Fully Dynamic Maximal Matching in $O(\log n)$ update time

Manoj Gupta, IIT Delhi
Joint work with Sandeep Sen, IIT Delhi and Surender Baswana, IIT Kanpur

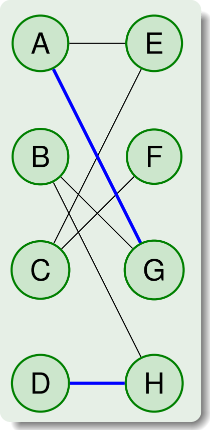
2nd Annual Mysore Park Workshop in Theoretical Computer Science

Outline

1. **Introduction**
   - The Problem
   - A Simple Agorithm
2. **$\sqrt{n}$ algorithm**
   - The overview of the approach
   - Overview of the analysis
   - Algorithm
3. **From $\sqrt{n}$ to $\log n$**
   - Speeding up the algorithm
4. **Open Problem**

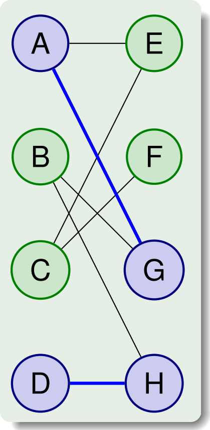
Some Definitions

- A matching in a graph is a set of edges M such that no two edges in M share a common endpoint.



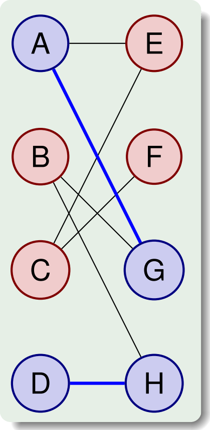
Some Definitions

- A matching in a graph is a set of edges M such that no two edges in M share a common endpoint.
- Matched vertex



Some Definitions

- A matching in a graph is a set of edges M such that no two edges in M share a common endpoint.
- Matched vertex
- Free vertex



The Problem

A matching is maximal if for each vertex v:

- v is matched or
- v does not have a free neighbor.

The Problem

A matching is maximal if for each vertex v:

- v is matched or
- v does not have a free neighbor.

Problem

Maintain maximal matching in a dynamic graph

The Problem

A matching is maximal if for each vertex v:

- v is matched or
- v does not have a free neighbor.

Problem

Maintain maximal matching in a dynamic graph

Expectation from the algorithm

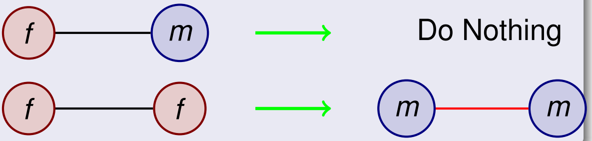
Update time should be polylog(n)

Previous Work

- Ivkovic and Llyod(1994) - $O((n+m)^{0.7072})$
- Onak and Rubinfeld(2010) gave a c-approximation of maximum matching in $O(\log^2 n)$ update time.

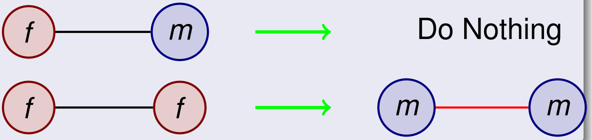
A Naive Approach

Insertion of an edge



A Naive Approach

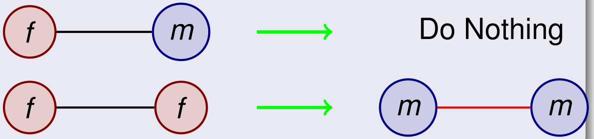
Insertion of an edge



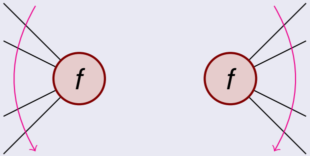
Deletion of an edge

A Naive Approach

Insertion of an edge



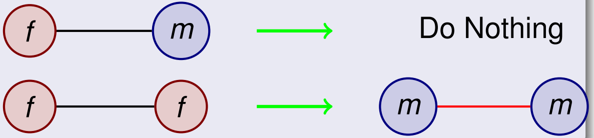
Deletion of an edge



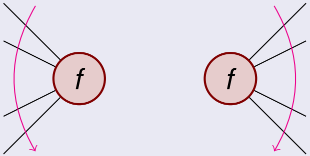
Search neighborhood of both vertex for free vertex

A Naive Approach

Insertion of an edge



Deletion of an edge



Search neighborhood of both vertex for free vertex

- Insertion = $O(1)$
- Deletion = $O(n)$

The difficulty

- Deletion of a matched edge
- Handling high degree vertex

The difficulty

- Deletion of a matched edge
- Handling high degree vertex

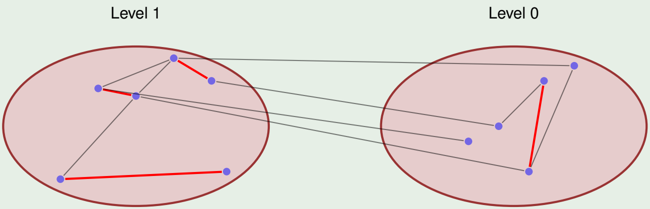
Possible ways to solve

- Make sure that high degree vertex are always matched
- Make sure that a matched edge is deleted rarely

- Partition the vertices into two buckets(level 1 and 0) such that most of the vertices have high "degree" when they come to level 1
- The partition is dynamic and the vertices may move from level 1 and level 0
- Maintain the following invariant
  - The vertex at level 1 are always matched
  - The vertex at level 0 has degree $< \sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched

The overview of the approach

- Partition the vertices into two buckets(level 1 and 0) such that most of the vertices have high "degree" when they come to level 1
- The partition is dynamic and the vertices may move from level 1 and level 0
- Maintain the following invariant
  - The vertex at level 1 are always matched
  - The vertex at level 0 has degree $< \sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched



Notion of ownership

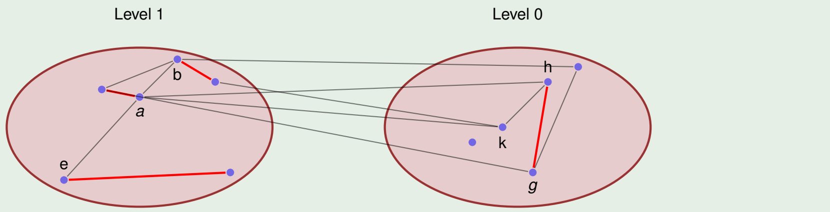
Each edge present in the graph will be owned by one or both of its end points as follows

- If both the end points are at level 0, then it is owned by both the endpoints
- If only one endpoint is at level 1, then it owns the edge
- If both the end points are at the same level, we can break the tie arbitrarily

Notion of ownership

Each edge present in the graph will be owned by one or both of its end points as follows

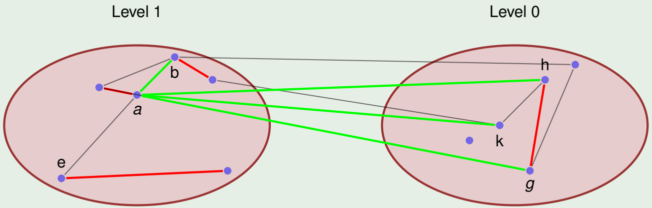
- If both the end points are at level 0, then it is owned by both the endpoints
- If only one endpoint is at level 1, then it owns the edge
- If both the end points are at the same level, we can break the tie arbitrarily



Notion of ownership

Each edge present in the graph will be owned by one or both of its end points as follows

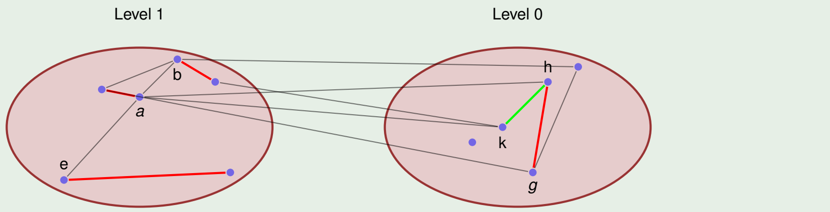
- If both the end points are at level 0, then it is owned by both the endpoints
- If only one endpoint is at level 1, then it owns the edge
- If both the end points are at the same level, we can break the tie arbitrarily



Notion of ownership

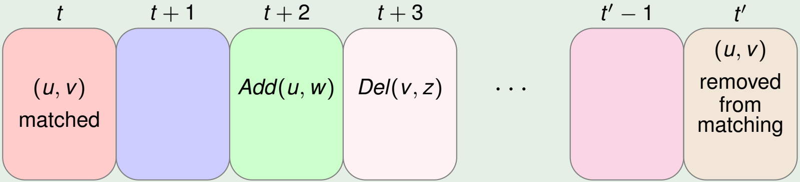
Each edge present in the graph will be owned by one or both of its end points as follows

- If both the end points are at level 0, then it is owned by both the endpoints
- If only one endpoint is at level 1, then it owns the edge
- If both the end points are at the same level, we can break the tie arbitrarily



Notion of matched epoch

Epoch of (u, v) is the maximal continuous time period for which it remains in the matching.



Epoch of Level 0

- Inv2:The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched

Start of the epoch

But...what if u has more than $\sqrt{n}$ edges in $G[V_0]$ after edge insertion?

Epoch of Level 0

- Inv2:The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched

Start of the epoch

But...what if u has more than $\sqrt{n}$ edges in $G[V_0]$ after edge insertion?

End of the epoch

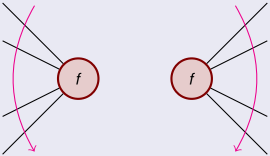
Epoch of Level 0

- Inv2:The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched

Start of the epoch

But...what if u has more than $\sqrt{n}$ edges in $G[V_0]$ after edge insertion?

End of the epoch



Search only the edges whose other endpoint are at level 0

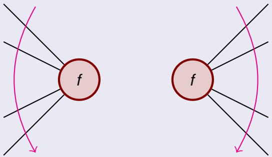
Epoch of Level 0

- Inv2:The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched

Start of the epoch

But...what if u has more than $\sqrt{n}$ edges in $G[V_0]$ after edge insertion?

End of the epoch



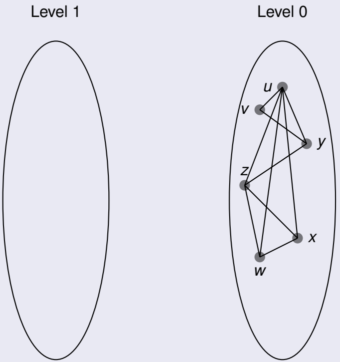
Search only the edges whose other endpoint are at level 0

- Start = $O(1)$
- End = $O(\sqrt{n})$

Epoch at level 1: Start

- Inv2:The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched

Invariant 2 does not hold for vertex u



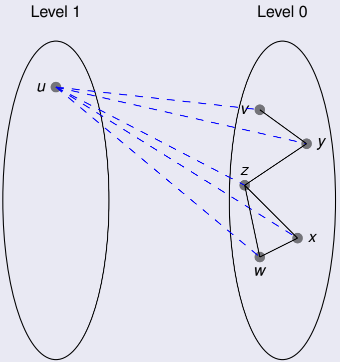
Epoch at level 1: Start

- Inv2:The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched

$\longrightarrow$ Vertex u moves to level 1

$\longrightarrow$ Make u the *owner* of all its edges at level 0

$\longrightarrow$ Find a random edge from the owned edges, say (u, v)



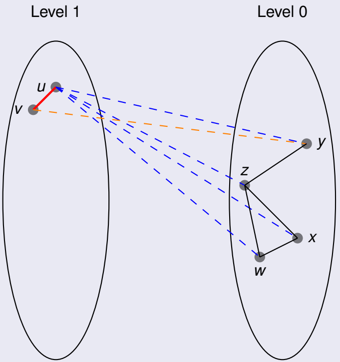
Epoch at level 1: Start

- Inv2:The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched

$\longrightarrow$ Add (u, v) to M

$\longrightarrow$ Move v to level 1

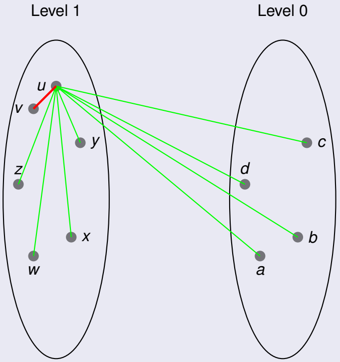
$\longrightarrow$ Make v the *owner* of all its adjacent edges



Start of an epoch at level 1 = $O(\sqrt{n})$

Epoch at level 1: End

Edge (u, v) is deleted

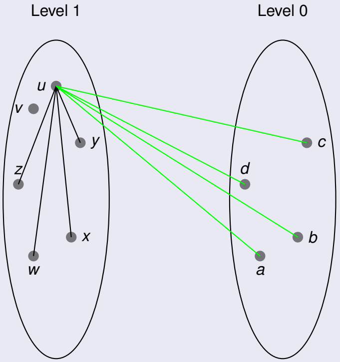


Epoch at level 1: End

$\longrightarrow$ Give up the ownership of the edges at level 1

$\longrightarrow$ If u is still the owner of $\geq \sqrt{n}$ edges

the the procedure is same as in the previous slide



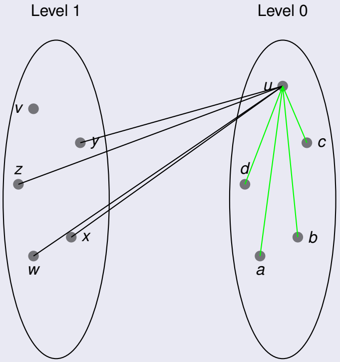
Epoch at level 1: End

⟶ Else u moves to level 1 and starts level 0 epoch there

⟶ But the degree of vertex a in $G[V_0]$ increses by 1

and may move to level 1

⟶ All such vertex move up and start a epoch at level 1



End of an epoch at level 1 = $O(n)$

Epochs	Start	End	Total cost	Total number of Epochs	Total computation cost
Level 0	$O(1)$	$O(\sqrt{n})$	$O(\sqrt{n})$	T	$O(T\sqrt{n})$
Level 1	$O(\sqrt{n})$	$O(n)$	$O(n)$		

Epochs	Start	End	Total cost	Total number of Epochs	Total computation cost
Level 0	$O(1)$	$O(\sqrt{n})$	$O(\sqrt{n})$	T	$O(T\sqrt{n})$
Level 1	$O(\sqrt{n})$	$O(n)$	$O(n)$	$O(T/\sqrt{n})$	$O(T\sqrt{n})$

Epochs	Start	End	Total cost	Total number of Epochs	Total computation cost
Level 0	$O(1)$	$O(\sqrt{n})$	$O(\sqrt{n})$	T	$O(T\sqrt{n})$
Level 1	$O(\sqrt{n})$	$O(n)$	$O(n)$	$O(T/\sqrt{n})$	$O(T\sqrt{n})$

The algorithm has $O(\sqrt{n})$ update time.

Balance

In the two level algorithm, we define a threshold $\alpha(n)$ for a vertex to move from level 0 to level 1

- The update time at level 0 is $O(\alpha(n))$
- The update time at level 1 is $O(n/\alpha(n))$
- Both the update time are same when $\alpha(n) = \sqrt{n}$

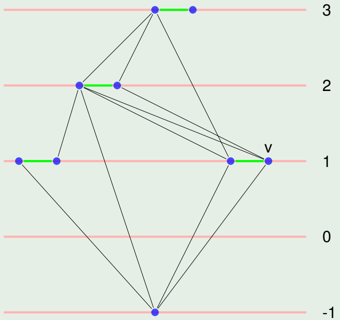
Speeding up the algorithm

- Try to minimize the gap between the number of edges a vertex can own in an epoch and the number of edges it owned at the moment it created the epoch
- This ratio is $\sqrt{n}$ in 2-level algorithm

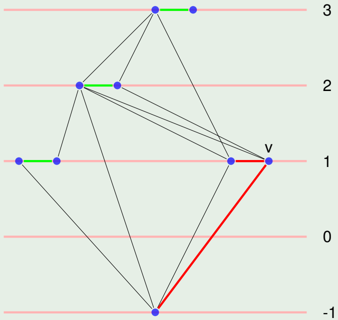
An overview of the log n-level algorithm

- Maintain log n levels
- When a vertex creates an epoch at level i, it would own at least 2^i edges, and during the epoch it will be allowed to own at most 2^{i+1} edges
- The ratio is a constant
- In implementing these ideas, an extra factor of $O(\log n)$ comes up due to the log n level hierarchy

Example

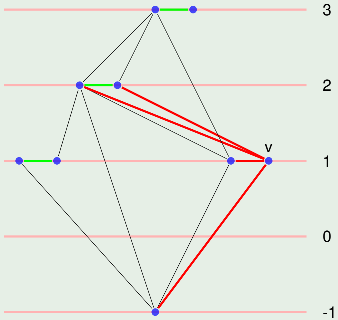


Example



The number of edges v can own if it rises to level 2 = 2 $< 2^2$

Example



The number of edges v can own if it rises to level 3 = 4 $< 2^3$

Open Problem

- There exists an algorithm for maximal matching in $O(\log n)$ update time but is there a algorithm which maintains $c-approximation$ of maximum matching where $c < 2$
- Is there any combinatorial algorithm which maintains maximum matching in $o(m)$ time

Questions?