CS 3100
Data Structures and Algorithms 2
Lecture 7: Divide and Conquer

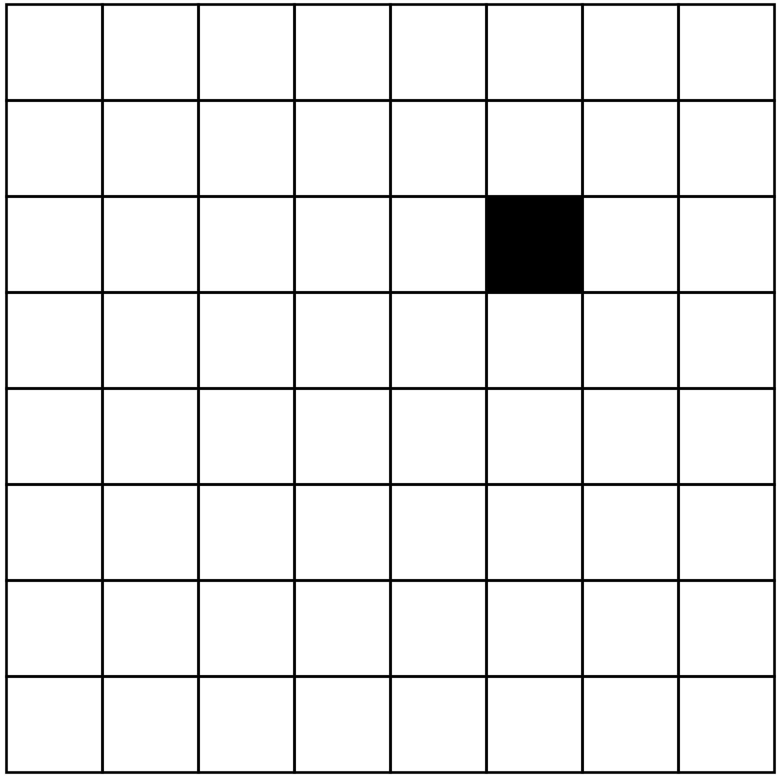
Co-instructors: Robbie Hott and Tom Horton

Fall 2023

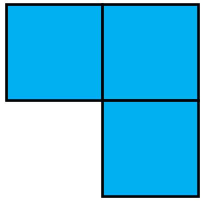
Readings in CLRS 4th edition:

- Section 22.3, Chapter 4, 4.3, 4.4

Question



Can you cover an 8×8 grid with 1 square missing using "trominoes?"



Tromino

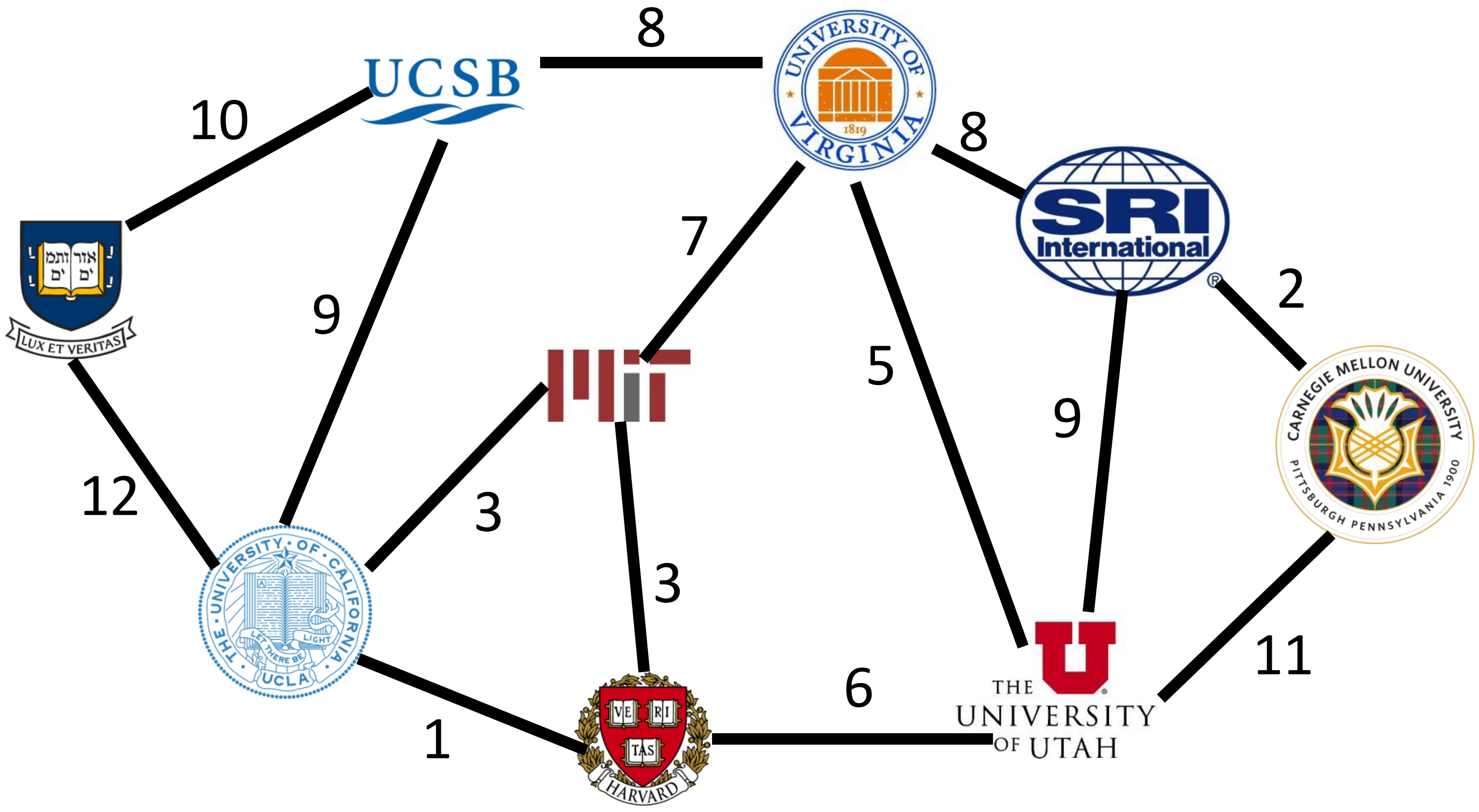
https://nstarr.people.amherst.edu/trom/puzzle-8by8/

Announcements

- Upcoming dates
  - PS1 due tonight at 11:59pm
  - PA1 due Sept 17 (Sunday) at 11:59pm
- Course email (comes to both professors and head TAs):

cs3100@cshelpdesk.atlassian.net

Single-Source Shortest Path Problem



Find the <u>shortest path</u> based on sum of edge-weights from UVA to each of these other places.

The problem: Given a graph $G = (V, E)$ and a start node (i.e., source) $s \in V$,

for each $v \in V$ find the minimum-weight path from $s \to v$ (call this weight $\delta(s, v)$)

Assumption (for this unit): all edge weights are positive

Dijkstra's Algorithm Implementation

1. Start with an empty tree S and add the source to S
2. Repeat $|V| - 1$ times:
   - Add the node to S that's not yet in S and that's "nearest" to source

Implementation:

initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
  $v = \mathrm{PQ}.\mathrm{extractMin}()$
  for each $u \in V$ such that $(v, u) \in E$:
    if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
      $\mathrm{PQ}.\mathrm{decreaseKey}\big(u, d_v + w(v, u)\big)$
      $u.\mathrm{parent} = v$

each node also maintains a parent, initially `NULL`

key: length of shortest path $s \to u$ using nodes in PQ

Dijkstra's Algorithm Implementation

Implementation:

initialize $d_v = \infty$ for each node v

add all nodes $v \in V$ to the priority queue PQ, using d_v as the key

set $d_s = 0$

while PQ is not empty:

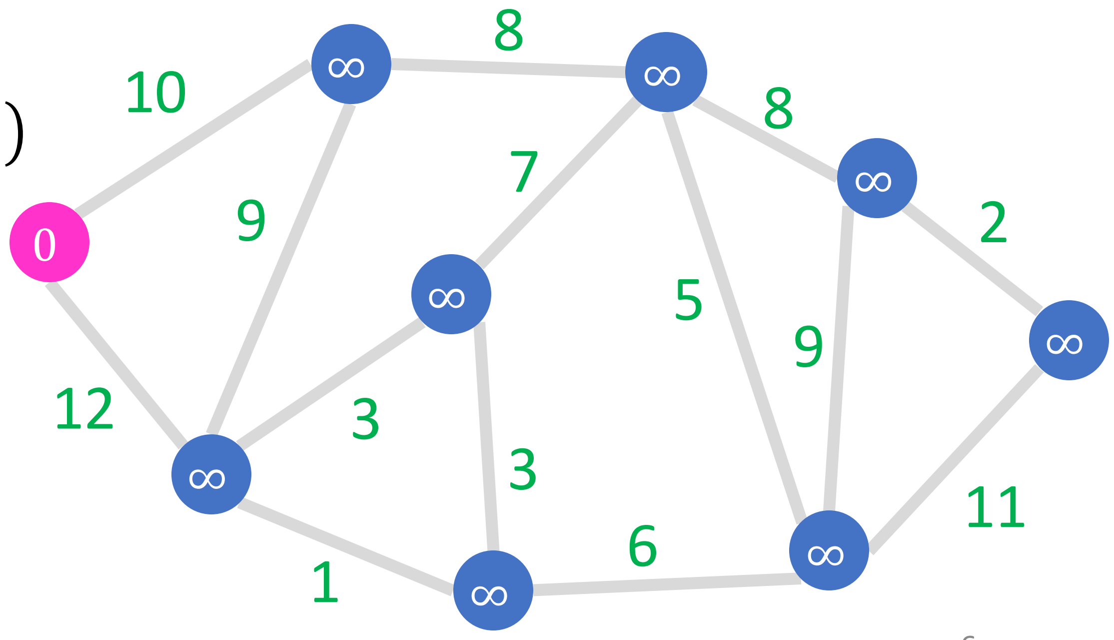
$v = \mathrm{PQ}.\mathrm{extractMin}()$

for each $u \in V$ such that $(v, u) \in E$:

if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:

$\mathrm{PQ}.\mathrm{decreaseKey}\big(u, d_v + w(v, u)\big)$

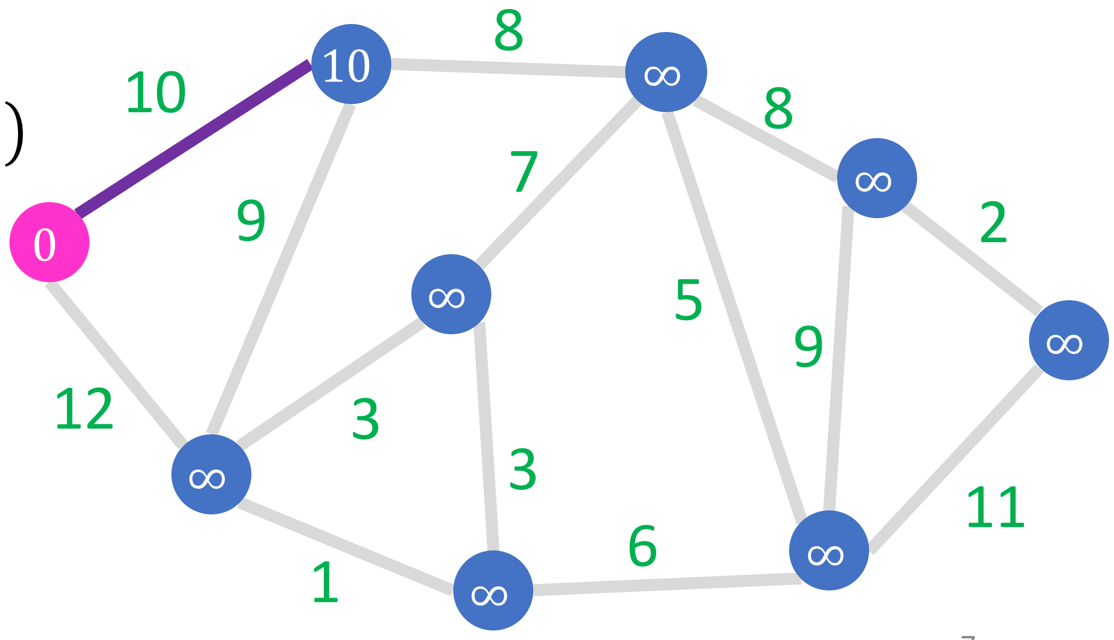
$u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

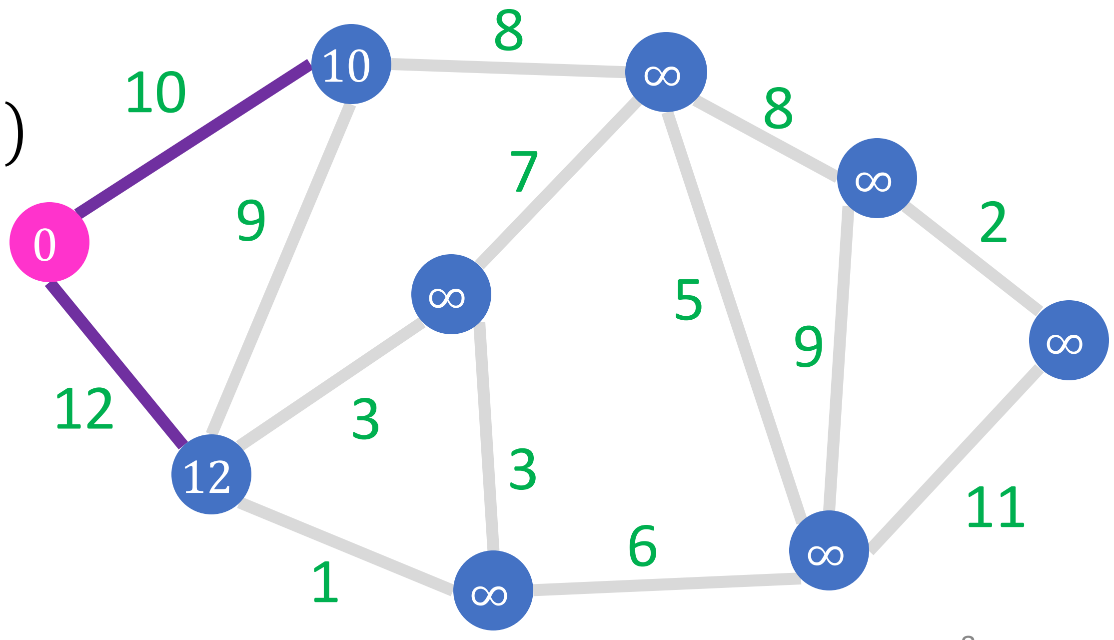
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
  $v = \mathrm{PQ.extractMin}()$
  for each $u \in V$ such that $(v, u) \in E$:
    if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
      $\mathrm{PQ.decreaseKey}\big(u, d_v + w(v, u)\big)$
      $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

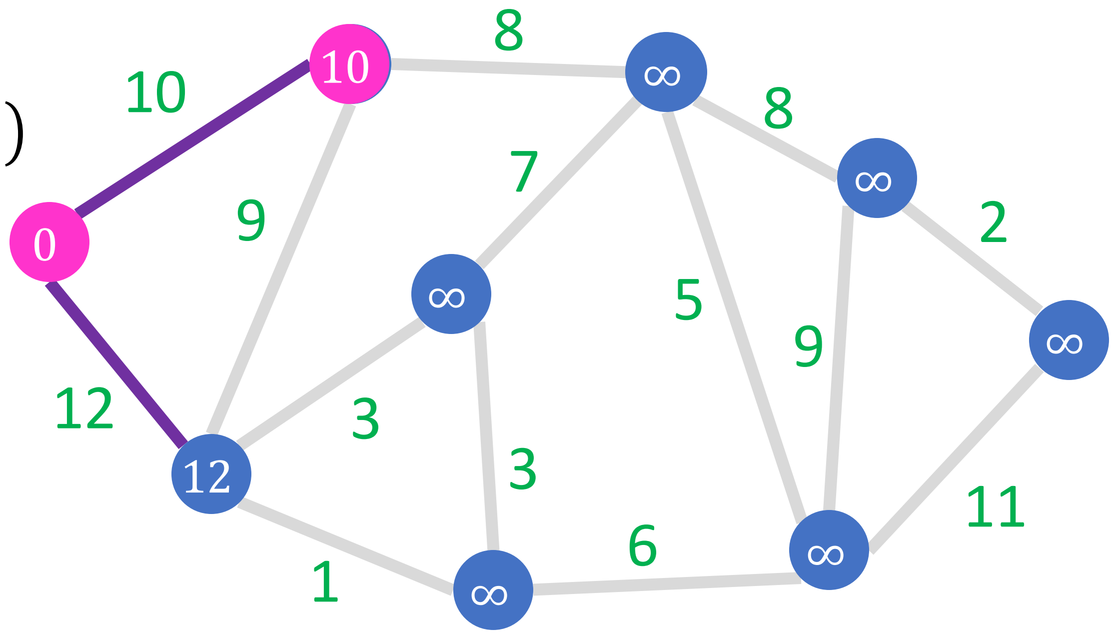
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
    $v = \mathrm{PQ.extractMin}()$
    for each $u \in V$ such that $(v, u) \in E$:
        if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
            $\mathrm{PQ.decreaseKey}(u, d_v + w(v, u))$
            $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

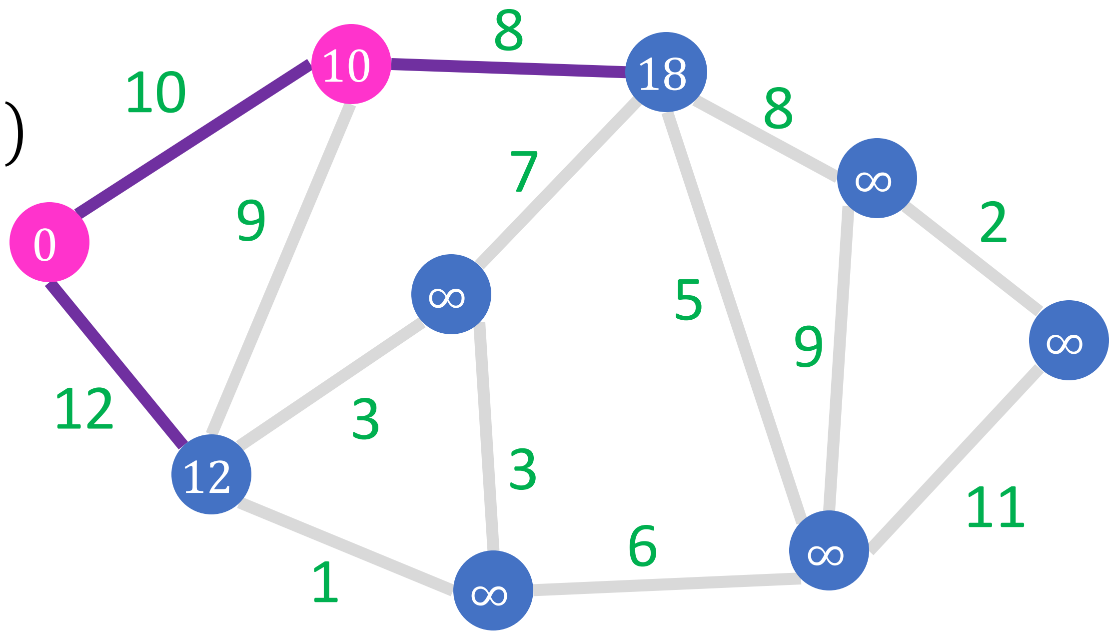
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
    $v = \mathrm{PQ}.\mathrm{extractMin}()$
    for each $u \in V$ such that $(v, u) \in E$:
        if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
            $\mathrm{PQ}.\mathrm{decreaseKey}\big(u, d_v + w(v, u)\big)$
            $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

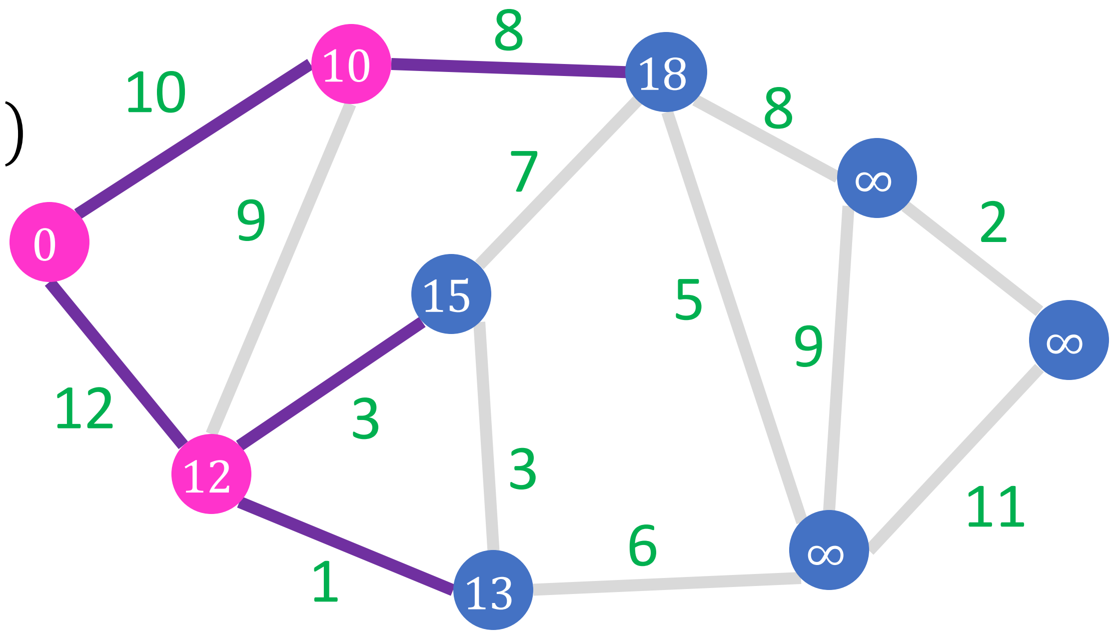
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
  $v = \mathrm{PQ}.\mathrm{extractMin}()$
  for each $u \in V$ such that $(v, u) \in E$:
    if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
      $\mathrm{PQ}.\mathrm{decreaseKey}\big(u, d_v + w(v, u)\big)$
      $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

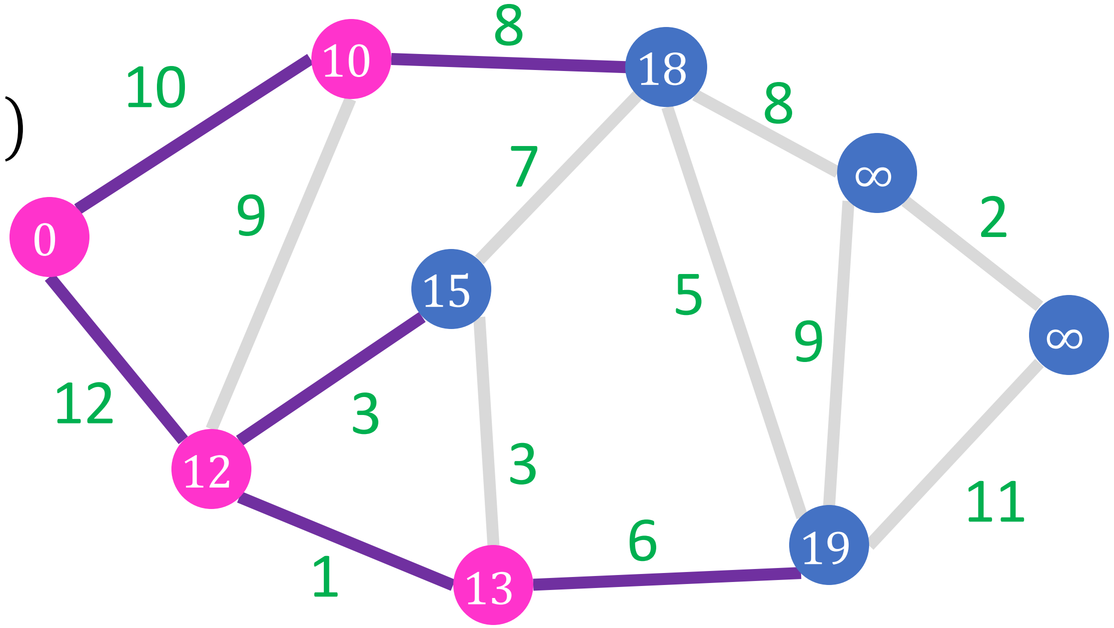
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
  $v = \mathrm{PQ}.\mathrm{extractMin}()$
  for each $u \in V$ such that $(v, u) \in E$:
    if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
      $\mathrm{PQ}.\mathrm{decreaseKey}\big(u, d_v + w(v, u)\big)$
      $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

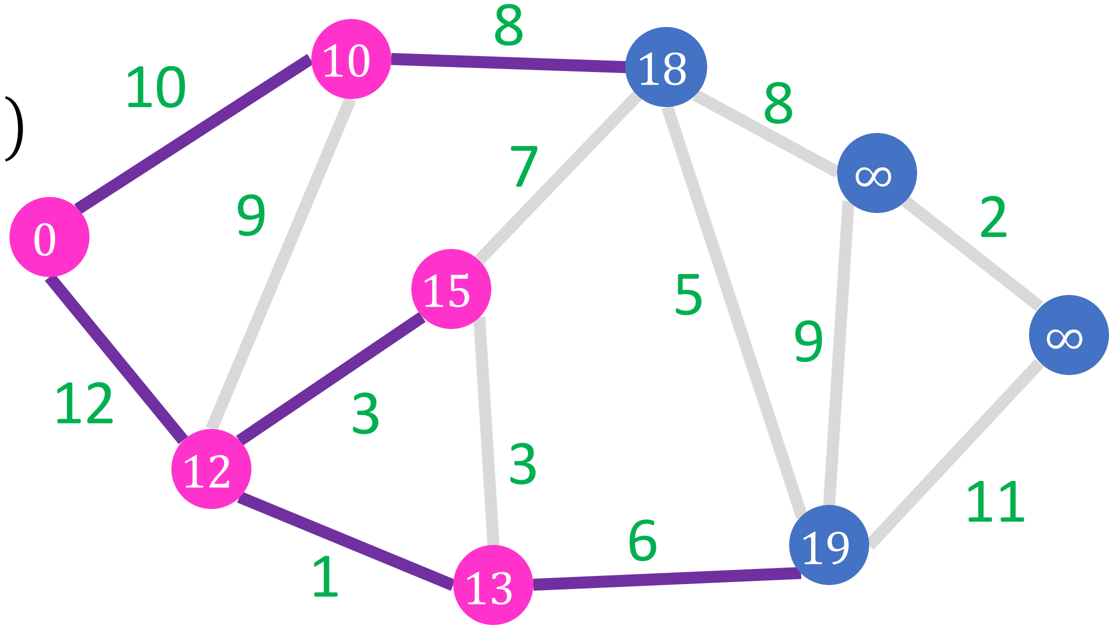
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
    $v = \mathrm{PQ.extractMin}()$
    for each $u \in V$ such that $(v, u) \in E$:
        if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
            $\mathrm{PQ.decreaseKey}\big(u, d_v + w(v, u)\big)$
            $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

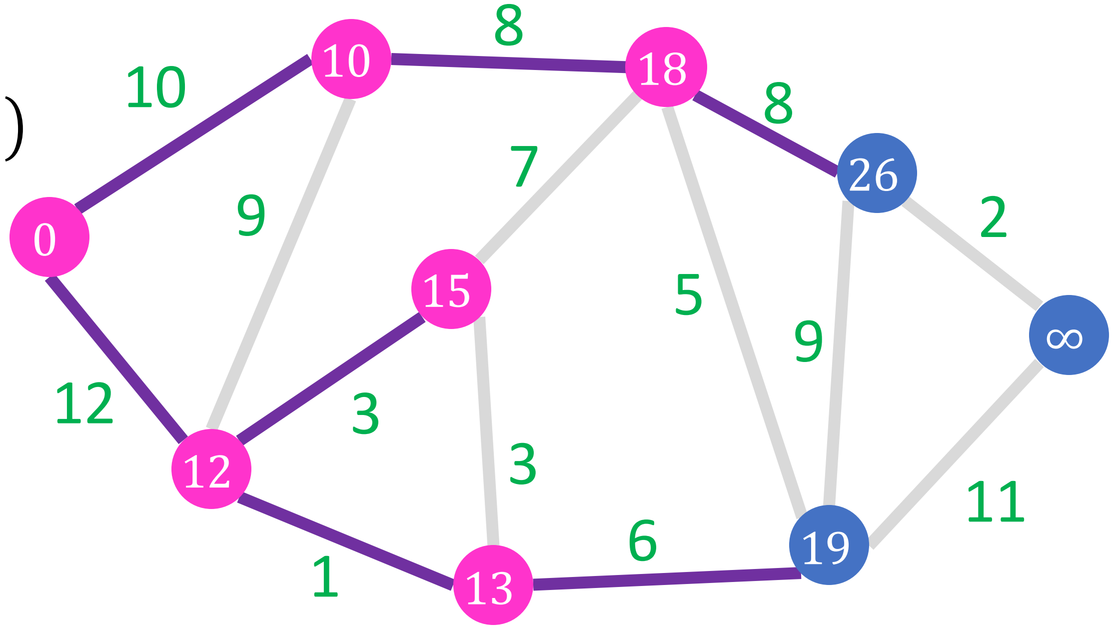
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
    $v = \mathrm{PQ.extractMin}()$
    for each $u \in V$ such that $(v, u) \in E$:
        if $u \in \mathrm{PQ}$ and $d_v + w(v,u) < d_u$:
            $\mathrm{PQ.decreaseKey}\big(u, d_v + w(v,u)\big)$
            $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

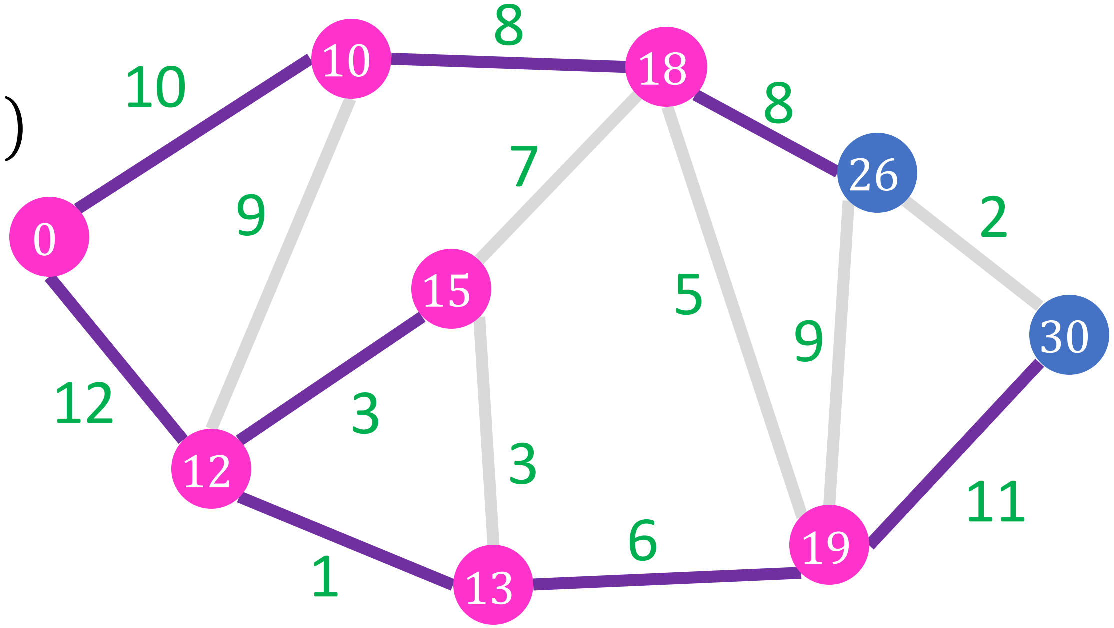
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
    $v = \mathrm{PQ.extractMin}()$
    for each $u \in V$ such that $(v, u) \in E$:
        if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
            $\mathrm{PQ.decreaseKey}(u, d_v + w(v, u))$
            $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

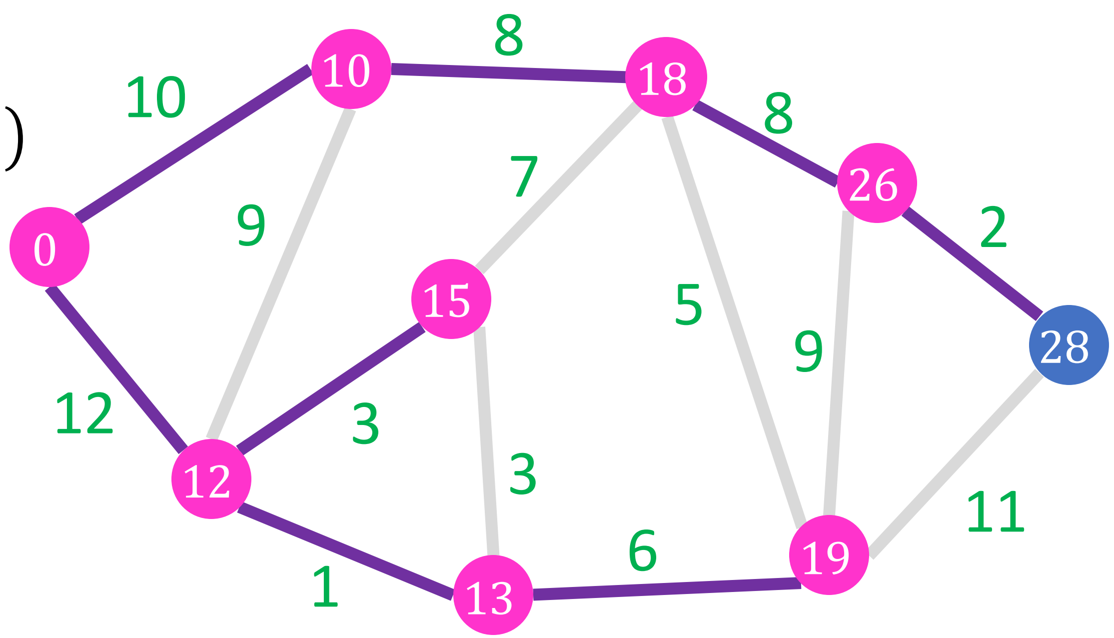
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
  $v = \mathrm{PQ}.\mathrm{extractMin}()$
  for each $u \in V$ such that $(v, u) \in E$:
    if $u \in \mathrm{PQ}$ and $d_v + w(v,u) < d_u$:
      $\mathrm{PQ}.\mathrm{decreaseKey}(u, d_v + w(v,u))$
      $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

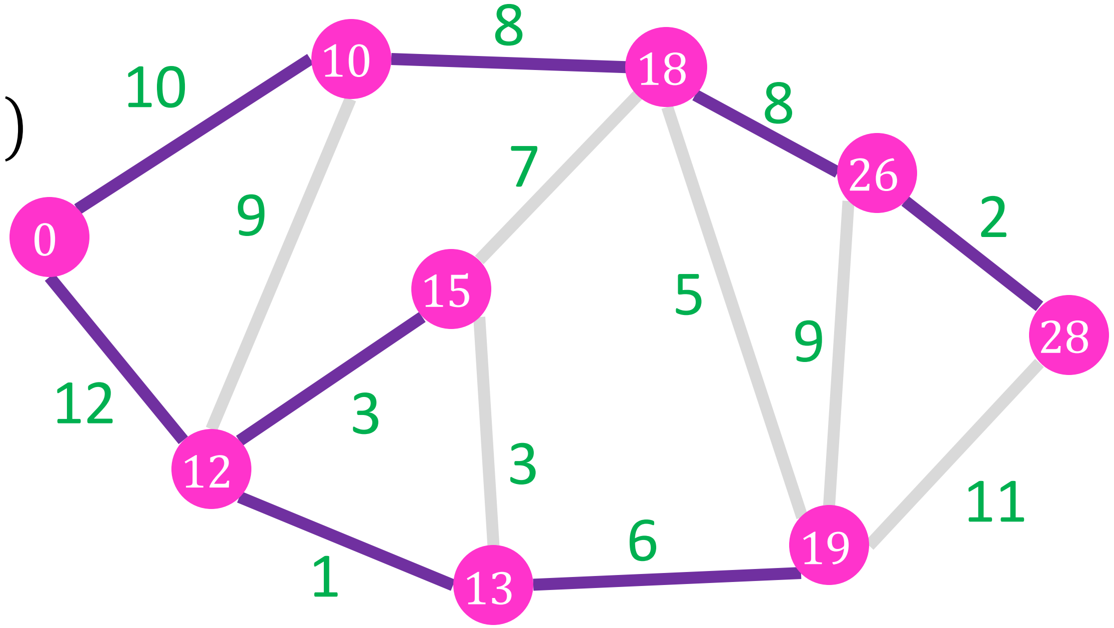
initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
  $v = \mathrm{PQ.extractMin}()$
  for each $u \in V$ such that $(v, u) \in E$:
    if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
      $\mathrm{PQ.decreaseKey}\big(u, d_v + w(v, u)\big)$
      $u.\mathrm{parent} = v$



Dijkstra's Algorithm Implementation

Implementation:

initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
  $v = \mathrm{PQ.extractMin()}$
  for each $u \in V$ such that $(v, u) \in E$:
    if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
      $\mathrm{PQ.decreaseKey}\big(u, d_v + w(v, u)\big)$
      $u.\mathrm{parent} = v$



Observe: shortest paths from a source forms a <u>tree</u>, shortest path to every reachable node

Every subpath of a shortest path is itself a shortest path. (This is called the *optimal substructure property*.)

Dijkstra's Algorithm Running Time

Implementation:

initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
    $v = \mathrm{PQ}.\mathrm{extractMin}()$
    for each $u \in V$ such that $(v, u) \in E$:
        if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
            $\mathrm{PQ}.\mathrm{decreaseKey}\big(u, d_v + w(v, u)\big)$
            $u.\mathrm{parent} = v$

Initialization:
$O(|V|)$

$|V|$ iterations

$O(\log|V|)$

$|E|$ iterations total

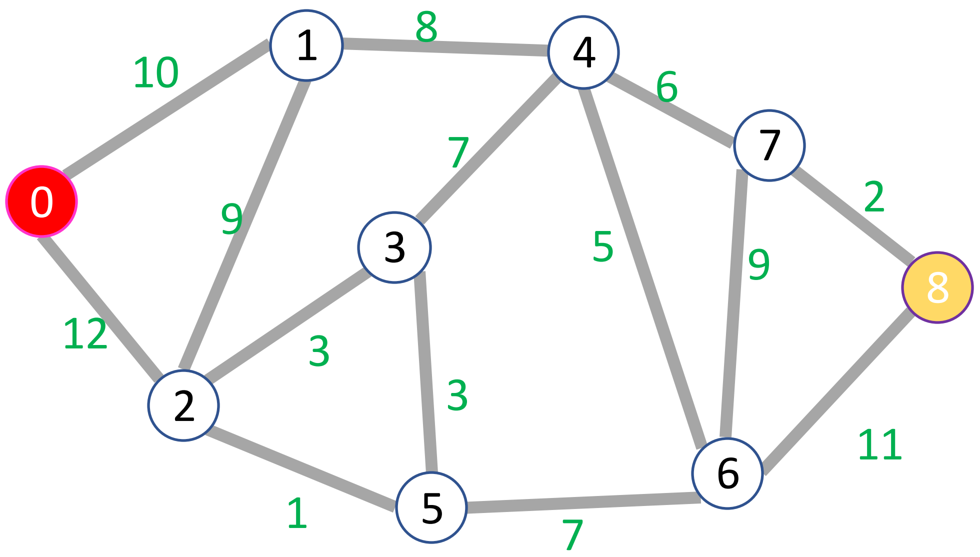
$??$ $O(\log|V|)$ if we use indirect heaps

$\boxed{\begin{array}{l}|V| = n \\ |E| = m\end{array}}$

Overall running time: $O(|V| \log|V| + |E| \log|V|) = O(|E| \log|V|)$
or, $O(m \log n)$

Python-like Code for Dijkstra's Algorithm

```
def Dijkstras(graph, start, end):
        distances = [∞, ∞, ∞,…]  # one index per node
        done = [False,False,False,…]  # one index per node
        PQ = priority queue  # e.g. a min heap
        PQ.insert((0, start))
        distances[start] = 0
        while PQ is not empty:
                current = PQ.extractmin()
                if done[current]: continue
                done[current] = True
                for each neighbor of current:
                        if not done[neighbor]:
                                new_dist = distances[current]+weight(current,neighbor)
                                if new_dist < distances[neighbor]:
                                        distances[neighbor] = new_dist
                                        PQ.insert((new_dist,neighbor))
        return distances[end]
```



Dijkstra's Algorithm

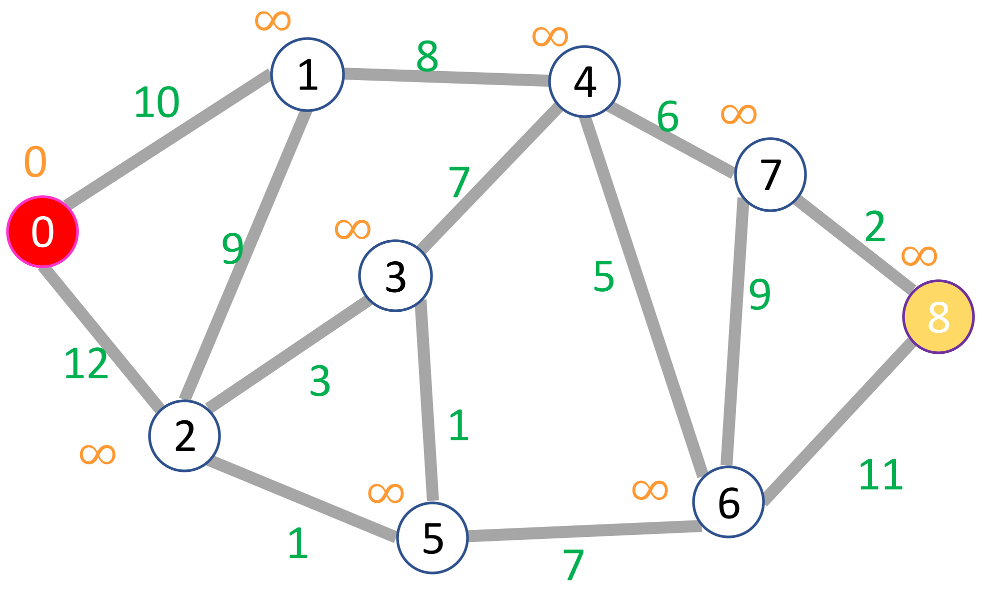
Start: 0

End: 8

Idea: When a node is the closest undiscovered thing to the start, we have found its shortest path

Node	Done?
0	F
1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F

Node	Distance
0	0
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞



Dijkstra’s Algorithm

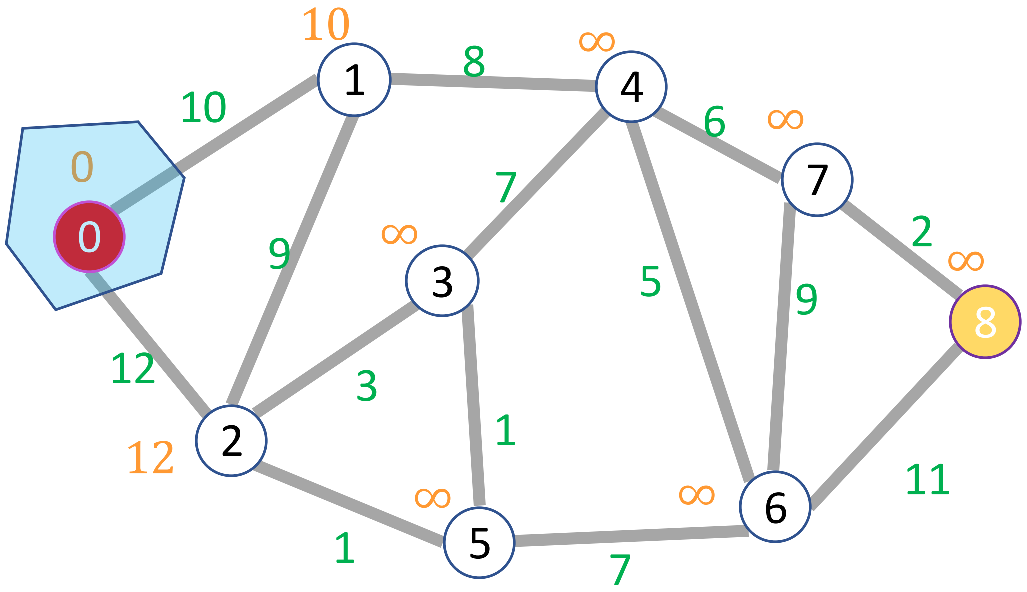
Start: 0

End: 8

Idea: When a node is the closest undiscovered thing to the start, we have found its shortest path

Node	Done?
0	T
1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F

Node	Distance
0	0
1	10
2	12
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞



Dijkstra's Algorithm

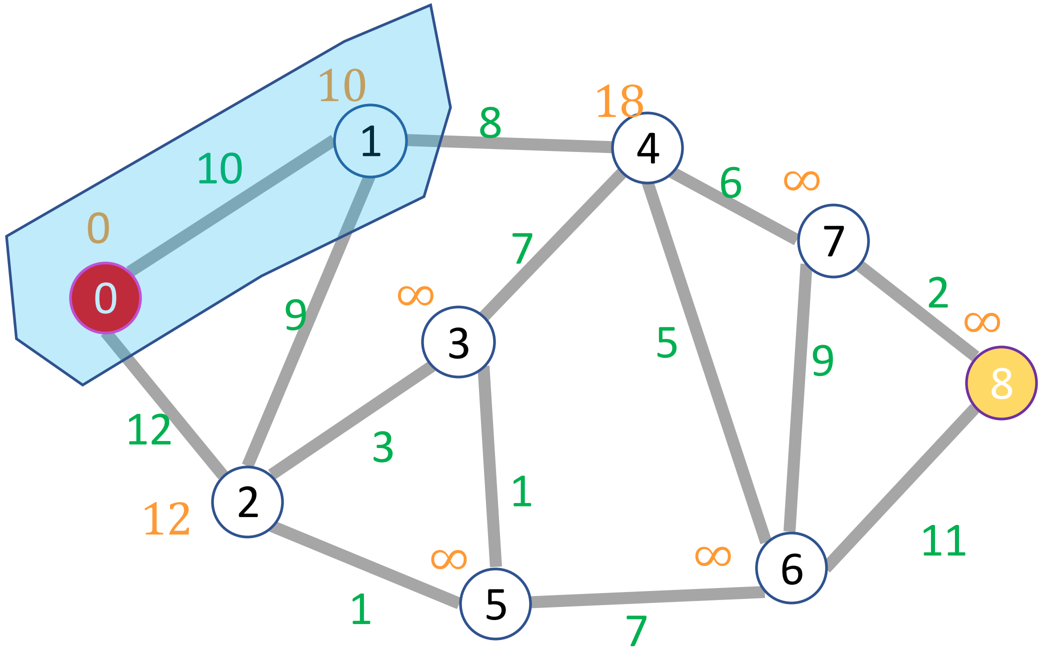
Start: 0

End: 8

Idea: When a node is the closest undiscovered thing to the start, we have found its shortest path

Node	Done?
0	T
1	T
2	F
3	F
4	F
5	F
6	F
7	F
8	F

Node	Distance
0	0
1	10
2	12
3	∞
4	18
5	∞
6	∞
7	∞
8	∞



Dijkstra's Algorithm

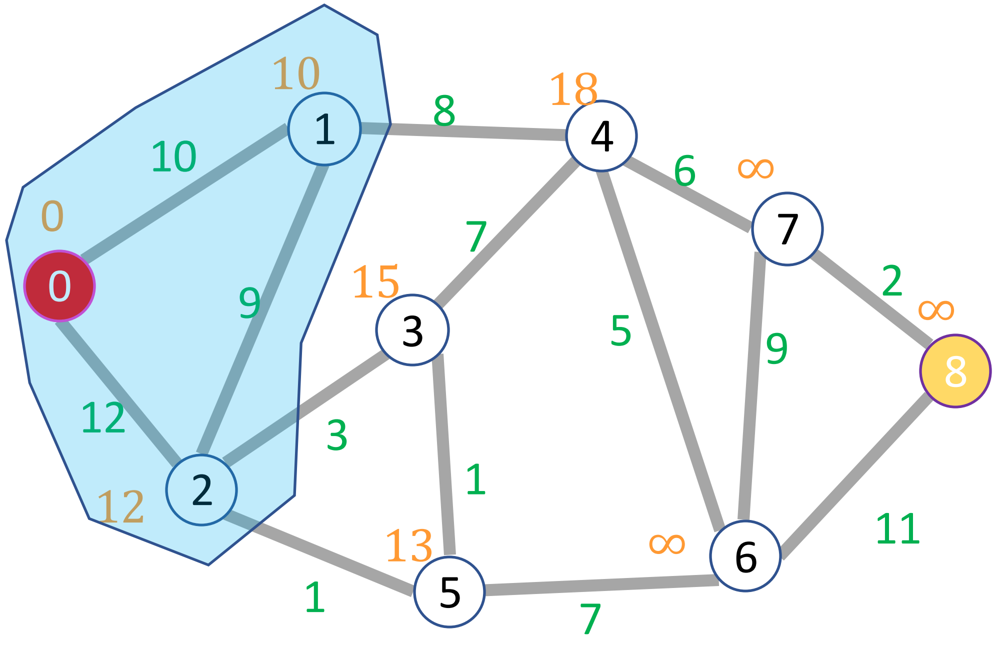
Start: 0

End: 8

Idea: When a node is the closest undiscovered thing to the start, we have found its shortest path

Node	Done?
0	T
1	T
2	T
3	F
4	F
5	F
6	F
7	F
8	F

Node	Distance
0	0
1	10
2	12
3	15
4	18
5	13
6	∞
7	∞
8	∞



Dijkstra's Algorithm

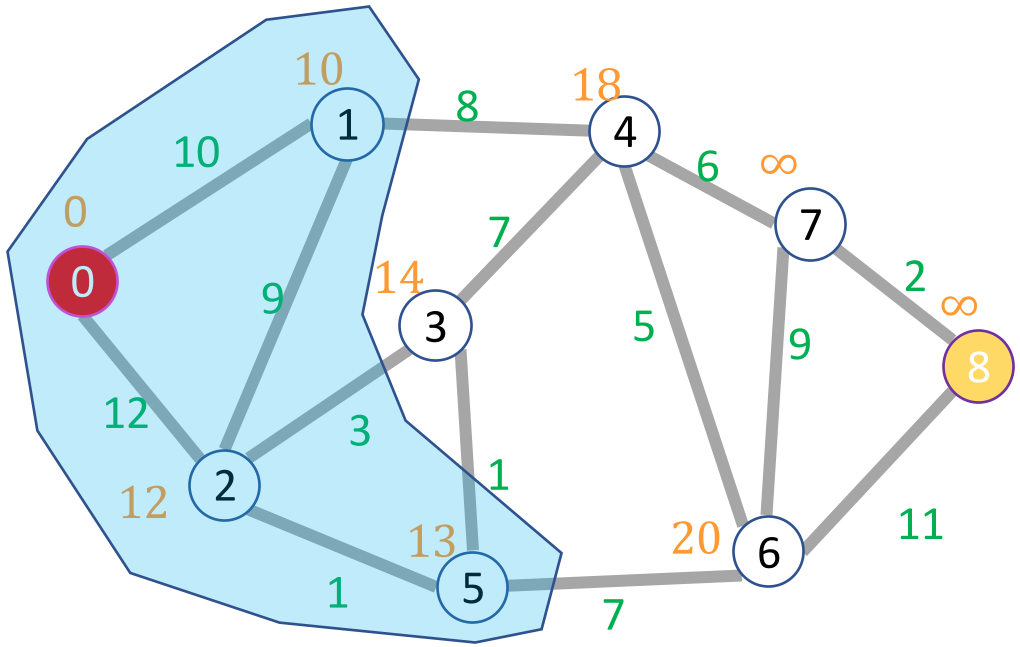
Start: 0

End: 8

Idea: When a node is the closest undiscovered thing to the start, we have found its shortest path

Node	Done?
0	T
1	T
2	T
3	F
4	F
5	T
6	F
7	F
8	F

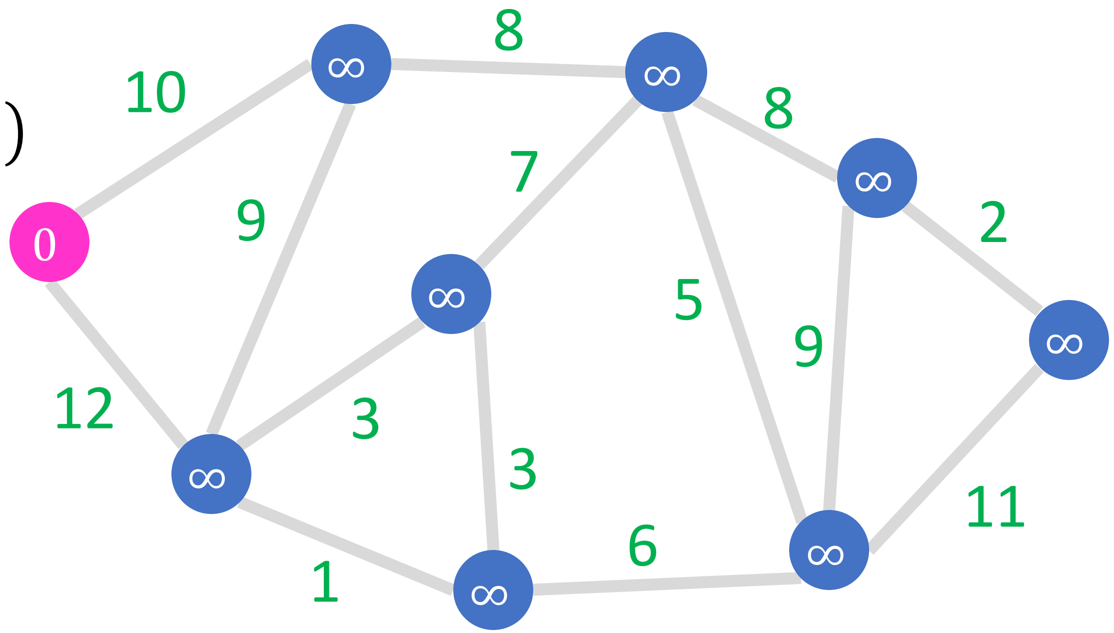
Node	Distance
0	0
1	10
2	12
3	14
4	18
5	13
6	∞
7	20
8	∞



Dijkstra's Algorithm Implementation

Implementation:

initialize $d_v = \infty$ for each node v
add all nodes $v \in V$ to the priority queue PQ, using d_v as the key
set $d_s = 0$
while PQ is not empty:
    $v = \mathrm{PQ.extractMin}()$
    for each $u \in V$ such that $(v, u) \in E$:
        if $u \in \mathrm{PQ}$ and $d_v + w(v, u) < d_u$:
            $\mathrm{PQ.decreaseKey}\big(u, d_v + w(v, u)\big)$
            $u.\mathrm{parent} = v$



Dijkstra's Algorithm Proof Strategy

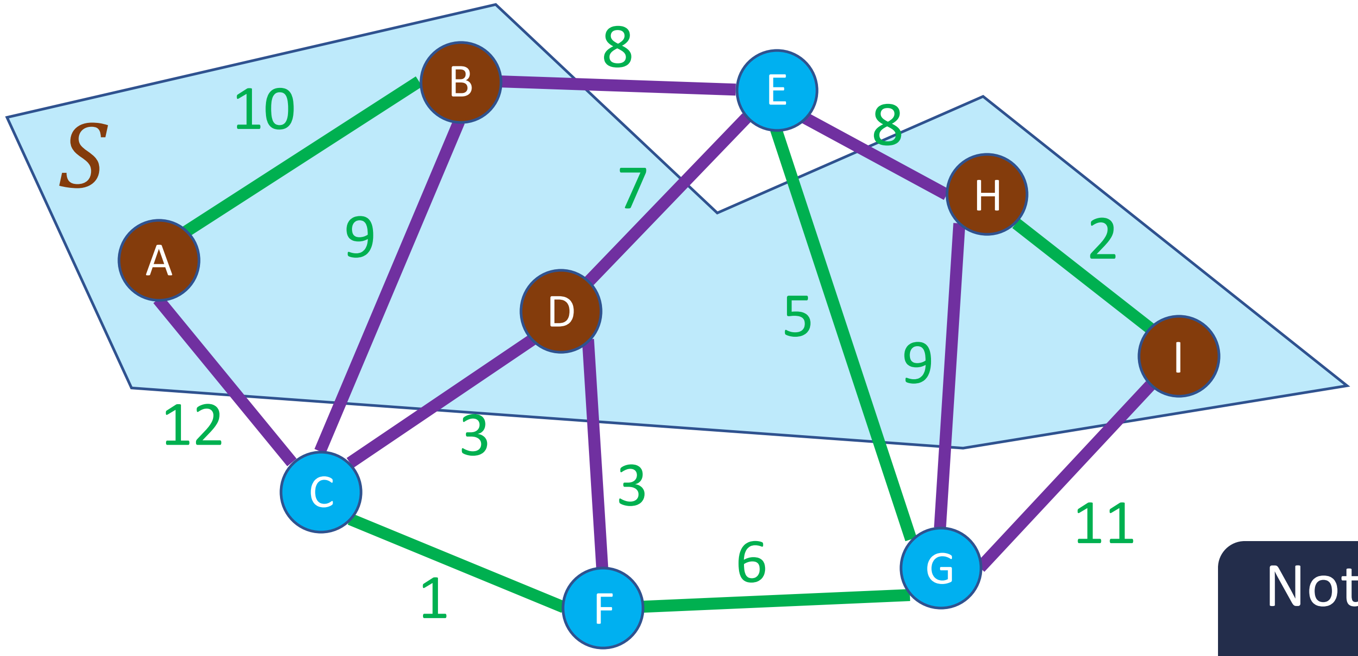
Proof by induction

Proof Idea: we will show that when node u is removed from the priority queue, $d_u = \delta(s,u)$ where $\delta(s,u)$ is the shortest distance

- **Claim 1:** There is a path of length d_u (as long as $d_u < \infty$) from s to u in G
- **Claim 2:** For every path $(s, \ldots, u)$, $w(s, \ldots, u) \geq d_u$

Graph Cuts

A **cut** of a graph $G = (V, E)$ is a partition of the nodes into two sets, S and $V - S$



Notion extends naturally to a set of edges

An edge $(v_1, v_2) \in E$ crosses a cut if $v_1 \in S$ and $v_2 \in V - S$

An edge $(v_1, v_2) \in E$ respects a cut if $v_1, v_2 \in S$ or if $v_1, v_2 \in V - S$

Correctness of Dijkstra's Algorithm

Inductive hypothesis: Suppose that nodes $v_1 = s, \ldots, v_i$ have been removed from PQ, and for each of them $d_{v_i} = \delta(s, v_i)$, and there is a path from s to v_i with distance d_{v_i} (whenever $d_{v_i} < \infty$)

Base case:

- $i = 0$: $v_1 = s$
- Claim holds trivially

Correctness of Dijkstra's Algorithm: Claim 1

Let u be the $(i+1)^{\text{st}}$ node extracted

Claim 1: There is a path of length d_u (as long as $d_u < \infty$) from s to u in G

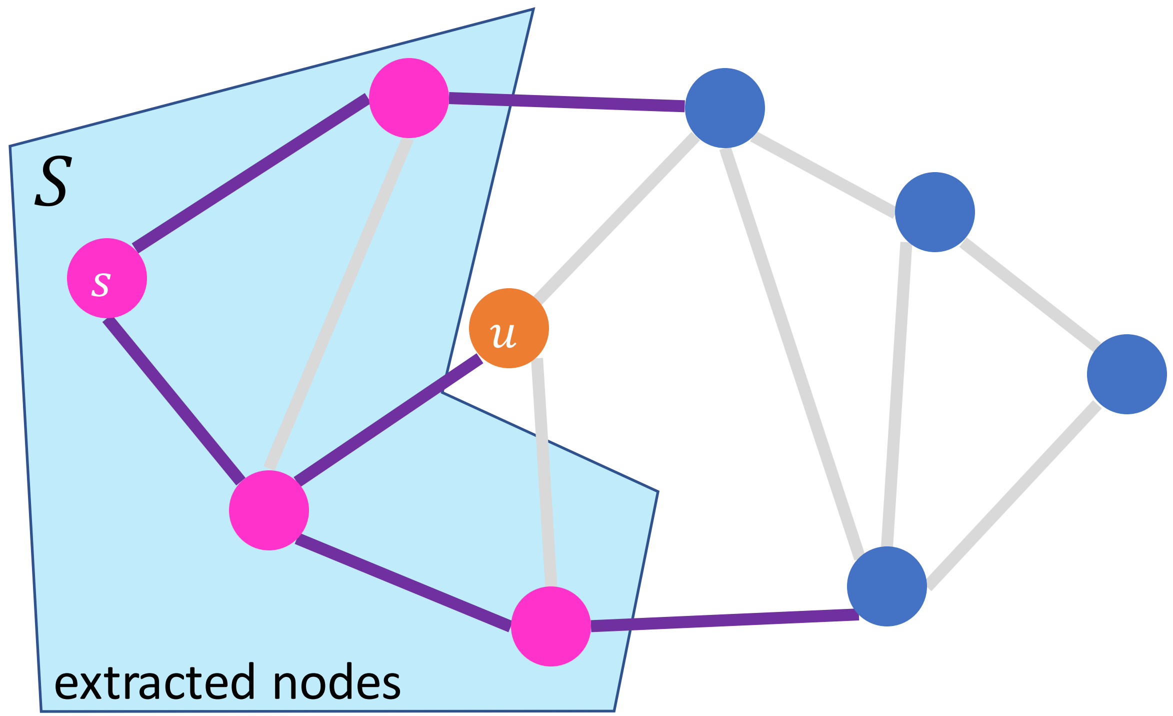
Proof:

- Suppose $d_u < \infty$
- This means that $\mathrm{PQ.decreaseKey}$ was invoked on node u on an earlier iteration
- Consider the last time $\mathrm{PQ.decreaseKey}$ is invoked on node u
- $\mathrm{PQ.decreaseKey}$ is only invoked when there exists an edge $(v, u) \in E$ and node v was extracted from PQ in a previous iteration
- In this case, $d_u = d_v + w(v, u)$
- By the inductive hypothesis, there is a path $s \to v$ of length d_v in G and since there is an edge $(v, u) \in E$, there is a path $s \to u$ of length d_u in G

Correctness of Dijkstra's Algorithm: Claim 2

Let u be the $(i+1)^{\text{st}}$ node extracted

Claim 2: For every path $(s, \dots, u)$, $w(s, \dots, u) \geq d_u$

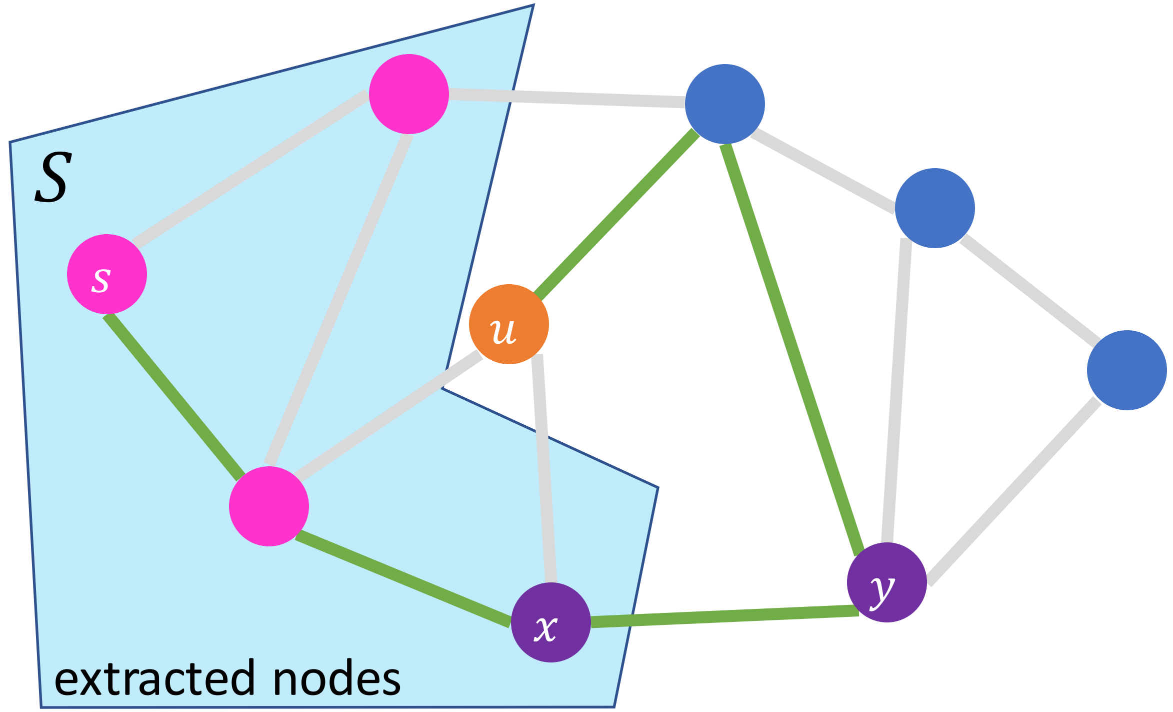


Extracted nodes "cuts" G into two subsets, $(S, V - S)$

Correctness of Dijkstra's Algorithm: Claim 2

Let u be the $(i+1)^{\text{st}}$ node extracted

Claim 2: For every path $(s, \dots, u)$, $w(s, \dots, u) \geq d_u$



Extracted nodes "cuts" G into $(S, V - S)$

Take any path $(s, \dots, u)$

Since $u \notin S$, $(s, \dots, u)$ crosses the cut somewhere

- Let (x, y) be last edge in the path that crosses the cut

$$w(s, \dots, u) \;\; \geq \;\; \delta(s, x) + w(x, y) + w(y, \dots, u)$$

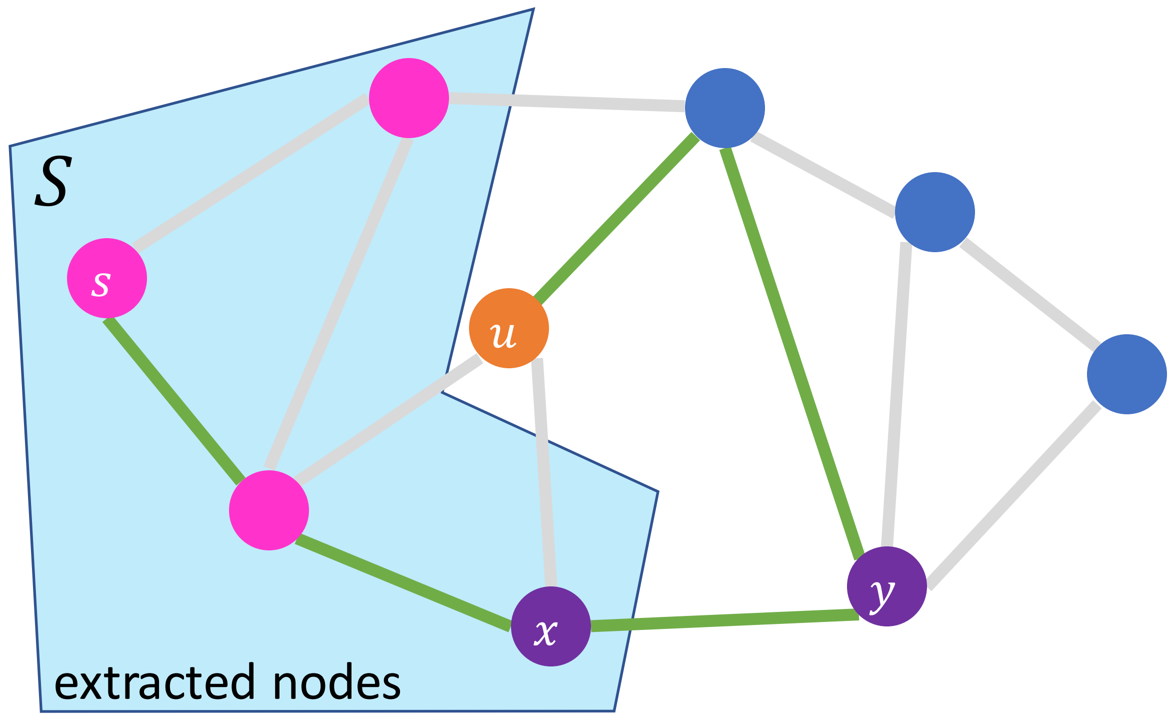
$$w(s, \dots, u) = w(s, \dots, x) + w(x, y) + w(y, \dots, u)$$

$w(s, \dots, x) \geq \delta(s, x)$ since $\delta(s, x)$ is weight of shortest path from s to x

Correctness of Dijkstra's Algorithm: Claim 2

Let u be the $(i+1)^{\text{st}}$ node extracted

Claim 2: For every path $(s, \ldots, u)$, $w(s, \ldots, u) \geq d_u$



Extracted nodes "cuts" G into $(S, V - S)$

Take any path $(s, \ldots, u)$

Since $u \notin S$, $(s, \ldots, u)$ crosses the cut somewhere

- Let (x, y) be last edge in the path that crosses the cut

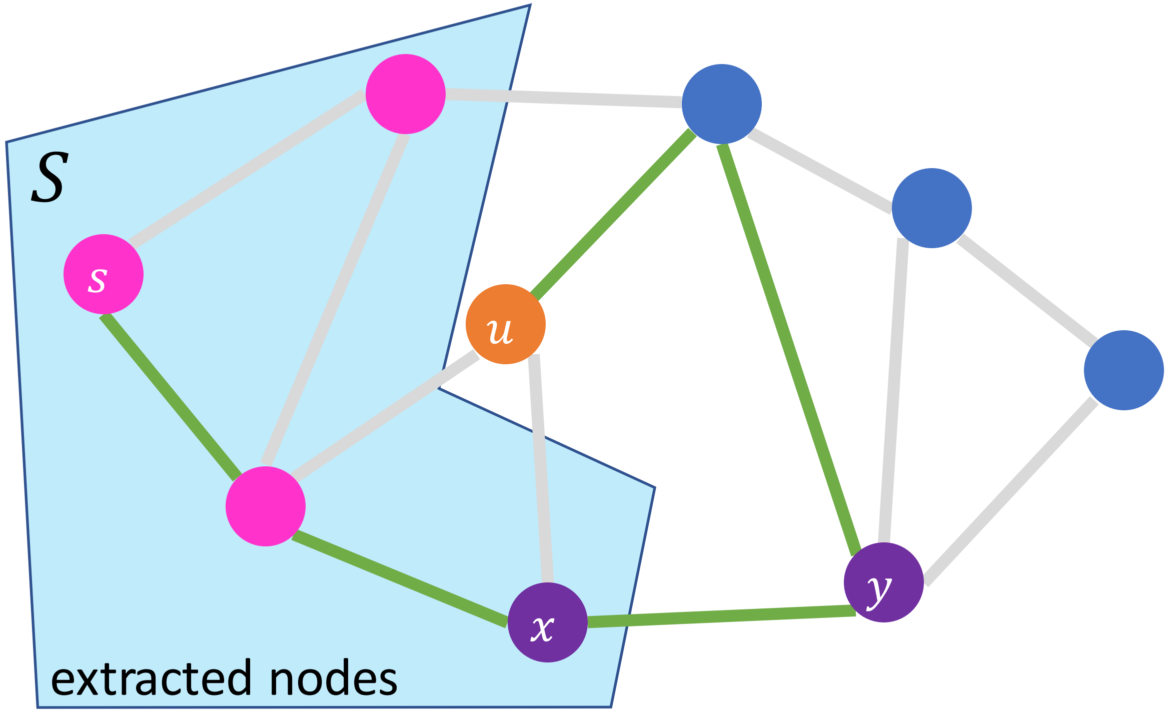
$$\begin{aligned} w(s, \ldots, u) &\geq \delta(s, x) + w(x, y) + w(y, \ldots, u) \\ &= d_x + w(x, y) + w(y, \ldots, u) \end{aligned}$$

Inductive hypothesis: since x was extracted before, $d_x = \delta(s, x)$

Correctness of Dijkstra's Algorithm: Claim 2

Let u be the $(i+1)^{\text{st}}$ node extracted

Claim 2: For every path $(s, \dots, u)$, $w(s, \dots, u) \geq d_u$



Extracted nodes "cuts" G into $(S, V - S)$

Take any path $(s, \dots, u)$

Since $u \notin S$, $(s, \dots, u)$ crosses the cut somewhere

- Let (x, y) be last edge in the path that crosses the cut

$$\begin{aligned} w(s, \dots, u) &\geq \delta(s, x) + w(x, y) + w(y, \dots, u) \\ &= d_x + w(x, y) + w(y, \dots, u) \\ &\geq d_y + w(y, \dots, u) \end{aligned}$$

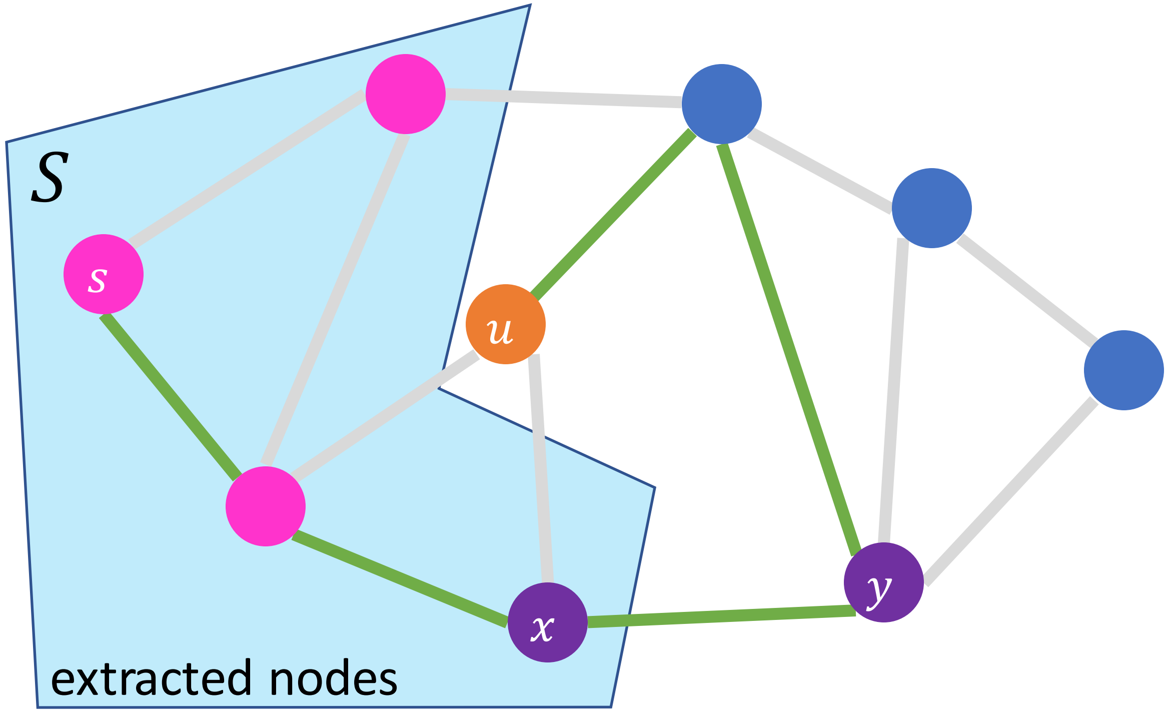
By construction of Dijkstra's algorithm, when x is extracted, d_y is updated to satisfy

$$d_y \leq d_x + w(x, y)$$

Correctness of Dijkstra's Algorithm: Claim 2

Let u be the $(i+1)^{\text{st}}$ node extracted

Claim 2: For every path $(s, \dots, u)$, $w(s, \dots, u) \geq d_u$



Extracted nodes "cuts" G into $(S, V - S)$

Take any path $(s, \dots, u)$

Since $u \notin S$, $(s, \dots, u)$ crosses the cut somewhere

- Let (x, y) be last edge in the path that crosses the cut

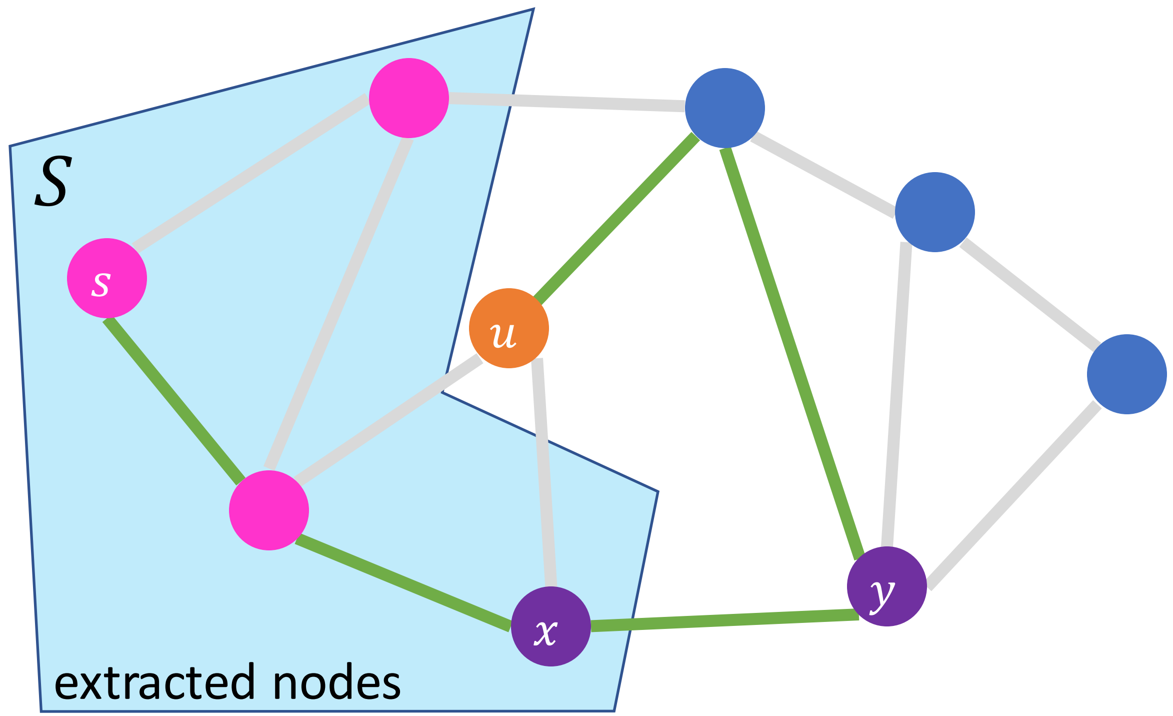
$$
\begin{aligned}
w(s, \dots, u) &\geq \delta(s, x) + w(x, y) + w(y, \dots, u) \\
&= d_x + w(x, y) + w(y, \dots, u) \\
&\geq d_y + w(y, \dots, u) \\
&\geq d_u + w(y, \dots, u)
\end{aligned}
$$

Greedy choice property: we always extract the node of minimal distance so $d_u \leq d_y$

Correctness of Dijkstra's Algorithm: Claim 2

Let u be the $(i+1)^{\text{st}}$ node extracted

Claim 2: For every path $(s, \dots, u)$, $w(s, \dots, u) \geq d_u$



Extracted nodes "cuts" G into $(S, V-S)$

Take any path $(s, \dots, u)$

Since $u \notin S$, $(s, \dots, u)$ crosses the cut somewhere

- Let (x, y) be last edge in the path that crosses the cut

$$
\begin{aligned}
w(s, \dots, u) &\geq \delta(s, x) + w(x, y) + w(y, \dots, u) \\
&= d_x + w(x, y) + w(y, \dots, u) \\
&\geq d_y + w(y, \dots, u) \\
&\geq d_u + w(y, \dots, u) \\
&\geq d_u
\end{aligned}
$$

All edge weights assumed to be positive

Correctness of Dijkstra's Algorithm

Conclusion: We used proof by induction to show:

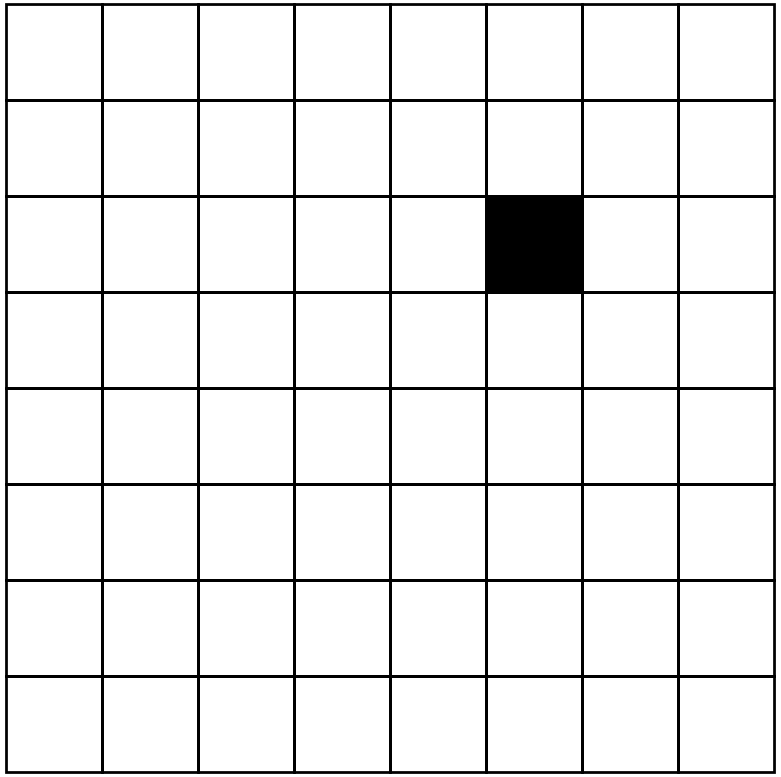
When node u is removed from the priority queue, $d_u = \delta(s, u)$

- **Claim 1:** There is a path of length d_u (as long as $d_u < \infty$) from s to u in G
- **Claim 2:** For every path $(s, \ldots, u)$, $w(s, \ldots, u) \geq d_u$

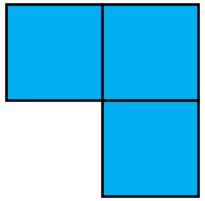
In other words, all paths $(s, \ldots, u)$ are no shorter than d_u
which makes it the shortest path (or one of equally shortest paths).

Divide and Conquer, Recurrences

Question



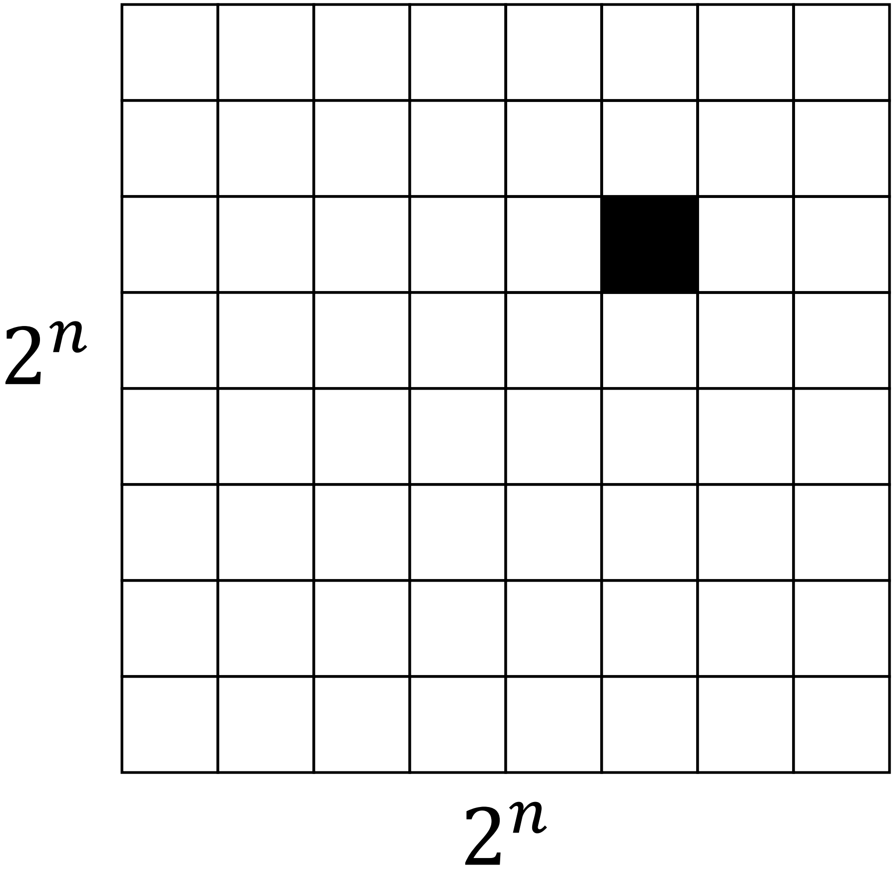
Can you cover an 8×8 grid with 1 square missing using "trominoes?"



Tromino

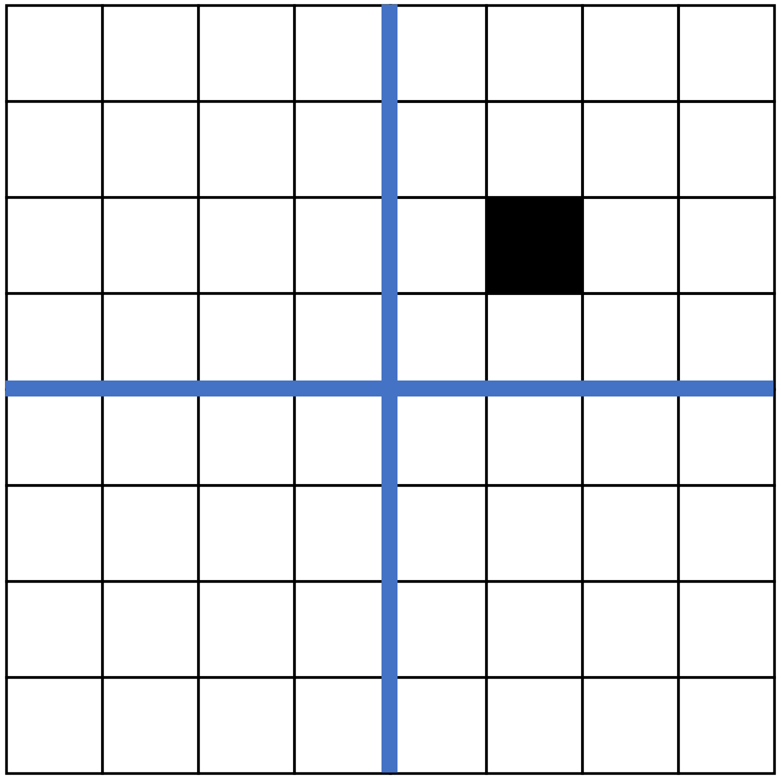
https://nstarr.people.amherst.edu/trom/puzzle-8by8/

Trominoes



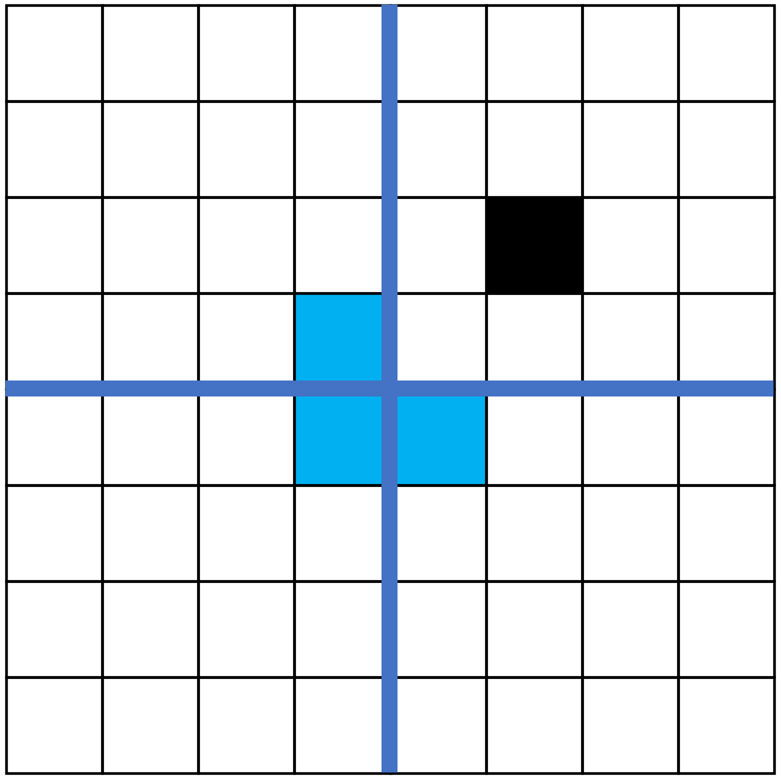
What about larger boards?

Trominoes Puzzle Solution



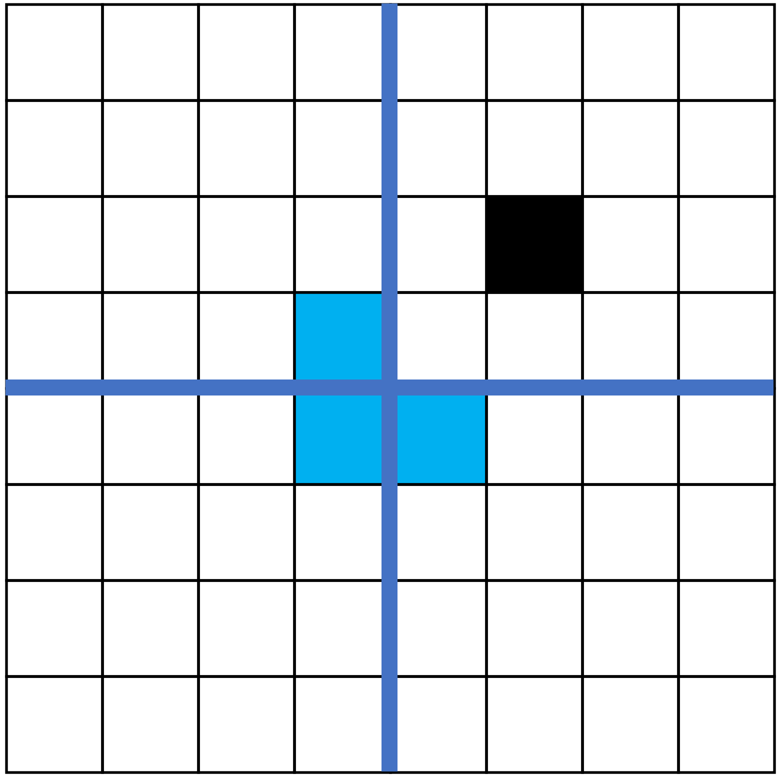
Divide the board into quadrants

Trominoes Puzzle Solution



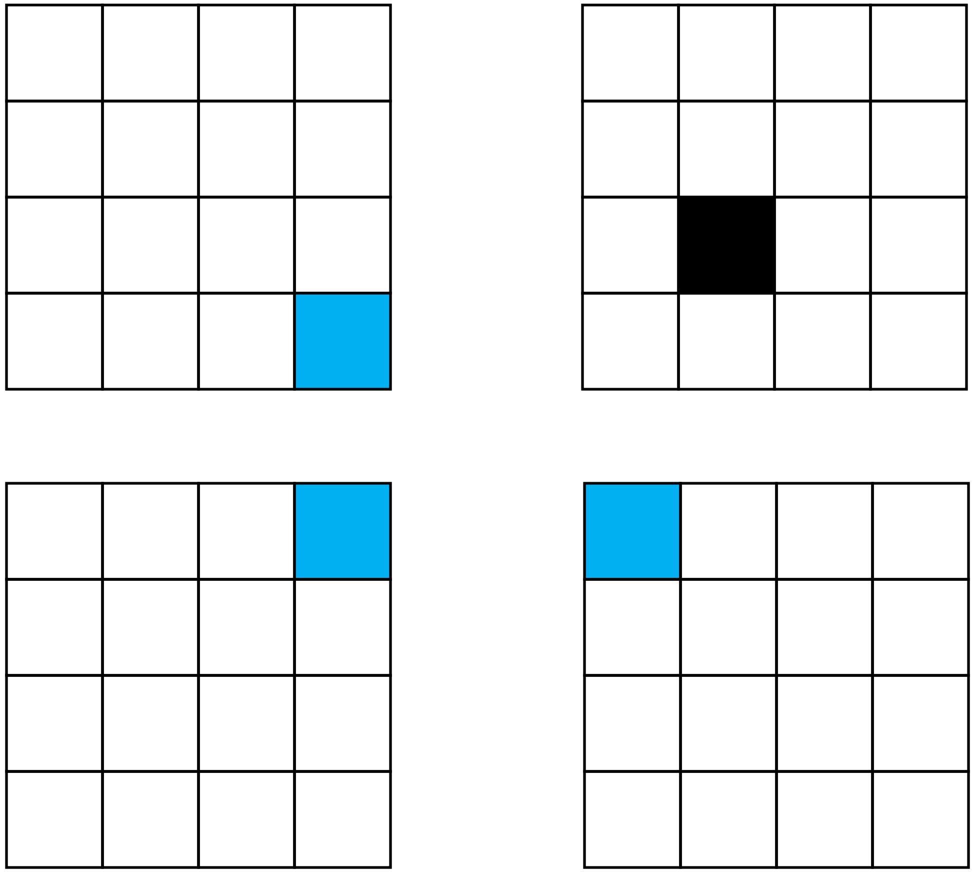
Place a tromino to occupy the three quadrants without the missing piece

Trominoes Puzzle Solution



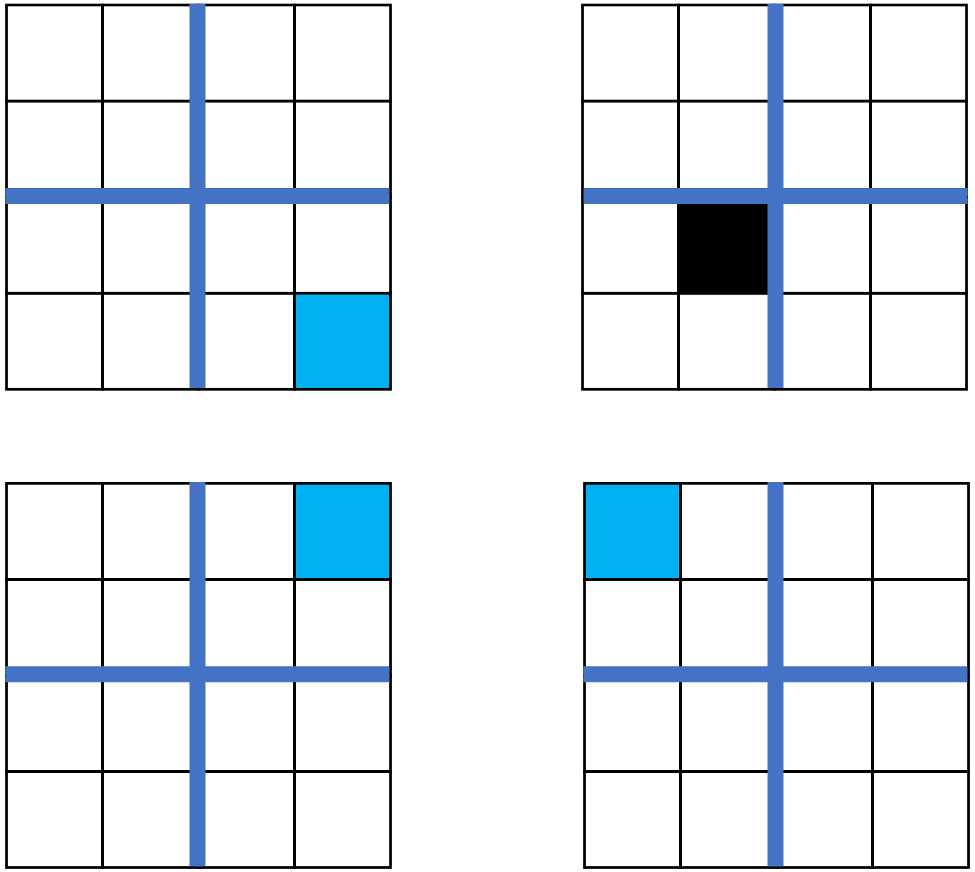
Place a tromino to occupy the three quadrants without the missing piece

Trominoes Puzzle Solution



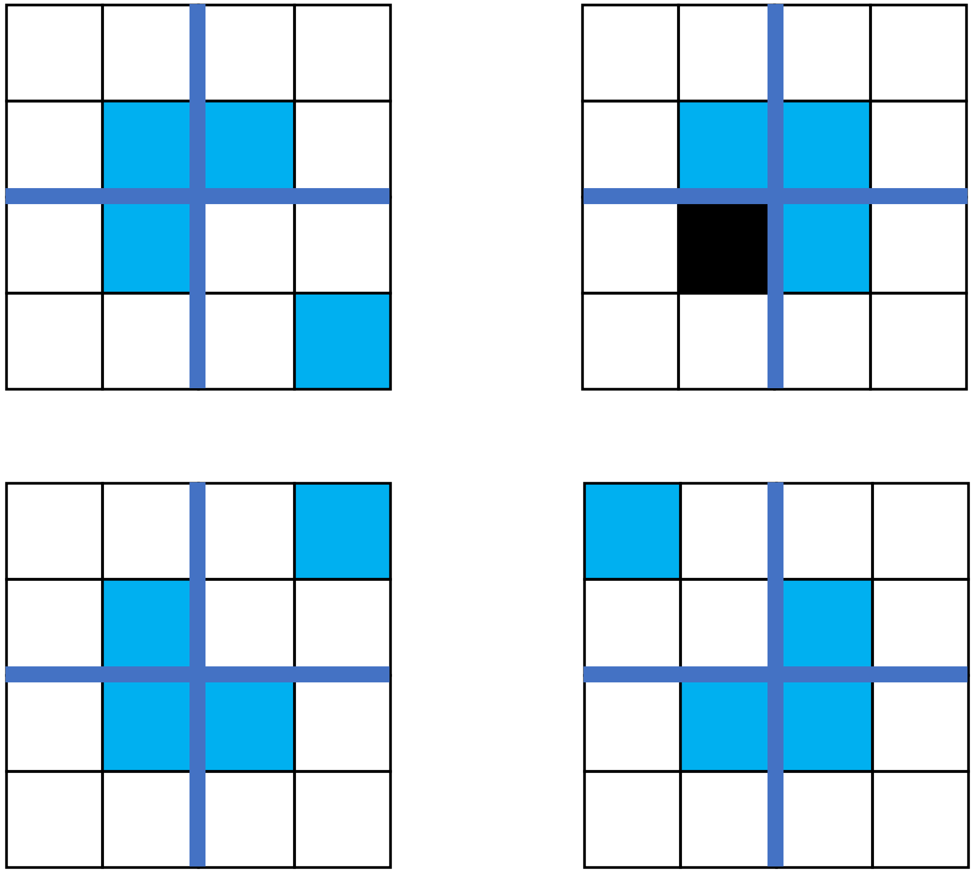
Observe: Each quadrant is now a smaller subproblem!

Trominoes Puzzle Solution



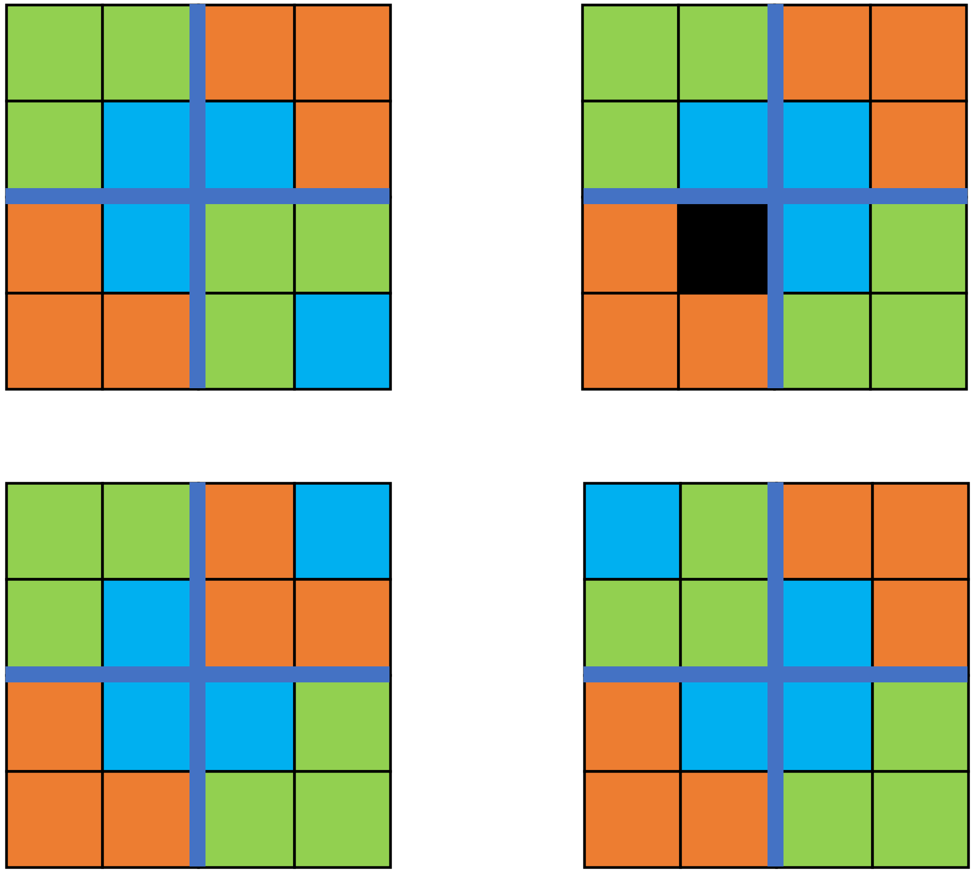
Solve **Recursively**

Trominoes Puzzle Solution



Solve **Recursively**

Trominoes Puzzle Solution



Our first algorithmic technique!

Divide and Conquer

[CLRS Chapter 4]

Divide:

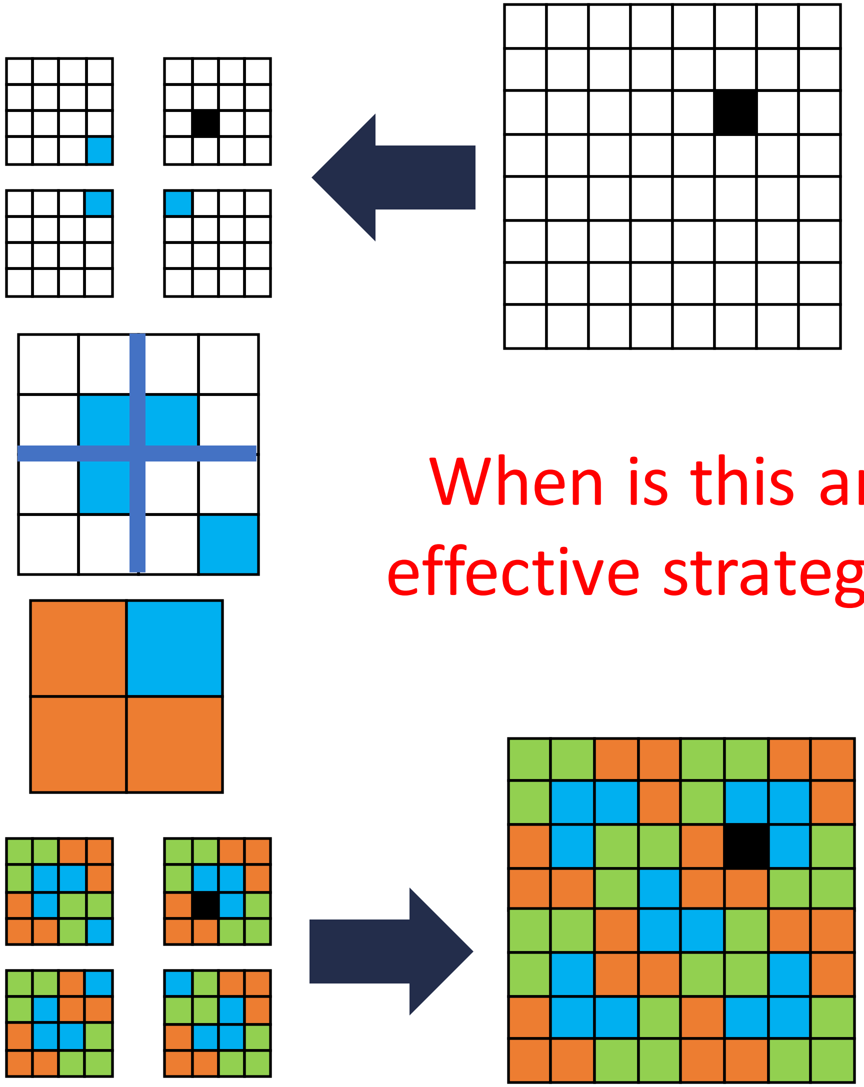
- Break the problem into multiple subproblems, each smaller instances of the original

Conquer:

- If the suproblems are "large":
  - Solve each subproblem recursively
- If the subproblems are "small":
  - Solve them directly (base case)

Combine:

- Merge solutions to subproblems to obtain solution for original problem



When is this an effective strategy?

Analyzing Divide and Conquer

1. Break into smaller subproblems
2. Use recurrence relation to express recursive running time
3. Use asymptotic notation to simplify

Divide: $D(n)$ time

Conquer: Recurse on smaller problems of size $s_1, \ldots, s_k$

Combine: $C(n)$ time

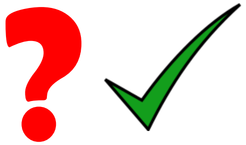
Recurrence:

- $T(n) = D(n) + \sum_{i \in [k]} T(s_i) + C(n)$

Recurrence Solving Techniques

Tree

get a picture of recursion



Guess/Check

guess and use induction to prove

"Cookbook"

MAGIC!

Substitution

substitute in to simplify

Merge Sort

Divide:

- Break n-element list into two lists of ${}^{n}/_{2}$ elements

Conquer:

- If $n > 1$:
  - Sort each sublist recursively
- If $n = 1$:
  - List is already sorted (base case)

Combine:

- Merge together sorted sublists into one sorted list

Merge

Combine: Merge sorted sublists into one sorted list

Inputs:

- 2 sorted lists (L_1, L_2)
- 1 output list (L_{out})

While (L_1 and L_2 not empty):

    If $L_1[0] \leq L_2[0]$:

        L_{out}.append(L_1.pop())

    Else:

        L_{out}.append(L_2.pop())

L_{out}.append(L_1)

L_{out}.append(L_2)

Analyzing Merge Sort

1. Break into smaller subproblems
2. Use recurrence relation to express recursive running time
3. Use asymptotic notation to simplify

Divide: 0 comparisons

Conquer: recurse on 2 small problems, size $\frac{n}{2}$

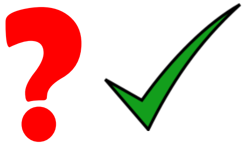
Combine: n comparisons

Recurrence:

- $T(n) = 2T(n/2) + n$

Recurrence Solving Techniques

Tree



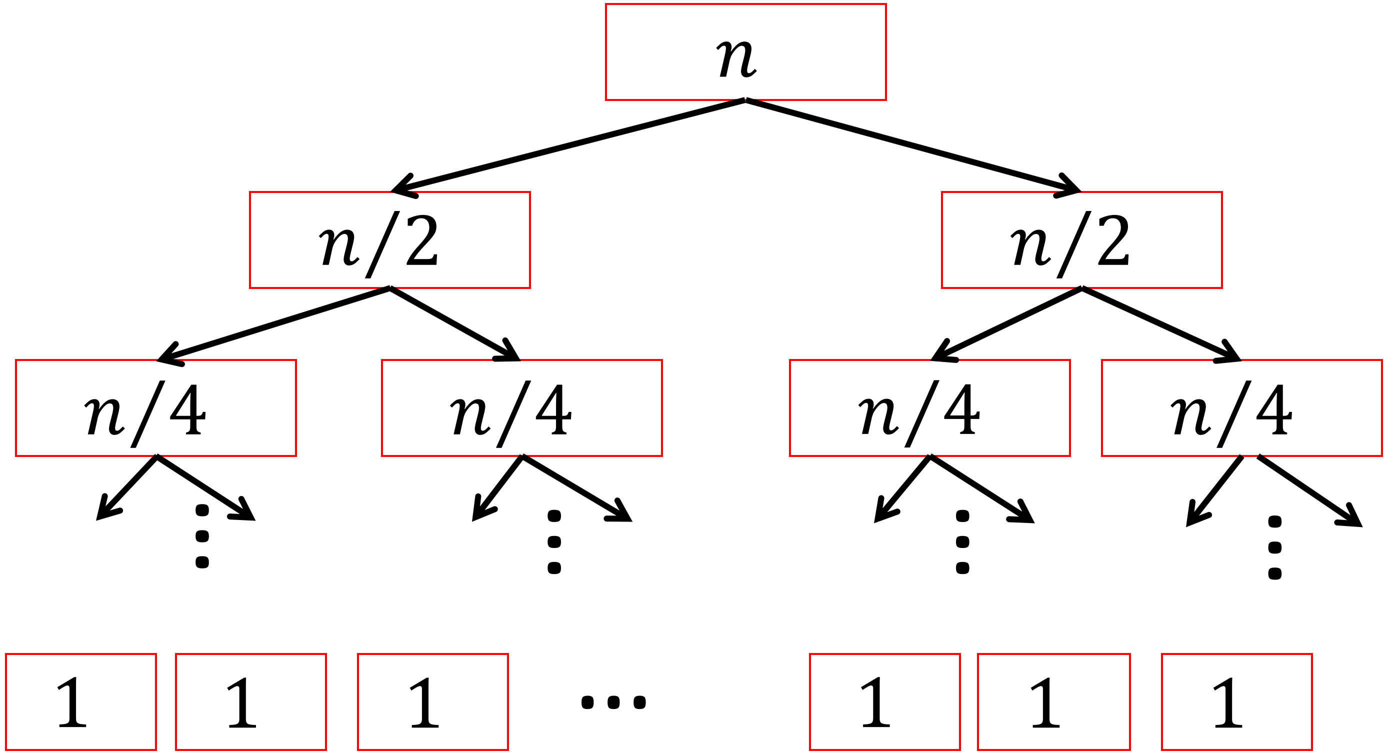
Guess/Check

“Cookbook”

Substitution

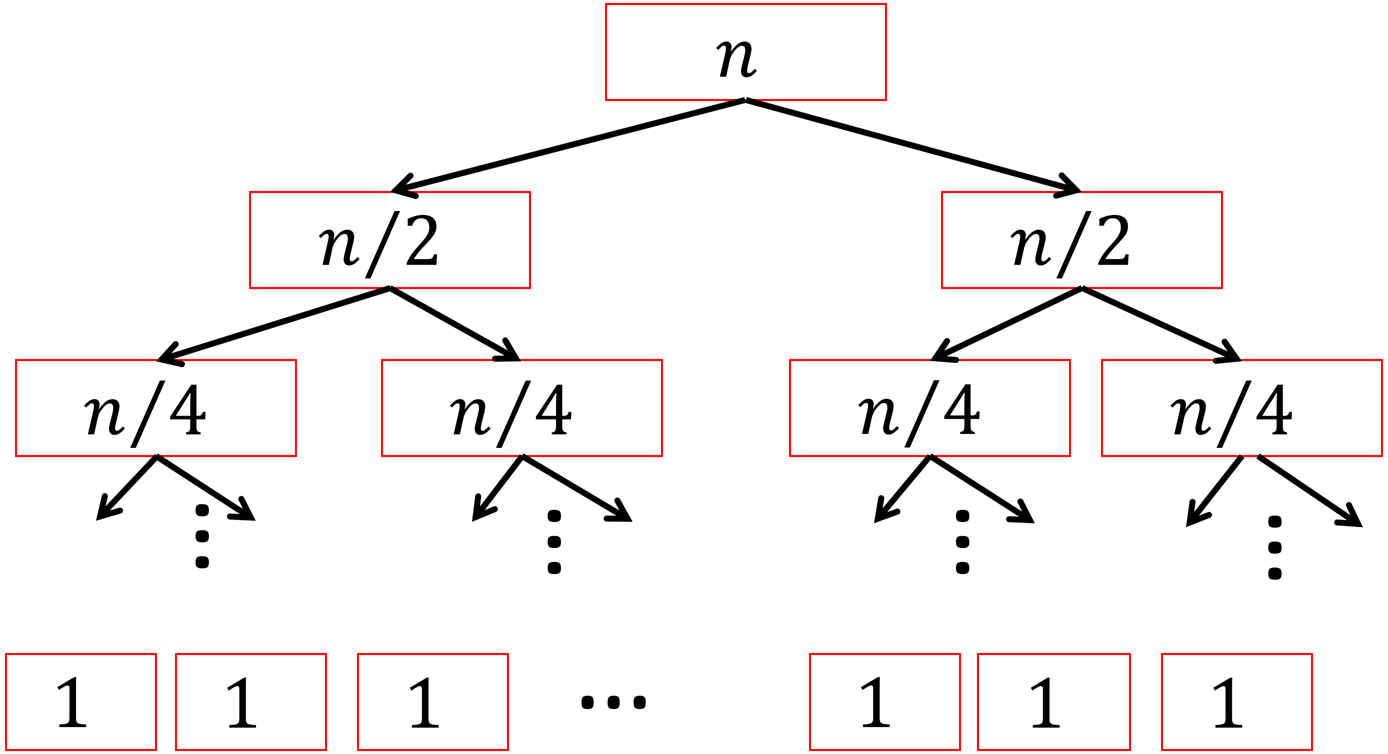
Tree Method

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$



Tree Method

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

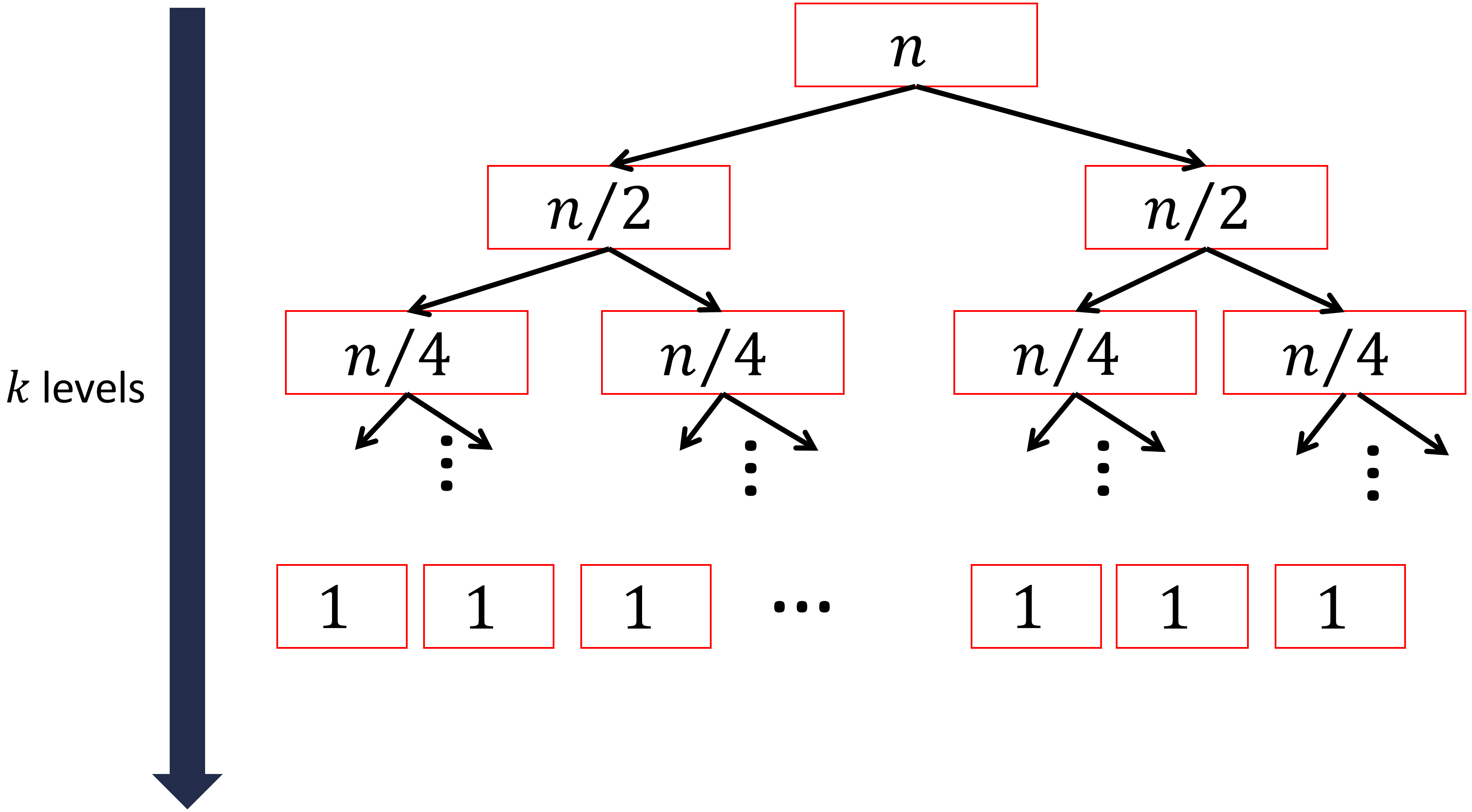


Number of subproblems

1

Tree Method

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$



Number of subproblems	Cost of subproblem
1	n
2	$n/2$
4	$n/4$
2^k	$\frac{n}{2^k} = 1$

Tree Method

3. Use asymptotic notation to simplify

$$T(n) = 2T(n/2) + n$$

How many levels?

Problem size at k^{th} level: $\frac{n}{2^k}$

Base case: $n = 1$

At level k, it should be the case that $\frac{n}{2^k} = 1$

$$n = 2^k \Rightarrow k = \log_2 n$$

Number of subproblems	Cost of subproblem
1	n
2	$n/2$
4	$n/4$
2^k	$\frac{n}{2^k} = 1$

Tree Method

3. Use asymptotic notation to simplify

$$T(n) = 2T(n/2) + n$$

Number of subproblems	Cost of subproblem
1	n
2	$n/2$
4	$n/4$
2^k	$\frac{n}{2^k} = 1$

$$k = \log_2 n$$

What is the cost?

Cost at level i: $2^i \cdot \frac{n}{2^i} = n$

Total cost: $T(n) = \sum_{i=0}^{\log_2 n} n = n \sum_{i=0}^{\log_2 n} 1 = n \log_2 n$

$$= \Theta(n \log n)$$

Multiplication

Want to multiply large numbers together

$$\begin{array}{r} 4\,1\,0\,2 \\ \times\;\; 1\,8\,1\,9 \\ \hline \end{array}$$

n-digit numbers

How do we measure input size?

number of digits

What do we "count" for run time?

number of <u>elementary</u> operations (single-digit multiplications)

“Schoolbook” Multiplication

How many multiplications?

$$
\begin{array}{r}
4\,1\,0\,2 \\
\times\ \boxed{1}\,\boxed{8}\,\boxed{1}\,\boxed{9} \\
\hline
3\,6\,9\,1\,8 \\
4\,1\,0\,2 \\
3\,2\,8\,1\,6 \\
+\ 4\,1\,0\,2 \\
\hline
7\,4\,6\,1\,5\,3\,8
\end{array}
$$

n-digit numbers

n mults

n mults

n mults

n mults

n levels $\Rightarrow \Theta(n^2)$

What about cost of additions?

$\Theta(n^2)$

"Schoolbook" Multiplication

Can we do better?

How many multiplications?

$$
\begin{array}{r}
4\,1\,0\,2 \\
\times\;\; 1\,8\,1\,9 \\
\hline
3\,6\,9\,1\,8 \\
4\,1\,0\,2\;\; \\
3\,2\,8\,1\,6\;\;\;\; \\
+\;\; 4\,1\,0\,2\;\;\;\;\;\; \\
\hline
7\,4\,6\,1\,5\,3\,8
\end{array}
$$

n-digit numbers

$$\left.\begin{array}{l} n \text{ mults} \\ n \text{ mults} \\ n \text{ mults} \\ n \text{ mults} \end{array}\right\} n \text{ levels}$$

$\Rightarrow \Theta(n^2)$

What about cost of additions?

$\Theta(n^2)$

Divide and Conquer Multiplication

1. Break into smaller subproblems

$$\begin{array}{r} \boxed{a}\boxed{b} \\ \times\ \boxed{c}\boxed{d} \\ \hline \end{array} \quad \begin{array}{l} = 10^{\frac{n}{2}}\boxed{a} + \boxed{b} \\ = 10^{\frac{n}{2}}\boxed{c} + \boxed{d} \end{array}$$

$$= \quad 10^{n}(\boxed{a} \times \boxed{c}) +$$

$$10^{\frac{n}{2}}(\boxed{a} \times \boxed{d} + \boxed{b} \times \boxed{c}) +$$

$$(\boxed{b} \times \boxed{d})$$

Divide and Conquer Multiplication

Divide:

- Break n-digit numbers into four numbers of $n/2$ digits each (call them a, b, c, d)

Conquer:

- If $n > 1$:
  - Recursively compute ac, ad, bc, bd
- If $n = 1$: (i.e. one digit each)
  - Compute ac, ad, bc, bd directly (base case)

Combine:

- $10^n(ac) + 10^{n/2}(ad + bc) + bd$

For simplicity, assume that $n = 2^k$ is a power of 2

Divide and Conquer Multiplication

2. Use recurrence relation to express recursive running time

$$10^n(ac) + 10^{n/2}(ad + bc) + bd$$

Recursively solve

$$T(n)$$

Divide and Conquer Multiplication

2. Use recurrence relation to express recursive running time

$$10^n(ac) + 10^{n/2}(ad + bc) + bd$$

Recursively solve

$$T(n) = 4T\left(\frac{n}{2}\right)$$

Need to compute 4 multiplications, each of size $n/2$

Divide and Conquer Multiplication

2. Use recurrence relation to express recursive running time

$$10^n(ac) + 10^{n/2}(ad + bc) + bd$$

Recursively solve

$$T(n) = 4T\left(\frac{n}{2}\right) + 5n$$

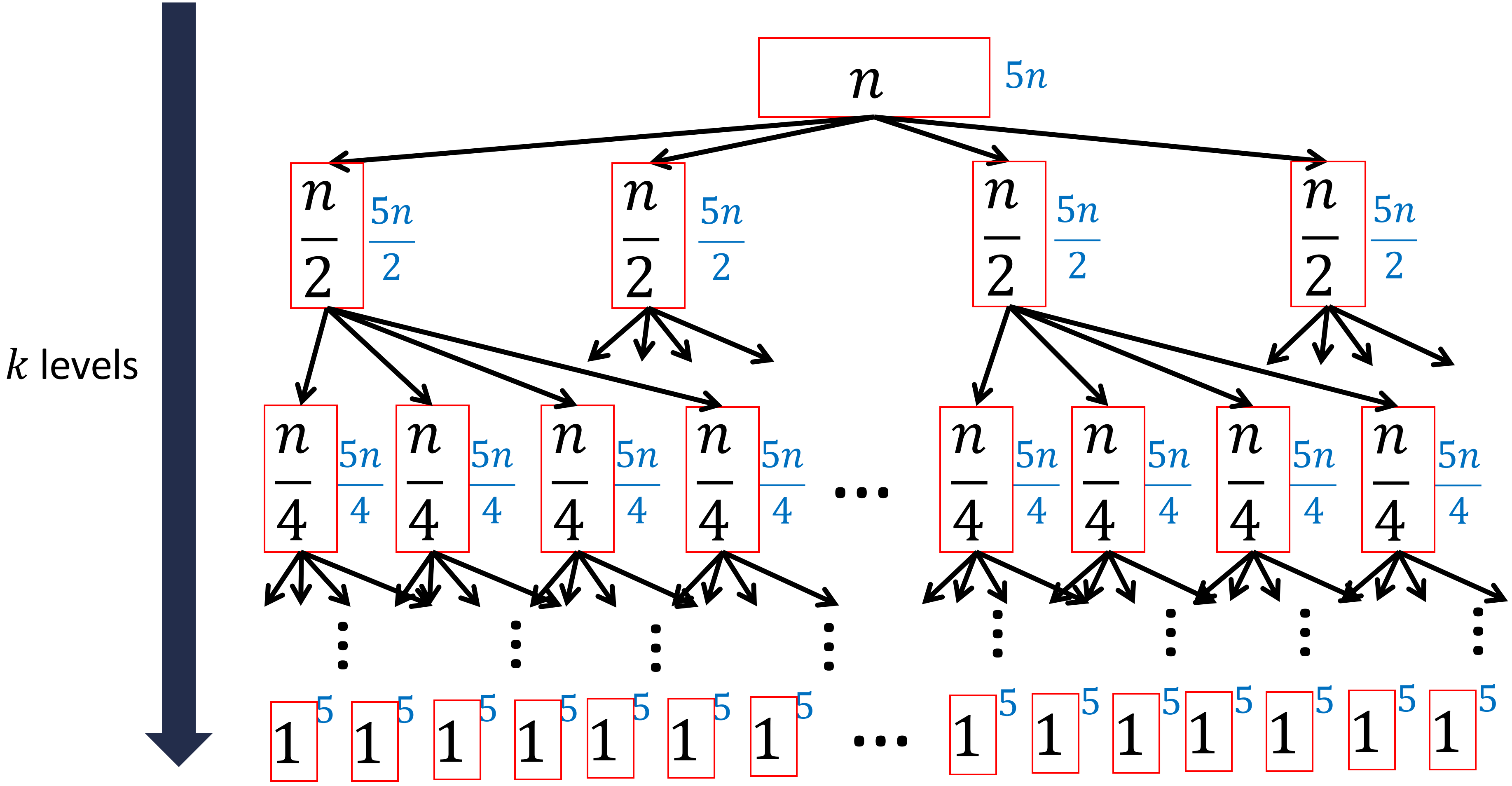
Need to compute 4 multiplications, each of size $n/2$

2 shifts and 3 additions on n-bit values

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$T(n) = 4T(n/2) + 5n$$



Number of subproblems	Cost of subproblem
1	$5n$
4	$\frac{5n}{2}$
16	$\frac{5n}{4}$
4^k	$\frac{5n}{2^k} = 5$

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$T(n) = 4T(n/2) + 5n$$

How many levels?

Problem size at k^{th} level: $\frac{n}{2^k}$

Base case: $n = 1$

At level k, it should be the case that $\frac{n}{2^k} = 1$

$$n = 2^k \Rightarrow k = \log_2 n$$

Number of subproblems	Cost of subproblem
1	$5n$
4	$\frac{5n}{2}$
16	$\frac{5n}{4}$
4^k	$\frac{5n}{2^k} = 5$

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$T(n) = 4T(n/2) + 5n$$

$$k = \log_2 n$$

What is the cost?

Cost at level i: $4^i \cdot \frac{5n}{2^i} = 2^i \cdot 5n$

Total cost: $T(n) = \sum_{i=0}^{\log_2 n} 2^i \cdot 5n = 5n \sum_{i=0}^{\log_2 n} 2^i$

Number of subproblems	Cost of subproblem
1	$5n$
4	$\frac{5n}{2}$
16	$\frac{5n}{4}$
4^k	$\frac{5n}{2^k} = 5$

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$\sum_{i=0}^{L} a^i = \frac{a^{L+1} - 1}{a - 1}$$

$$T(n) = 4T(n/2) + 5n$$

$$= 5n \sum_{i=0}^{\log_2 n} 2^i$$

$$= 5n \cdot \frac{2^{\log_2 n + 1} - 1}{2 - 1}$$

$$= 5n(2n - 1) = \Theta(n^2)$$

No better than the schoolbook method!

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$\sum_{i=0}^{L} a^i = \frac{a^{L+1} - 1}{a - 1}$$

$$T(n) = 4T(n/2) + 5n$$

$$= 5n \sum_{i=0}^{\log_2 n} 2^i$$

$$= 5n \cdot \frac{2^{\log_2 n+1} - 1}{2 - 1}$$

$$= 5n(2n - 1) = \Theta(n^2)$$

Is there a $o(n^2)$ algorithm for multiplication?