CPCFI

Lecture: MINIMUM SPANNING TREE 1

Unit: 6 - GRAPHS 1

Instructor: NÉSTOR

0. MINIMUM SPANNING TREE

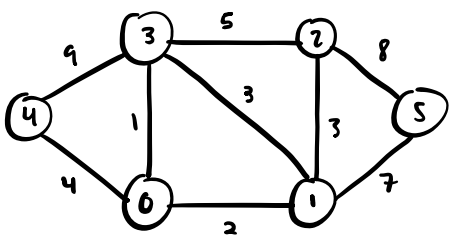
- SPANNING TREE: SET OF EDGES SUCH THAT ANY VERTEX CAN REACH ANY OTHER BY EXACTLY ONE PATH

- MINIMUM SPANNING TREE: SPANNING TREE WITH THE SMALLEST WEIGHT POSSIBLE

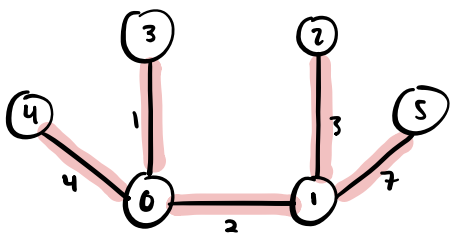
⇓

CONTAINS N-1 EDGES AND N VERTICES

→ THE MST DOESN'T EXIST IF THE GRAPH IS DISCONNECTED



UNDIRECTED WEIGHTED GRAPH G1



MINIMUM SPANNING TREE OF G1

MOTIVATION PROBLEM:

"GIVEN N CITIES AND M ROADS EACH WITH A COST OF BUILDING, WE WANT TO BUILD A SET OF ROADS SUCH THAT WE CAN GET FROM EACH TO ANY CITY AND THE COST OF CONSTRUCTION IS MINIMAL"

- PROPERTIES:

  - THE MST OF A GRAPH IS UNIQUE IF THE WEIGHT OF ALL EDGES ARE DIFFERENT

  - MAXIMUM SPANNING TREE CAN BE OBTAINED JUST BY CHANGING THE SIGNS OF THE WEIGHTS OF THE EDGES AND APPLYING A MINIMUM SP. TR. ALGORITHM

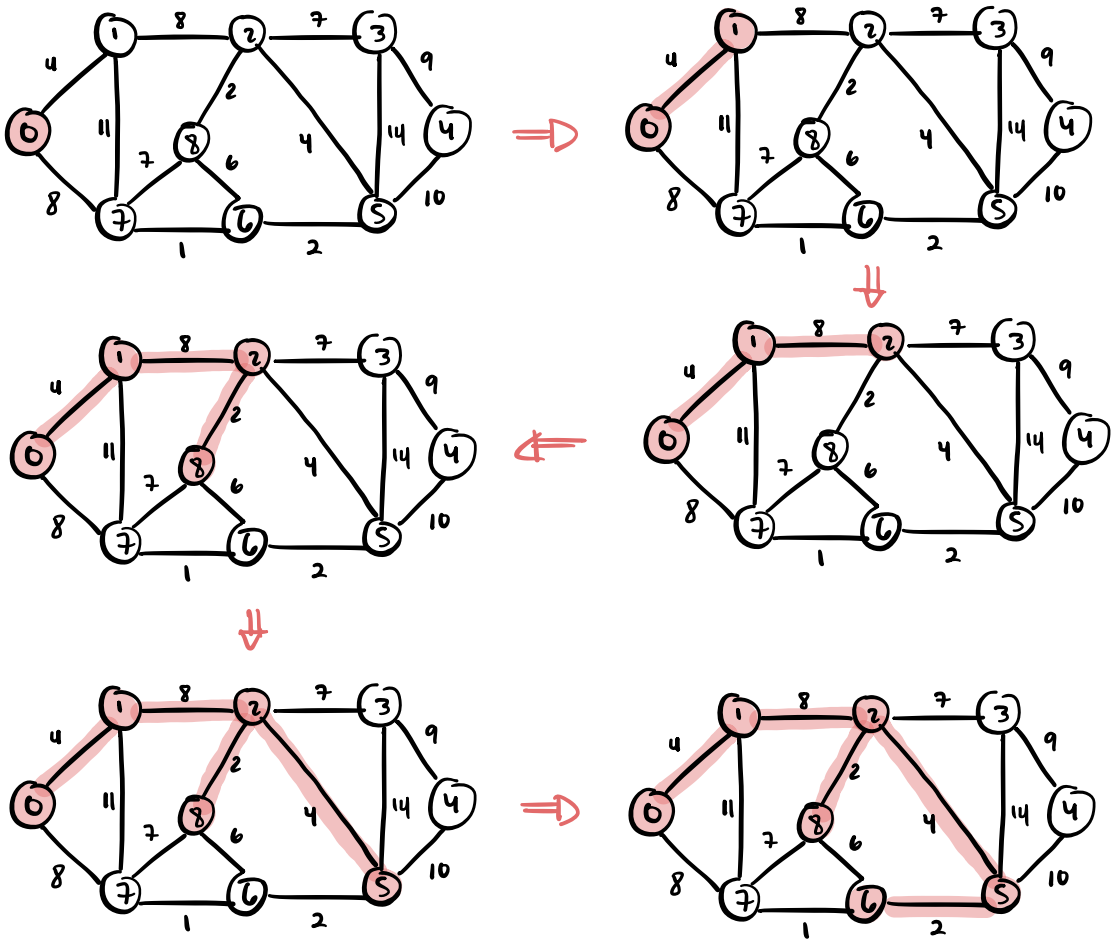
I. PRIM'S ALGORITHM

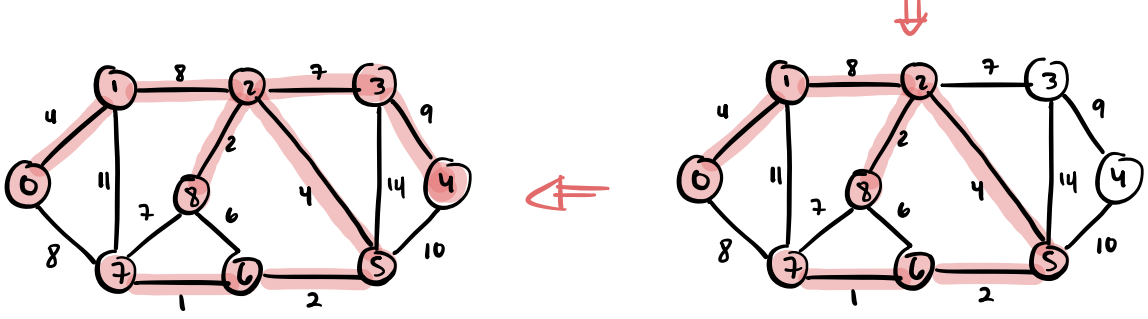
[IDEA]

- THE MST IS BUILT GRADUALLY BY ADDING EDGES ONE AT A TIME
- CHOOSE A RANDOM VERTEX AND ADD THE EDGE WITH SMALLEST WEIGHT TO THE MST
- SELECT AND ADD THE EDGE WITH THE MINIMUM WEIGHT THAT HAS ONE END IN AN ALREADY SELECTED VERTEX AND THE OTHER END IN AN UNSELECTED VERTEX
- REPEAT UNTIL V-1 EDGES OR N EDGES

" THE TREE STARTS FROM AN ARBITRARY ROOT VERTEX r AND GROWS UNTIL THE TREE SPANS ALL VERTICES IN G "

→ AT EACH STEP WE ONLY ADD EDGES THAT CONTRIBUTE TO THE MINIMUM AMOUNT POSSIBLE TO THE TREE'S WEIGHT





[IMPLEMENTATION]

1. DENSE GRAPHS $O(n^2)$ → $\approx$ MAX # OF EDGES

INITIALIZATION
- ADJ. MATRIX (n^2 SPACE) → ADJ[i][j] = WEIGHT OF EDGE (i,j)
- MIN_EDGE VECTOR → min_edge[i]
  - WEIGHT (0 DEFAULT) = EDGE WITH MIN WEIGHT FOR VERTEX i
  - NODE TO (-1 DEFAULT)
- FOR EACH VERTEX i : $O(n)$
  - SELECT NEXT VERTEX v WITH MINIMUM EDGE WEIGHT AMONGST ALL REMAINING NOW SELECTED VERTICES $O(n)$
  - IF MIN_EDGE[v].WEIGHT = ∞ : } GRAPH IS DISCONNECTED
    - PRINT ("NO MST")
    - RETURN
  - MARK VERTEX v AS SELECTED
  - TOTAL WEIGHT += MIN_EDGE[v].WEIGHT
  - UPDATE MINIMUM WEIGHT EDGE FOR ALL VERTICES $O(n)$
- PRINT (TOTAL_WEIGHT)

2. SPARSE GRAPHS $O(m \log n)$ → $\approx$ MIN # OF EDGES

- SAME ALGORITHM REMAINS THE SAME BUT USES A SET TO FIND THE MINIMUM WEIGHTED EDGE IN $O(\log n)$

II. KRUSKAL'S ALGORITHM

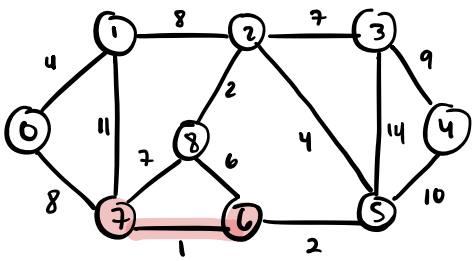
- INITIALLY PLACES ALL THE NODES OF THE ORIGINAL GRAPH ISOLATED FROM EACH OTHER
- GRADUALLY INTERLOVES THESE TREES COMBINING IN EACH ITERATION ANY TWO OF ALL THE TREES WITH SOME EDGE OF THE ORIGINAL GRAPH

} ↓ FOREST OF SINGLE NODE TREES

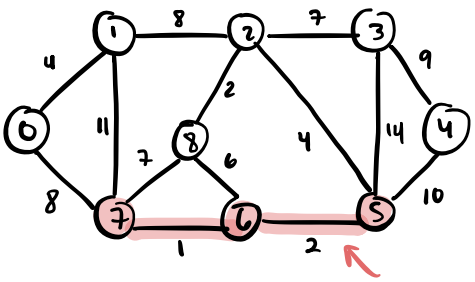
---

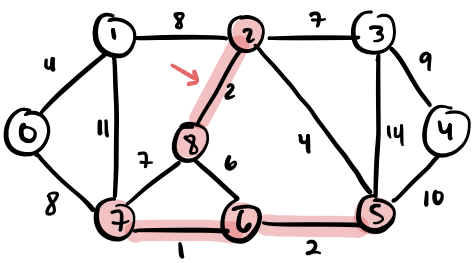
(1) INITIALLY, ALL EDGES HAVE SORTED BY WEIGHT IN ASCENDING ORDER

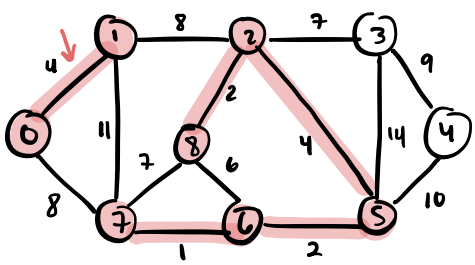
(2) PICK ALL EDGES FROM FIRST TO LAST AND IF THE NODES OF THE CURRENT EDGE BELONGS TO DIFFERENT TREES, THEY ARE COMBINED

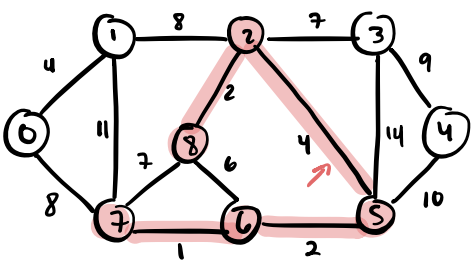


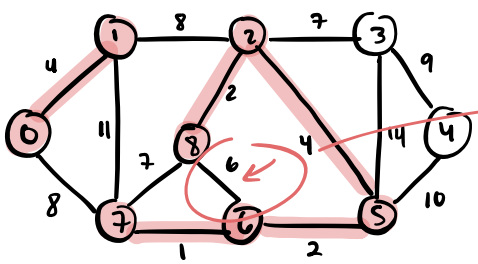
START FROM THE LEAST WEIGHTED EDGE AND MERGE THE TREES IF THE NODES BELONG TO DIFFERENT TREES



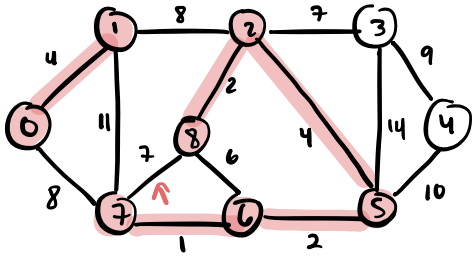
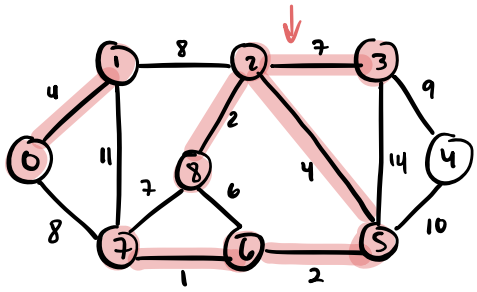
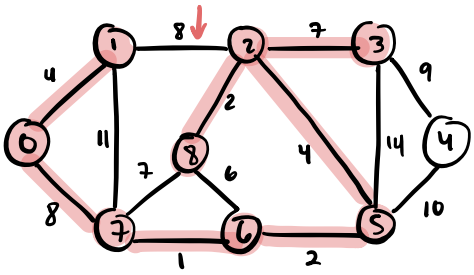
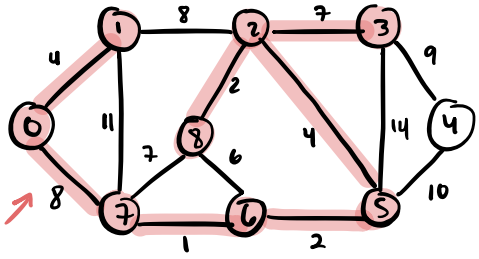
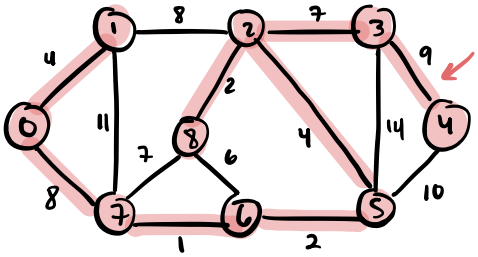
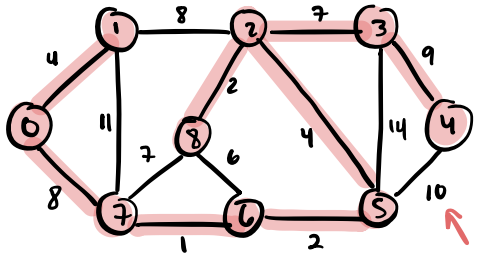
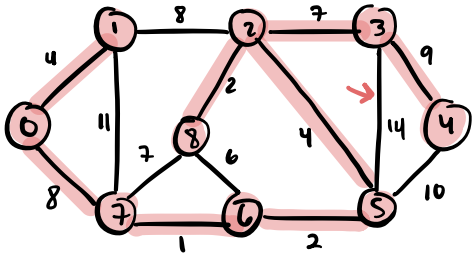
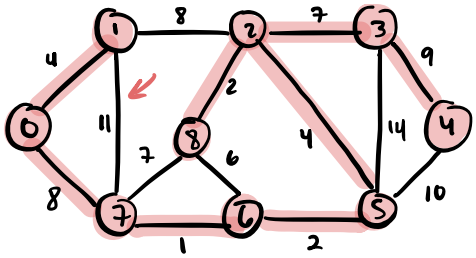








WE DON'T ADD EDGE (8,6) EVEN THOUGH IT'S THE NEXT SMALLEST WEIGHTED EDGE BECAUSE (8) AND (6) BELONG TO THE SAME TREE

- TIME COMPLEXITY: $O(m \log m + n^2)$

III KRUSKAL'S ALGORITHM WITH DSU

- TIME COMPLEXITY: $O(m \log n)$

1.- SORT ALL EDGES: $O(m \log n)$

2.- BUILD A TREE FOR EACH VERTEX: $O(n)$ → MAKE_SET(v)

3.- DETERMINE IF TWO NODES BELONG TO THE SAME TREE: $O(\log n)$ FIND_SET(v)

4.- MAKESET TAKES: $O(1)$ → UNION-SETS (a, b)