GRAPH ALGORITHMS

- Introduction to graphs - representations
- Single source shortest path: Dijkstra's algorithm
- Minimum cost spanning tree: Prim's algorithm
- All source shortest paths

Definition: A graph $G = (V, E)$ consists of a set V of vertices and a set E of edges. The elements of E are pairs (u, v) with $u, v \in V$. If the pairs are ordered then the graph is directed, otherwise it is undirected.

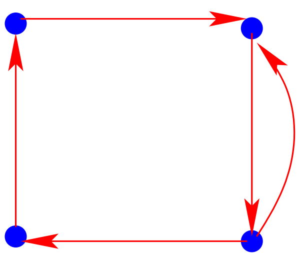
Terminology:

- Digraph = Directed graph.
- When $(u, v) \in E$, we say that u and v are adjacent and that the edge (u, v) is incident to u and v.

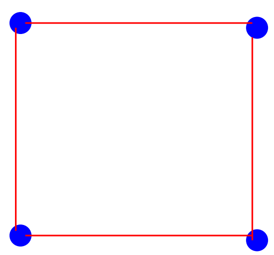
A graph $G = (V, E)$ is a representation of a certain binary relation. If R is a binary relation between elements in V then, we can represent it by a graph $G = (V, E)$ as follows:

$$(u, v) \in E \leftrightarrow u\ R\ v$$

- Undirected graph $\leftrightarrow$ symmetric relation.



First graph: (1) R (2); (4) R (1); (2) R (3); (3) R (2); (3) R (4);



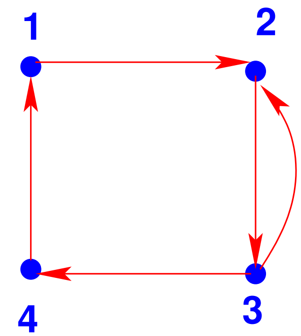
Second graph: (1) R (2); (2) R (3); (3) R (4); (4) R (1).

Graphs – Adjacency matrix representation

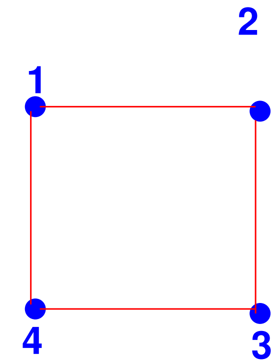
- Assume that there are n vertices. Then the adjacency matrix is an $n \times n$ matrix, with

$$a_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{Otherwise} \end{cases}$$

$$\begin{bmatrix} & 1 & & \\ & & 1 & \\ & 1 & & 1 \\ 1 & & & \end{bmatrix}$$



$$\begin{bmatrix} & 1 & & 1 \\ 1 & & 1 & \\ & 1 & & 1 \\ 1 & & 1 & \end{bmatrix}$$



- Matrix is symmetric when graph is undirected
- OK scheme but wasteful for sparse graphs
- More on sparse matrices later.

Graphs – shortest part algorithm

Definition: A weighted graph $G = (V, E, W)$ is a graph in which each edge is weighted, i.e., each edge has an associated weight. The length of a path is the sum of the weights of all the edges in the path.

➤ The weights are usually positive numbers. [but can also be nonpositive in some applications]

Problem: Given a node s, find the shortest paths from s to all other nodes in the weighted graph G.

➤ Called One-source shortest path problem

➤ Another problem: Find shortest path from any vertex to any other vertex

Graphs – Dijkstra's algorithm

- Idea of shortest path algorithm very similar to breadth-first-search.
- Good implementation for sparse graphs: Priority Queue

<u>Differences with BFS:</u>

- Need distances from starting node. Update these distances as we do the traversal;
- Always take the next node to be removed from queue to be the one with smallest distance.

- We will consider simple implementations for dense graphs

ALGORITHM : 1. *Shortest_Path(G,r)*

Initialize:

1. For each $v \in V$ set:
2. $d[v] = 0$ if $v == r$ and $d[v] = \infty$ otherwise.
3. Set $V_T = \emptyset$.

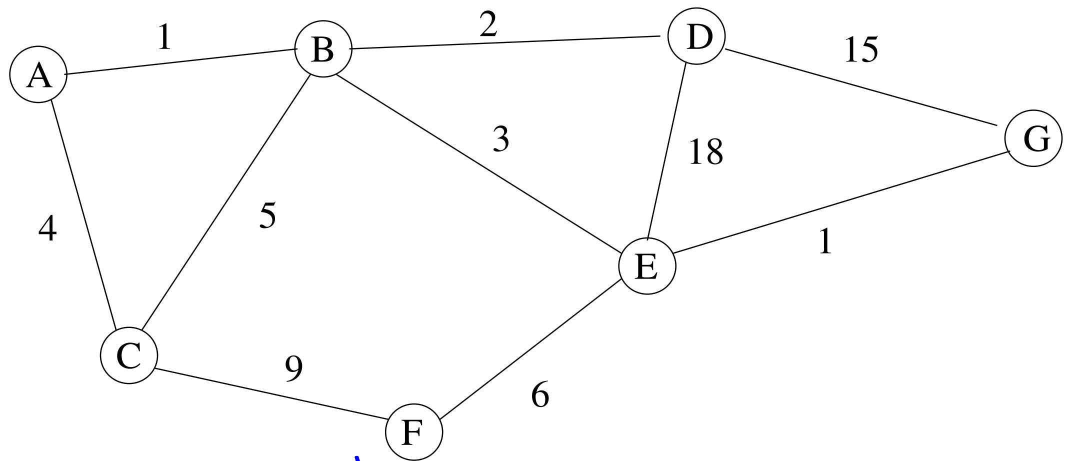
Iterate:

4. While $V_T \neq V$ do
5.   Find u s.t. $d[u] = \min[d[v], v \in V - V_T]$
6. $V_T = V_T \cup \{u\}$
7.   For each $v \in V - V_T$ set:
8. $d[v] = \min[d[v], d[u] + w(u, v)]$
9.   End
10. EndWhile

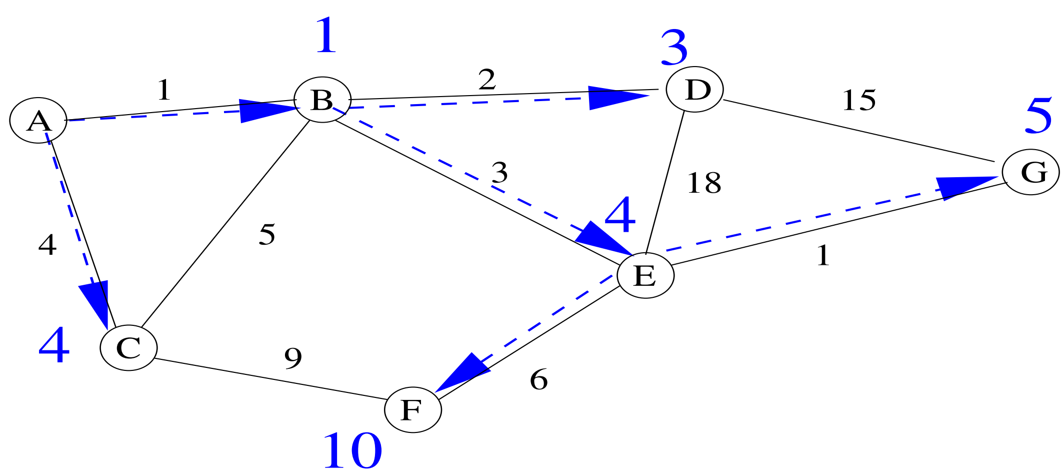
- Cost: $O(n^2)$.

Dijkstra's Algorithm – Example

Original Graph



Resulting Tree & Distances

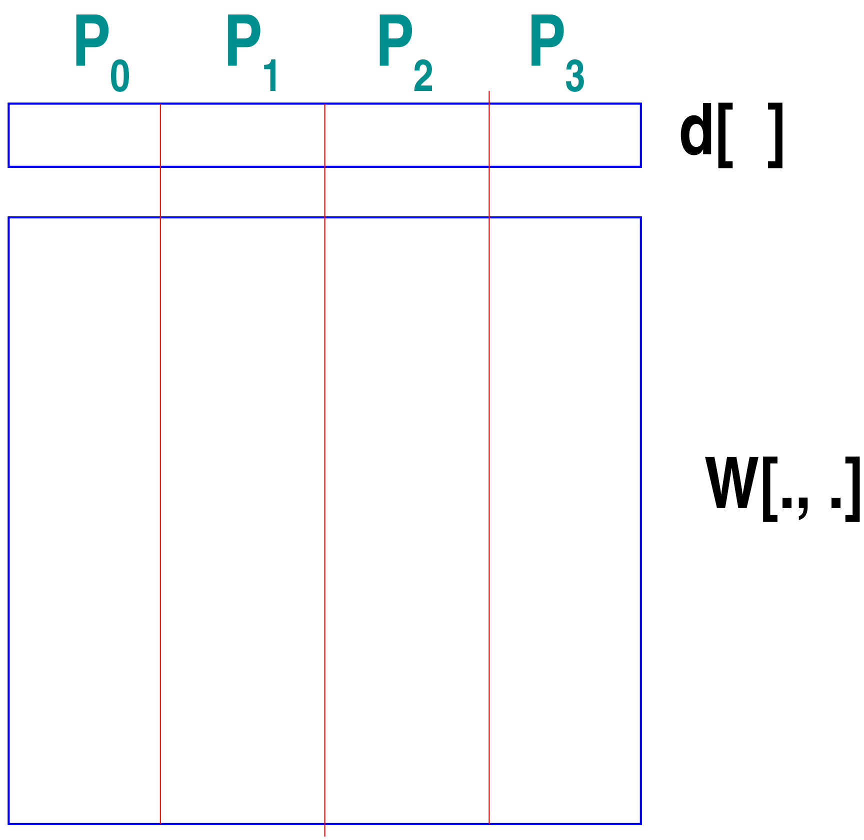


Dijkstra's Algorithm – Parallel Implementation

- First observation: Difficult to parallelize the while loop..
- Fairly easy to parallelize costlier steps of while loop within each iteration.

Decomposition:

- Split Distance array in p parts, uniformely.
- Split weight matrix column-wise in p blocks
- Goal: should get cost down from $O(n^2)$ to $O(n^2/p)$



➤ Main steps – Assume we are at step $\boldsymbol{k}$.

* Each process finds its min. distance.

* Reduction to find global min.

* Process which achieves min. broadcasts the node $\boldsymbol{u}$ that achieves min

* Also marks $\boldsymbol{u}$ as a Tree-node

* All processes updates their distances.

➤ Line 5 of Algorithm: Requires computing a local min. and doing a reduction operation. Cost of k-th step:

$$\frac{(n-k)}{p}\omega + \log(p)(t_s + t_w)$$

➤ Line 6: Broadcast of $u, d(u)$ to all. Cost:

$$\log(p)(t_s + 2 * t_w)$$

➤ Lines 7-8-9: require no communication. But update itself costs $\frac{n-k}{p}\omega$ (Assuming $V - V_T$ uniformly distributed each time)

➤ Total (Order only) $\Theta(n^2/p) + \Theta(n\log(p))$

➤ Cost-optimal if $p = O(n/\log(n))$.

✍1 Show the above result.

Minimum Cost Spanning Tree (Undirected Graphs)

Definitions: A spanning tree of a graph $G = (V, E)$ is a connected subgraph $T = (V_T, E_T)$ of G, which is a tree and whose vertices are all the vertices of G, i.e., $V_T = V$. The cost of T is the sum of the weights of all edges e of the tree,

$$Cost(T) = \sum_{e \in E_T} w(e)$$

Problem: Given a weighted graph find its minimum cost spanning tree. (MCST)

- Easy to see that the MCST must indeed be a tree.

- Applications:

- Minimum cost transit system: want to link all localities in a given city; but would like the total of all distances over all route segments to be minimum.
- Network of computers: need to broadcast a message to all nodes in a network from arbitrary nodes. The minimum cost spanning tree allows to do so in best time on the average

- Two solutions to the problem:

1. Prim's algorithm: almost identical with Dijkstra's shortest path algorithm;
2. Kruskal's algorithm: Adds one edge at a time, in increasing order of weight.

Minimum Cost Spanning Tree: Prim's Algorithm

ALGORITHM : 2. *Prim(G,r)*

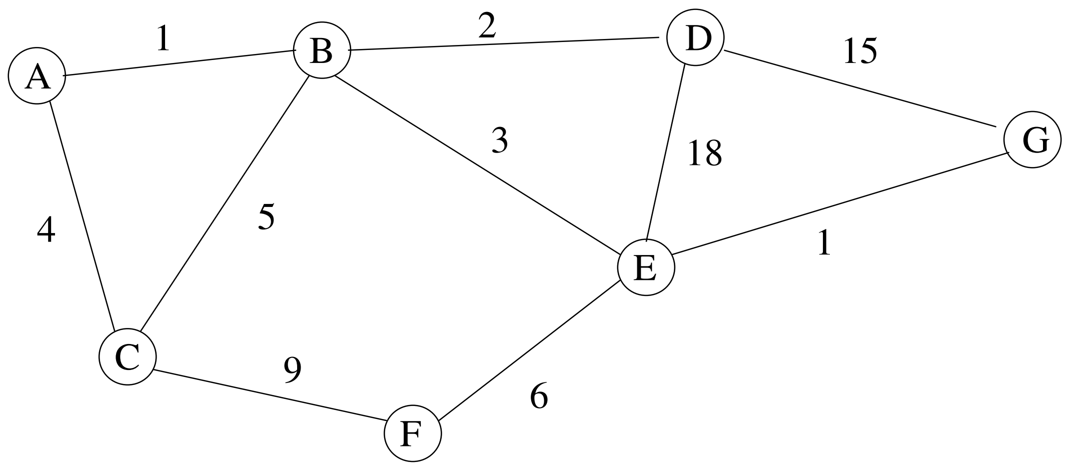
Initialize:

1. For each $v \in V$ set:
2. $d[v] = 0$ if $v == r$ and $d[v] = \infty$ otherwise.
3. Set $V_T = \emptyset$.

Iterate:

4. While $V_T \neq V$ do
5. Find u s.t. $d[u] = \min[d[v], v \in V - V_T]$
6. $V_T = V_T \cup \{u\}$
7. For each $v \in V - V_T$ set:
8. $d[v] = \min[d[v], w(u, v)]$ $\leftarrow$ Only Change from Dijkstra
9. End
10. EndWhile

Prim's Algorithm – Example



Step	Tree	Pseudo-Distances
0	$\emptyset$	$[\mathbf{0}, \infty, \infty, \infty, \infty, \infty, \infty]$
1	A	$[\quad, \mathbf{1}, \mathbf{4}, \infty, \infty, \infty, \infty]$
2	A B	$[\quad, \quad, \mathbf{4}, \mathbf{2}, \mathbf{3}, \infty, \infty]$
$\vdots$		
$\vdots$		
7	A B C D E F G	

Prim's Algorithm – Parallel implementation

- Cost = identical with Dijkstra's algorithm
- Parallel Implementation = identical with Dijkstra's algorithm

The all-pairs Shortest path problem

The problem:

Find the shortest path between any pair of vertices i and j

➤ Can be solved by using the shortest path algorithm from each node in turn. Cost $= O(n^3)$.

➤ Another solution: Floyd's algorithm [also referred to as Floyd-Warshall algorithm] – whose cost is also $O(n^3)$.

➤ Builds incrementally shortest paths between i and j where all intermediate vertices are in the set

$$S_k = \{1, 2, \cdots, k\}.$$

Observation:

Shortest path throuh S_k = either shortest path throuh S_{k-1} or shortest path from i to k followed by shortest path from k to j through S_{k-1}. Hence,

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k == 0 \\ \min[d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}] & \text{if } k \geq 1 \end{cases}$$

- Algorithm: compute these distances for $k = 1, ..., n$

- Computation can be done in place [i.e., only one matrix is needed.] This is because k-th column (and row) of $D^{(k)}$ does not change from $D^{(k-1)}$ [set $i = k$ and then $j = k$ in above formulas]

ALGORITHM : 3. *Floyd(G)*

0. $\boldsymbol{D^{(0)} = W}$
1. For $\boldsymbol{k = 1 : n}$ Do:
2. For $\boldsymbol{i = 1 : n}$ Do:
3. For $\boldsymbol{j = 1 : n}$ Do:
4. $\boldsymbol{d_{ij}^{(k)} = \min[d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}]}$
5. End
6. End
7. End

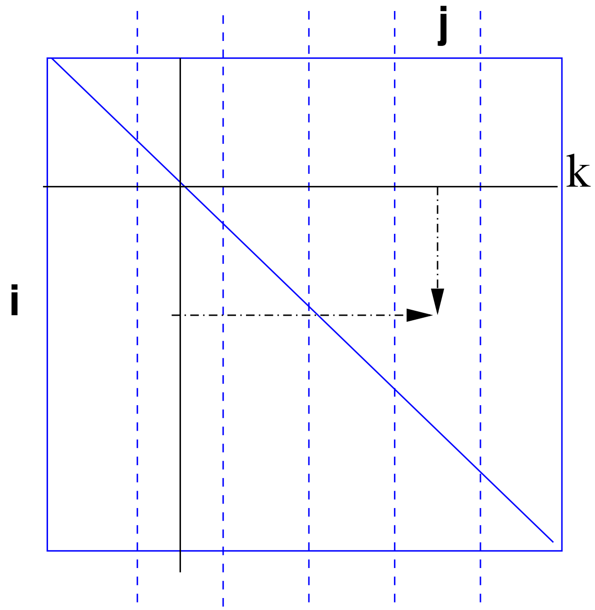
➤ Note: computation pattern somewhat similar to Gaussian Elimination.

✍2 Explore these similarities

➤ Like GE we can define a broadcast version and a pipelined version of the algorithm.

- Can devise a row-based algorithm with broadcasts [No need to interleave rows into processors for better load balance] – Can also devise a pipelined row algorithm -
- Can devise 2-D mapping generalizations of the above two options.

Illustration of Floyd's algorithm



d(i,j) = min (d(i,j), d(i,k)+d(k,j))