Greedy Technique

CS3230: Design and Analysis of Algorithms

Roger Zimmermann

National University of Singapore

Spring 2017

Chapter 9: Greedy Technique

- “ ‘Greed,’ for lack of a better word, is good!”
  - Michael Douglas, US actor in the role of Gordon Gecko, in the film *Wall Street*, 1987.

Change Making Problem

- How to make 48 cents of change using coins of denominations of 25 cents (quarter), 10 cents (dime), 5 cents (nickel), and 1 cent (penny) so that the total number of coins is the smallest?
- The idea:
  - Make the locally best choice at each step
- A: 1 quarter, 2 dimes, and 3 pennies

- Q: Is the solution optimal?

Greedy Strategy (1)

- A greedy algorithm makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
- The choice made at each step must be:
  - Feasible
    - Satisfy the problem's constraints
  - Locally optimal
    - Be the best local choice among all feasible choices
  - Irrevocable
    - Once made, the choice cannot be changed on subsequent steps

Greedy Strategy (2)

- Q: Do greedy algorithms always yield optimal solutions?
  - Example of 46 cents: change making problem with a denomination set of 7, 5, and 1?

- Additional reference: MIT Textbook.

Example: Coin Denominations [Dr. Ecco; Dennis Shasha]

- How would you design the coin denominations for a country, assuming that the average number of coins needed for a purchase should be as small as possible?
- Notation: $NC(v)$ represents the number of coins for a specific value v.
- Need to be able to represent from 1 cent to 99 cents.
- Example 1: 2 coins $\Rightarrow$ 1 cent and 10 cents.
  - E.g.: $NC(71) = 8$, $NC(99) = 18$.
  - The average number of coins would be:
  - $NC_{avg} = (\sum_{v=1}^{99} NC(v))/99 = 9.1$.

Example: Coin Denominations [Dr. Ecco; Dennis Shasha]

- Q1: What would be the best set of denominations that consist of:
  - 3 coin values, including a 1 cent coin?
  - 4 coin values, including a 1 cent coin?
  - 5 coin values, including a 1 cent coin?
  - 6 coin values, including a 1 cent coin?
- Q2: With the optimal coin denominations (i.e., the average number of coins needed for a purchase is minimal), does a greedy strategy to count out money work?
- Assumption: The number of cents of all purchase prices is uniformly distributed between $1, \ldots, 99$.

Example: Coin Denominations [Dr. Ecco; Dennis Shasha]

– 3 coin values: 1, 5, 22 or 1, 5, 23, $NC_{avg} = 5.3131\ldots$.
  Greedy strategy works.
– 3 coin values: 1, 12, 19, $NC_{avg} = 5.2020\ldots$.
  Greedy strategy does not work.
– 4 coin values: 1, 3, 11, 37 or 1, 3, 11, 38, $NC_{avg} = 4.1414\ldots$.
  Greedy strategy works.
– 4 coin values: 1, 5, 18, 25, $NC_{avg} = 3.9292\ldots$.
  Greedy strategy does not work.
– 5 coin values: 1, 3, 7, 16, 40, $NC_{avg} = 3.4949\ldots$.
  Greedy strategy works.
– 5 coin values: 1, 5, 16, 23, 33, $NC_{avg} = 3.3232\ldots$.
  Greedy strategy does not work.

Applications of Greedy Strategy

- Optimal solutions:
  - Change making, but only with some denominations
  - Minimum Spanning Tree (MST)
  - Single-source shortest paths
  - Huffman codes

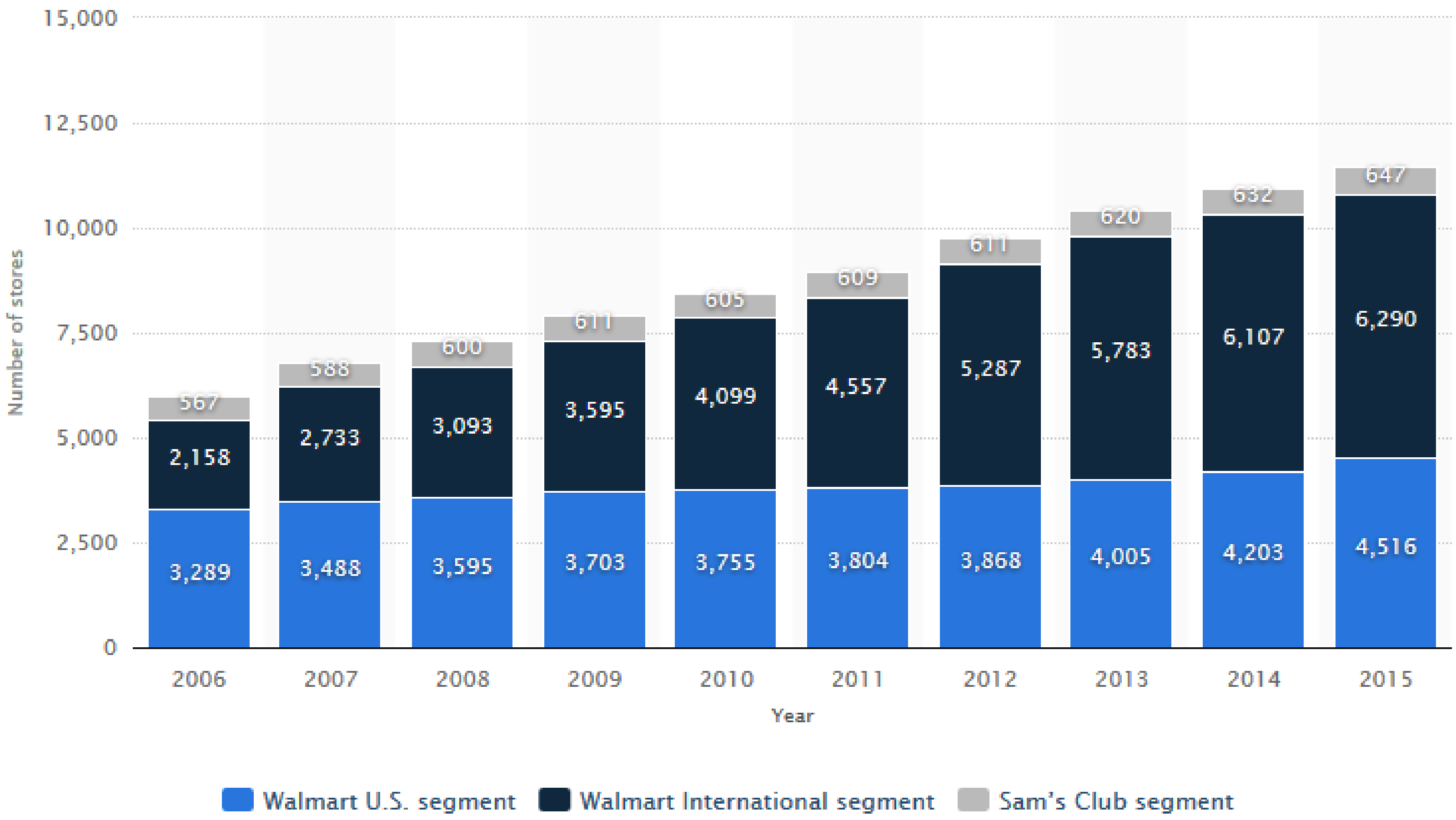
- Approximations:
  - Traveling Salesman Problem (TSP)
  - Knapsack problem
  - Other optimization problems

Approximations

- Q: Why may it be interesting to have a slightly “incorrect” (*i.e.*, approximate) solution?
- Tradeoff: time versus accuracy.
- Some problems are difficult to solve within acceptable time for any practical input size.
- Examples:
  - Class of $\mathcal{NP}$-*hard* problems
  - Weather forecast
  - Computations on very large data sets: e.g., data warehousing

Approximations

- Ex.: Walmart data warehousing. Chart shows no. of stores.



Minimum Spanning Tree (MST)

- Spanning Tree of a connected graph G: a connected, acyclic subgraph (tree) of G that includes all of G's vertices.

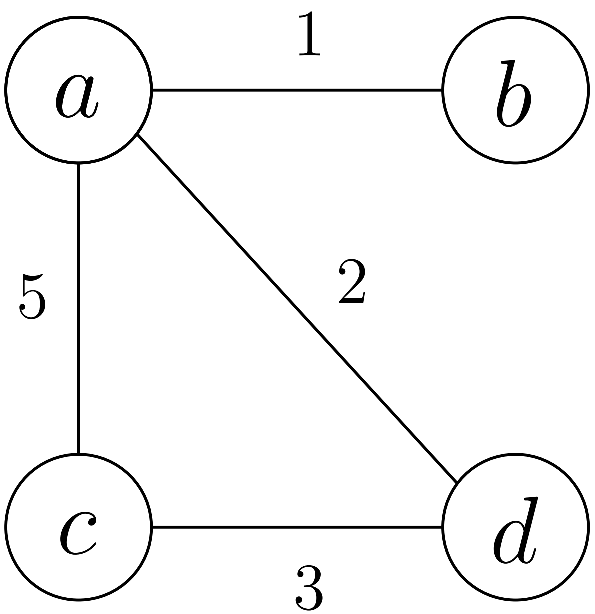
- Minimum Spanning Tree of a weighted, connected graph G: a spanning tree of G of minimum total weight.

Other Spanning Trees (MST)

- k-Minimum Spanning Tree (k-MST) is a tree that spans some subset of k vertices in the graph with minimum weight.
- Euclidean Minimum Spanning Tree is a spanning tree of a graph with edge weights corresponding to the Euclidean distance between vertices.
- Degree-constrained Minimum Spanning Tree (d-MST) is minimum spanning tree whose degree at every vertex is limited by some constraint. The d-MST problem is $\mathcal{NP}$-complete.

Example Spanning Tree

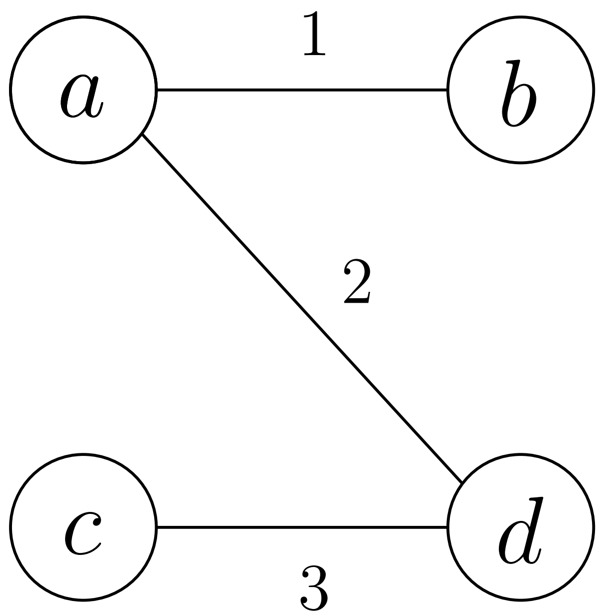
- Graph and its spanning trees; T_1 is the minimum spanning tree.



Graph

Example Spanning Tree

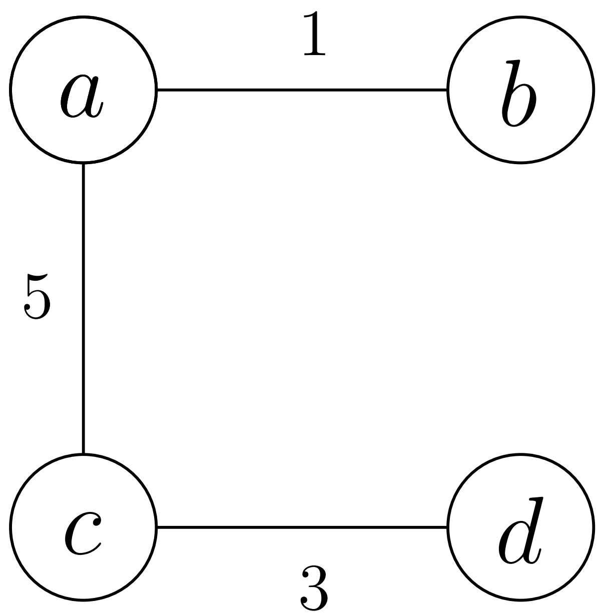
- Graph and its spanning trees; T_1 is the minimum spanning tree.



$$w(T_1) = 6$$

Example Spanning Tree

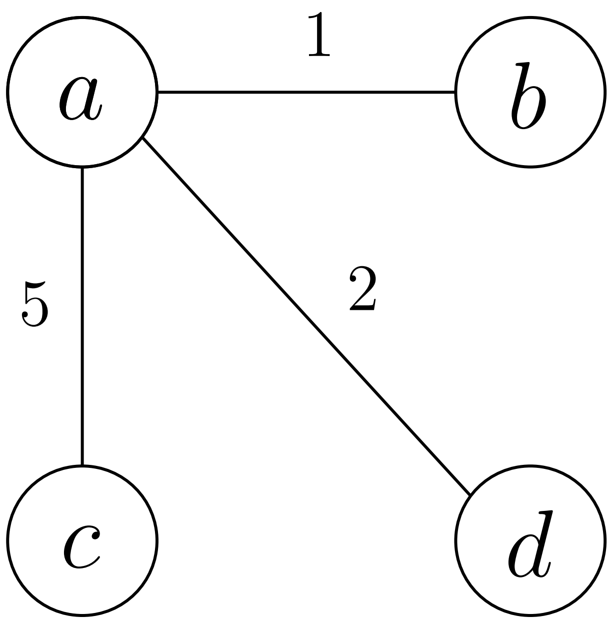
- Graph and its spanning trees; T_1 is the minimum spanning tree.



$$w(T_2) = 9$$

Example Spanning Tree

- Graph and its spanning trees; T_1 is the minimum spanning tree.



$$w(T_3) = 8$$

Prim's MST Algorithm

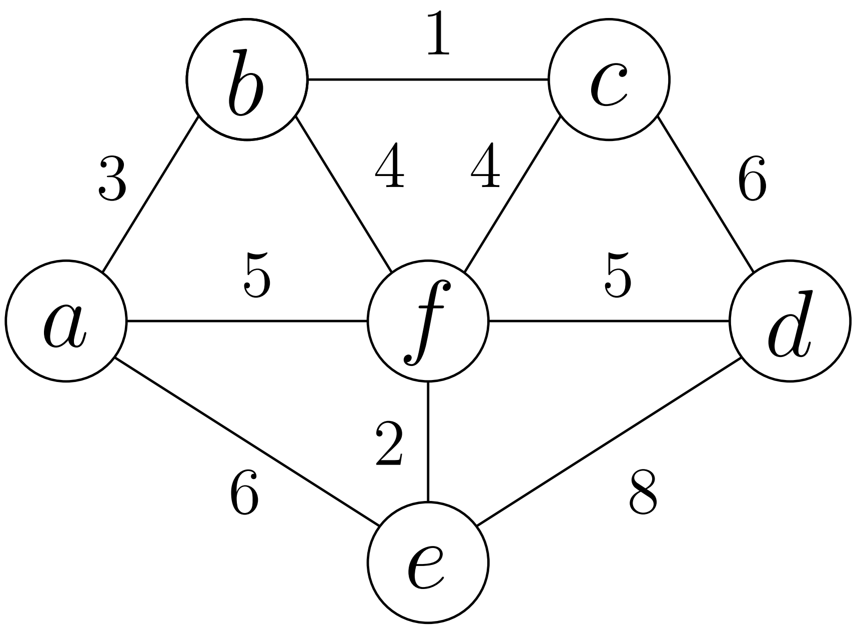
- Start with a tree, T_0, consisting of one vertex.
- "Grow" the tree one vertex/edge at a time.
  - Construct a series of expanding subtrees $T_1, T_2, \ldots, T_{n-1}$.
  - At each stage: construct T_{i+1} from T_i in a greedy manner by attaching to it the nearest vertex not in that tree.
  - Definition of *nearest vertex*: a vertex not in the tree connected to a vertex in the tree by an edge of the smallest weight.
- Algorithm stops when all vertices are included.

Prim's MST Algorithm

PRIM (G)
// Input: a weighted connected graph $G = \langle V, E \rangle$
// Output: E_T, the set of edges composing the MST of G
$V_T \leftarrow \{v_0\}$
$E_T \leftarrow \emptyset$
for $i \leftarrow 1$ to $|V| - 1$ do
find a minimum-weight edge $e^* = (v^*, u^*)$ among all edges (v, u) such that v is in V_T and u is in $V - V_T$
$V_T \leftarrow V_T \cup \{u^*\}$
$E_T \leftarrow E_T \cup \{e^*\}$
od
return E_T

Example Prim

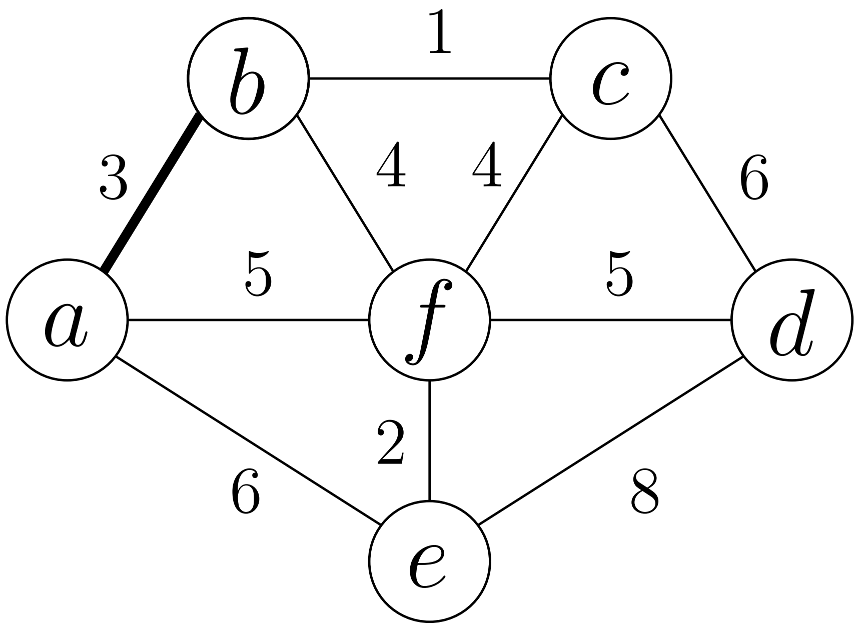
- Starting vertex: a



Graph

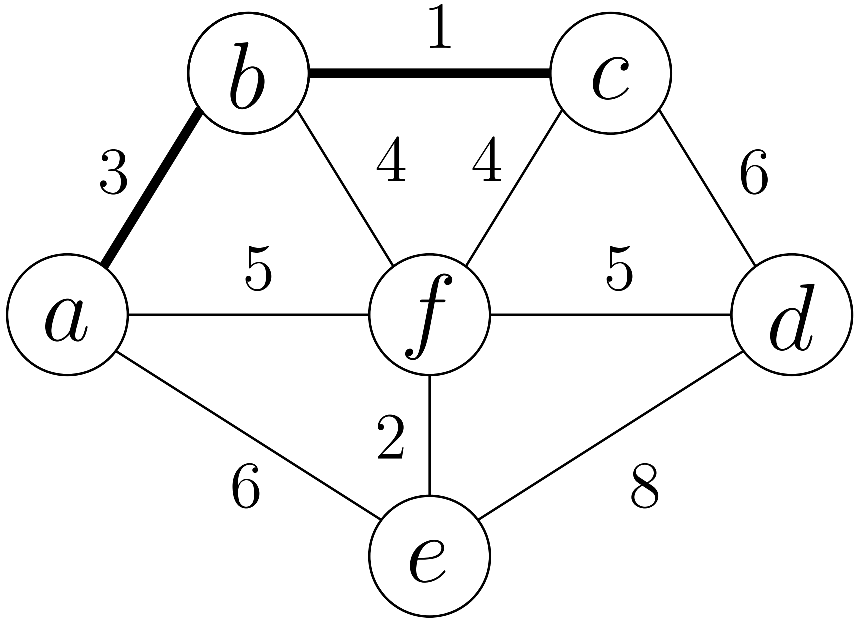
Example Prim

- Tree vertices: a(–,–)
- Remaining vertices: b(a,3), c(–,∞), d(–,∞), e(a,6), f(a,5)



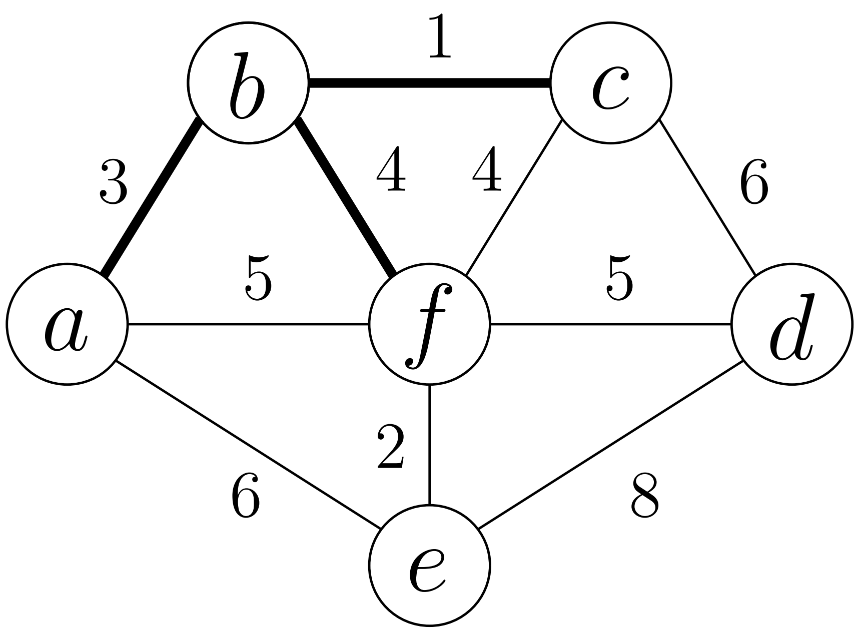
Example Prim

- Tree vertices: b(a,3)
- Remaining vertices: c(b,1), d(–,∞), e(a,6), f(b,4)



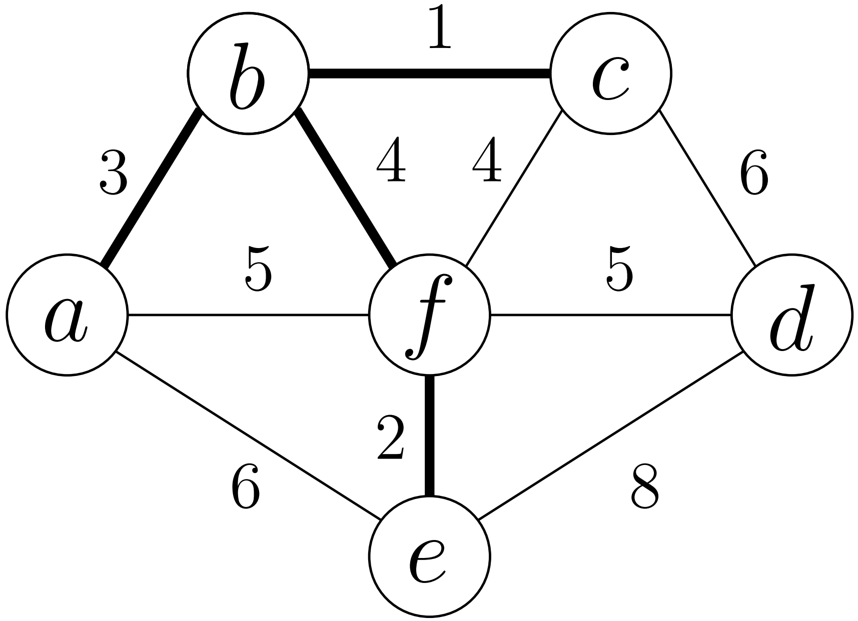
Example Prim

- Tree vertices: c(b,1)
- Remaining vertices: d(c,6), e(a,6), f(b,4)



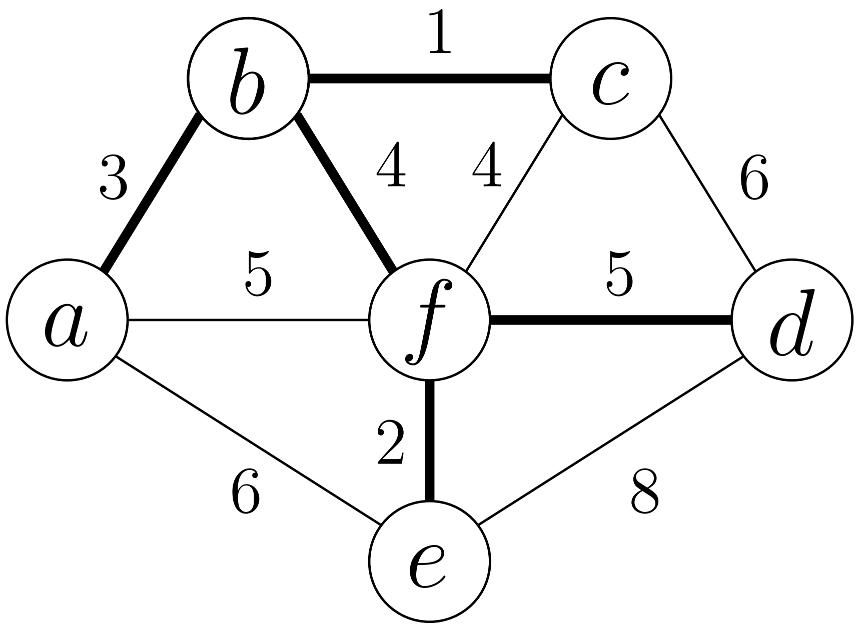
Example Prim

- Tree vertices: f(b,4)
- Remaining vertices: d(f,5), e(f,2)



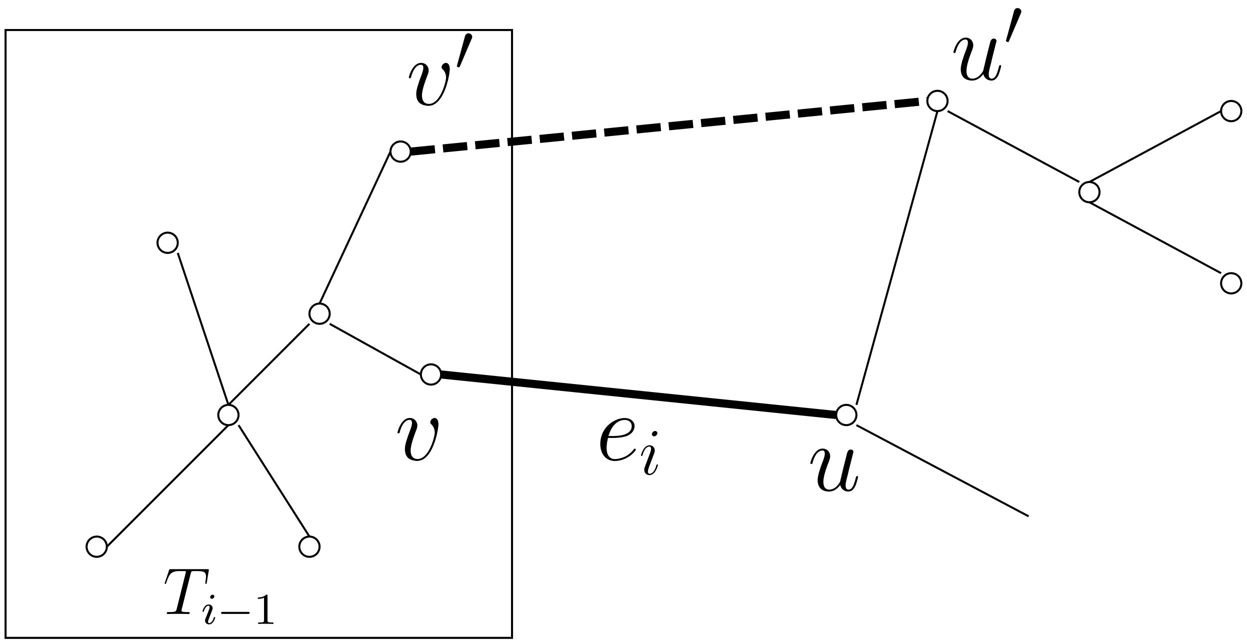
Example Prim

- Tree vertices: e(f,2)
- Remaining vertices: d(f,5)



Correctness

- Proved by induction and contradiction, page 312 of textbook.



Proof Sketch (1)

- Prove by induction that each of the subtrees $T_i, i = 0, \ldots, n-1$, generated by Prim's algorithm is a part (i.e., a subgraph) of some miminum spanning tree.
- Assume T_{i-1} is part of some minimum spanning tree T.
- Need to prove that T_i (generated with Prim's) is also part of a minimum spanning tree.
- Proof by contradiction: assume no spanning tree contains T_i.
- Let $e_i = (v, u)$ be the minimum weight edge from a vertex in T_{i-1} to a vertex not in T_{i-1}, used by Prim's.
- By our assumption e_i cannot belong to the minimum spanning tree T.

Proof Sketch (2)

- Therefore, if we add e_i to T, a cycle must be formed.
- This cycle must contain another edge (v', u') connecting a vertex $v' \in T_{i-1}$ to a vertex u' that is not in T_{i-1}.
- If we now delete edge (v', u') from this cycle, we obtain another spanning tree whose weight is less than or equal to the weight of T since the weight of e_i is less than or equal to the weight of (v', u').
- Hence this spanning tree is a minimum spanning tree, which contradicts our assumption that no minimum spanning tree contains T_i.

Efficiency

- Locating the nearest vertex:
  - Use unordered array to store the priority queue: $\Theta(|V|^2)$
  - Use min-heap to store the priority queue: $(|V| - 1 + |E|)O(\log |V|) = O(|E| \log |V|)$

Another Greedy Algorithm for MST: Kruskal

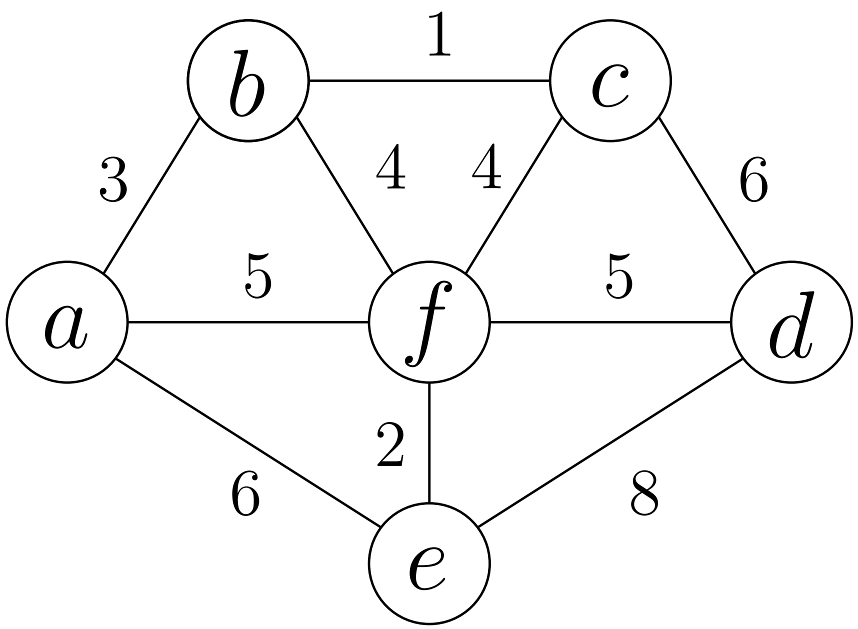
- Edges are initially sorted by increasing weight.
- Start with a forest of $|V|$ isolated vertices.
- “Grow” MST one edge at a time
  - Intermediate stages have forest of trees (not connected).
- At each stage add minimum weight edge among those not yet used that does not create a cycle.
  - At each stage the edge may:
    - Expand an existing tree.
    - Combine two existing trees into a single tree.
    - Create a new tree.
- Algorithms stops when all vertices are included.

Kruskal's Algorithm

Kruskal (G)
// Input: a weighted connected graph $G = \langle V, E\rangle$
// Output: E_T, the set of edges composing the MST of G
Sort E in non-decreasing order of the edge weights
$w(e_{i_1}) \leq \ldots \leq w(e_{i_{|E|}})$
$E_T \leftarrow \emptyset$, $ecounter \leftarrow 0$, $k \leftarrow 0$
while $ecounter < |V| - 1$ do
$k \leftarrow k + 1$
if $E_T \cup \{e_{i_k}\}$ is acyclic
$E_T \leftarrow E_t \cup \{e_{i_k}\}$; $ecounter \leftarrow ecounter + 1$
od
return E_T

Example Kruskal

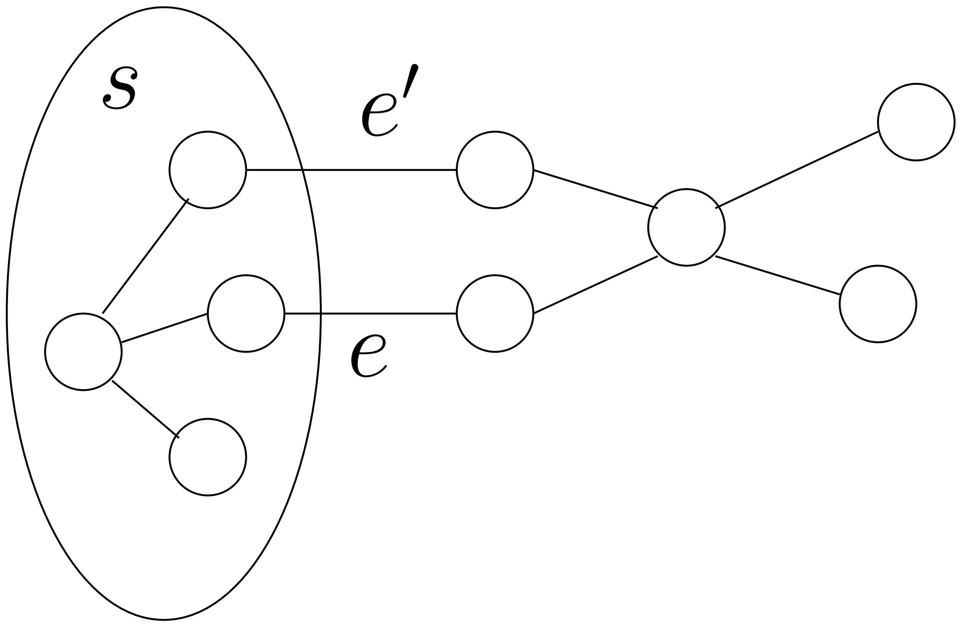
- Sorted edges: bc 1, ef 2, ab 3, bf 4, cf 4, af 5, df 5, ae 6, cd 6, de 8



Graph

Correctness

- $w(e)$: the smallest

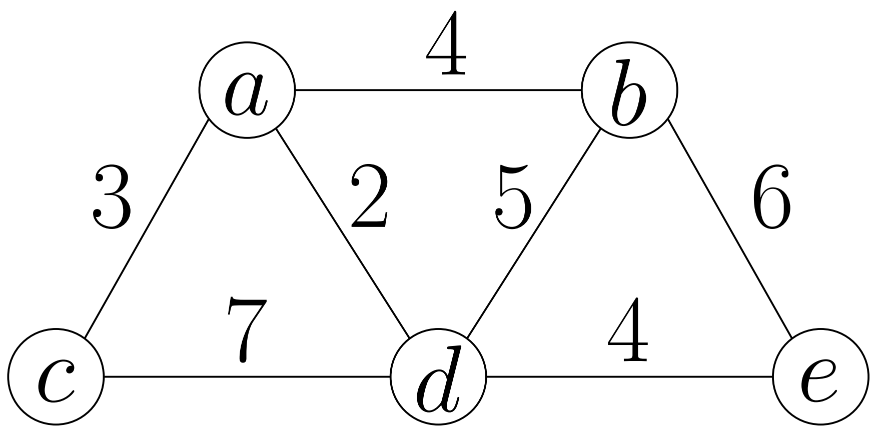


Efficiency

- Start from a forest with $|V|$ isolated vertices, find next edge (u, v) from the sorted edge list, if u and v belong to 2 different trees, add edge.
- Union find algorithm (page 317)
  $O(n + m \log n)$, n unions, m found.
- Kruskal's algorithm is dominated by the time sorting the edges by their weight:
  $O(|E| \log |E|)$

Shortest Paths – Dijkstra's Algorithm

- Shortest Path Problems:
  - All pair shortest paths (Floyd's algorithm).
  - Single source shortest path problem (Dijkstra's algorithm): Given a weighted graph G, find the shortest paths from a source vertex s to each of the other vertices.



Prim's and Dijkstra's Algorithms

- Generate different kinds of spanning trees.
  - Prim's: a minimum spanning tree.
  - Dijkstra's: a spanning tree rooted at a given source s, such that the distance from s to every other vertex is the shortest.
- Different greedy strategies:
  - Prim's: always choose the closest (to the tree) vertex in the priority Q to add to the expanding tree V_T.
  - Dijkstra's: always choose the closest (to the source) vertex in the priority queue Q to add to the expanding tree V_T.

Prim's and Dijkstra's Algorithms

- Different labels for each vertex:
  - Prim's: parent vertex and the distance from the tree to the vertex.
  - Dijkstra's: parent vertex and the distance from the source to the vertex.

Shortest Paths – Dijkstra's Algorithm

- Dijkstra's algorithm: Similar to Prim's MST algorithm, with the following difference:
  - Start with a tree, T_0, consisting of one vertex.
  - "Grow" the tree one vertex/edge at a time.
    - Construct a series of expanding subtrees $T_1, T_2, \ldots$
  - 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 edge (u^*, u) with the lowest $d_{u^*} + w(u^*, u)$ connecting a vertex in tree (T_i) to one not yet in the tree.
    - Choose from "fringe" edges ($\Rightarrow$ "greedy" step!).
- Algorithm stops when all vertices are included.

Dijkstra's Algorithm (1)

DIJKSTRA (G, s)
// Input: a weighted connected graph $G = \langle V, E \rangle$ and a
// source vertex s.
// Output: The length d_v of the shortest path from s to v
// and its penultimate vertex p_v for every vertex v in V.
INITIALIZE(Q) // Initialize vertex priority queue to empty.
for every vertex v in V do
    $d_v \leftarrow \infty; p_v \leftarrow$ **null**
    INSERT(Q, v, d_v) // Init. vertex priority in the priority queue.
od
$d_s \leftarrow 0$; DECREASE(Q, s, d_s) // Update priority of s with d_s.
$V_T \leftarrow \emptyset$

Dijkstra's Algorithm (2)

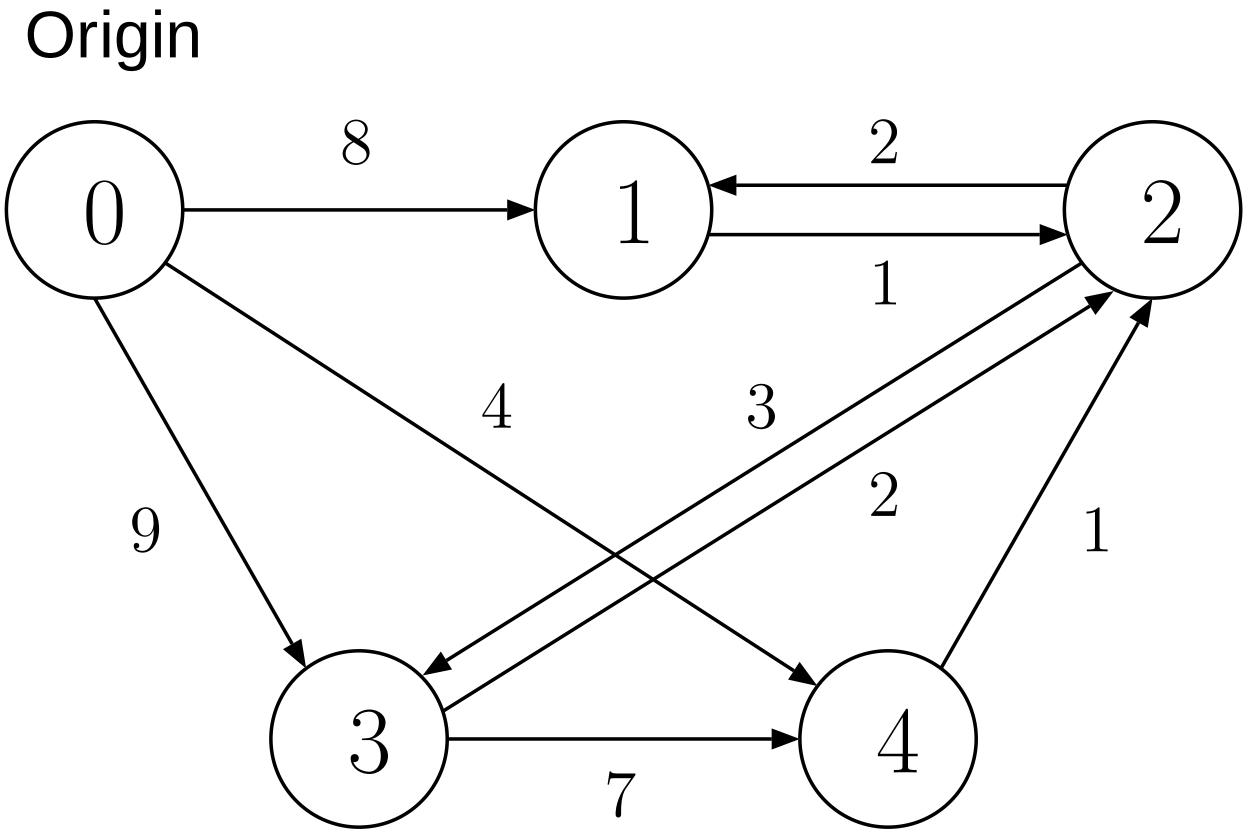
// DIJKSTRA (G, s) (cont.)
for $i \leftarrow 0$ to $|V| - 1$ do
$u^* \leftarrow$ DELETEMIN(Q) // Delete the min. priority element.
$V_T \leftarrow V_T \cup \{u^*\}$
for every vertex u in $V - V_T$ that is adjacent to u^* do
if $d_{u^*} + w(u^*, u) < d_u$
$d_u \leftarrow d_{u^*} + w(u^*, u); p_u \leftarrow u^*$
DECREASE(Q, u, d_u) // Update priority of u with d_u.
fi
od
od

Dijkstra's Algorithm (3)

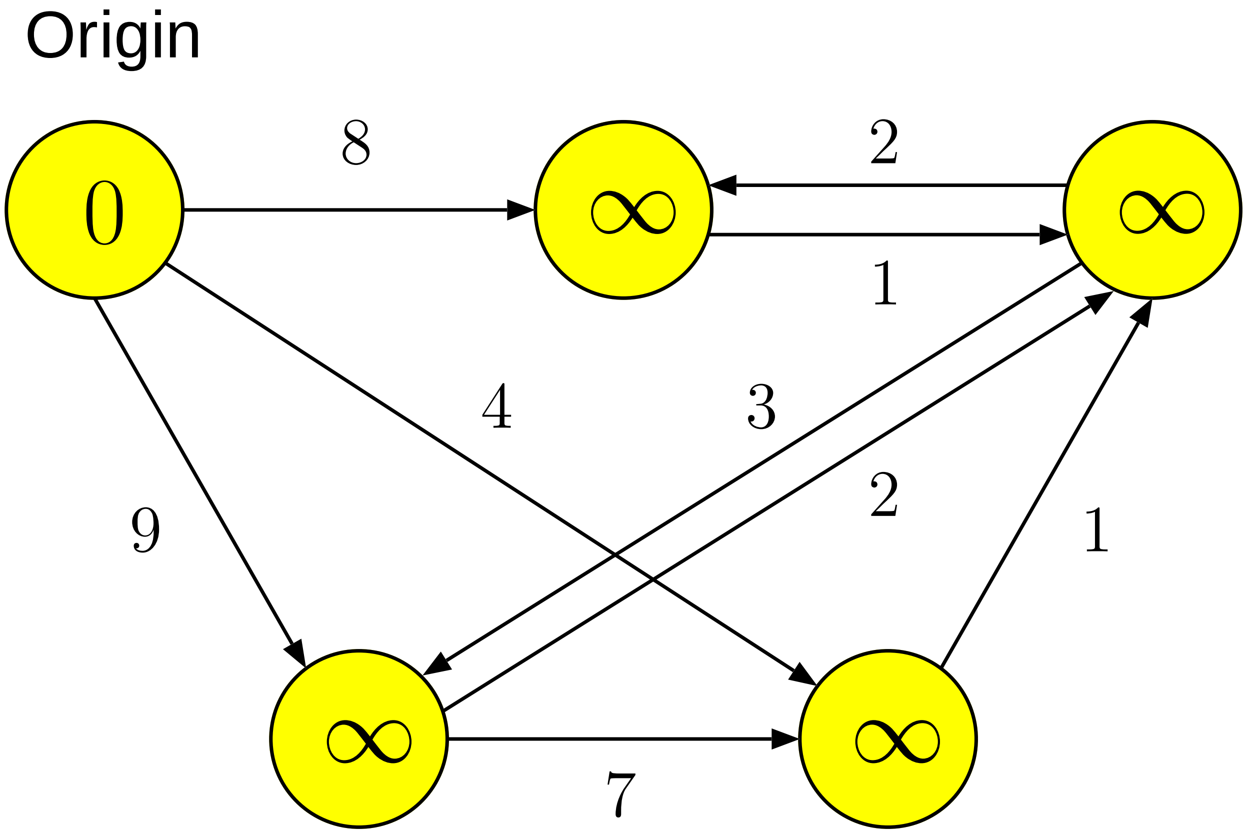
- Pseudo-code algorithm:

color all vertices yellow
for each vertex w do
    $distance(w) \leftarrow \infty$
$distance(s) \leftarrow 0$
while there are yellow vertices do
    $v \leftarrow$ yellow vertex with min $distance(v)$
    color v red
    for each neighbor w of v
        $relax(v, w)$

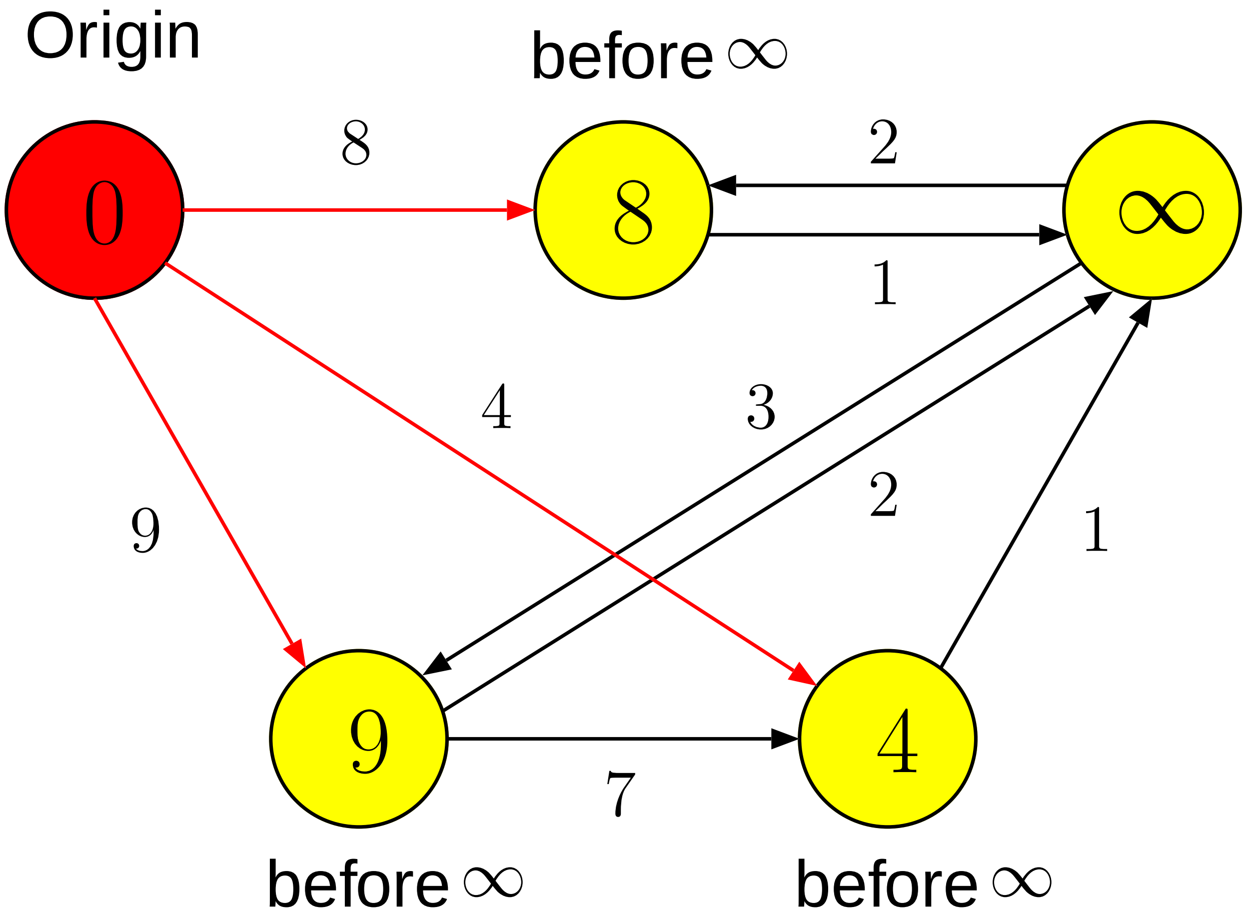
Example – Dijkstra's Algorithm



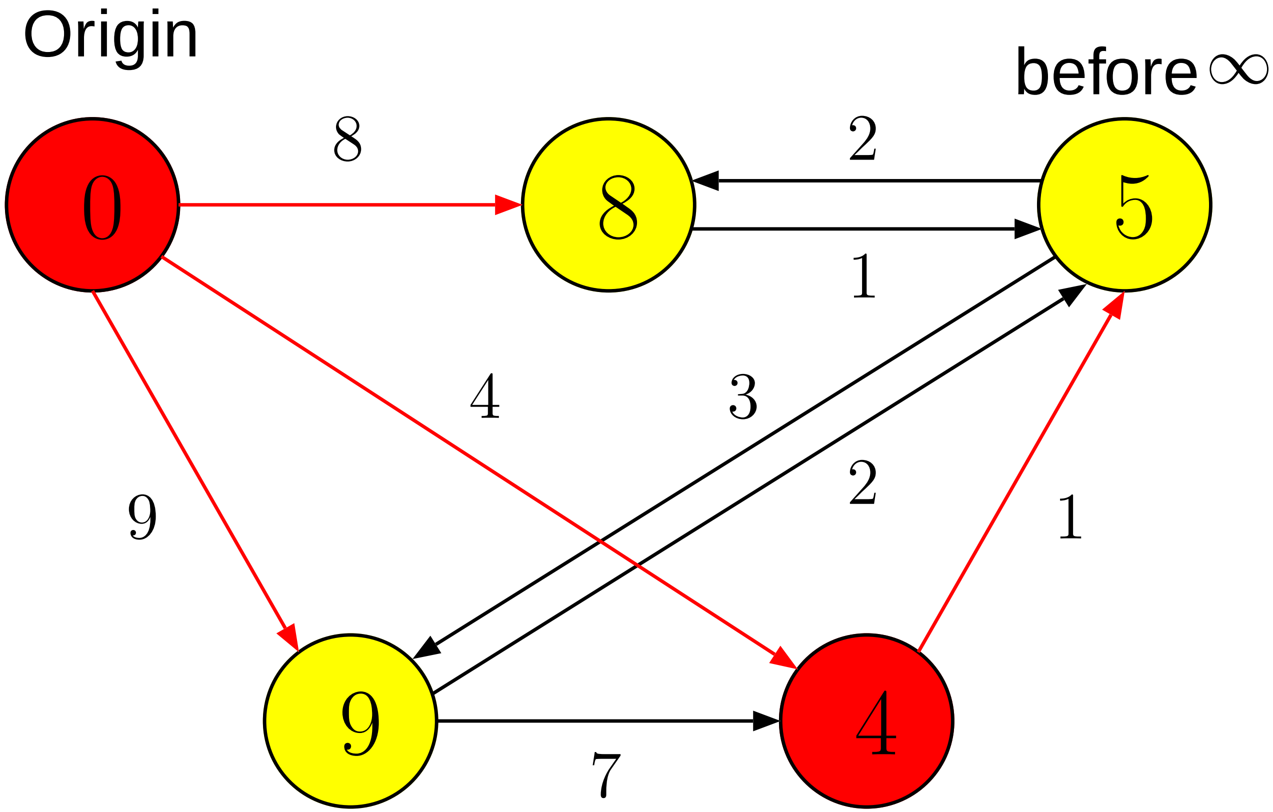
Example – Dijkstra's Algorithm



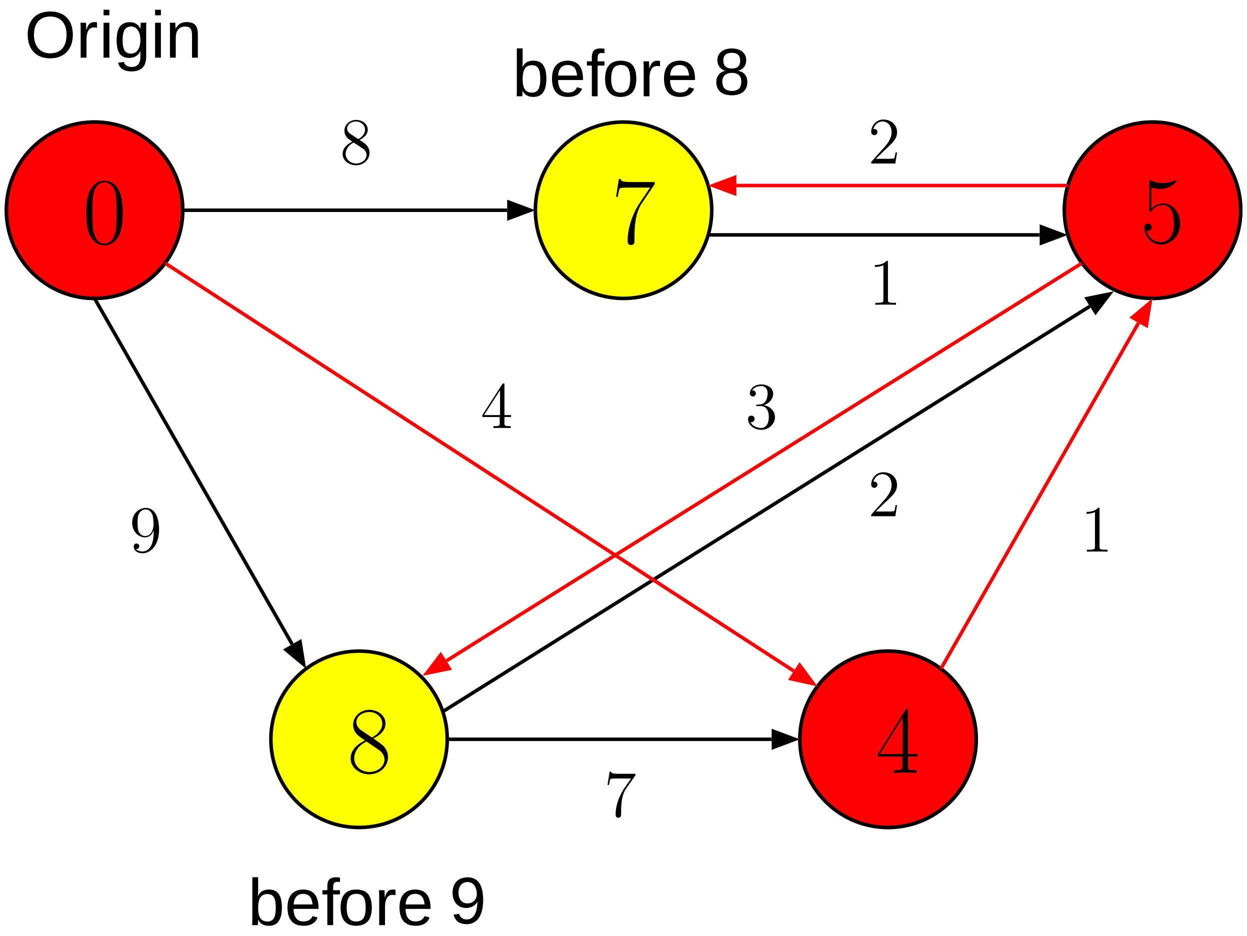
Example – Dijkstra's Algorithm



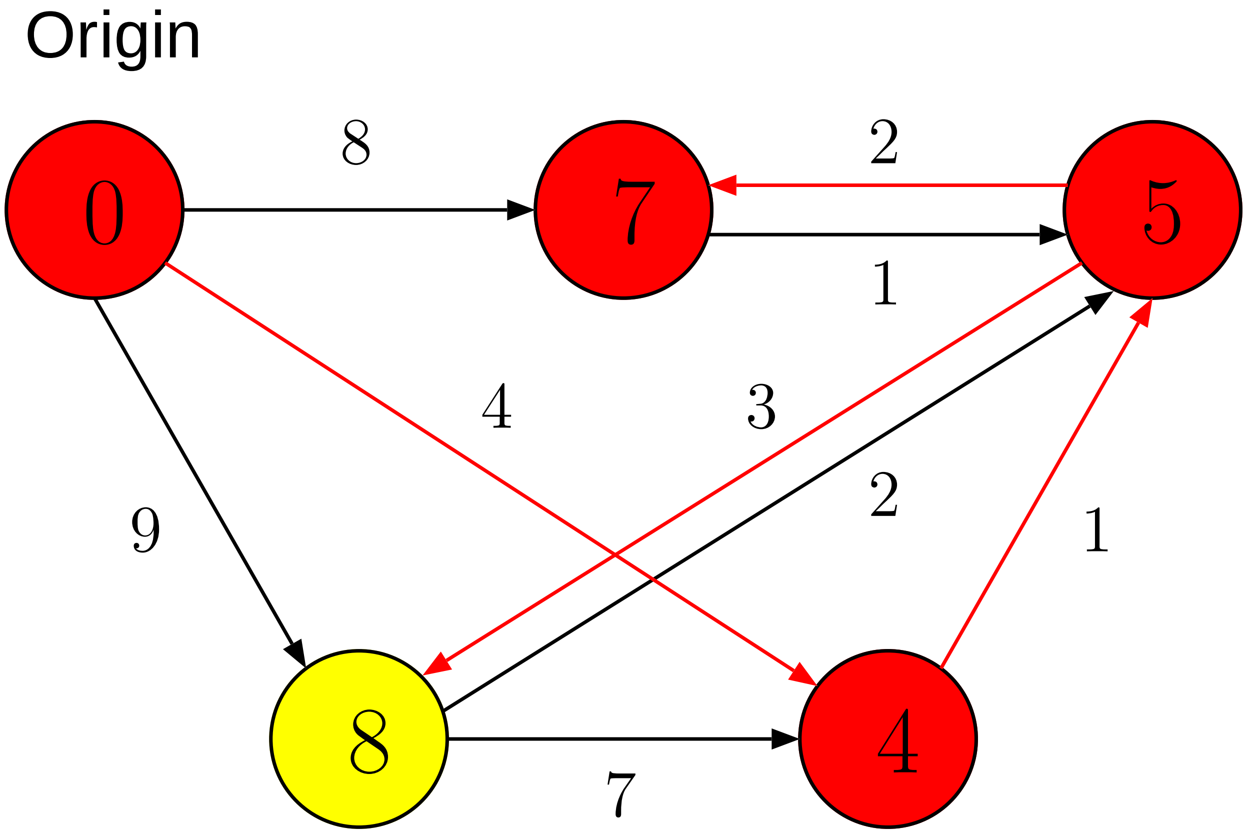
Example – Dijkstra's Algorithm



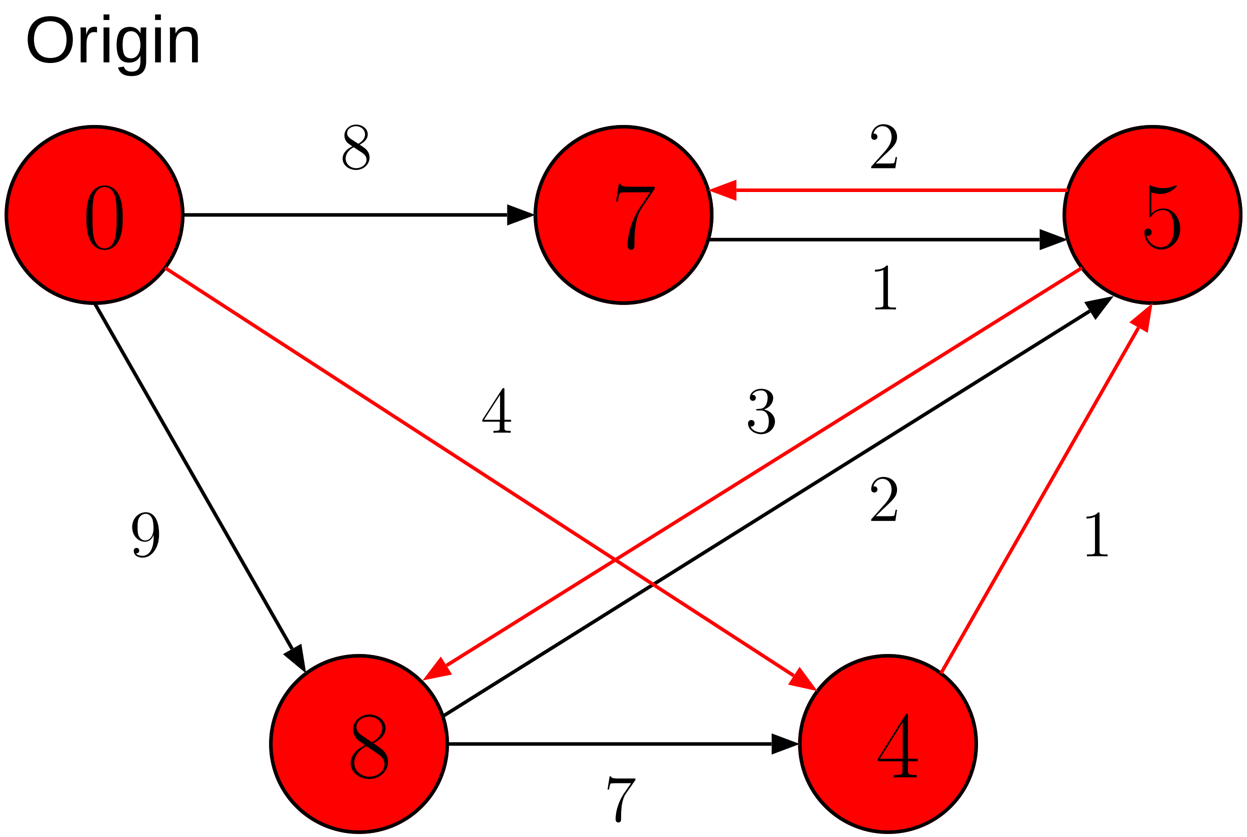
Example – Dijkstra's Algorithm



Example – Dijkstra's Algorithm

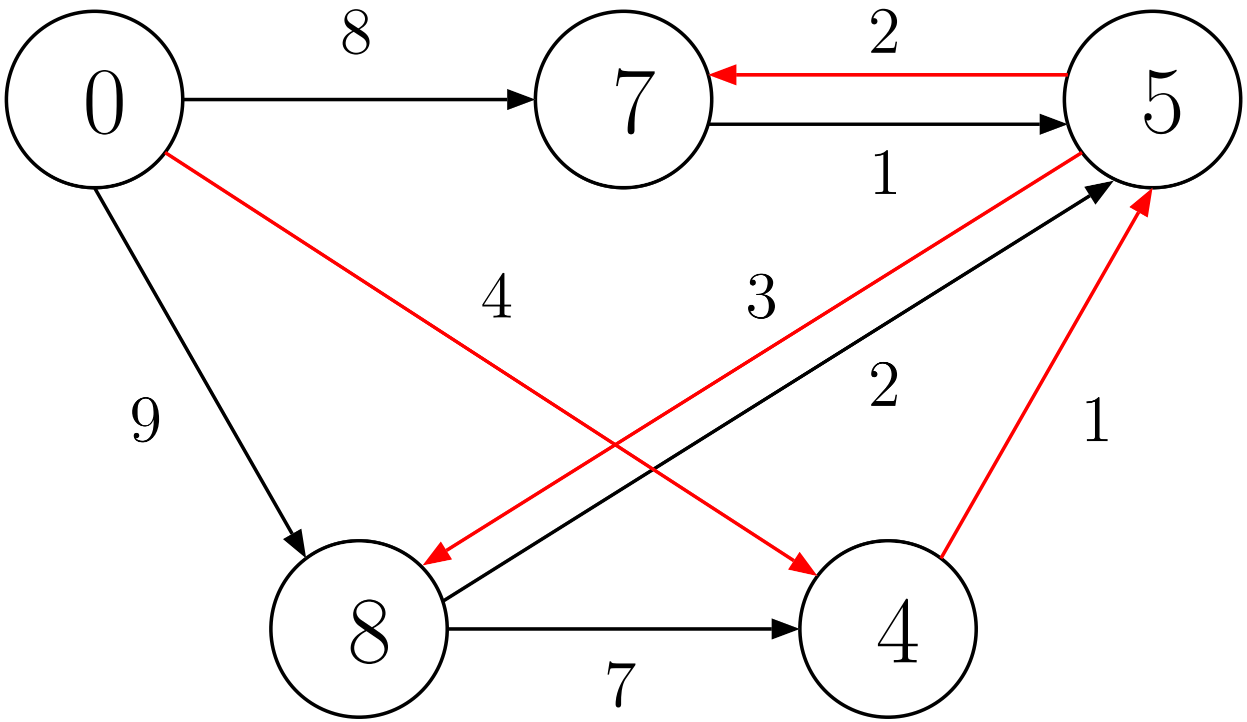


Example – Dijkstra's Algorithm



Example – Dijkstra's Algorithm

- Final solution:



Efficiency

- Using weight matrix: $\Theta(|V|^2)$.
- Using adjacency list and min-heap: $O(|E| \log |V|)$.

Some Notes on Dijkstra's Algorithm

- Used in Internet routing protocols, such as OSPF (Open Shortest Path First).
- OSPF is used in interior gateway protocols (e.g., within an enterprise).
- OSPF detects changes in topology (e.g., link failures) and creates a loop-free routing topology very quickly.

Take Away Message on Greedy Techniques

- A greedy algorithm makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
- The choice made at each step must be:
  - Feasible
    - Satisfy the problem's constraints
  - Locally optimal
    - Be the best local choice among all feasible choices
  - Irrevocable
    - Once made, the choice cannot be changed on subsequent steps