Overview

• HW7 due in two weeks right after the break
• 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
• Today’s lecture: Completing Graphs & Trees
  – Planar Graphs
  – Shortest path problem: Dijkstra algorithm
  – Traveling salesman problem
  – Trees: Definitions & Spanning trees

A planar graph is one that can be represented (on a sheet of paper, that is, in the plane) so that its edges intersect only at endpoints.

Right graph = Left graph. It is clearly a planar graph.

What about $K_4$ and $K_5$?
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:

- 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.
- 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
    dist[v] := infinity  // Unknown distance function from source to v
    prev[v] := undefined  // Previous node in optimal path from source
    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 dist[]
    remove u from Q
    if u = target, exit, done!
    for each neighbor v of u and v still in Q,
        alt := dist[u] + dist_between(u, v)
        if alt < dist[v]  // Relax (u,v)
            dist[v] := alt  // update v's data
            prev[v] := u  // update v's previous node to u
            Q := Q - v  // iterate for all neighbors of u!

\[
\text{dist[v] = } \infty \\
\text{prev[v] = undefined}
\]

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

- dist[a] = 0  (Source)
- Pop node w/ smallest dist(c) \rightarrow a
- look at all neighbors of a \rightarrow b, c
  - b) alt = dist[a] + 4 = 4
    since dist[b] = \infty
    d[cb] = 4, p[cb] = 4
  - c) alt = dist[a] + 2 = 2
    since dist[c] = \infty
    d[ec] = 2, p[ec] = 2
- remove a from Q

\[
\begin{array}{c}
\text{dist} \\
\text{prev}
\end{array}
\]

\[
\begin{array}{cccccc}
\text{a} & 4 & b & 5 & d & 6 \\
2 & 1 & 8 & 2 & 3 & z \\
& 10 & & & &
\end{array}
\]
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?

![Diagram of cities](image)

**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.
6.2 Introduction to Trees

- 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.

\[ \text{Acyclic but not connected.} \]
• 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)

$$G' = (V', E') \qquad G = (V, E)$$

**Several Spanning trees $G'$ of $G$**

**Several Spanning forests $G'$ of $G$**

---

**Kruskal Algorithm** (to find a spanning tree):

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

**Output:** a spanning tree $(V,T)$ of $G$ $\forall e \in E$

1. $T = \emptyset$
2. for each $e \in E$ {
   1. if $(V, \{e\} \cup T)$ is acyclic then
      1. $T = T \cup \{e\}$
   }
3. 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
Minimum Cost Spanning Tree