Welcome to Analysis of Algorithms (CS483 - 001)

Amarda Shehu

Spring 2017

Class Information

Tentative Syllabus

Date	Topic	Chapters	Assignment
Jan 23	Course Overview and Introduction to Analysis of Algorithms	C1-3, D0, D2.0-.3 [pdf]	Self-eval. quizz
Jan 30	Algorithm Analysis	C3-5, D2.0-.3 [pdf]	Quizz
Sorting			
Feb 6	Algorithm Analysis	C3-5, D2.0-.3 [pdf]	Quizz
Feb 13	Heapsort, Quicksort, Linear-Time Sorting	C6-8, D2.3 [pdf]	Quizz
Feb 20	Order Statistics	C9, C11, D1.5, D2.4 [pdf]	Quizz, Hw1 Out [pdf]
Data Structures for Searching and Mapping			
Feb 27	Hash Tables	C11, C12, D2.4 [pdf]	Hw1 Due
Mar 06	Balanced Search Trees, Binomial Heaps	C12-C13, C19 [pdf]	Quizz
Optimization and Advanced Analysis			
Mar 20	Dynamic Programming, Greedy Algorithms	C15-16, D5-6 [pdf]	Quizz
Mar 27	**Exam 1**		Exam 1
Graph Algorithms			
Apr 03	Amortized Analysis, Graph Representation	C17, C22, D3, D6 [pdf]	Hw2 Out[pdf]
Apr 10	Elementary Graph Algorithms, Applications of DSF	C22-23, D3, D4.1-.3[pdf]	Quizz
Apr 17	Topological Sorting, SCCs, Minimum Spanning Trees	C23-24, D4.4-.7 [pdf]	Quizz
Apr 24	All Pairs Shortest Paths and Maximum Flow	C25-26, D4.4-.7 [pdf]	Hw2 Due
Complexity Theory			
May 01	NP-completeness	C34, D8 [pdf]	Quizz
May 08	**Exam 2**	Robinson Hall B113	4:30 pm -- 7:15 pm

Instructor: Amarda Shehu

Office: ENG #4452
Email: amarda AT gmu.edu
Web: cs.gmu.edu/∼ashehu

CS483 Hours

Class: MW 9:00 - 10:15 am
Place: Robinson Hall A111
Office Hours: MW 1:30 - 2:30 pm

TA: Xiaosheng Li

Email: xli22 AT gmu.edu
ENG#5321, W 3:00 - 5:00 pm

Why are we Here?

- In *Calculation with Hindu Numerals*, 825 A.D., Muhammad ibn Musa al-Khwarizmi introduced Indian decimal system
- The book was translated into latin in 12^{th} century as *Algoritmi de numero Indorum*
- *algorithm* was introduced to refer to a procedure for calculations with numbers
- Short answer: We are here to design and analyze algorithms - procedures to solve useful problems

Figure: Soviet stamp for al-Khwarizmi's 1200^{th} birthday. ©wikipedia.

Seach Engines: Google, Yahoo, Ask

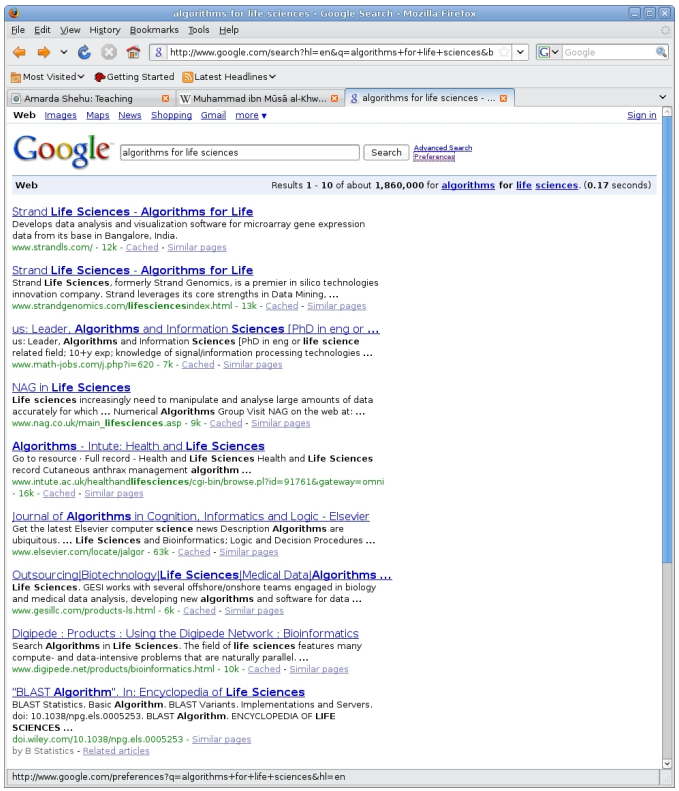

Figure: Searched for *algorithms for life sciences* in text in the web.

- Pattern matching algorithms and information searching algorithms are fundamental to our ability to parse through an overwhelming amount of information
- Google was founded on the ability of two Stanford University Ph.D. students, Sergey Brin and Larry Page, to design a fast information searching algorithm, *BackRub*. They quit school after that.

Orientation Software: Google maps, GPS navigators

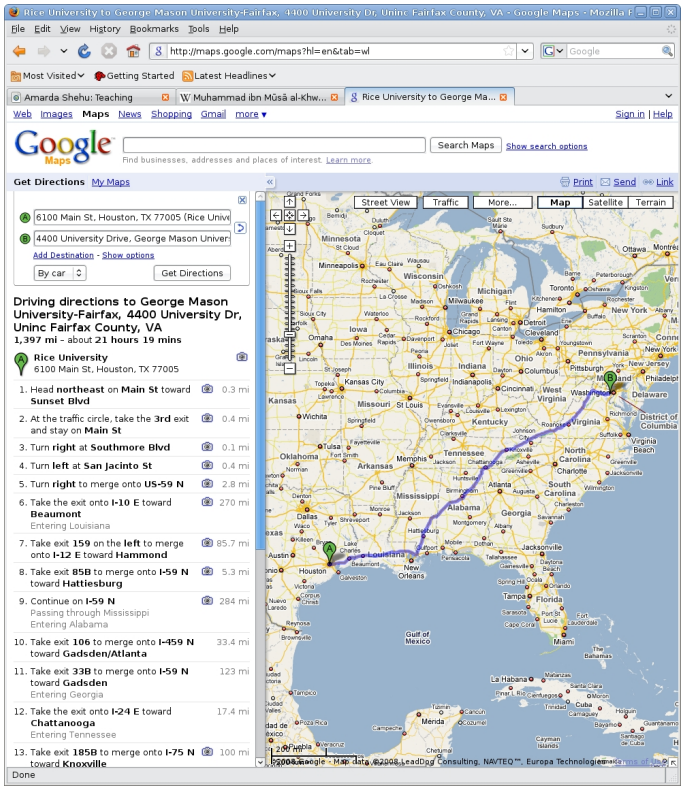

Figure: Output of a path finding algorithm.

- Path from Rice University, Houston, Texas to George Mason University, Fairfax, Virginia
- Path finding algorithms can be found in portable GPS navigators
- Most versions of the algorithm work with a static map (static conditions on the ground)

Exploration, Search and Rescue, and Motion Planning

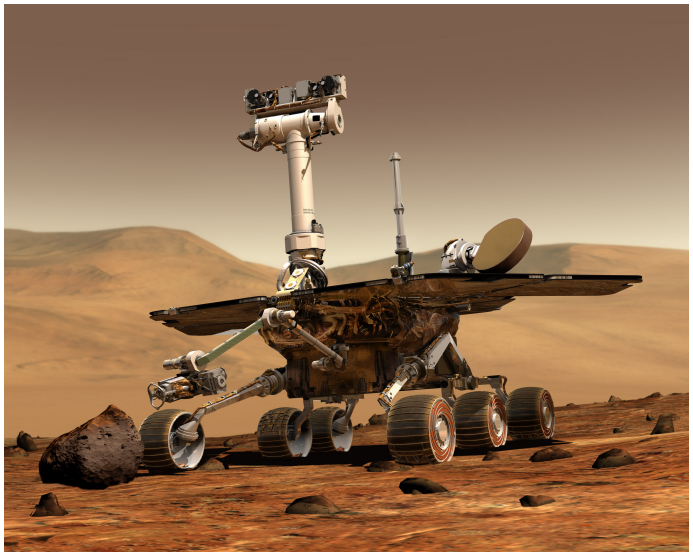

Figure: Ron Li and his research team are developing algorithms to help the rovers, Spirit and Opportunity, to navigate and find a safe path to a winter resting area. ©NASA.

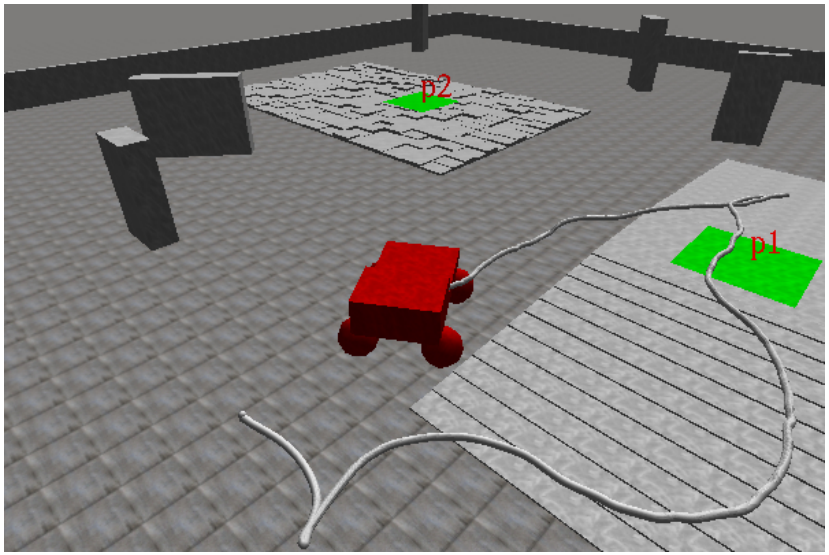

Figure: Erion Plaku at Rice University is developing algorithms that plan paths for car-like robots in cluttered environments. ©E. Plaku.

Simulating Molecular Properties for Drug Design

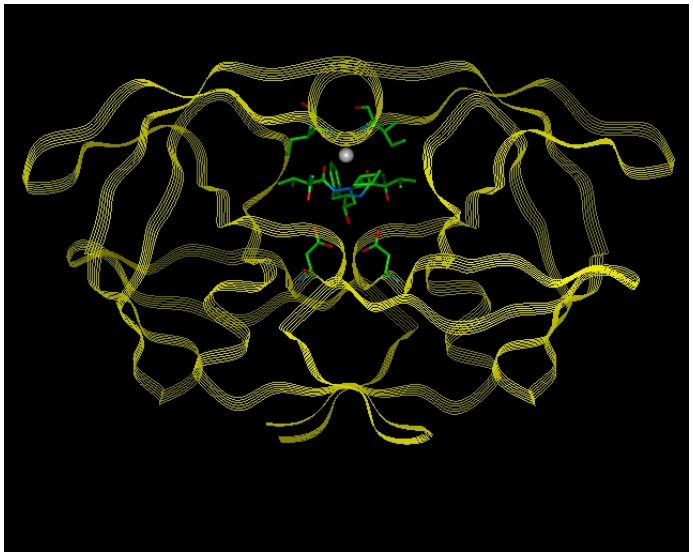

Figure: Successful docking of HIV protease with a small inhibitor ligand. ©A. R. Leach.

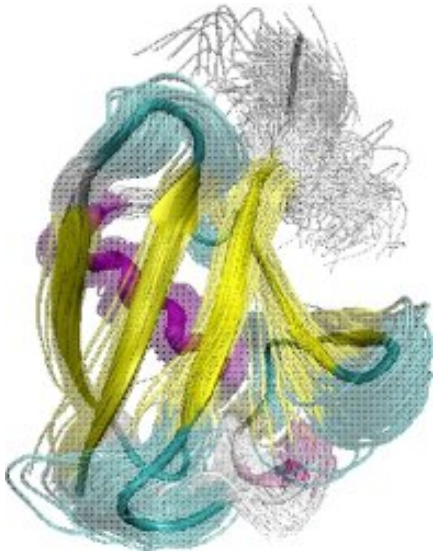

Figure: Simulating the ability of proteins like ubiquitin to change shape as needed to accommodate and dock with different partner molecules. ©A. Shehu.

What is an Algorithm?

- Recipe, computational procedure that transforms input into output, tool to solve well-defined problems, sequence of instructions

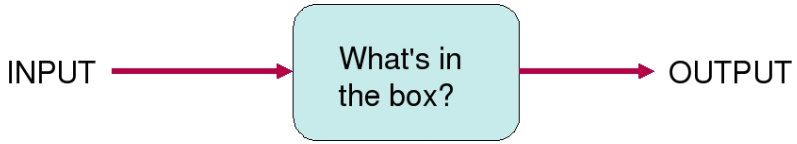

Figure:

- An algorithm is a finite sequence of unambiguous instructions for completing a task, that:

What is an Algorithm?

- Recipe, computational procedure that transforms input into output, tool to solve well-defined problems, sequence of instructions

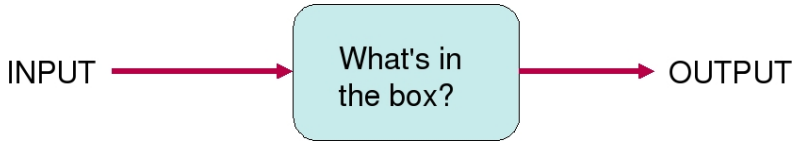

Figure:

- An algorithm is a finite sequence of unambiguous instructions for completing a task, that:
  - when given an initial state (≥ 0 inputs),
  - proceeds through a well-defined series of successive states,
  - eventually terminating in an end-state ($\geq$ outputs)

What is an Algorithm?

- Recipe, computational procedure that transforms input into output, tool to solve well-defined problems, sequence of instructions

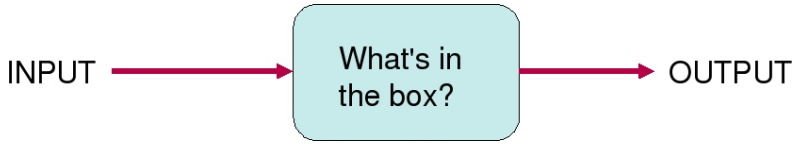

Figure:

- An algorithm is a finite sequence of unambiguous instructions for completing a task, that:
  - when given an initial state (≥ 0 inputs),
  - proceeds through a well-defined series of successive states,
  - eventually terminating in an end-state ($\geq$ outputs)
- Examples of algorithms?

What is an Algorithm?

- Recipe, computational procedure that transforms input into output, tool to solve well-defined problems, sequence of instructions

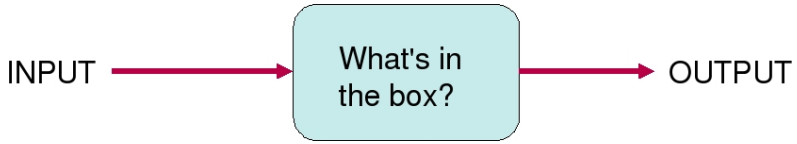

Figure:

- An algorithm is a finite sequence of unambiguous instructions for completing a task, that:
  - when given an initial state (≥ 0 inputs),
  - proceeds through a well-defined series of successive states,
  - eventually terminating in an end-state ($\geq$ outputs)
- Examples of algorithms?

Why do we Write Algorithms?

Given a real problem:

- Model it as a well-defined computational problem
- Is the solution to the problem *computable*?

Why do we Write Algorithms?

Given a real problem:

- Model it as a well-defined computational problem
- Is the solution to the problem *computable*?
- If it is, does there exist an *efficient* algorithm for it?

Why do we Write Algorithms?

Given a real problem:

- Model it as a well-defined computational problem
- Is the solution to the problem *computable*?
- If it is, does there exist an *efficient* algorithm for it?
- How do we come up with an efficient algorithm?

Why do we Write Algorithms?

Given a real problem:

- Model it as a well-defined computational problem
- Is the solution to the problem *computable*?
- If it is, does there exist an *efficient* algorithm for it?
- How do we come up with an efficient algorithm?

How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?

How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?

How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?

How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?
    - Less space? Storage Concerns?
    - Shorter running time? Faster is always better.

How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?
    - Less space? Storage Concerns?
    - Shorter running time? Faster is always better.
  - Can we make the algorithm simpler?

How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?
    - Less space? Storage Concerns?
    - Shorter running time? Faster is always better.
  - Can we make the algorithm simpler?
    - Use a simpler data structure?
    - Allow developers to extend and generalize?
    - Aesthetics concerns?

How do we Write Algorithms?

- Are there any useful paradigms from which we can choose?
- How do we know the algorithm we have written is correct?
- Can we improve the algorithm? On what do we improve?
  - Efficiency?
    - Less space? Storage Concerns?
    - Shorter running time? Faster is always better.
  - Can we make the algorithm simpler?
    - Use a simpler data structure?
    - Allow developers to extend and generalize?
    - Aesthetics concerns?

Paradigms we Will See in this Class

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Space and time tradeoffs
- Dynamic Programming
- Greedy Approach
- Iterative improvement
- Backtracking
- Branch and bound