Algorithm and Data Structure Analysis (ADSA)

Minimum Spanning Trees

Properties of MSTs

An MST of a given graph G can be constructed by greedy algorithms.

Crucial properties:

- Cut property (Let e be an edge of minimum cost in a cut C. Then there is an MST that contains e)
- Cycle property (an edge of maximal cost in any cycle does not need to be considered for computing an MST)

Jarnik-Prim Algorithm

- Similar to Dijkstra’s algorithm for the single-source shortest path problem.
- Start with an arbitrary node s of V.
- Let S be the set of already connected nodes.
- In the beginning S={s} holds.
- Insert in each iteration an edge of minimal cost that connects a node u of S to a node v not contained in S (it’s an edge of minimal cost in this cut).
- Add v to S and continue until all nodes are contained in S.

Jarnik-Prim Algorithm Implementation

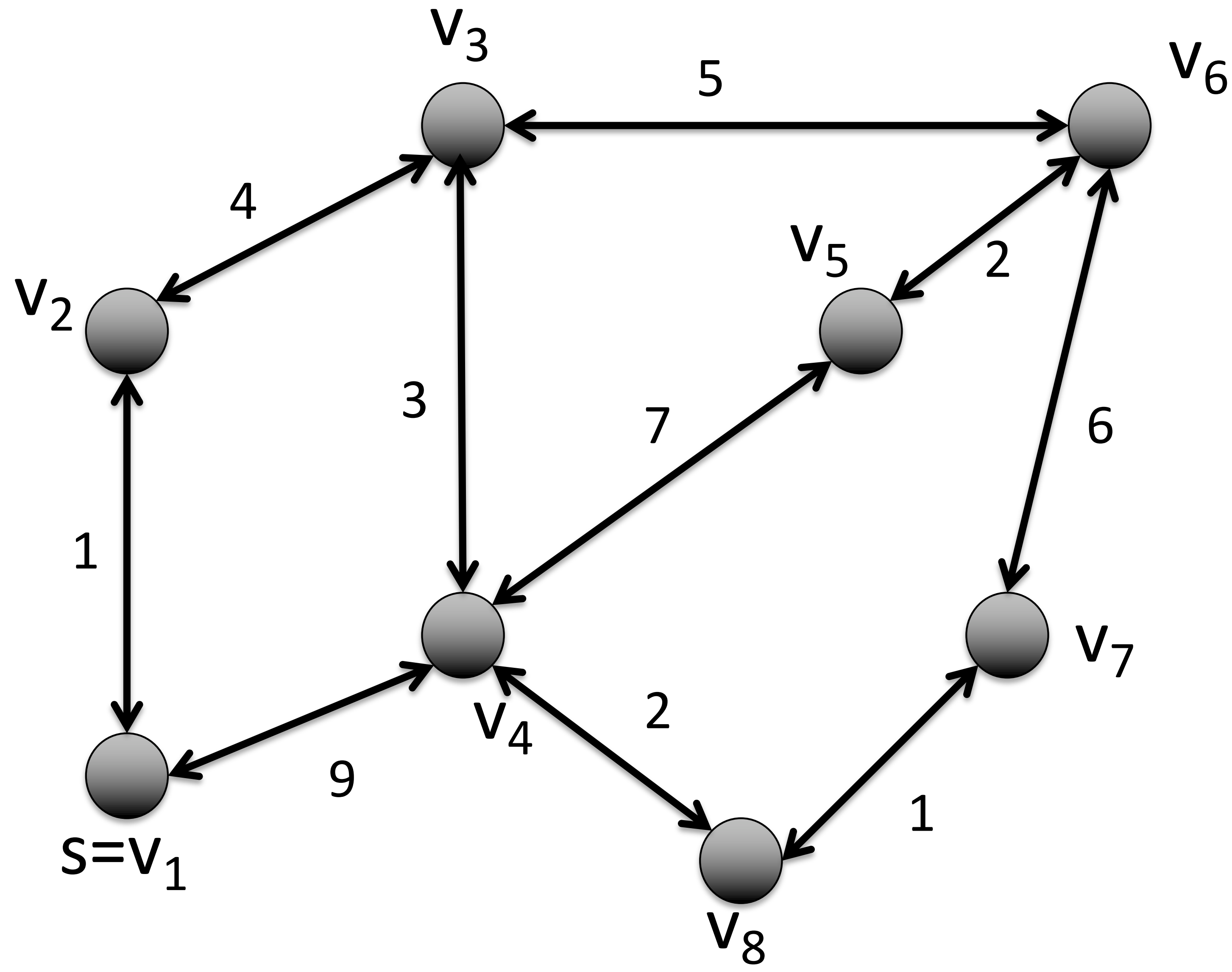
Function *jpMST* : *Set* **of** *Edge*
  $d = \langle\infty,\ldots,\infty\rangle$: $NodeArray[1..n]$ **of** $\mathbb{R}\cup\{\infty\}$ // $d[v]$ is the distance of v from the tree
  parent : *NodeArray* **of** *NodeId* // parent$[v]$ is shortest edge between S and v
  Q : *NodePQ* // uses $d[\cdot]$ as priority
  $Q.insert(s)$ for some arbitrary $s \in V$
  while $Q \neq \emptyset$ **do**
    $u := Q.deleteMin$
    $d[u] := 0$ // $d[u] = 0$ encodes $u \in S$
    foreach *edge* $e = (u,v) \in E$ **do**
      if $c(e) < d[v]$ **then** // $c(e) < d[v]$ implies $d[v] > 0$ and hence $v \notin S$
        $d[v] := c(e)$
        $parent[v] := u$
        if $v \in Q$ **then** $Q.decreaseKey(v)$ **else** $Q.insert(v)$
    invariant $\forall v \in Q : d[v] = \min\{c((u,v)) : (u,v) \in E \wedge u \in S\}$
  return $\{(v, parent[v]) : v \in V \setminus \{s\}\}$

Fig 11.3 Mehlhorn/Sanders

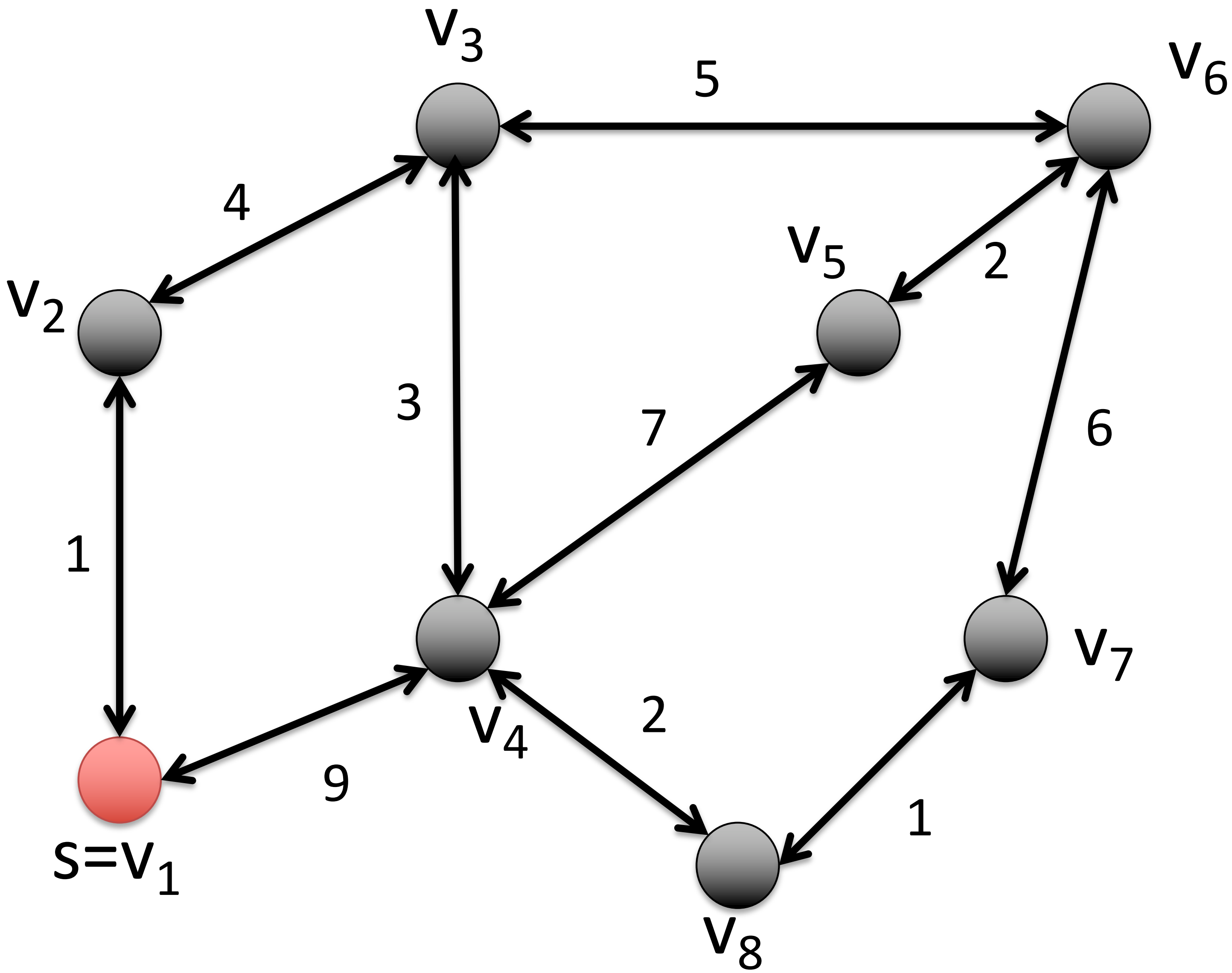
Runtime

- We can carry over the analysis for Dijkstra's algorithm.
- Crucial again is the implementation of the priority queue.
- Overall runtime is O(m + n log n) when using Fibonacci heaps for the implementation of the priority queue.

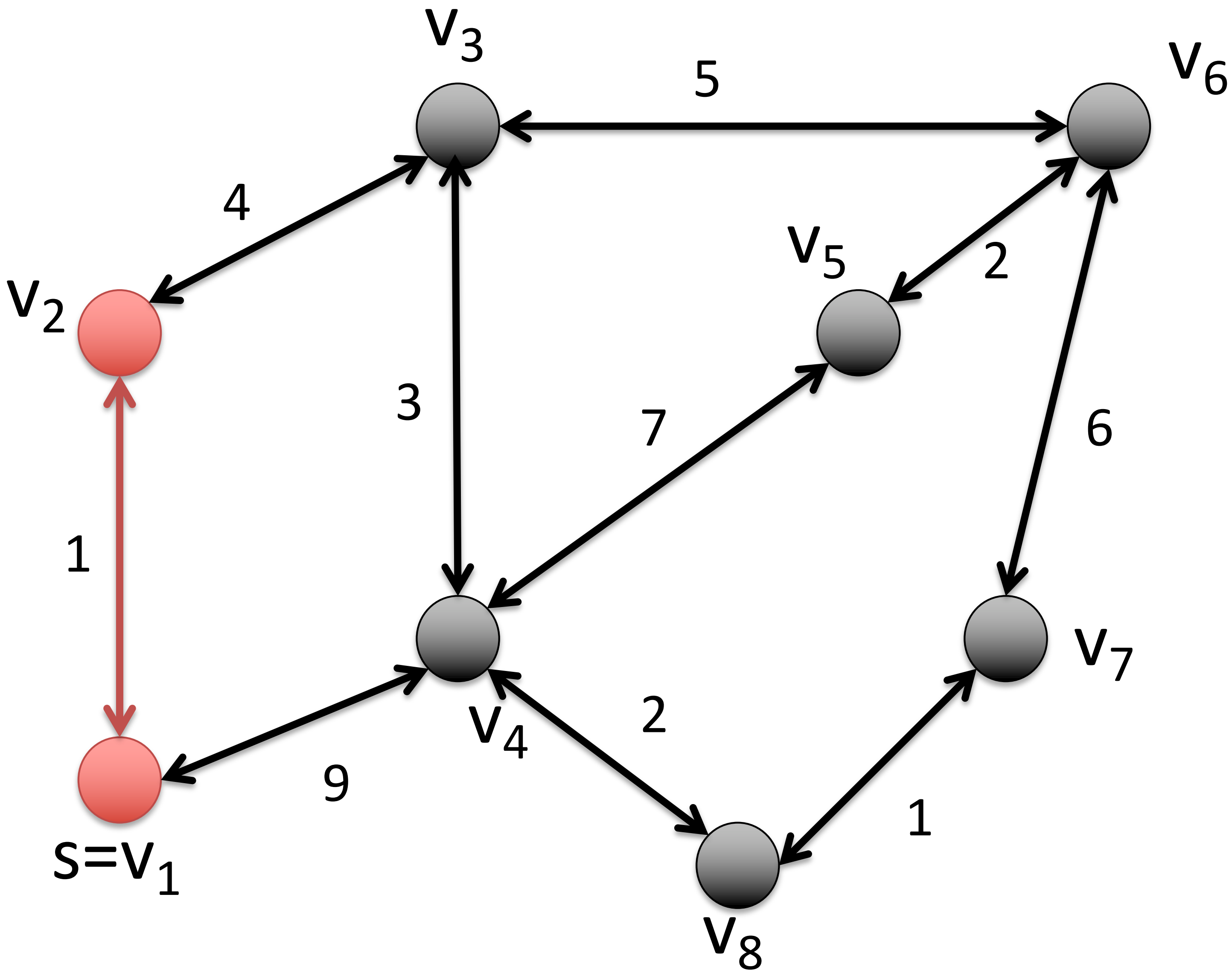
Example



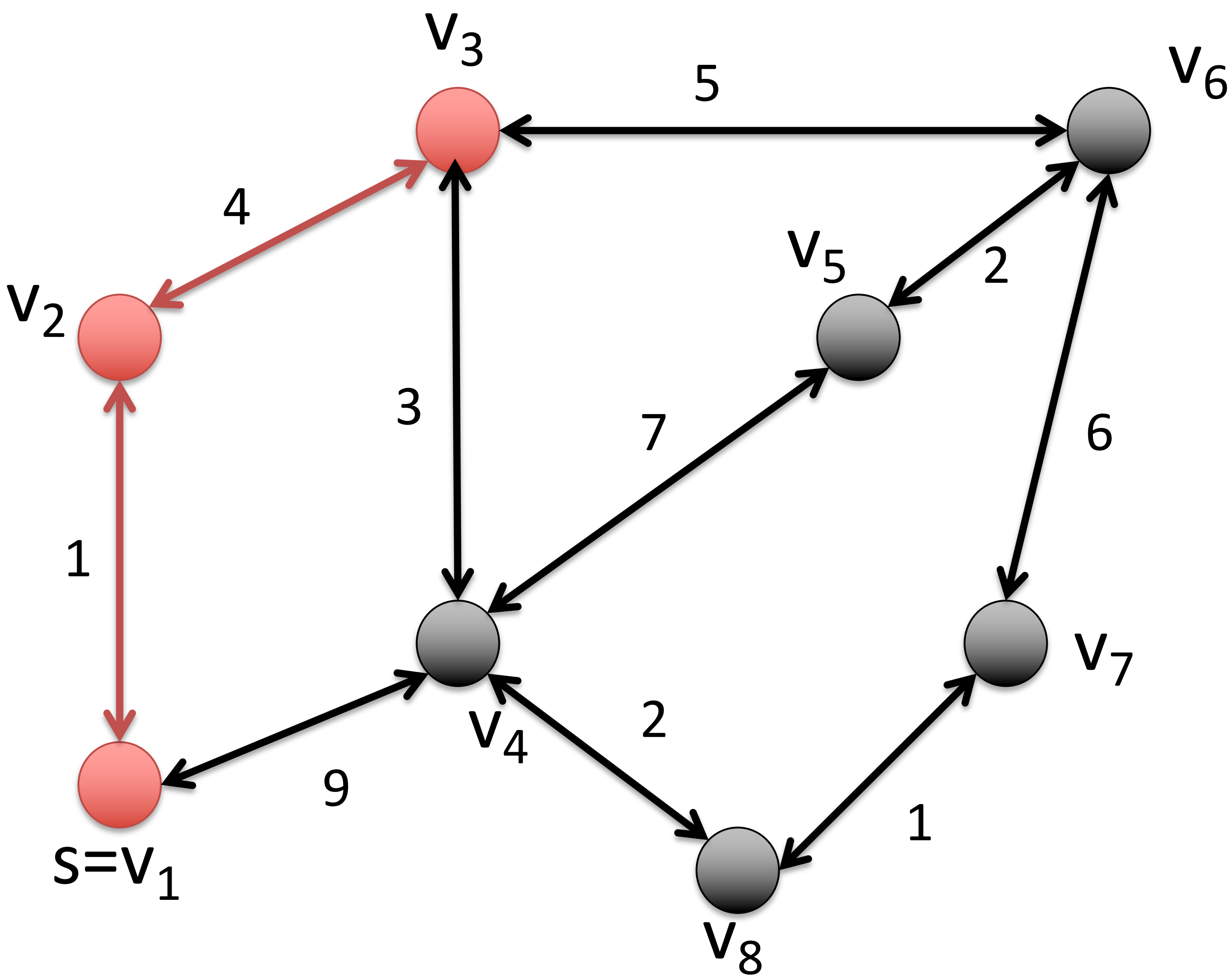
Example



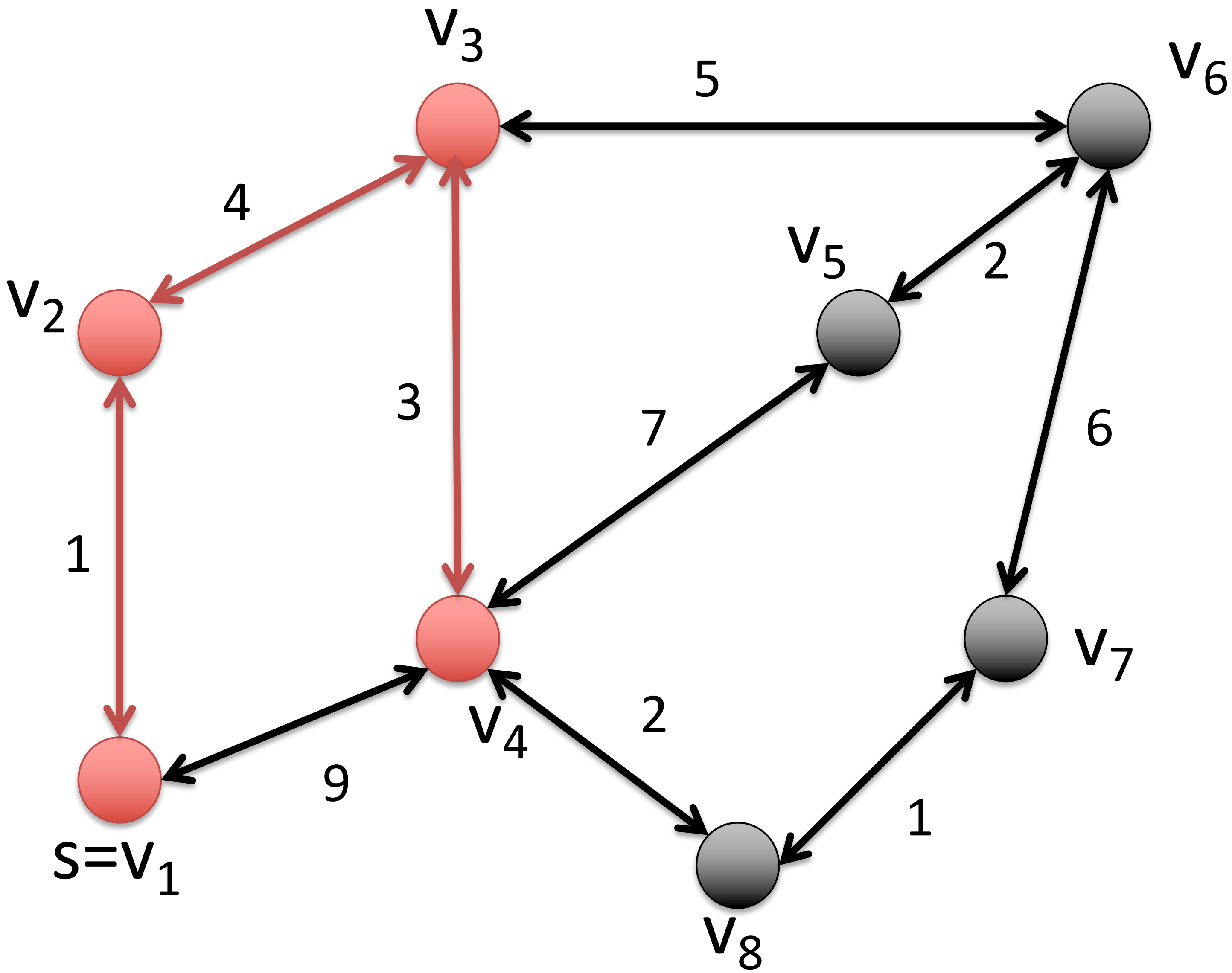
Example



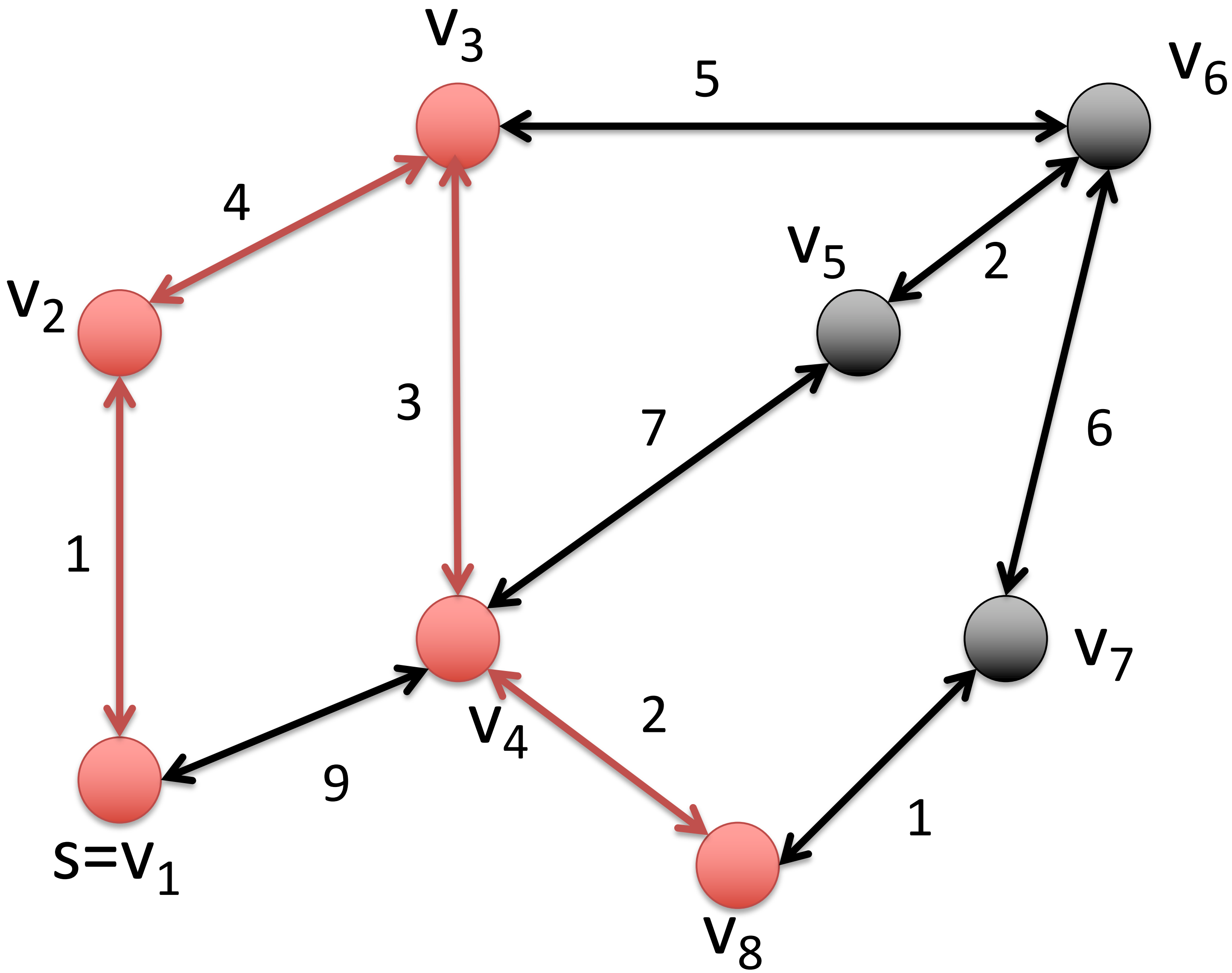
Example



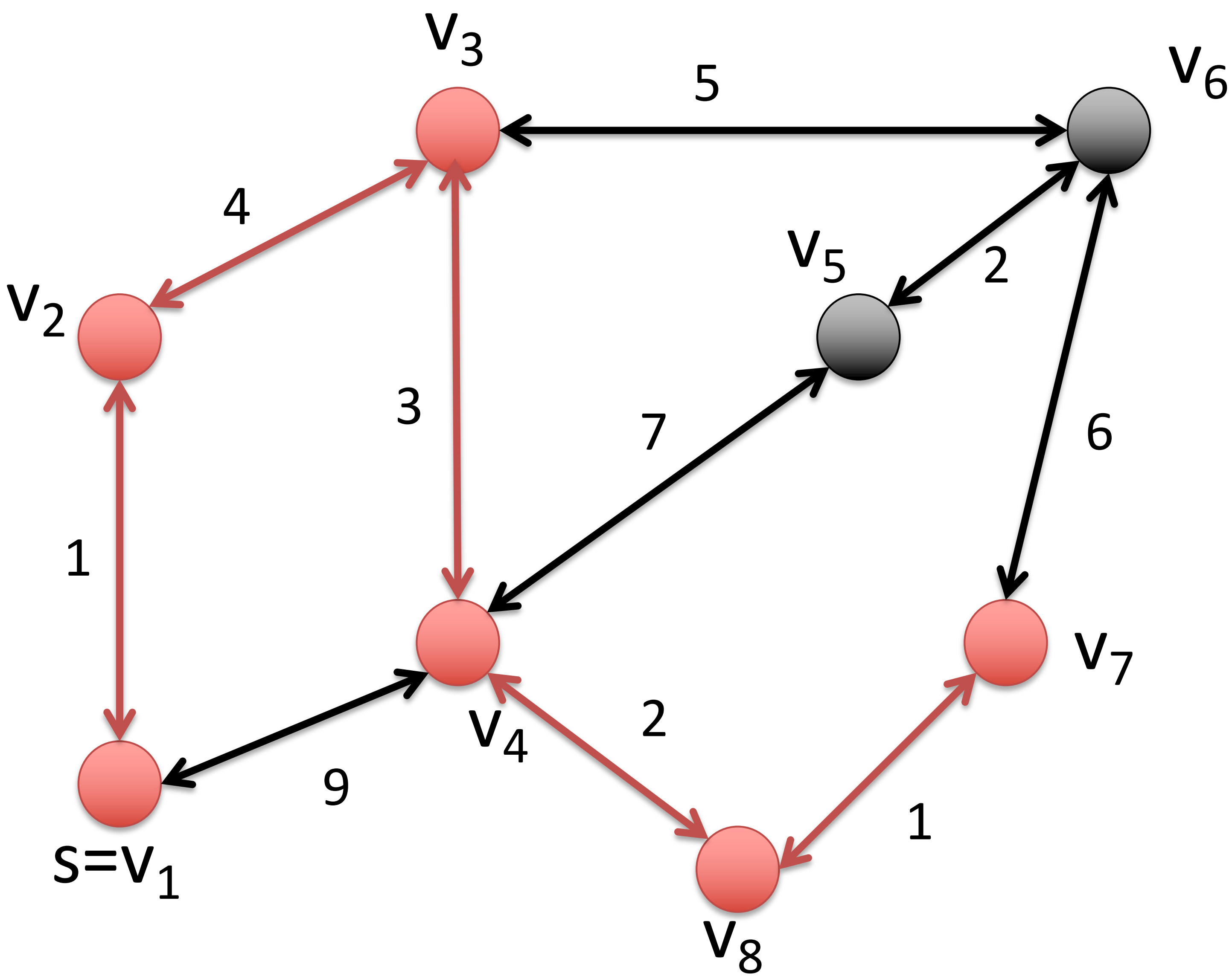
Example



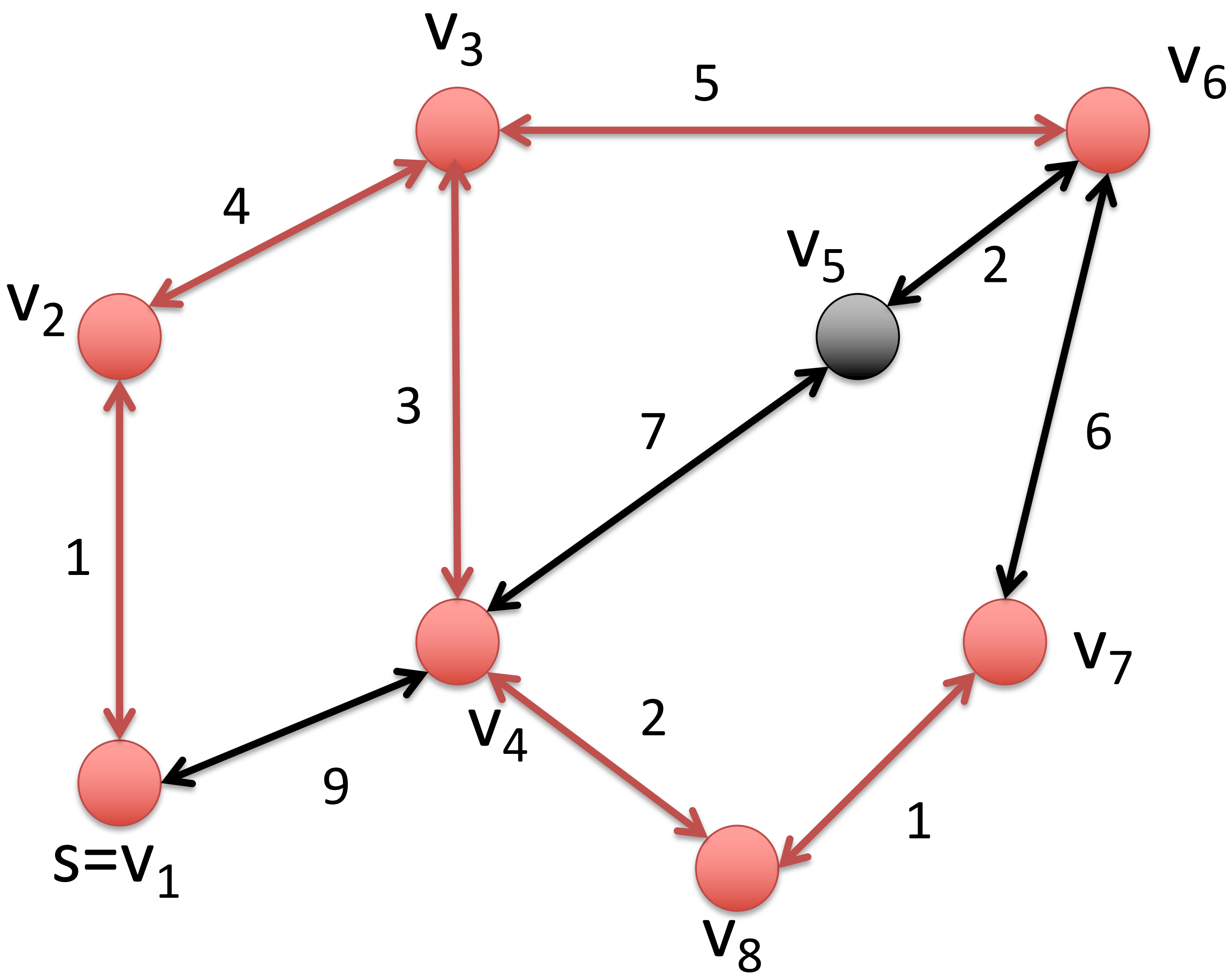
Example



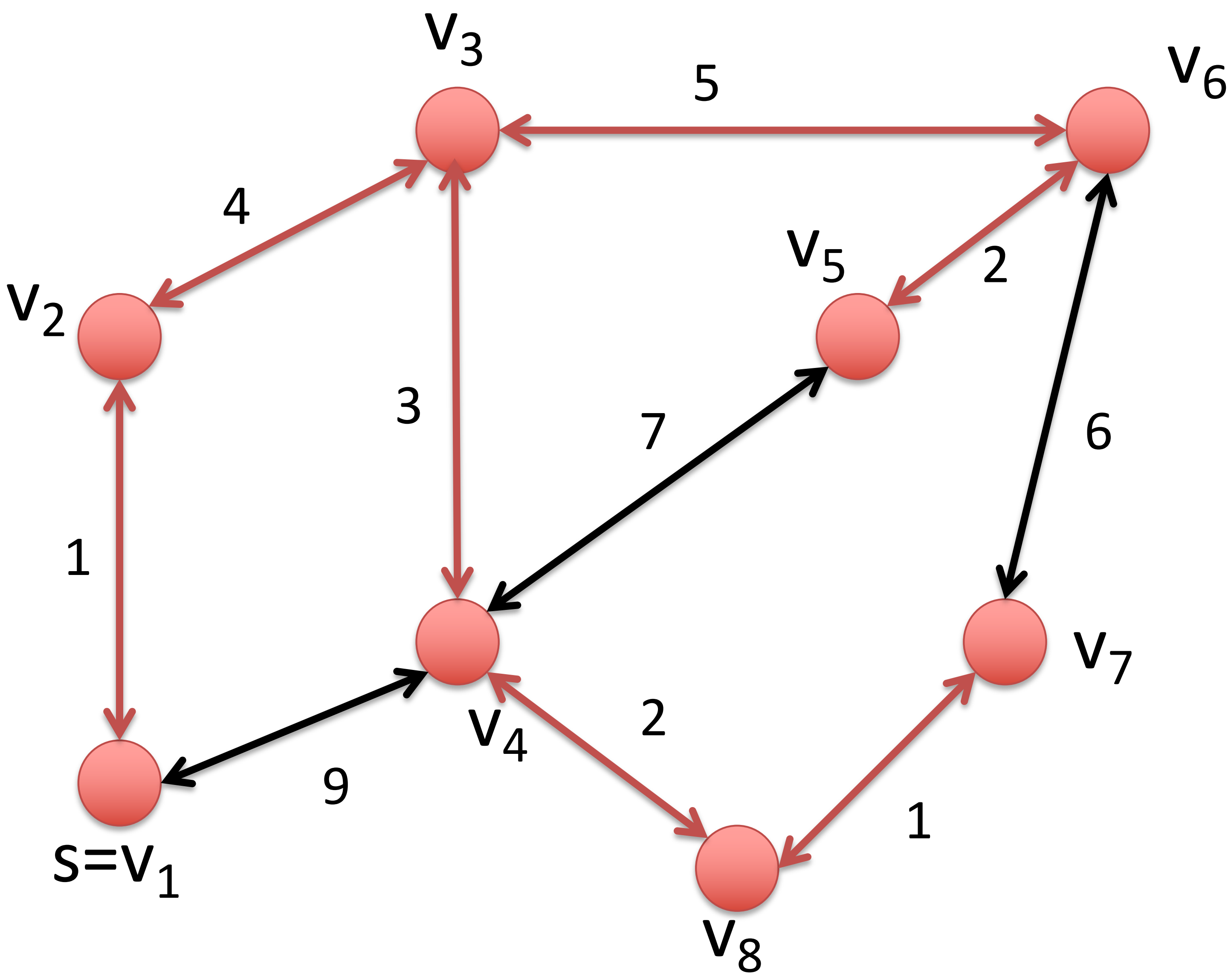
Example



Example



Example



Comparison

- Jarnik-Prim Algorithm can be implemented in time $O(n \log n + m)$
- Kruskal's Algorithm can be implemented in time $O(m \log m)$

Jarnik-Prim Algorithm is more efficient for dense graphs, i. e. where $m = \Theta(n^2)$ holds.

Algorithm and Data Structure Analysis (ADSA)

P and NP

Overview

- Complexity of Problems
- Classes P and NP

Efficient Algorithms

Major Questions:

- When do we call an algorithm efficient?
- Are there problems for which there is no efficient algorithm?

Efficient Algorithms

- An algorithm A runs in polynomial time (is a polynomial time algorithm), if there is a polynomial p(n) such that its execution time on inputs of size n is O(p(n)).

- A problem can be solved in polynomial time if there is a polynomial time algorithm that solves it.

We call an algorithm efficient iff it runs in polynomial time.

Examples

Problems that can be solved in polynomial time:

- Integer Addition and Multiplication
- Computation of shortest paths and minimum spanning trees.
- All problems that we considered so far in this course.

Two problems

First Problem: Compute a spanning tree of a given undirected connected graph G=(V,E).

Second Problem: Compute a spanning of G where each node has degree at most 2.

Such a spanning tree may not exist. Try to answer the following question.

Question: Is there a spanning tree of G where each node has degree at most 2? (Decision problem, answer yes/no)

Difficult Problems

There are many problems for which no efficient algorithm is known.

Examples (see Mehlhorn/Sanders page 54):

- Hamiltonian cycle problem
- Traveling Salesman Problem
- Boolean Satisfiability Problem
- Clique Problem
- Graph Coloring Problem
- Multi-objective Minimum Spanning Trees
- Multi-Objective Shortest Paths

Hamiltonian Path Problem

- Given: Undirected graph G=(V,E).
- Decide whether G contains a Hamiltonian path. A Hamiltonian path is a path that visits each node exactly once. (A spanning tree where each node has degree at most 2.)

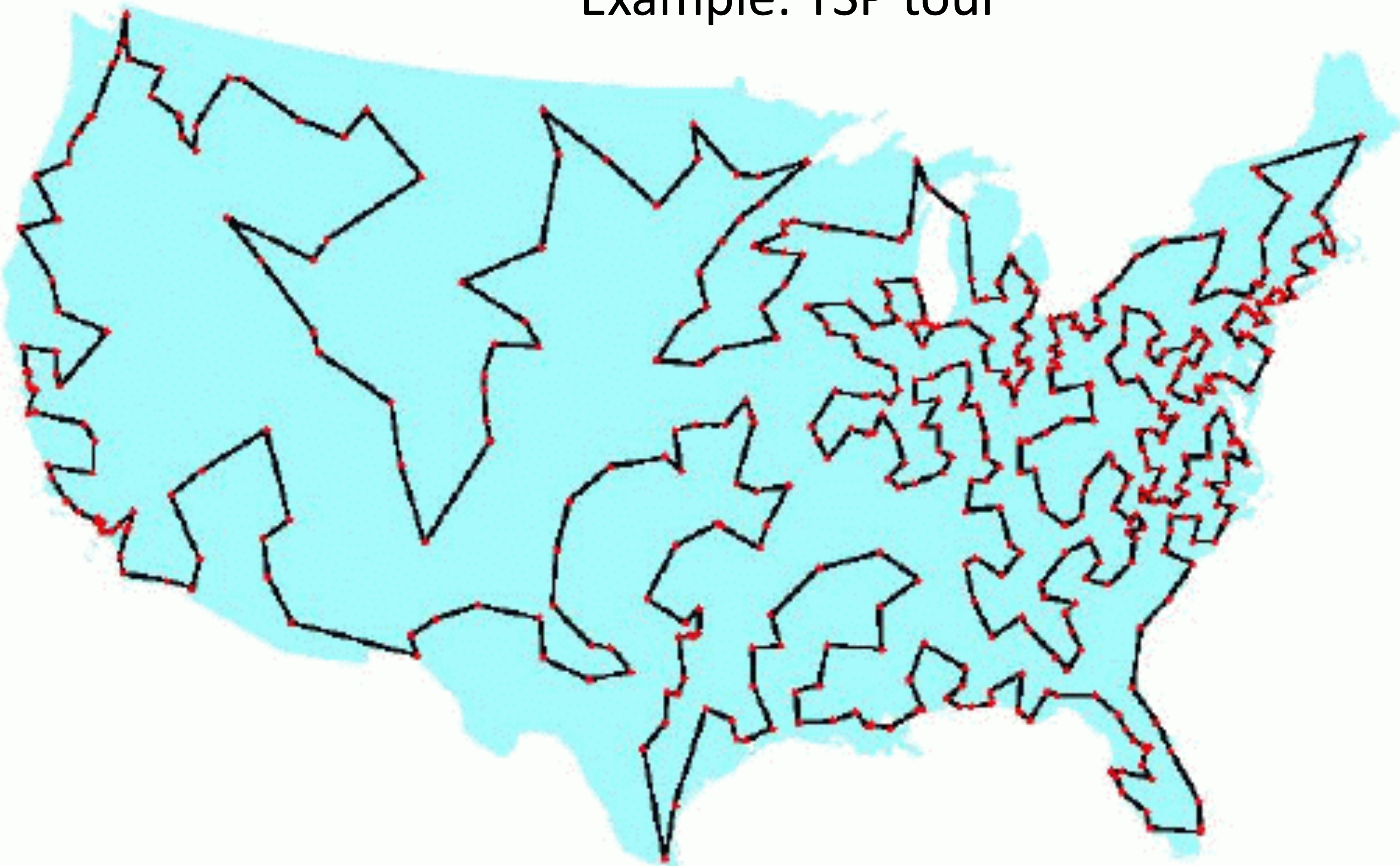
Hamiltonian Cycle Problem

- Given: Undirected graph G=(V,E).
- Decide whether G contains a Hamiltonian cycle. A Hamiltonian cycle is cycle that visits each node exactly once and returns to the start vertex.

Traveling Salesman Problem

- Given: Complete edge-weighted undirected graph G=(V,E) and an integer C.
- Decide whether G contains a Hamiltonian cycle of cost at most C.

Example: TSP tour



Optimal tour for 532 AT&T switch locations in the USA.
(from http://www.tsp.gatech.edu)

Graph Coloring Problem

- Given: Undirected graph G=(V,E) and an integer k.
- Decide whether there is a coloring of the nodes with k color such that any two adjacent nodes are colored differently.

Multi-Objective Minimum Spanning Trees

Given: Undirected graph connected graph G=(V,E) with two weight functions w_1 and w_2 on the edges, and two numbers k_1 and k_2.

- Decide whether there is a spanning tree T of G for which

$$w_1(T) \leq k_1 \text{ and } w_2(T) \leq k_2$$

holds.

Boolean Satisfiability problem

- Given: A Boolean expression in conjunctive normal form.
- Decide whether it has a satisfying assignment.

Conjunctive normal form is conjunction of clauses $C_1 \wedge C_2 \wedge \ldots \wedge C_k$

Clause is disjunction of literals $l_1 \vee l_2 \vee \ldots \vee l_h$.

Literal is variable or a negated variable.

NP-Complete Problems

- We don’t know whether polynomial time algorithms exists for the mentioned problems.
- It is very likely (and almost all people in computer science believe) that there are no polynomial time algorithms for these problems.
- They belong to a class of equivalent problems known as NP-complete problems. (NP stands for “nondeterministic polynomial time”)