12/03/2024

# LEZ 3: MST = MINIMO SPANNING TREE

→ ANCORA ALGO. GREEDY

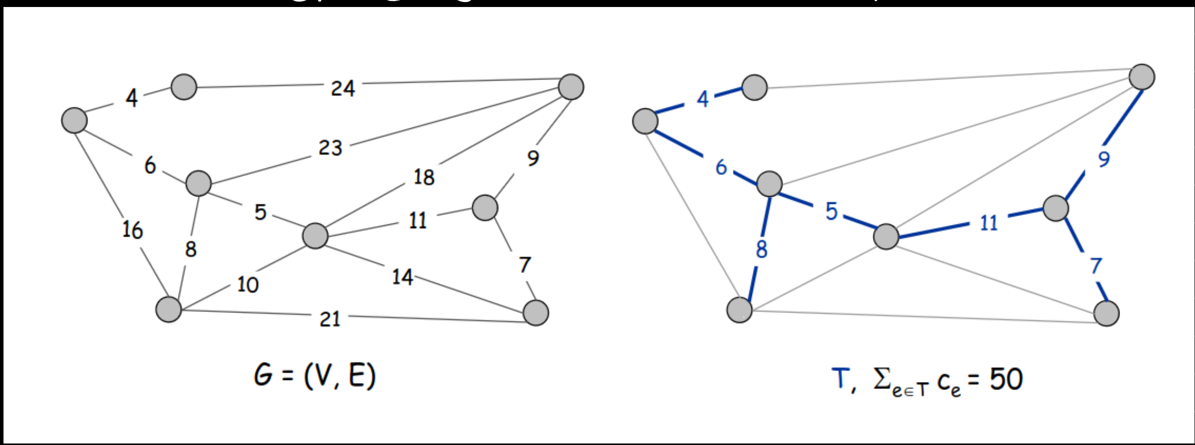
MST = PROBLEMA DI OTTIMIZZAZIONE
↳ MINIMO ALBERO RICOPRENTE.

## DESCRIZIONE PROBLEMA

- INPUT = GRAFO G NOT ORIENTED, CONNESSO E PESATO.
- OUTPUT = SUBSET DEGLI ARCHI $T \subseteq E$ (E ARCHI DI G) t.c. G SIA ANCORA CONNESSO E LA SOMMA DEI PESI OF ARCHI SIA MINIMO.

ESEMPIO:

GRAFO G MST




## FORMALIZZAZIONE PROBLEMA:

- INPUT = GRAFO G = (V, E) ¬ORIENTATO, PESATO E CONNESSO
- SOL. FLESSIBILE = ALBERO RICOPRENTE T (CHE CONTIENE I NODI DI G, TUTTI CONNESSI), t.c. EDGE DI $T \subseteq E$
- MISURA = PESO DI TUTTI GLI ARCHI DI T
(DA MINIMIZE)

# APPLICAZIONI DEL PROBLEMA

- COSTRUZIONI URBANE (STRADE, COLL. TUBATURE, ETC.)
- NETWORK DESIGN (COLL. CORRENTE ELETTRICA)
- ALGO APPROSSIMATIVI PER PROBLEMI NP-DIFFICILI.

# CREAZIONE ALGORITMO

prima alcune propietà:

- IL MST NON È UNICO IN GENERALE

  MA SE ASSUMIAMO PESI DISTINTI LA SOLUZIONE È UNICA.

  DA DIMOSTRARE -> X CASA

# ALGORITMI

① **Kruskal's algorithm.** Start with T = $\phi$. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

② **Reverse-Delete algorithm.** Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

③ **Prim's algorithm.** Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

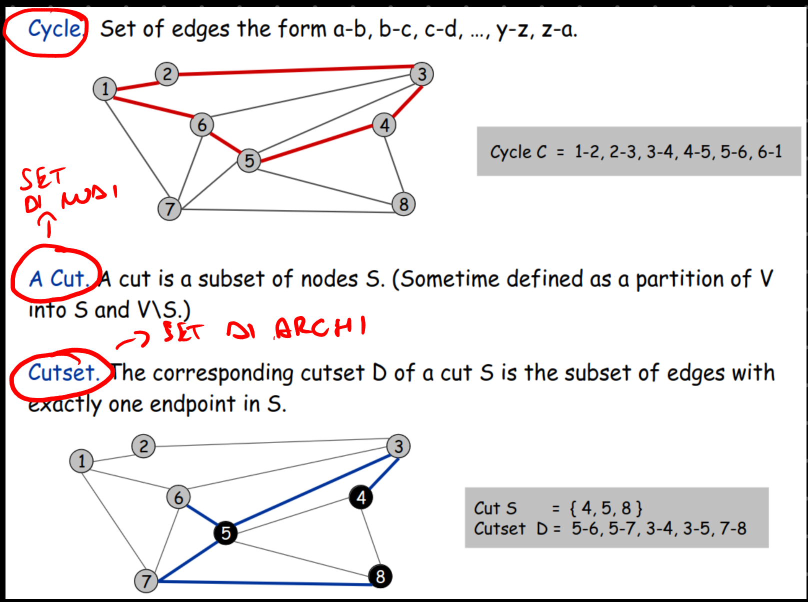
**Remark.** All three algorithms produce an MST.

OGGI VEDREMO IN DETTAGLIO KRUSKAL.

ma prima di vedre nel dettaglio l'algoritmo, dobbiamo studiare delle propietà forti per la dimostrazione di correttezza

# PROPIETÀ GRAFO: CICLI E TAGLI

CONSIDERIAMO PRIMA QUESTI CONCETTI:

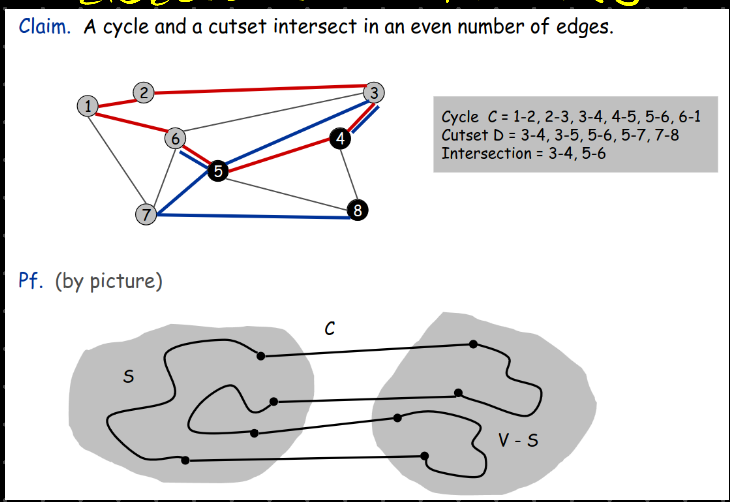



NODI DI G.

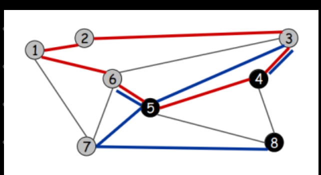
→ "DIVIDO" IN 2 SET, S e V\S

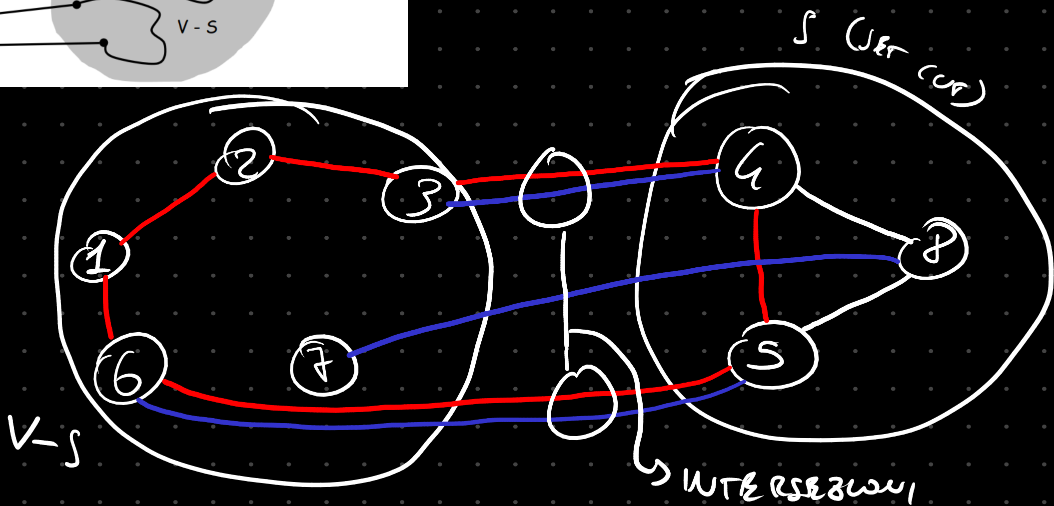
→ $\subseteq E$ (ARCHI), t.c. IN OGNI ARCO 1 NODO $\in$ INSIEME CUT L'ALTRO NODO $\notin$ //

## INTERSEZIONE CICLO-TAGLIO → QUALSIASI




|{ARCHI DEL CICLO}| $\cap$ |{ARCHI DEL CUT}| = n. PARI






se il ciclo interseca con il set taglio, vuol dire che i nodi del ciclo in parte saranno in V-S, altri sono in S quindi il ciclo deve passare tra un insieme e l'altro almeno 2 volte o un numero pari di volte, poiche deve necessariamente chiudersi ( essendo un ciclo).

A COSA SERVONO QUINDI? × LE NEXT PROPIETÀ:

SIA G GRAFO

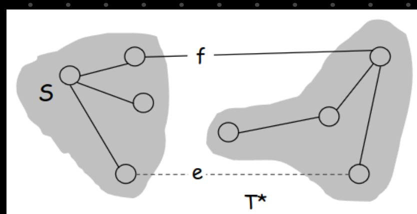
- **CUT PROPERTY** : UN ARCO $e$ ∈ AD UN CUT E AD UN CUTSET
'PROPIETÀ DEL TAGLIO' E HA PESO MINIMO TRA I CUTSET, SICURAMENTE ∃ UN MST CHE CONTIENE $e$.

- **CYCLE PROPERTY** : L'ARCO DI PESO MAX CONTENUTO IN UN CICLO PROBABILMENTE ∉ AD UN MST DEL GRAFO
'PROPIETÀ DEL CICLO'

DIM CUT PROPERTY

- SIA S CUT QUALSIASI, E $e$ ARCO MINIMO CONTENUTO NEL CUTSET.

- SUPPONIAMO CHE $e \notin T^*$ (M.S.T.) :




- SE ADD $e$, CREO UN CICLO IN T.
  supponendo che e sia stato 'scartato' perchè sennò avrebbe creato un ciclo (cosa che non deve avere un MST).
- X PROPIETÀ CUT-CYCLE INTERSECT, ∃ UN ALTRO ARCO F CHE ∈ AL CUTSET
- SE CONSIDERIAMO $T' = T^* \cup \{e\} - \{F\}$ NOTIAMO CHE ANCHE LUI È UN M.S.T.
- E SAPENDO CHE $C_e \le C_f \Rightarrow COST(T') \le COST(T^*)$
  ↳ COSTO

$T'$ QUINDI È UN MST CONTENENTE $e$

DIM CYCLE PROPERTY

- SIA $C$ CICLO IN $G$ e $f$ ARCO CON PESO MAX IN $C$ ALLORA ∃ MST CHE ∄ CONTAIN $f$.

- SUPPONIAMO CHE MST $T^*$ CONTIENE $f$.

  - SE TOGLIAMO $f$, CREO 1 CUT $S$ e INSIEME $V = G - S$
  - X CYCLE PROPERTY, ∃ UN ARCO $e$ CHE ∈ A $S$ e A $V$.
  - SE ADD $e$, $T'$ RISULTANTE È SEMPRE M.S.T.
  - SICCOME $c_f \geq c_e$, ALLORA $cost(T^*) \geq cost(T')$

  QUINDI, ∃ MST ($T'$) CHE NON CONTIENE F.

ORA VEDIAMO L'ALGORITMO

DESCRIZIONE: START DA $G$ e $T$ (M.S.T.) VUOTO. ORDINO GLI ARCHI DI $G$ IN ORDINE CRESCENTE E VISITO UN ARCO X VOLTA E LI ADD TO $T$, EVITANDO GLI ARCHI CHE FORMANO CICLI.

**IMPLEMENTAZIONE:** X ESSERE FAST OCCORRE USARE LA S.D. UNION-FIND. PER:

- MANTENERE LE COMPONENTI CONNESSE DELLA SOLUZIONE
- X VERIFY USIAMO L'OP FIND() X TROVARE e CHE E IN STESSI COMP. CONNESSE. (ARCO DA SCARTARE QUINDI).

## PSEUDOCODICE

```
algorithm Kruskal (graph G=(V,E,c))
    UnionFind UF
    T=∅
    sort the edges in ascending order of costs
    for each vertex v do UF.makeset(v)
    for each edge (x,y) in order do
        T_x=UF.find(x)
        T_y=UF.find(y)
        if T_x≠ T_y then
            UF.union(T_x,T_y)
            add edge (x,y) to T
    return T
```

E STESSO GRUPPO, & DISTINTI QUINDI.

] -> SE UGUALI QUINDI FORMANO UN CICLO

-> ADD ARCO.

## CORRETTEZZA ∧ COMPLEXITY

**CORRETTO?** SÌ

COMPLESSITÀ :

```
algorithm Kruskal (graph G=(V,E,c))
    UnionFind UF
    T=∅
    sort the edges in ascending order of costs      (1)
    for each vertex v do UF.makeset(v)              (2)
    for each edge (x,y) in order do
        Tx=UF.find(x)                               (3)
        Ty=UF.find(y)
        if Tx≠ Ty then
            UF.union(Tx,Ty)                         (4)
            add edge (x,y) to T
    return T
```

$n$ = # DI NODI

① SORT → $O(m \lg m)$, MA $m = O(n^2)$ → $O(m \lg n)$

$m$ = # ARCHI

② $n$ MAKESET = $O(n)$

③ $m$ FIND =

④ AL MAX $n-1$ UNION = → $UL(n, m)$ +

DIPENDE DA CHE S.D. USI

QUINDI

$O(n)$ e $O(m)$ APPROSS.

$O(m \log n + UF(n, m))$

DOVE SE USO:

- QUICK FIND BY UN. SIZE

-using QuickFind with *union by size*

O(m log n + m + n log n)=O(m log n)

o T

-using QuickUnion with *union by size*

O(m log n + m log n + n)=O(m log n)

$\boxed{O(m \log n)}$