Lecture 1: Dijkstra’s algorithm

Ramin Zabih

Some slides from: K. Wayne
Administrivia

• Web site is: https://github.com/cornelltech/CS5112-F18
  – As usual, this is pretty much all you need to know

• Course staff:
  – Instructors: Ramin Zabih (rdz@cs.cornell.edu) & Greg Zecchini (gez3@cornell.edu)
  – TA: Richard Bowen (rsb349@cornell.edu), TBA
  – Consultants/graders: Iris Zhang (wz337@cornell.edu), TBA
Basic information

- CS5112 work is constant but not very time intensive
  - Homework every 2 weeks or so, quiz every week
- 1 prelim 10/25 and final 12/4, both in-class closed book
  - Open book doesn’t actually help in my experience
- Greg will teach the 5 evening clinics 6:30pm-8pm
- Greg and Ramin will lecture, with a few guest lectures
- We are working on getting consultants to help students who don’t have a lot of programming experience
Academic integrity

• Each student is expected to abide by the Cornell University Code of Academic Integrity
  – [https://theuniversityfaculty.cornell.edu/academic-integrity/](https://theuniversityfaculty.cornell.edu/academic-integrity/)

• Any work submitted by a student in this course for academic credit will be the student's own work
  – Exception: you do the homework assignments in groups of two

• We take this seriously. Students have been expelled from Cornell for violations. Copying code is easy to catch.
Today

• Clinic this evening (here!)
• HW1 out tonight, due in 2 weeks
  – Based primarily on today’s lecture and clinic
• First quiz a week from today
• Placement exam available tonight, due in 24 hours
• If you were a CS major, possibly worth your time
Clinics schedule

• Evening sessions to review some of the fundamentals underlying much of this course's content
• Attendance is mandatory
• Schedule (dates final, topics tentative):

  – 8/23  Graphs and Graph Algorithms
  – 8/30  Hashing and Related Data Structures
  – 9/6   Sorting and Searching
  – 9/20  Development tools (UNIX commands, Github, etc.)
  – 10/4  Cloud Development
Course theme: algorithms and applications

- Algorithms are the key tool in CS, but without applications it’s hard to appreciate their importance
- We will focus on 3 key application areas:
  - Cryptocurrency
  - AI (artificial intelligence)
  - AR/VR (augmented/virtual reality)
- BUT this is not a course about those applications
- Application of algorithms is often not obvious!
Lecture Outline

• The shortest path problem
• Dijkstra’s algorithm
• Applications: image editing and pirate grammar
Two very common approaches in CS

• Given a problem where you are searching for a solution:
  – Try everything (exhaustive search)
  – Do what seems best at the moment, repeatedly (greedy algorithms)
• Exhaustive search (almost) never works on serious problems
• Greedy algorithms are widely used
  – Currently famous example: SGD for neural networks
• Note: there are other approaches we will cover
  – Such as dynamic programming
The shortest path problem

- General version: given a graph with edge weights, a starting node $s$ and a target $t$, find shortest path from $s$ to $t$
- Claim: this problem is impossible to solve!
Obvious application of shortest paths: airfare

- Nodes are cities, edges are direct flights, weights are airfare
- What is the **cheapest** way to get from LGA to Ithaca?
  - Presumably you can charter a plane
Fixing the problem definition

- Suppose that there is a flight from Boise to El Paso, and back again, that the airline pays you $1 to fly around
- Further, suppose that you can get to Boise (or El Paso)
- You can make an arbitrary amount of money by just flying back and forth!
- This is a cycle in the graph whose sum of weights is negative
- Easy solution: require positive edge weights
  - Or maybe detect negative cycles?
Not so obvious applications

• Making fake photographs
• Speech recognition/predicting stock prices by DTW
• Pirate grammar!
• Modeling a Cornell student (at end of class)
Making fake photographs

• Sneak preview: will cover this in the AR/VR section of CS5112
• How do we create images like this:

• Given an image, how do you cut out an object from it?
• You don’t want to manually select the pixels
Intelligent scissors

• Idea: shortest paths
  – E.N. Mortensen and W.A. Barrett, Interactive Segmentation with Intelligent Scissors, SIGGRAPH 1995

• Adobe calls this the “Magnetic Lasso”

• Video [here](#)

• More details in November!
Dynamic Time Warping (DTW)
Rules of Pirate grammar

• Pirates always start their sentences with “Barkeep!”
  – 90% of the time they next say “More” (i.e., they order)
  – 10% of the time they next say “Yer a” (i.e., they insult)
  – If they say “More”, they next say:
    • 60% “Of your best”
    • 40% “Of the same”

• Lots more rules, discovered by experts in pirate linguistics

• Question: what sentence is a pirate most likely to say?
Pirate grammar as a graph

"Barkeep!" → "More" (.9) "of your best" (.6) "grog!"

"Yer a" (.1) "of the same" (.4) "wimpy"

"scurvy" "chicken."
Simplified pirate grammar

"Barkeep!" → "More" → "of your best grog!" 0.9

"Barkeep!" → "Yer a" → "of the same wimpy grub." 0.4

"Barkeep!" → "scurvy chicken." 0.1
How to make this into shortest paths?

- On the surface this is not at all obvious
  - Which is why this is worth thinking about carefully
- What we actually need to determine is the probability of any individual sentence
  - Example: “Barkeep! More of your best grog!” = .9 * .6 = .45
- So we look at all paths from the root to a leaf node
  - Each edge has a probability
  - Multiply these together and find the max
- This looks like “find the path where the product of the edges is maximized”, not “find the shortest path from s to t”
Easy part: Add a fake source and sink

- Red links have probability 1
- Now we need to find the “highest product path” from $s$ to $t$
Algebra to the rescue

• We want to maximize the product of edge probabilities
  – Which are numbers between 0 and 1
• Instead we need to minimize the sum of edge weights
• We know that log is monotonic, and \( \log \prod_i p_i = \sum_i \log p_i \)
• Maximize the product of edge probabilities = maximize the sum of log probabilities
  – Which are negative: \( 0 < p_i \leq 1 \Rightarrow \log p_i \leq 0 \)
• Maximizing anything is the same as minimizing its negative
Algebra in action

\[
\log_{10}(0.9) \approx -0.046 \\
\log_{10}(0.1) = -1
\]
Key property of shortest paths

• Suppose the shortest path from $s$ to $t$ goes via $v$
  – I.e., $s \ldots v \ldots t$
  – Otherwise, we would take that “shortcut” instead, and create an even shorter path
• Considering $s - v - t$ paths, only need shortest $s - v$ path
  – Don’t need to try everything!
Shortest paths by increasing budgets

• Here is the basic idea, which we will simply speed up
• Where can you fly from LGA on a $1 budget?
  – Does that get you to Ithaca?
• If it does, you are done
• If not, add $1 to your budget and do it again
• You can think of this as expanding a ball around $s$ until you eventually get to $t$
  – Though we are doing this on a graph
Example

• For $1 can get to \( u \)
• For $2 can also get to \( v \)
• Gray area shows budget at $2
• At $3 we can also get to \( x \) via \( u \)
• Key concepts:
  – Explored nodes: \( \{s, u, v\} \)
  – Fringe: \( \{x, y, z\} \)
Key concepts

• Explored nodes: we know the cheapest way to get there
  – Shown as inside the gray zone
• Fringe nodes: adjacent to an explored node
• When we increase the budget we add a fringe node into the set of explored nodes
  – This is pretty inefficient, hold that thought
• Keep on doing this until $t$ (i.e. Ithaca) is in the explored nodes
Budget approach is crazy

• Suppose the cheapest flight from LGA is $500
• In our example, imagine increasing by $.01
  – So we consider $2.01, $2.02, ...
• But we know that nothing will happen until we increase our budget to $3
  – Why not just do this directly?
Dijkstra’s algorithm

• We maintain an explored set $S$ with an **invariant**:  
  – For each $u \in S$ hold the **shortest** path from $s$ to $u$, write this as $d(u)$
    • Both the distance and the actual path, see HW1
    • Easiest to just think about the distance $d(u)$
  – Add an unexplored node $v$ to $S$
    • But, which one to choose?
    • On the fringe of $S$, so we add just one edge
Choice of edge for a fringe node

- The fringe node $v$ can be adjacent to several nodes in $S$.
  - If we choose to add $v$, pick the right node in $S$ to connect it to:

\[ d(u_1) + w_1 \text{ versus } d(u_2) + w_2 \]
Choice of fringe node

- If we pick \( v \) to add to \( S \), we will connect it to the \( u \) in \( S \) that minimizes \( d(u) + \) the length of the \((u, v)\) edge
  - Call this shortest path length \( \pi(v) \)
  - Think of this as “cheapest way to add \( v \) to \( S \)”
  - But can we pick an arbitrary \( v \) to add?
- Can prove that this would break our invariant about \( S \)!
- Pick \( v \) with smallest \( \pi(v) \), then add it to \( S \) with \( d(v) = \pi(v) \)
Shortest path example

Cost of path s-2-3-5-t

\[= 9 + 23 + 2 + 16 \]

\[= 48.\]
HW1 algorithm

• Start with $S = \{s\}$, all other nodes in $Q$
  – $d(s) = 0$, else $d(v) = \infty$ (i.e. upper bound)
• Pick $v$ on fringe of $S$ that minimizes $\pi(v)$
  – I.e., the $v \in Q$ with a neighbor in $S$ that is cheapest to add to $S$
• On recursive call, we will have
  – $d(v) = \Xi(v)$
  – $v$ is now in $S$, and no longer in $Q$
• Done when we pick target $t$
  – Computes more than shortest $s - t$ path!
Dijkstra's Shortest Path Algorithm

- Find shortest path from s to t.
- Blue edges: shortest path to a node within S.
- Green edges: what we would add for each fringe vertex.
\[ S = \{ s \} \]
\[ Q = \{ 2, 3, 4, 5, 6, 7, t \} \]
\[ S = \{ s \} \]
\[ Q = \{ 2, 3, 4, 5, 6, 7, t \} \]
\[ S = \{ s \} \]
\[ Q = \{ 2, 3, 4, 5, 6, 7, t \} \]
$S = \{ s, 2 \}$

$Q = \{ 3, 4, 5, 6, 7, t \}$
\[ S = \{ s, 2 \} \]
\[ Q = \{ 3, 4, 5, 6, 7, t \} \]
$S = \{ s, 2 \}$
$Q = \{ 3, 4, 5, 6, 7, t \}$
$S = \{ s, 2, 6 \}$
$Q = \{ 3, 4, 5, 7, t \}$
$S = \{ s, 2, 6 \}$
$Q = \{ 3, 4, 5, 7, t \}$
\[ S = \{ s, 2, 6, 7 \} \]
\[ Q = \{ 3, 4, 5, t \} \]
$S = \{ s, 2, 6, 7 \}$
$Q = \{ 3, 4, 5, t \}$
\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ Q = \{ 4, 5, t \} \]
\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ Q = \{ 4, 5, \infty \} \]
$S = \{ s, 2, 3, 5, 6, 7 \}$
$Q = \{ 4, t \}$
\[ S = \{ s, 2, 3, 5, 6, 7 \} \]
\[ Q = \{ 4, t \} \]
$S = \{ s, 2, 3, 4, 5, 6, 7 \}$
$Q = \{ t \}$
S = \{ s, 2, 3, 4, 5, 6, 7 \}
Q = \{ t \}
\[ S = \{ s, 2, 3, 4, 5, 6, 7, \top \} \]
\[ Q = \{ \} \]
\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
\[ Q = \{ \} \]
Implementation notes

- There are many ways to speed this up in practice
- Graph representations
  - You will explore 2 of these in HW1
- Naïve Dijkstra with \( n \) nodes and \( m \) edges is \( O(mn) \)
- We need to remove from \( Q \) the smallest node \( v \) with smallest value of \( \pi(v) \)
  - Priority queue implements remove-min in \( O(\log n) \)
  - This makes Dijkstra run in \( O(m \log n) \) time
Another class of examples

- Let’s model student behavior over time (hourly basis)
- Students have 3 possible states:
  - Awake (A)
  - Sleeping (S)
  - Doing CS5112 homework (H)
- If you know their state at time $t$, you know the probability of their other states at time $t + 1$
  - Example: A goes to A (.5), S (.49), H (.01)
Trellis graph

- We want to find the most likely 12 hour day for a student.
- At every time $t$ there are 3 nodes, for A/S/H.
- There are edges with transition probabilities.
  - Just like pirate grammar!
- So a day is a 12-node path through the graph.
- This is closely related to a “Hidden Markov Model”.
  - Widely used! Famous examples include speech, handwriting, computer vision, bioinformatics, etc.
Example

- Important note: with $S$ states and time $T$ there are $O(ST)$ nodes in the graph and $O(S^2T)$ edges
- So running time of naïve Dijkstra is $O(S^3T^2)$
- Can reduce this to $O(S^2T)$ with dynamic programming (Viterbi)