CS 380
ALGORITHM DESIGN
AND ANALYSIS

Lecture 1: Course Introduction and Motivation
Text Reference: Chapters 1, 2

Syllabus
- Book
- Schedule
- Grading: Assignments/Projects/Exams/Quizzes
- Policies
  - Late Policy
  - Grade Complaints
- Website: zeus.cs.pacificu.edu/lanec/cs380s17
Overview

- Topics:
  - Data structures
  - Complexity classes (Big Oh, etc)
  - Sorting (insertion sort, mergesort, quicksort)
  - Searching (binary search trees, red-black trees, hash tables)
  - Graphs (depth-first, breadth-first, spanning trees, shortest paths)
  - Strings (radix sorting, substring search)
  - Proofs

- Methods
  - Divide and conquer
  - Greedy algorithms
  - Backtracking
  - Dynamic Programming

What is an Algorithm?

- A sequence of computational steps that transforms the input into the desired output.
- To be interesting, an algorithm has to solve a general, specified problem. An algorithmic problem is specified by describing the set of instances that it must work on and the desired properties of the output.
History of Algorithms

- 1600 BC: integer factorization - Babylonians
- 300 BC: Euclid’s algorithm
- 1400: Substitution Ciphers
- 1936: Turing machine
- 1942: Fast Fourier Transform
- 1945: Merge Sort: John von Neumann
- 1952: Huffman Coding
- 1956: Bellman–Ford
- 1956: Kruskal’s algorithm
- 1959: Dijkstra’s algorithm
- 1960: Karatsuba multiplication
- 1962: Quicksort
- 1964: Heapsort
- 1968: A* algorithm
- 1972: Red-black/B trees
- 1973: RSA encryption
- 1998: Google’s PageRank
- 1998: rsynch

Why study algorithms?

- Internet: Google PageRank, Google maps, packet routing
- Biology: Human Genome Project
- Social Networks: advertisements, data mining
- File Compression: MP3, JPG, etc.
- Finance: Trading algorithms (must be FAST!)
- Business: Inventory management, flight scheduling, recommendation systems (Amazon, Netflix)
- Computer Graphics: CGI, video games
- Security: Encryption, RSA, MD5, SHA-1
Internet: Google PageRank

THE $25,000,000,000* EIGENVECTOR
THE LINEAR ALGEBRA BEHIND GOOGLE

KURT BRYAN AND TANYA LEISE

Abstract. Google's success derives in large part from its PageRank algorithm, which ranks the importance of webpages according to an eigenvector of a weighted link matrix. Analysis of the PageRank formula provides a wonderful applied topic for a linear algebra course. Instructors may assign this article as a project to more advanced students, or spend one or two lectures presenting the material with assigned homework from the exercises. This material also complements the discussion of Markov chains in matrix algebra. Maple and Mathematica files supporting this material can be found at www.rose-hulman.edu/~bryan.

Key words. linear algebra, PageRank, eigenvector, stochastic matrix

AMS subject classifications. 15-01, 15A18, 15A51

Web Page URL: http://zeus.cs.pacificu.edu/lanec/index.html

The Page Rank: 0/10

(the page rank value is 0 from 10 possible points)

Human Genome Project

Learning About Huntington's Disease

- What do we know about heredity and Huntington's disease?
- Is there a test for Huntington's disease?
- NIH Clinical Research on Huntington's Disease
- Additional Resources for Huntington's Disease Information

What do we know about heredity and Huntington's disease?

Huntington's disease (HD) is an inherited neurological disease causing involuntary movements, severe emotional disturbance and cognitive decline. In the United States alone, about 30,000 people have HD. In addition, 35,000 people exhibit some symptoms and 75,000 people carry the abnormal gene that will cause them to develop the disease. There is no cure for this fatal disease.

A single abnormally long gene produces HD. In 1993, scientists finally isolated the HD gene on chromosome 4. The gene codes for production of a protein called "huntingtin," whose function is still unknown. But the defective version of the gene has abnormal repeats of a three-base sequence. "CAG." In the normal huntingtin gene, this sequence is repeated between 11 and 29 times. In the mutant gene, the repeated occurs over and over again, from 40 times to more than 80.

This defect causes the resulting huntingtin protein to be malformed, prone to clumping in the brain and causing the death of nearby nerve cells. Cells of the basal ganglia, a brain area responsible for coordinating movement, and of the cortex, which controls thought, perception and memory, are most often affected.
Human Genome Project

- Following subsequence has 7 ‘CAG’ repeats:
  - TTTTCTTTTA CAGTTTTCAGT TTTTCAGCAGTTTTTTAGTTTTGTTTCTCAGTTTTGGCT

<table>
<thead>
<tr>
<th>Repeats</th>
<th>Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 9</td>
<td>not human</td>
</tr>
<tr>
<td>11 - 29</td>
<td>normal</td>
</tr>
<tr>
<td>30 - 39</td>
<td>high risk</td>
</tr>
<tr>
<td>40 - 180</td>
<td>Huntington’s disease</td>
</tr>
<tr>
<td>181 - ...</td>
<td>not human</td>
</tr>
</tbody>
</table>

- Note: Human genome has approximately 4 billion base pairs, and Chromosome 4 has approximately 190 million base pairs.

Finance: Algorithmic Trading

Mysterious Algorithm Causes 4% of Trading Activity

4% of the total trading activity last week was caused by one algorithm, as reported by Nanex, a market data firm. According to CNBC, as told to them by Nanex, the algorithm placed orders once every 25 milliseconds and then cancelled them. The orders were made in bursts of 200, then 400 and then 1000 orders. The algorithm stopped placing orders on Friday at 10:30 am.

According to CNBC, this is a part of the high-frequency trading firms to test the market. The objective of such programs is to mess up the system so it slows down the quote feed to others and allows the computer traders to gain a money-making arbitrage opportunity.
Real quantitative trading operations need practical algorithmic edge. They're not looking for comp sci geeks that know 20 languages and all the dark corners of computing languages. We only need people that can produce algorithmic trading edge. Of course to do that you need experience, ability to understand and create algorithms, relevant languages, strong understanding of statistics, and some combination of signal processing, data mining, and/or linear algebra. Any applicant able to demonstrate a true algorithmic trading edge, or the ability to create and/or understand such edge can get a job, internship or at least some kind of research collaboration.
Recommendation Systems

Computer Graphics: Matrix Multiplication

https://www.youtube.com/watch?v=SMAnIPTmAwE
Video Games: Scene Creation

GTA V - Graphics Study

The GTA V series has come a long way since the first game came out back in 1997. About 7 years ago, Rockstar released GTA V. The game was an instant success, selling 11 million units over the first 24 hours and recently reaching 5 million units.

Having played it on PS3 I was quite impressed by the level of polish and the technical quality of the game.

Nothing kills immersion more than a loading screen; in GTA V you can play for hours, drive hundreds of kilometers into a huge open world without a single interruption. Considering the heavy processing of assets going on and the types of the PS4 (2 GB of RAM and 256 MB of video memory) (unavoidable) the game doesn’t crash after 20 minutes, it’s a real technical prowess.

Here I will be talking about the PC version in DirectX 11 mode, which uses up several GBs of memory from both the RAM and the GPU. Even if optimizations are PC-specific I believe many can apply to the PS4 and for a certain extent the PS5.

- Part 1: Introduction Frame
- Part 2: 100 and Reflection
- Part 3: Post-Game

http://www.adriancourreges.com/blog/2015/11/02/gta-v-graphics-study/

Image Compression: SVD

Math 306 — Linear Algebra

SVD

Singular Value Decomposition (SVD):

Any \( m \times n \) matrix \( A \) of rank \( r \) can be written as

\[
A = U \Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T
\]

where \( U \) is an orthonormal \( m \times m \) matrix, \( \Sigma \) is an \( m \times n \) matrix, and \( V \) is an orthonormal \( n \times n \) matrix such that

1. the columns of \( U \) are the orthonormal eigenvectors of \( AA^T \)
2. the columns of \( V \) are the orthonormal eigenvectors of \( A^T A \)
3. the diagonal entries of \( \Sigma \) are \( \sigma_i = \sqrt{\lambda_i} \) where \( \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_r > 0 \) are the positive eigenvalues for \( A^T A \) (or \( AA^T \)).

Furthermore, the equation \( AV = U \Sigma \) implies that we must order the columns \( u_i \) and \( v_i \) such that \( Av_i = \sigma_i u_i \). In particular, this implies that the \( i \)th columns of each matrix must correspond to the \( i \)th singular value in the matrix \( \Sigma \).
Image compression: SVD, cont.

rank=2  
rank=5  
rank=10  
rank=50  
original

Pathfinding

Towards Fully Autonomous Driving: Systems and Algorithms

Josef Levison, Jake Ackland, Jan Bröcker, Jennifer Dohm, David Hild, Steven Kamerer, J. Zico Kolter, Dirk Langer, Oliver Prklic, Vaughan Pratt, Michael Sokolowski, Gangemi Stanz, David Stavens, Alvis Tkachenko, Marta Wiering, and Sebastian Thrun

Abstract—In order to achieve autonomous operation of a vehicle in urban situations, with unpredictable traffic, several other issues need to be addressed. In particular, autonomous path planning and execution are required, as well as a sophisticated sensor technology that is able to provide perception, localization, planning, and control. In addition, a robust vehicle platform with appropriate sensors, computational power, networking, and software infrastructure is required. The perception module is responsible for terrain mapping, object detection, and classification. The localization system must provide a clear measure of position, while deciding and making path decisions in the field, we use our approach at path planning to ensure a smooth and safe navigation of the autonomous vehicle. In this paper, we present an overview of our research toward the goal of designing a robust and efficient autonomous path planning in various urban situations.

1. INTRODUCTION

Path planning for autonomous vehicles in typical urban settings is an important and difficult task. Many notable attempts have been made, and several successful autonomous vehicles have been deployed and are now operating in urban environments. In parallel, over the last several decades, autonomous vehicles have been used to operate autonomous vehicles in public streets, in general, these efforts have been restricted to a subset of the full driving task, e.g., highways, parking only, or in vulnerable only. [2]

In this paper, we describe a variety of algorithmic improvements and validation that our lab has implemented in our system called the Urban Challenge, and shows which are motivated by the goal of safe, efficient, and complex driving in a variety of urban environments. We also provide a detailed description of our autonomous vehicle and some of the components that have been published recently, and as we are presenting a wide variety of ideas here, this paper can serve as a brief overview of our entire system as it has evolved over the last three years.
Pathfinding Algorithms

- [https://qiao.github.io/PathFinding.js/visual/](https://qiao.github.io/PathFinding.js/visual/)

Performance

- Algorithms is the study of computer-program performance
- What is more important than performance in computer programs?
  - 
  - 
  - 
  - 
  - 
  - 
Correctness

• For any algorithm, we must prove that it always returns the desired output for all legal instances of the problem.
• What does this mean for sorting?

Induction and Recursion

• Mathematical induction is a very useful method for proving the correctness of recursive algorithms.
• Recursion and induction are the same basic idea: (1) basis case, (2) general assumption, (3) general case.
Demonstrating Incorrectness

- Searching for counterexamples is the best way to disprove the correctness of a heuristic.
- Failure to find a counterexample to a given algorithm does not mean “it is obvious” that the algorithm is correct.
- Think about all small examples.
- Think about examples with ties on your decision criteria (e.g. pick the nearest point).
- Think about examples with extremes of big and small.

Correctness is Not Obvious!

- Suppose you have a robot arm equipped with a tool, say a soldering iron. To enable the robot arm to do a soldering job we must construct an ordering of the contact points so the robot visits (and solders) the first contact point, then visits the second point, third, and so forth until the job is done.
- Since robots are expensive, we need to find the order which minimizes the time (i.e. travel distance) it takes to assemble the circuit board.
Correctness is Not Obvious!

- You are given the job to program the robot arm. Give an algorithm to find the shortest tour.

Nearest Neighbor Tour

- A very popular solution starts at some point \( p_0 \) and then walks to its nearest neighbour \( p_1 \) first, then repeats from \( p_1 \), etc. until done.
- Pick and visit an initial point \( p_0 \)
  - \( p = p_0 \)
  - \( i = 0 \)
- While there are still unvisited points
  - \( i = i + 1 \)
  - Let \( p_i \) be the closest unvisited point to \( p_{i-1} \)
  - Visit \( p_i \)
- Return to \( p_0 \) from \( p_i \)
Nearest Neighbor Tour

A Correct Algorithm

d = infinity
For each of the n! permutations P(i) of the n points
    if (cost(P(i) <= d) then
        d = cost(P(i)) and Pmin = P(i)
return Pmin

Issue: 10! ~ 3 million
TSP: NP Hard/Complete:
    verifiable: Polynomial time on deterministic TM
    solvable: Polynomial time by a nondeterministic TM
Why not use a supercomputer?

“Everyone knows Moore's Law—a prediction made in 1965 by Intel co-founder Gordon Moore that the density of transistors in integrated circuits would continue to double every 1 to 2 years….in many areas, performance gains due to improvements in algorithms have vastly exceeded even the dramatic performance gains due to increased processor speed.”

- Excerpt from Report to the President and Congress: Designing a Digital Future, December 2010 (page 71).

https://www.whitehouse.gov/sites/default/files/microsites/os tp/pcast-nitrd-report-2010.pdf (No longer available)

Why not use a supercomputer, cont.

- A faster algorithm running on a slower computer will always win for sufficiently large instances
- Usually, problems don’t have to get that large before the faster algorithm wins
Expressing Algorithms

• What are the possible ways to express an algorithm?
  • English
  • Pseudocode
  • Programming Language

• Two common types of algorithm design:
  • Incremental:
    • Iteratively run an algorithm (insert sort, next class)
    • Analyze by counting iterations (as in CS 300)
  • Recursive
    • Divide and conquer (merge sort, following class)
    • Analyze using recurrence equations (three methods to solve)

Examples: Counting Iterations I

```c
DoubleLoop()
{
  int i, j
  for (i=1 to n)
    for (j=1 to n)
      print i*j
}
```

Time Complexity:

```c
Square()
{
  for (i=1;i^2<=n;i++)
    print i
}
```

Time Complexity:
Examples: Counting Iterations II

TimesTwo()
{
    for (i=1; i<=n; i=2*i)
        print i
}

Time Complexity:

Examples: Counting Iterations III

Sum()
{
    int i=1, s=1
    while (s <= n)
    {
        i = i+1
        s = s+i;
        print s;
    }
}

Time Complexity:
Examples: Counting Iterations IV

TripleLoop()
{
    int i, j, k, n
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= i; j++)
        {
            for (k = 1; k <= 100; k++)
            {
                print i*j*k
            }
        }
    }
}

Time Complexity:

Examples: Counting Iterations V

AnotherDouble()
{
    int i, j, n
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j = j+i)
        {
            print i*j
        }
    }
}

Time Complexity:
For Next Time

- Read Chapters 1 and 2 from the book.