solutions

8

What is an Algorithm?

- Any well-defined computational procedure that takes some values as input and produces some values as output.
- A step-by-step instruction for solving a computational problem
- Enough details so that a competent computer scientist can implement without questions

9

Why Algorithms?

- Programming is a remarkably complex task
  - structural complexity
  - large data sets and complex data structures
  - complex problems that requires efficient algorithms
- There are standard ways of approaching algorithm design (patterns)
- High-level tools for run-time analysis

10

Nothing New Under the Sun

- If we can identify broad categories of algorithms and solve them, then when we see those problems again, they're easy to solve.
- If we encounter a new problem, it's probably similar to a solved one, or a few solved problems put together.

11

Algorithm Design

- Algorithms are mathematical objects
  - Correctness
  - Efficiency
    - worst-case complexity
    - average-case complexity
- Algorithms
  - provide solutions
  - provide a language to express problems

13

Stable Matching Problem

- Match employers/schools to applicants so that no one wants to switch
  - What's the input?
  - What's the goal?
- What's a switch?
  - employer e has been matched with applicant a, but prefers applicant a' AND
  - applicant a' has been matched with employer e', but prefers employer e

16

Example

Employers			Applicants		
Google(G)	Intel(I)	Apple(A)	Kate(K)	Clara(C)	Lisa(L)
K	K	L	A	A	G
C	L	K	G	I	I
L	C	C	I	G	A

- Assignments
  - G-C, I-L, A-K
  - G-L, I-K, A-C
  - G-L, I-C, A-K

17

Example

Employers			Applicants		
Google(G)	Intel(I)	Apple(A)	Kate(K)	Clara(C)	Lisa(L)
K	K	L	A	A	G
C	L	K	G	I	I
L	C	C	I	G	A

- Assignments
  - G-C, I-L, A-K G[K]-C[A,I], I[K]-L[G], A[L]-K[]
  - G-L, I-K, A-C G[K,C]-L[], I[]-K[A,G], A[L,K]-C[]
  - G-L, I-C, A-K G[K,C]-L[], I[K,L]-C[A], A[L]-K[]

18

Steps to finding an algorithm

1. Construct a good example (not too small).
2. Solve the example and check your solution (by hand, without worrying about the algorithm)
3. Think about how you solved and/or checked the problem and how you can use that to solve a general instance.
4. Formalize an algorithm (might have to formalize the problem too)
5. Construct a new and somehow different example and run your algorithm on it and check the solution.
6. Repeat until you are confident it works.

Definitions

- Good definitions are vital
  - the problems usually come with definitions. Use them, do not make up new ones unnecessarily
  - learn to write definitions like the ones you were given
- Outline/formulate your inputs and outputs with the definitions

20

Matching

- Given a pair of sets X and Y, a *matching*, is a collection of pairs (x, y), where $x \in X$ and $y \in Y$, and each element of X and Y appears in at most one pair.
- A matching is *perfect* if every element of X and Y occurs in some pair.

21

Stable Matching

- Given sets X and Y of equal size and a preference ordering for each element of each set, a perfect matching is *stable* if there is no pair (x, y) that is *not* in the matching where x prefers y to its current match and y prefers x to its current match.
- An *unstable* matching involves at least two pairs

22

Input/Output

- Input
  - A set of employers: E
  - A set of applicants: A
  - $|E| = |A| = n$
  - $2n$ preference lists, each of size n
  - How are the preference lists stored?
    - linked lists? arrays? hash maps? why?
  - M, a list/set of n pairs (the matching)
- Output: yes/no

23

Challenge

- Write up the algorithm we just discussed that decides if a matching is stable
- Even if you have a pretty good idea on how it's supposed to work, writing it up cleanly and succinctly is still not an easy task!

24