Kruskal's Algorithm for MST

Start w/ vertex set
KRUSKAL'S ALGORITHM for MST

SORT edges
Kruskal's Algorithm for MST

Are endpoints in different components?

Scan sorted list
Kruskal’s Algorithm for MST

Yes. so add edge to MST

Why?
Kruskal's Algorithm for MST

Suppose this edge is not in MST

Hypothesis: endpoints in different components

Eventually we must link the endpoints with some path

Some edge will be scanned later. Higher weight
Kruskal's Algorithm for MST

Suppose this edge is not in MST

Hypothesis: endpoints in different components

Eventually we must link the endpoints with some path

Some edge will be scanned later. 

Higher weight
Kruskal’s Algorithm for MST

Resume example
just added edge & merged
2 components
Kruskal's Algorithm for MST

Continue scanning endpoints in different components
Kruskal's Algorithm for MST

add to MST & merge
Kruskal's Algorithm for MST

1. Sort edges by weight.
2. Add edges to MST, maintaining several components (forest) of trees.
3. Merge components if adding an edge does not create a cycle.

Example:
- Sort edges: (10, 11), (10, 31), (10, 47), ...
- Add edges: (10, 11), (10, 31), ...
- Merge components: if adding (10, 31) does not create a cycle, merge components.
Kruskal's Algorithm for MST

must add & merge
KRUSKAL'S ALGORITHM for MST

Finally, example of endpoints in same component.

\[ \text{IGNORE edge} \]
Kruskal's Algorithm for MST

...DONE

why?
KRUSKAL’S ALGORITHM for MST

DONE
Kruskal's Algorithm for MST

1) Make forest of vertices \( O(v) \)
2) Sort edges \( O(E \log E) \)
3) For each edge \( \cdots O(E) \) ?
   - Check endpoints (same component?)
   - No? Add edge \( O(1) \)
   - Merge.
Given an edge (~ two vertices) & a forest

CHECK ENDPOINTS
(same component?)

→ No? Add edge.
MERGE

→ use a variable for each vertex \( O(1) \)

→ change variable for all vertices in component \( O(v) \)
basic UNION-FIND

Given an edge (~ two vertices) & a forest

- **CHECK ENDPOINTS** (same component?)
  - **No? Add edge.**
  - **MERGE**

... for an application that will do this many times

- **Kruskal's algo**
  - $O(E) : O(V^2)$
Store each component as a linked list

Every node points to one representative.

"Find/add/delete : O(1)
we don't need these
NOT search.
Assume pointer to node
Store each component as a linked list

Every node points to one representative.

Find/add/delete: \(O(1)\)

Query "same component":
check representatives: \(O(1)\)

Merge: looks like \(O(n)\)

\(n = |V|\)
Store each component as a linked list

Every node points to one representative.

Find/add/delete: $O(1)$

Query "same component": check representatives: $O(1)$

Merge: looks like $O(n)$

Why?
Store each component as a linked list

Every node points to one representative.

Find/add/delete: $O(1)$
Query "same component": check representatives: $O(1)$
Merge: looks like $O(n)$

WHY?
Store each component as a linked list

Every node points to one representative.

Find/add/delete: $O(1)$

Query "same component":
check representatives: $O(1)$

Merge: looks like $O(n)$
How many times can an item merge?

(= how many times can an item change representatives?)

\[ n = \sum n_i \]

Component size doubles each time

Component cost of \( n \) merges:

\[ O(n \log n) \]
basic UNION-FIND

Given an edge (~ two vertices) & a forest

CHECK ENDPOINTS
(same component?)
→ O(1)

↓ No? Add edge.
MERGE
→ O(1)

ODE
worse case
O(VlogV) total
↑ many times

... for an application that will do this

Kruskal’s algo

O(E) checks
O(V) merges
Kruskal's Algorithm for MST $\rightarrow O(E \log V)$

1) make forest of vertices $O(V)$
2) sort edges $O(E \log E) = O(E \log V)$
3) for each edge
   CHECK ENDPOINTS (same component?)
   \[ \begin{cases} O(E) \text{ total} \\ \text{No? Add edge, MERGE} \end{cases} \]
   $O(V \log V)$ total $= O(E \log V)$

$E \geq V+1$
assuming connected $G$