Campus Tour
Graph Assignment

Goals
- Learn and implement the adjacency matrix structure and Kruskal’s minimum spanning tree algorithm
- Understand and use the decorator pattern and various JDSL classes and interfaces

Your task
- Implement the adjacency matrix structure for representing a graph
- Implement Kruskal’s MST algorithm

Frontend
- Computation and visualization of an approximate traveling salesman tour
Adjacency Matrix Structure

- Edge list structure
- Augmented vertex objects
  - Integer key (index) associated with vertex
- 2D-array adjacency array
  - Reference to edge object for adjacent vertices
  - Null for non nonadjacent vertices
Kruskal’s Algorithm

The vertices are partitioned into clouds
- We start with one cloud per vertex
- Clouds are merged during the execution of the algorithm

Partition ADT:
- \text{makeSet}(o): create set \{o\} and return a locator for object \(o\)
- \text{find}(l): return the set of the object with locator \(l\)
- \text{union}(A,B): merge sets \(A\) and \(B\)

Algorithm \textit{KruskalMSF}(G)
\begin{itemize}
  \item \textbf{Input} weighted graph \(G\)
  \item \textbf{Output} labeling of the edges of a minimum spanning forest of \(G\)
\end{itemize}

\(Q \leftarrow \text{new heap-based priority queue}\)
\begin{itemize}
  \item for all \(v \in G.\text{vertices}()\) do
    \begin{itemize}
      \item \(l \leftarrow \text{makeSet}(v)\) \{ elementary cloud \}
      \item \(\text{setLocator}(v,l)\)
    \end{itemize}
  \end{itemize}
\begin{itemize}
  \item for all \(e \in G.\text{edges}()\) do
    \begin{itemize}
      \item \(Q.\text{insert}(\text{weight}(e), e)\)
    \end{itemize}
\end{itemize}
\begin{itemize}
  \item while \(\neg Q.\text{isEmpty}()\) do
    \begin{itemize}
      \item \(e \leftarrow Q.\text{removeMin}()\)
      \item \([u,v] \leftarrow G.\text{endVertices}(e)\)
      \item \(A \leftarrow \text{find}(\text{getLocator}(u))\)
      \item \(B \leftarrow \text{find}(\text{getLocator}(v))\)
      \item if \(A \neq B\) do
        \begin{itemize}
          \item \(\text{setMSFedge}(e)\)
          \{ merge clouds \}
          \item \(\text{union}(A,B)\)
        \end{itemize}
    \end{itemize}
\end{itemize}
Example

© 2004 Goodrich, Tamassia

Campus Tour
Example (contd.)

1. [Diagram of a network with nodes A, B, C, D, E, F, G, and H, and edges labeled with numbers from 1 to 11.]

2. [Arrow indicating "two steps" between the first and second diagrams.]

3. [Arrow indicating "four steps" between the second and third diagrams.]
Partition Implementation

Partition implementation
- A set is represented by the sequence of its elements.
- A position stores a reference back to the sequence itself (for operation \textit{find}).
- The position of an element in the sequence serves as a locator for the element in the set.
- In operation \textit{union}, we move the elements of the smaller sequence into the larger sequence.

Worst-case running times
- \textit{makeSet, find}: $O(1)$
- \textit{union}: $O(\min(n_A, n_B))$

Amortized analysis
- Consider a series of $k$ Partition ADT operations that includes $n$ \textit{makeSet} operations.
- Each time we move an element into a new sequence, the size of its set at least doubles.
- An element is moved at most $\log_2 n$ times.
- Moving an element takes $O(1)$ time.
- The total time for the series of operations is $O(k + n \log n)$.
Analysis of Kruskal’s Algorithm

- **Graph operations**
  - Methods *vertices* and edges are called once
  - Method *endVertices* is called \( m \) times

- **Priority queue operations**
  - We perform \( m \) *insert* operations and \( m \) *removeMin* operations

- **Partition operations**
  - We perform \( n \) *makeSet* operations, \( 2m \) *find* operations and no more than \( n - 1 \) *union* operations

- **Label operations**
  - We set vertex labels \( n \) times and get them \( 2m \) times

- **Kruskal’s algorithm runs in time** \( O((n + m) \log n) \) time provided the graph has no parallel edges and is represented by the adjacency list structure
Decorator Pattern

Labels are commonly used in graph algorithms
- Auxiliary data
- Output

Examples
- DFS: unexplored/visited label for vertices and unexplored/forward/back labels for edges
- Dijkstra and Prim-Jarnik: distance, locator, and parent labels for vertices
- Kruskal: locator label for vertices and MSF label for edges

The decorator pattern extends the methods of the Position ADT to support the handling of attributes (labels)
- \(\text{has}(a)\): tests whether the position has attribute \(a\)
- \(\text{get}(a)\): returns the value of attribute \(a\)
- \(\text{set}(a, x)\): sets to \(x\) the value of attribute \(a\)
- \(\text{destroy}(a)\): removes attribute \(a\) and its associated value (for cleanup purposes)

The decorator pattern can be implemented by storing a dictionary of (attribute, value) items at each position
Traveling Salesperson Problem

- A tour of a graph is a spanning cycle (e.g., a cycle that goes through all the vertices).
- A traveling salesperson tour of a weighted graph is a tour that is simple (i.e., no repeated vertices or edges) and has minimum weight.
- No polynomial-time algorithms are known for computing traveling salesperson tours.
- The traveling salesperson problem (TSP) is a major open problem in computer science.
  - Find a polynomial-time algorithm computing a traveling salesperson tour or prove that none exists.

Example of traveling salesperson tour (with weight 17)
TSP Approximation

We can approximate a TSP tour with a tour of at most twice the weight for the case of Euclidean graphs
- Vertices are points in the plane
- Every pair of vertices is connected by an edge
- The weight of an edge is the length of the segment joining the points

Approximation algorithm
- Compute a minimum spanning tree
- Form an Eulerian circuit around the MST
- Transform the circuit into a tour