VE281

Data Structures and Algorithms

Minimum Spanning Trees and Sorting

Review

- Shortest Path Problem for Weighted Graph
  - Dijkstra's algorithm: grow the set of nodes to which we know the shortest path.

- Minimum Spanning Tree
  - Prim's algorithm: grow the set of nodes we have added to the MST.

Outline

- Prim’ s Algorithm for MST
- Kruskal’ s Algorithm for MST
- Sorting

Prim’s Algorithm

- Separate V into two sets:
  - T: the set of nodes that we have added to the MST.
  - T': those nodes that have not been added to the MST, i.e., $T' = V - T$.

- Prim’s algorithm initially sets T as empty and T' as V. The algorithm moves one node from T' to T in each iteration. After the last iteration, $T = V$ and we have constructed the MST.

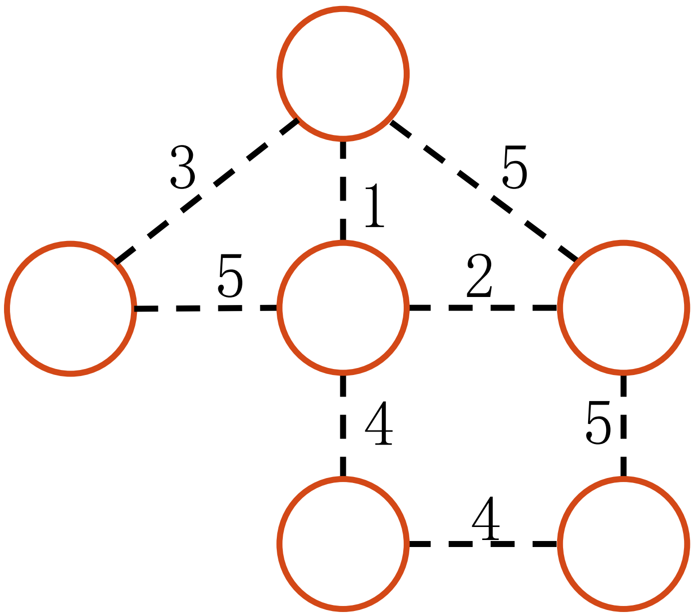
Prim's Algorithm

- For each node $v \in T'$, we keep a measure $D(v)$, storing the **smallest weight** of any edge that connects any node in T to v. We also keep previous node $P(v)$ for each node v to record the edges chosen in the MST.

1. Arbitrarily pick one node s. Set $D(s) = 0$. For any other node v, set $D(v)$ as infinite and $P(v)$ as unknown.
2. While $T' \neq \emptyset$
   1. Choose node v in T' such that $D(v)$ is the smallest. Remove v from the set T'.
   2. For each of v's neighbors u that is still in T', if $D(u) > w(v, u)$, then update $D(u)$ as $w(v, u)$ and $P(u)$ as v.

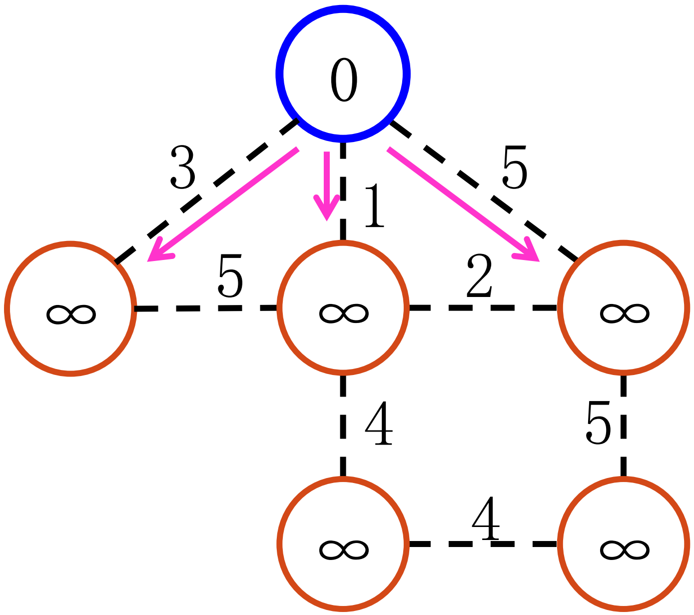
Prim’s Algorithm

Example



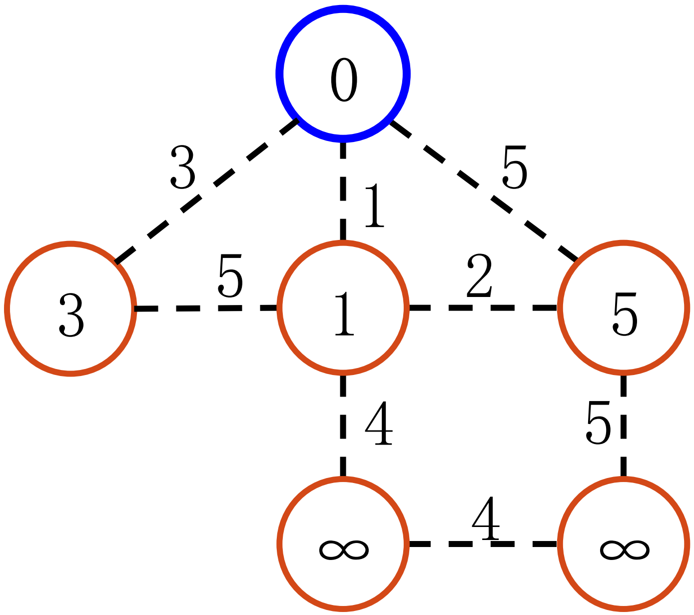
Prim’s Algorithm

Example



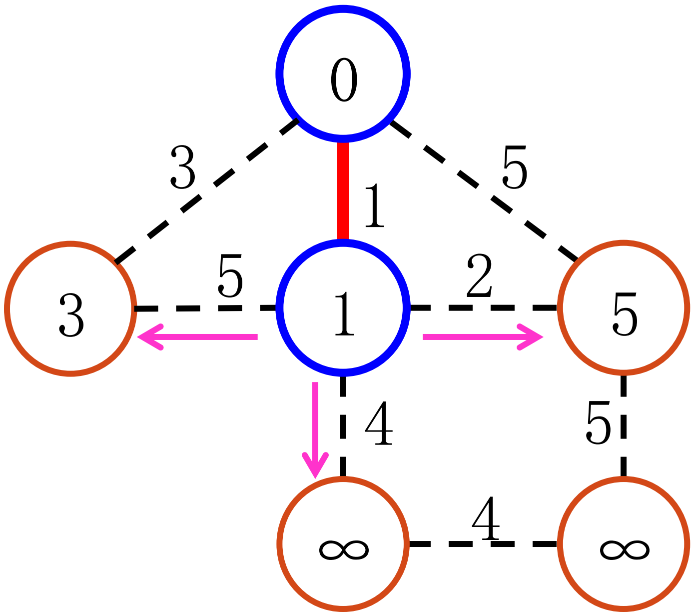
Prim’ s Algorithm

Example

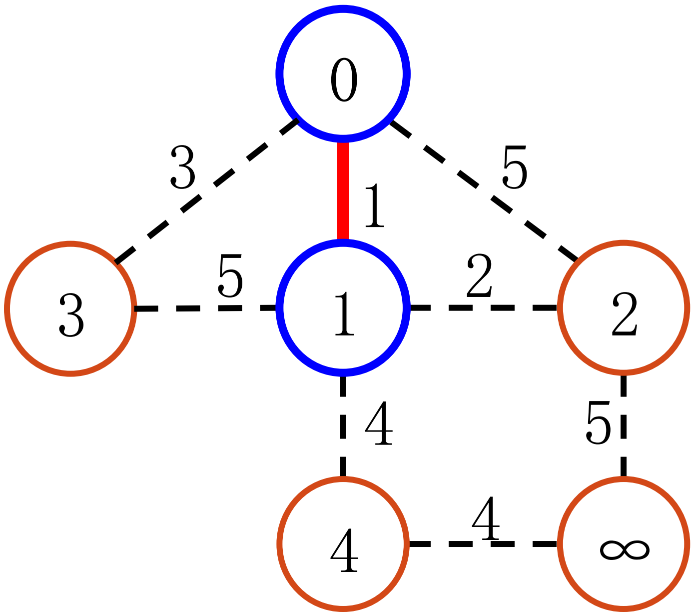


Prim’s Algorithm

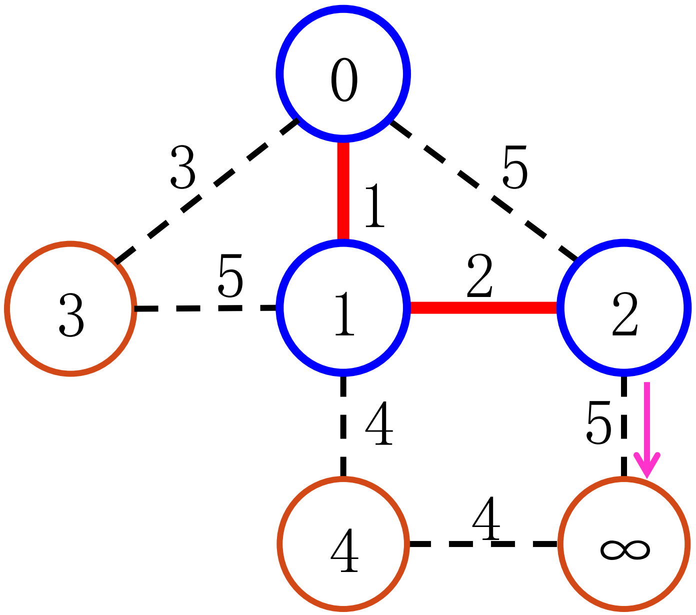
Example



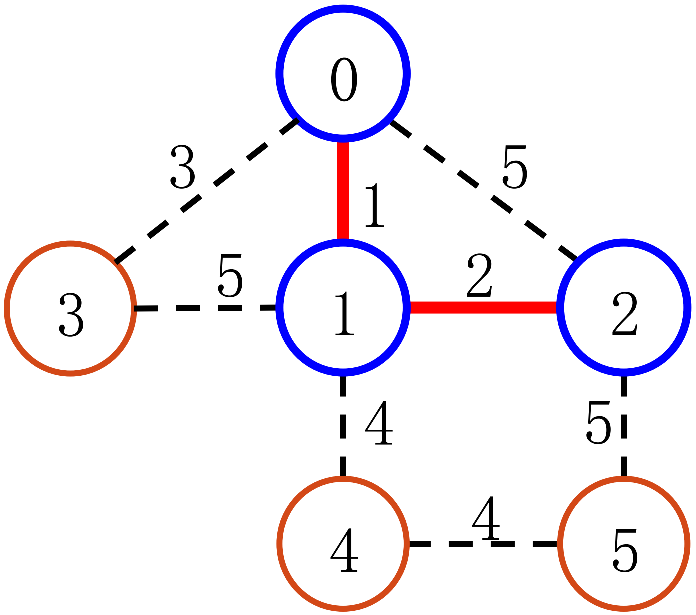
Prim’s Algorithm
Example



Prim’s Algorithm
Example

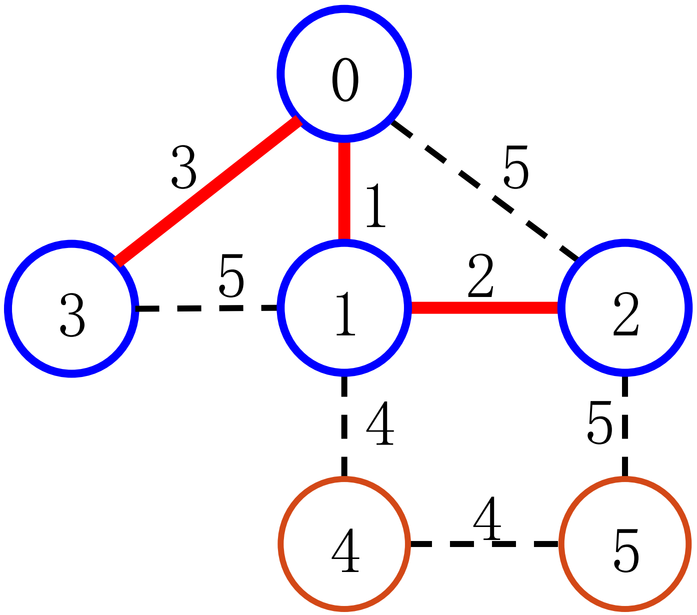


Prim’s Algorithm
Example



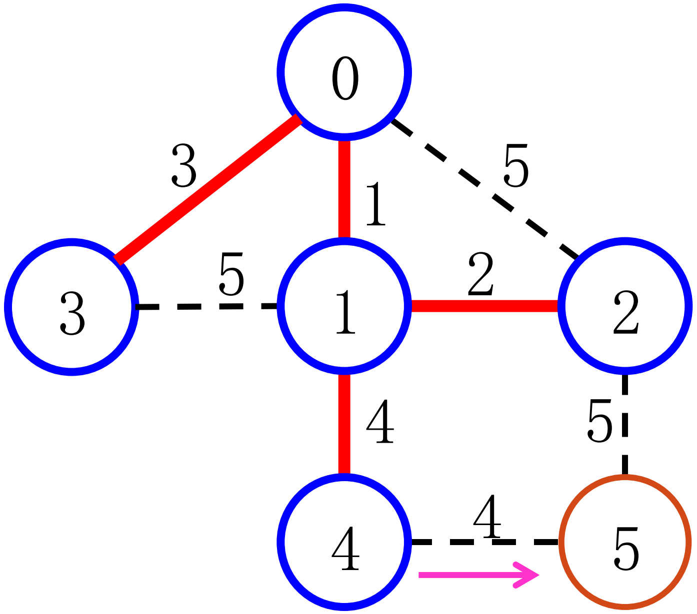
Prim’s Algorithm

Example



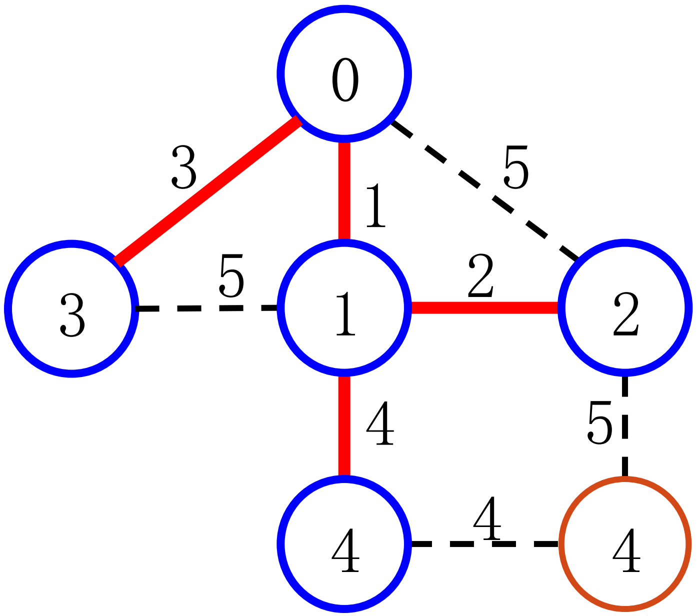
Prim' s Algorithm

Example

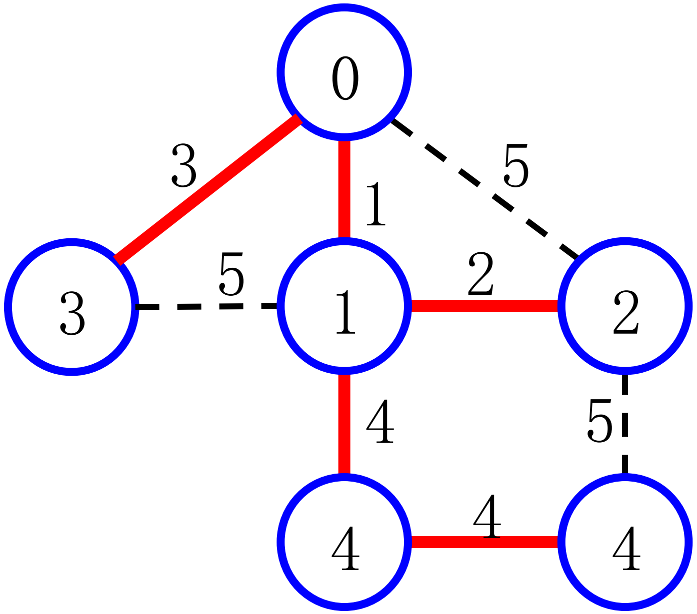


Prim’s Algorithm

Example



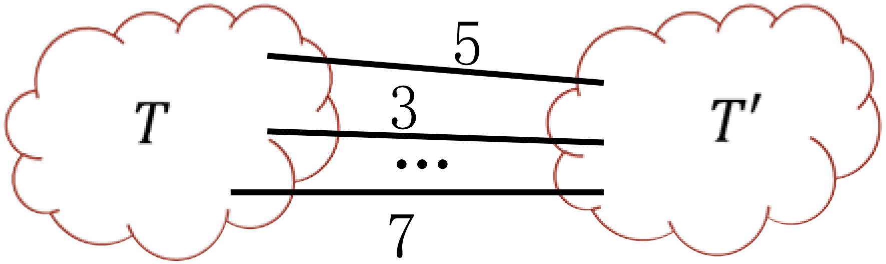
Prim’s Algorithm
Example



Prim's Algorithm

Justification

- Let T and T' be a partition of V. In a spanning tree, there must exist at least one edge that connects one node in T to another node in T'.
  - Otherwise, it is not a spanning tree.



- Prim's algorithm grows set T and each time greedily picks the edge with the smallest weight that connects a node in T to a node in T'. It ensures:
  1. All nodes are connected and there are no cycles, i.e., a tree.
  2. The sum of all edge weights is minimal.

Prim's Algorithm

Time Complexity

- Number of times to find the smallest $D(v)$: $|V|$.
  - Cost? $\boxed{\text{Linear scan: } O(|V|)\text{; Priority queue: } O(\log|V|)}$
- Total number of times to update the neighbors: $|E|$.
  - Since each neighbor of each node could be potentially updated.
  - Cost? $\boxed{\text{Linear scan: } O(1)\text{; Priority queue: } O(\log|V|)}$

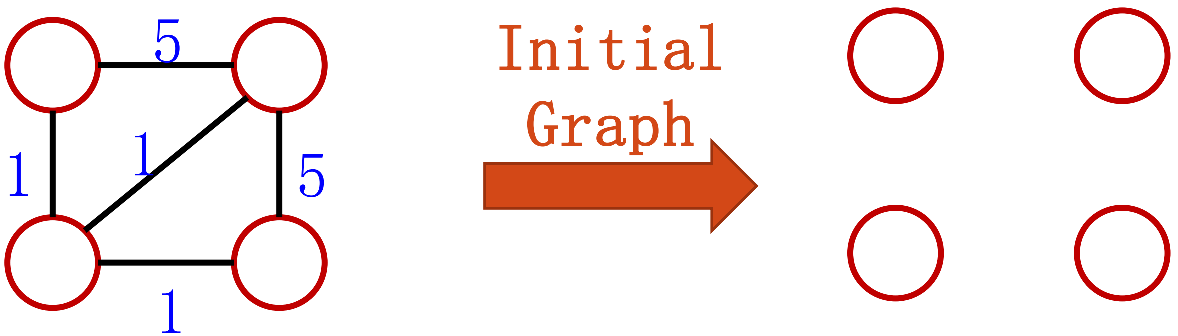
- Total time complexity
  - Linear scan: $O(|E| + |V|^2) = O(|V|^2)$.
  - Priority queue:
    $O(|V|\log|V| + |E|\log|V|) = O(|E|\log|V|)$.

Outline

- Prim’s Algorithm for MST
- Kruskal’s Algorithm for MST
- Sorting

Kruskal’s Algorithm

- Start with a graph containing $|V|$ nodes and no edges

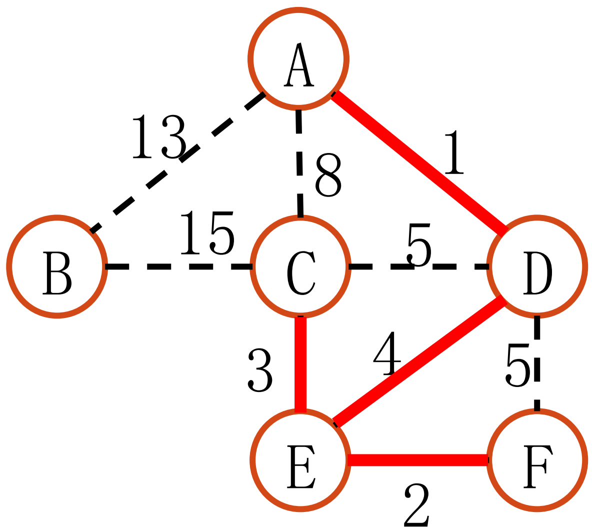


- This initial graph can be viewed as a **forest** of trees.
  - Each tree has only a single node.
- Main idea: repeatedly add the edge with the **smallest weight** that **does not cause a cycle** until no such edges exist.
  - Each added edge performs a union on two trees in the forest.
  - After adding $|V| - 1$ edges, there is only one tree. This tree is the MST.

Kruskal's Algorithm

Example

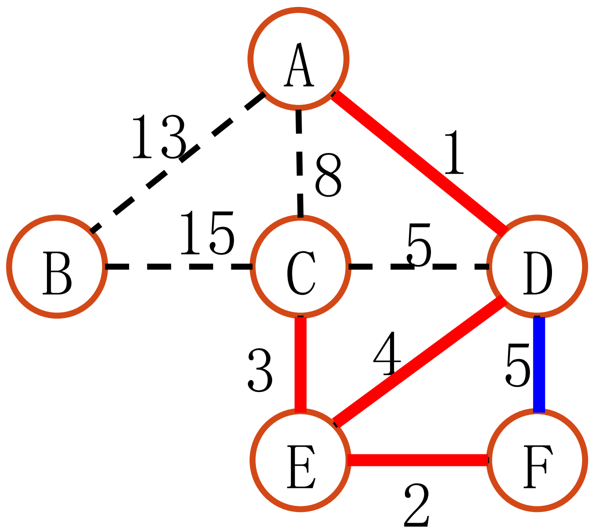
Repeatedly add the edge with the **smallest weight** that does not cause a cycle until no such edges exist.



Kruskal’s Algorithm

Example

Repeatedly add the edge with the **smallest weight** that **does not cause a cycle** until no such edges exist.



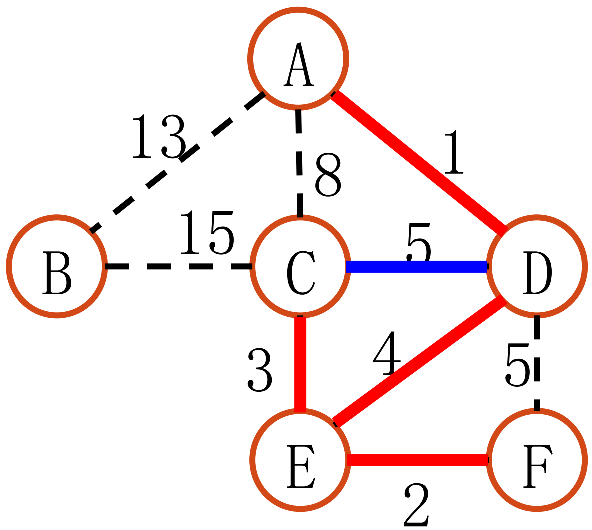
The next edge with the smallest weight is (D,

However, adding it causes a cycle. So it is discarded.

Kruskal's Algorithm

Example

Repeatedly add the edge with the **smallest weight** that **does not cause a cycle** until no such edges exist.



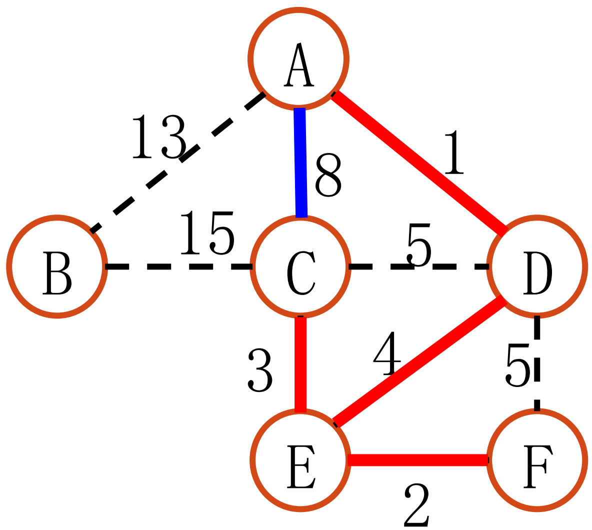
The next edge with the smallest weight is (C,

However, adding it causes a cycle. So it is discarded.

Kruskal's Algorithm

Example

Repeatedly add the edge with the **smallest weight** that **does not cause a cycle** until no such edges exist.



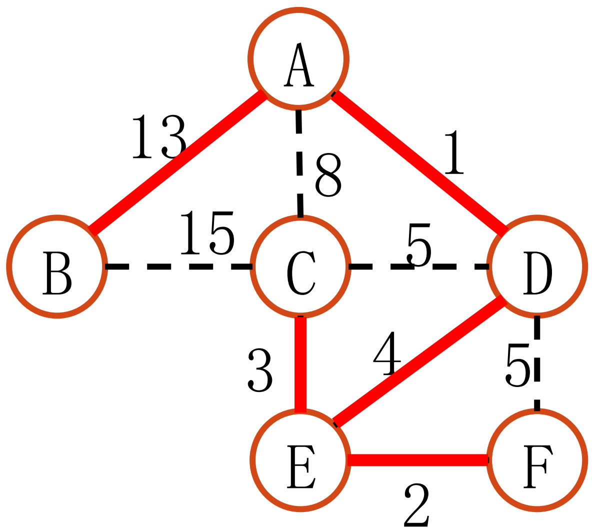
The next edge with the smallest weight is (A,

However, adding it causes a cycle. So it is discarded.

Kruskal's Algorithm

Example

Repeatedly add the edge with the **smallest weight** that does not cause a cycle until no such edges exist.



The next edge with the smallest weight is (A,

MST construction done.

Detecting Cycles

- Not simple.
- Connected nodes form a **component**.
- Detecting cycle: an edge (u, v) causes a cycle if nodes u and v are in the same component.
- If the edge does not cause a cycle, we add the edge and make union on the two different components connected by the edge.
  - Update the set of components for later detecting cycle purpose.

Kruskal’s Algorithm

Implementation and Time Complexity

- Sorting the edges by weights
  - Time complexity: $O(|E| \log |E|)$.
- Detecting cycle. If no cycle, add edge and merge two trees.
  - Time complexity: $O(\log |V|)$. (Not covered)
  - In the worst case, we detect cycles for all edges. The time complexity is $O(|E| \log |V|)$.

- Since $|E| = O(|V|^2)$, the total running time is $O(|E| \log |V|)$.

Outline

- Prim’s Algorithm for MST
- Kruskal’s Algorithm for MST
- Sorting

Sorting

- Given array A of size N, reorder A so that its elements are in order.
  - "In order" with respect to a consistent comparison function, such as "$\leq$" or "$\geq$".

- Sorting order
  - Ascending order
  - Descending order
- Unless otherwise specified, we consider sorting in ascending order.

Characteristics of Sorting Algorithms

- Average case time complexity
- Worst case time complexity
- Space usage: **in place** or not?
  - **in place**: requires $O(1)$ additional memory.
  - Don't forget the stack space used in recursive calls.
- **Stability**: whether the algorithm maintains the relative order of records with equal keys.
  - Usually there is a secondary key whose ordering you want to keep. Stable sort is thus useful for sorting over multiple keys.

(4, b), (3, e), (3, b), (5, b)

(3, e), (3, b), (4, b), (5,

Sort on the first number

Stable!

Types of Sorting Algorithms

- Sorting algorithms can be classified as comparison sort and non-comparison sort.

- Comparison sort: each item is compared against others to determine its order.

- Non-comparison sort: each item is put into predefined “bins” independent of the other items presented.
  - No comparison with other items needed.
  - It is also known as distribution-based sort.

Types of Sorting Algorithms

- General types of comparison sort
  - Insertion-based: insertion sort, shell sort
  - Selection-based: selection sort, heap sort
  - Exchange-based: bubble sort, quick sort
  - Merging-based: merge sort

- Non-comparison sort:
  counting sort, bucket sort, radix sort

Insertion Sort

- **A[0]** alone is a sorted array.
- For **i=1** to **N-1**
  - Insert **A[i]** into the appropriate location in the sorted array **A[0], …, A[i-1]**, so that **A[0], …, A[i]** is sorted.
  - To do so, save **A[i]** in a temporary variable **t**, shift sorted elements greater than **t** right, and then insert **t** in the gap.

$O(N^2)$

- Time comlexity?

Yes. O(1) additional memory.

- In place?
- Stable?
  - Yes, because elements are visited in order and equal elements are inserted after its equals.

Insertion Sort

Best Case Time Complexity

- Separate V into two sets:
  - T: the set of nodes that we have added to the MST.
  - T': those nodes that have not been added to the MST, i.e., $T' = V - T$.

- Prim's algorithm initially sets T as empty and T' as V. The algorithm moves one node from T' to T in each iteration. After the last iteration, $T = V$ and we have constructed the MST.

Selection Sort

- For **`i=0`** to **`N-2`**
  - Find the smallest item in the array **`A[i], …, A[N-1]`**. Then, swap that item with **`A[i]`**.
- Finding the smallest item requires **linear search**.
- Time complexity?
  - $O(N^2)$
- In place?
  - Yes. O(1) additional memory.
- Stable?
  - No. (3, e), (3, b), (2, a) → (2, a), (3, b), (3, e)