Recall: Floyd’s Algorithm: All pairs shortest paths

In a weighted graph, find shortest paths between every pair of vertices

Same idea: construct solution through series of matrices $D^{(0)}$, $D^{(1)}$, … using an initial subset of the vertices as intermediaries.

![Graph diagram]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>∞</td>
<td>3</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>3</td>
<td>∞</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>

$Weight$ $matrix$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>16</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

$Distance$ $matrix$
Similar to Warshall's Algorithm

\( d_{ij}^{(k)} \) in \( D^{(k)} \) is equal to the length of shortest path among all paths from the \( i \)th vertex to \( j \)th vertex with each intermediate vertex, if any, numbered not higher than \( k \)

\[
d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} \quad \text{for } k \geq 1, d_{ij}^{(0)} = w_{ij}
\]
Pseudocode of Floyd’s Algorithm

The next matrix in sequence can be written over its predecessor

```
ALGORITHM Floyd(W[1..n,1..n])
    D ← W
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
    return D
```
Chapter 9: Greedy algorithms

Change-making problem
- Coin-system in US: 25(quarter), 10 (dime), 5(nickel), 1(penny)
- If you need to give a change of 48 cents using coins,
  - 48 cents = 1 quarter + 2 dimes + 3 pennies
  - This is a greedy algorithm: reduce the amount in the fastest way

The greedy approach constructs a solution through a sequence of steps until a complete solution is reached, On each step, the choice made must be
- *Feasible*: Satisfy the problem’s constraints
- *locally optimal*: the best choice
- *Irrevocable*: Once made, it cannot be changed later
Minimum Spanning Tree (MST)

**Motivation:** Planning the layout of cables or water pipes with the minimum length to cover all houses in a community

→ a tree structure (a connected acyclic graph)

**Spanning tree** of a connected graph $G$

- A connected acyclic subgraph of $G$ that includes all of $G$’s vertices.
- At least one spanning tree exists for $G$.

**Minimum Spanning Tree** of a weighted, connected graph $G$:

- A spanning tree of $G$ of minimum total weight.
Prim’s MST algorithm

Start with tree consisting of one vertex

“Grow” tree one vertex/edge at a time to produce MST
  • Construct a series of expanding subtrees \( T_1, T_2, \ldots \)

Greedy step: at each stage construct \( T_{i+1} \) from \( T_i \); add an edge with minimum weight connecting a vertex in tree \( (T_i) \) to one not yet in tree

For all vertices that are not yet in the tree, we have two groups
  • Fringe nodes: has an edge to at least one node in current tree \( T_i \)
  • unseen nodes: no edge to any node in \( T_i \)

A priority queue is used
  • The node with highest priority will be select
  • The priority queue will be updated every time when a new vertex is added

Algorithm stops when all vertices are included
**ALGORITHM**  \( Prim(G) \)

//Prim’s algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph \( G = (V, E) \)
//Output: \( E_T \), the set of edges composing a minimum spanning tree of \( G \)

\( V_T \leftarrow \{v_0\} \) //the set of tree vertices can be initialized with any vertex

\( E_T \leftarrow \emptyset \)

\textbf{for } i \leftarrow 1 \textbf{ to } |V| - 1 \textbf{ do}

\hspace{1em} \text{find a minimum-weight edge } e^* = (v^*, u^*) \text{ among all the edges } (v, u)

\hspace{1em} \text{such that } v \text{ is in } V_T \text{ and } u \text{ is in } V - V_T

\hspace{1em} V_T \leftarrow V_T \cup \{u^*\}

\hspace{1em} E_T \leftarrow E_T \cup \{e^*\}

\textbf{return } E_T
An Example:

Finding the MST of the following graph using Prim’s algorithm
Step 1:

Start from empty tree $T$, pick one vertex, $a(-,-)$ and add it to $T$

Priority queue: $b(a,3)$, $f(a,5)$, $e(a,6)$, $c(-,\infty)$, $d(-,\infty)$
Step 2:

Add the minimum-weight fringe edge \( b(a,3) \) into \( T \)

Priority queue: \( c(b,1), f(b,4), e(a,6), d(-,\infty) \)
Step 3:

Add the minimum-weight fringe edge $c(b,1)$ into $T$

Priority queue: $f(b,4), d(c,6), e(a,6)$
Step 4:

Add the minimum-weight fringe edge $f(b,4)$ into $T$

Priority queue: $e(f,2), d(f,5)$
Step 5:

Add the minimum-weight fringe edge $e(f,2)$ into $T$

Priority queue: $d(f,5)$
Step 6:

Add the minimum-weight fringe edge $d(f,5)$ into $T$

No remaining vertices and the algorithm is done!
### An Example

<table>
<thead>
<tr>
<th>Tree vertices</th>
<th>Priority queue for the fringe vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>a((-, -))</td>
<td>b(a,3), d(a,4), c(a,5), e((-, \infty)), f((-, \infty)), g((-, \infty)), h((-, \infty)), i((-, \infty)), j((-, \infty)), k((-, \infty)), l((-, \infty))</td>
</tr>
<tr>
<td>b(a,3)</td>
<td>e(b,3), d(a,4), c(a,5), f(b,6), g((-, \infty)), h((-, \infty)), i((-, \infty)), j((-, \infty)), k((-, \infty)), l((-, \infty))</td>
</tr>
<tr>
<td>e(b,3)</td>
<td>d(e,1), f(e,2), i(e,4), c(a,5), g((-, \infty)), h((-, \infty)), j((-, \infty)), k((-, \infty)), l((-, \infty))</td>
</tr>
<tr>
<td>d(e,1)</td>
<td>c(d,2), f(e,2), i(e,4), h(d,5), g((-, \infty)), j((-, \infty)), k((-, \infty)), l((-, \infty))</td>
</tr>
<tr>
<td>c(d,2)</td>
<td>f(e,2), g(c,4), i(e,4), h(d,5), j((-, \infty)), k((-, \infty)), l((-, \infty))</td>
</tr>
<tr>
<td>f(e,2)</td>
<td>g(c,4), i(e,4), h(d,5), j(f,5), k((-, \infty)), l((-, \infty))</td>
</tr>
<tr>
<td>g(c,4)</td>
<td>h(g,3), i(e,4), j(f,5), k(g,6), l((-, \infty))</td>
</tr>
<tr>
<td>h(g,3)</td>
<td>i(e,4), j(f,5), k(g,6), l((-, \infty))</td>
</tr>
<tr>
<td>i(e,4)</td>
<td>j(i,3), l(i,5), k(g,6)</td>
</tr>
<tr>
<td>j(i,3)</td>
<td>l(i,5), k(g,6)</td>
</tr>
<tr>
<td>l(i,5)</td>
<td>k(g,6)</td>
</tr>
<tr>
<td>k(g,6)</td>
<td></td>
</tr>
</tbody>
</table>
An Example

The MST consists of the edges $ab$, $be$, $ed$, $dc$, $ef$, $cg$, $gh$, $ei$, $ij$, $il$, and $gk$
**Lemma:** Let $T_{i-1}$ be part of the minimum spanning tree $T$, which contains a subset of the vertices of $G(X)$. Let edge $e$ be the smallest-weight edge connecting $X$ (tree $T_{i-1}$ ) to $G - X$ (remaining vertices). Then $e$ (minimum-weight fringe edge) is part of the MST

**Proof:** Using contradiction, suppose $e = (u, v)$ is not part of MST. Then there is another edge $e' = (u', v')$ between $X$ and $G - X$ and belongs to MST. Replace $e'$ by $e$ will result in a spanning tree with smaller total weight than MST. Contradiction!
Notes on Prim’s algorithm

To locate the minimum-weight fringe edge, we can use the heap structure. But here we use the min-heap, where the root has a key smaller than both children.

- Construct the min-heap -- $O(|V|)$
- Delete the min -- $O(\log |V|)$, it can be performed $|V|-1$ times
- Verify minimum weight from any remaining vertex to the tree -- this may be performed $|E|$ times. Each verification may result in a key priority change in the heap, which takes $O(\log |V|)$.
- Therefore, the total complexity is $O[|V|+(|V|-1+|E|)\log |V|]=O(|E| \log |V|)$
MinHeap and Prim’s Algorithm

\( b(a, 3), f(a, 5), e(a, 6), c(-\infty), d(-\infty) \)

\( c(b, 1), f(b, 4), e(a, 6), d(-\infty) \)

\( f(b, 4), d(c, 6), e(a, 6) \)
Dijkstra’s Algorithm – Single-Source Shortest Paths

Single-source short paths problem – find the shortest path starting from a given vertex to any other vertex

Example: hub airports for airplane planning

Using greedy strategy to find the single-source shortest paths

• In Floyd’s algorithm, we find the all-pair shortest paths, which may not be necessary in many applications
• Certainly, all-pair shortest paths contain the single-source shortest paths. But Floyd’s algorithm has \(O(|V|^3)\) complexity!

There are many algorithms that can solve this problem, here we introduce the Dijkstra’s algorithm

Note: Dijkstra’s algorithm only works when all the edge-weight are nonnegative.
Dijkstra’s Algorithm on Undirected Graph

Similar to Prim’s MST algorithm, with the following difference:

• Start with tree consisting of one vertex – **source**
• “grow” tree one vertex/edge, which has minimum length of path, at a time to produce spanning tree
  – Construct a series of expanding subtrees $T_1$, $T_2$, …
• Keep track of shortest path from source to each of the vertices in $T_i$
• at each stage **construct** $T_{i+1}$ from $T_i$: add minimum weight edge connecting a vertex in tree ($T_i$) to one not yet in tree
  – choose from “fringe” nodes
  – (this is the “greedy” step!)
• algorithm stops when all vertices are included
Example:

*Find the shortest paths starting from vertex a*
Step 1:

Tree vertices: $a(-,0)$

Priority queue: $b(a,3), d(a,7), c(-,\infty), e(-,\infty)$
Step 2:

Tree vertices: $a(-,0)$, $b(a,3)$,

Priority queue: $d(a,7) \rightarrow d(b,3+2)$, $c(-,\infty) \rightarrow c(b,3+4)$, $e(-,\infty)$
Step 3:

Tree vertices: \(a(-0), b(a,3), d(b,5)\)

Priority queue: \(c(b,3+4), e (-\infty) \rightarrow e(d,5+4)\)
Step 4:

Tree vertices: $a(-,0), b(a,3), d(b,5), c(b,7)$

Priority queue: $e(d,9)$
Step 5:

Tree vertices: a(-,0), b(a,3), d(b,5), c(b,7), e(d,9)

Remaining vertices: none → the algorithm is done!
Output the Single-Source Shortest Paths

Tree vertices: \( a(-,0), b(a,3), d(b,5), c(b,7), e(d,9) \)

- from a to b: a-b of length 3
- from a to d: a-b-d of length 5
- from a to c: a-b-c of length 7
- from a to e: a-b-d-e of length 9