Each node is one of two types:
- source (supply) node
- destination (demand) node

Every arc has:
- its tail at a supply node
- its head at a demand node

Such a graph is called bipartite.

Notoriously not planar.
Transportation problem in which

- Equal number of supply and demand nodes.
- Every supply node has a supply of one.
- Every demand node has a demand for one.
- Each supply node is connected to every demand node (called a complete bipartite graph).
- Solution is required to be all integers.

Notes:

- These problems are very common.
- They are notoriously degenerate ($2n$ constraints but only $n$ nonzero flows).
Shortest Paths Problem

Given:

- Network: \((\mathcal{N}, \mathcal{A})\)
- Costs = Travel Times: \(c_{ij}, (i, j) \in \mathcal{A}\)
- Home (root): \(r \in \mathcal{N}\)

Problem: Find shortest path from every node in \(\mathcal{N}\) to root.
Network Flow Formulations

First Thought...

• Put \( b_i = \begin{cases} 1 & i = \text{starting point} \\ -1 & i = \text{destination} \end{cases} \)

• Solve min-cost network flow problem.

• Shortest path from source to destination: follow tree arcs.

• Highly degenerate. Most tree arcs have zero flow.

A Better Method

• Put \( b_i = \begin{cases} 1 & i \neq r \\ -(m - 1) & i = r \end{cases} \)

• Shortest path from \( i \) to \( r \): follow tree arcs.

• Length (of time) of shortest path = \( y_r^* - y_i^* \).

Notation Used in Following Algorithms

NOTE: NOT COVERED IN CLASS

• Put \( v_i = \text{minimum time from } i \text{ to } r \)
  – Called \( \text{label} \) in networks literature.
  – Called \( \text{value} \) in dynamic programming literature.
• **Bellman’s Equation = Principle of Dynamic Programming**

\[
v_r = 0 \\
v_i = \min\{c_{ij} + v_j : (i, j) \in A\} \\
T = \{(i, j) \in A : v_i = c_{ij} + v_j\} \quad \text{– not necessarily a tree}
\]

• **Method of Successive Approximation**

  – Let \( k \) denote an iteration counter.
  – Fix root node’s value to zero for all iterations: \( v_r^{(k)} = 0 \) for all \( k \).
  – For all other nodes...
    * Initialize: \( v_i^{(0)} = \infty \).
    * Iterate: \( v_i^{(k+1)} = \min\{c_{ij} + v_j^{(k)} : (i, j) \in A\} \quad i \neq r \).
    * Stop: when a pass leaves \( v_i \)’s unchanged.

• **Complexity**

  – \( v_i^{(k)} \) = length of shortest path having \( k \) or fewer arcs.
  – Requires at most \( m - 1 \) passes.
  – \( n \) adds/compares per pass.
  – \( mn \) operations in total.
Label Setting Algorithm = Dijkstra’s Algorithm

Notations:

• $F = \text{set of finished nodes (labels are set)}$.
• $h_i, i \in \mathcal{N} = \text{next node to visit after } i \text{ (heading)}$.

Dijkstra’s Algorithm:

• Initialize:

\[
F = \emptyset, \quad v_j = \begin{cases}
0 & j = r \\
\infty & j \neq r
\end{cases}
\]

• Iterate:

– While unfinished nodes remain, select the one with smallest $v_k$. Call it $j$. Add it to set of finished nodes $F$.
– For each unfinished node $i$ having an arc connecting it to $j$:
  * If $c_{ij} + v_j < v_i$, then set
    \[
    v_i = c_{ij} + v_j \\
    h_i = j
    \]
• Each iteration finishes one node: \( m \) iterations
• Work per iteration:
  – Selecting an unfinished node:
    * Naively, \( m \) comparisons.
    * Using appropriate data structures, a heap, \( \log m \) comparisons.
  – Update adjacent arcs.
• Overall: \( m \log m + n \).