Kruskal's Algorithm for MST

Start w/ vertex set
Kruskal's Algorithm for MST

Sort edges

Diagram showing a network with labeled edges and a spanning tree marked with red lines.
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

endpoints in different components

continue scanning
Kruskal's Algorithm for MST

add to MST
& merge
Kruskal’s Algorithm for MST

- Add to MST
- Merge

Maintaining several components

A forest of trees
Kruskal's Algorithm for MST

must add & merge
Kruskal's Algorithm for MST

Finally, example of endpoints in same component.

Ignore edge
KRUSKAL'S ALGORITHM for MST

...DONE why?
KRUSKAL'S ALGORITHM for MST
Kruskal's Algorithm for MST

1) make forest of vertices $O(v)$
2) sort edges $O(E \log E)$
3) for each edge $\ldots O(E \cdot ?)$
   CHECK ENDPOINTS
   (same component?)
   $\downarrow$ No? Add edge. $O(1)$
   MERGE.
Given an edge (~ two vertices) & a forest

- **CHECK ENDPOINTS**
  - (same component?)
  - \( \rightarrow \) No? Add edge.
  - \( \rightarrow \) Yes? 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

$n = |V|$

Every node points to one representative.

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.

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.

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.

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

Merge: looks like $O(n)$
\[ \sum n_i = n \]

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

\[ \text{merge}(j,k) = O(\min(n_j, n_k)) \]

\[ \text{Component size doubles each time} \]

\[ \text{total cost of } n \text{ merges: } O(n \log n) \]
basic UNION-FIND

Given an edge (~ two vertices) & a forest

CHECK ENDPOINTS
(same component?)

$\rightarrow O(1)$

$\rightarrow O(1)$

MERGE

$\{ O(V) \text{ worst case} \}
       \uparrow
O(V\log V) \text{ total}
       \uparrow
       \uparrow
\rightarrow O(E) \text{ checks}
\rightarrow O(V) \text{ merges}$

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

Kruskal’s algo
Kruskal's Algorithm for MST → $O(E \log V)$

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?)
     - No? Add edge, MERGE
     - Yes? Skip
   → $O(E)$ total

$O(V \log V)$ total = $O(E \log V)$

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