Algoritmo di Kruskal

Consiste nell'eliminazione dei sottocircuiti. Sta attento a non creare dei sottocircuiti. Prendo gli archi meno costosi e li tengo.

- Non conviene usarlo sui **grafi completi** (hanno tutti gli archi)
- Funziona bene per i **grafi sparsi** (pochi archi)

1. **Inizializzazione**

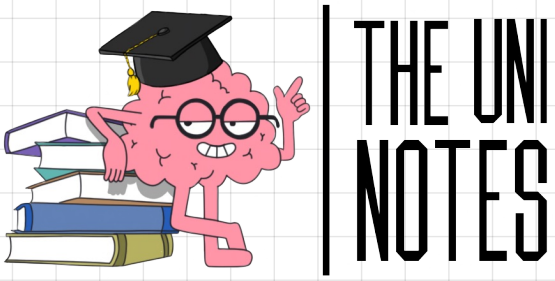
   si ordinano gli archi del grafo $G = (V, E)$ per costo non decrescente (**crescente → dal meno costoso al più costoso**), formando una lista ordinata L. si pone $T = \emptyset$.

2. **Scelta dell'arco entrante**

   si seleziona il primo arco e appartenente alla lista L. Se l'aggiunta di e a T (insieme degli archi già scelti), non determina la formazione di un circuito, si pone $T = T \cup \{e\}$ ← aggiungo l'arco scelto. Si elimina allora l'arco e dalla lista L.

3. **Condizione di arresto**

   se $card(T) = n - 1$ l'algoritmo si arresta e il sottografo (V, T) costituisce l'albero di supporto di costo minimo. Altrimenti si procede al passo 2



Algoritmo di Prim

Gioca sulla connessione dei nodi. È **greedy** ma arriviamo comunque all'ottimo.

- Conviene usarlo se abbiamo un **grafo completo**.

1. **Inizializzazione**

   Si seleziona arbitrariamente un nodo $v \in V$.

   Si pone $U = \{v\}, T = \emptyset$ e si forma l'albero (U, T).

2. **Scelta dell'arco entrante**: arco di connessione

   si seleziona l'arco (p, q) di costo minimo tale che $p \in U, q \notin U$ → ($p \in U$ sono i nodi già connessi, $q \notin U$ sono i nodi che devo ancora connettere).

   Si pone $U = U \cup \{q\}$ → aggiungo il nodo, e $T = T \cup \{(p, q)\}$ → aggiungo l'arco

   > !! Ovviamente non si crea un circuito, perchè ad ogni nodo ne lego un altro diverso. Non dobbiamo scrivere questa cosa nella definizione.

3. **Condizione di arresto**:

   Se $card(T) = n - 1$ l'algoritmo si arresta e il sottografo (V, T) costituisce l'albero di supporto di costo minimo. Altrimenti si procede al passo 2.

   oppure ci si aresta se $\#U, |U| = n$ ← cardinalità del grafo U = n

→ La complessità (tempo polinomiale di risoluzione) è $O(n^2)$

Algoritmo di Dijkstra (label setting)

È chiamato algoritmo di **label setting** → c'è un'etichetta associata ad ogni nodo che una volta fissata non può essere cambiata.

Ipotesi: $c_{ij} \geq 0 \; \forall (i, j) \in E$ dove c_{ij} sono i pesi degli archi.

> Se $c_{ij} < 0$ si utilizza l'algoritmo di Floyd Warshall (che non studieremo). È un algoritmo label connected, il valore dell'etichetta può essere aggiornato ogni volta.

Risultato/output dell'algoritmo: tutti i cammini minimi da s ad ogni altro nodo del grafo.

1. Inizializzazione

   si assegna ad ogni nodo un'etichetta $l(i)$ che al termine dell'esecuzione esprime la lunghezza del cammino minimo da s al nodo i e un'etichetta $r(i)$ che indica il nodo immediato predecessore di i lungo il cammino minimo.

   Si pone $l(i) = \infty, r(i) = 0 \; \forall i \neq s$ e $\underbrace{l(s) = 0, r(s) = 0}_{\text{nell'origine}}$

   Si definisce la lista L di nodi con etichetta permanente (non più modificabile), ponendo $L = \emptyset$.

2. Scelta di un nodo permanente:

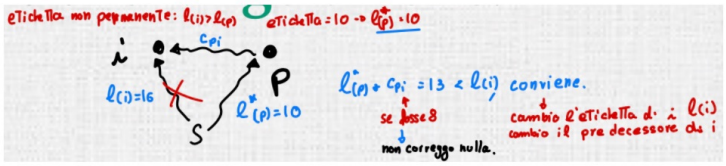
   Tra i nodi non ancora permanenti si seleziona un p tale che $l(p) = \min\{l(i) : i \notin L\}$ → **Etichetta di valore minimo**

   Si rende permanente l'etichetta di $p : L = L \cup \{p\}$.

3. Aggiornamento delle etichette:

   Per i nodi non permanenti i raggiungibili da $p(\{i : i \notin L, (p, i) \in E\})$, si aggiornano le etichette secondo l'operazione di **triangolazione**:

   $$l(i) = \min\{l(i), l(p) + c_{pi}\}$$



   Se l'etichetta viene modificata, si aggiorna anche l'etichetta del predecessore $r(i) = p$.

4. Condizione d'arresto: **complessità o(n^2)**

   Se tutti i nodi di G hanno etichetta permanente (cioè $L = V$), l'algoritmo si arresta. Altrimenti si procede al Passo 2. (Nota: se interessa solo il cammino da s a t, è possibile arrestare l'algoritmo non appena l'etichetta di t diviene permanente)

Ford - Fulkerson

Step dell'algoritmo

1. **Inizializzazione**

   si annulla il flusso iniziale ponendo $f_{ij} = 0 \; \forall (i, j) \in E, v = 0$.

   Dove f_{ij} è il **flusso iniziale** e v è il flusso totale

2. **Rete incrementale e ricerca del cammino aumentante da s a t:**

   Inviamo flusso da s a t con un unico cammino. in corrispondenza del flusso corrente si rappresenta la rete incrementale $H(V, F)$ e si ricerca un cammino aumentante da s a t nella rete incrementale. Se non esiste alcun cammino aumentante l'algoritmo si arresta e il flusso v è ottimale.

3. **Calcolo dell'incremento di flusso**

   Sfrutto al massimo il cammino aumentante. Si calcola il massimo incremento di flusso conseguibile lungo il cammino aumentante P come $\delta = \min\{p_{ij} : (i, j) \in P\}$. Si aggiorna $v = v + \delta$

   Dove δ è il **massimo flusso che posso inviare**, il quale corrisponde al $\min p_{ij}$ (minimo valore della capacità residuale)

4. **Aggiornamento dei flussi**

   Il flusso lungo gli archi di P viene aggiornato secondo le formule:

   $$f_{ij} = f_{ij} + \delta, \; \forall (i, j) \in P_d \text{ (archi diretti: incremento del flusso)}$$
   $$f_{ij} = f_{ij} - \delta, \; \forall (i, j) \in P_n \text{ (archi inversi: decremento del flusso)}$$

   Si ripete il ciclo dal passo 2. **Cerco un altro cammino aumentante.**