Lecture 7

Greedy Graph Algorithms

• Shortest paths
• Minimum Spanning Tree
The (Algorithm) Design Process

1. Work out the answer for some examples
2. Look for a general principle
   – Does it work on *all* your examples?
3. Write pseudocode
4. Test your algorithm by hand or computer
   – Does it work on *all* your examples?
   – Python is a great language for testing algorithms
5. Prove your algorithm is always correct
6. Check running time

Be prepared to go back to step 1!
Writing algorithms

• Clear and unambiguous
  – Test: You should be able to hand it to any student in the class, and have them convert it into working code.

• Homework pitfalls:
  – remember to specify data structures (list, stack, hash table,…)
  – For each function invocation, specify clearly what variables are passed to the function and what the function is returning.
  – writing recursive algorithms: don’t confuse the recursive subroutine with the first call
  – label global variables clearly
Writing proofs

• State upfront the claim you are proving.

• Purpose
  – **Determine for yourself** that algorithm is correct
  – Convince reader

• Who is your audience?
  – **Yourself**
  – Your classmates
  – Not the TA/grader

• **Main goal:** Find your own mistakes
Homework

• Goals:
  – Reinforce and clarify material from lecture
  – Develop your skills
    • Problem-solving
    • Communication

• Make sure you understand the solution
• Use the feedback
• If you don’t understand something, ask!
  – Me or the TA or on Piazza
• Do not copy from other sources
Shortest Paths
Shortest Path Problem

- **Input:**
  - Directed graph $G = (V, E)$.
  - Source node $s$, destination node $t$.
  - for each edge $e$, length $\ell(e) = \text{length of } e$.
  - length of a path = sum of lengths of edges on the path

- **Find:** shortest directed path from $s$ to $t$.
Dijkstra’s Algorithm: Overview

- Maintain a set of **explored nodes** $S$ whose shortest path distance $d(u)$ from $s$ to $u$ is known.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes
  $$\pi(v) = \min_{e=(u,v): u \in S} (d(u) + \ell(e))$$
- add $v$ to $S$, and set $d(v) = \pi(v)$.

![Diagram of Dijkstra's algorithm](image)

9/14/2016

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
Dijkstra’s Algorithm: Overview

- Maintain a set of explored nodes $S$ whose shortest path distance $d(u)$ from $s$ to $u$ is known.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes
  \[ \pi(v) = \min_{e=(u,v):u \in S} (d(u) + \ell(e)) \]
  - add $v$ to $S$, and set $d(v) = \pi(v)$.

Intuition: like BFS, but with weighted edges

Invariant: $d(u)$ is known for all vertices in $S$

Shortest path to some $u$ in explored part, followed by a single edge $(u, v)$
Correctness Proof of Dijkstra’s (Greedy Stays Ahead)

**Invariant.** For each node $u \in S$, $d(u)$ is the length of the shortest path from $s$ to $u$.

**Proof:** (by induction on $|S|$)

- **Base case:** $|S| = 1$; $d(s) = 0$.

- **Inductive hypothesis:** Assume for $|S| = k \geq 1$.
  - Let $v$ be next node added to $S$, and let $(u, v)$ be the chosen edge.
  - The shortest $s$-$u$ path plus $(u, v)$ is an $s$-$v$ path of length $\pi(v)$.
  - Consider any $s$-$v$ path $P$. We'll see that it's no shorter than $\pi(v)$.
  - Let $(x, y)$ be the first edge in $P$ that leaves $S$, and let $P'$ be the subpath to $x$.
  - $P' + (x, y)$ has length $\geq d(x) + \ell(x, y) \geq \pi(y) \geq \pi(v)$.

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
Implementation

• For unexplored nodes, maintain

\[ \pi(v) = \min_{e=(u,v): u \in S} (d(u) + \ell(e)) \]

– Next node to explore = node with minimum \( \pi(v) \).

– When exploring \( v \), for each edge \( e = (v,w) \), update

\[ \pi(w) = \min\{\pi(w), \pi(v) + \ell(e)\} \]

• Efficient implementation: Maintain a priority queue \( Q \) of unexplored nodes, prioritized by \( \pi(v) \).
Implementation: priority queues

- Maintain a set of items with priorities (\(= \text{“keys”}\))
  - Example: jobs to be performed
- Operations:
  - \text{INSERT}
  - \text{DECREASE-KEY}
  - \text{EXTRACT-MIN}: find and remove item with least key
- Common data structure: heap
  - Time: \(O(\log n)\) per operation
Demo of Dijkstra’s Algorithm

Graph with nonnegative edge lengths:
Demo of Dijkstra’s Algorithm

Initialize:

\[ Q: \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty & \infty \\
\end{array} \]

\[ \pi(v): \begin{array}{cccccc}
0 & \infty & \infty & \infty & \infty & \infty \\
\end{array} \]

\[ S: \{ \} \]
**Demo of Dijkstra’s Algorithm**

**EXTRACT-MIN**(Q) is A:

- Q: \(A\ B\ C\ D\ E\)
- \(\pi(v)\): 0 \(\infty\) \(\infty\) \(\infty\) \(\infty\)

- S: \{ A \}

Diagram: A connected graph with weights on edges.
Demo of Dijkstra’s Algorithm

Explore edges leaving $A$:

$Q$: $\begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty \\
\end{array}$

$\pi(v)$: $\begin{array}{cccccc}
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty \\
\end{array}$

$S$: $\{A\}$
**Demo of Dijkstra’s Algorithm**

**EXTRACT-MIN(Q) is C:**

Q:  
\[ \begin{array}{cccccc} & A & B & C & D & E \\ \pi(v): & 0 & \infty & \infty & \infty & \infty \\ & 10 & 3 & \infty & \infty & \infty \end{array} \]

\[ S: \{ A, C \} \]
Demo of Dijkstra’s Algorithm

Explore edges leaving $C$:

$Q$: $\begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty \\
7 & 11 & 5 & & \\
\end{array}$

$\pi(v)$: $\begin{array}{c}
0 \\
10 \\
7 \\
\end{array}$

$S$: $\{ A, C \}$
**Demo of Dijkstra’s Algorithm**

**EXTRACT-MIN(Q) is E:**

- **Q:**
  - A: 0
  - B: ∞
  - C: ∞
  - D: ∞
  - E: ∞

- **π(v):**
  - A: 0
  - B: ∞
  - C: ∞
  - D: ∞
  - E: ∞

- **S:** \{ A, C, E \}

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
Demo of Dijkstra’s Algorithm

Explore edges leaving $E$:

$Q$:  

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi(v)$:</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$S$: \{ A, C, E \}
Demo of Dijkstra’s Algorithm

**EXTRACT-MIN**(Q) is B:

\[ Q: \begin{array}{cccccc}
    \text{A} & \text{B} & \text{C} & \text{D} & \text{E} \\
    0 & \infty & \infty & \infty & \infty \\
    10 & 3 & \infty & \infty & \\
    7 & 11 & 5 & \\
    7 & 11 & \\
\end{array} \]

\[ \pi(v): \text{A} & \text{B} & \text{C} & \text{D} & \text{E} \\
    0 & \infty & \infty & \infty & \infty \\
    10 & 3 & \infty & \infty \\
    7 & 11 & 5 \\
    7 & 11 \\
\]

\[ S: \{ A, C, E, B \} \]
Demo of Dijkstra’s Algorithm

Explore edges leaving $B$:

$Q:$

$\pi(v):$

$S: \{ A, C, E, B \}$

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
Demo of Dijkstra’s Algorithm

**Extract-Min**(Q) is D:

\[ Q: \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \\
7 & 7 & 11 & 5 & \\
\end{array} \]

\[ \pi(v): \begin{array}{cccccc}
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \\
7 & 7 & 11 & 5 & \\
\end{array} \]

\[ S: \{ A, C, E, B, D \} \]
Pseudocode for Dijkstra($G, \ell$)

\[
d[s] \leftarrow 0
\]

for each $v \in V - \{s\}$
\[
do \ d[v] \leftarrow \infty; \ \pi[v] \leftarrow \infty
\]

$S \leftarrow \emptyset$

$Q \leftarrow V$ \quad \triangleright Q \text{ is a priority queue maintaining } V - S, \text{ keyed on } \pi[v]

while $Q \neq \emptyset$

\[
do \ u \leftarrow \text{EXTRACT-MIN}(Q)\]

\[
S \leftarrow S \cup \{u\}; \ d[u] \leftarrow \pi[u]
\]

for each $v \in \text{Adjacency-list}[u]$

\[
do \ \text{if } \pi[v] > \pi[u] + \ell(u, v) \]
then $\pi[v] \leftarrow d[u] + \ell(u, v)$

Implicit DECREASE-KEY

\[
9/14/2016
S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
L7.24
\]
### Analysis of Dijkstra

While $Q \neq \emptyset$

1. **do** $u \leftarrow \text{Extract-Min}(Q)$
2. **S** $\leftarrow S \cup \{u\}$
3. For each $v \in \text{Adj}[u]$
   - **do if** $d[v] > d[u] + \ell(u, v)$
     - **then** $d[v] \leftarrow d[u] + \ell(u, v)$

\[ m \] implicit Decrease-Key’s.

---

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Dijkstra</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap †</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExtractMin</td>
<td>$n$</td>
<td>$n$</td>
<td>log $n$</td>
<td>HW</td>
<td>log $n$</td>
</tr>
<tr>
<td>DecreaseKey</td>
<td>$m$</td>
<td>1</td>
<td>log $n$</td>
<td>HW</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>$n^2$</td>
<td>$m \log n$</td>
<td>$m \log m/n$</td>
<td>$m + n \log n$</td>
<td></td>
</tr>
</tbody>
</table>

† Individual ops are amortized bounds

---

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
Physical intuition

- System of pipes filling with water
  - Vertices are intersections
  - Edge length = pipe length
  - $d(v)$ = time at which water reaches $v$

- Balls and strings
  - Vertices $\mapsto$ balls
  - Edge $e$ $\mapsto$ string of length $\ell(e)$
  - Hold ball $s$ up in the air
  - $d(v) = (\text{height of } s) - (\text{height of } v)$

- Nature uses greedy algorithms
Review

• Is Dijkstra’s algorithm correct with **negative** edge weights?
  Give either
  – a proof of correctness, or
  – an example of a graph where Dijkstra fails
Further reading

- Erickson’s lecture notes:
  http://web.engr.illinois.edu/~jeffe/teaching/algorithms/notes/21-sssp.pdf
Minimum Spanning Tree
Minimum spanning tree (MST)

Input: A connected undirected graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{R}$.
- For now, assume all edge weights are distinct.

Definition: A spanning tree is a tree that connects all vertices.

Output: A spanning tree $T$ of minimum weight:

$$w(T) = \sum_{(u, v) \in T} w(u, v).$$
Example of MST

![MST Diagram]
Example of MST
Greedy Algorithms for MST

- **Kruskal's:** Start with $T = \emptyset$. Consider edges in ascending order of weights. Insert edge $e$ in $T$ unless doing so would create a cycle.

- **Reverse-Delete:** Start with $T = E$. Consider edges in descending order of weights. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

- **Prim's:** Start with some root node $s$. Grow a tree $T$ from $s$ outward. At each step, add to $T$ the cheapest edge $e$ with exactly one endpoint in $5$.

- **Borůvka’s:** Start with $T = \emptyset$. At each round, add the cheapest edge leaving each connected component of $T$. 

Cycles and Cuts

• **Cycle:** Set of edges of the form \((a,b),(b,c),\ldots,(y,z),(z,a)\).

  ![Graph Diagram]

  - **Cycle:** \(C = (1,2),(2,3),(3,4),(4,5),(5,6),(6,1)\)

• **Cut:** a subset of nodes \(S\). The corresponding **cutset** \(D\) is the subset of edges with exactly one endpoint in \(S\).

  ![Graph Diagram]

  - **Cut:** \(S = \{4, 5, 8\}\)
  - **Cutset:** \(D = (5,6), (5,7), (3,4), (3,5), (7,8)\)
Claim. A cycle and a cutset intersect in an even number of edges.

Proof: A cycle has to leave and enter the cut the same number of times.
**Cut and Cycle Properties**

- **Cut property.** Let $S$ be a subset of nodes. Let $e$ be the min weight edge with exactly one endpoint in $S$. Then the MST contains $e$.

- **Cycle property.** Let $C$ be a cycle, and let $f$ be the max weight edge in $C$. Then the MST does not contain $f$.

![Diagram illustrating the cut and cycle properties](image-url)
Proof of Cut Property

**Cut property:** Let $S$ be a subset of nodes. Let $e$ be the min weight edge with exactly one endpoint in $S$. Then the MST $T^*$ contains $e$.

**Proof:** (exchange argument)

- Suppose $e$ does not belong to $T^*$.
- Adding $e$ to $T^*$ creates a cycle $C$ in $T^*$.
- Edge $e$ is both in the cycle $C$ and in the cutset $D$ corresponding to $S \Rightarrow$ there exists another edge, say $f$, that is in both $C$ and $D$.
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$. Contradiction."
Proof of Cycle Property

**Cycle property:** Let $C$ be a cycle in $G$. Let $f$ be the max weight edge in $C$. Then the MST $T^*$ does not contain $f$.

- **Proof:** (exchange argument)
  - Suppose $f$ belongs to $T^*$.
  - Deleting $f$ from $T^*$ creates a cut $S$ in $T^*$.
  - Edge $f$ is both in the cycle $C$ and in the cutset $D$ corresponding to $S \Rightarrow$ there exists another edge, say $e$, that is in both $C$ and $D$.
  - $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
  - Since $c_e < c_f$, cost($T'$) < cost($T^*$). Contradiction. □
Greedy Algorithms for MST

• **Kruskal's**: Start with $T = \emptyset$. Consider edges in ascending order of weights. Insert edge $e$ in $T$ unless doing so would create a cycle.

• **Reverse-Delete**: Start with $T = E$. Consider edges in descending order of weights. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

• **Prim's**: Start with some root node $s$. Grow a tree $T$ from $s$ outward. At each step, add to $T$ the cheapest edge $e$ with exactly one endpoint in $5$.

• **Borůvka's**: Start with $T = \emptyset$. At each round, add the cheapest edge leaving each connected component of $T$. 
Prim's Algorithm: Correctness

• Prim's algorithm. [Jarník 1930, Prim 1959]
  – Apply cut property to $S$.
  – When edge weights are distinct, every edge that is added must be in the MST
  – Thus, Prim’s algorithm outputs the MST
Correctness of Kruskal

- [Kruskal, 1956]: Consider edges in ascending order of weight.
  - **Case 1**: If adding $e$ to $T$ creates a cycle, discard $e$ according to cycle property.
  
  - **Case 2**: Otherwise, insert $e = (u, v)$ into $T$ according to cut property where $S =$ set of nodes in $u$'s connected component.