COSC 520 Project
Breaking the Sorting Barrier

Hamza Muhammad Anwar

Bakdauren Narbayev

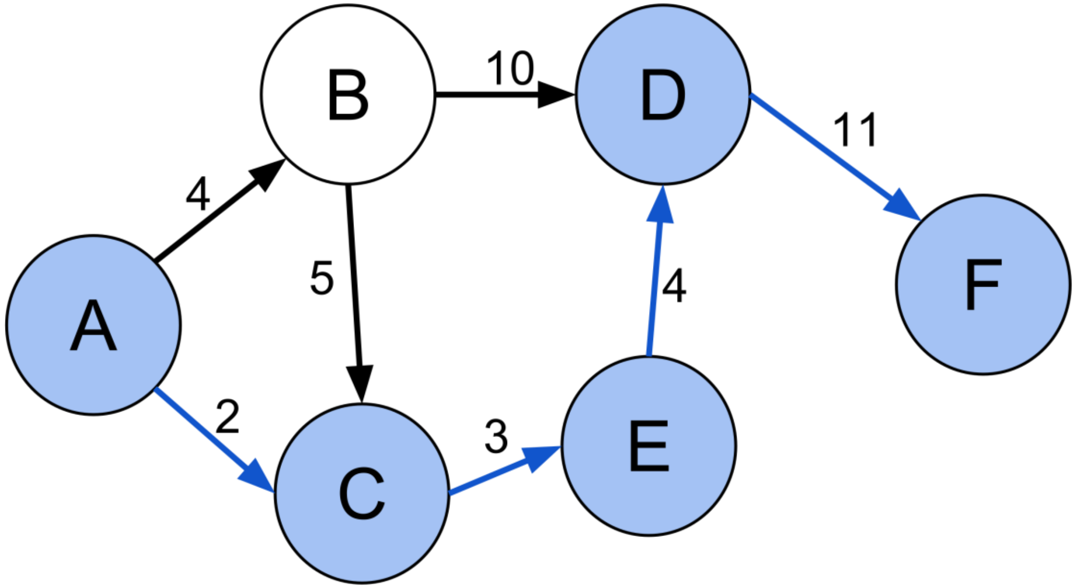
Ghaith Chrit

Overview

- Chose and implemented a new algorithm: BMSSP
- Benchmarked it with other similar algorithms
- Made a GUI to visualize their performance

Shortest Path Problem

- Finding the path between two vertices with minimum weights
- Directed, undirected, mixed



Motivation

- BMSSP: Do theoretical guarantees materialize during empirical comparisons?
- Beat a 66-year-old record
- Award-winning algorithm

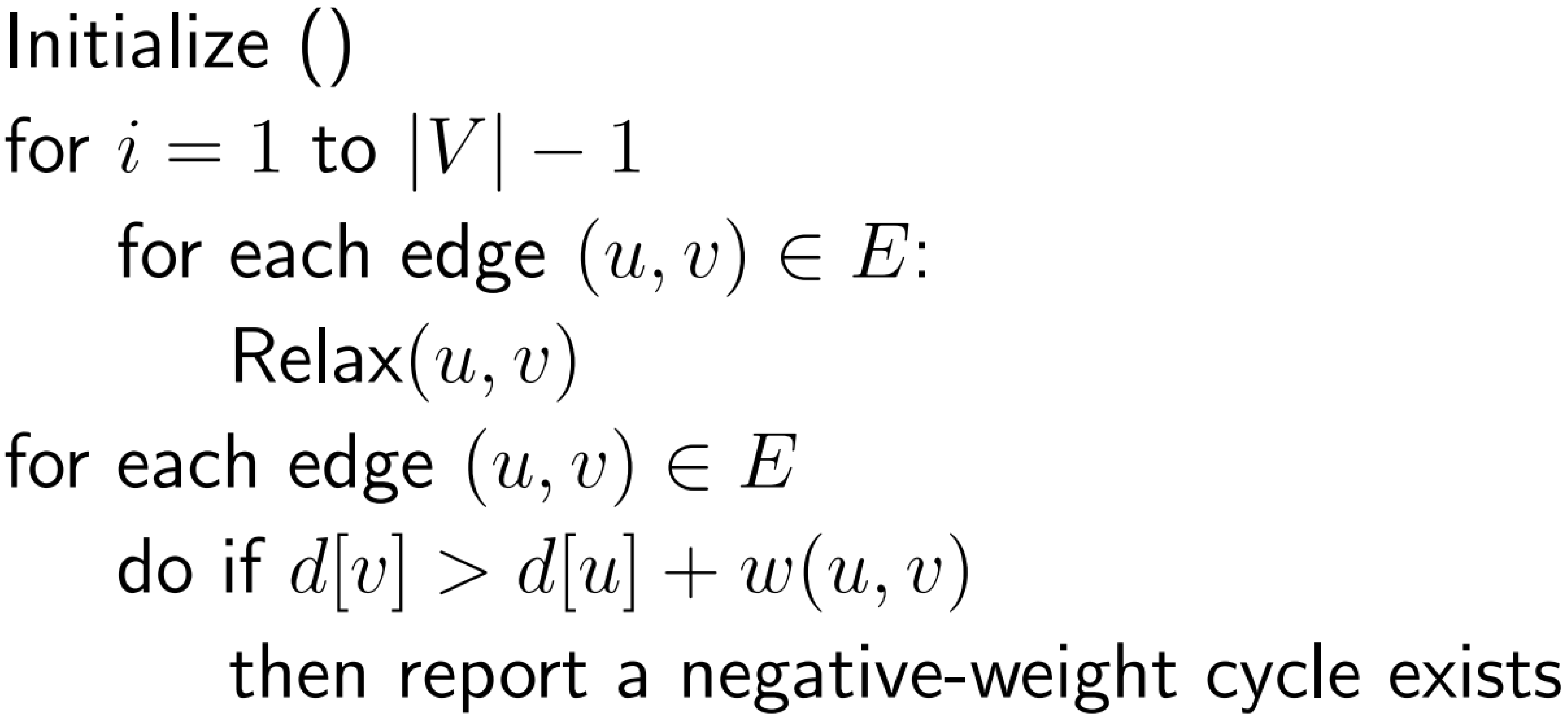
Algorithms

- Bellman-Ford (Bellman, 1958)
- Dijkstra (Dijkstra, 1959)
- BMSSP (Duan et al. 2025)

Bellman-Ford Algorithm

- Slower than Dijkstra
- Can work with negative-edge weights
- Max |V|-1 edges in the path (No cycles)
- Time complexity: O(VE)
- Space complexity: O(V)

Bellman-Ford Algorithm Pseudocode



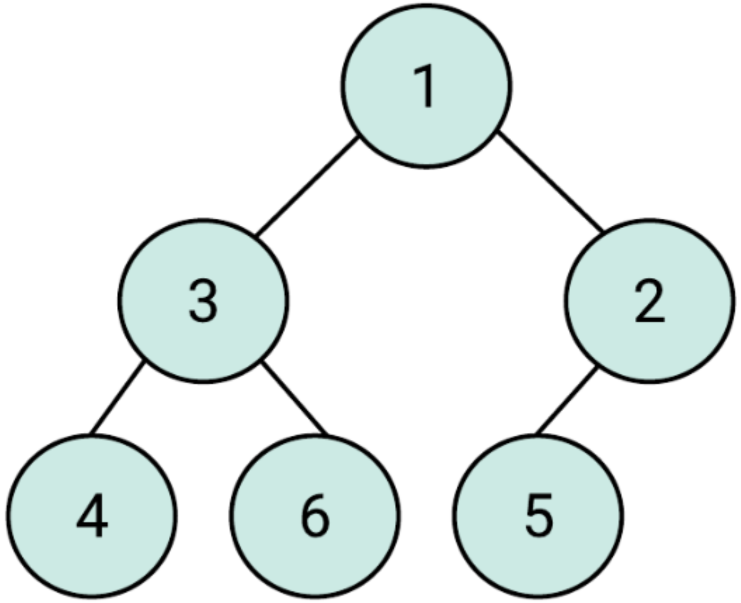
```
Initialize ()
for $i = 1$ to $|V| - 1$
    for each edge $(u, v) \in E$:
        Relax$(u, v)$
for each edge $(u, v) \in E$
    do if $d[v] > d[u] + w(u, v)$
        then report a negative-weight cycle exists
```

Dijkstra's Algorithm

- Cannot handle negative-edge weights
- Greedy
- Time complexity: O((V+E) log V)
- Space complexity: O(V)

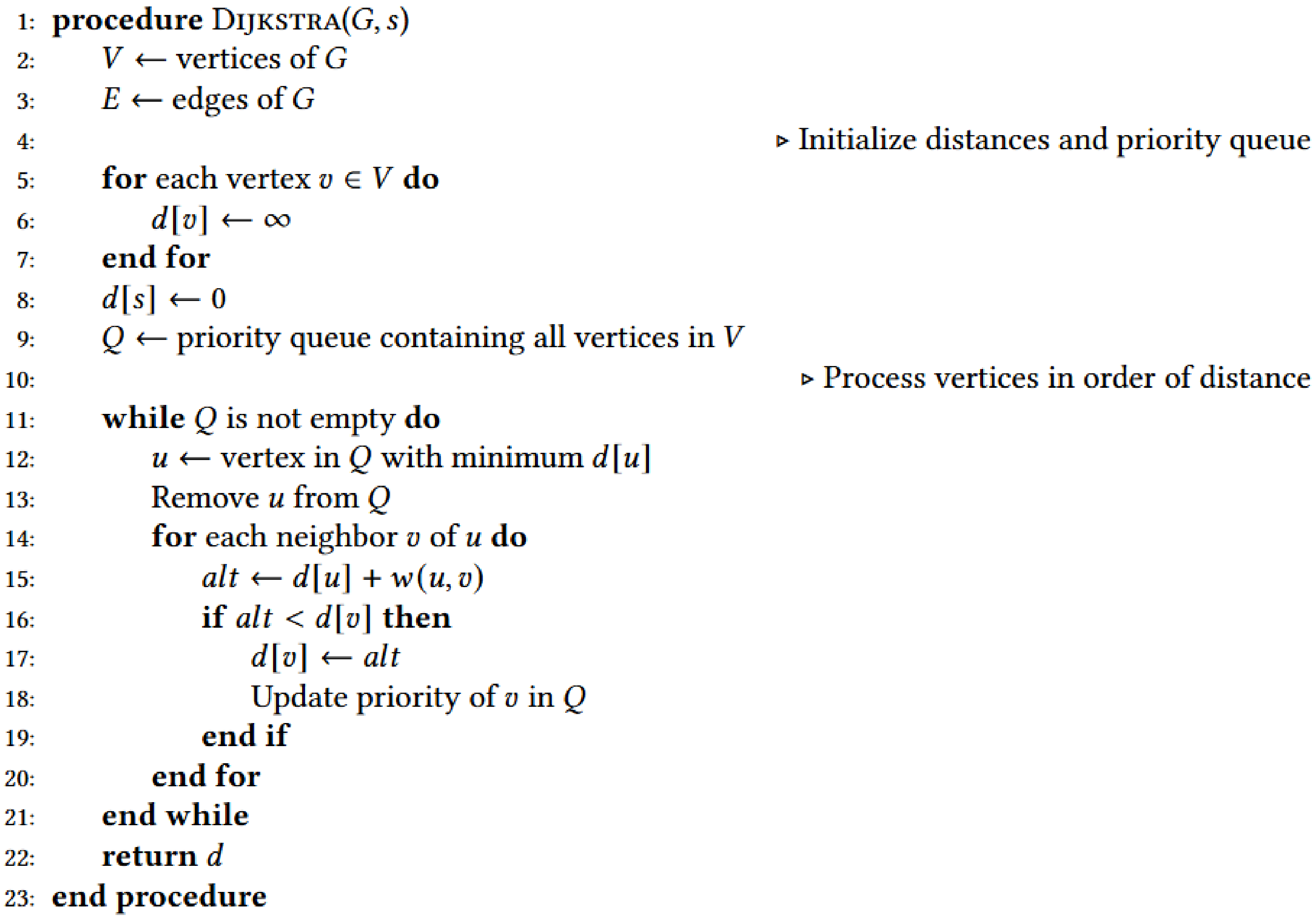
Dijkstra's Algorithm – Min Heap

- Priority queue
- Root node contains the smallest element
- All nodes contain elements less than or equal to their child nodes



Min heap

Dijkstra's Algorithm Pseudocode



```
procedure DIJKSTRA($G, s$)
    $V \leftarrow$ vertices of $G$
    $E \leftarrow$ edges of $G$
                                          ▷ Initialize distances and priority queue
    for each vertex $v \in V$ do
        $d[v] \leftarrow \infty$
    end for
    $d[s] \leftarrow 0$
    $Q \leftarrow$ priority queue containing all vertices in $V$
                                          ▷ Process vertices in order of distance
    while $Q$ is not empty do
        $u \leftarrow$ vertex in $Q$ with minimum $d[u]$
        Remove $u$ from $Q$
        for each neighbor $v$ of $u$ do
            $alt \leftarrow d[u] + w(u, v)$
            if $alt < d[v]$ then
                $d[v] \leftarrow alt$
                Update priority of $v$ in $Q$
            end if
        end for
    end while
    return $d$
end procedure
```

Applications

- Network Routing
- Transportation and GIS
- Pathfinding for Autonomous Robots

BMSSP

Dijkstra's algorithm is still the fastest if the distances need to be obtained in increasing order!

- Sparse directed graph with non-negative weights
- Time complexity: $O(E \log^{2/3} V)$

Beats Dijkstra's algorithm!

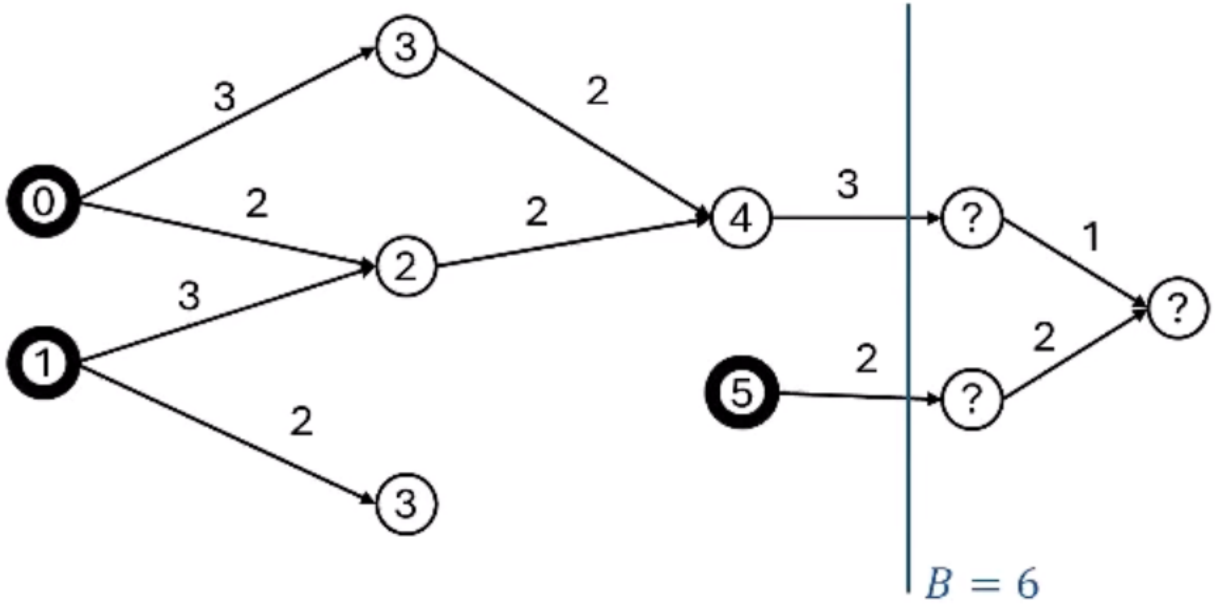
- Space complexity: $\Omega(V \log V^{1/3})$

Bounded Multi-Source Shortest Path (BMSSP)

BMSSP Problem

Given a set *S* of sources with initial distances, and an upper bound *B*, for every vertex *x*, suppose the shortest path from any vertex in *S* to *x* is d[x].

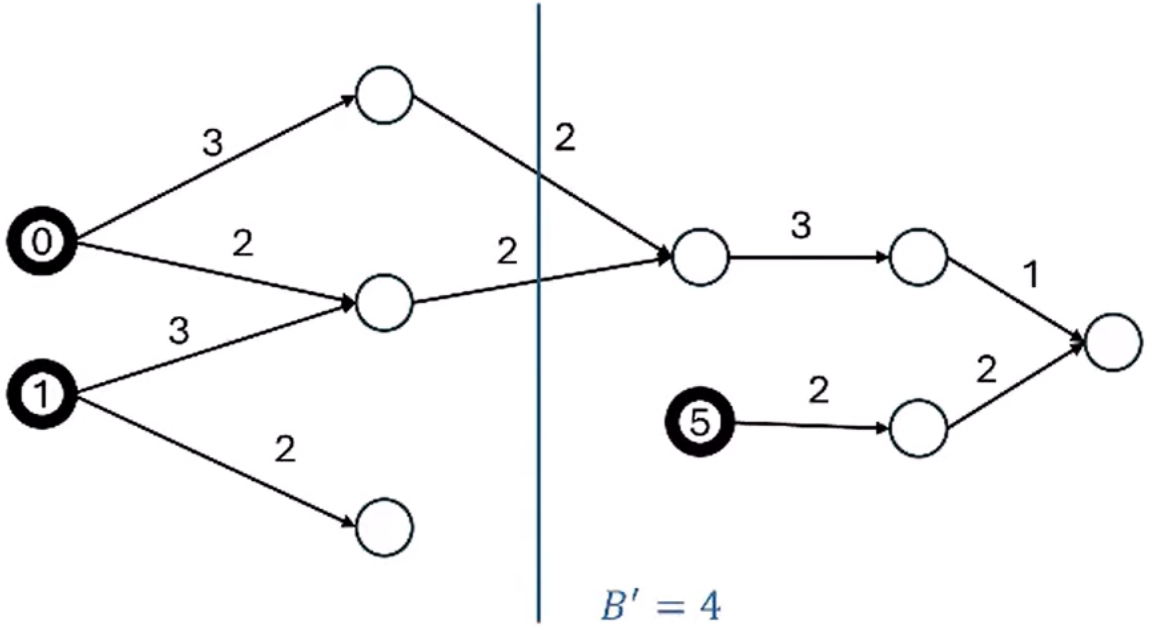
The algorithm reports whether d[x] < *B*, and if d[x] < *B*, the algorithm finds d[x].[1]



[1] https://www.youtube.com/watch?v=LzvvcadKbd0

Divide and Conquer

- Suppose we're given $B' \leq B$ such that exactly half of the vertices satisfy d[x] < B'.
- We can divide the problem into 2 equal halves:
  1. vertices with d[x] < B'
  2. vertices with $B' \leq$ d[x] < B

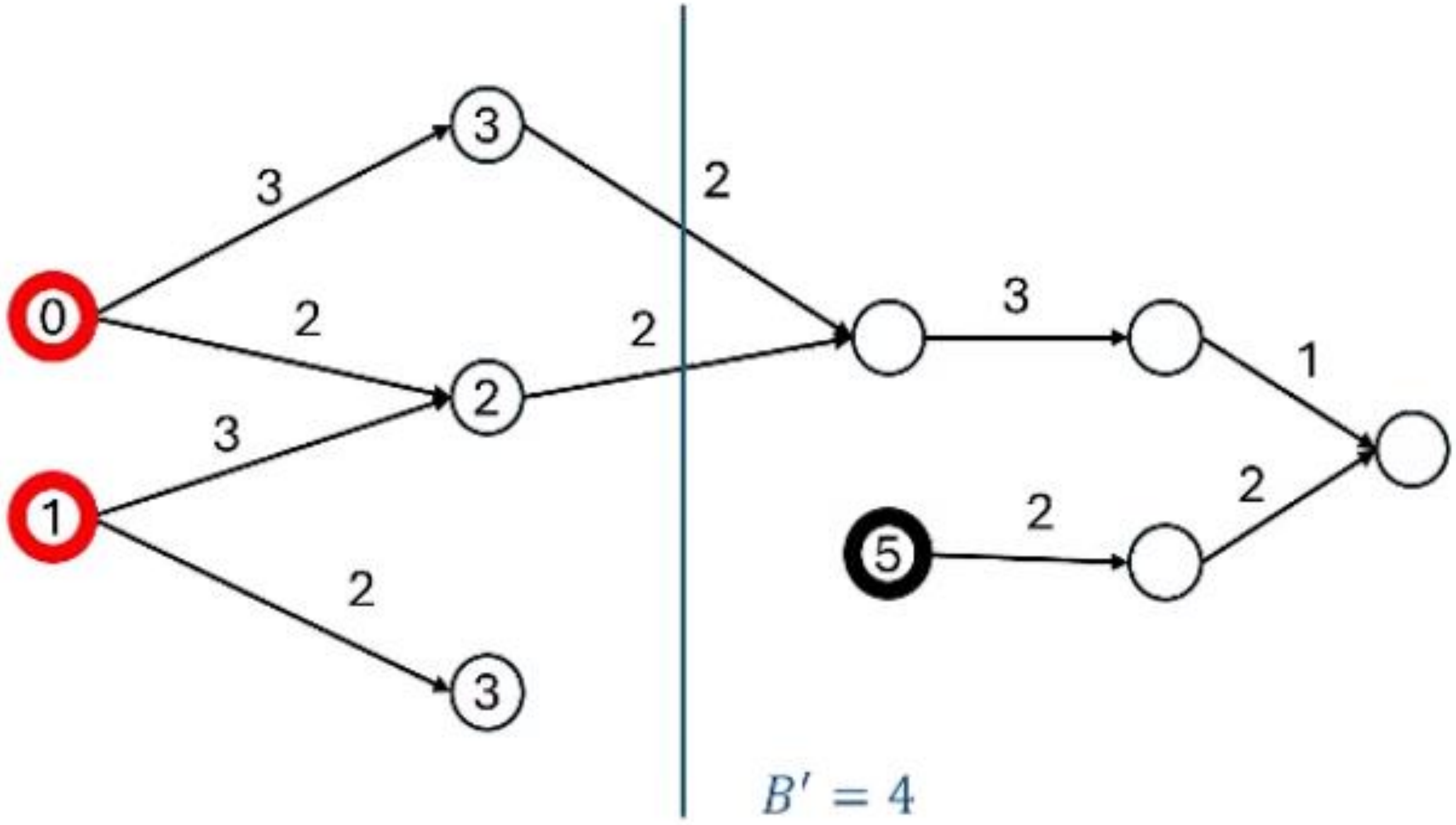


BMSSP Problem

Given a set S of sources with initial distances, and an upper bound B, for every vertex x, suppose the shortest path from any vertex in S to x is d[x].

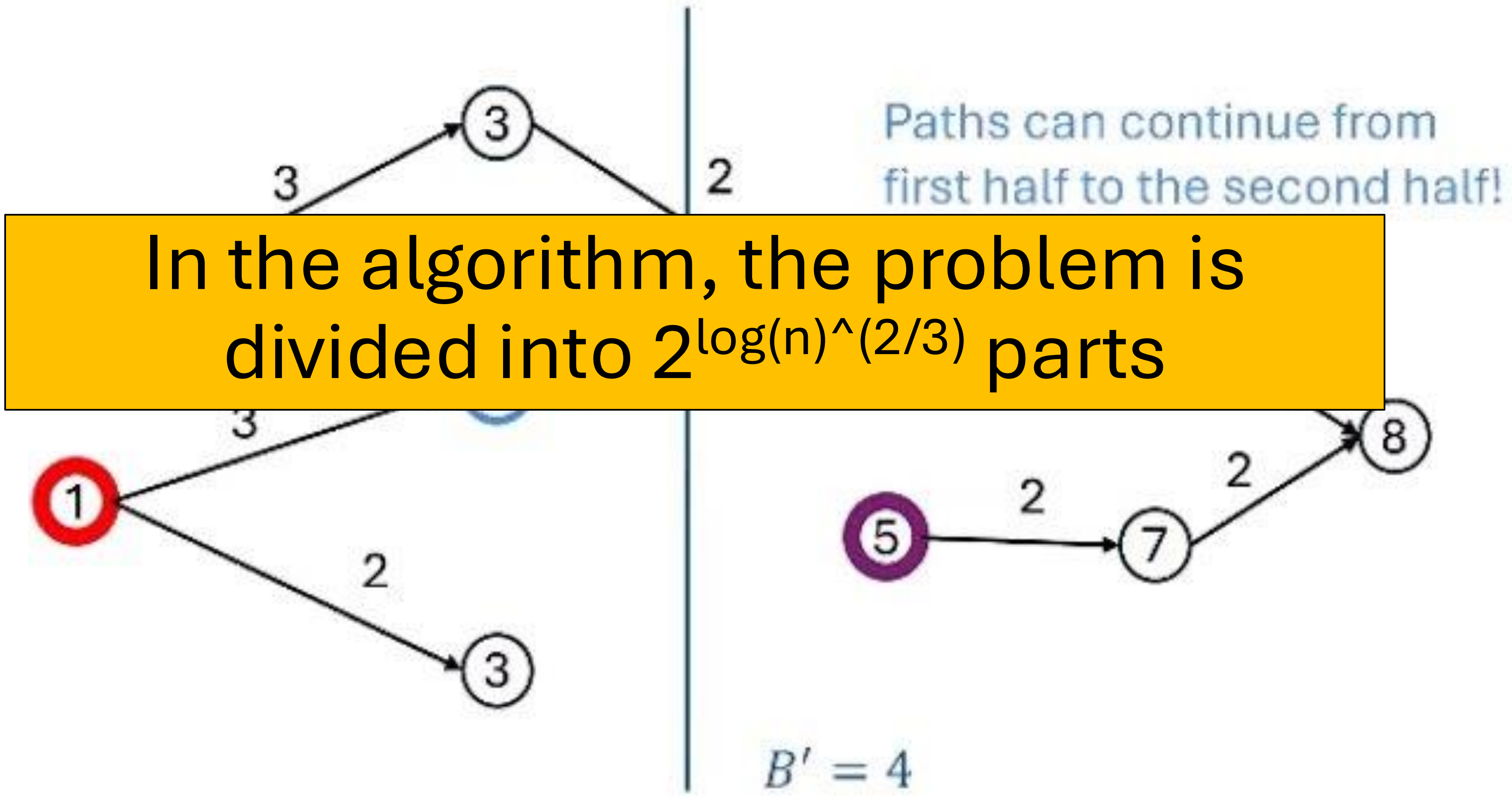
The algorithm reports whether d[x] < B, and if d[x] < B, the algorithm finds d[x].

Divide and Conquer (1st half)



Divide and Conquer (2nd half)

In the algorithm, the problem is divided into $2^{\log(n)\hat{}(2/3)}$ parts

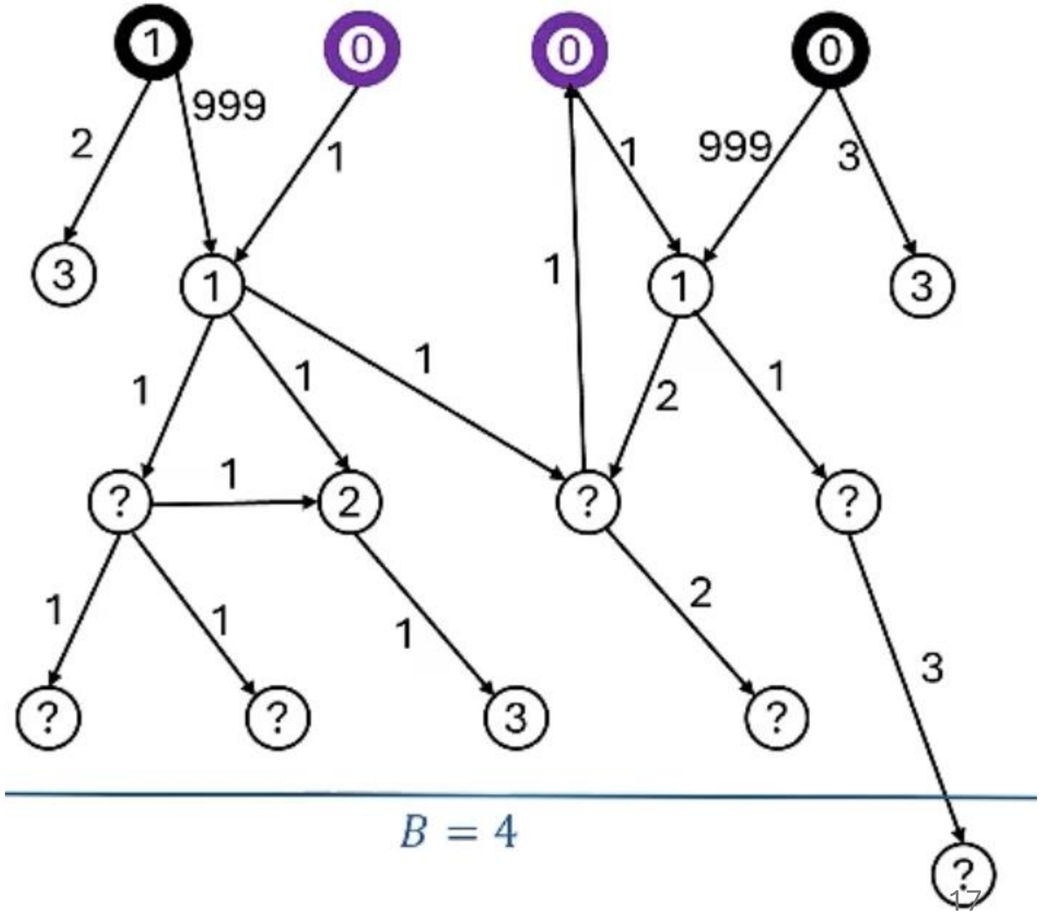


Paths can continue from first half to the second half!

$B' = 4$

Pivot Pruning

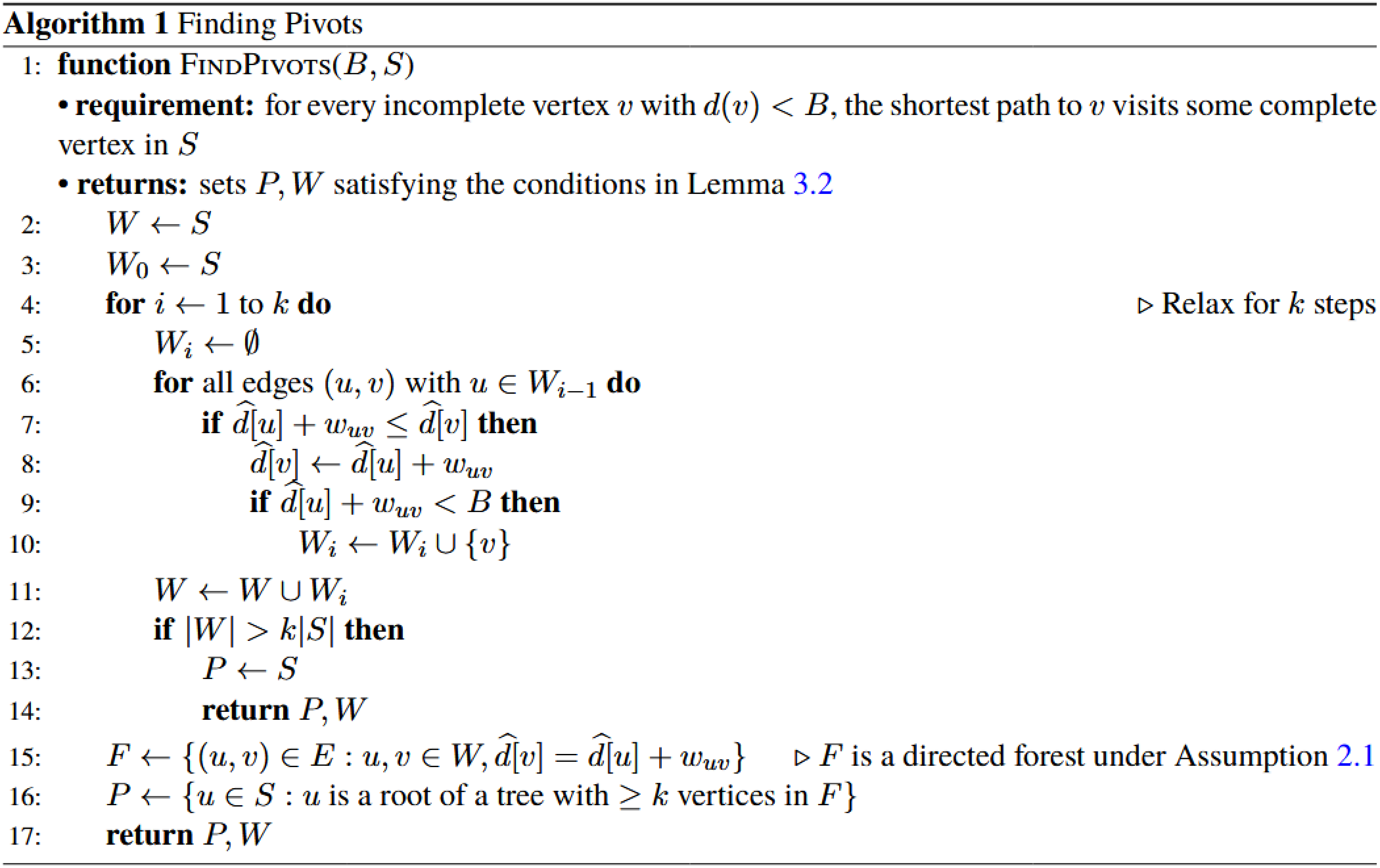
- The set of source vertices might contain all the vertices... Too bad!
- Need to choose vertices for a sub-problem carefully
- Let's "shrink" *S* by a factor of $\log^{1/3}(n)$



Piv

- R
  c
  tl
- T
  o
  v

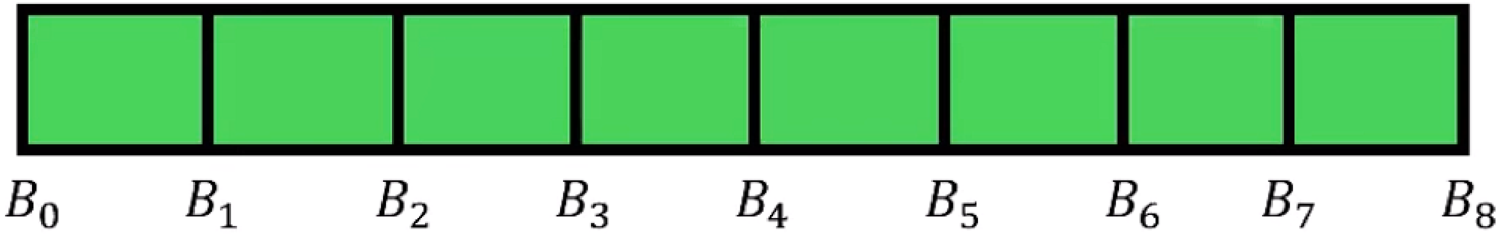
Sho
in S



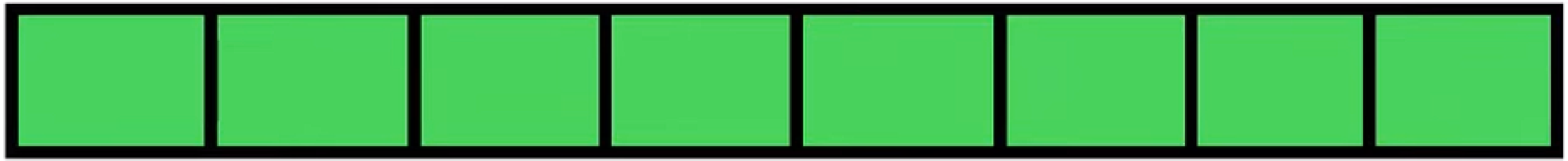
Algorithm 1 Finding Pivots

```
function FindPivots($B, S$)
    • requirement: for every incomplete vertex $v$ with $d(v) < B$, the shortest path to $v$ visits some complete vertex in $S$
    • returns: sets $P, W$ satisfying the conditions in Lemma 3.2
    $W \leftarrow S$
    $W_0 \leftarrow S$
    for $i \leftarrow 1$ to $k$ do                                        ▷ Relax for $k$ steps
        $W_i \leftarrow \emptyset$
        for all edges $(u, v)$ with $u \in W_{i-1}$ do
            if $\widehat{d}[u] + w_{uv} \leq \widehat{d}[v]$ then
                $\widehat{d}[v] \leftarrow \widehat{d}[u] + w_{uv}$
                if $\widehat{d}[u] + w_{uv} < B$ then
                    $W_i \leftarrow W_i \cup \{v\}$
        $W \leftarrow W \cup W_i$
        if $|W| > k|S|$ then
            $P \leftarrow S$
            return $P, W$
    $F \leftarrow \{(u, v) \in E : u, v \in W, \widehat{d}[v] = \widehat{d}[u] + w_{uv}\}$    ▷ $F$ is a directed forest under Assumption 2.1
    $P \leftarrow \{u \in S : u$ is a root of a tree with $\geq k$ vertices in $F\}$
    return $P, W$
```

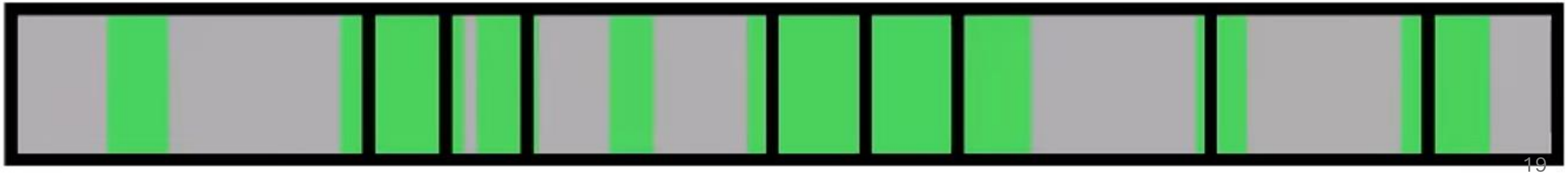
How to divide into sub-problems?



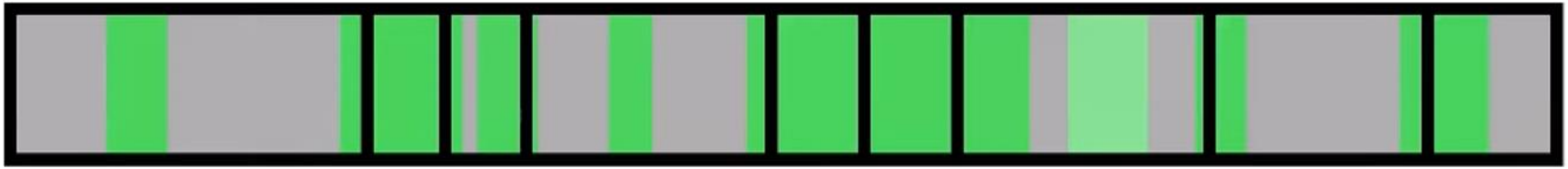
Ideally



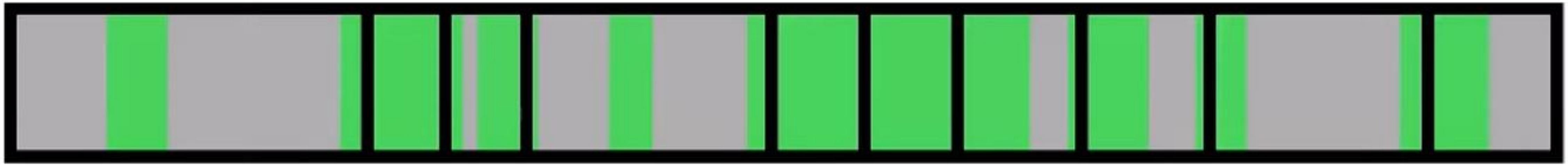
Reality (guessing!)



What if a sub-problem becomes too large?

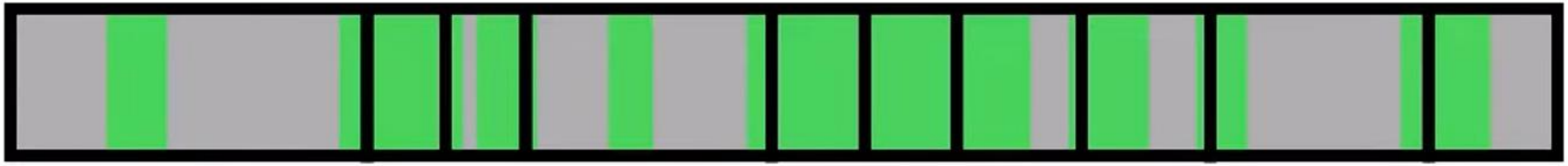


New distances were computed

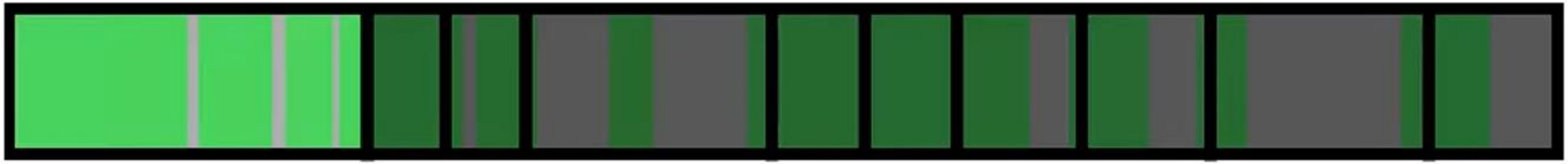


Split the sub-problem

What if already started solving such sub-problem?

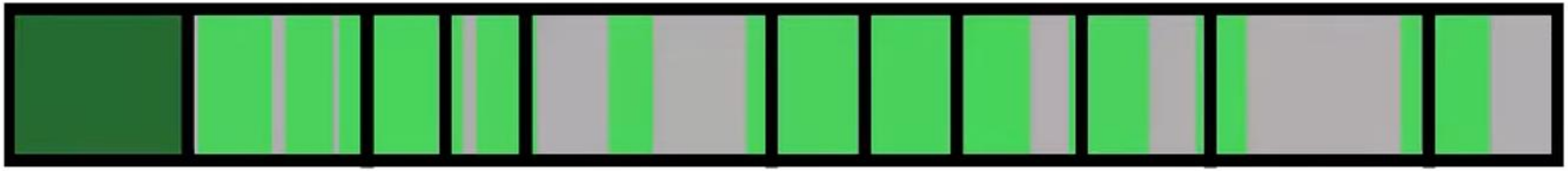


Started this sub-problem

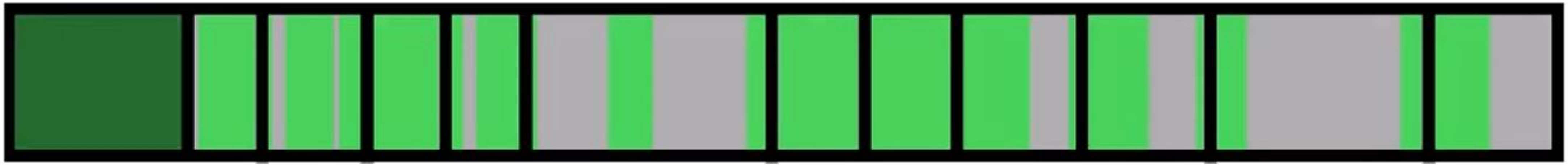


More distances were computed

What if already started solving such sub-problem?

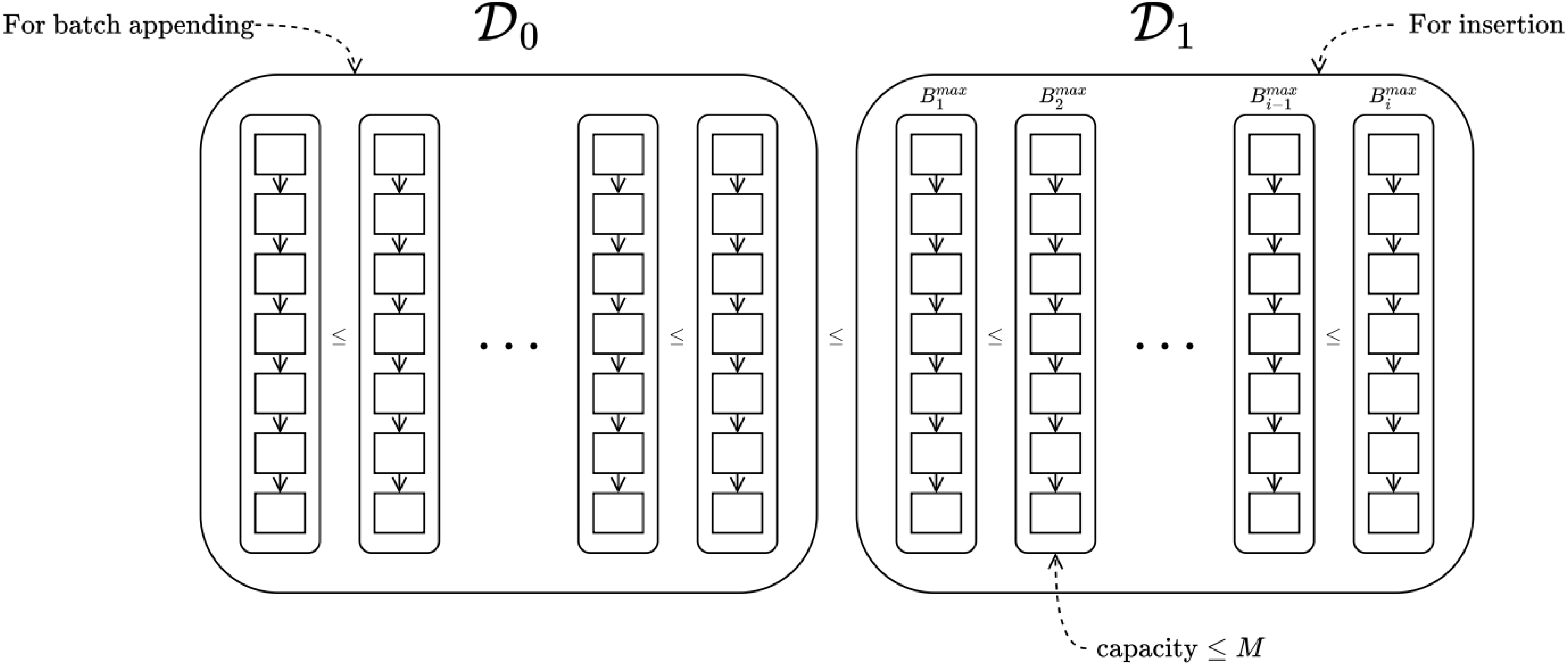


Go back to the parent problem

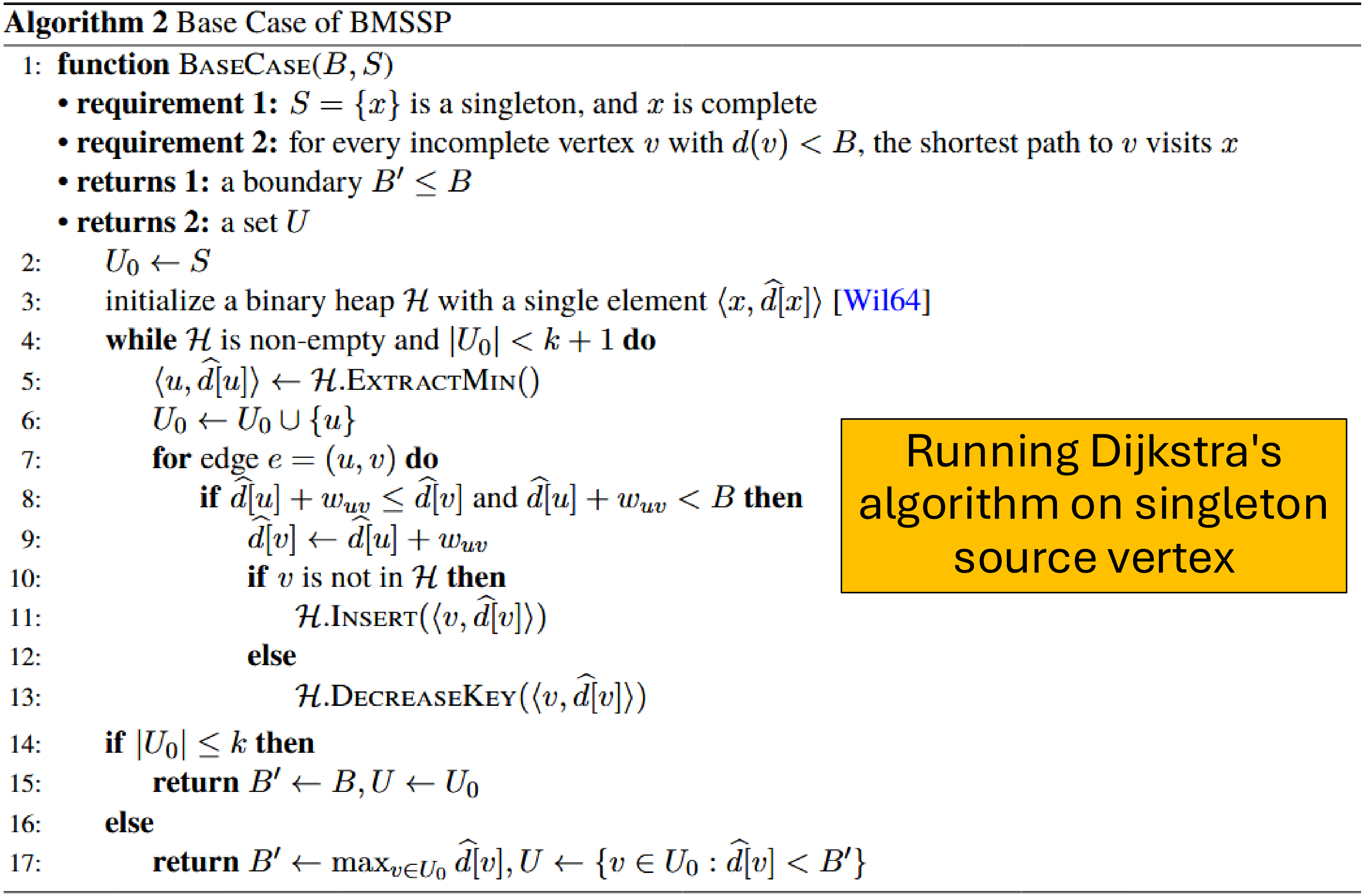


Rebalance the unfinished work

How are the bounds managed?



Base Case

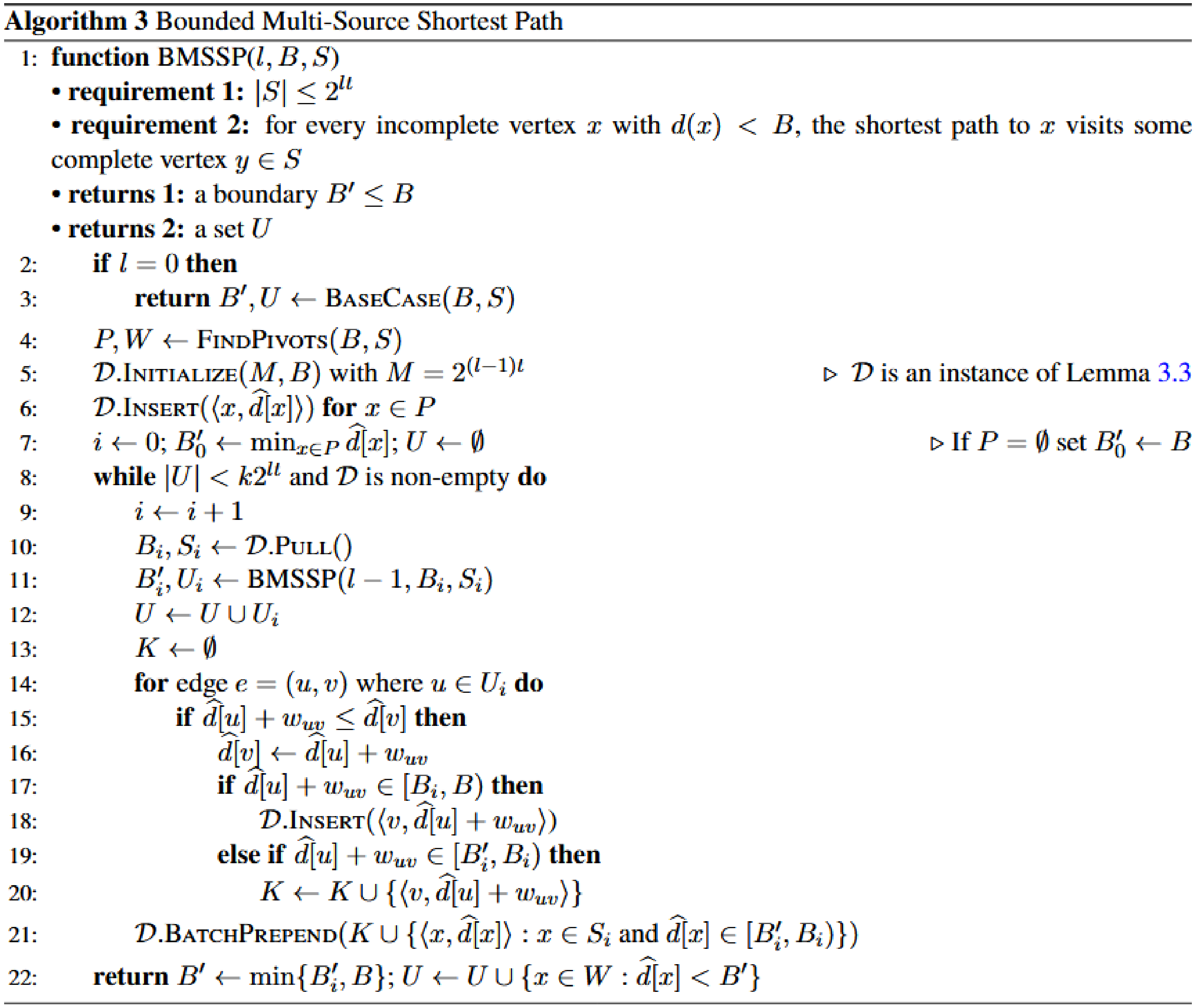


Algorithm 2 Base Case of BMSSP

```
function BaseCase($B, S$)
    • requirement 1: $S = \{x\}$ is a singleton, and $x$ is complete
    • requirement 2: for every incomplete vertex $v$ with $d(v) < B$, the shortest path to $v$ visits $x$
    • returns 1: a boundary $B' \leq B$
    • returns 2: a set $U$
    $U_0 \leftarrow S$
    initialize a binary heap $\mathcal{H}$ with a single element $\langle x, \widehat{d}[x] \rangle$ [Wil64]
    while $\mathcal{H}$ is non-empty and $|U_0| < k + 1$ do
        $\langle u, \widehat{d}[u] \rangle \leftarrow \mathcal{H}$.ExtractMin()
        $U_0 \leftarrow U_0 \cup \{u\}$
        for edge $e = (u, v)$ do
            if $\widehat{d}[u] + w_{uv} \leq \widehat{d}[v]$ and $\widehat{d}[u] + w_{uv} < B$ then
                $\widehat{d}[v] \leftarrow \widehat{d}[u] + w_{uv}$
                if $v$ is not in $\mathcal{H}$ then
                    $\mathcal{H}$.Insert($\langle v, \widehat{d}[v] \rangle$)
                else
                    $\mathcal{H}$.DecreaseKey($\langle v, \widehat{d}[v] \rangle$)
    if $|U_0| \leq k$ then
        return $B' \leftarrow B, U \leftarrow U_0$
    else
        return $B' \leftarrow \max_{v \in U_0} \widehat{d}[v], U \leftarrow \{v \in U_0 : \widehat{d}[v] < B'\}$
```

Running Dijkstra's algorithm on singleton source vertex

Recursive Case



Algorithm 3 Bounded Multi-Source Shortest Path

```
function BMSSP($l, B, S$)
  • requirement 1: $|S| \le 2^{lt}$
  • requirement 2: for every incomplete vertex $x$ with $d(x) < B$, the shortest path to $x$ visits some complete vertex $y \in S$
  • returns 1: a boundary $B' \le B$
  • returns 2: a set $U$
    if $l = 0$ then
        return $B', U \leftarrow$ BaseCase($B, S$)
    $P, W \leftarrow$ FindPivots($B, S$)
    $\mathcal{D}$.Initialize($M, B$) with $M = 2^{(l-1)t}$          ▷ $\mathcal{D}$ is an instance of Lemma 3.3
    $\mathcal{D}$.Insert($\langle x, \widehat{d}[x] \rangle$) for $x \in P$
    $i \leftarrow 0$; $B'_0 \leftarrow \min_{x \in P} \widehat{d}[x]$; $U \leftarrow \emptyset$          ▷ If $P = \emptyset$ set $B'_0 \leftarrow B$
    while $|U| < k2^{lt}$ and $\mathcal{D}$ is non-empty do
        $i \leftarrow i + 1$
        $B_i, S_i \leftarrow \mathcal{D}$.Pull()
        $B'_i, U_i \leftarrow$ BMSSP($l - 1, B_i, S_i$)
        $U \leftarrow U \cup U_i$
        $K \leftarrow \emptyset$
        for edge $e = (u, v)$ where $u \in U_i$ do
            if $\widehat{d}[u] + w_{uv} \le \widehat{d}[v]$ then
                $\widehat{d}[v] \leftarrow \widehat{d}[u] + w_{uv}$
                if $\widehat{d}[u] + w_{uv} \in [B_i, B)$ then
                    $\mathcal{D}$.Insert($\langle v, \widehat{d}[u] + w_{uv} \rangle$)
                else if $\widehat{d}[u] + w_{uv} \in [B'_i, B_i)$ then
                    $K \leftarrow K \cup \{\langle v, \widehat{d}[u] + w_{uv} \rangle\}$
        $\mathcal{D}$.BatchPrepend($K \cup \{\langle x, \widehat{d}[x] \rangle : x \in S_i$ and $\widehat{d}[x] \in [B'_i, B_i)\}$)
    return $B' \leftarrow \min\{B'_i, B\}$; $U \leftarrow U \cup \{x \in W : \widehat{d}[x] < B'\}$
```

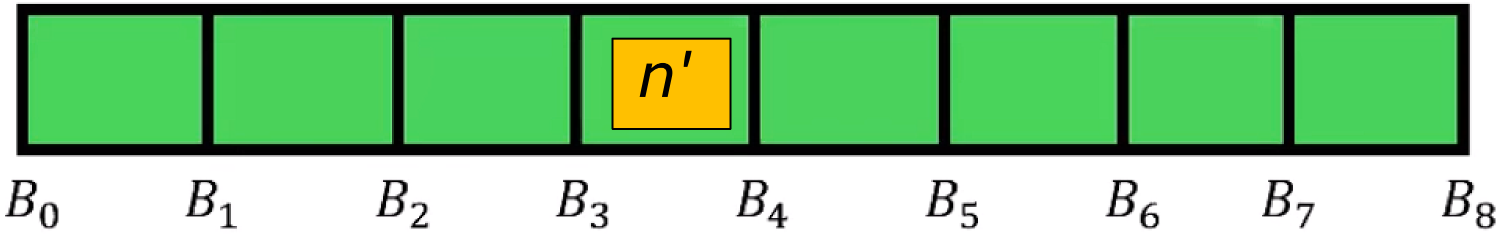
Intuition for Time Complexity

Per vertex:

- Operations in the data structure take O(1) time
- Might be in pivot pruning: $O(k^2)$
- Might be in the base case: O(1) (constant degree assumption)

Total: $O(nk^2)=O(n \log^{2/3}(n))$

Intuition for Space Complexity



Per sub-problem: $\Omega(n')$

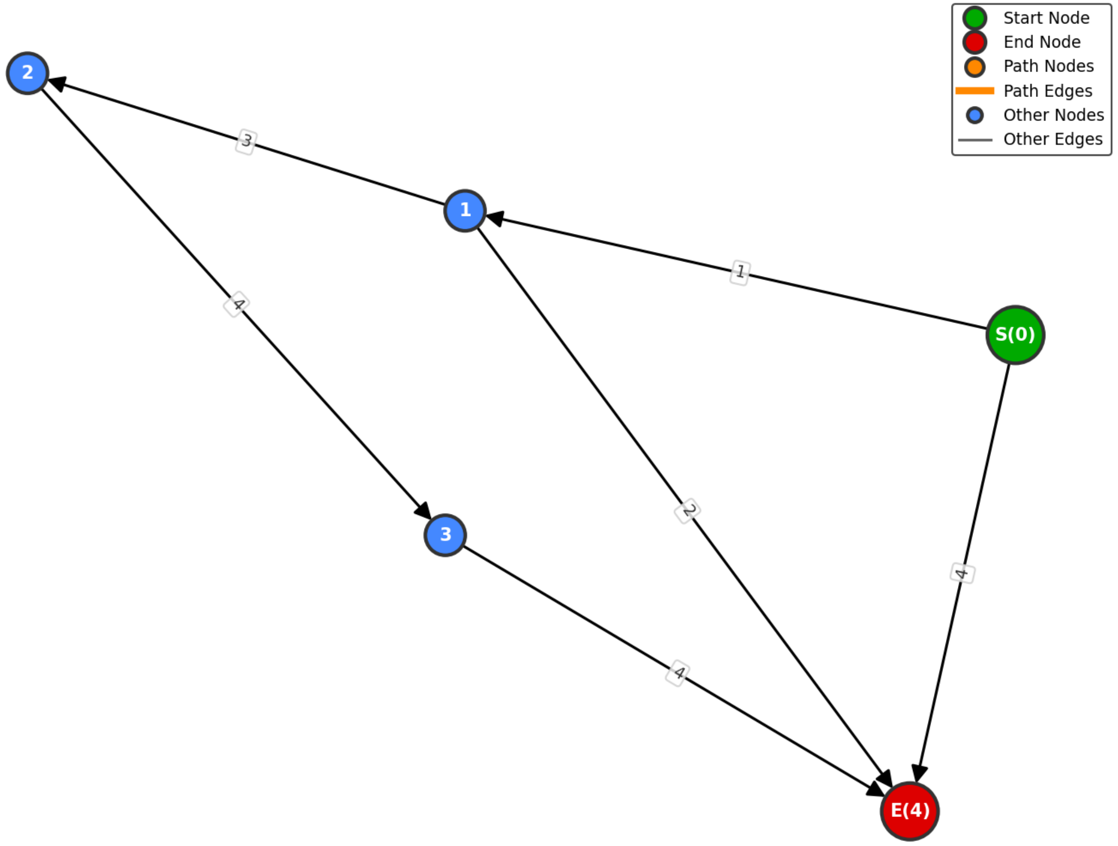
Per level: $\Omega(n)$

Total: $\Omega(n \log(n) / t) = \Omega(n \log^{1/3}(n))$

Much worse in our implementation :(

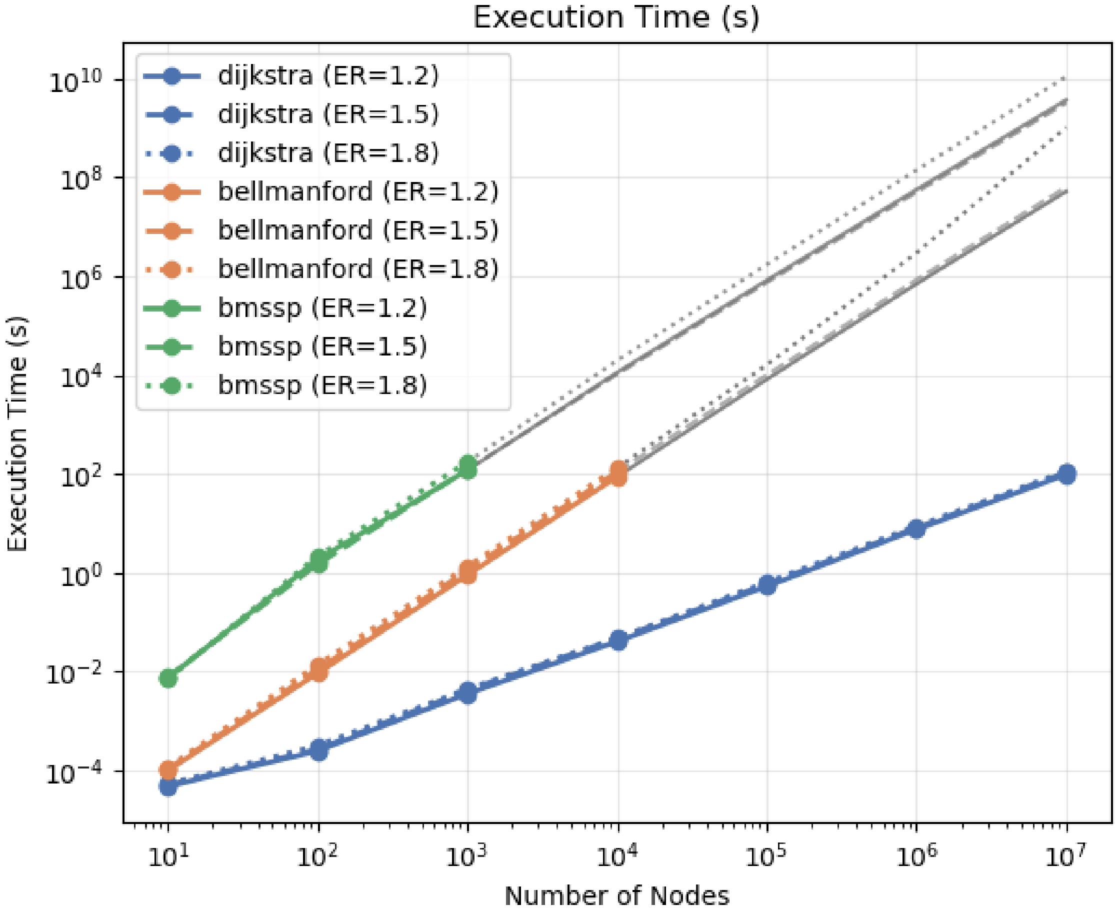
Dataset Generation

- Limited our study to directed sparse-graphs.
- Created a synthetic dataset with specific number of nodes and varying edge ratios.
- Experiments ran for:
  - 10 to 10^7 nodes
  - 1.2, 1.5, 1.8 edge ratios
- Weights were uniformly sampled from 1 to 10.
- Controlled in- and out-degree of nodes during benchmarking.



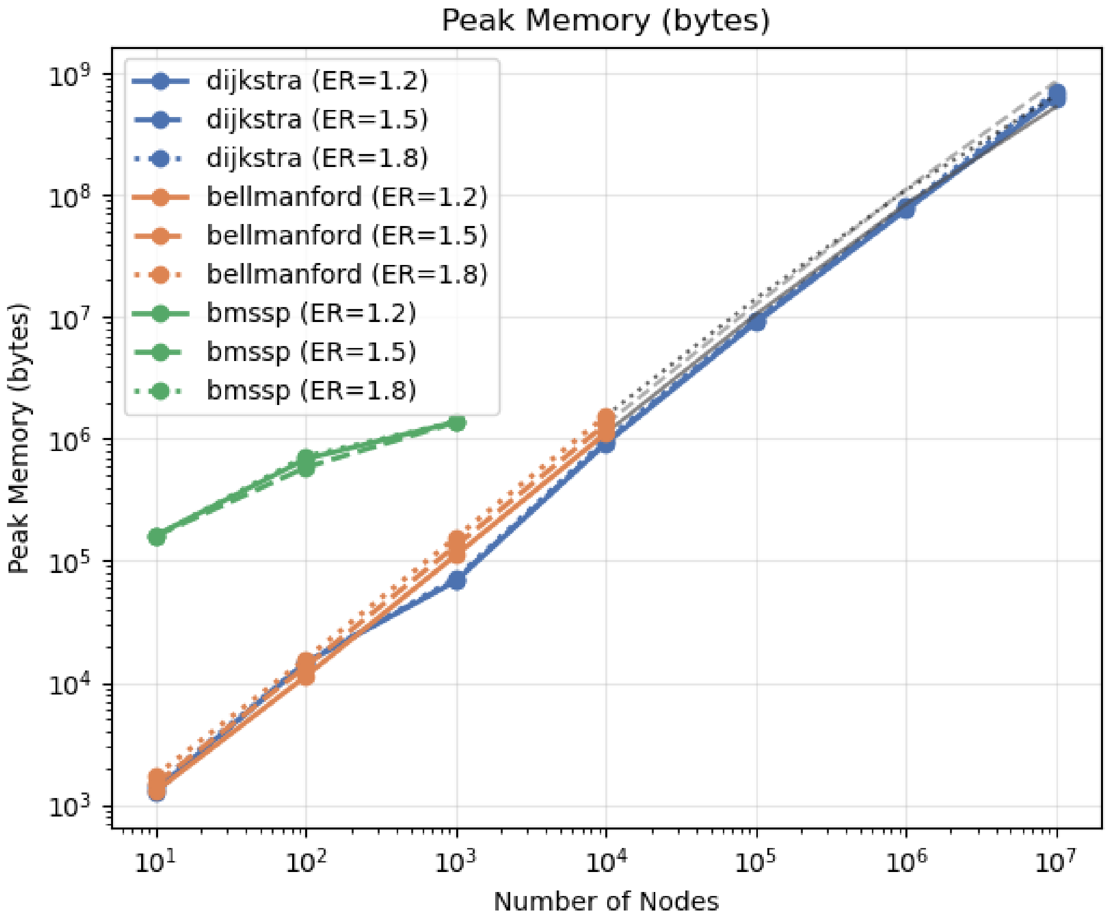
Results: Time Complexity

- As expected, Dijkstra's algorithm was significantly faster than the other two algorithms.
- BMSSP took considerably longer but still exhibited a quasi-polynomial growth pattern.
- Although BMSSP is theoretically superior to Dijkstra's, its advantages may be hard to justify for graph sizes under billions of nodes.



Results: Space Complexity

- Dijkstra and Bellman-Ford showed similar peak memory usage across graphs.
- BMSSP’s memory footprint was much larger due to repeated data structure reinitialization and recursion.



DEMO TIME