Fundamental Algorithms & Data Structures

basic concepts
Dijkstra’s algorithm and priority queues
dynamic programming for the Traveling Salesperson Problem

Plan for this week

revisit preliminaries

today
- fundamental concepts for algorithms and data structures
- Dijkstras algorithm & priority queues for Shortest Paths Computation
- heuristic & dynamic programming for Traveling Salesperson Problem

Plan for this week

revisit preliminaries

today

- fundamental concepts for algorithms and data structures
- Dijkstras algorithm & priority queues for Shortest Paths Computation
- heuristic & dynamic programming for Traveling Salesperson Problem

next lecture

- amortized analysis

Plan for this week

revisit preliminaries

today

- fundamental concepts for algorithms and data structures
- Dijkstras algorithm & priority queues for Shortest Paths Computation
- heuristic & dynamic programming for Traveling Salesperson Problem

next lecture

- amortized analysis

If you need to refresh any of the concepts discussed today,
this week is a good time for it!

Prerequisites

- asymptotic analysis (O-notation, solving recurrences)

Prerequisites

- asymptotic analysis (O-notation, solving recurrences)
- amortized analysis (aggregate, accounting, potential method)

Prerequisites

- asymptotic analysis (O-notation, solving recurrences)
- amortized analysis (aggregate, accounting, potential method)
- fundamental data structures (heaps, binary search trees, hash tables)

Prerequisites

- asymptotic analysis (O-notation, solving recurrences)
- amortized analysis (aggregate, accounting, potential method)
- fundamental data structures (heaps, binary search trees, hash tables)
- fundamental algorithm paradigms (incremental, divide & conquer, dynamic programming)

Prerequisites

- asymptotic analysis (O-notation, solving recurrences)
- amortized analysis (aggregate, accounting, potential method)
- fundamental data structures (heaps, binary search trees, hash tables)
- fundamental algorithm paradigms (incremental, divide & conquer, dynamic programming)
- fundamental graph algorithms (BFS, DFS, shortest path, minimum spanning tree)

Prerequisites

- asymptotic analysis (O-notation, solving recurrences)
- amortized analysis (aggregate, accounting, potential method)
- fundamental data structures (heaps, binary search trees, hash tables)
- fundamental algorithm paradigms (incremental, divide & conquer, dynamic programming)
- fundamental graph algorithms (BFS, DFS, shortest path, minimum spanning tree)
- possibly algorithms for matching, flow (Ford-Fulkerson, MaxFlow-MinCut)

Prerequisites

- asymptotic analysis (O-notation, solving recurrences)
- amortized analysis (aggregate, accounting, potential method)
- fundamental data structures (heaps, binary search trees, hash tables)
- fundamental algorithm paradigms (incremental, divide & conquer, dynamic programming)
- fundamental graph algorithms (BFS, DFS, shortest path, minimum spanning tree)
- possibly algorithms for matching, flow (Ford-Fulkerson, MaxFlow-MinCut)
- NP-hardness (complexity class, reduction)

Prerequisites

- asymptotic analysis (O-notation, solving recurrences)
- amortized analysis (aggregate, accounting, potential method)
- fundamental data structures (heaps, binary search trees, hash tables)
- fundamental algorithm paradigms (incremental, divide & conquer, dynamic programming)
- fundamental graph algorithms (BFS, DFS, shortest path, minimum spanning tree)
- possibly algorithms for matching, flow (Ford-Fulkerson, MaxFlow-MinCut)
- NP-hardness (complexity class, reduction)
- possibly approximation algorithms (guarantees, techniques)

Prerequisites

- asymptotic analysis (O-notation, solving recurrences)
- amortized analysis (aggregate, accounting, potential method)
- fundamental data structures (heaps, binary search trees, hash tables)
- fundamental algorithm paradigms (incremental, divide & conquer, dynamic programming)
- fundamental graph algorithms (BFS, DFS, shortest path, minimum spanning tree)
- possibly algorithms for matching, flow (Ford-Fulkerson, MaxFlow-MinCut)
- NP-hardness (complexity class, reduction)
- possibly approximation algorithms (guarantees, techniques)

Which of these do you know?

Which lectures on algorithms have you taken?

Asymptotic Analysis

O-Notation: $O(f(n)) = \{g(n) \mid \exists c > 0 \; \exists n_0 \geq 1 \; \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$

Asymptotic Analysis

O-Notation: $O(f(n)) = \{g(n) \mid \exists c > 0 \, \exists n_0 \geq 1 \, \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$

Solving Recurrences: substitution, recursion trees, Master theorem

Asymptotic Analysis

O-Notation: $O(f(n)) = \{g(n) \mid \exists c > 0\ \exists n_0 \geq 1\ \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$

Solving Recurrences: substitution, recursion trees, Master theorem

Question: What is the runtime of this algorithm?

Algorithm dummy-alg(a, i, j)

$n \leftarrow j - i$
if $n \leq 1$ **then**
  return $a[i]$
x $\leftarrow$ dummy-alg$(a, i + 0, i + \lceil n/2 \rceil)$
y $\leftarrow$ dummy-alg$(a, i + \lceil n/4 \rceil, i + \lceil 3n/4 \rceil)$
z $\leftarrow$ dummy-alg$(a, i + \lceil n/2 \rceil, i + n)$
return $x + y + z$

Asymptotic Analysis

O-Notation: $O(f(n)) = \{g(n) \mid \exists c > 0 \, \exists n_0 \geq 1 \, \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$

Solving Recurrences: substitution, recursion trees, Master theorem

Question: What is the runtime of this algorithm?

Algorithm dummy-alg(a, i, j)

$n \leftarrow j - i$
if $n \leq 1$ **then**
  return $a[i]$
x $\leftarrow$ dummy-alg$(a, i + 0, i + \lceil n/2 \rceil)$
y $\leftarrow$ dummy-alg$(a, i + \lceil n/4 \rceil, i + \lceil 3n/4 \rceil)$
z $\leftarrow$ dummy-alg$(a, i + \lceil n/2 \rceil, i + n)$
return $x + y + z$

$$T(n) = 3T(n/2) + O(1)$$

Asymptotic Analysis

O-Notation: $O(f(n)) = \{g(n) \mid \exists c > 0\ \exists n_0 \geq 1\ \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$

Solving Recurrences: substitution, recursion trees, Master theorem

Question: What is the runtime of this algorithm?

Algorithm dummy-alg(a, i, j)

$n \leftarrow j - i$
if $n \leq 1$ **then**
  return $a[i]$
x $\leftarrow$ dummy-alg$(a, i+0, i + \lceil n/2 \rceil)$
y $\leftarrow$ dummy-alg$(a, i + \lceil n/4 \rceil, i + \lceil 3n/4 \rceil)$
z $\leftarrow$ dummy-alg$(a, i + \lceil n/2 \rceil, i + n)$
return $x + y + z$

$$T(n) = 3T(n/2) + O(1)$$

Master-Theorem:

$$T(n) = \Theta(n^{\log_2(3)})$$

Data structures

Abstract Data Types: Priority Queues, Sorted Sequences, Dictionaries

Data Structures: Heaps, Balanced Search Trees, Hash Tables

Data structures

Abstract Data Types: Priority Queues, Sorted Sequences, Dictionaries

Data Structures: Heaps, Balanced Search Trees, Hash Tables

Questions:

- Which of the data structures implement a priority queue efficiently?

Data structures

Abstract Data Types: Priority Queues, Sorted Sequences, Dictionaries

Data Structures: Heaps, Balanced Search Trees, Hash Tables

Questions:

- Which of the data structures implement a priority queue efficiently?
- What is the difference between the heap property and the binary search tree property?

Data structures

Abstract Data Types: Priority Queues, Sorted Sequences, Dictionaries

Data Structures: Heaps, Balanced Search Trees, Hash Tables

Questions:

- Which of the data structures implement a priority queue efficiently?
- What is the difference between the heap property and the binary search tree property?
- What is the difference between chaining and open addressing?

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure
- divide & conquer: divide input, solve on parts, combine solutions of parts

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure
- divide & conquer: divide input, solve on parts, combine solutions of parts
- dynamic programming: solve and store solutions to recurring subproblems

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure
- divide & conquer: divide input, solve on parts, combine solutions of parts
- dynamic programming: solve and store solutions to recurring subproblems
- greedy: find next element of solution based on simple criterion

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure
- divide & conquer: divide input, solve on parts, combine solutions of parts
- dynamic programming: solve and store solutions to recurring subproblems
- greedy: find next element of solution based on simple criterion

Quiz: what paradigm is insertion sort ?

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure
  $\rightarrow$ insertion sort
- divide & conquer: divide input, solve on parts, combine solutions of parts
- dynamic programming: solve and store solutions to recurring subproblems
- greedy: find next element of solution based on simple criterion

Quiz: what paradigm is selection sort ?

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure

  $\longrightarrow$ insertion sort
- divide & conquer: divide input, solve on parts, combine solutions of parts
- dynamic programming: solve and store solutions to recurring subproblems
- greedy: find next element of solution based on simple criterion

  $\longrightarrow$ selection sort

Quiz: what paradigm is merge sort ?

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure
  - $\rightarrow$ insertion sort
- divide & conquer: divide input, solve on parts, combine solutions of parts
  - $\rightarrow$ merge sort
- dynamic programming: solve and store solutions to recurring subproblems
- greedy: find next element of solution based on simple criterion
  - $\rightarrow$ selection sort

Quiz: what paradigm is quick sort ?

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure
  $\rightarrow$ insertion sort
- divide & conquer: divide input, solve on parts, combine solutions of parts
  $\rightarrow$ merge sort , quick sort
- dynamic programming: solve and store solutions to recurring subproblems
- greedy: find next element of solution based on simple criterion
  $\rightarrow$ selection sort

Quiz: what paradigm is Dijkstras algorithm ?

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure
  $\rightarrow$ insertion sort
- divide & conquer: divide input, solve on parts, combine solutions of parts
  $\rightarrow$ merge sort , quick sort
- dynamic programming: solve and store solutions to recurring subproblems
  $\rightarrow$ Dijkstra's algorithm (also Bellman-Ford and Floyd-Warshall)
- greedy: find next element of solution based on simple criterion
  $\rightarrow$ selection sort

Quiz: what paradigm is Jarnik-Prim and Kruskal's algorithm ?

Algorithm Paradigms

Common paradigms

- incremental: insert elements one-by-one and update structure
  $\longrightarrow$ insertion sort
- divide & conquer: divide input, solve on parts, combine solutions of parts
  $\longrightarrow$ merge sort , quick sort
- dynamic programming: solve and store solutions to recurring subproblems
  $\longrightarrow$ Dijkstra's algorithm (also Bellman-Ford and Floyd-Warshall)
- greedy: find next element of solution based on simple criterion
  $\longrightarrow$ selection sort , Jarnik-Prim & Kruskals algorithms

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Example: $n = 4$ and profits

length i	1	2	3	4
profit p_i	1	5	8	9

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Example: $n = 4$ and profits

length i	1	2	3	4
profit p_i	1	5	8	9

what is an optimal cutting here?

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Example: $n = 4$ and profits

length i	1	2	3	4
profit p_i	1	5	8	9

an optimal cutting is into two pieces of length 2, with total profit 10

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Example: $n = 4$ and profits

length i	1	2	3	4
profit p_i	1	5	8	9

an optimal cutting is into two pieces of length 2, with total profit 10

how do we find an optimal cutting?

how many possible cuttings are there?

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Example: $n = 4$ and profits

length i	1	2	3	4
profit p_i	1	5	8	9

an optimal cutting is into two pieces of length 2, with total profit 10

how do we find an optimal cutting?

how many possible cuttings are there? $\rightarrow$ there are 2^{n-1} possible cuttings

(less but still superpolynomial if ordered by size)

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Example: $n = 4$ and profits

length i	1	2	3	4
profit p_i	1	5	8	9

an optimal cutting is into two pieces of length 2, with total profit 10

how do we find an optimal cutting?

how many possible cuttings are there? $\rightarrow$ there are 2^{n-1} possible cuttings

(less but still superpolynomial if ordered by size)

straight-forward recursive solution:

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Example: $n = 4$ and profits

length i	1	2	3	4
profit p_i	1	5	8	9

an optimal cutting is into two pieces of length 2, with total profit 10

how do we find an optimal cutting?

how many possible cuttings are there? $\rightarrow$ there are 2^{n-1} possible cuttings

(less but still superpolynomial if ordered by size)

straight-forward recursive solution:

let $r(n)$ = maximum profit for length n then $r(n) = \max_{1 \leq i \leq n}\{p_i + r(n-i)\}$

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n)$ = maximum profit for length n then $r(n) = \max_{1 \leq i \leq n}\{p_i + r(n-i)\}$

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n) =$ maximum profit for length n then $r(n) = \max_{1 \le i \le n}\{p_i + r(n-i)\}$

how to compute this efficiently?

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n)$ = maximum profit for length n then $r(n) = \max_{1 \leq i \leq n}\{p_i + r(n-i)\}$

Algorithm CUT-ROD (p, n)

if $n = 0$ **then**
 ⌊ **return** 0

$q = -\infty$

for $i = 1$ **to** n **do**
 ⌊ $q = \max\{q, p_i + \text{CUT-ROD}(p, n-i)\}$

return q

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n) =$ maximum profit for length n then $r(n) = \max_{1 \leq i \leq n}\{p_i + r(n-i)\}$

Algorithm CUT-ROD (p, n)

what is the runtime?

if $n = 0$ **then**
  return 0
$q = -\infty$
for $i = 1$ **to** n **do**
  $q = \max\{q, p_i + \text{CUT-ROD}(p, n-i)\}$
return q

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n)$ = maximum profit for length n then $r(n) = \max_{1 \le i \le n}\{p_i + r(n-i)\}$

Algorithm CUT-ROD (p, n)

if $n = 0$ **then**
  └ **return** 0
$q = -\infty$
for $i = 1$ **to** n **do**
  └ $q = \max\{q, p_i + \text{CUT-ROD}(p, n-i)\}$
return q

what is the runtime?

$T(0) = 1$

$T(n) = 1 + \sum_{i=1}^{n-1} T(i)$

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n)$ = maximum profit for length n then $r(n) = \max_{1 \leq i \leq n}\{p_i + r(n-i)\}$

Algorithm CUT-ROD (p, n)

if $n = 0$ **then**
  return 0
$q = -\infty$
for $i = 1$ **to** n **do**
  $q = \max\{q, p_i + \text{CUT-ROD}(p, n-i)\}$
return q

what is the runtime?

$T(0) = 1$

$T(n) = 1 + \sum_{i=1}^{n-1} T(i)$

what does this solve to?

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n)$ = maximum profit for length n then $r(n) = \max_{1 \leq i \leq n}\{p_i + r(n-i)\}$

Algorithm CUT-ROD (p, n)

if $n = 0$ **then**
 ⌊ **return** 0
$q = -\infty$
for $i = 1$ **to** n **do**
 ⌊ $q = \max\{q, p_i +$ CUT-ROD$(p, n-i)\}$
return q

what is the runtime?

$T(0) = 1$

$T(n) = 1 + \sum_{i=1}^{n-1} T(i)$

what does this solve to?

$T(n) = 2^n$

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n) =$ maximum profit for length n then $r(n) = \max_{1 \leq i \leq n}\{p_i + r(n-i)\}$

Algorithm CUT-ROD (p, n)

if $n = 0$ **then**
  ⌊ **return** 0
$q = -\infty$
for $i = 1$ **to** n **do**
  ⌊ $q = \max\{q, p_i +$ CUT-ROD$(p, n-i)\}$
return q

what is the runtime?

$T(0) = 1$

$T(n) = 1 + \sum_{i=1}^{n-1} T(i)$

what does this solve to?

$T(n) = 2^n$

why is the algorithm so slow?

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n) =$ maximum profit for length n then $r(n) = \max_{1 \le i \le n}\{p_i + r(n-i)\}$

Algorithm CUT-ROD (p, n)

if $n = 0$ **then**
  $\lfloor$ **return** 0
$q = -\infty$
for $i = 1$ **to** n **do**
  $\lfloor$ $q = \max\{q, p_i + \text{CUT-ROD}(p, n-i)\}$
return q

what is the runtime?

$T(0) = 1$

$T(n) = 1 + \sum_{i=1}^{n-1} T(i)$

what does this solve to?

$T(n) = 2^n$

why is the algorithm so slow?

$\rightarrow$ repeated computations

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n) =$ maximum profit for length n then $r(n) = \max_{1 \leq i \leq n}\{p_i + r(n-i)\}$

Algorithm CUT-ROD (p, n)

if $n = 0$ **then**
 └ **return** 0
$q = -\infty$
for $i = 1$ **to** n **do**
 └ $q = \max\{q, p_i +$ CUT-ROD$(p, n-i)\}$
return q

what is the runtime?

$T(0) = 1$

$T(n) = 1 + \sum_{i=1}^{n-1} T(i)$

what does this solve to?

$T(n) = 2^n$

how to do this faster?

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

straight-forward recursive solution:

let $r(n) =$ maximum profit for length n then $r(n) = \max_{1 \leq i \leq n}\{p_i + r(n-i)\}$

Algorithm CUT-ROD (p, n)

> **if** $n = 0$ **then**
> $\quad$ **return** 0
> $q = -\infty$
> **for** $i = 1$ **to** n **do**
> $\quad$ $q = \max\{q, p_i + \text{CUT-ROD}(p, n-i)\}$
> **return** q

what is the runtime?

$T(0) = 1$

$T(n) = 1 + \sum_{i=1}^{n-1} T(i)$

what does this solve to?

$T(n) = 2^n$

how to do this faster? $\rightarrow$ memoization
$\rightarrow$ bottom-up

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Algorithm BOTTOM-UP-CUT-ROD (p, n)

```
r[0, n] new array
r[0] = 0
for j = 1 to n do
    q = −∞
    for i = 1 to j do
        q = max{q, p_i + r(j − i)}
    r[j] = q
return r[n]
```

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Algorithm BOTTOM-UP-CUT-ROD (p, n)

what is the runtime?

$r[0, n]$ new array
$r[0] = 0$
for $j = 1$ **to** n **do**
  $q = -\infty$
  for $i = 1$ **to** j **do**
    $q = \max\{q, p_i + r(j - i)\}$
  $r[j] = q$
return $r[n]$

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Algorithm BOTTOM-UP-CUT-ROD (p, n)

$r[0, n]$ new array
$r[0] = 0$
for $j = 1$ **to** n **do**
  $q = -\infty$
  for $i = 1$ **to** j **do**
    $q = \max\{q, p_i + r(j - i)\}$
  $r[j] = q$
return $r[n]$

what is the runtime?

$O(n^2)$

$\rightarrow$ *much faster than* 2^n

Interlude: Dynamic Program for Cutting Rods

Problem: Given a steel rod of length n, cut it into pieces to maximize the profit, where profit for a piece of length i is given as p_i.

Algorithm BOTTOM-UP-CUT-ROD (p, n)

$r[0, n]$ new array
$r[0] = 0$
for $j = 1$ **to** n **do**
  $q = -\infty$
  for $i = 1$ **to** j **do**
    $q = \max\{q, p_i + r(j-i)\}$
  $r[j] = q$
return $r[n]$

what is the runtime?

$O(n^2)$

$\rightarrow$ *much faster than* 2^n

Dynamic Programs apply to problems that have overlapping optimal substructures

Graph algorithms

graph representations: adjacency matrix; adjacency list

graph algorithms: traversal (BFS, DFS), shortest paths, minimum spanning tree

Graph algorithms

graph representations: adjacency matrix; adjacency list

graph algorithms: traversal (BFS, DFS), shortest paths, minimum spanning tree

Questions:

- What is a good graph representation for breadth-first search?

Graph algorithms

graph representations: adjacency matrix; adjacency list

graph algorithms: traversal (BFS, DFS), shortest paths, minimum spanning tree

Questions:

- What is a good graph representation for breadth-first search? adj. lists

Graph algorithms

graph representations: adjacency matrix; adjacency list

graph algorithms: traversal (BFS, DFS), shortest paths, minimum spanning tree

Questions:

- What is a good graph representation for breadth-first search? adj. lists
- Which algorithm is used for topological sorting?

Graph algorithms

graph representations: adjacency matrix; adjacency list

graph algorithms: traversal (BFS, DFS), shortest paths, minimum spanning tree

Questions:

- What is a good graph representation for breadth-first search? adj. lists
- Which algorithm is used for topological sorting? DFS

Graph algorithms

graph representations: adjacency matrix; adjacency list

graph algorithms: traversal (BFS, DFS), shortest paths, minimum spanning tree

Questions:

- What is a good graph representation for breadth-first search? adj. lists
- Which algorithm is used for topological sorting? DFS
- How many edges does a tree on n vertices have?

Graph algorithms

graph representations: adjacency matrix; adjacency list

graph algorithms: traversal (BFS, DFS), shortest paths, minimum spanning tree

Questions:

- What is a good graph representation for breadth-first search? adj. lists
- Which algorithm is used for topological sorting? DFS
- How many edges does a tree on n vertices have? $n - 1$

Graph algorithms

graph representations: adjacency matrix; adjacency list

graph algorithms: traversal (BFS, DFS), shortest paths, minimum spanning tree

Questions:

- What is a good graph representation for breadth-first search? adj. lists
- Which algorithm is used for topological sorting? DFS
- How many edges does a tree on n vertices have? $n - 1$
- How fast can one compute a minimum spanning tree?

Graph algorithms

graph representations: adjacency matrix; adjacency list

graph algorithms: traversal (BFS, DFS), shortest paths, minimum spanning tree

Questions:

- What is a good graph representation for breadth-first search? adj. lists
- Which algorithm is used for topological sorting? DFS
- How many edges does a tree on n vertices have? $n-1$
- How fast can one compute a minimum spanning tree?
  - Kruskal: $O(|E|\log|V|)$
  - Jarnik-Prim: $O(|E|+|V|\log|V|)$ with Fibonacci heaps
  - faster algorithms exist

NP-hardness

complexity class P: all problems that can be solved in polynomial time by a deterministic turing machine

complexity class NP: all problems that can be solved in polynomial time by a non-deterministic turing machine

NP-hardness

complexity class P: all problems that can be solved in polynomial time by a deterministic turing machine

complexity class NP: all problems that can be solved in polynomial time by a non-deterministic turing machine

famous open problem: $P \subsetneq NP$?

NP-hardness

complexity class P: all problems that can be solved in polynomial time by a deterministic turing machine

complexity class NP: all problems that can be solved in polynomial time by a non-deterministic turing machine

famous open problem: $P \subsetneq NP$?

Questions:

- What is an NP-hard problem? What is an NP-complete problem?

NP-hardness

complexity class P: all problems that can be solved in polynomial time by a deterministic turing machine

complexity class NP: all problems that can be solved in polynomial time by a non-deterministic turing machine

famous open problem: $P \subsetneq NP$?

Questions:

- What is an NP-hard problem? What is an NP-complete problem?
- Which NP-complete problems do you know?

NP-hardness

complexity class P: all problems that can be solved in polynomial time by a deterministic turing machine

complexity class NP: all problems that can be solved in polynomial time by a non-deterministic turing machine

famous open problem: $P \subsetneq NP$?

Questions:

- What is an NP-hard problem? What is an NP-complete problem?
- Which NP-complete problems do you know?
- How do you prove NP-hardness of a problem?

NP-hardness

complexity class P: all problems that can be solved in polynomial time by a deterministic turing machine

complexity class NP: all problems that can be solved in polynomial time by a non-deterministic turing machine

famous open problem: $P \subsetneq NP$?

Questions:

- What is an NP-hard problem? What is an NP-complete problem?
- Which NP-complete problems do you know?
- How do you prove NP-hardness of a problem?
- How to solve NP-hard problems?

Two well-known problems

shortest path

What is the shortest path from Hamburg to Munich?

shortest round trip

What is the shortest round trip through these major German cities?

Shortest Path Problem

Common Algorithms

- Dijkstras Algorithm
- Bellmann-Ford
- Floyd-Warshall

Wikipedia

Shortest Path Problem

Common Algorithms

- Dijkstras Algorithm
- Bellmann-Ford
- Floyd-Warshall

What are advantages and disadvantages of the algorithms?

Shortest Path Problem

Common Algorithms

- Dijkstras Algorithm
- Bellmann-Ford
- Floyd-Warshall

What are advantages and disadvantages of the algorithms?

What else?

- many more approaches in practice
- shortest path queries make heavy use of data structures

Wikipedia

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

  $u \leftarrow$ such a node with minimal distance $d[u]$

  for all outgoing edges (u, v)

    check and possibly update distance to v

  mark u as *finished*

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

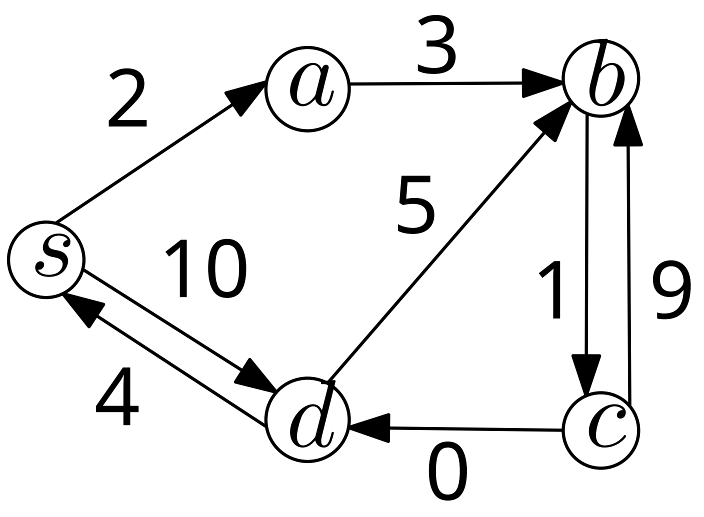
$u \leftarrow$ such a node with minimal distance $d[u]$

for all outgoing edges (u, v)

check and possibly update distance to v

mark u as *finished*

store these nodes in priority queue Q



Q

	d	parent
s		
a		
b		
c		
d		

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

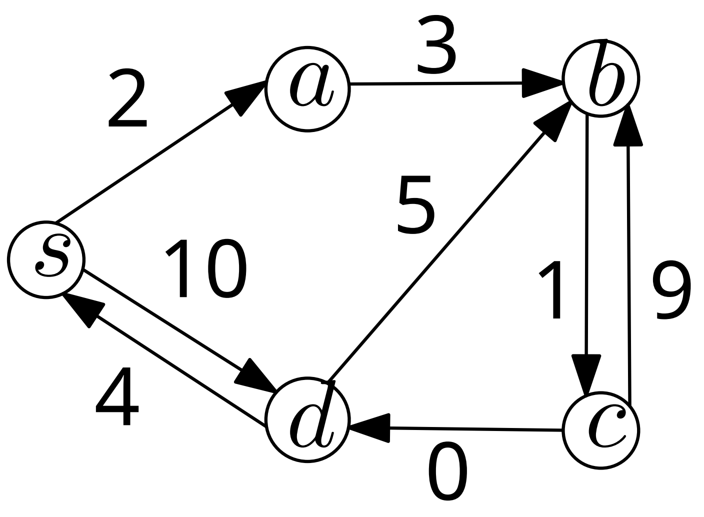
  $u \leftarrow$ such a node with minimal distance $d[u]$

  for all outgoing edges (u, v)

    check and possibly update distance to v

  mark u as *finished*

store these nodes in priority queue Q



Q
$(s, 0)$

init

	d	parent
s	0	s
a	∞	$\perp$
b	∞	$\perp$
c	∞	$\perp$
d	∞	$\perp$

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

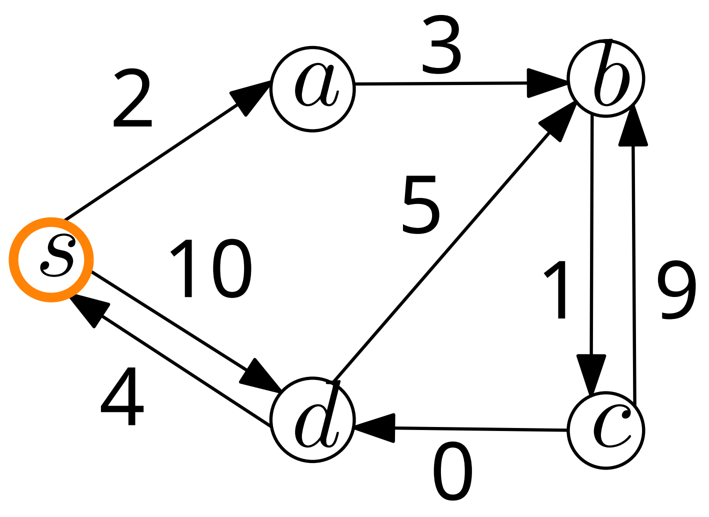
  $u \leftarrow$ such a node with minimal distance $d[u]$

  for all outgoing edges (u, v)

    check and possibly update distance to v

  mark u as *finished*

store these nodes in priority queue Q



Q
$(a, 2)$
$(d, 10)$

$u \leftarrow s$

	d	parent
s	0	s
a	2	s
b	∞	$\perp$
c	∞	$\perp$
d	10	s

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

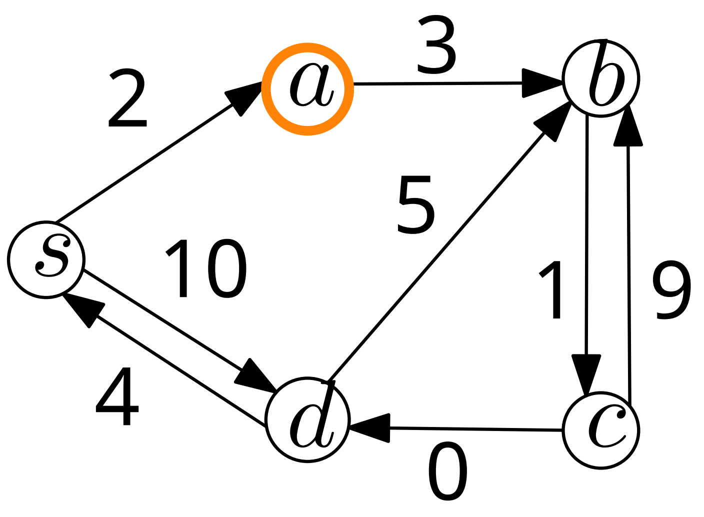
  $u \leftarrow$ such a node with minimal distance $d[u]$

  for all outgoing edges (u, v)

    check and possibly update distance to v

  mark u as *finished*

store these nodes
in priority queue Q



Q
$(b, 5)$
$(d, 10)$

$u \leftarrow a$

	d	parent
s	0	s
a	2	s
b	5	a
c	∞	$\perp$
d	10	s

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

$u \leftarrow$ such a node with minimal distance $d[u]$

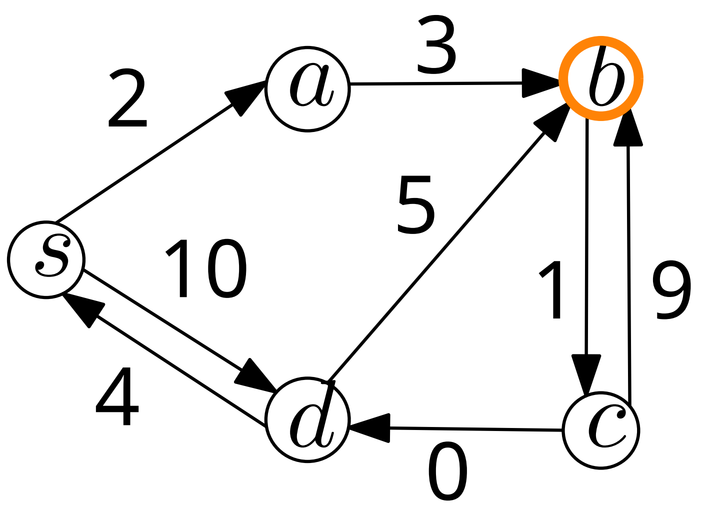
for all outgoing edges (u, v)

check and possibly update distance to v

mark u as *finished*

store these nodes

in priority queue Q



Q
$(c, 6)$
$(d, 10)$

$u \leftarrow b$

	d	parent
s	0	s
a	2	s
b	5	a
c	6	b
d	10	s

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

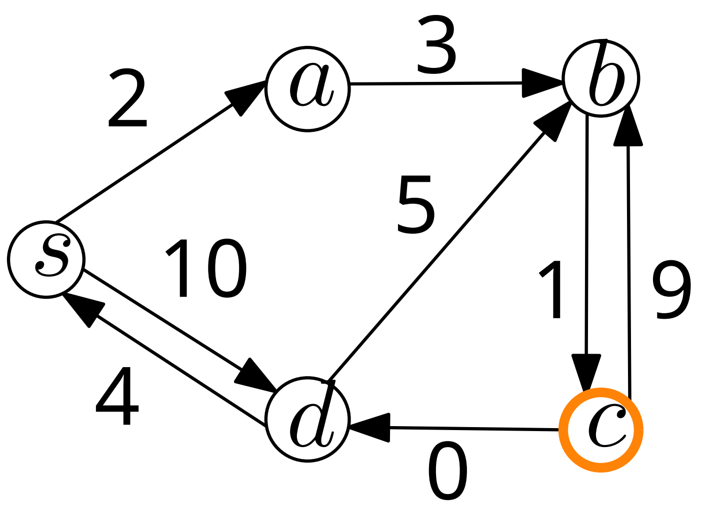
  $u \leftarrow$ such a node with minimal distance $d[u]$

  for all outgoing edges (u, v)

    check and possibly update distance to v

  mark u as *finished*

store these nodes in priority queue Q



Q
$(d, 6)$

$u \leftarrow c$

	d	parent
s	0	s
a	2	s
b	5	a
c	6	b
d	6	c

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

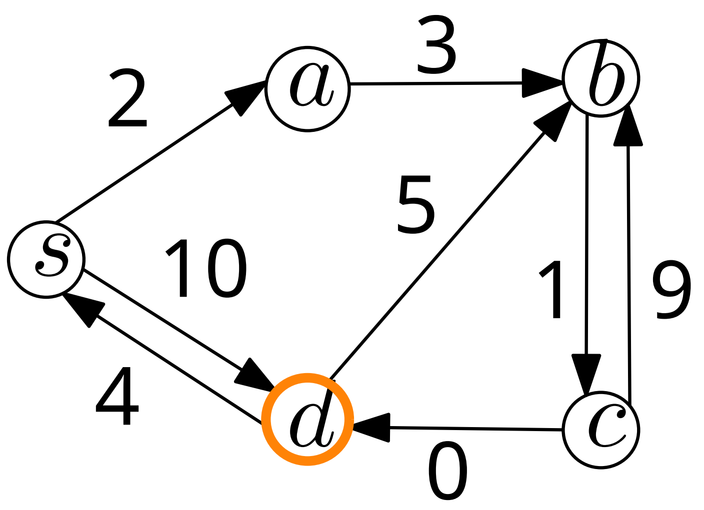
  $u \leftarrow$ such a node with minimal distance $d[u]$

  for all outgoing edges (u, v)

    check and possibly update distance to v

  mark u as *finished*

store these nodes
in priority queue Q



Q

$u \leftarrow d$

	d	parent
s	0	s
a	2	s
b	5	a
c	6	b
d	6	c

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

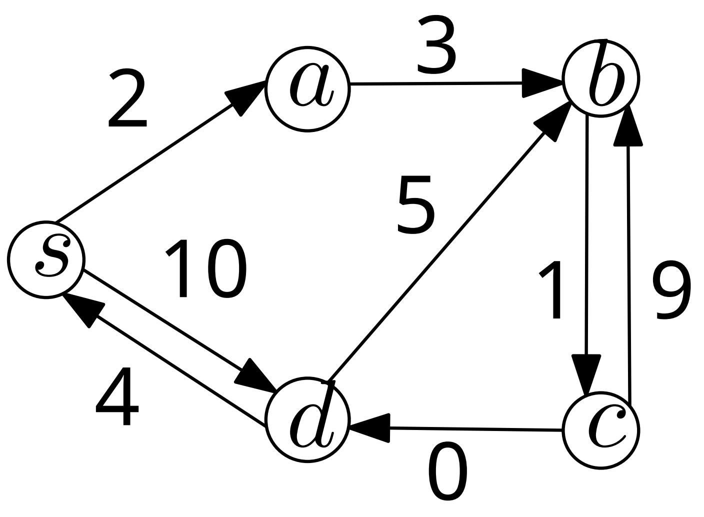
  $u \leftarrow$ such a node with minimal distance $d[u]$

  for all outgoing edges (u, v)

    check and possibly update distance to v

  mark u as *finished*

store these nodes in priority queue Q



Q

	d	parent
s	0	s
a	2	s
b	5	a
c	6	b
d	6	c

Dijkstras Algorithm

Algorithm

unmark all nodes;

init arrays d and $parent$;

while there is unmarked node u with $d[u] < \infty$ **do**

  $u \leftarrow$ such a node with minimal distance $d[u]$

  for all outgoing edges (u, v)

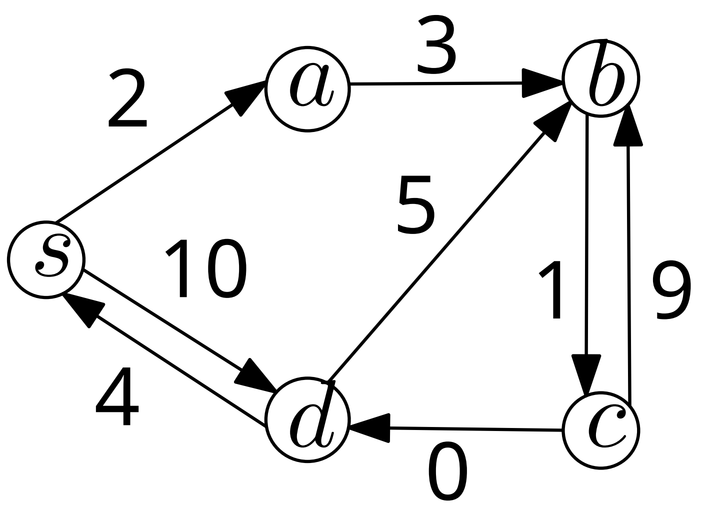
    check and possibly update distance to v

  mark u as *finished*

store these nodes

in priority queue Q

What is the runtime?



Q

	d	parent
s	0	s
a	2	s
b	5	a
c	6	b
d	6	c

Dijkstras Algorithm

Algorithm

unmark all nodes;
init arrays d and $parent$;
while there is unmarked node u with $d[u] < \infty$ **do**
  $u \leftarrow$ such a node with minimal distance $d[u]$
  for all outgoing edges (u, v)
    check and possibly update distance to v
  mark u as *finished*

store these nodes in priority queue Q

What is the runtime?

$$O(m \cdot \underbrace{T_{decreaseKey}(n)}_{\text{each edge may decrease the distance of a node in } Q} + n \cdot (\underbrace{T_{deleteMin}(n) + T_{insert}(n)}_{\text{each node is inserted and deleted once in } Q}))$$

Dijkstras Algorithm

Algorithm

unmark all nodes;
init arrays d and $parent$;
while there is unmarked node u with $d[u] < \infty$ **do**
  $u \leftarrow$ such a node with minimal distance $d[u]$
  for all outgoing edges (u, v)
    check and possibly update distance to v
  mark u as *finished*

store these nodes in priority queue Q

What is the runtime? $O(m \cdot \underbrace{T_{decreaseKey}(n)}_{\text{each edge may decrease the distance of a node in } Q} + n \cdot (\underbrace{T_{deleteMin}(n) + T_{insert}(n)}_{\text{each node is inserted and deleted once in } Q}))$

depends on implementation of priority queue Q

Priority Queues

Abstract data typ: manage a set of elements with keys (their priority) under the operations *insert, minimum, deleteMinimum*, and optionally *decreaseKey, remove* and *merge*.

Priority Queues

Abstract data typ: manage a set of elements with keys (their priority) under the operations *insert, minimum, deleteMinimum*, and optionally *decreaseKey, remove* and *merge*.

Implementations:

Priority Queues

Abstract data typ: manage a set of elements with keys (their priority) under the operations *insert, minimum, deleteMinimum*, and optionally *decreaseKey, remove* and *merge*.

Implementations:

	deleteMin	decreaseKey	insert	build
Binary heaps				
Balanced Search Trees				
Fibonaccis Heaps				

Priority Queues

Abstract data typ: manage a set of elements with keys (their priority) under the operations *insert, minimum, deleteMinimum*, and optionally *decreaseKey, remove* and *merge*.

Implementations:

	deleteMin	decreaseKey	insert	build
Binary heaps	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
Balanced Search Trees	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
Fibonaccis Heaps				

Priority Queues

Abstract data typ: manage a set of elements with keys (their priority) under the operations *insert, minimum, deleteMinimum*, and optionally *decreaseKey, remove* and *merge*.

Implementations:

	deleteMin	decreaseKey	insert	build
Binary heaps	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
Balanced Search Trees	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
Fibonaccis Heaps	$O(\log n)^*$	$O(1)^*$	$O(1)$	$O(n)$

* amortised

Priority Queues

Abstract data typ: manage a set of elements with keys (their priority) under the operations *insert, minimum, deleteMinimum*, and optionally *decreaseKey, remove* and *merge*.

Implementations:

	deleteMin	decreaseKey	insert	build
Binary heaps	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
Balanced Search Trees	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
Fibonaccis Heaps	$O(\log n)^*$	$O(1)^*$	$O(1)$	$O(n)$

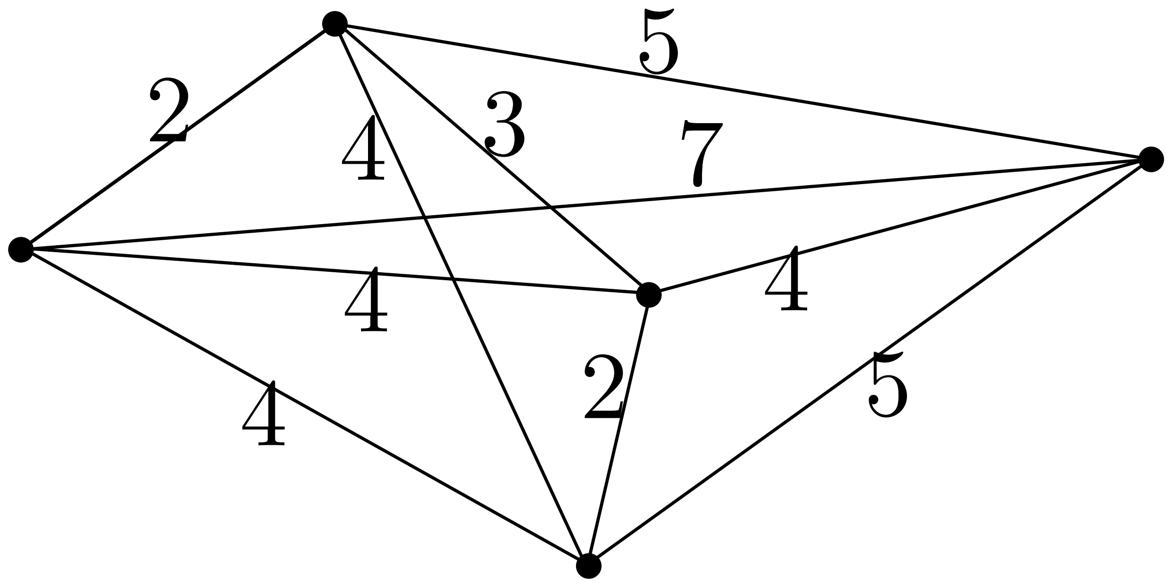
* amortised

Runtime for Dijkstras Algorithm: $O(m + n \log n)$ using Fibonacci heaps

Traveling Salesperson Problem

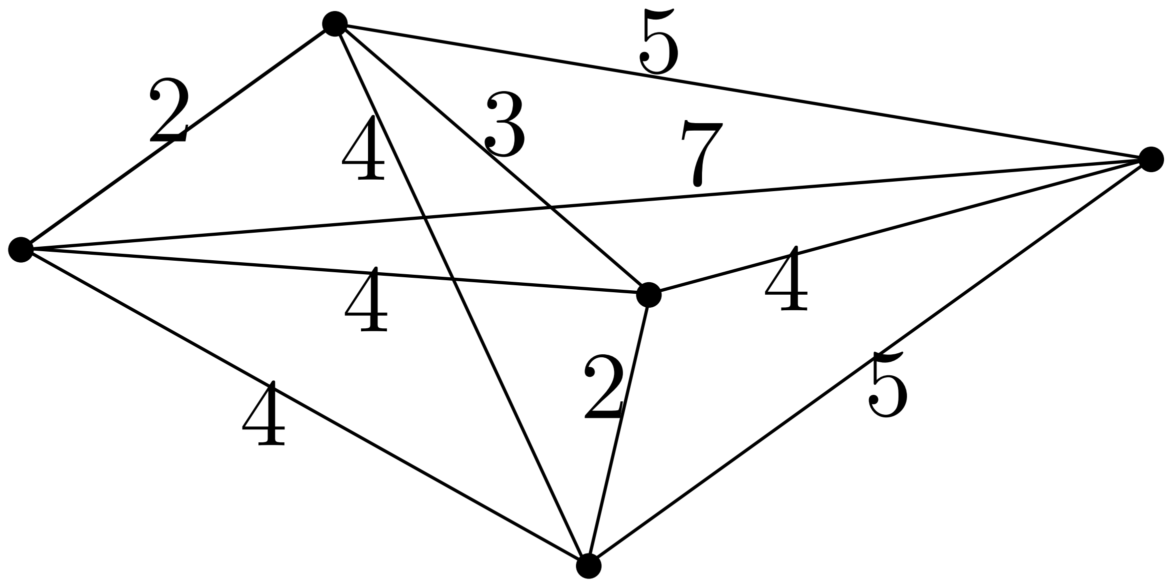
Traveling Salesperson Problem

Given an undirected complete Graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}$, find a shortest Hamilton circuit in G.



Traveling Salesperson Problem

Given an undirected complete Graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}$, find a shortest Hamilton circuit in G.

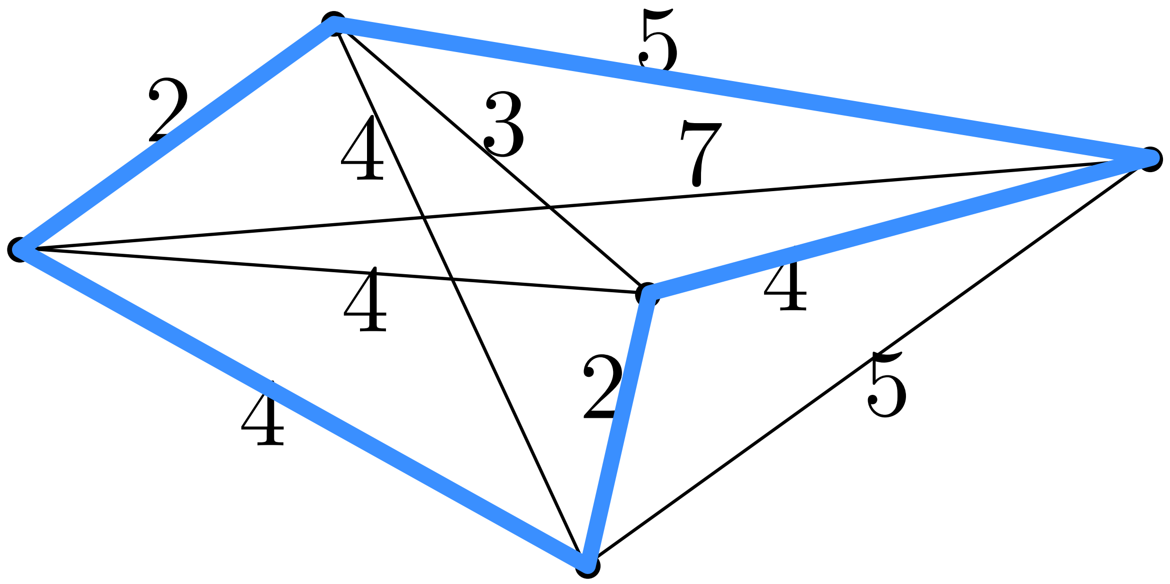


what is it here?

Traveling Salesperson Problem

Wikipedia

Given an undirected complete Graph $G = (V, E)$ with edge weights $c \colon E \to \mathbb{R}$, find a shortest Hamilton circuit in G.



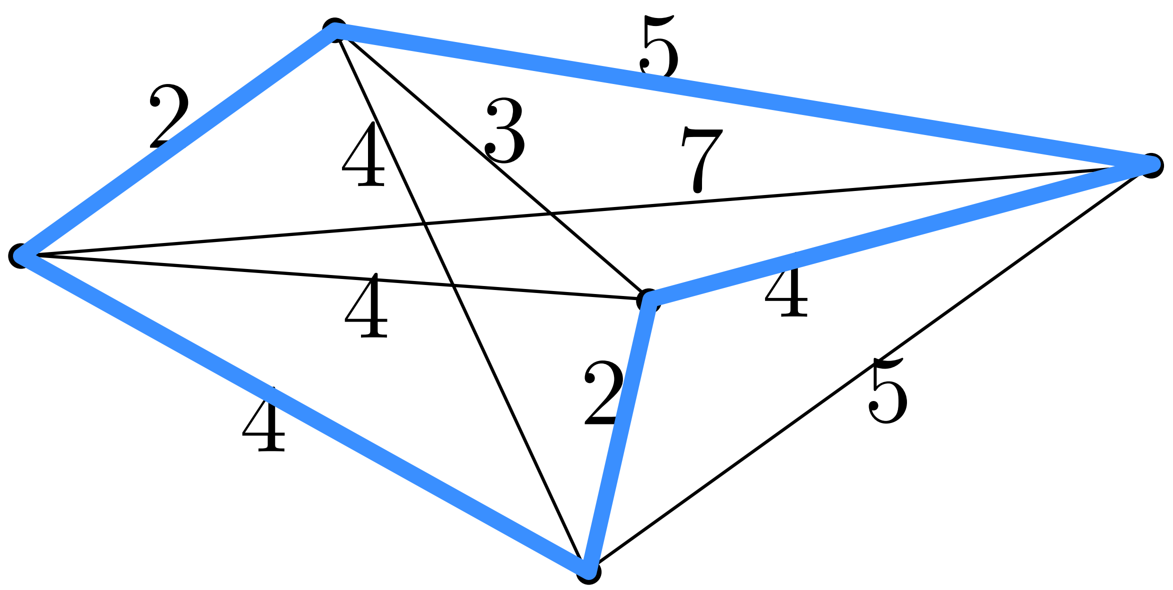
what is it here?

Traveling Salesperson Problem

NP-hardness

- for approximating general version
- for deciding metric version

Given an undirected complete Graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}$, find a shortest Hamilton circuit in G.



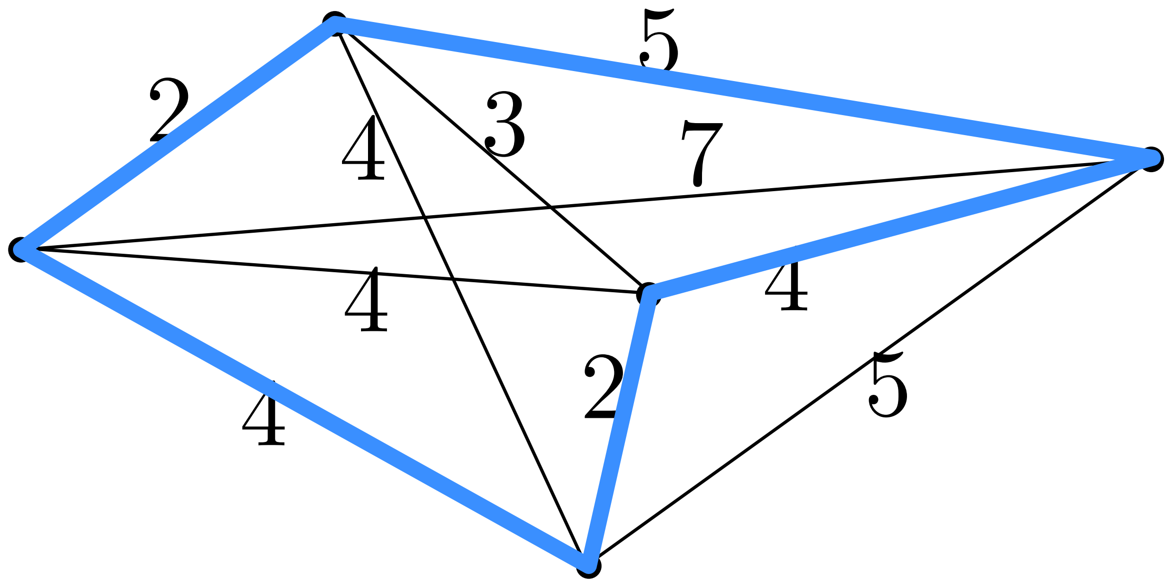
Traveling Salesperson Problem

NP-hardness

- for approximating general version
- for deciding metric version

what now?

Given an undirected complete Graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}$, find a shortest Hamilton circuit in G.



Traveling Salesperson Problem

NP-hardness

- for approximating general version
- for deciding metric version

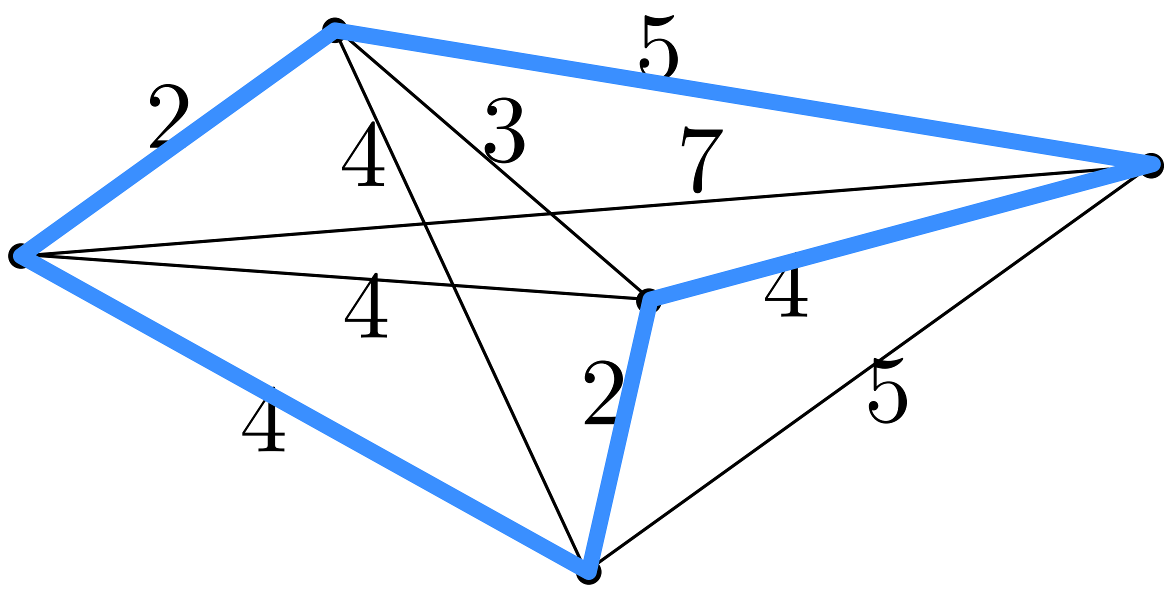
what now?

Exact Algorithms

Approximation algorithms

Heuristics

Given an undirected complete Graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}$, find a shortest Hamilton circuit in G.



Traveling Salesperson Problem

NP-hardness

- for approximating general version
- for deciding metric version

what now?

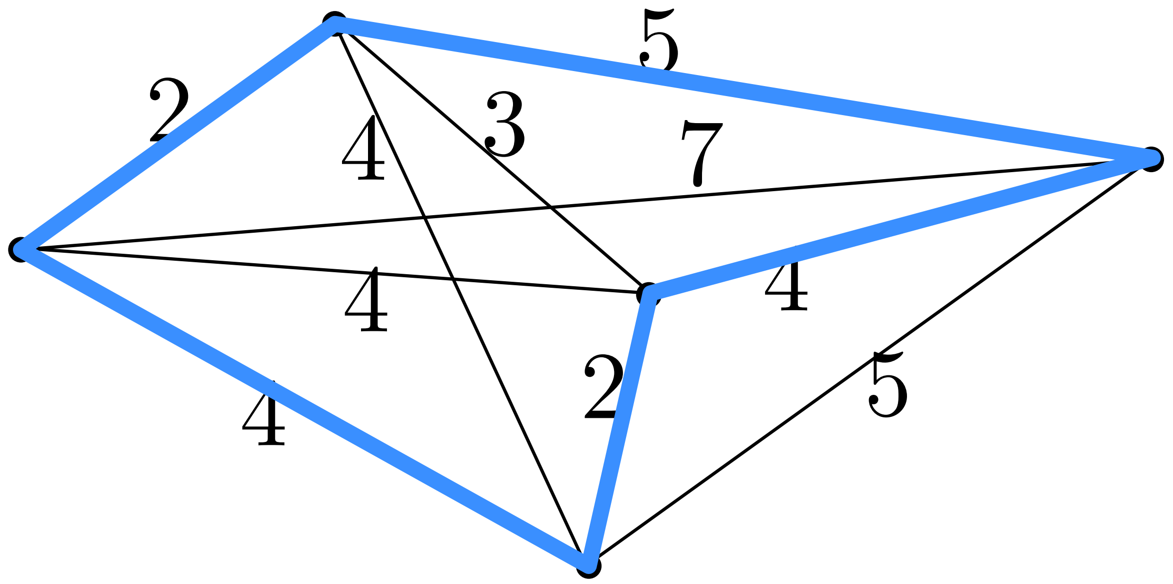
Exact Algorithms

- Held-Karp algorithm (dynamic program)
- ILP formulation

Approximation algorithms

Heuristics

Given an undirected complete Graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}$, find a shortest Hamilton circuit in G.



Traveling Salesperson Problem

NP-hardness

- for approximating general version
- for deciding metric version

what now?

Exact Algorithms

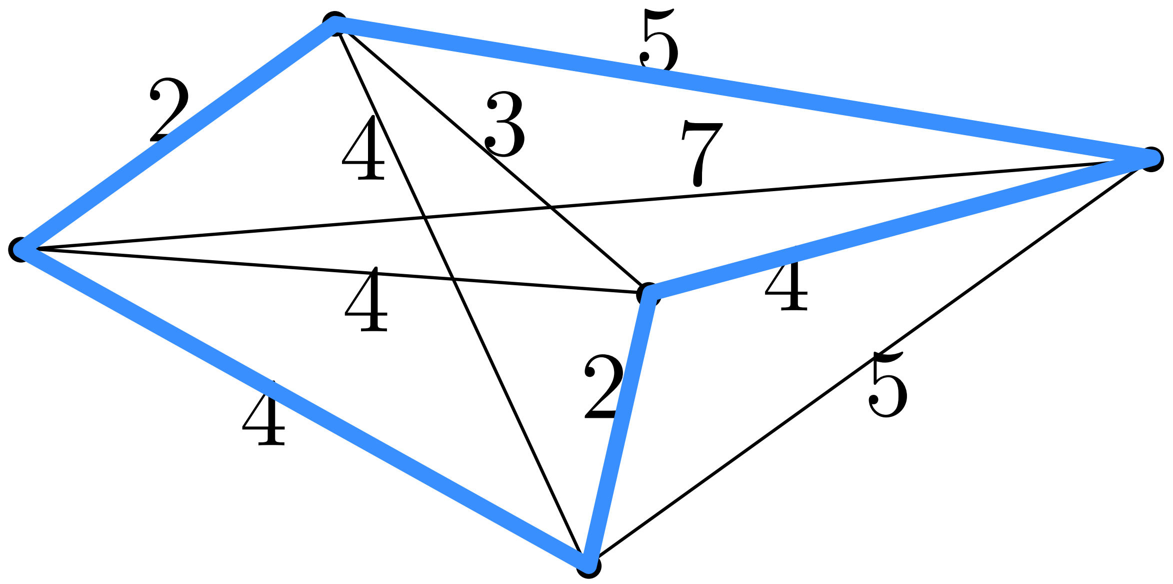
- Held-Karp algorithm (dynamic program)
- ILP formulation

Approximation algorithms

- 3/2-Approx for Metric TSP (Christofides)
- PTAS for Euclidean TSP (Arora, Mitchel)

Heuristics

Given an undirected complete Graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}$, find a shortest Hamilton circuit in G.



Traveling Salesperson Problem

NP-hardness

- for approximating general version
- for deciding metric version

what now?

Exact Algorithms

- Held-Karp algorithm (dynamic program)
- ILP formulation

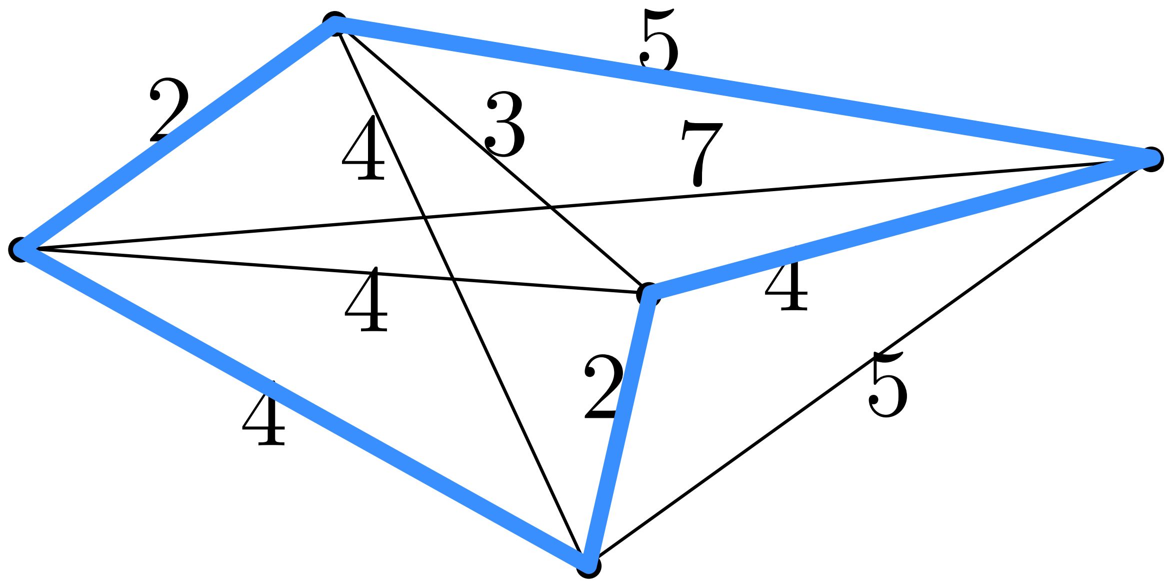
Approximation algorithms

- 3/2-Approx for Metric TSP (Christofides)
- PTAS for Euclidean TSP (Arora, Mitchel)

Heuristics

- Nearest Neighbor
- 2-OPT and many more

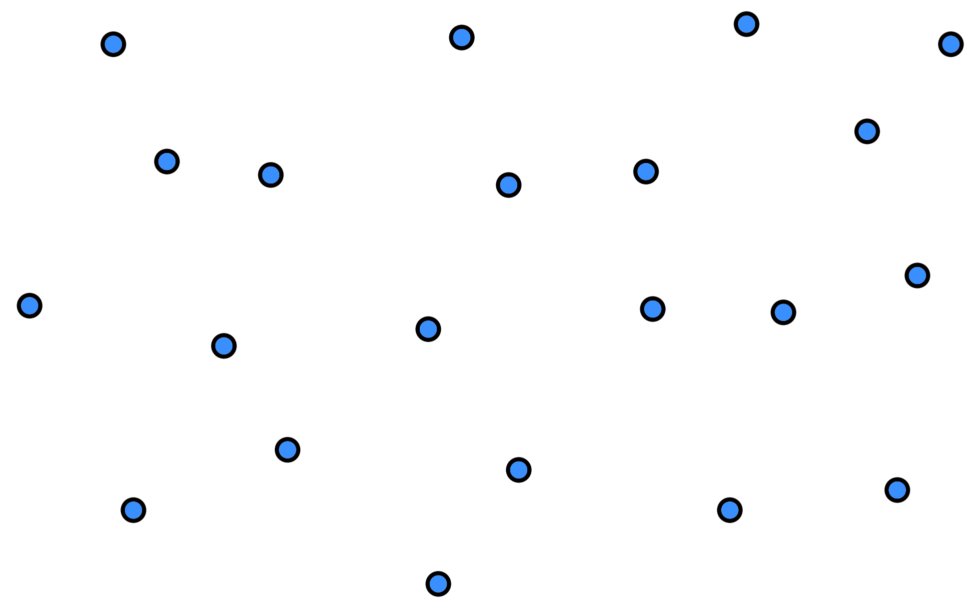
Given an undirected complete Graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}$, find a shortest Hamilton circuit in G.



Heuristics for TSP

Example: Nearest-Neighbor Heuristic

- start at an arbitrary city
- always go the nearest unvisited city
- at the end: go back to starting city



Heuristics for TSP

Example: Nearest-Neighbor Heuristic

Algorithm Nearest-Neighbor Heuristic$(G = (V, E), c)$

Initialize tour as empty

Select arbitrary $s \in V$ (as starting vertex)

Set $v = s$ and $U = V - s$

while $U \neq \emptyset$ **do**

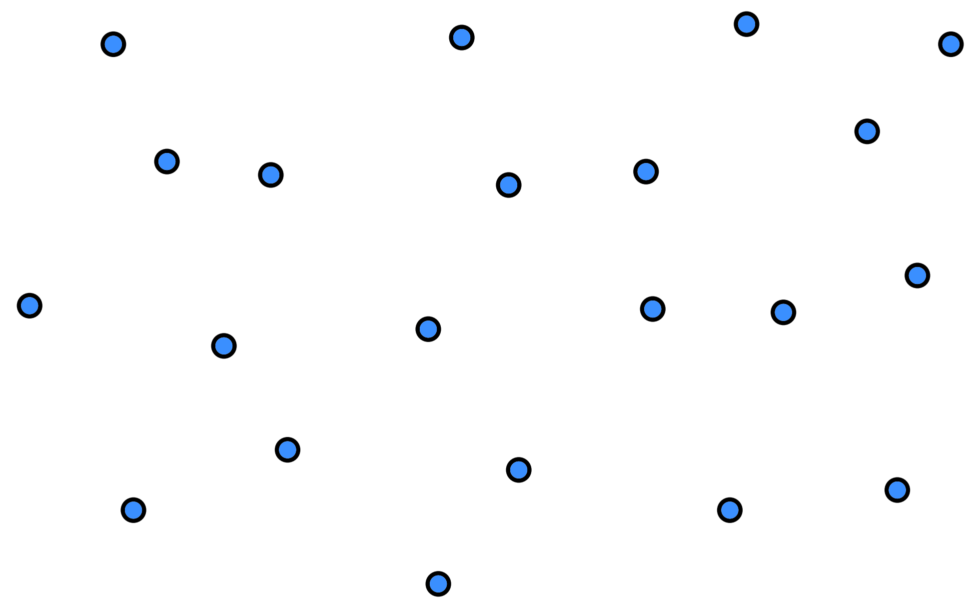
  Select $u \in U$ with $c(v, u) = \min_{w \in U} c(v, w)$

  Add edge {v, u} to tour

  Set $v = u$ and $U = U - u$

Add edge $\{v, s\}$ to tour

return tour



Heuristics for TSP

Example: Nearest-Neighbor Heuristic

Algorithm Nearest-Neighbor Heuristic$(G = (V, E), c)$

Initialize tour as empty

Select arbitrary $s \in V$ (as starting vertex)

Set $v = s$ and $U = V - s$

while $U \neq \emptyset$ **do**

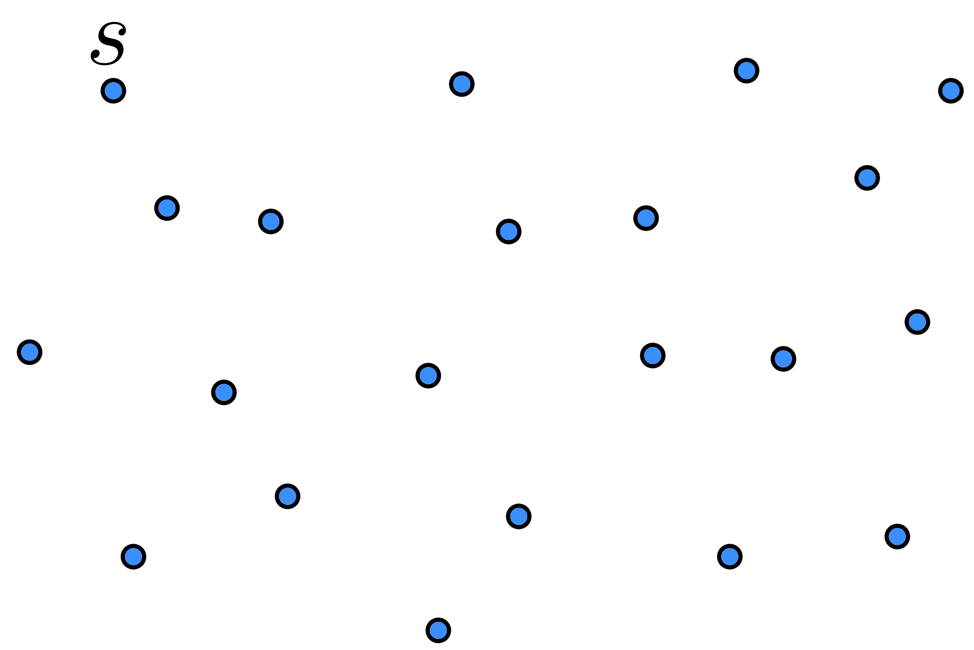
  Select $u \in U$ with $c(v, u) = \min_{w \in U} c(v, w)$

  Add edge {v, u} to tour

  Set $v = u$ and $U = U - u$

Add edge $\{v, s\}$ to tour

return tour



Heuristics for TSP

Example: Nearest-Neighbor Heuristic

Algorithm Nearest-Neighbor Heuristic$(G = (V, E), c)$

Initialize tour as empty

Select arbitrary $s \in V$ (as starting vertex)

Set $v = s$ and $U = V - s$

while $U \neq \emptyset$ **do**

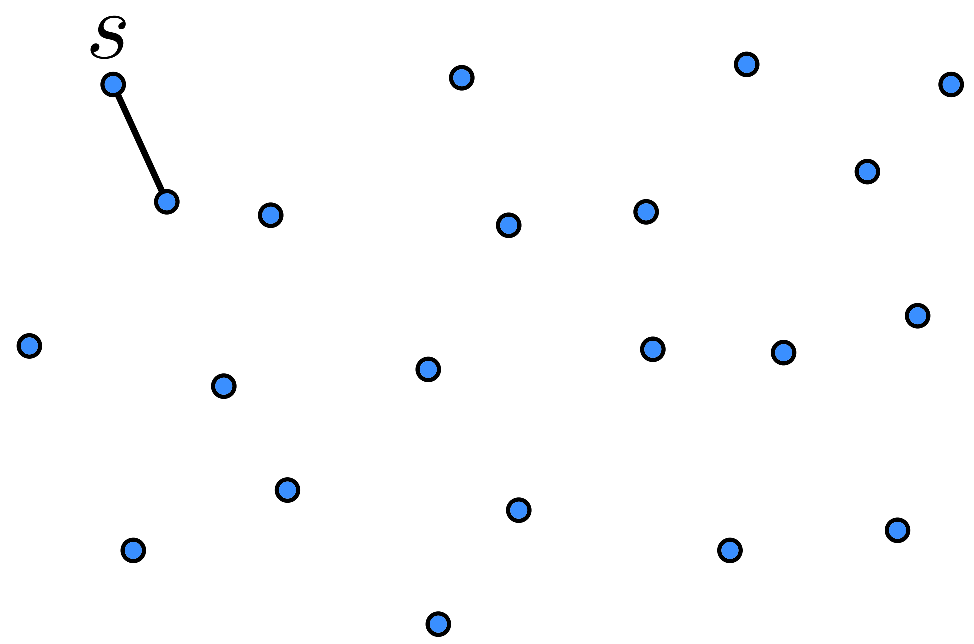
  Select $u \in U$ with $c(v, u) = \min_{w \in U} c(v, w)$

  Add edge {v, u} to tour

  Set $v = u$ and $U = U - u$

Add edge $\{v, s\}$ to tour

return tour



Heuristics for TSP

Example: Nearest-Neighbor Heuristic

Algorithm Nearest-Neighbor Heuristic$(G = (V, E), c)$

Initialize tour as empty

Select arbitrary $s \in V$ (as starting vertex)

Set $v = s$ and $U = V - s$

while $U \neq \emptyset$ **do**

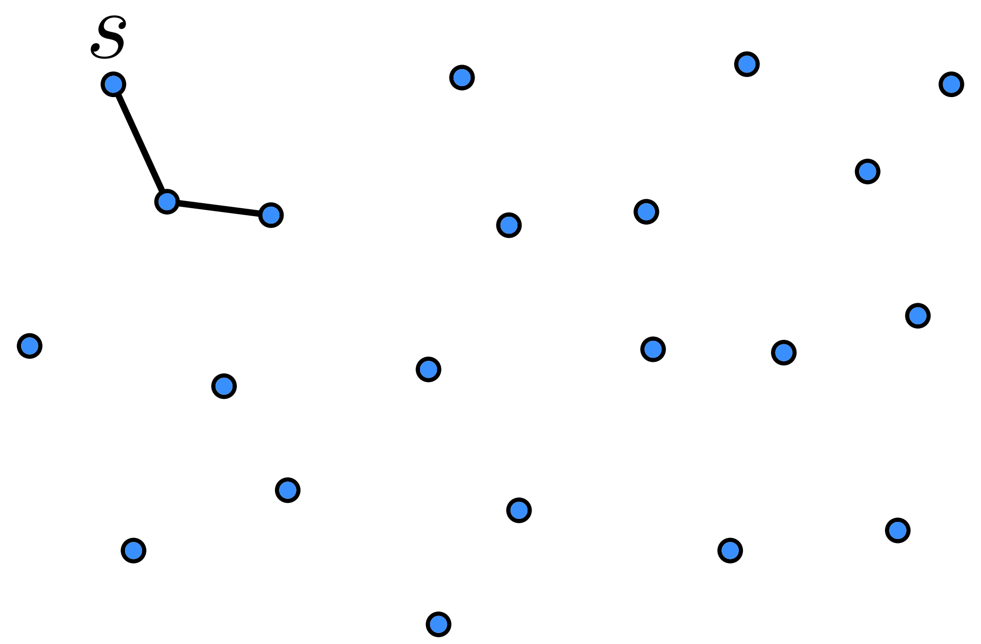
Select $u \in U$ with $c(v, u) = \min_{w \in U} c(v, w)$

Add edge {v, u} to tour

Set $v = u$ and $U = U - u$

Add edge $\{v, s\}$ to tour

return tour



Heuristics for TSP

Example: Nearest-Neighbor Heuristic

Algorithm Nearest-Neighbor Heuristic$(G = (V, E), c)$

Initialize tour as empty

Select arbitrary $s \in V$ (as starting vertex)

Set $v = s$ and $U = V - s$

while $U \neq \emptyset$ **do**

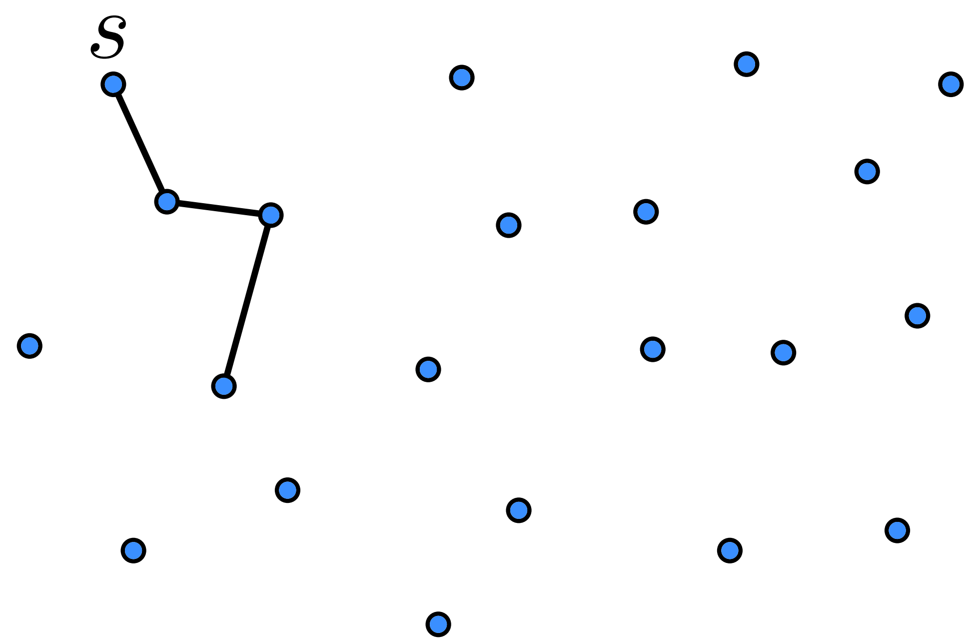
  Select $u \in U$ with $c(v, u) = \min_{w \in U} c(v, w)$

  Add edge {v, u} to tour

  Set $v = u$ and $U = U - u$

Add edge $\{v, s\}$ to tour

return tour



Heuristics for TSP

Example: Nearest-Neighbor Heuristic

Algorithm Nearest-Neighbor Heuristic$(G = (V, E), c)$

Initialize tour as empty

Select arbitrary $s \in V$ (as starting vertex)

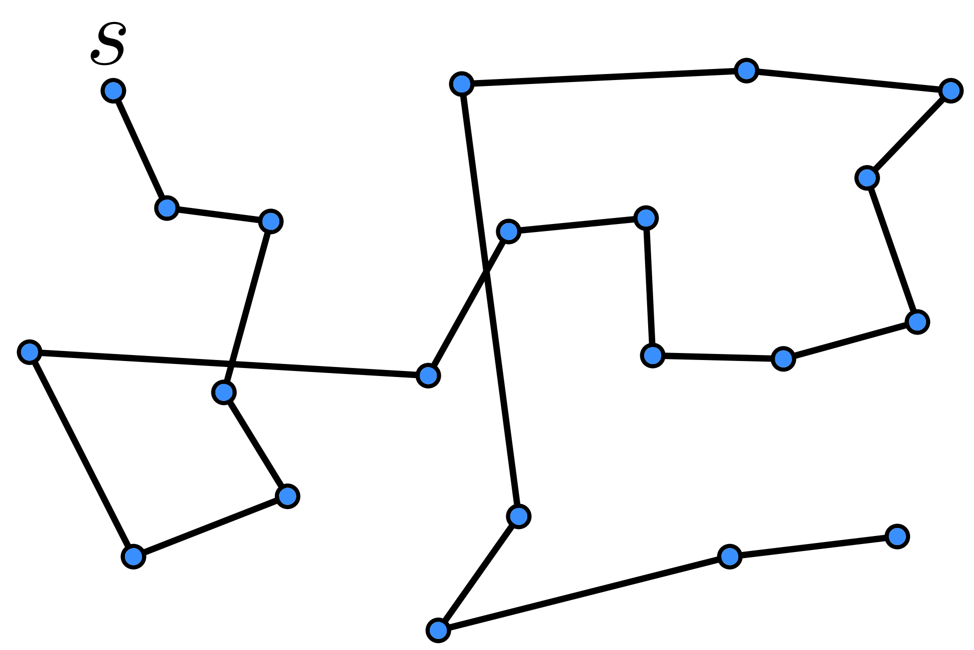
Set $v = s$ and $U = V - s$

while $U \neq \emptyset$ **do**

- Select $u \in U$ with $c(v, u) = \min_{w \in U} c(v, w)$
- Add edge {v, u} to tour
- Set $v = u$ and $U = U - u$

Add edge $\{v, s\}$ to tour

return tour



Heuristics for TSP

Example: Nearest-Neighbor Heuristic

Algorithm Nearest-Neighbor Heuristic$(G = (V, E), c)$

Initialize tour as empty

Select arbitrary $s \in V$ (as starting vertex)

Set $v = s$ and $U = V - s$

while $U \neq \emptyset$ **do**

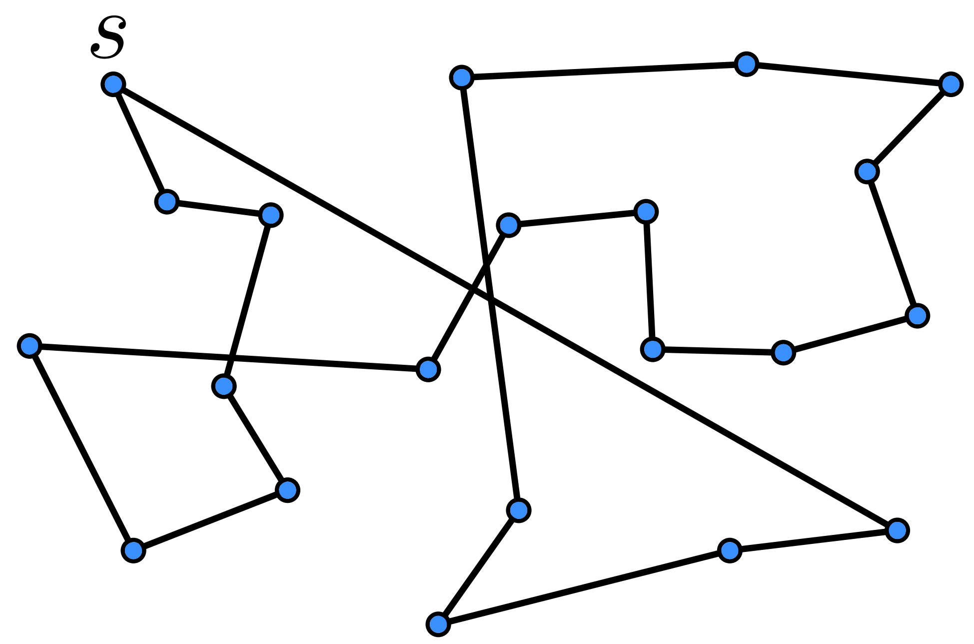
  Select $u \in U$ with $c(v, u) = \min_{w \in U} c(v, w)$

  Add edge {v, u} to tour

  Set $v = u$ and $U = U - u$

Add edge $\{v, s\}$ to tour

return tour

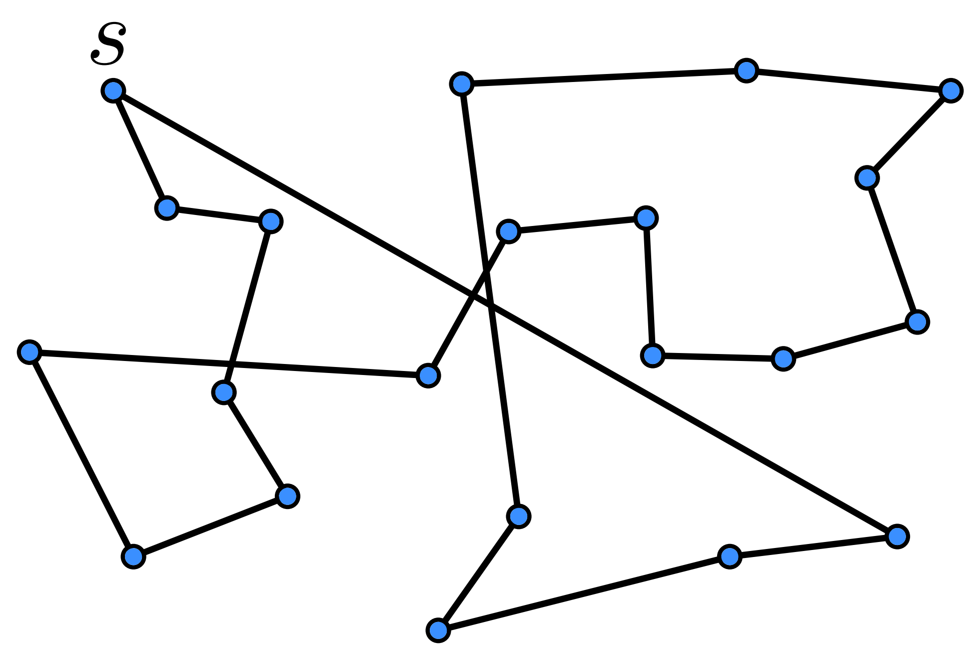


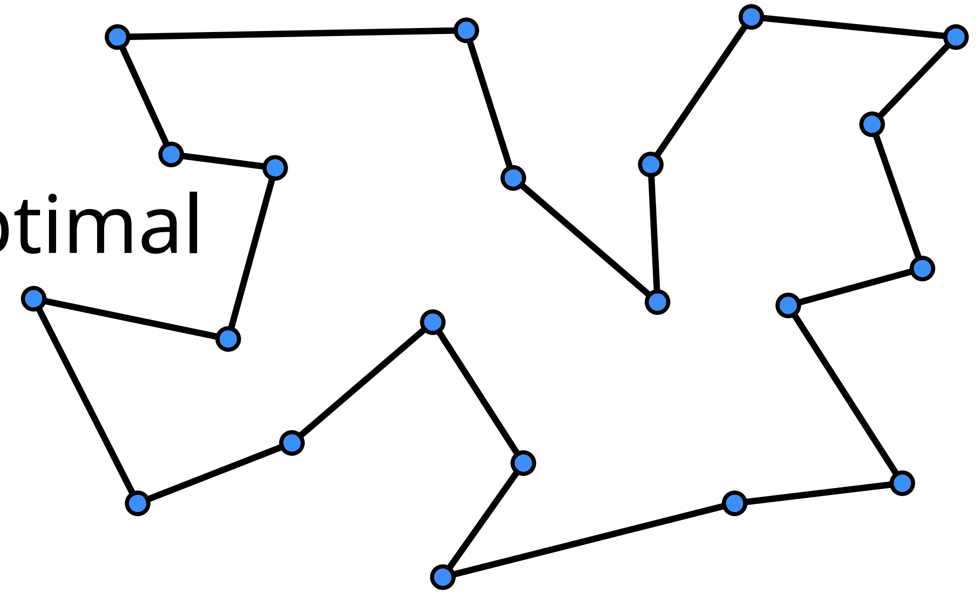
Heuristics for TSP

Example: Nearest-Neighbor Heuristic

Algorithm Nearest-Neighbor Heuristic$(G = (V, E), c)$

Initialize tour as empty
Select arbitrary $s \in V$ (as starting vertex)
Set $v = s$ and $U = V - s$
while $U \neq \emptyset$ **do**
  Select $u \in U$ with $c(v, u) = \min_{w \in U} c(v, w)$
  Add edge {v, u} to tour
  Set $v = u$ and $U = U - u$
Add edge $\{v, s\}$ to tour
return tour



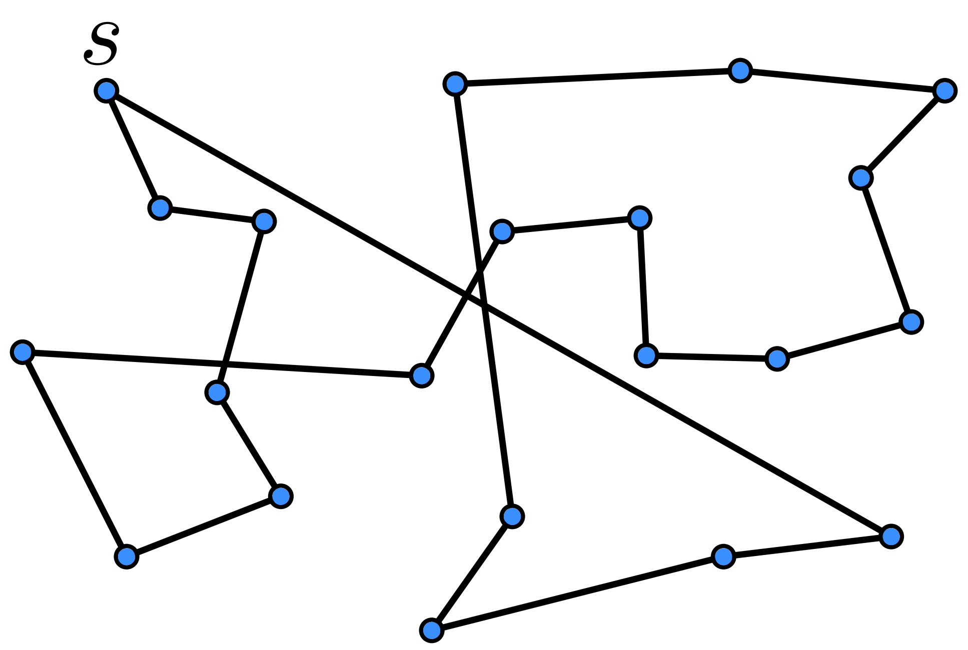


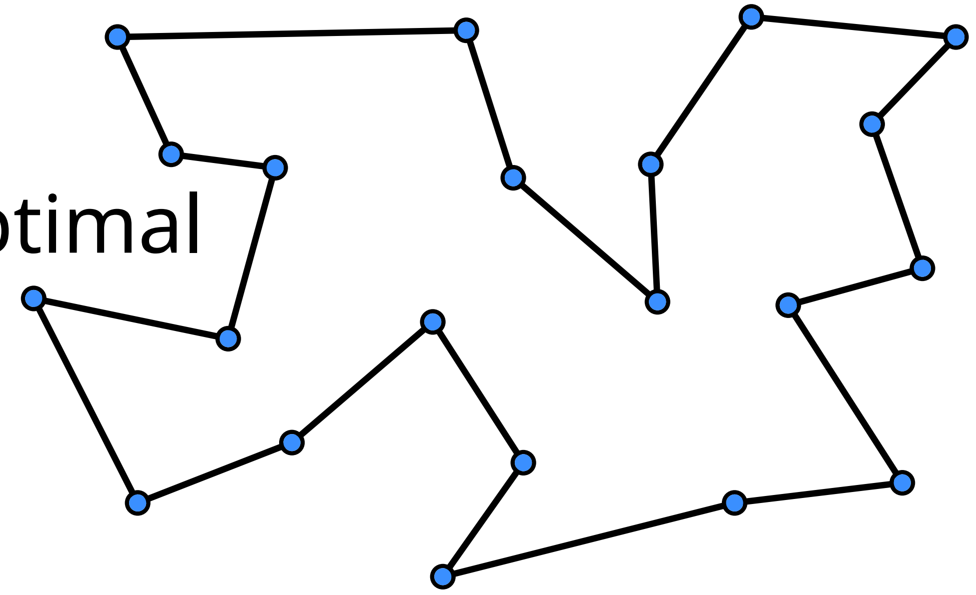
Heuristics for TSP

Example: Nearest-Neighbor Heuristic

Algorithm Nearest-Neighbor Heuristic$(G = (V, E), c)$

- Initialize tour as empty
- Select arbitrary $s \in V$ (as starting vertex)
- Set $v = s$ and $U = V - s$
- **while** $U \neq \emptyset$ **do**
  - Select $u \in U$ with $c(v, u) = \min_{w \in U} c(v, w)$
  - Add edge {v, u} to tour
  - Set $v = u$ and $U = U - u$
- Add edge $\{v, s\}$ to tour
- **return** tour



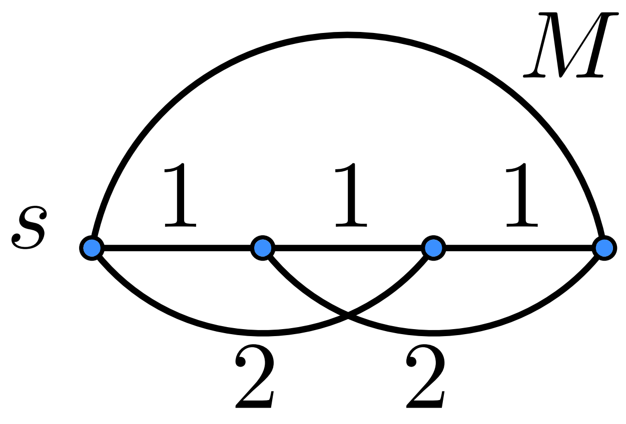


optimal

How bad can the Nearest-Neighbor Heuristic get?

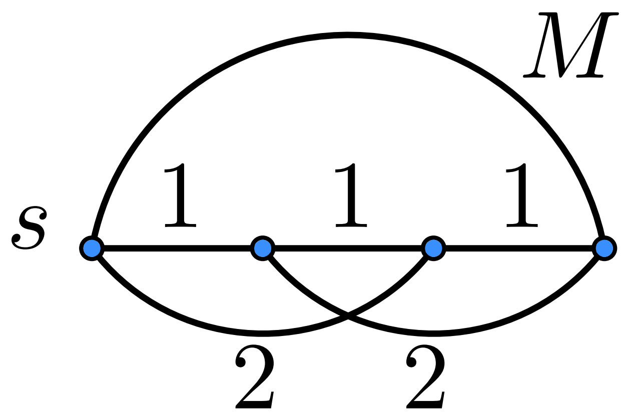
Nearest-Neighbor Heuristic – Lower Bound

Consider the following graph with large M



Nearest-Neighbor Heuristic – Lower Bound

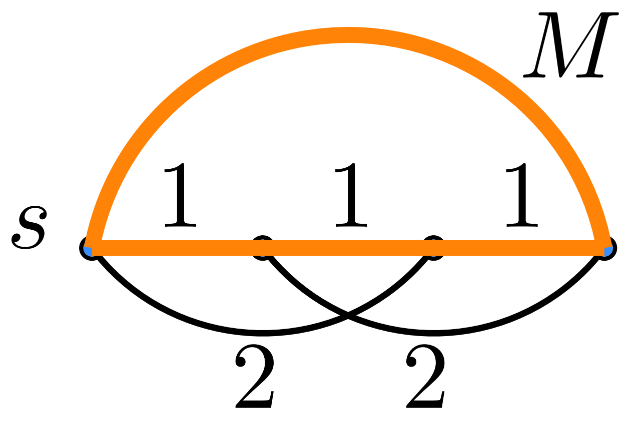
Consider the following graph with large M



How long is the tour of the Nearest-Neighbor Heuristic?

Nearest-Neighbor Heuristic – Lower Bound

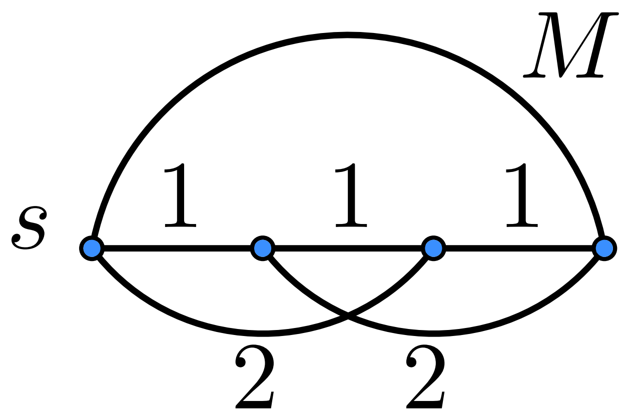
Consider the following graph with large M



How long is the tour of the Nearest-Neighbor Heuristic? $3 + M$

Nearest-Neighbor Heuristic – Lower Bound

Consider the following graph with large M

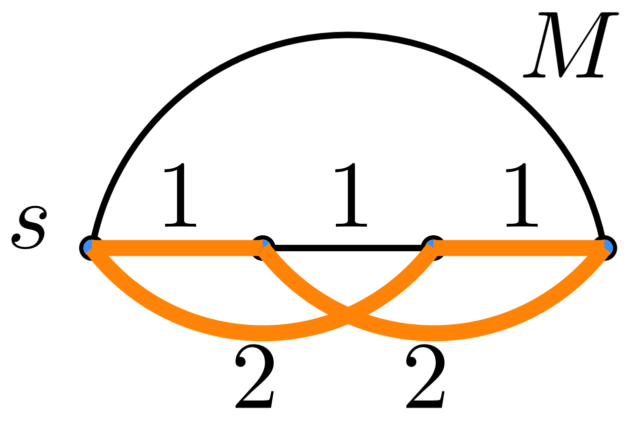


How long is the tour of the Nearest-Neighbor Heuristic? $3 + M$

How long is the optimal tour?

Nearest-Neighbor Heuristic – Lower Bound

Consider the following graph with large M

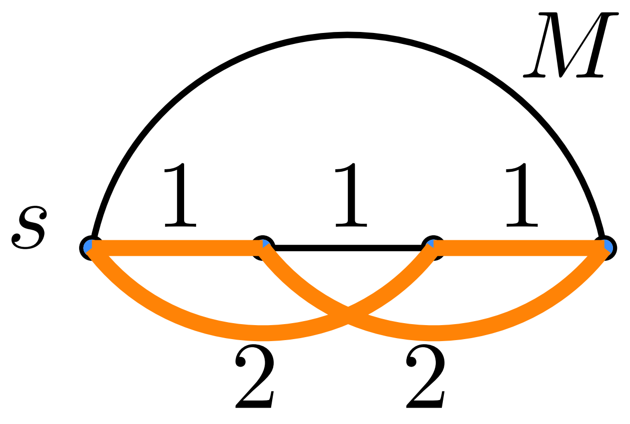


How long is the tour of the Nearest-Neighbor Heuristic? $3 + M$

How long is the optimal tour? 6

Nearest-Neighbor Heuristic – Lower Bound

Consider the following graph with large M



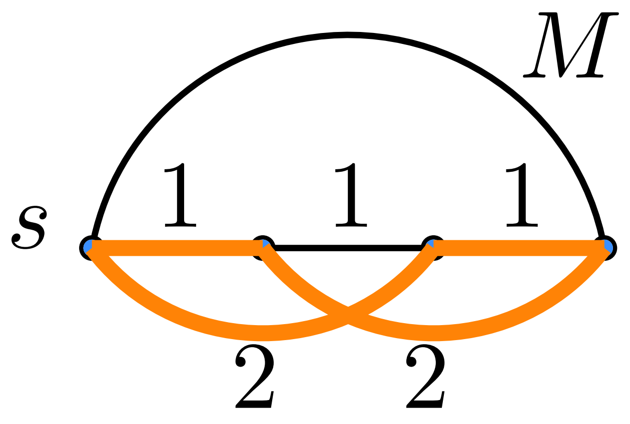
How long is the tour of the Nearest-Neighbor Heuristic? $3 + M$

How long is the optimal tour? 6

- can be arbitrary bad by making M arbitrary large

Nearest-Neighbor Heuristic – Lower Bound

Consider the following graph with large M



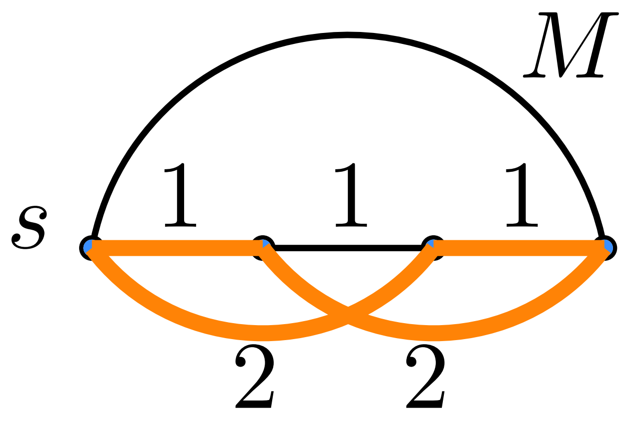
How long is the tour of the Nearest-Neighbor Heuristic? $3 + M$

How long is the optimal tour? 6

- can be arbitrary bad by making M arbitrary large
- but for large n, any polynomial-time algorithm is arbitrarily bad if $P \neq NP$

Nearest-Neighbor Heuristic – Lower Bound

Consider the following graph with large M



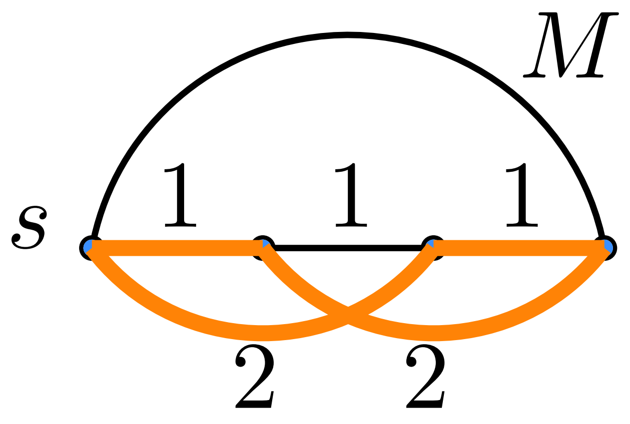
How long is the tour of the Nearest-Neighbor Heuristic? $3 + M$

How long is the optimal tour? 6

- can be arbitrary bad by making M arbitrary large
- but for large n, any polynomial-time algorithm is arbitrarily bad if $P \neq NP$
- for metric TSP (triangle inequality): at most a factor $\Theta(\log n)$ from optimal

Nearest-Neighbor Heuristic – Lower Bound

Consider the following graph with large M



How long is the tour of the Nearest-Neighbor Heuristic? $3 + M$

How long is the optimal tour? 6

- can be arbitrary bad by making M arbitrary large
- but for large n, any polynomial-time algorithm is arbitrarily bad if $P \neq NP$
- for metric TSP (triangle inequality): at most a factor $\Theta(\log n)$ from optimal
- fast but other heuristics give better results

Dynamic Program for TSP

TSP can easily be solved in exponential time by enumerating all solutions, but that is too expensive already for small instances.

Can we do better using dynamic programming?

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

How can the problem be solved recursively?

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

How can the problem be solved recursively?

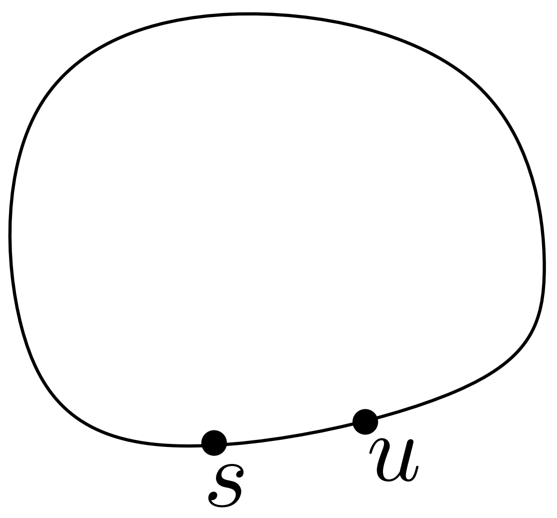
What are the optimal substructures?

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

How can the problem be solved recursively?
What are the optimal substructures?

Choose a starting vertex $s \in V$ and decompose the tour at s:



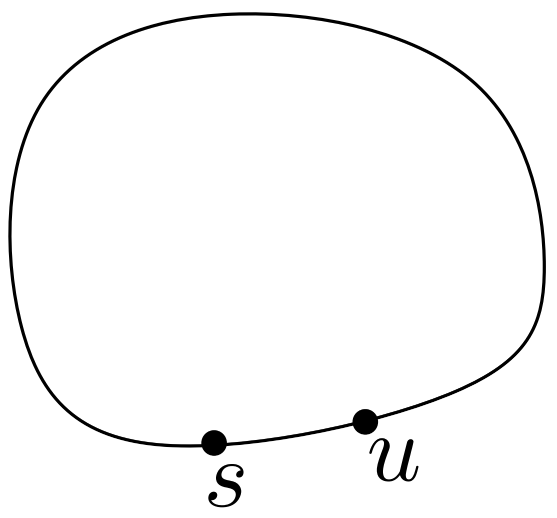
- path from s through all nodes in $V - s$ ending in some $u \in V - s$
- edge from u to s

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

How can the problem be solved recursively?
What are the optimal substructures?

Choose a starting vertex $s \in V$ and decompose the tour at s:



- path from s through all nodes in $V - s$ ending in some $u \in V - s$
- edge from u to s

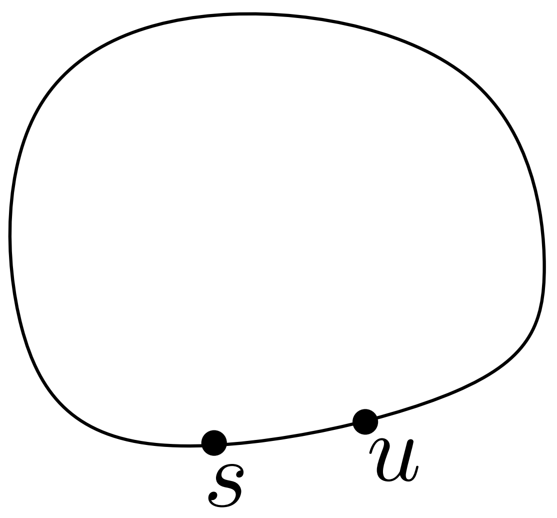
short for $V \setminus \{s\}$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

How can the problem be solved recursively?
What are the optimal substructures?

Choose a starting vertex $s \in V$ and decompose the tour at s:



- path from s through all nodes in $V - s$ ending in some $u \in V - s$
- edge from u to s

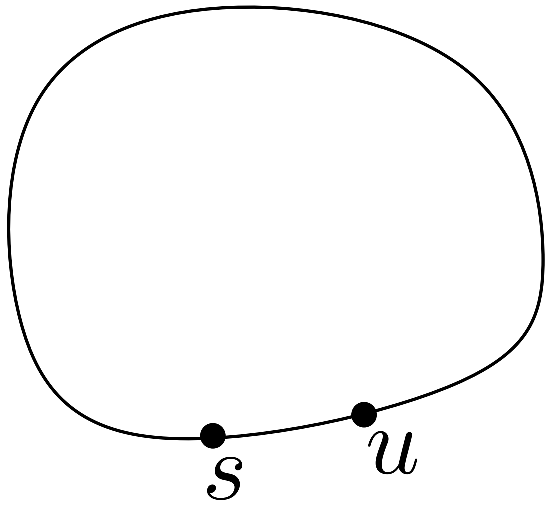
minimize the sum of both lengths!

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

How can the problem be solved recursively?
What are the optimal substructures?

Choose a starting vertex $s \in V$ and decompose the tour at s:



- path from s through all nodes in $V - s$ ending in some $u \in V - s$ — compute recursively
- edge from u to s — fixed

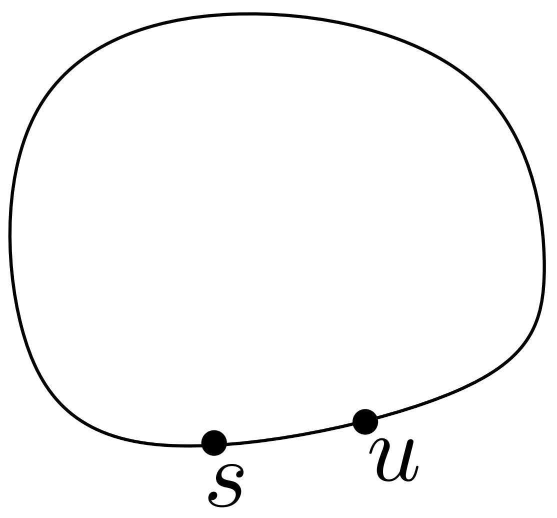
minimize the sum of both lengths!

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

How can the problem be solved recursively?
What are the optimal substructures?

Choose a starting vertex $s \in V$ and decompose the tour at s:



- path from s through all nodes in $V - s$ ending in some $u \in V - s$
- edge from u to s

compute recursively

fixed

minimize the sum of both lengths!

Parameter

Dynamic Program for TSP

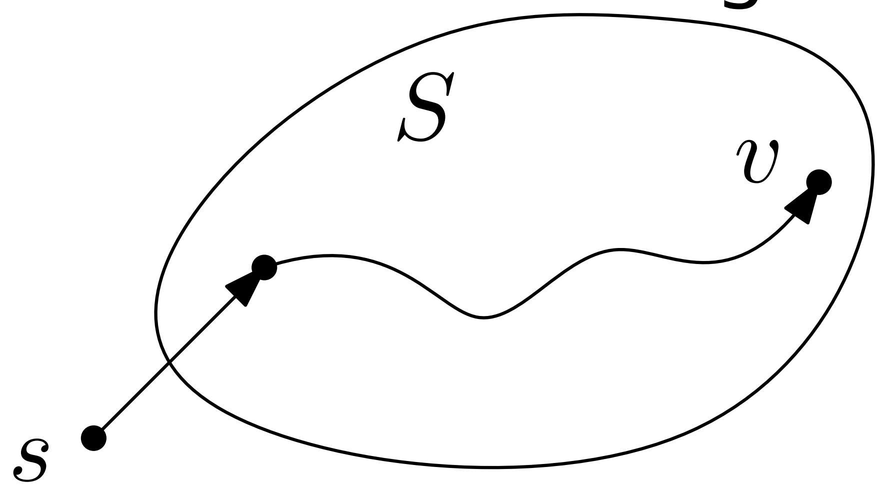
Algorithm of Bellmann, Held and Karp

How can the problem be solved recursively?
What are the optimal substructures?

Choose a starting vertex $s \in V$ and decompose the tour at s:

For $S \subseteq V - s$ and $v \in S$ let:

$\text{OPT}[S, v] =$ length of shortest s-v-path,
visiting all vertices in $S \cup \{s\}$.



Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Start of recursion: $S = \{v\}$ is simply:

$$\text{OPT}[\{v\}, v] = c(s, v).$$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Start of recursion: $S = \{v\}$ is simply:

$$\text{OPT}[\{v\}, v] = c(s, v).$$

For $|S| \geq 2$, compute $\text{OPT}[S, v]$ recursively:

Dynamic Program for TSP

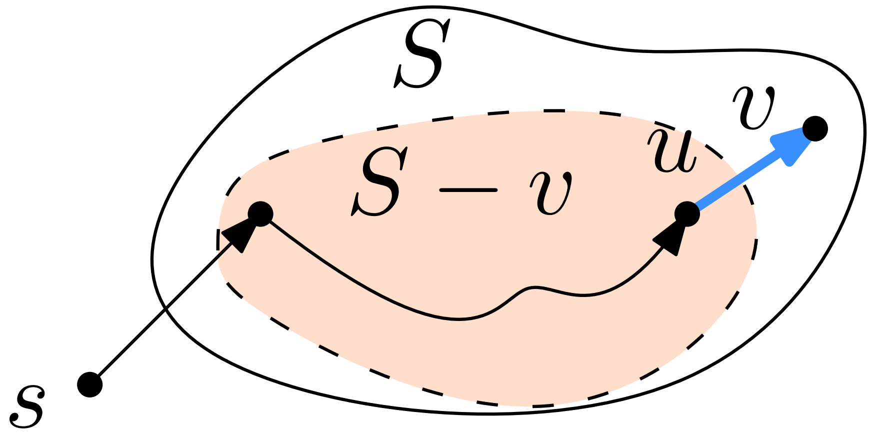
Algorithm of Bellmann, Held and Karp

Start of recursion: $S = \{v\}$ is simply:

$$\text{OPT}[\{v\}, v] = c(s, v).$$

For $|S| \geq 2$, compute $\text{OPT}[S, v]$ recursively:

$$\text{OPT}[S, v] = \min\{\ \text{OPT}[S - v, u] + c(u, v) \mid u \in S - v\ \}$$



Dynamic Program for TSP

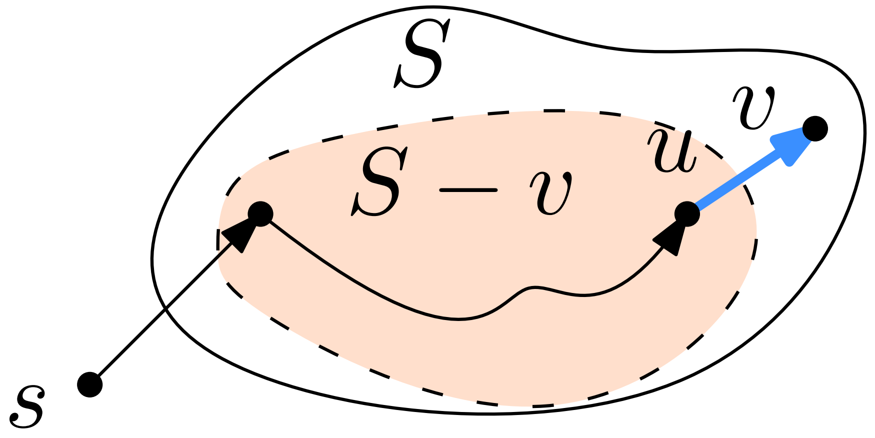
Algorithm of Bellmann, Held and Karp

Start of recursion: $S = \{v\}$ is simply:

$$\text{OPT}[\{v\}, v] = c(s, v).$$

For $|S| \geq 2$, compute $\text{OPT}[S, v]$ recursively:

$$\text{OPT}[S, v] = \min\{ \text{OPT}[S - v, u] + c(u, v) \mid u \in S - v \}$$



The optimal TSP-tour can then be easily obtained as

Dynamic Program for TSP

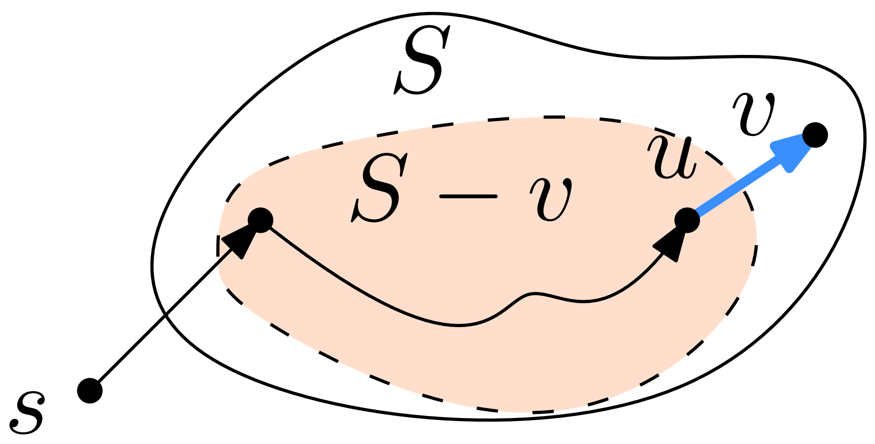
Algorithm of Bellmann, Held and Karp

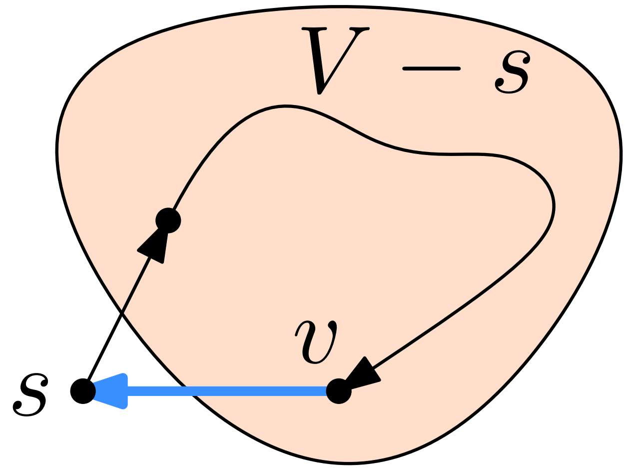
Start of recursion: $S = \{v\}$ is simply:

$$\text{OPT}[\{v\}, v] = c(s, v).$$

For $|S| \geq 2$, compute OPT$[S, v]$ recursively:

$$\text{OPT}[S, v] = \min\{\ \text{OPT}[S - v, u] + c(u, v)\ \mid u \in S - v\,\}$$





The optimal TSP-tour can then be easily obtained as

$$\text{OPT} = \min\{\ \text{OPT}[V - s, v] + c(v, s)\ \mid v \in V - s\,\}$$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Algorithm Bellmann-Held-Karp(G, c)

foreach $v \in V - s$ **do**
- OPT$[\{v\}, v] = c(s, v)$

for $j = 2$ **to** $n - 1$ **do**
- **foreach** $S \subseteq V - s$ with $|S| = j$ **do**
  - **foreach** $v \in S$ **do**
    - OPT$[S, v] = \min\{$ OPT$[S - v, u] + c(u, v) \mid u \in S - v \}$

return $\min\{$ OPT$[V - s, v] + c(v, s) \mid v \in V - s \}$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

```
Algorithm Bellmann-Held-Karp(G, c)

  foreach v ∈ V − s do
    OPT[{v}, v] = c(s, v)

  for j = 2 to n − 1 do
    foreach S ⊆ V − s with |S| = j do
      foreach v ∈ S do
        OPT[S, v] = min{ OPT[S − v, u] + c(u, v) | u ∈ S − v }

  return min{ OPT[V − s, v] + c(v, s) | v ∈ V − s }
```

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

```
Algorithm Bellmann-Held-Karp(G, c)

  foreach v ∈ V − s do
    OPT[{v}, v] = c(s, v)

  for j = 2 to n − 1 do
    foreach S ⊆ V − s with |S| = j do
      foreach v ∈ S do
        OPT[S, v] = min{ OPT[S − v, u] + c(u, v) | u ∈ S − v }

  return min{ OPT[V − s, v] + c(v, s) | v ∈ V − s }
```

Runtime: ???

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Algorithm Bellmann-Held-Karp(G, c)

foreach $v \in V - s$ **do** $\quad O(n)$
  OPT$[\{v\}, v] = c(s, v)$

for $j = 2$ **to** $n - 1$ **do** $\quad \big\} \; O(2^n)$
  foreach $S \subseteq V - s$ with $|S| = j$ **do**
    foreach $v \in S$ **do** $\quad \} \; O(j)$
      OPT$[S, v] = \min\{$ OPT$[S - v, u] + c(u, v) \mid u \in S - v \}$ $\quad O(j)$

return $\min\{$ OPT$[V - s, v] + c(v, s) \mid v \in V - s \}$ $\quad O(n)$

Runtime: $O(2^n \cdot n^2)$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Algorithm Bellmann-Held-Karp(G, c)

foreach $v \in V - s$ **do** $\quad O(n)$
  OPT$[\{v\}, v] = c(s, v)$

for $j = 2$ **to** $n - 1$ **do** $\quad \Big\} \; O(2^n)$
  foreach $S \subseteq V - s$ with $|S| = j$ **do**
    foreach $v \in S$ **do** $\quad \} \; O(j)$
      OPT$[S, v] = \min\{$ OPT$[S - v, u] + c(u, v) \mid u \in S - v \}$ $\quad O(j)$

return $\min\{$ OPT$[V - s, v] + c(v, s) \mid v \in V - s \}$ $\quad O(n)$

Runtime: $O(2^n \cdot n^2)$
Space: $\Theta(2^n \cdot n)$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Algorithm Bellmann-Held-Karp(G, c)

foreach $v \in V - s$ **do** $\qquad O(n)$
  OPT$[\{v\}, v] = c(s, v)$

for $j = 2$ **to** $n - 1$ **do** $\qquad \Big\} \; O(2^n)$
  foreach $S \subseteq V - s$ with $|S| = j$ **do**
    foreach $v \in S$ **do** $\qquad \} \; O(j)$
      OPT$[S, v] = \min\{$ OPT$[S - v, u] + c(u, v) \mid u \in S - v \}$ $\qquad O(j)$

return $\min\{$ OPT$[V - s, v] + c(v, s) \mid v \in V - s \}$ $\quad O(n)$

Runtime: $O(2^n \cdot n^2)$
Space: $\Theta(2^n \cdot n)$

do we need this much space, even though for value j we only need the values for $j - 1$?

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Algorithm Bellmann-Held-Karp(G, c)

foreach $v \in V - s$ **do** $\quad O(n)$
  OPT$[\{v\}, v] = c(s, v)$

for $j = 2$ **to** $n - 1$ **do**
  foreach $S \subseteq V - s$ with $|S| = j$ **do** $\quad \Big\} \; O(2^n)$
    foreach $v \in S$ **do** $\quad \} \; O(j)$
      OPT$[S, v] = \min\{\,$OPT$[S - v, u] + c(u, v) \mid u \in S - v\,\}$ $\quad O(j)$

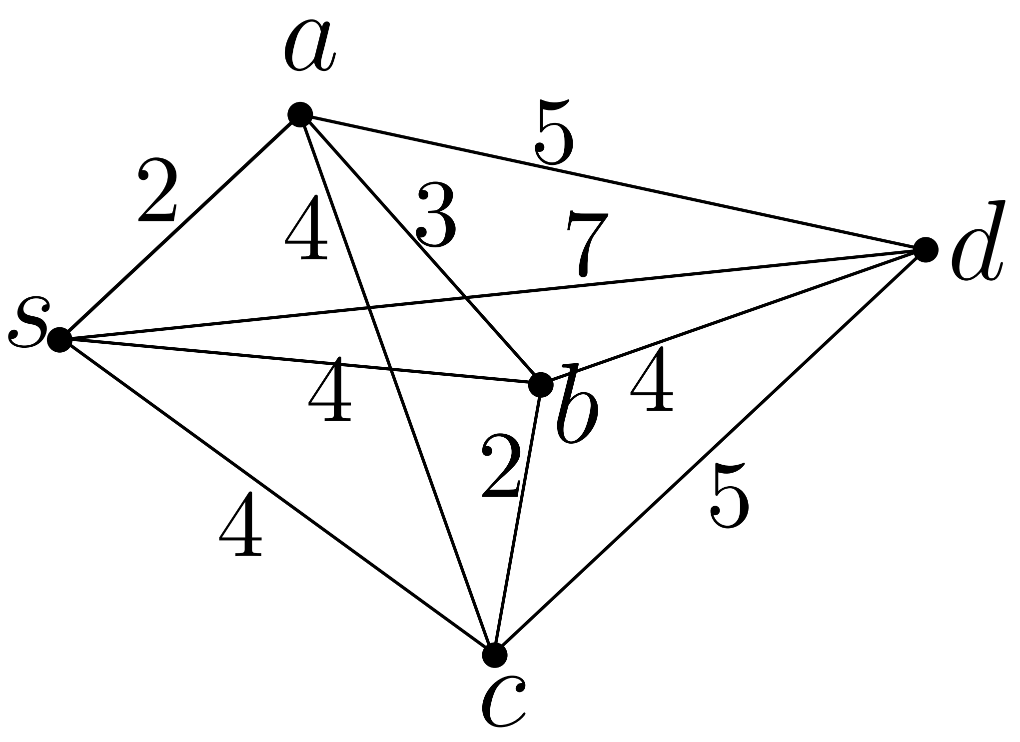
return $\min\{\,$OPT$[V - s, v] + c(v, s) \mid v \in V - s\,\}$ $\quad O(n)$

Runtime: $O(2^n \cdot n^2)$
Space: $\Theta(2^n \cdot n)$
A shortest tour can be found through backtracking in the table.

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Example

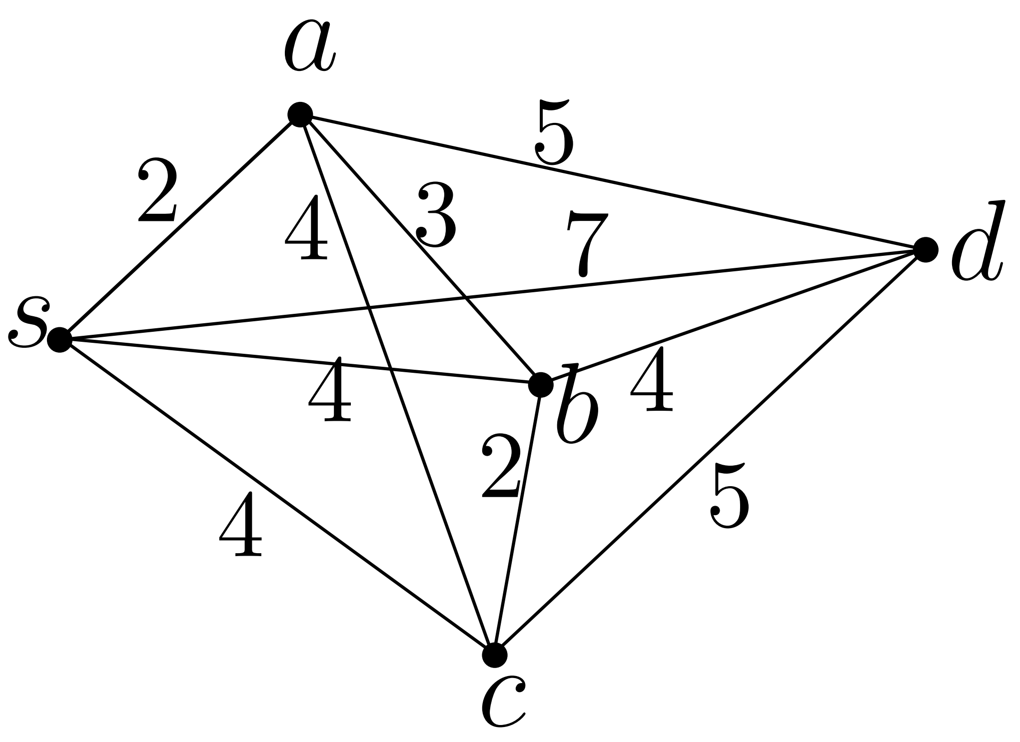


Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Example

$j \backslash v$	$S - v \downarrow$ a	b	c	d	S
1					
2					
3					
4					
$\sum$					



OPT$[S, v]$

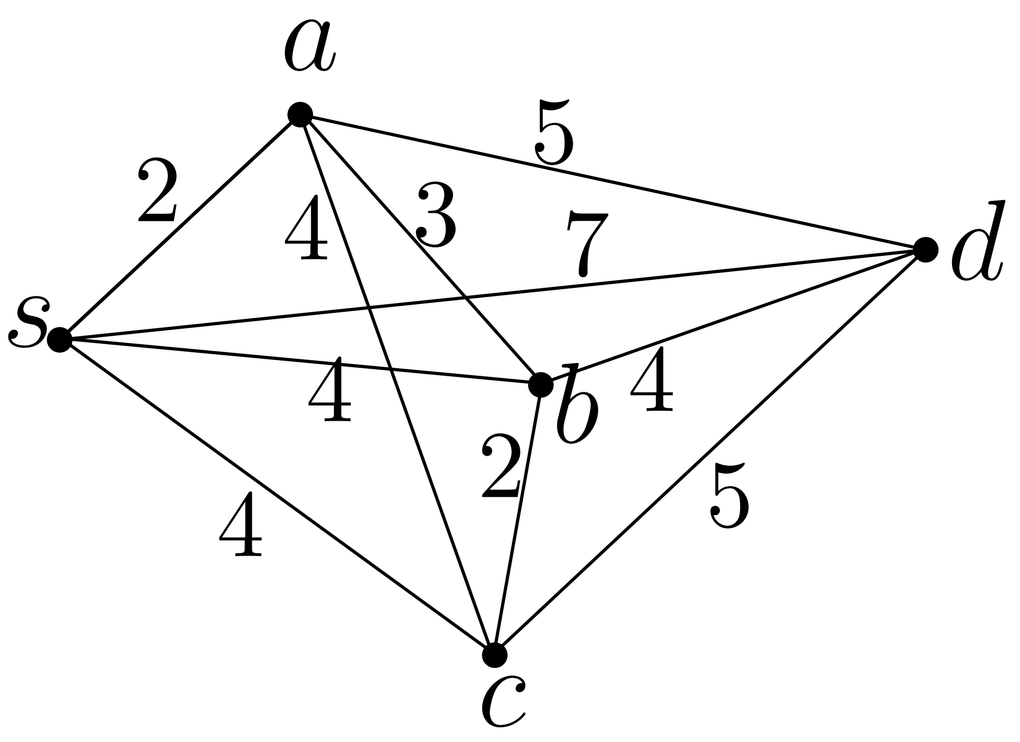
OPT$[V - s, v] + c(v, s)$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Example

v j	$S - v$ ↓ a	b	c	d	S
1	– 2	– 4	– 4	– 7	$\{a\}, \{b\},$ $\{c\}, \{d\}$
2					
3					
4					
$\sum$					



$\mathsf{OPT}[S, v]$

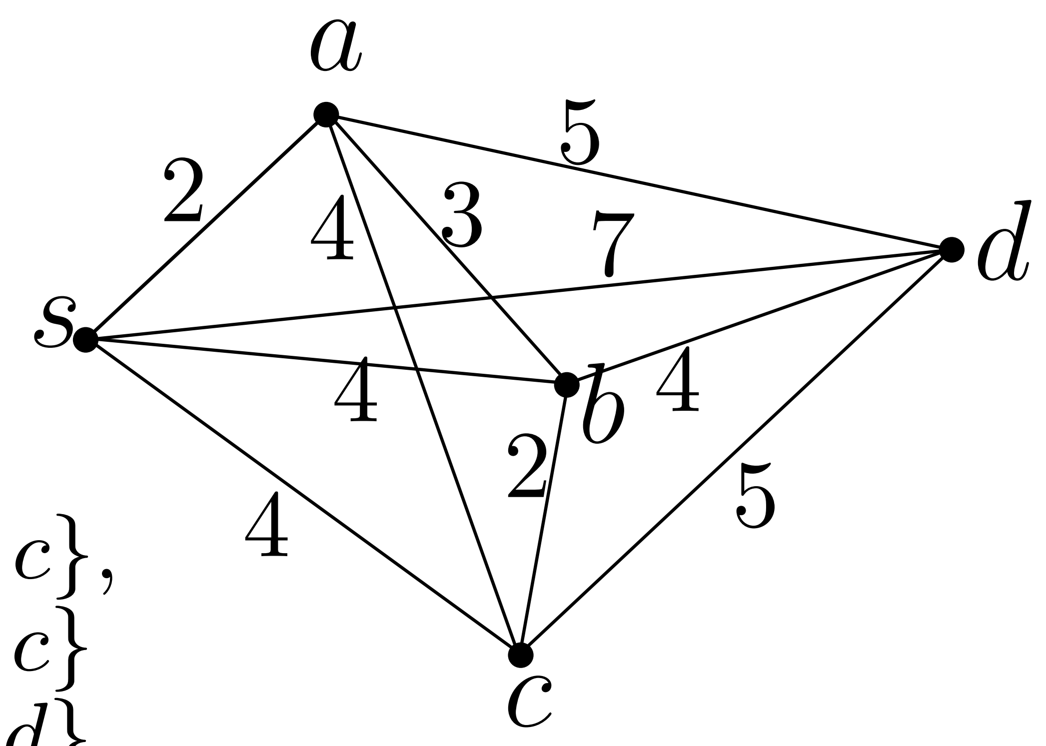
$\mathsf{OPT}[V - s, v] + c(v, s)$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Example

$j \backslash v$	$S-v$ ↓	a		b		c		d	S
1	–	2	–	4	–	4	–	7	$\{a\}, \{b\}, \{c\}, \{d\}$
2	b c d	7 8 12	a c d	5 6 11	a b d	6 6 12	a b c	7 8 9	$\{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}$
3									
4									
$\sum$									



OPT$[S, v]$

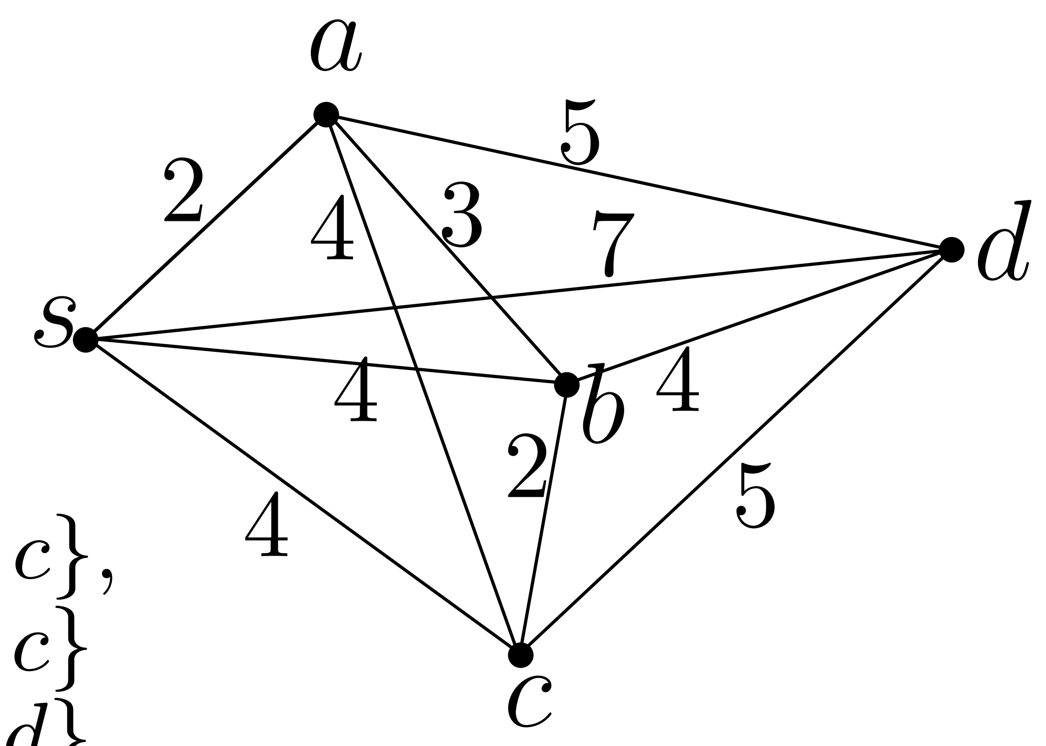
OPT$[V - s, v] + c(v, s)$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Example

$j \setminus v$	$S - v$ ↓	a		b		c		d	S
1	–	2	–	4	–	4	–	7	$\{a\}, \{b\}, \{c\}, \{d\}$
2	b c d	7 8 12	a c d	5 6 11	a b d	6 6 12	a b c	7 8 9	$\{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}$
3	b,c b,d c,d	9 13 14	a,c a,d c,d	8 11 13	a,b a,d b,d	7 12 13	a,b a,c b,c	9 11 10	$\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}$
4									
$\sum$									



OPT$[S, v]$

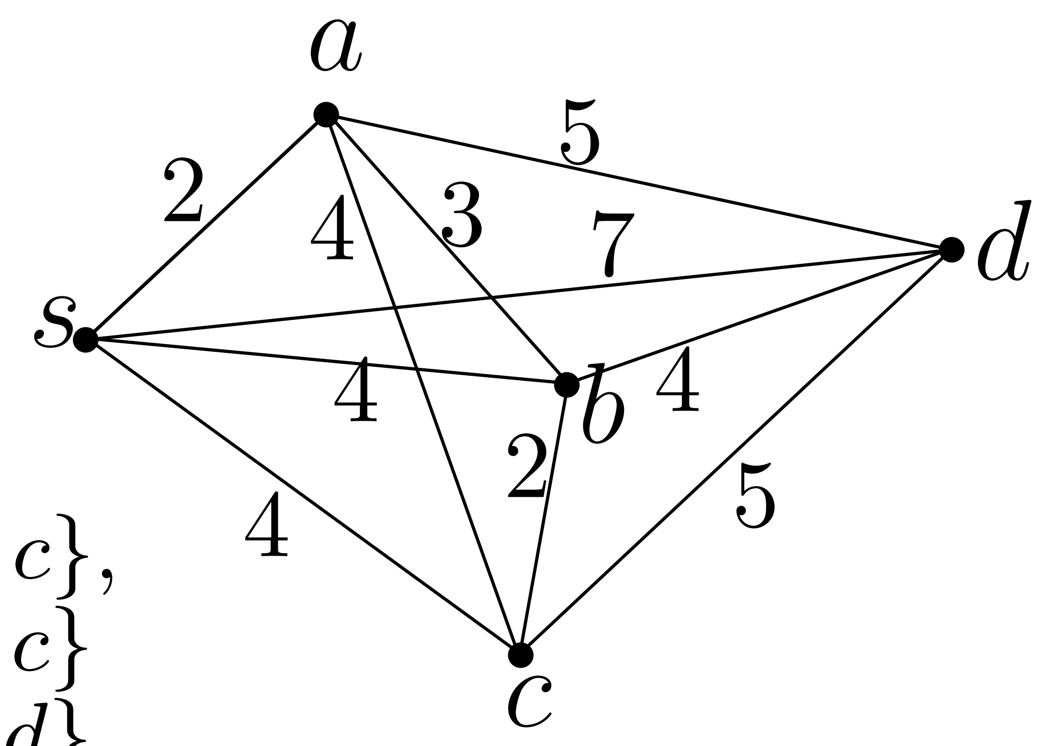
OPT$[V - s, v] + c(v, s)$

Dynamic Program for TSP

Algorithm of Bellmann, Held and Karp

Example

j \ v	$S-v$ ↓	a		b		c		d	S
1	–	2	–	4	–	4	–	7	$\{a\}, \{b\}, \{c\}, \{d\}$
2	b c d	7 8 12	a c d	5 6 11	a b d	6 6 12	a b c	7 8 9	$\{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}$
3	b,c b,d c,d	9 13 14	a,c a,d c,d	8 11 13	a,b a,d b,d	7 12 13	a,b a,c b,c	9 11 10	$\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}$
4	b,c,d	15	a,c,d	14	a,b,d	13	a,b,c	12	$\{a,b,c,d\}$
$\sum$									



$\mathrm{OPT}[S, v]$

$\mathrm{OPT}[V - s, v] + c(v, s)$

Dynamic Program for TSP

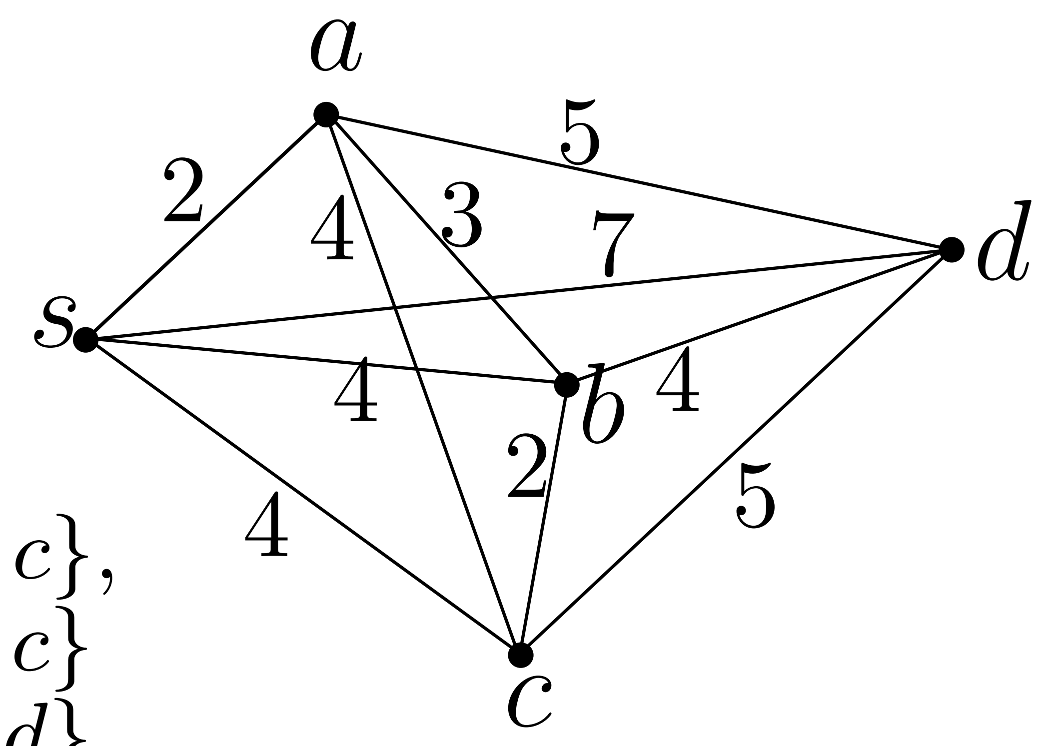
Algorithm of Bellmann, Held and Karp

Example

$j \setminus v$	$S-v \downarrow$	a		b		c		d	S
1	–	2	–	4	–	4	–	7	$\{a\}, \{b\}, \{c\}, \{d\}$
2	b c d	7 8 12	a c d	5 6 11	a b d	6 6 12	a b c	7 8 9	$\{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}$
3	b,c b,d c,d	9 13 14	a,c a,d c,d	8 11 13	a,b a,d b,d	7 12 13	a,b a,c b,c	9 11 10	$\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}$
4	b,c,d	15	a,c,d	14	a,b,d	13	a,b,c	12	$\{a,b,c,d\}$
$\sum$		17		18		17		19	

OPT$[S, v]$

OPT$[V-s, v] + c(v, s)$



Dynamic Program for TSP

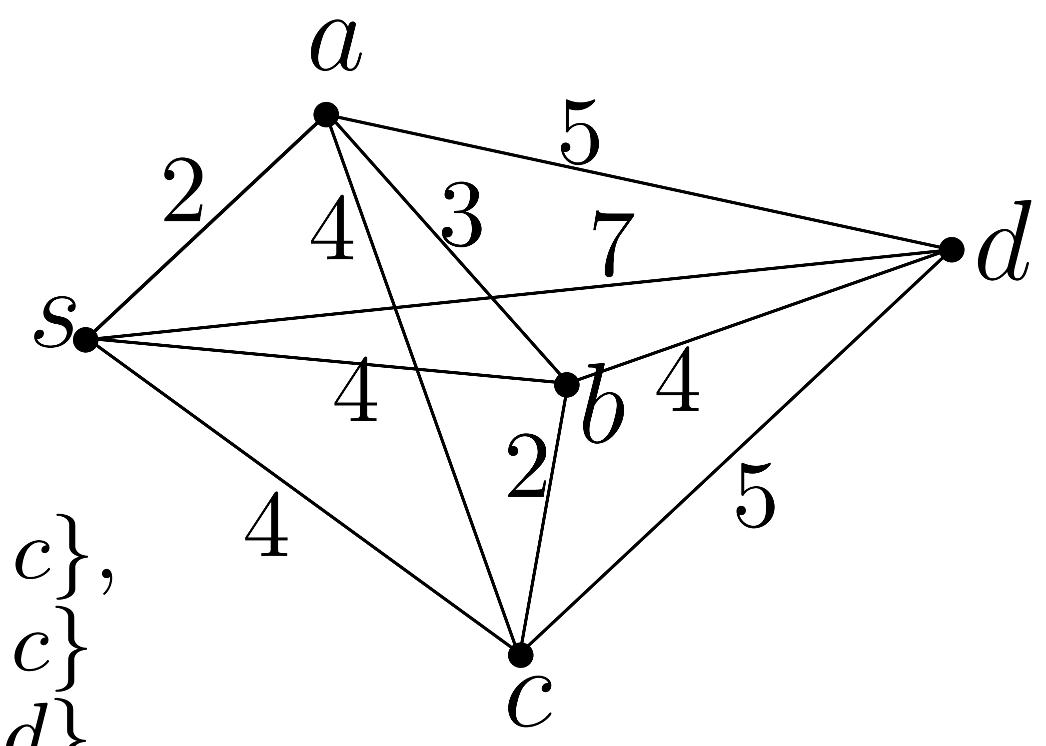
Algorithm of Bellmann, Held and Karp

Example

$j \setminus v$	$S-v$ ↓	a		b		c		d	S
1	–	2	–	4	–	**4**	–	7	$\{a\}, \{b\},$ $\{c\}, \{d\}$
2	b c d	7 8 12	a c d	5 **6** 11	a b d	6 6 12	a b c	7 8 9	$\{a,b\}, \{a,c\},$ $\{a,d\}, \{b,c\}$ $\{b,d\}, \{c,d\}$
3	b,c b,d c,d	9 13 14	a,c a,d c,d	8 11 13	a,b a,d b,d	7 12 13	a,b a,c b,c	9 11 **10**	$\{a,b,c\},$ $\{a,b,d\},$ $\{a,c,d\},$ $\{b,c,d\}$
4	b,c,d	**15**	a,c,d	14	a,b,d	13	a,b,c	12	$\{a,b,c,d\}$
$\sum$		**17**		18		17		19	

OPT$[S, v]$

OPT$[V - s, v] + c(v, s)$



Dynamic Program for TSP

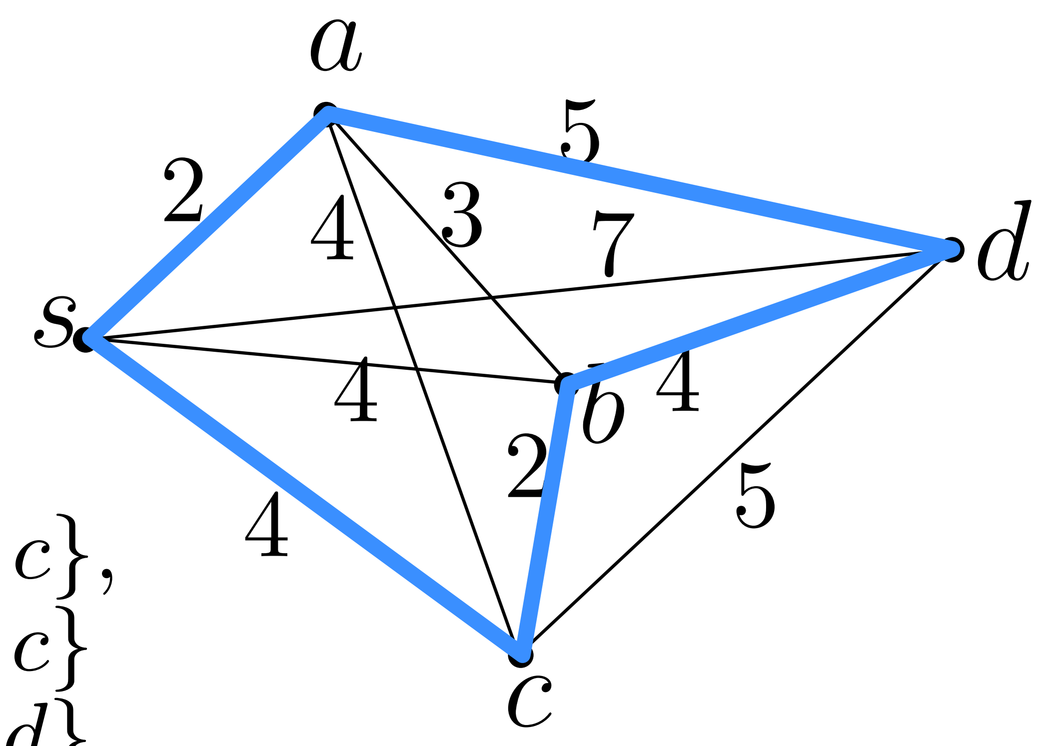
Algorithm of Bellmann, Held and Karp

Example

$j \backslash v$	$S - v$ ↓	a		b		c		d	S
1	–	2	–	4	–	$\mathbf{4}$	–	7	$\{a\}, \{b\}, \{c\}, \{d\}$
2	b c d	7 8 12	a c d	5 $\mathbf{6}$ 11	a b d	6 6 12	a b c	7 8 9	$\{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}$
3	b,c b,d c,d	9 13 14	a,c a,d c,d	8 11 13	a,b a,d b,d	7 12 13	a,b a,c b,c	9 11 $\mathbf{10}$	$\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}$
4	b,c,d	$\mathbf{15}$	a,c,d	14	a,b,d	13	a,b,c	12	$\{a,b,c,d\}$
$\sum$		$\mathbf{17}$		18		17		19	

OPT$[S, v]$

OPT$[V - s, v] + c(v, s)$



Summary

shortest path

- efficiently solvable
- heaps with $O(1)$ decreaseKey

shortest round trip

- NP-hard
- many algorithmic approaches
- dynamic program: exponential-time algorithm

Summary

shortest path

- efficiently solvable
- heaps with $O(1)$ decreaseKey

shortest round trip

- NP-hard
- many algorithmic approaches
- dynamic program: exponential-time algorithm

next lectures:

- amortized analysis
- binomial heaps
- fibonacci heaps

Summary

shortest path

- efficiently solvable
- heaps with $O(1)$ decreaseKey

next lectures:

- amortized analysis
- binomial heaps
- fibonacci heaps

shortest round trip

- NP-hard
- many algorithmic approaches
- dynamic program: exponential-time algorithm

later lectures:

fast algorithms for hard problems

- approximation
- parameterized