

Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving.
- Robot navigation.
- Texture mapping.
- Typesetting in LaTeX.
- Urban traffic planning.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Optimal truck routing through given traffic congestion pattern.



Network Flows: Theory, Algorithms, and Applications,
by Ahuja, Magnanti, and Orlin, Prentice Hall, 1993.

7

Edsger Dijkstra

“What’s the shortest way to travel from Rotterdam to Groningen?
It is the algorithm for the shortest path, which I designed in
about 20 minutes. One morning I was shopping in Amsterdam
with my young fiancée, and tired, we sat down on the café
terrace to drink a cup of coffee and I was just thinking about
whether I could do this, and I then designed the algorithm for
the shortest path.” — Edsger Dijkstra



8

Dijkstra’s algorithm: single-source shortest paths



Greedy approach. Maintain a set of explored vertices S for which the algorithm has determined $d[u]$ = the length of a shortest $s \rightsquigarrow u$ path.

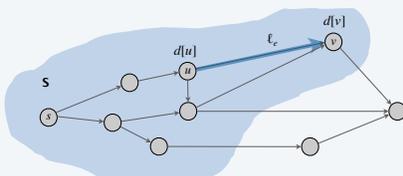
- Initialize $S \leftarrow \{s\}$, $d[s] \leftarrow 0$.
- Repeatedly choose an unexplored vertex $v \notin S$ which minimizes

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

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

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

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



10

Dijkstra's algorithm: proof of correctness

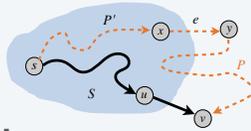
Invariant. For each vertex $u \in S$, $d[u] = \text{length of a shortest } s \rightsquigarrow u \text{ path}$.

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 the next vertex 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 :



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

↑ non-negative lengths
 ↑ inductive hypothesis
 ↑ definition of $\pi(y)$
 ↑ Dijkstra's alg. chose v instead of y

11

Dijkstra's algorithm: efficient implementation



Critical optimization 1. For each unexplored vertex $v \notin S$: explicitly maintain $\pi[v]$ instead of computing directly from definition

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

- For each $v \notin S$: $\pi(v)$ can only decrease (because set S increases).
- More specifically, suppose u is added to S and there is an edge $e = (u, v)$ leaving u . Then, it suffices to update:

$$\pi[v] \leftarrow \min \{ \pi[v], \pi[u] + \ell_e \}$$

↑ recall: for each $u \in S$,
 $\pi[u] = d[u] = \text{length of shortest } s \rightsquigarrow u \text{ path}$

Critical optimization 2. Use a min-oriented priority queue (PQ) to choose an unexplored vertex that minimizes $\pi[v]$.

12

Dijkstra's algorithm: efficient implementation

Implementation.

- Algorithm maintains $\pi[v]$ for each node v .
- Priority queue stores unexplored vertices, using $\pi[\cdot]$ as priorities.
- Once u is deleted from the PQ, $\pi[u] = \text{length of a shortest } s \rightsquigarrow u \text{ path}$.

```

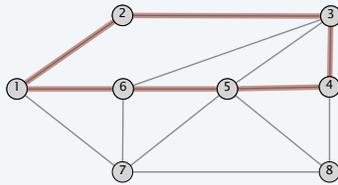
DUKSTRA (V, E, ℓ, s)
  FOREACH v ≠ s : π[v] ← ∞, pred[v] ← null; π[s] ← 0.
  Create an empty priority queue pq.
  FOREACH v ∈ V : INSERT(pq, v, π[v]).
  WHILE (IS-NOT-EMPTY(pq))
    u ← DEL-MIN(pq).
    FOREACH edge e = (u, v) ∈ E leaving u:
      IF (π[v] > π[u] + ℓ_e)
        DECREASE-KEY(pq, v, π[u] + ℓ_e).
        π[v] ← π[u] + ℓ_e; pred[v] ← e.
    
```

13

Paths and cycles

Def. A **path** is a sequence of edges which connects a sequence of vertices.

Def. A **cycle** is a path with no repeated vertices or edges other than the starting and ending vertices.



path $P = \{ (1, 2), (2, 3), (3, 4), (4, 5), (5, 6) \}$

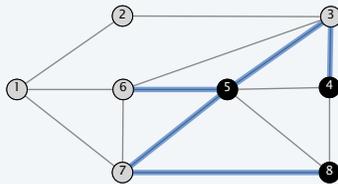
cycle $C = \{ (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) \}$

19

Cuts

Def. A **cut** is a partition of vertices into two nonempty subsets S and $V - S$.

Def. The **cutset** of a cut S is the set of edges with exactly one endpoint in S .



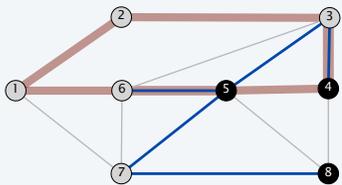
cut $S = \{ 4, 5, 8 \}$

cutset $D = \{ (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) \}$

20

Cycle-cut intersection

Proposition. A cycle and a cutset intersect in an **even** number of edges.



cycle $C = \{ (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) \}$

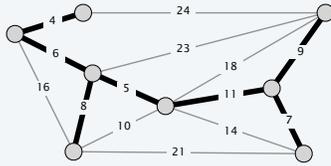
cutset $D = \{ (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) \}$

intersection $C \cap D = \{ (3, 4), (5, 6) \}$

23

Minimum spanning trees (MSTs)

Def. Given a connected, undirected graph $G = (V, E)$ with edge costs c_e , a **minimum spanning tree** (V, T) is a spanning tree of G such that the sum of the edge costs in T is minimized.



$$\text{MST cost} = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7$$

Cayley's theorem. The complete graph on n nodes has n^{n-2} spanning trees.

↑
can't solve by brute force

29

MST applications

MST is a fundamental problem with diverse applications.

- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- Model locality of particle interactions in turbulent fluid flows.
- Reducing data storage in sequencing amino acids in a protein.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).
- Network design (communication, electrical, hydraulic, computer, road).



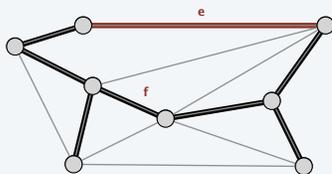
Network Flows: Theory, Algorithms, and Applications,
by Ahuja, Magnanti, and Orlin, Prentice Hall, 1993.

31

Fundamental cycles

Fundamental cycle. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any non-tree edge $e \in E$, $T \cup \{e\}$ contains a unique cycle, say C .
- For any edge $f \in C$, $(V, T \cup \{e\} - \{f\})$ is a spanning tree.



graph $G = (V, E)$
spanning tree $H = (V, T)$

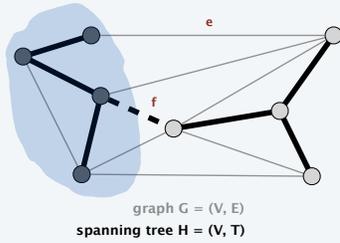
Observation. If $c_e < c_f$, then (V, T) is not an MST.

32

Fundamental cutsets

Fundamental cutset. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any tree edge $f \in T$, $(V, T - \{f\})$ has two connected components.
- Let D denote the corresponding cutset.
- For any edge $e \in D$, $(V, T - \{f\} \cup \{e\})$ is a spanning tree.



Observation. If $c_e < c_f$, then (V, T) is not an MST.

33

Greedy algorithm: MSTs



Red rule.

- Let C be a cycle with no red edges.
- Select an uncoloured edge of C of max cost and colour it red.

Blue rule.

- Let D be a cutset with no blue edges.
- Select an uncoloured edge in D of min cost and colour it blue.

Greedy algorithm.

- Apply the red and blue rules (nondeterministically!) until all edges are coloured. The blue edges form an MST.
- Note: we can stop once we have $n - 1$ edges coloured blue.

34

Greedy algorithm: proof of correctness

Colour invariant. There exists an MST (V, T^*) containing every blue edge and no red edge.

Pf. [by induction on number of iterations]

Base case. No edges coloured \rightarrow every MST satisfies the invariant.

35

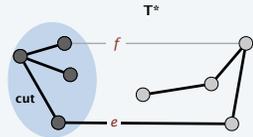
Greedy algorithm: proof of correctness

Colour invariant. There exists an MST (V, T^*) containing every blue edge and no red edge.

Pf. [by induction on number of iterations]

Induction step (blue rule). Suppose the colour invariant is true before applying the blue rule.

- let D be the chosen cutset, and let f be the edge coloured blue.
- if $f \in T^*$, then T^* still satisfies the invariant.
- Otherwise, consider the fundamental cycle C by adding f to T^* .
- let $e \in C$ be another edge in D .
- e is uncoloured and $c_e \geq c_f$ since
 - $e \in T^* \Rightarrow e$ is not red
 - blue rule $\Rightarrow e$ is not blue and $c_e \geq c_f$
- Thus, $T^* \cup \{f\} - \{e\}$ satisfies the invariant.



36

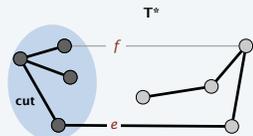
Greedy algorithm: proof of correctness

Colour invariant. There exists an MST (V, T^*) containing every blue edge and no red edge.

Pf. [by induction on number of iterations]

Induction step (red rule). Suppose the colour invariant is true before applying the red rule.

- let C be the chosen cycle, and let e be the edge coloured red.
- if $e \notin T^*$, then T^* still satisfies the invariant.
- Otherwise, consider the fundamental cutset D by deleting e from T^* .
- let $f \in D$ be another edge in C .
- f is uncoloured and $c_e \geq c_f$ since
 - $f \notin T^* \Rightarrow f$ is not blue
 - red rule $\Rightarrow f$ is not red and $c_e \geq c_f$
- Thus, $T^* \cup \{f\} - \{e\}$ satisfies the invariant. ■



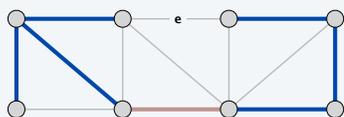
37

Greedy algorithm: proof of correctness

Theorem. The greedy algorithm terminates, and blue edges form an MST.

Pf. We need to show that either the red or blue rule (or both) applies.

- Suppose edge e is left uncoloured.
- Blue edges form a forest.
- Case 1: both endpoints of e are in the same blue tree.
 - \Rightarrow apply the red rule to the cycle formed by adding e to the blue forest.
- Case 2: both endpoints of e are in different blue trees.
 - \Rightarrow apply the blue rule to the cutset induced by either of the two blue trees. ■



Case 2

39

CSCI 355: ALGORITHM DESIGN AND ANALYSIS

4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Borůvka*
- *single-link clustering*

Review: the greedy MST algorithm

Red rule.

- Let C be a cycle with no red edges.
- Select an uncoloured edge of C of max cost and colour it red.

Blue rule.

- Let D be a cutset with no blue edges.
- Select an uncoloured edge in D of min cost and colour it blue.

Greedy algorithm.

- Apply the red and blue rules (nondeterministically!) until all edges are coloured. The blue edges form an MST.
- Note: we can stop once we have $n - 1$ edges coloured blue.

Theorem. The greedy algorithm is correct.

41

Special cases of MST algorithms

Special cases. Prim, Kruskal, reverse-delete, Borůvka, ...

Prim's algorithm.

- Adds edges outward from an arbitrary starting vertex.
- Works well on graphs with many edges (dense graphs).

Kruskal's algorithm.

- Adds edges in order from least cost to greatest cost.
- Works well on graphs with few edges (sparse graphs).

Reverse-delete algorithm.

- Deletes edges in order from greatest cost to least cost.

Borůvka's algorithm.

- Finds all min-cost edges incident to each connected component, and adds those edges to a forest.
- Adapts well to parallelization.

42

Prim's algorithm: MSTs

see 4B

Initialize $S = \{s\}$ for any vertex s , and set $T = \emptyset$.

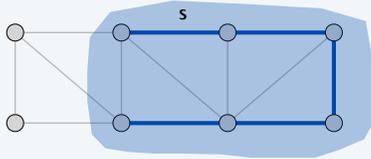
Repeat $n - 1$ times:

- Add to T a min-cost edge with exactly one endpoint in S .
- Add the other endpoint of the edge to S .

Theorem. Prim's algorithm computes an MST.

by construction, edges in cutset are uncolored

Pf. Special case of greedy algorithm (blue rule repeatedly applied to S). ▀



43

Prim's algorithm: implementation

Theorem. Prim's algorithm can be implemented in $O(m \log n)$ time.

Pf. Implementation almost identical to Dijkstra's algorithm.

```

PRIM ( $V, E, c$ )
 $S \leftarrow \emptyset, T \leftarrow \emptyset$ .
 $s \leftarrow$  any node in  $V$ .
FOREACH  $v \neq s : \pi[v] \leftarrow \infty, pred[v] \leftarrow null; \pi[s] \leftarrow 0$ .
Create an empty priority queue  $pq$ .
FOREACH  $v \in V : INSERT(pq, v, \pi[v])$ .
WHILE (IS-NOT-EMPTY( $pq$ ))
     $u \leftarrow DEL-MIN(pq)$ .
     $S \leftarrow S \cup \{u\}, T \leftarrow T \cup \{pred[u]\}$ .
    FOREACH edge  $e = (u, v) \in E$  with  $v \notin S$ :
        IF ( $c_e < \pi[v]$ )
            DECREASE-KEY( $pq, v, c_e$ ).
             $\pi[v] \leftarrow c_e; pred[v] \leftarrow e$ .
    
```

$\pi[v]$ = cost of cheapest known edge between v and S

44

Kruskal's algorithm: MSTs

see 4B

Consider edges in ascending order of cost:

- Add the edge to the tree unless it would create a cycle.

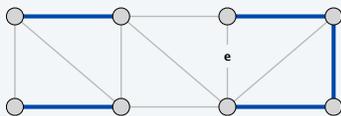
Theorem. Kruskal's algorithm computes an MST.

Pf. Special case of greedy algorithm.

- Case 1: both endpoints of e in same blue tree.
 - color e red by applying red rule to unique cycle.
- Case 2: both endpoints of e in different blue trees.
 - color e blue by applying blue rule to cutset defined by either tree. ▀

all other edges in cycle are blue

no edge in cutset has smaller cost (since Kruskal chose it first)



45

Kruskal's algorithm: implementation

Theorem. Kruskal's algorithm can be implemented in $O(m \log m)$ time.

Pf.

- Sort edges by cost.
- Use **union-find** data structure to dynamically maintain connected components.

```

KRUSKAL ( $V, E, c$ )
  SORT  $m$  edges by cost and renumber so that  $c(e_1) \leq c(e_2) \leq \dots \leq c(e_m)$ .
   $T \leftarrow \emptyset$ .
  FOREACH  $v \in V$ : MAKE-SET( $v$ ).
  FOR  $i = 1$  TO  $m$ 
     $(u, v) \leftarrow e_i$ .
    IF (FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ))  $\leftarrow$  are  $u$  and  $v$  in
      same component?
       $T \leftarrow T \cup \{e_i\}$ .
      UNION( $u, v$ ).  $\leftarrow$  make  $u$  and  $v$  in
        same component
  RETURN  $T$ .
    
```

46

Reverse-delete algorithm: MSTs

see 4B

Start with all edges in T and consider them in descending order of cost:

- Delete each edge from T unless doing so would disconnect T .

Theorem. The reverse-delete algorithm computes an MST.

Pf. Special case of greedy algorithm.

- Case 1. [deleting edge e does not disconnect T]
 - \rightarrow apply red rule to cycle C formed by adding e to another path in T between its two endpoints.
 - \leftarrow no edge in C is more expensive (it would have already been considered and deleted)
- Case 2. [deleting edge e disconnects T]
 - \rightarrow apply blue rule to cutset D induced by either component.
 - \leftarrow e is the only remaining edge in the cutset (all other edges in D must have been colored red / deleted)

Fact. [Thorup 2000] Reverse-delete can be implemented in $O(m \log n (\log \log n)^3)$ time.

47

Borůvka's algorithm: MSTs

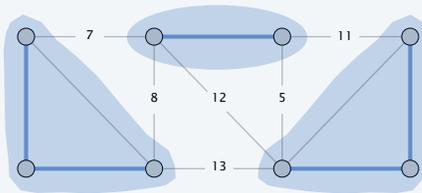
see 4B

Repeat until only one tree remains:

- Apply blue rule to the cutset corresponding to **each** blue tree.
- Color **all** selected edges blue.

Theorem. Borůvka's algorithm computes the MST. \leftarrow assuming edge costs are distinct

Pf. Special case of greedy algorithm (repeatedly apply blue rule). ▀



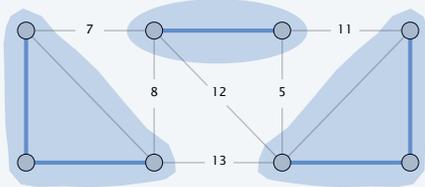
48

Borůvka's algorithm: implementation

Theorem. Borůvka's algorithm can be implemented in $O(m \log n)$ time.

Pf.

- To implement a phase in $O(m)$ time:
 - compute connected components of blue edges
 - for each edge $(u, v) \in E$, check if u and v are in different components; if so, update each component's best edge in cutset
- $\leq \log_2 n$ phases since each phase (at least) halves total # components. ■



49

Does a linear-time comparison-based MST algorithm exist?

year	worst case	discovered by
1975	$O(m \log \log n)$	Yao
1976	$O(m \log \log n)$	Cheriton–Tarjan
1984	$O(m \log^* n)$, $O(m + n \log n)$	Fredman–Tarjan
1986	$O(m \log (\log^* n))$	Gabow–Galil–Spencer–Tarjan
1997	$O(m \alpha(n) \log \alpha(n))$	Chazelle
2000	$O(m \alpha(n))$	Chazelle
2002	asymptotically optimal	Pettie–Ramachandran
20xx	$O(m)$???

deterministic compare-based MST algorithms

iterated logarithm function

$$\lg^* n = \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + \lg^*(\lg n) & \text{if } n > 1 \end{cases}$$

n	$\lg^* n$
$(-\infty, 1]$	0
(1, 2]	1
(2, 4]	2
(4, 16]	3
(16, 2^{16}]	4
(2^{16} , 2^{65536}]	5

(α : inverse Ackermann function)

Theorem. [Fredman–Willard 1990] $O(m)$ in word RAM model.

Theorem. [Dixon–Rauch–Tarjan 1992] $O(m)$ MST verification algorithm.

Theorem. [Karger–Klein–Tarjan 1995] $O(m)$ randomized MST algorithm.

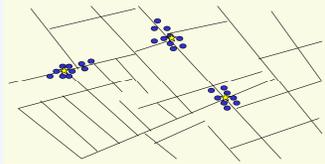
50

CSCI 355: ALGORITHM DESIGN AND ANALYSIS 4. GREEDY ALGORITHMS II

- ▶ Dijkstra's algorithm
- ▶ minimum spanning trees
- ▶ Prim, Kruskal, Borůvka
- ▶ single-link clustering

Clustering

Goal. Given a set U of n objects labeled p_1, \dots, p_n , partition the objects into clusters so that objects in different clusters are far apart.



outbreak of cholera deaths in London in 1850s (Nina Mishra)



Applications.

- Routing in mobile ad-hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases
- Cluster celestial objects into stars, quasars, galaxies.

52

Clustering with maximum spacing

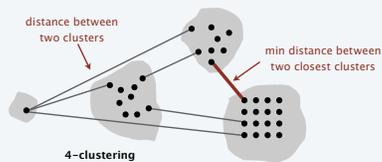
k-clustering. Divide objects into k non-empty groups.

Distance function. Numeric value specifying "closeness" of two objects.

- $d(p_i, p_j) = 0$ iff $p_i = p_j$ [identity of indiscernibles]
- $d(p_i, p_j) \geq 0$ [non-negativity]
- $d(p_i, p_j) = d(p_j, p_i)$ [symmetry]

Spacing. Min distance between any pair of points in different clusters.

Goal. Given an integer k , find a k -clustering with maximum spacing.



53

Greedy clustering algorithm

"Well-known" algorithm for single-linkage k -clustering:

- Form a graph on the vertex set U , corresponding to n clusters.
- Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
- Repeat $n - k$ times (until there are exactly k clusters).



Key observation. This procedure is precisely Kruskal's algorithm (except we stop when there are k connected components).

Alternative. Find an MST and delete the $k - 1$ longest edges.

54

Dendrogram of cancers in human

Tumors in similar tissues cluster together.

