Lecture 26

Graphs

• Graph representation
• Dijkstra’s algorithm for shortest paths
Review Question

• Recall last lectures were interval search: given interval \( i = [a,b] \), find some interval in the tree that overlaps \( i \)

\[ Q: \text{What if we want all intervals that overlap } i? \]
  
  - Express the running time as a function of \# of overlapping intervals \( k \)

\[ A: \]

  • Search, list, delete, repeat.
  
  - Insert them all again at the end.

  • Time = \( O(k \lg n) \), where \( k \) is the total number of overlapping intervals.
    
    - This is an output-sensitive bound.
    
    - Best algorithm to date: \( O(k + \lg n) \).
**Review Question 2**

**Q:** Suppose I want a dynamic set which allows searching on two different keys (say either name or SSN)

- Insert, Delete, NameSearch, SSNSearch
- Give a simple augmented data structure which allows $O(1)$ time insert, delete and search.

**A:** Maintain two hash tables (keyed on name, ssn)

- Hash tables just contain pointers to the full records
Graphs (review CLRS appendix B)

Definition. A directed graph (digraph) $G = (V, E)$ is an ordered pair consisting of
- a set $V$ of vertices (synonym: nodes),
- a set $E \subseteq V \times V$ of edges
  An edge $e = (u, v)$ goes “from $u$ to $v$”
- In an undirected graph $G = (V, E)$, the edge set $E$ consists of unordered pairs of vertices.
  Sometimes write $e = \{u, v\}$
- How many edges can a graph have?
  In either case, we have $|E| = O(V^2)$.
- $G = (V, E)$ is connected if there is a path between any pair of vertices
  - if $G$ is connected, then $|E| \geq |V| - 1$, which implies that $\lg |E| = \Theta(\lg V)$. 
Graph examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transportation network: airline routes</td>
<td>airports</td>
<td>nonstop flights</td>
</tr>
<tr>
<td>Comm. networks</td>
<td>computers, hubs, routers</td>
<td>physical wires</td>
</tr>
<tr>
<td>Information network: web</td>
<td>pages</td>
<td>hyperlinks</td>
</tr>
<tr>
<td>Info. network: scientific papers</td>
<td>articles</td>
<td>references</td>
</tr>
<tr>
<td>Social networks</td>
<td>people</td>
<td>“u is v’s friend”, “u sends email to v”, “u’s MySpae pag links to v”</td>
</tr>
</tbody>
</table>
Adjacency-matrix representation

The *adjacency matrix* of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \ldots n, 1 \ldots n]$ given by

$$A[i, j] = \begin{cases} 
1 & \text{if } (i, j) \in E, \\
0 & \text{if } (i, j) \notin E.
\end{cases}$$
Adjacency-matrix representation

The *adjacency matrix* of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \ldots n, 1 \ldots n]$ given by

$$A[i, j] = \begin{cases} 
1 & \text{if } (i, j) \in E, \\
0 & \text{if } (i, j) \notin E.
\end{cases}$$

$$
\begin{array}{c|cccc}
   & 1 & 2 & 3 & 4 \\
\hline
1 & 0 & 1 & 1 & 0 \\
2 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 \\
\end{array}
$$

$\Theta(V^2)$ storage $\Rightarrow$ *dense* representation.
Adjacency-list representation

An *adjacency list* of a vertex \( v \in V \) is the list \( Adj[v] \) of vertices adjacent to \( v \).

\[
\begin{aligned}
Adj[1] &= \{2, 3\} \\
Adj[2] &= \{3\} \\
Adj[3] &= \{} \\
Adj[4] &= \{3\}
\end{aligned}
\]
Adjacency-list representation

An *adjacency list* of a vertex $v \in V$ is the list $\text{Adj}[v]$ of vertices adjacent to $v$.

For undirected graphs, $|\text{Adj}[v]| = \text{degree}(v)$. For digraphs, $|\text{Adj}[v]| = \text{out-degree}(v)$.
Adjacency-list representation

An adjacency list of a vertex \( v \in V \) is the list \( Adj[v] \) of vertices adjacent to \( v \).

\[ Adj[1] = \{2, 3\} \]
\[ Adj[2] = \{3\} \]
\[ Adj[3] = {} \]
\[ Adj[4] = \{3\} \]

For undirected graphs, \( |Adj[v]| = \text{degree}(v) \).
For digraphs, \( |Adj[v]| = \text{out-degree}(v) \).

**Handshaking Lemma:** \( \sum_{v \in V} \text{degree}(v) = 2|E| \) for undirected graphs \( \Rightarrow \) adjacency lists use \( \Theta(V + E) \) storage — a sparse representation.

S. Raskhodnikova and A. Smith. Based on slides by E. Demaine and C.E. Leiserson
Review Question

• Suppose that an undirected graph $G$
  
  - has no cycles
  - has exactly $n-1$ edges (and no self-loops)

• Prove or disprove: $G$ is connected.
  
  - (Answer: Connected. The proof can be reduced to the following, simpler claim: if $G$ has more than $n-1$ edges, then it contains a cycle.)
Basic Graph Algorithms

• Over next few lectures, we’ll look at answering question such as:
  ❖ What is the shortest path from $s$ to $t$?
  ❖ How can one explore a graph systematically?
  ❖ What is a minimal spanning tree of $G$?
Paths in graphs

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The weight of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$
Consider a digraph $G = (V, E)$ with edge-weight function $w : E \rightarrow \mathbb{R}$. The weight of path $p = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k$ is defined to be

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

**Example:**

$w(p) = -2$
Shortest paths

A *shortest path* from $u$ to $v$ is a path of minimum weight from $u$ to $v$. The *shortest-path weight* from $u$ to $v$ is defined as

$$ \delta(u, v) = \min \{ w(p) : p \text{ is a path from } u \text{ to } v \} . $$

**Note:** $\delta(u, v) = \infty$ if no path from $u$ to $v$ exists.
Well-definedness of shortest paths

If a graph $G$ contains a negative-weight cycle, then some shortest paths do not exist.
Well-definedness of shortest paths

If a graph $G$ contains a negative-weight cycle, then some shortest paths do not exist.

Example:
Optimal substructure

**Theorem.** A subpath of a shortest path is a shortest path.
Optimal substructure

**Theorem.** A subpath of a shortest path is a shortest path.

**Proof.** Cut and paste:
Optimal substructure

**Theorem.** A subpath of a shortest path is a shortest path.

**Proof.** Cut and paste:
Triangle inequality

**Theorem.** For all $u, v, x \in V$, we have $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$. 
Theorem. For all $u, v, x \in V$, we have $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$.

Proof.
Dijkstra’s algorithm for shortest paths
Single-source shortest paths
(nonnegative edge weights)

**Problem.** Assume that \( w(u, v) \geq 0 \) for all \((u, v) \in E\). From a given source vertex \( s \in V \), find the shortest paths for all \( v \in V \).

In fact, today we only compute **distances** \( \delta(s, v) \)

**Idea:** Greedy.

1. Maintain a distance estimate for all vertices
2. Maintain a set \( S \) of vertices whose shortest-path distances from \( s \) are known.
3. At each step, add to \( S \) the vertex \( v \in V – S \) with smallest distance estimate.
4. Update the distance estimates of vertices adjacent to \( v \).
Dijkstra’s algorithm

\[ d[s] \leftarrow 0 \quad \triangleright \quad \text{Eventually, } d[v] = \text{distance } s \text{ to } v \text{ for all } v \]

for each \( v \in V - \{s\} \)
\[ d[v] \leftarrow \infty \]

\( S \leftarrow \emptyset \)

\( Q \leftarrow V \quad \triangleright \quad Q \text{ is a priority queue maintaining } V - S, \] using \( d[v] \) as the key of \( v \)
Dijkstra’s algorithm

\[
d[s] \leftarrow 0 \quad \triangleright \text{Eventually, } d[v] = \text{distance } s \text{ to } v \text{ for all } v
\]

\[\text{for each } v \in V - \{s\} \quad \text{do } d[v] \leftarrow \infty\]
\[S \leftarrow \emptyset\]
\[Q \leftarrow V \quad \triangleright Q \text{ is a priority queue maintaining } V - S, \text{ using } d[v] \text{ as the key of } v\]

\[\text{while } Q \neq \emptyset \quad \text{do } u \leftarrow \text{EXTRACT-MIN}(Q)\]
\[S \leftarrow S \cup \{u\}\]
\[\text{for each } v \in \text{Adj}[u] \quad \text{do}\]
\[\quad \text{if } d[v] > d[u] + w(u, v) \quad \text{then } d[v] \leftarrow d[u] + w(u, v)\]
Dijkstra’s algorithm

\[
d[s] \leftarrow 0 \quad \triangleright \text{Eventually, } d[v] = \text{distance } s \text{ to } v \text{ for all } v
\]

for each \( v \in V - \{s\} \)

\[
d[v] \leftarrow \infty
\]

\( S \leftarrow \emptyset \)

\( Q \leftarrow V \quad \triangleright Q \text{ is a priority queue maintaining } V - S, \)

using \( d[v] \) as the key of \( v \)

while \( Q \neq \emptyset \)

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

\[
S \leftarrow S \cup \{u\}
\]

for each \( v \in \text{Adj}[u] \)

\[
\text{do if } d[v] > d[u] + w(u, v) \quad \text{relaxation step}
\]

\[
\text{then } d[v] \leftarrow d[u] + w(u, v)
\]

\[
\downarrow \text{Requires DECREASE-KEY}
\]

S. Raskhodnikova and A. Smith. Based on slides by E. Demaine and C.E. Leiserson
Example of Dijkstra’s algorithm

Graph with nonnegative edge weights:
Example of Dijkstra’s algorithm

Initialize:

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

\[ S: \quad \{\} \]
Example of Dijkstra’s algorithm

“$A$” $\leftarrow$ \textbf{Extract-Min}(Q):

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

S: \{ A \}
Example of Dijkstra’s algorithm

Relax all edges leaving \( A \):

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

\[ S: \{ A \} \]
Example of Dijkstra’s algorithm

“C” ← \text{\textsc{extract-min}}(Q):

\begin{align*}
Q & : \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty \\
\end{array} \\
S & : \{ A, C \}
\end{align*}
Example of Dijkstra’s algorithm

Relax all edges leaving \( C \):
Example of Dijkstra’s algorithm

“E” ← \textsc{Extract-Min}(Q):

\begin{align*}
Q: & \quad A & B & C & D & E \\
   & 0 & \infty & \infty & \infty & \infty \\
   & 10 & 3 & \infty & \infty & \\
   & 7 & 11 & 5 & & \\
\end{align*}

\begin{align*}
S: & \quad \{A, C, E\} \\
\end{align*}
Example of Dijkstra’s algorithm

Relax all edges leaving $E$:

$Q$: $A$ $B$ $C$ $D$ $E$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>$\infty$</th>
<th>$\infty$</th>
<th>$\infty$</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>0</td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$B$</td>
<td>$\infty$</td>
<td>3</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>5</td>
</tr>
<tr>
<td>$C$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>3</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>$D$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$E$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>5</td>
</tr>
</tbody>
</table>

$S$: { $A$, $C$, $E$ }
Example of Dijkstra’s algorithm

“$B$” ← $\text{EXTRACT-MIN}(Q)$:

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

$S$: $\{A, C, E, B\}$
Example of Dijkstra’s algorithm

Relax all edges leaving $B$:

$Q$: 

<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>Q</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>11</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$S$: \{ A, C, E, B \}
Example of Dijkstra’s algorithm

"D" ← \text{Extract-Min}(Q):

\[
\begin{array}{c|cccccc}
Q: & A & B & C & D & E \\
\hline
0 & \infty & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty & \\
7 & 7 & 11 & 11 & 5 & \\
7 & 7 & 11 & 11 & 9 & \\
\end{array}
\]

S: \{ A, C, E, B, D \}
Correctness — Part I

**Lemma.** At every step after initialization, $d[v] \geq \delta(s, v)$ for all $v \in V$.

**Proof.** Suppose not. Let $v$ be the first vertex for which $d[v] < \delta(s, v)$, and let $u$ be the vertex that caused $d[v]$ to change: $d[v] = d[u] + w(u, v)$. Then,

\[
\begin{align*}
    d[v] &< \delta(s, v) & \text{supposition} \\
    \leq \delta(s, u) + \delta(u, v) & \text{triangle inequality} \\
    \leq \delta(s, u) + w(u, v) & \text{sh. path} \leq \text{specific path} \\
    \leq d[u] + w(u, v) & v \text{ is first violation}
\end{align*}
\]

Contradiction. \qed
Lemma. Let $u$ be $v$’s predecessor on a shortest path from $s$ to $v$. Then, if $d[u] = \delta(s, u)$ and edge $(u, v)$ is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.

Proof. Observe that $\delta(s, v) = \delta(s, u) + w(u, v)$. Suppose that $d[v] > \delta(s, v)$ before the relaxation. (Otherwise, we’re done.) Then, the test $d[v] > d[u] + w(u, v)$ succeeds, because $d[v] > \delta(s, v) = \delta(s, u) + w(u, v) = d[u] + w(u, v)$, and the algorithm sets $d[v] = d[u] + w(u, v) = \delta(s, v)$. 

S. Raskhodnikova and A. Smith. Based on slides by E. Demaine and C.E. Leiserson
Correctness — Part III

**Theorem.** Dijkstra’s algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

**Proof.** It suffices to show that $d[v] = \delta(s, v)$ for every $v \in V$ when $v$ is added to $S$. Suppose $u$ is the first vertex added to $S$ for which $d[u] > \delta(s, u)$. Let $y$ be the first vertex in $V - S$ along a shortest path from $s$ to $u$, and let $x$ be its predecessor:

\[
S, \text{ just before adding } u.
\]
Correctness — Part III (continued)

Since $u$ is the first vertex violating the claimed invariant, we have $d[x] = \delta(s, x)$. When $x$ was added to $S$, the edge $(x, y)$ was relaxed, which implies that $d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$. But, $d[u] \leq d[y]$ by our choice of $u$. Contradiction.
Analysis of Dijkstra

while $Q \neq \emptyset$
do $u \leftarrow \text{Extract-Min}(Q)$
$S \leftarrow S \cup \{u\}$
for each $v \in Adj[u]$
do if $d[v] > d[u] + w(u, v)$
then $d[v] \leftarrow d[u] + w(u, v)$
Analysis of Dijkstra

\[
\text{while } Q \neq \emptyset \\
\text{do } u \leftarrow \text{Extract-Min}(Q) \\
S \leftarrow S \cup \{u\} \\
\text{for each } v \in \text{Adj}[u] \\
\text{do if } d[v] > d[u] + w(u, v) \\
\text{then } d[v] \leftarrow d[u] + w(u, v)
\]
Analysis of Dijkstra

\[
\text{while } Q \neq \emptyset \\
\text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \\
S \leftarrow S \cup \{u\} \\
\text{for each } v \in \text{Adj}[u] \\
\quad \text{do if } d[v] > d[u] + w(u, v) \\
\quad \quad \text{then } d[v] \leftarrow d[u] + w(u, v)
\]

\(|V|\) times

\(\text{degree}(u)\) times
Analysis of Dijkstra

\[
\text{while } Q \neq \emptyset \\
\text{do } u \leftarrow \text{Extract-Min}(Q) \\
S \leftarrow S \cup \{u\} \\
\text{for each } v \in \text{Adj}[u] \\
\text{do if } d[v] > d[u] + w(u, v) \\
\text{then } d[v] \leftarrow d[u] + w(u, v)
\]

At most once for each edge $\Rightarrow \Theta(E)$ Decrease-Key’s.
Analysis of Dijkstra

\[ |V| \text{ times} \]

\[ \text{while } Q \neq \emptyset \]
\[ \text{do } u \leftarrow \text{Extract-Min}(Q) \]
\[ S \leftarrow S \cup \{u\} \]
\[ \text{for each } v \in \text{Adj}[u] \]
\[ \text{do if } d[v] > d[u] + w(u, v) \]
\[ \text{then } d[v] \leftarrow d[u] + w(u, v) \]

Handshaking Lemma $\Rightarrow \Theta(E)$ implicit Decrease-Key's.

Time $= \Theta(V \cdot T_{\text{Extract-Min}} + E \cdot T_{\text{Decrease-Key}})$

**Note:** Same formula as in the analysis of Prim’s minimum spanning tree algorithm.

S. Raskhodnikova and A. Smith. Based on slides by E. Demaine and C.E. Leiserson
Analysis of Dijkstra (continued)

\[ \text{Time} = \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}} \]

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$T_{\text{Extract-Min}}$</th>
<th>$T_{\text{Decrease-Key}}$</th>
<th>Total</th>
</tr>
</thead>
</table>
Analysis of Dijkstra (continued)

\[
\text{Time} = \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}}
\]

<table>
<thead>
<tr>
<th>(Q)</th>
<th>(T_{\text{Extract-Min}})</th>
<th>(T_{\text{Decrease-Key}})</th>
<th>\text{Total}</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>(O(V))</td>
<td>(O(1))</td>
<td>(O(V^2))</td>
</tr>
</tbody>
</table>
Analysis of Dijkstra (continued)

\[
\text{Time} = \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}
\]

<table>
<thead>
<tr>
<th></th>
<th>( Q )</th>
<th>( T_{\text{EXTRACT-MIN}} )</th>
<th>( T_{\text{DECREASE-KEY}} )</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>( O(V) )</td>
<td>( O(1) )</td>
<td>( O(V^2) )</td>
<td></td>
</tr>
<tr>
<td>heap</td>
<td>( O(\lg V) )</td>
<td>( O(\lg V) )</td>
<td>( O(E \lg V) )</td>
<td></td>
</tr>
</tbody>
</table>
What else does the algorithm give us?

• Every time we update $d[u]$, we can keep track of its predecessor (that is, the node that caused the most recent update of $d[u]$)
  - This tells us the last step of a shortest path from $s$ to $u$
  - Putting these steps together, we can quickly construct the shortest path from $s$ to any other given node