Overview

• **HW7 due in one week**

• **Q&A on graphs/trees on Thursday!**

• Last lecture: Graph Theory
  – Adjacency Matrix, Adjacency Lists
  – Isomorphism
  – Path, Circuit, Cycle: Hamiltonian Cycle, Eulerian Circuit
  – Connectivity, Connected Components, Planar Graphs

• Today’s lecture: Completing Graphs & Trees
  – Shortest path problem: Dijkstra algorithm
  – Traveling salesman problem
  – Trees: Definitions & Spanning trees

**Application: Shortest Path Problems**

• We can assign **weights** to the edges of graphs, for example, to represent the distance between cities in a railway network:

![Graph diagram]

• One of the most interesting questions that we can investigate with such graphs is:

> What is the **shortest path** between two vertices in the graph, that is, the path with the **minimal sum of weights** along the way?

> This corresponds to the shortest train connection or the fastest connection in a computer network (edge weight = response time).
Dijkstra’s Algorithm

- Dijkstra’s algorithm is an iterative procedure that finds the shortest path between two vertices (e.g., \( a \) to \( z \)) in a weighted graph.
- It proceeds by finding the length of the shortest path from \( a \) to successive vertices. \( a \rightarrow z \)
- The algorithm terminates once it reaches the destination vertex \( z \). The final shortest path is then back-tracked from \( z \).

Example:

Answer: \( a, c, b, d, e, z \) keep min cost value on each vertex

```
function Dijkstra(Graph, source(a), target(z)):
    for each vertex \( v \) in Graph: // Initializations
        \( \text{dist}[v] := \infty \) // Unknown distance function from source to \( v \)
        \( \text{prev}[v] := \text{undefined} \) // Previous node in optimal path from source
        \( \text{dist}[source] := 0 \) // Distance from source to source
    \( Q := \) the set of all nodes in Graph
    while \( Q \) is not empty: // The main loop
        \( u := \) node in \( Q \) with smallest \( \text{dist}[u] \)
        remove \( u \) from \( Q \)
        if \( u = \text{target} \), exit, done!
        for each neighbor \( v \) of \( u \) and \( v \) still in \( Q \).
            \( \text{alt} := \text{dist}[u] + \text{dist}_\text{between}(u, v) \)
            if \( \text{alt} < \text{dist}[v] \) // Relax \((u,v)\)
                \( \text{dist}[v] := \text{alt} \) // update \( v \)'s data
                \( \text{prev}[v] := u \) // update \( v \)'s previous node to \( u \)
```

© by Kazunori Okada, 2021
\[ \text{dist}[V = a, \ldots z] = \infty \]
\[ \text{prev}[V = 0, \ldots z] = \text{undefined} \]

1. \[ \text{dist}[a] = 0 \quad \text{Source} \]
2. Pop node w/smallest \( d(c) \) from \( Q \)
3. Look at all neighbors of \( a \rightarrow b, c \)
   - (b) \( alt = d(a) + 4 + 4 \)
     \[ d(c) = 4, p(c) = a \]
   - (c) \( alt = d(a) + 2 + 2 \)
     \[ d(c) = 4, p(c) = a \]
     Remove \( a \) from \( Q \)

4. Pop \( Q \rightarrow c \)
5. Remove \( c \) from \( Q \)
6. Look at neighbors of \( c \) and \( b, d, e \)
   - (b) \( alt = d(c) + 3 < d(b) = 4 \)
     \[ d(b) = 3, p(b) = c \]

\[ Q = \{ a, b, c, d, e, z \} \]

1. \( alt = d(c) + 10 < d(e) = 10 \)
   \[ d(c) = 10, p(c) = z \]
2. \( alt = d(c) + 2 < d(e) = 10 \)
   \[ d(c) = 10, p(c) = z \]
3. \( alt = d(c) + 14 < d(e) = 10 \)
   \[ d(c) = 10, p(c) = z \]
4. Pop \( Q \rightarrow e \)
5. Remove \( e \)
6. Pop \( Q \rightarrow z \)
7. Pop \( Q \rightarrow z \)
8. Now back track from \( z \) to \( b, a \)
Application: The Traveling Salesman Problem

• The **traveling salesman problem** is one of the classical problems in computer science.

A traveling salesman wants to visit all major cities and then return to his starting point. Of course he wants to save time and energy, so he wants to determine the **shortest path** for his trip.

We can represent the cities and the distances between them by a **weighted, complete, undirected graph**.

The problem then is to find a **Hamiltonian cycle of minimum total weight that visits each vertex exactly one**.

---

**Example:** What path would the traveling salesman take to visit the following cities?

**Solution:** The shortest path is Boston, New York, Chicago, Toronto, Boston (2,000 miles).

**Question:** Given $n$ vertices, how many different cycles (with all $n$ vertices) can we form by connecting these vertices with edges?
Solution: We first choose a starting point. Then we have \((n - 1)\) choices for the second vertex in the cycle, \((n - 2)\) for the third one, and so on, so there are \((n - 1)!\) choices for the whole cycle. However, this number includes identical cycles that were constructed in opposite directions. Therefore, the actual number of different cycles is \((n - 1)!/2\).

• Unfortunately, no algorithm solving the traveling salesman problem with polynomial worst-case time complexity has been devised yet.

• This means that for large numbers of vertices, solving the traveling salesman problem is not tractable (impractical).

• In these cases, we can use efficient approximation algorithms that determine a path whose length may be slightly larger than the traveling salesman’s path.
• Definition: A graph $G$ is said to be a **tree** if it is **connected** and has no cycle (**acyclic**).

• $G$ is said to be a **forest** if it consists of several trees.

Definition: a **spanning tree** $G'$ of an undirected graph $G$ satisfies:

- $G'$ is a subgraph of $G$
- $G'$ consists of all vertices in $G$
- $G'$ is a tree (forest then you have spanning forest)

$G$

Several Spanning trees $G'$ of $G$

$\begin{array}{c}
\text{Several Spanning forests } G' \text{ of } G \\
\end{array}$
**Kruskal Algorithm** (to find a spanning tree):

**Input:** a connected graph \( G = (V, E) \)

**Output:** a spanning tree \((V, T)\) of \( G \)

\[
T = \emptyset \quad \text{— initial set of edges}
\]

for each \( e \in E \) {
    if \((V, \{e\} \cup T)\) is acyclic then
        \( T = T \cup \{e\} \)
}

return \((V, T)\)

Note: above algorithm can be modified to obtain **minimum cost spanning tree**. How?

\( \rightarrow \) Sort \( E \) by weights in the increasing order

© by Kazunori Okada, 2021

---

**Minimum Cost Spanning Tree**