Georgia Tech

CSE 6140/ CX 4140

Computational Science and Engineering

ALGORITHMS

Greedy Algorithms - 3

Instructor: Xiuwei Zhang

School of Computational Science and Engineering

Course logistics

- Getting help
  - **Piazza**
  - **Office hours (Canvas -> Calendar)**
  - Email or Canvas messages
- Dates for exams
  - Test 1: Sep 18 ~~Sep 16~~
  - Test 2: Oct 28 ~~Oct 21~~
- Academic Honor Code

Lecture outline

- Review of last lecture
- Introduction to graphs
- Shortest paths in graphs

Greedy Analysis Strategies

Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's. (Interval Scheduling)

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality. (Minimizing Maximum Lateness)

Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound. (Interval Partitioning)

Greedy Algorithms

- **Greedy-choice property**: we can assemble a globally optimal solution by making locally optimal (greedy) choices

- i.e., we make the choice that looks best given the current partial solution
- Once the choice is made, it will not be changed

  Proofs: we usually argue that an optimal solution to a subproblem problem combined with the greedy choice also results in an optimal solution to the original problem

Proof of correctness for greedy algorithms

Show that your solution is **feasible** and **optimal** (or **correct**, if your problem is not an optimization problem)

Greedy stays ahead -- suggested structure:

• *Define Your Solution*. Your algorithm will produce some object *G* and you will probably compare it against some optimal solution *O*. Introduce some variables denoting your algorithm's solution and the optimal solution.

• *Define Your Measure*. Your goal is to find a series of measurements you can make of your solution and the optimal solution. Define some series of measures $m_1(G)$, $m_2(G)$, …, $m_k(G)$ such that $m_1(O)$, $m_2(O)$, …, $m_n(O)$ is also defined for some choices of *m* and *n*.

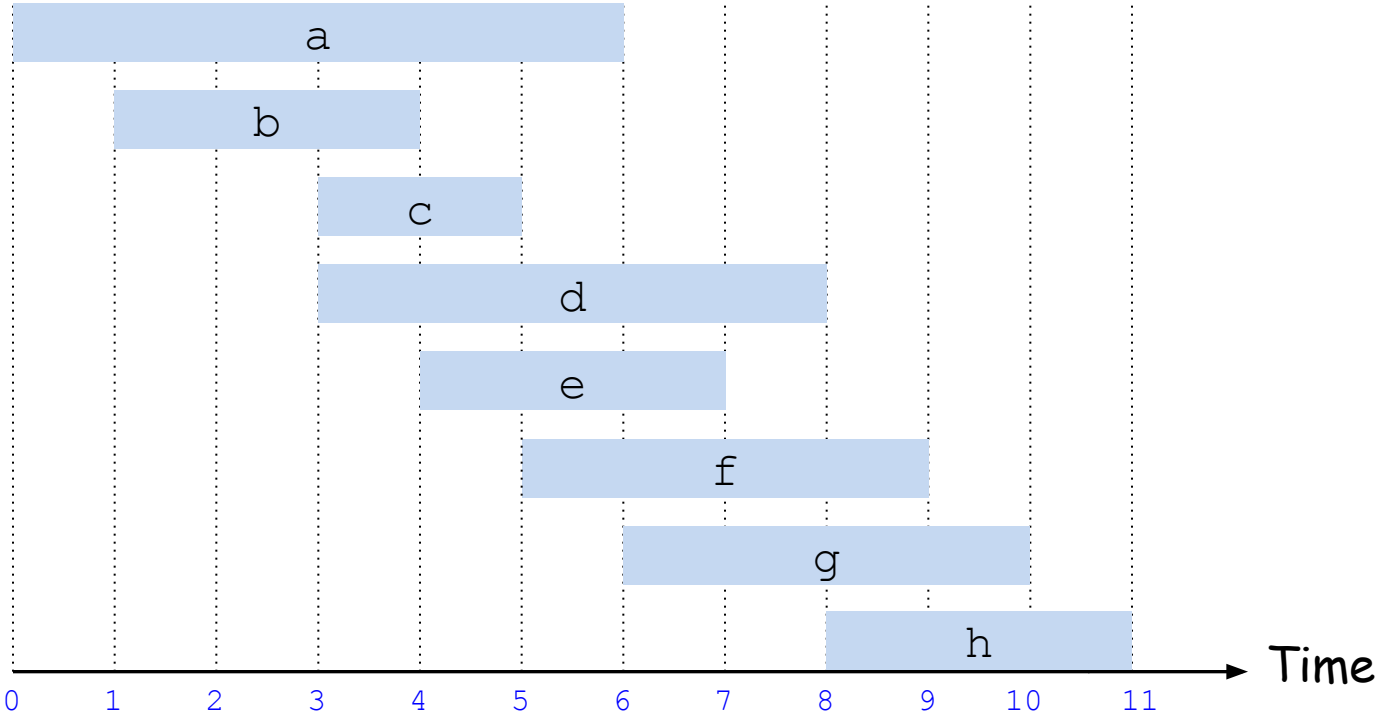
• *Prove Greedy Stays Ahead*. Prove that $m_i(G) \geq m_i(O)$ or that $m_i(G) \leq m_i(O)$, whichever is appropriate, for all reasonable values of *i*. This argument is usually done inductively.

• *Prove Optimality*. Using the fact that greedy stays ahead, prove that the greedy algorithm must produce an optimal solution. This argument is often done by contradiction by assuming the greedy solution isn't optimal and using the fact that greedy stays ahead to derive a contradiction.

Interval Scheduling

Interval scheduling.

- Job j starts at s_j and finishes at f_j.
- Two jobs compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Interval Scheduling: Greedy stays ahead argument

- Let A: $a_1, a_2, \ldots\ a_k$ denote set of jobs selected by greedy.
- Let O: $o_1, o_2, \ldots\ o_m$ denote set of jobs in the optimal solution.

Define Your Solution

What do we know about finish times of jobs in A?

$$f(a_1) < f(a_2) < \cdots < f(a_k)$$

Define Your Measure

We order the optimal solution in that way

$$f(o_1) < f(o_2) < \cdots < f(o_m)$$

Claim: For all indices $r \leq k$, $f(a_r) \leq f(o_r)$

Interval Scheduling: Greedy stays ahead argument

Claim: For all indices $r \leq k$, $f(a_r) \leq f(o_r)$

Pf. (by induction)

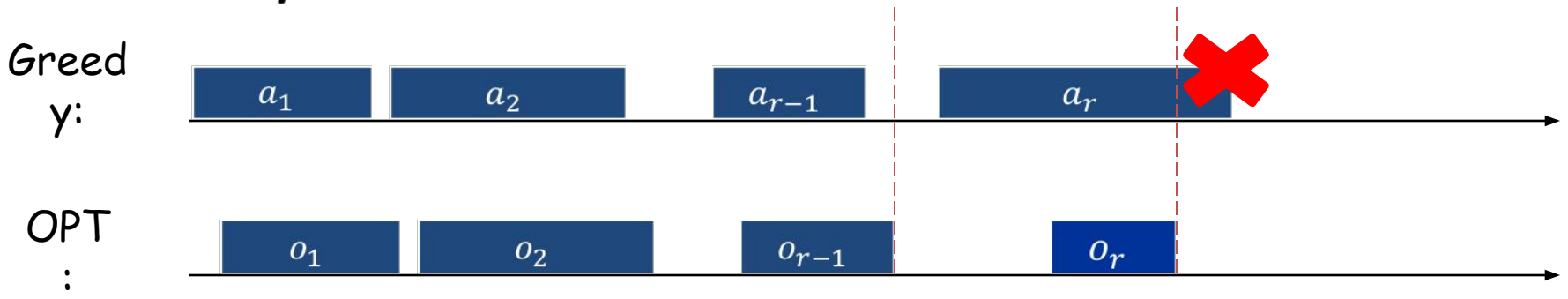
Base case: $r = 1$

$$f(a_1) \leq f(o_1)$$

True by greedy choice of earliest finish time.

Inductive hypothesis: Holds for $r - 1$, i.e., $f(a_{r-1}) \leq f(o_{r-1})$

Inductive step:



$$f(o_{r-1}) \leq s(o_r) \leq f(o_r)$$

Interval Scheduling: Greedy stays ahead argument

$f(o_{r-1}) \leq s(o_r) \leq f(o_r)$ by feasibility of optimum solution

$f(a_{r-1}) \leq f(o_{r-1})$ by ind. Hypothesis

$f(a_{r-1}) \leq s(o_r)$

$\Rightarrow o_r$ is compatible with $a_1, a_2, \ldots, a_{r-1}$

and was an option for greedy

a_r was the greedy choice among all compatible jobs at iteration r

$\Rightarrow f(a_r) \leq f(o_r)$

Prove Greedy Stays Ahead

Interval Scheduling: Greedy stays ahead argument

Theorem. Greedy algorithm is optimal $(k = m)$

A: $a_1, a_2, \ldots\ a_k$

O: $o_1, o_2, \ldots\ o_m$

Assume for the sake of contradiction $k < m$

k: $f(a_k) \leq f(o_k)$

Optimal solution must have o_{k+1} $(k < m)$

Optimal solution O is feasible

$\Rightarrow f(o_k) \leq s(o_{k+1}) \leq f(o_{k+1})$

$\Rightarrow o_{k+1}$ was still an option for greedy because $f(a_k) \leq s(o_{k+1})$ after iteration k

$\Rightarrow$ Contradiction greedy stopping at iteration k

$\Rightarrow k = m$■

Prove Optimality

Proof of correctness for greedy algorithms

Exchange arguments: show that you can iteratively transform any optimal solution into the solution produced by the greedy algorithm without worsening the cost of the optimal solution, thereby proving that the greedy solution is optimal.

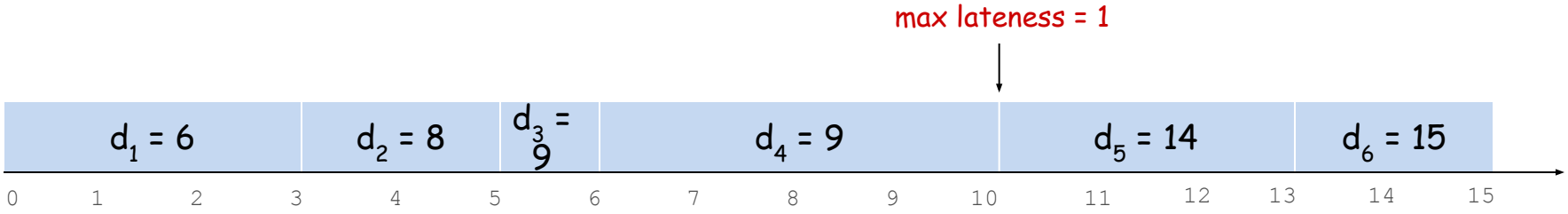
- *Define Your Solutions.* You will be comparing your greedy solution *G* to an optimal solution *O*.
- *Compare Solutions*. Show that if $G \neq O$, then they must differ in some way. This could mean that there's a piece of *G* that's not in *O*, or that two elements of *G* that are in a different order in *O*, etc. You might want to give those pieces names.
- *Exchange Pieces*. Show how to transform *O* by exchanging some piece of *O* for some piece of *G*. You'll typically use the piece you described in the previous step. Then, prove that by doing so, you did not worsen the score of *O* and therefore have a different optimal solution.
- *Iterate*. Argue that you have decreased the number of differences between *G* and *O* by performing the exchange, and that by iterating this process for a finite number of times you can turn *O* into *G* without impacting the quality of the solution. Therefore, *G* must be optimal.

Scheduling to minimizing lateness

- Single resource processes one job at a time.
- Job j requires t_j units of processing time and is due at time d_j.
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max_j \ell_j$.

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

Greedy strategy: earliest deadline first



Scheduling to minimizing lateness

Observation 1. There exists an optimal schedule with no idle time.

Observation 2. The earliest-deadline-first schedule has no idle time.

Observation 3. The earliest-deadline-first schedule is the unique idle-free schedule with no inversions, if all deadlines are different

Lemma. All schedules with no inversions and no idle time have the same lateness.

Observation 4. If an idle-free schedule has an inversion, then it has an adjacent inversion.

Key Claim. Exchanging two adjacent, inverted jobs i and j reduces the number of inversions by 1 and does not increase the max lateness.

Theorem. There is an optimal schedule with no inversions and no idle time.

Theorem. Greedy is optimal.

Scheduling to minimizing lateness - proof

Define Your Solutions. Denote greedy solution as *G*, and an optimal solution *O*. We pick the optimal solution which has no idle time. There exists such an optimal solution (Observation 1)

• *Compare Solutions*. Show that if $G \neq O$, then they must differ in some way. Greedy solution has no inversions and no idle time (Observation 3). Now we need to consider two cases: (1) *O* has no inversions: in this case *G* has the same lateness as *O* (lemma); so *G* is optimal. (2) *O* has inversions: in this case, we will move to our exchange argument next.

• *Exchange Pieces*. Here we want to show that *O* can be gradually converted into *G* without hurting the quality of *O*. *G* has no inversions and no idle time. *O* has inversions and no idle time. *O* must have an adjacent inversion (Observation 4). We can invert this adjacent inversion without hurting the quality of *O* (key claim).

• *Iterate*. This operation reduces the number of inversions by 1. There are at most n(n-1)/2 inversions. By inverting the adjacent inversions for at most n(n-1)/2 times O is turned into a solution with no inversions and no idle time. G also has no inversions and no idle time. G has the same lateness as O (lemma).

Therefore, *G* is optimal.

Greedy Algorithms

Greedy-choice property: we can assemble a globally optimal solution by making locally optimal (greedy) choices

With an optimal greedy strategy, we can show

1) There is always an optimal solution that performs the greedy choice.
2) If we combine the greedy choice with an optimal solution of the subproblem that we still need to solve, then we obtain an optimal solution.

We make a **local choice**, and then we have a single subproblem to solve, given this choice. **Top-down** algorithm.

[Exercise]: prove "1)" for the "interval scheduling" problem and the "minimizing maximum lateness" problem.

Topics coming next

1. Introduction to graphs
2. Shortest path problem on graphs and the Dijkstra's Algorithm
3. Proof of correctness of the Dijkstra's Algorithm

Georgia Tech

GRAPHS

Graphs & Applications

Graph. G = (V, E)

- V = nodes.
- E = edges between pairs of nodes (undirected, directed, weighted).
- Captures pairwise relationship between objects.
- Graph size parameters: $n = |V|$, $m = |E|$.

Graphs & Applications

Graph. G = (V, E)

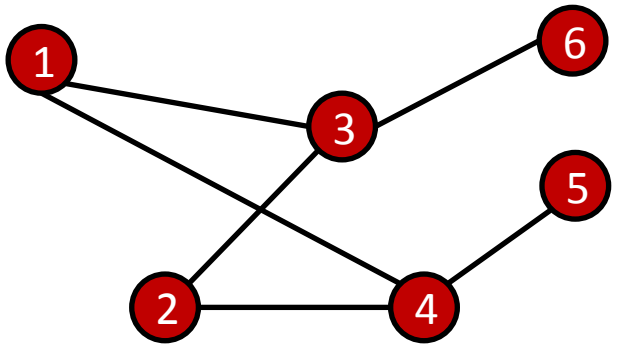
- V = nodes.
- E = edges between pairs of nodes (undirected, directed, weighted).
- Captures pairwise relationship between objects.
- Graph size parameters: n = |V|, m = |E|.

Graph	*Nodes*	*Edges*
transportation	street intersections	highways
communication	computers	fiber optic cables
World Wide Web	web pages	hyperlinks
social	people	relationships
food web	species	predator-prey
software systems	functions	function calls
scheduling	tasks	precedence constraints
biological networks	genes or proteins	regulatory relationships

Graph is a matrix

Representation:

- Adjacency matrix
- For n vertices and m edges:
  - Space proportional to n^2.
  - Checking if (u, v) is an edge takes Θ (1) time.
  - Identifying all edges takes $\Theta(n^2)$ time.
  - Undirected graph: binary, symmetric

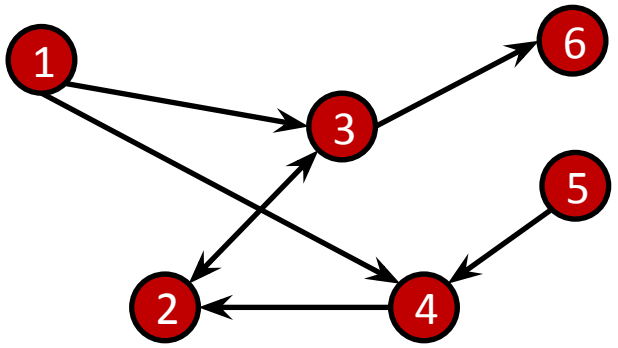


	1	2	3	4	5	6
1	0	0	1	1	0	0
2	0	0	1	1	0	0
3	1	1	0	0	0	1
4	1	1	0	0	1	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0

Graph is a matrix

Representation:

- Adjacency matrix
- For n vertices and m edges:
  - Space proportional to n^2.
  - Checking if (u, v) is an edge takes Θ (1) time.
  - Identifying all edges takes $\Theta(n^2)$ time.
  - Directed graph: binary, not symmetric

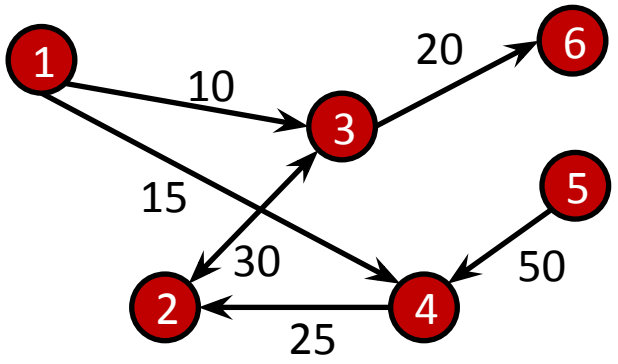


	1	2	3	4	5	6
1	0	0	1	1	0	0
2	0	0	1	0	0	0
3	0	1	0	0	0	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	0

Graph is a matrix

Representation:

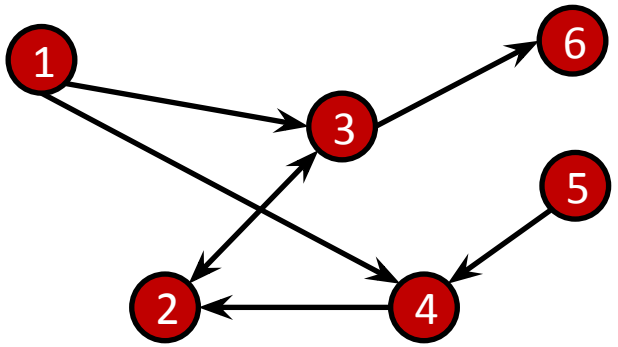
- Adjacency matrix
- For *n* vertices and *m* edges:
  - Space proportional to n^2.
  - Checking if (u, v) is an edge takes Θ (1) time.
  - Identifying all edges takes $\Theta(n^2)$ time.
  - Weighted directed graph: Not binary, not symmetric



	1	2	3	4	5	6
1	0	0	10	15	0	0
2	0	0	30	0	0	0
3	0	30	0	0	0	20
4	0	25	0	0	0	0
5	0	0	0	50	0	0
6	0	0	0	0	0	0

Graph is a sparse matrix

Representation:

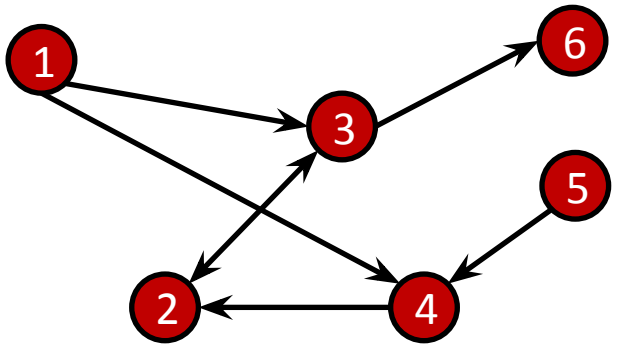


	1	2	3	4	5	6
1	0	0	1	1	0	0
2	0	0	1	0	0	0
3	0	1	0	0	0	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	0

Graph is a sparse matrix

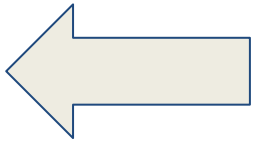
Representation:

- Adjacency list



1	3	4
2	3	
3	2	6
4	2	
5	4	
6		

Adjacency List

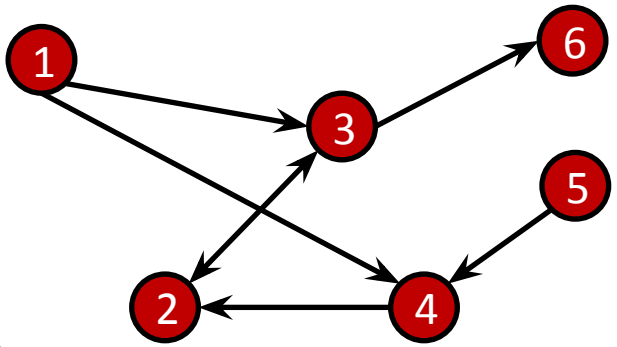


	1	2	3	4	5	6
1	0	0	1	1	0	0
2	0	0	1	0	0	0
3	0	1	0	0	0	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	0

Graph is a sparse matrix

Representation:

- Adjacency list
  - Space proportional to $m + n$.
  - Checking if (u, v) is an edge takes O(deg(u)) time.
  - Identifying all edges takes $\Theta(m + n)$ time.



1	3, 4
2	3
3	2, 6
4	2
5	4
6	

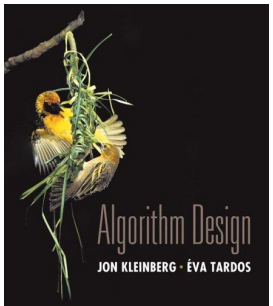
Adjacency List

	1	2	3	4	5	6
1	0	0	1	1	0	0
2	0	0	1	0	0	0
3	0	1	0	0	0	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	0

SHORTEST PATHS IN GRAPH [KT 4.4]

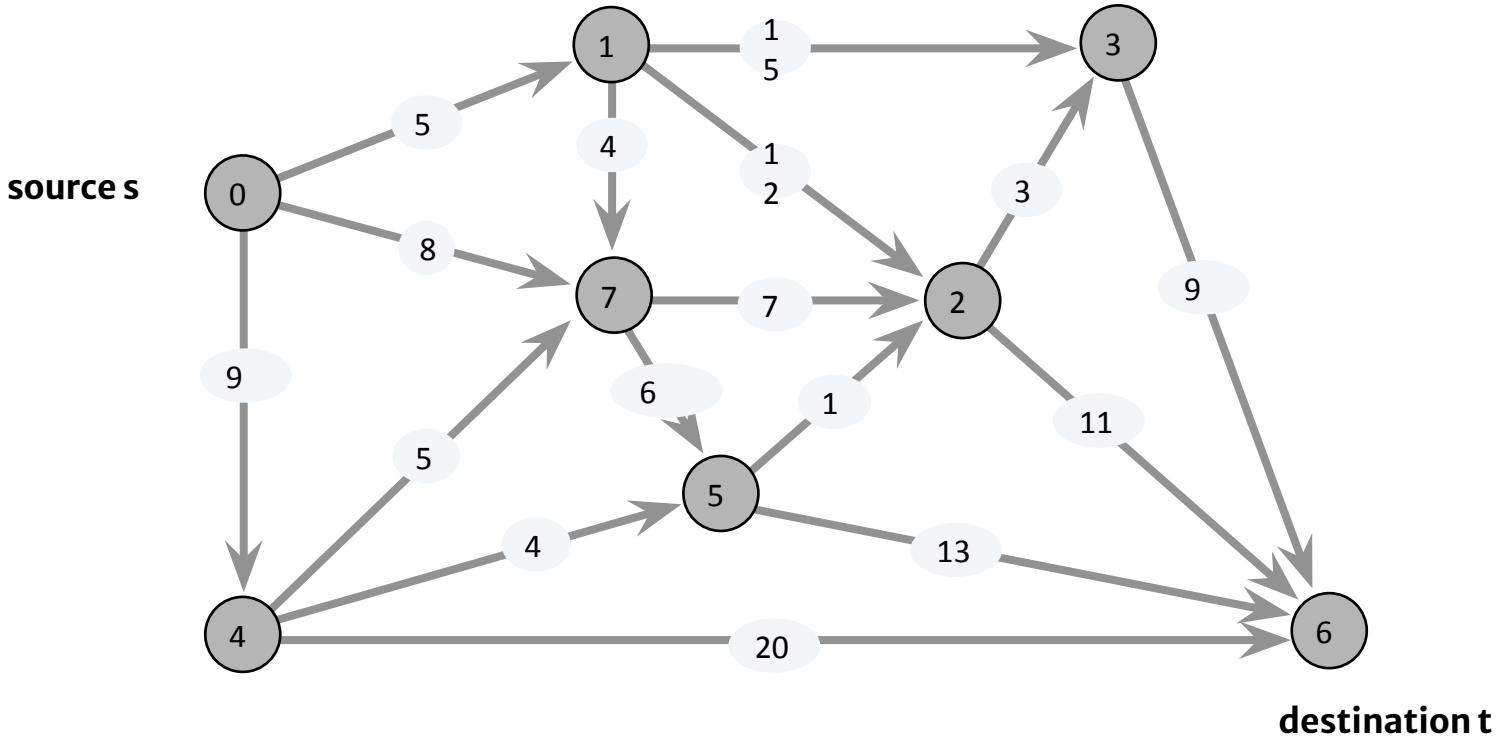
Adapted from Slides by
Kevin Wayne.

And Bistra Dilkina, Anne Benoit



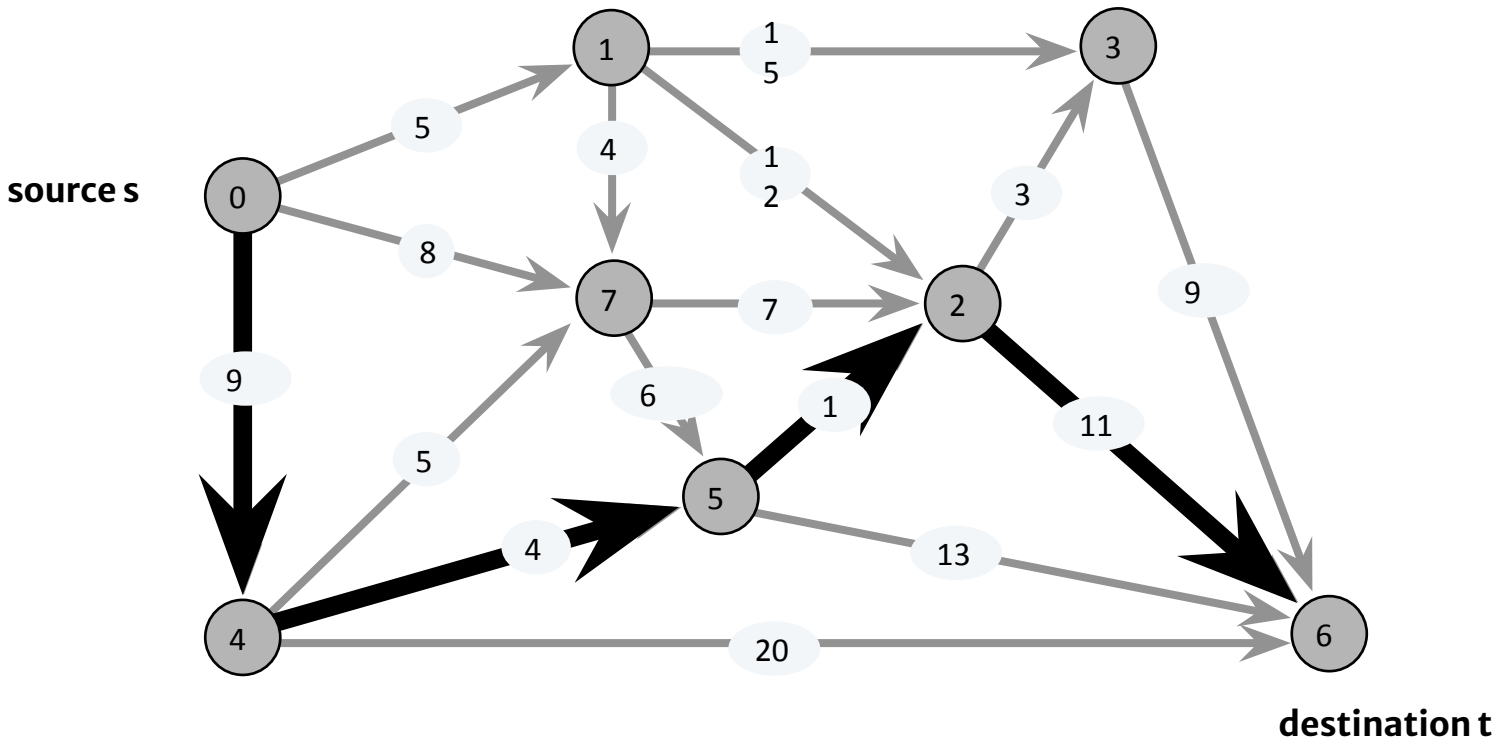
Single-pair shortest path problem

Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, and destination $t \in V$, find a shortest directed path from s to t.



Single-pair shortest path problem

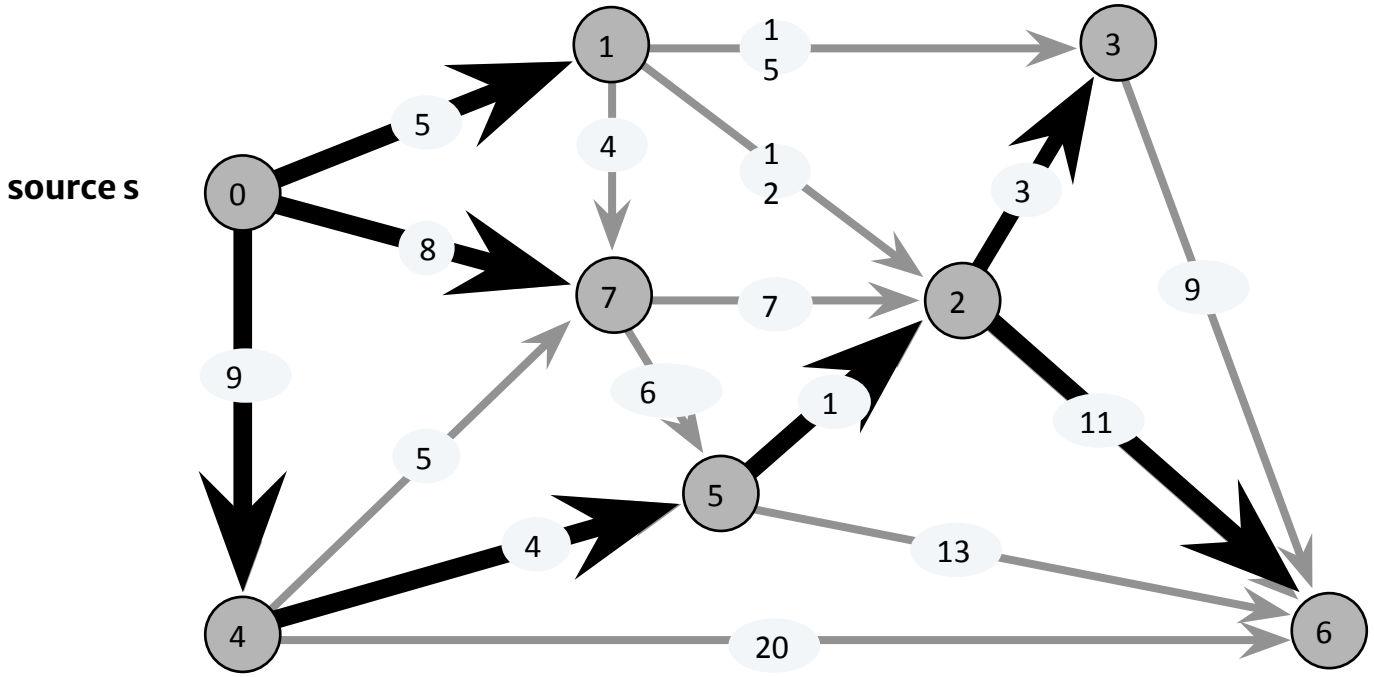
Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, and destination $t \in V$, find a shortest directed path from s to t.



length of path = 9 + 4 + 1 + 11 = 25

Single-source shortest paths problem

Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, find a shortest directed path from s to every node.

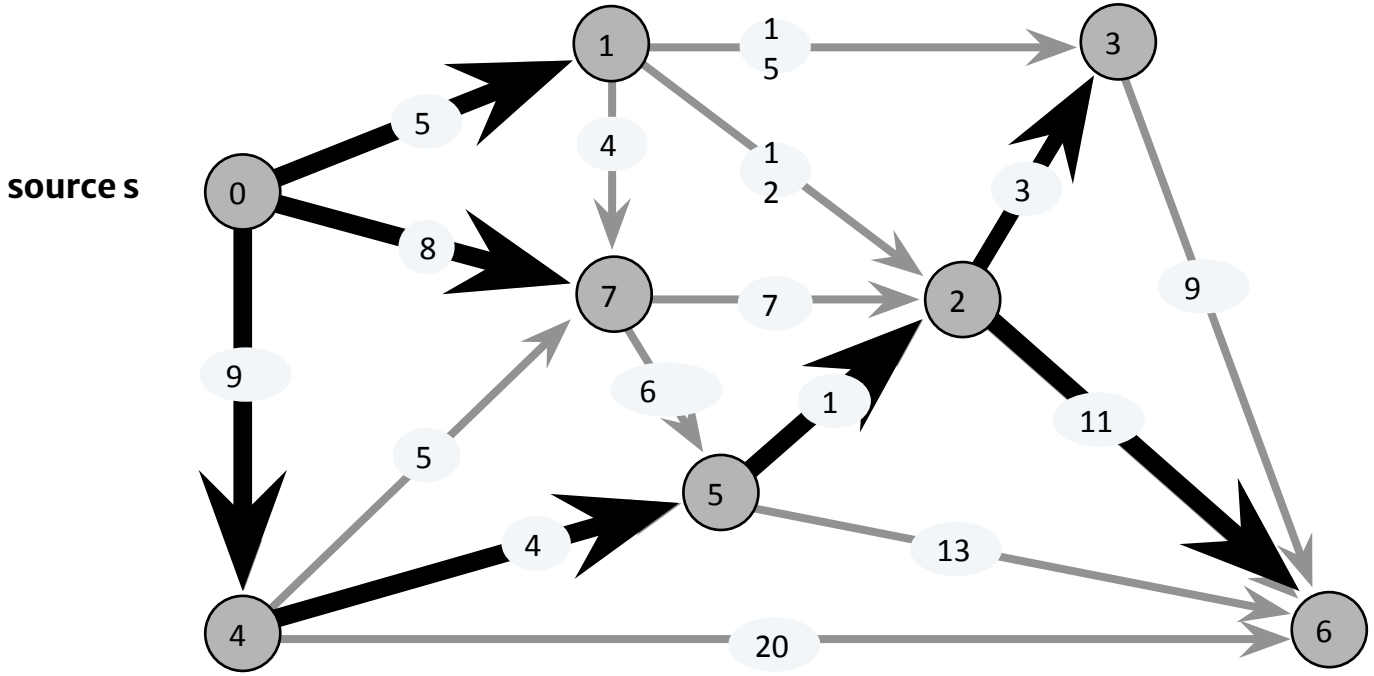


shortest-paths tree

Single-source shortest paths problem

Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, find a shortest directed path from s to every node.

Assumption. There exists a path from s to every node.



shortest-paths tree

Applications

- Map routing.
- Robot navigation.
- Urban traffic planning.
- Optimal truck routing through given traffic congestion pattern.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Circuit design – critical path analysis.

Dijkstra's Algorithm

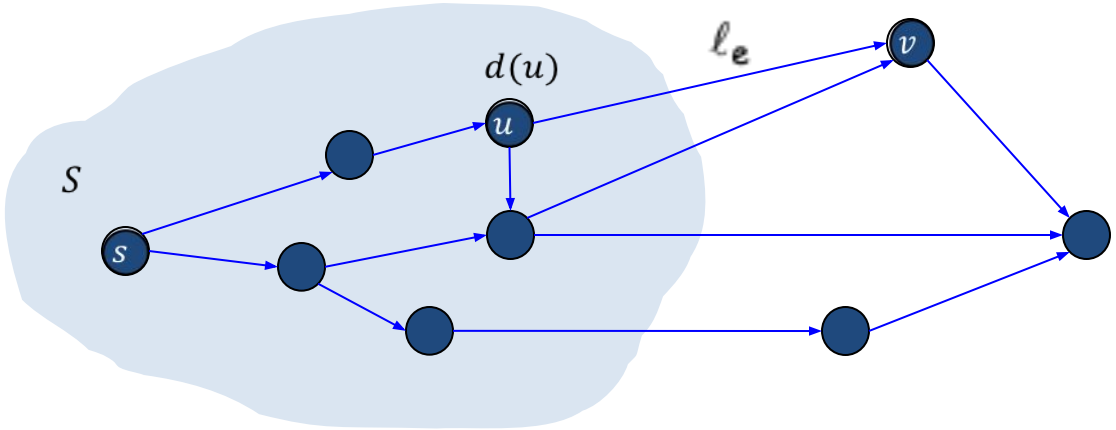
Dijkstra's algorithm.

- Maintain a set of explored nodes S for which we have determined the shortest path distance $d(u)$ from s to u.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose greedily unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e ,$$

add v to S, and set $d(v) = \pi(v)$.

shortest path to some u in explored part, followed by a single edge (u, v)



Dijkstra's Algorithm

Dijkstra's algorithm.

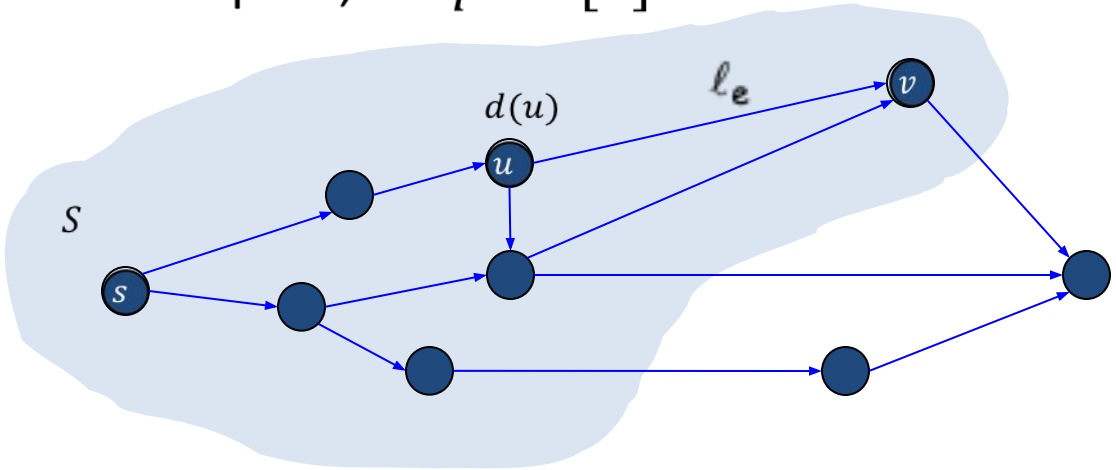
- Maintain a set of explored nodes S for which we have determined the shortest path distance $d(u)$ from s to u.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose greedily unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e ,$$

  add v to S, and set $d(v) = \pi(v)$.

  shortest path to some u in explored part, followed by a single edge (u, v)

- To recover path, set $pred[v] \leftarrow e$ that achieves min.



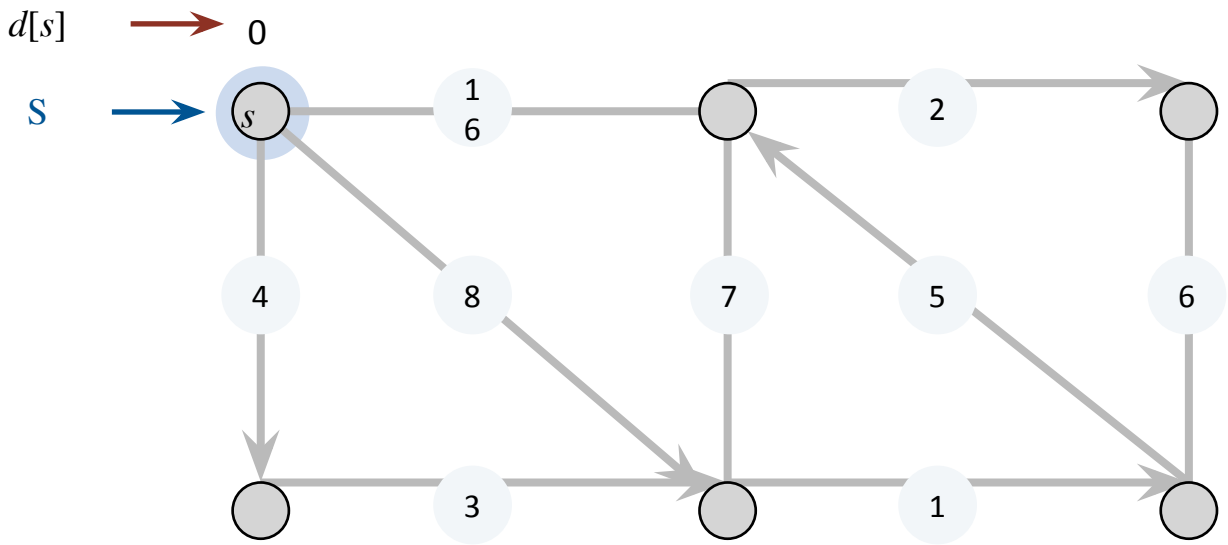
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e=(u,v) \,:\, u \in S} d[u] + \ell_e$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



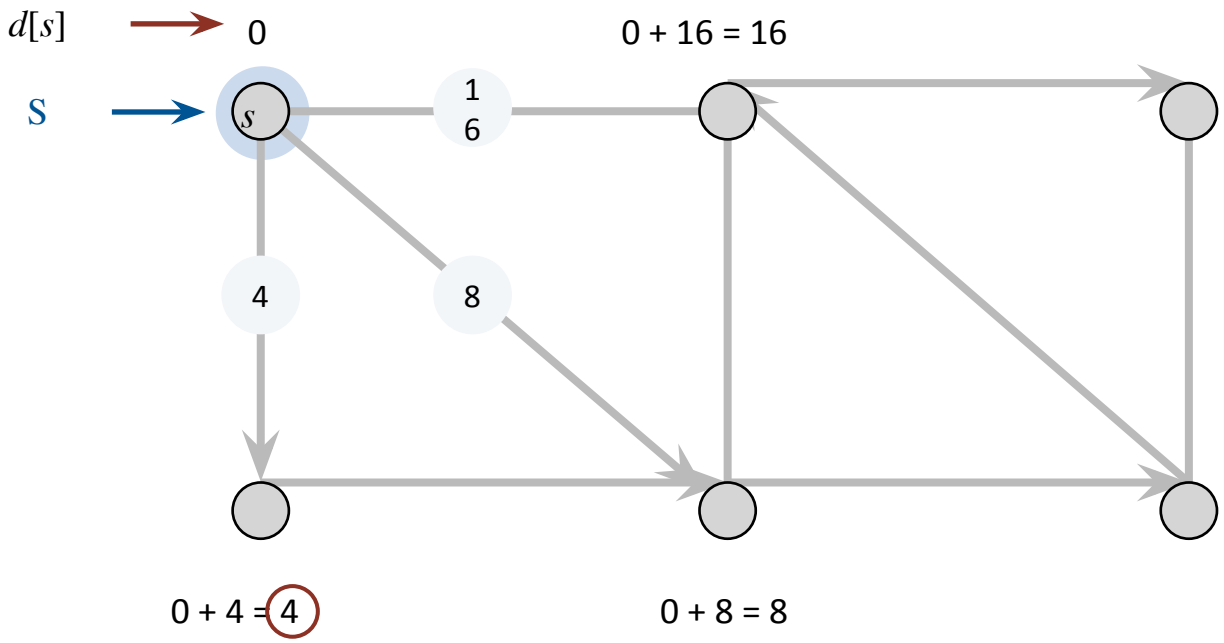
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e=(u,v) \,:\, u \in S} d[u] + \ell_e$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



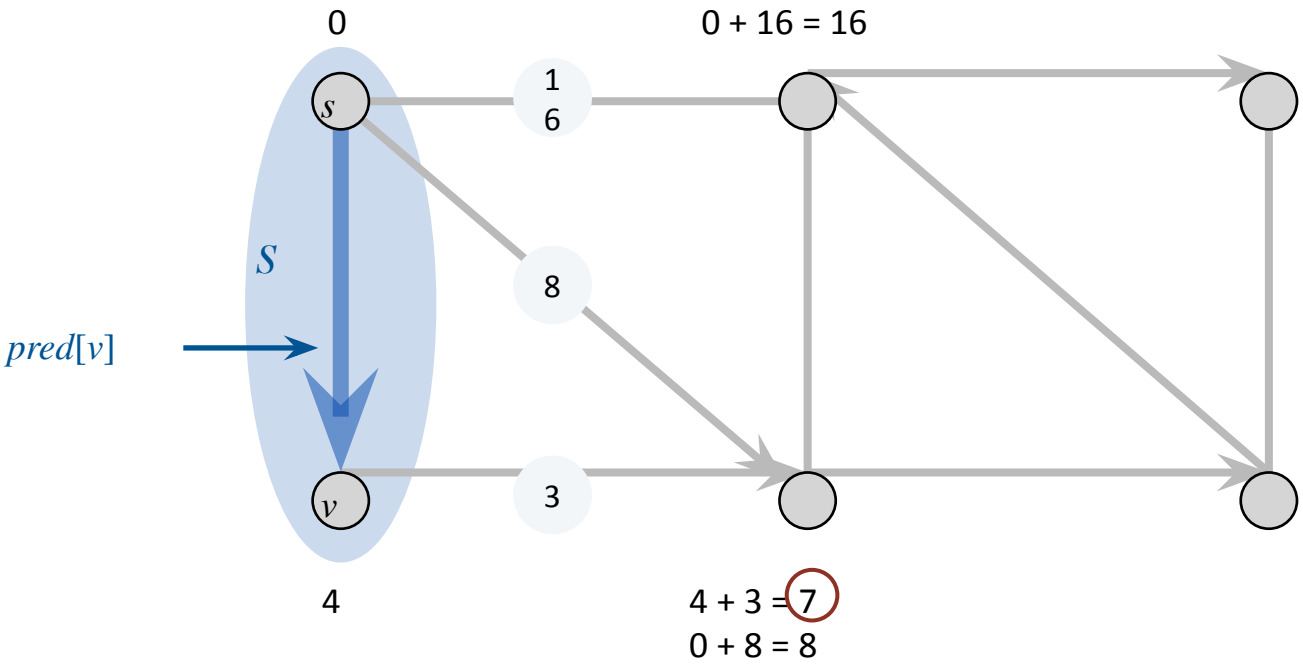
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e=(u,v) \,:\, u \in S} d[u] + \ell_e$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



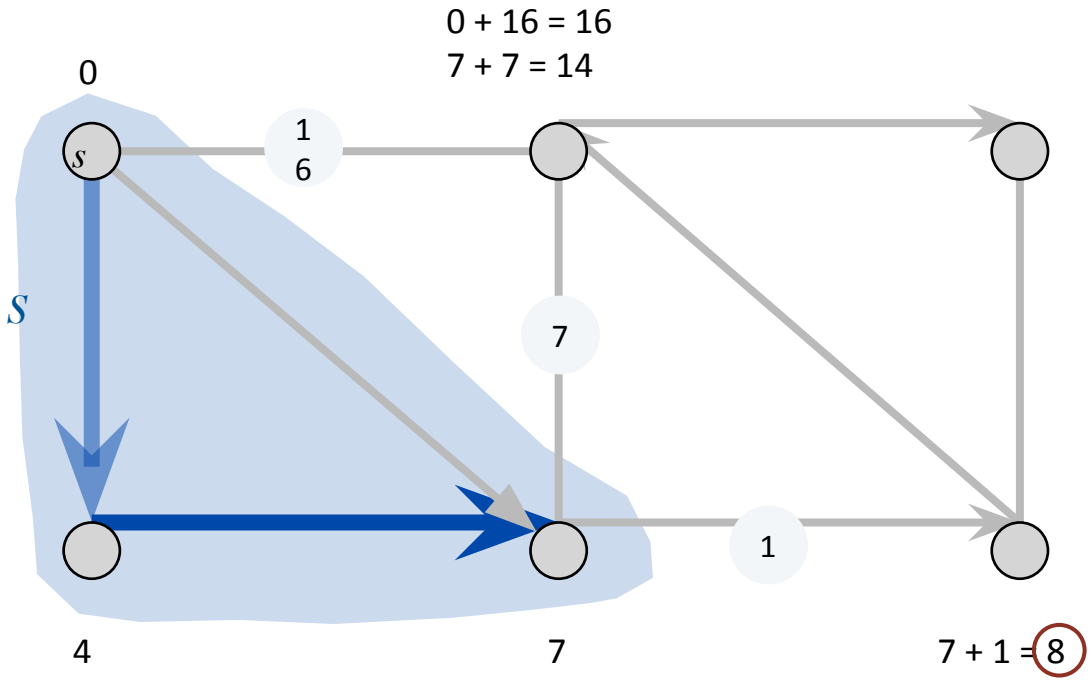
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e=(u,v) \,:\, u \in S} d[u] + \ell_e$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



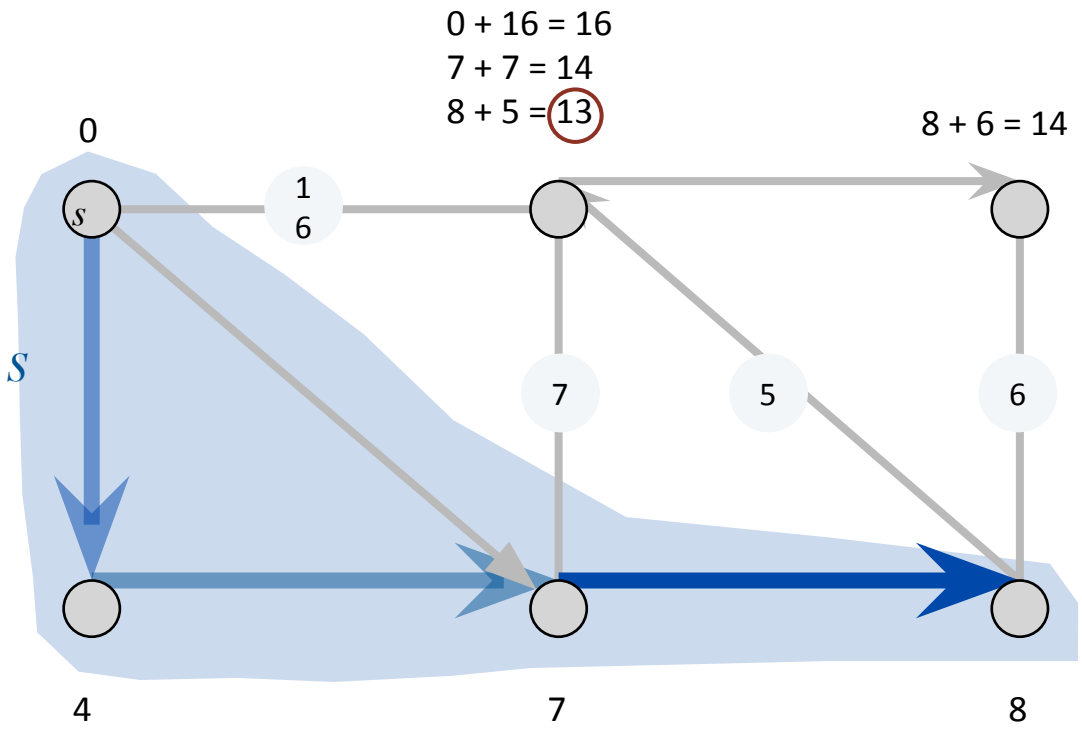
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e=(u,v) \,:\, u \in S} d[u] + \ell_e$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



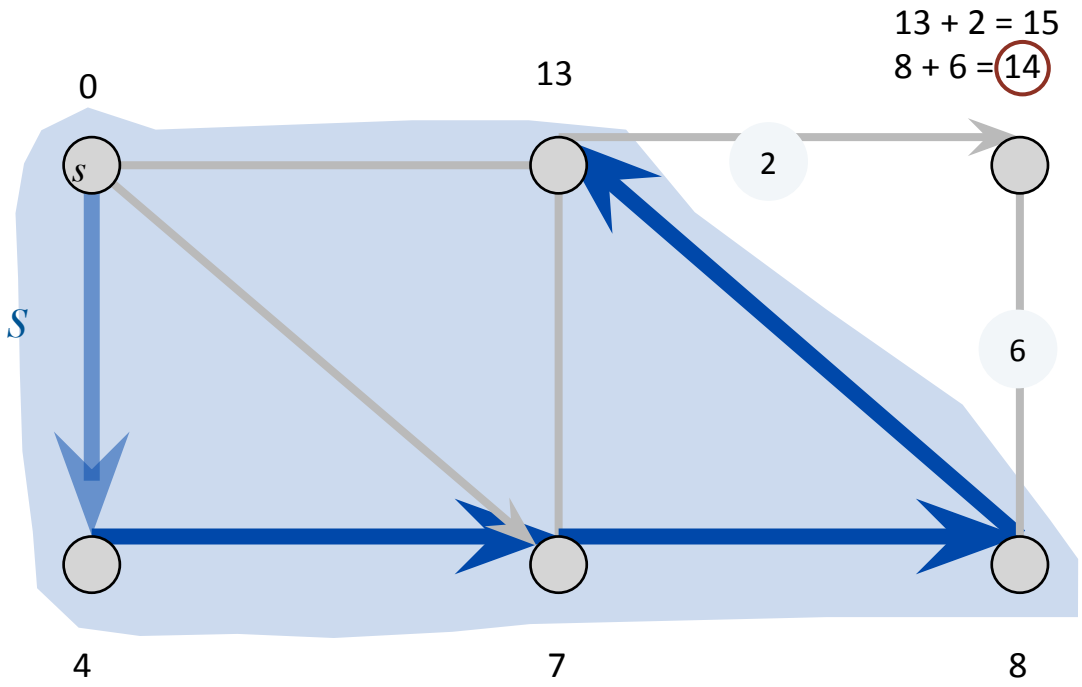
Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e=(u,v) \,:\, u \in S} d[u] + \ell_e$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.

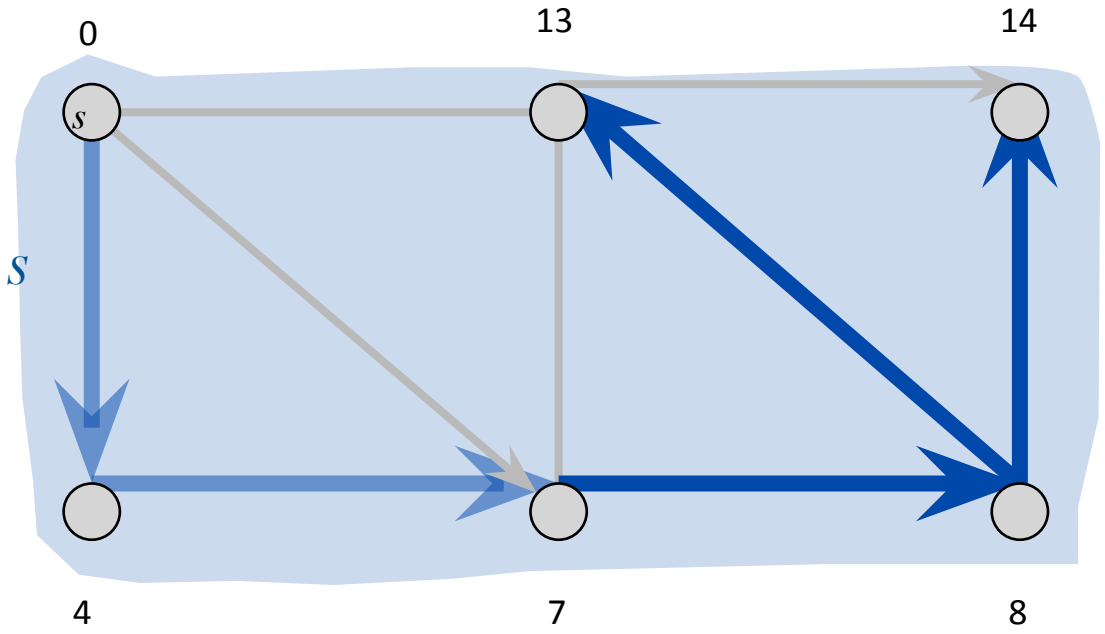


Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e=(u,v) \,:\, u \in S} d[u] + \ell_e$$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.

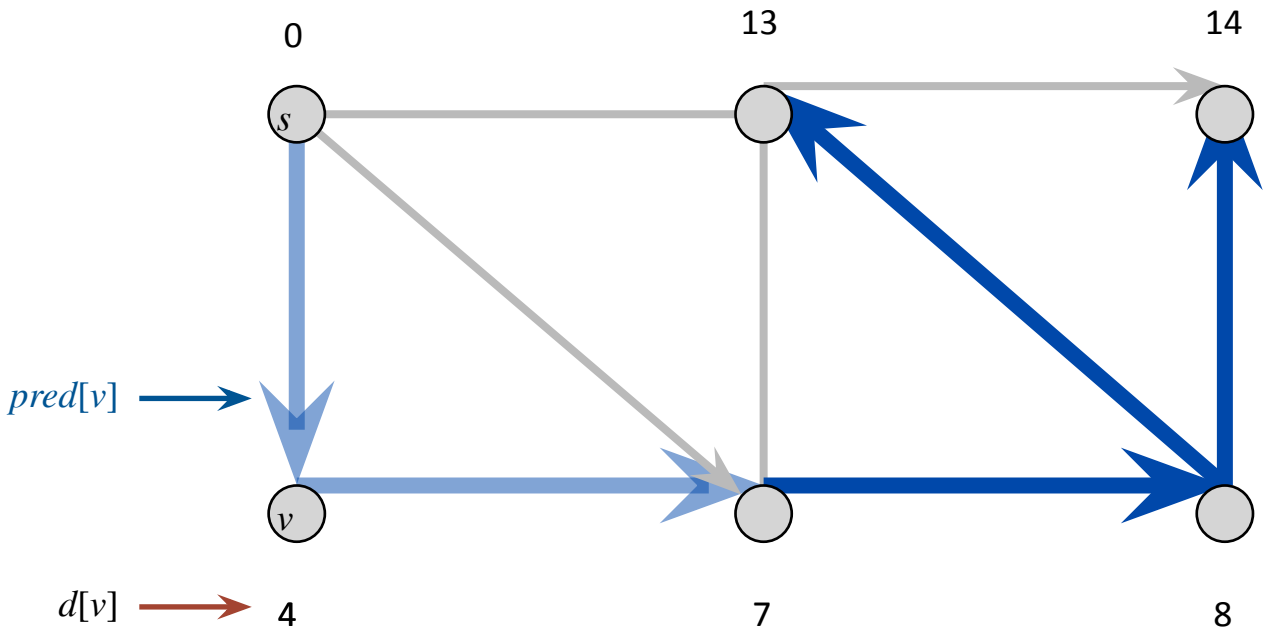


Dijkstra's algorithm demo

- Initialize $S \leftarrow \{ s \}$ and $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

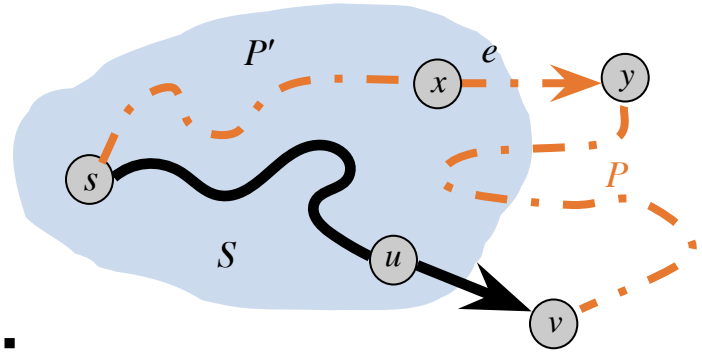
$$\pi(v) = \min_{e=(u,v) \,:\, u \in S} d[u] + \ell_e$$

add v to S; set $d[v] \leftarrow \pi(v)$ and $pred[v] \leftarrow$ argmin.



Dijkstra's algorithm: proof of correctness (greedy stays ahead)

- Invariant. For each node $u \in S$: $d[u]$ = length of a shortest $s \rightsquigarrow u$ path.
- Pf. [by induction on $|S|$]
- Base case: $|S| = 1$ is easy since $S = \{ s \}$ and $d[s] = 0$.
- Inductive hypothesis: Assume true for $|S| \geq 1$.
  - Let v be next node added to S, and let (u, v) be the final edge.
  - A shortest $s \rightsquigarrow u$ path plus (u, v) is an $s \rightsquigarrow v$ path of length $\pi(v)$.
  - Consider any other $s \rightsquigarrow v$ path P. We show that it is no shorter than $\pi(v)$.
  - Let $e = (x, y)$ be the first edge in P that leaves S, and let P' be the subpath from s to x.
  - The length of P is already $\geq \pi(v)$ as soon as it reaches y:



$$\ell(P) \geq \ell(P') + \ell_e \geq d[x] + \ell_e \geq \pi(y) \geq \pi(v) \quad \blacksquare$$

- $\ell(P) \geq \ell(P') + \ell_e$: non-negative lengths
- $\ell(P') + \ell_e \geq d[x] + \ell_e$: inductive hypothesis
- $d[x] + \ell_e \geq \pi(y)$: definition of $\pi(y)$
- $\pi(y) \geq \pi(v)$: Dijkstra chose v instead of y