Online graph exploration on trees

Md.Mahedi Hasan Rigan(1705031)
Asif Mustafa Hassan(1705041)

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1205, Bangladesh

July 7, 2021

Online Graph Exploration Example

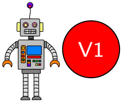


Figure: Online exploratoin example

Online Graph Exploration Example

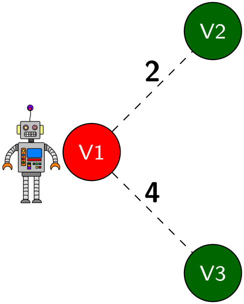


Figure: Online exploratoin example

Online Graph Exploration Example

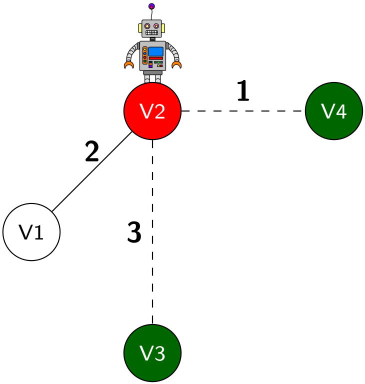


Figure: Online exploratoin example

Online Graph Exploration Example

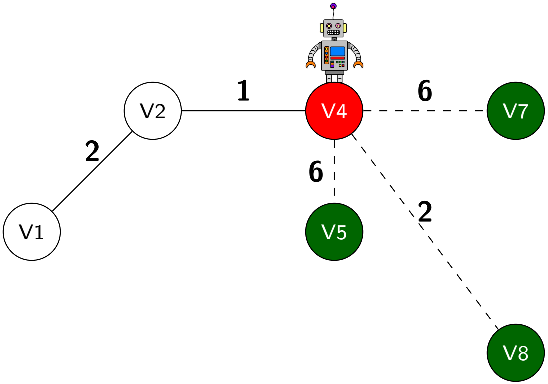


Figure: Online exploratoin example

Online Graph Exploration Example

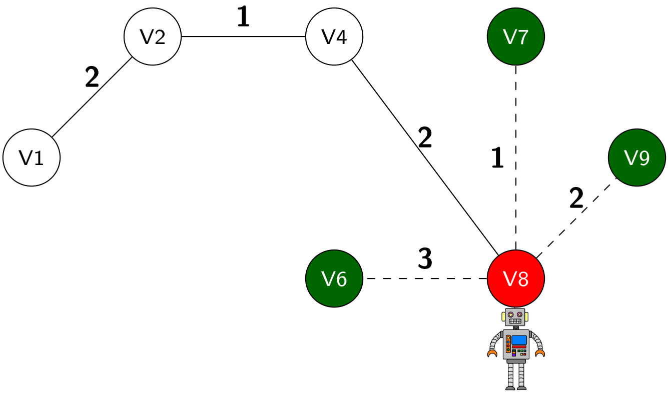


Figure: Online exploratoin example

Online Graph Exploration Example

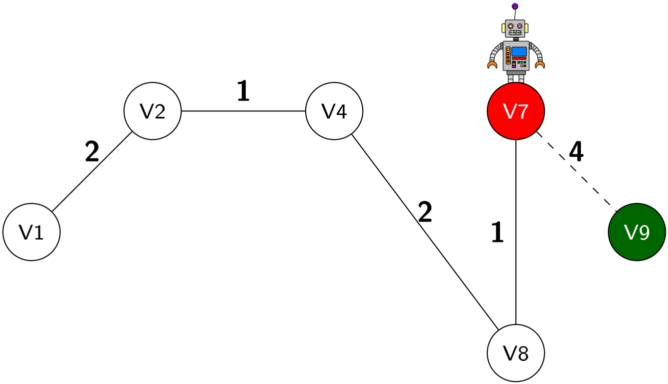


Figure: Online exploratoin example

Online Graph Exploration Example

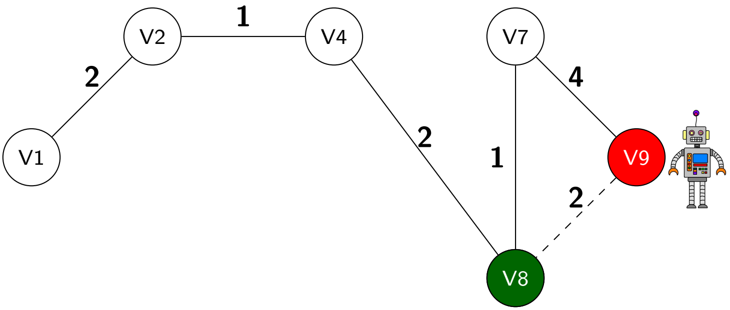


Figure: Online exploratoin example

Online Graph Exploration Example

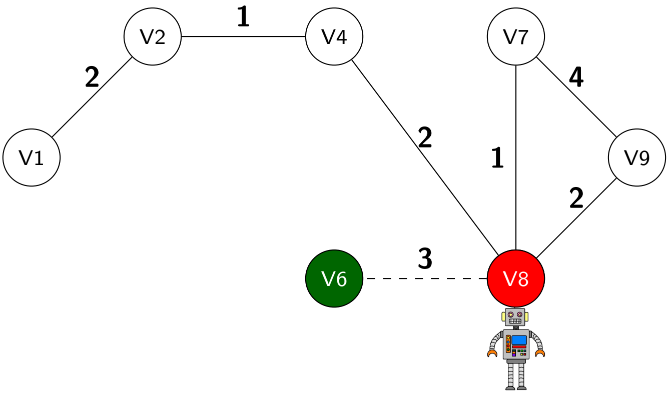


Figure: Online exploratoin example

Online Graph Exploration Example

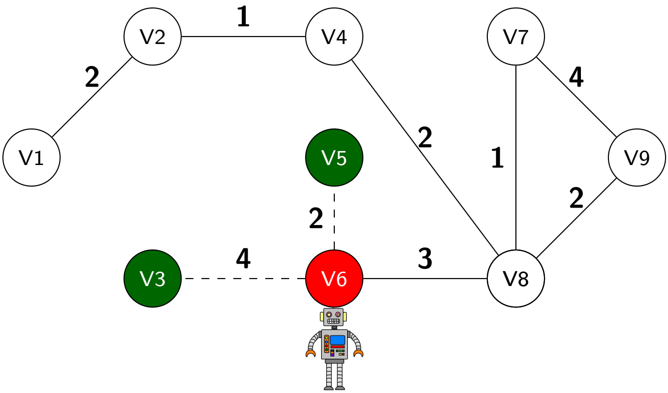


Figure: Online exploratoin example

Online Graph Exploration Example

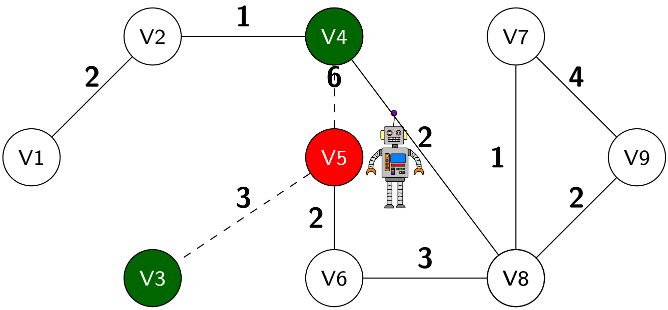


Figure: Online exploratoin example

Online Graph Exploration Example

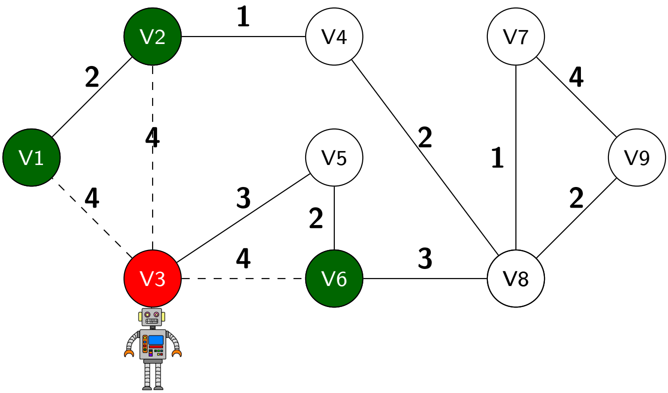


Figure: Online exploratoin example

Online Graph Exploration Example

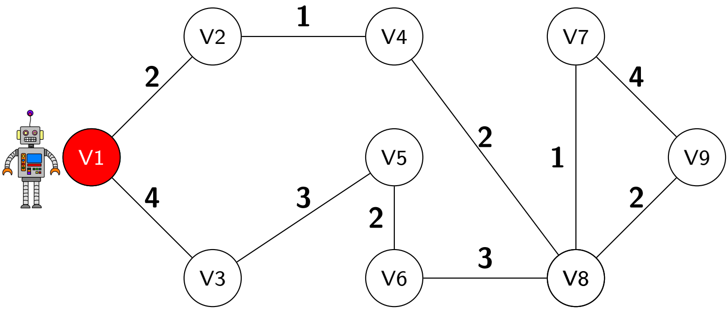


Figure: Online exploratoin example

Online Graph Exploration

- A technique to explore all vertices of a graph and return to the start

Online Graph Exploration

- A technique to explore all vertices of a graph and return to the start
- Initially all the vertices are unknown

Online Graph Exploration

- A technique to explore all vertices of a graph and return to the start
- Initially all the vertices are unknown
- While arriving at a vertex for the first time, the searcher will get the weights of all edges incident to the vertex.

Online Graph Exploration

- A technique to explore all vertices of a graph and return to the start
- Initially all the vertices are unknown
- While arriving at a vertex for the first time, the searcher will get the weights of all edges incident to the vertex.
- Must visit every vertex before returning to the start node.

How to measure the performance

How could we measure the performance of an online algorithm?

How to measure the performance

- Using Competitive analysis

How to measure the performance

- Using Competitive analysis
- **Competitive ratio:** A infimum ratio of an online problem solution to its solution of the corresponding offline problem

How to measure the performance

- Using Competitive analysis
- **Competitive ratio:** A infimum ratio of an online problem solution to its solution of the corresponding offline problem
- If an online tour is no longer than c times the optimal tour on offline graph, then it's called *c-competitive* algorithm.

Nearest Neighbor Algorithm

The algorithm is follows a **greedy** approach. At each vertex, follows the best choice without consideration of future

1. Select a starting point
2. Move to the nearest unvisited smallest vertex using the edge
3. Repeat until all the vertices are visited

Problem

- The best known algorithms on general graphs are:
  - Nearest Neighbour Approach (NN)
  - Hierarchical DFS
- Between NN and Hierarchical DFS
  NN has tighter worst case ratio (better)
- Competitive ratio of NN: $\Theta(\log_2 n)$
- But no constant competitive ratio on general graphs

Problem

- No constant competitive ratio on general graphs
- What do we do?
- We define bounds on competitive ratio:
  - Lower bound
  - Upper bound
- As there is no certain Competitive Ratio for a particular algorithm we work with the bounds
- Lower difference between the bounds defines better algorithms.

Improvement of Algorithms

- Work with the bounds of competitive ratio
- Try to improve the bounds
- Define a family of graph class
- Try to improve bounds of general class for the particular family

Our Goal

- We work on NN approach beacuse of its tighter bounds.
- Our goal is:
  - Construct a tree
  - Show that the tighter bounds of NN also holds on particular family of graph trees

Constructing the Tree Class

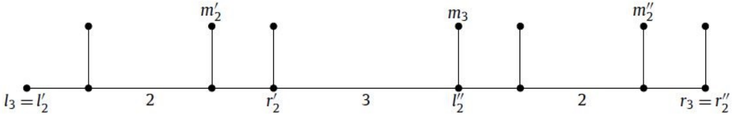


Figure: Graph for construction Hurkens and Woeginger

- Consider the graph by Hurkens and Woeginger
- We will modify this structure according to our interest
- We will:
  - Define a graph class G_k for $k \geq 1$
  - Calculate parameters of the graph

Constructing the Tree Class

- Define three vertices on tree
  - l_k : Left most node of tree G_k
  - r_k : Right most node of tree G_k
  - m_k : Node connected to the right recursive portion's l_k of G_k

Constructing the Tree Class

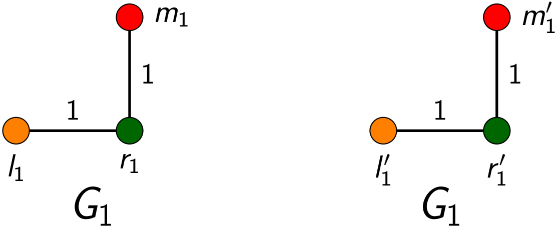
- Define three vertices on tree
  - l_k : Left most node of tree G_k
  - r_k : Right most node of tree G_k
  - m_k : Node connected to the right recursive portion's l_k of G_k
- This will be a recursive graph
- Build graph for G_1
  - Connect l_1 to r_1 with unit length edge
  - Connect m_1 to r_1 with unit length edge

Constructing the Tree Class

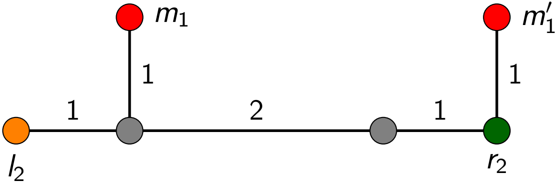
- Define three vertices on tree
  - l_k : Left most node of tree G_k
  - r_k : Right most node of tree G_k
  - m_k : Node connected to the right recursive portion's l_k of G_k
- This will be a recursive graph
- Build graph for G_1
  - Connect l_1 to r_1 with unit length edge
  - Connect m_1 to r_1 with unit length edge
- This will be a recursive graph
- Build graph for G_k
- Connect G_{k-1} to G'_{k-1}
  - Connect r_{k-1} to l'_{k-1} with edge of length k
  - Connect m_k to r'_{k-1} with unit length edge

Construction of the graph is done!!

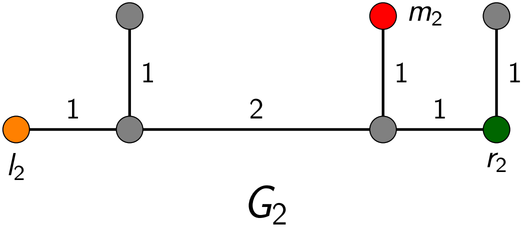
Example of Constructed Tree



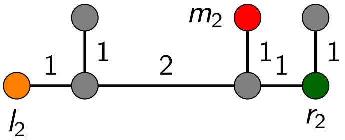
Example of Constructed Tree

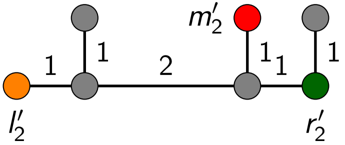


Example of Constructed Tree

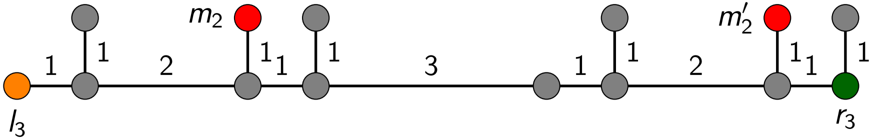


Example of Constructed Tree 2

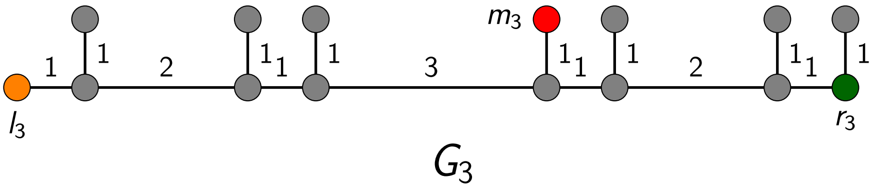




Example of Constructed Tree 2



Example of Constructed Tree 2



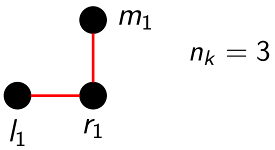


Figure: Graph for G_1

We define number of vertices as n_k

- For $k = 1$, $n_k = 3$

Parameters for Tree

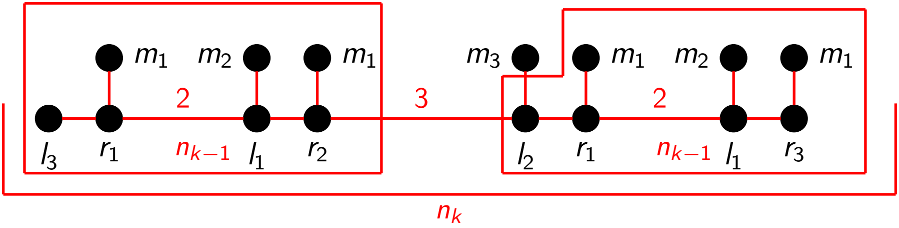


Figure: Graph for G_3

We define number of vertices as n_k

- For $k = 1$, $n_k = 3$
- For $k \geq 1$, $n_k = 2 * n_{k-1} + 1$

It can be proved by induction that:

$$n_k = 2^{k+1} - 1$$

Parameters for Tree

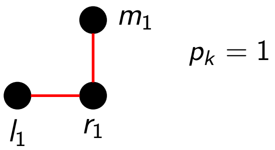


Figure: Graph for G_1

We define length from l_k to r_k as p_k

- For $k = 1$, $p_k = 1$

Parameters for Tree

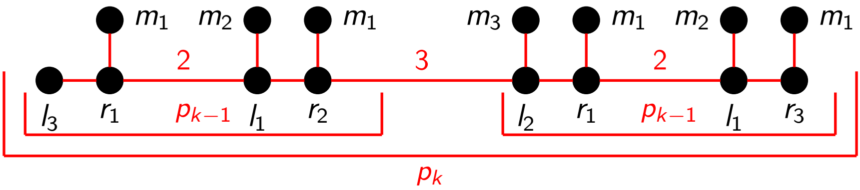


Figure: Graph for G_3

We define length from l_k to r_k as p_k

- For $k = 1$, $p_k = 1$
- For $k > 1$, $p_k = 2 * p_{k-1} + k$

It can be proved by induction that:

$$p_k = 2^{k+1} - k - 2$$

Parameters for Tree

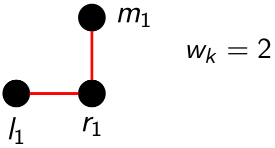


Figure: Graph for G_1

We define sum of weights of all edges from l_k to r_k as w_k

- For $k = 1$, $w_k = 2$

Parameters for Tree

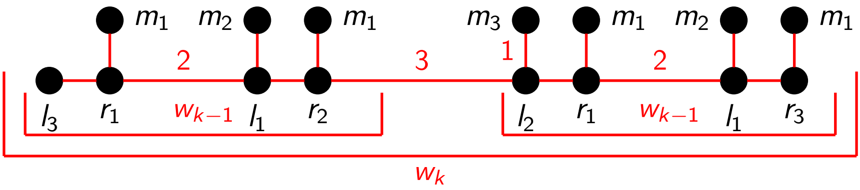


Figure: Graph for G_3

We define sum of weights of all edges from l_k to r_k as w_k

- For $k = 1$, $w_k = 2$
- For $k > 1$, $w_k = 2 * w_{k-1} + k + 1$

It can be proved by induction that:

$$w_k = 3 * 2^k - k - 3$$

Parameters for Tree

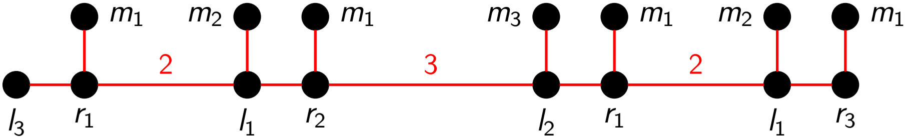


Figure: Graph for G_3

We define "optimal path weight for the graph to start at l_k and after visiting all the nodes, return to l_k" as $OPT(G_k)$

$$
\begin{aligned}
OPT(G_k) &= 2 * w_k \\
&= 2 * \left(3 * 2^k - k - 3\right) \\
&= 6 * 2^k - 2 * k - 6
\end{aligned}
$$

Parameters for Tree

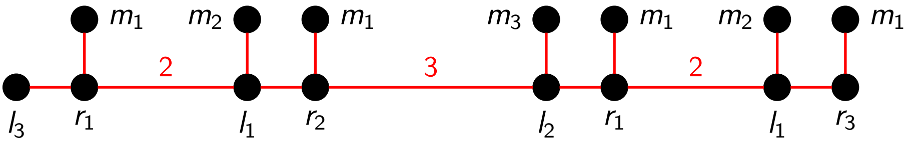


Figure: Graph for G_3

Number of vertices: $n_k = 2^{k+1} - 1$
Length from l_k to r_k: $p_k = 2^{k+1} - k - 2$
Sum of weights of all edges from l_k to r_k: $w_k = 3 * 2^k - k - 3$

Optimal path weight for the graph to start at l_k and after visiting all the nodes, return to l_k: $OPT(G_k) = 6 * 2^k - 2 * k - 6$

Lemma 1

Lemma: For $k \geq 1$, consider a graph G that contains G_k as a subgraph. Furthermore, assume that edges between G_k and $G - G_k$ are either incident to l_k and have a length of at least 1 or are incident to r_k and have a length of at least $k + 1$.

Then there exists a partial NN tour exploring all of G_k that starts in l_k, finishes in m_k and has a length of

$$(k + 1) * 2^k - 2$$

Proof of Lemma 1

- edges between G_k and $G - G_k$ are either incident to l_k and have a length of at least 1 or are incident to r_k and have a length of at least $k + 1$

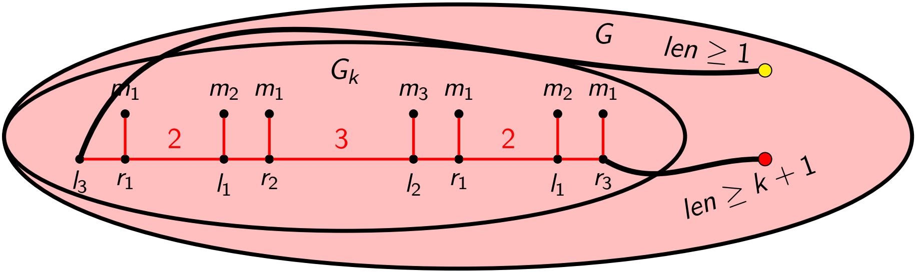


Figure: Graph for Lemma 1

Proof of Lemma 1

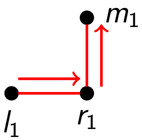


Figure: Condition for Lemma 1

We prove this lemma by induction.

- Let us consider graph G_1
- l_1 to m_1 path has length 3
- Satisfies the equation $(k+1) * 2^k - 2$ where $k = 1$

Proof of Lemma 1

In G_k, path starts at l_k, finishes at m_k and has a length of $(k+1) * 2^k - 2$

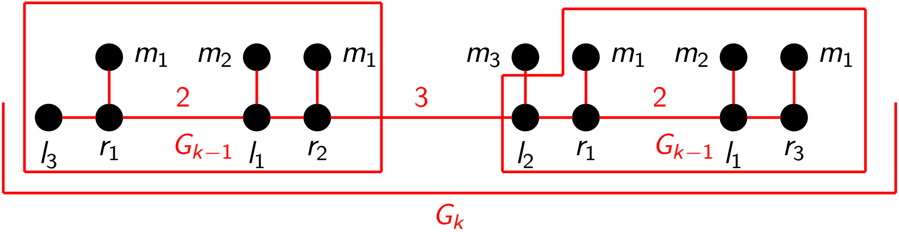


Figure: Graph for G_3

- G_k consists of two subgraph G_{k-1} and G'_{k-1}
- As this is proof by induction, we assume the conditions are met for the two subgraph G_{k-1} and G'_{k-1}

Proof of Lemma 1

In G_k, path starts at l_k, finishes at m_k and has a length of $(k+1)*2^k-2$

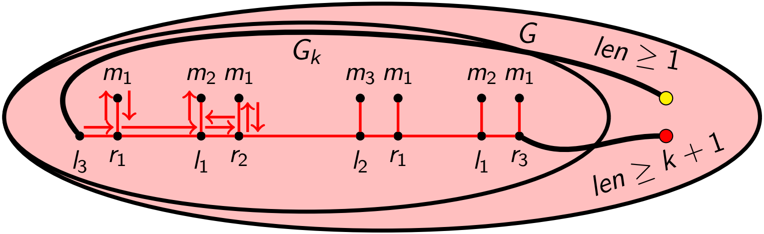


Figure: Traversal in G_{k-1}

- We start at l_{k-1} in G_{k-1} which is also l_k for G_k
- We reach m_{k-1} from l_{k-1}
- path length: $k*2^{k-1}-2$ as G_{k-1} satisfies the the lemma for $k-1$
- G_{k-1} graph traverse is done

Proof of Lemma 1

In G_k, path starts at l_k, finishes at m_k and has a length of $(k+1)*2^k - 2$
We are at m_{k-1}, we need to traverse subgraph G'_{k-1} and then reach m_k

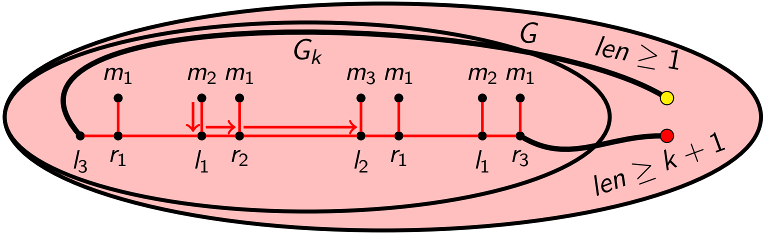


Figure: m_{k-1} to l'_{k-1}

- We can go outside of G_{k-1} and then traverse G'_{k-1} independently
  - Through l_{k-1}, length at least $1 + k + p_{k-2}$
  - Through r_{k-1}, length at least $1 + p_{k-2} + k$
- We can go straight to l'_{k-1}, length: $1 + p_{k-2} + k$

We choose going to l'_{k-1}

Proof of Lemma 1

In G_k, path starts at l_k, finishes at m_k and has a length of $(k+1)*2^k-2$

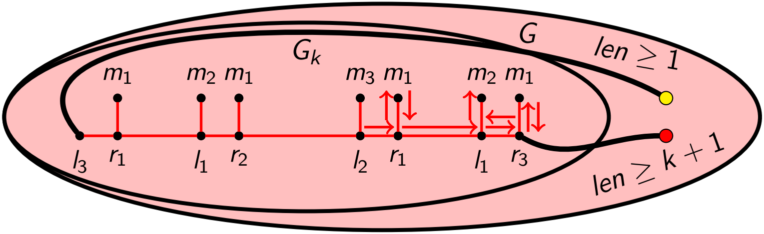


Figure: Caption

- We are at l'_{k-1} in G'_{k-1}
- We reach m'_{k-1} from l'_{k-1}
- path length: $k*2^{k-1}-2$ as G'_{k-1} satisfies the the lemma for $k-1$
- G'_{k-1} graph traverse is done

Proof of Lemma 1

In G_k, path starts at l_k, finishes at m_k and has a length of $(k+1) * 2^k - 2$

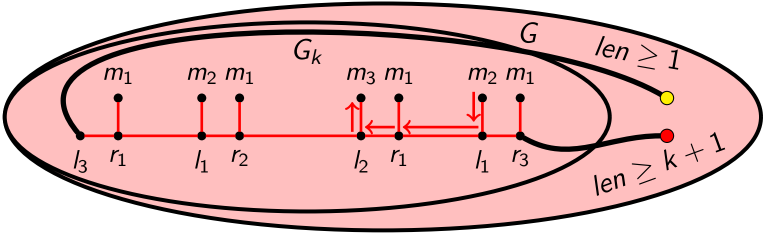


Figure: m'_{k-1} to m_k

Now, only m_k vertex traversal is left, we are at m'_{k-1}

- We can go outside of G_k and then reach m_k independently
  - Through l_{k-1}, length at least $1 + 3 * k + 3 * p_{k-2}$
  - Through r'_{k-1}, length at least $2 + p_{k-2} + k$
- We can go straight to m_k through l'_{k-1}, length: $1 + k + p_{k-2}$

Obviously, we choose to reach straight to m_k

Proof of Lemma 1

In G_k, path starts at l_k, finishes at m_k and has a length of $(k+1)*2^k-2$

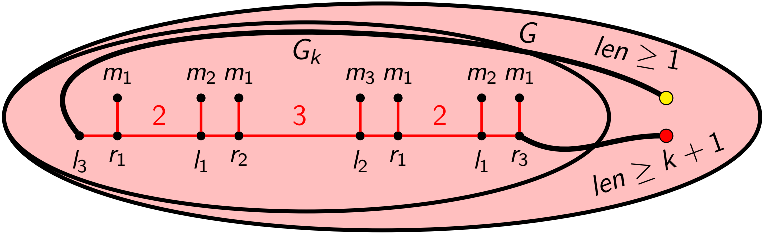


Figure: Graph for G_k

We have completed our traversal from l_k to m_k. Total length of the path:

- l_k to m_{k-1} in subgraph G_{k-1}: $k*2^{k-1}-2$
- m_{k-1} to l'_{k-1}: $1+k+p_{k-2}$
- l'_{k-1} to m'_{k-1} in subgraph G'_{k-1}: $k*2^{k-1}-2$
- m'_{k-1} to m_k: $1+k+p_{k-2}$

Proof of Lemma 1

Total length of whole traversal:

$$
\begin{aligned}
L_k &= 2 * (1 + k + p_{k-2}) + 2 * (k * 2^{k-1} - 2) \\
&= 2 + 2k + 2p_{k-2} + 2k * 2^{k-1} - 4 \\
&= 2k - 2 + 2k * 2^{k-1} + 2p_{k-2} \\
&= 2k - 2 + 2k * 2^{k-1} + 2(2^{k-1} - (k-2) - 2) \\
&= 2k - 2 + k * 2^k + 2^k - 2k + 4 - 4 \\
&= 2^k(k+1) - 2 \quad \square
\end{aligned}
$$

Competitive Ratio of Trees

Our goal was to show that the "tighter bounds of competitive ratio of NN" also holds on the family of the tree.

- The upper bound follows directly from the general case.
- We show that: The lower bound on trees remains same if we follow *NN* approach.

Competitive Ratio of Trees

Calculating the Competitive Ratio lower bound:

- Traversing graph G_k when G_k is a part of larger graph G with some conditions, the shortest path length from l_k to m_k is $(k+1) * 2^k - 2$. [Lemma 1]
- We can use Lemma 1 for only G_k as we did not need to go outside G_k in Lemma 1.
- Finally complete exploration by returning to l_k from m_k.

Competitive Ratio of Trees

Calculating the Competitive Ratio lower bound:

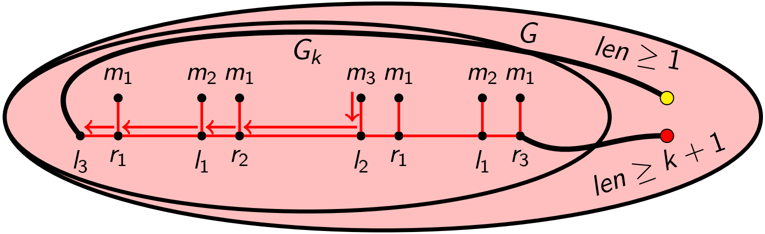


Figure: m_k to l_k

- l_k to m_k needs length: $(k+1) * 2^k - 2$
- m_k to l_k needs length:

$$
\begin{aligned}
Ret(G_k) &= 1 + k + p_{k-1} \\
&= 1 + k + (2^k - (k-1) - 2) \\
&= 1 + k + 2^k - k + 1 - 2 = 2^k
\end{aligned}
$$

Competitive Ratio of Trees

Calculating the Competitive Ratio lower bound:

- Total length for complete exploration on Online Tree G_k:

$$\begin{aligned} NN(G_k) &= Ret(G_k) + (k+1) * 2^k - 2 \\ &= 2^k + (k+1) * 2^k - 2 \\ &= (k+2) * 2^k - 2 \end{aligned}$$

Competitive Ratio of Trees

Calculating the Competitive Ratio lower bound:

- The NN approach on particular Online Tree family has length:

$$(k+2) * 2^k - 2$$

- The optimal solution on this offline Tree family should have length:

$$\begin{aligned} OPT(G_k) &= 2 * w_k \\ &= 6 * 2^k - 2k - 6 \end{aligned}$$

- G_k has vertices:

$$\begin{aligned} n_k &= 2^{k+1} - 1 \\ 2^{k+1} &= n_k + 1 \\ log_2 2^{k+1} &= log_2(n_k + 1) \\ k + 1 &= log_2(n_k + 1) \\ k &= log_2(n_k + 1) - 1 \end{aligned}$$

Competitive Ratio of Trees

Calculating the Competitive Ratio lower bound:

- Lower bound:

$$
\begin{aligned}
c &= \frac{NN(G_k)}{OPT(G_k)} \\
&= \frac{(k+2)*2^k - 2}{6*2^k - 2k - 6} \\
&\geq \frac{k+2}{6} \\
&= \frac{\log_2(n+1)+1}{6} \qquad \square
\end{aligned}
$$

Thank You!