APA Modulo 1 Lezione 6

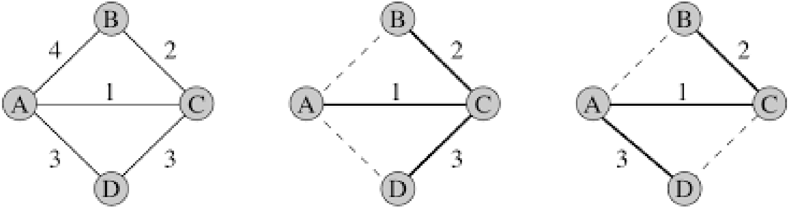
Elena Zucca

23 marzo 2020

Minimo albero ricoprente

- G connesso, non orientato e pesato
- minimo albero ricoprente (minimum spanning tree) di G è un albero ricoprente di G in cui la somma dei pesi degli archi è minima
- è quindi un sottografo di G tale che:
  - sia un albero libero, ossia connesso e aciclico
  - contenga tutti i nodi di G
  - la somma dei pesi degli archi sia minima.

Esempi di applicazione: costruzione di reti di computer, linee telefoniche, rete elettrica, etc.


B
4
2
A
1
C
3
3
D
B
2
A
1
C
3
D
B
2
A
1
C
3
D

Algoritmo di Prim

- idea: simile a Dijkstra
- si prende ogni volta, fra tutti i nodi adiacenti a quelli per cui si è già trovato il minimo (neri), quello connesso a un nodo nero dall'arco di costo minimo
- cioè si cerca il nodo "più vicino" all'albero già costruito
- poi, come in Dijkstra, si aggiornano gli altri nodi

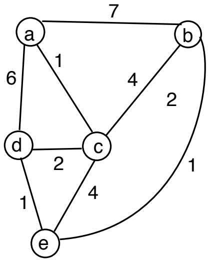
Pseudocodice

```
Prim(G,s)
  for each (u nodo in G)
    marca u come non visitato//necessario
  for each (u nodo in G) dist[u] = ∞
  parent[s] = null; dist[s] = 0
  Q = heap vuoto
  for each (u nodo in G) Q.add(u, dist[u])
  while(Q non vuota)
    u = Q.getMin()//nodo a minima distanza dai neri
    marca u come visitato (nero)
    for each ((u,v) arco in G)
      if (v non visitato  &&  c_{u,v} < dist[v] )
        parent[v] = u; dist[v] = c_{u,v}
        Q.changePriority(v,dist[v]) //moveUp
```

Osservazione

- a differenza di Dijkstra occorre controllo esplicito che i nodi adiacenti al nodo *u* estratto dalla coda non siano già stati visitati
- non è detto che per *v* già visitato il test $\mathtt{c}_{\mathtt{u,v}}$ `< dist[v]` sia falso
- vedi esempio dopo: quando estraggo *e* non devo modificare *d*

Esempio



estratto	*a*	*b*	*c*	*d*	*e*	*albero*
	0	∞	∞	∞	∞	
a	0	7	1	6		$(a, b), (a, c), (a, d)$
c		4		2	4	**(a,c)**, $(b, c), (c, d), (c, e)$
d					1	**(a,c)**, **(c,d)**, $(b, c), (d, e)$
e		1				**(a,c)**, **(c,d)**, **(d,e)**, (b, e)
b						**(a,c)**, **(c,d)**, **(d,e)**, **(b,e)**

prima colonna: nodo estratto; successive: nodi per i quali viene modificata `dist` e come; ultima: archi dell'albero ricoprente (in grassetto quelli definitivi)

Correttezza

Sia T l'albero di nodi neri (non in `Q`) corrente. L'invariante è composta di due parti:

1. $T \subseteq T_{MAR}$ per qualche T_{MAR} minimo albero ricoprente di G
2. per ogni nodo $u \neq s$ in `Q`
   $\texttt{dist}[u] =$ costo minimo di un arco che collega u a un nodo nero.

se questo arco non esiste consideriamo il costo minimo $= \infty$

L'invariante vale all'inizio

L'invariante vale all'inizio:

1. vale banalmente perché T è vuoto (non ci sono nodi neri)
2. vale banalmente perché non ci sono nodi neri e la distanza è per tutti infinito, tranne che per s

L'invariante si mantiene

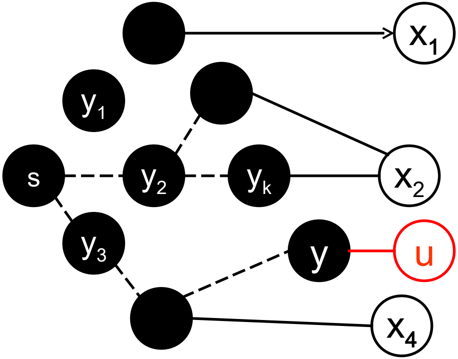
- $T \subseteq T_{MAR}$ per un certo T_{MAR} minimo albero ricoprente di G
- viene estratto u con $\texttt{dist}[u]$ minima, cioè connesso a un nodo nero y da un arco di costo minimo tra quelli tra nero e non nero
- (y, u) viene aggiunto a T
- tesi: $T + (y, u)$ è ancora parte di un minimo albero ricoprente di G
- per assurdo: supponiamo $T + (y, u)$ non sia parte di nessun minimo albero ricoprente di G
- quindi in T_{MAR} non c'è l'arco (y, u), ma allora ci deve essere un altro cammino da y a u
- questo cammino dovrà contenere un (primo) arco (x, z) tra un nodo di T e un nodo non di T

L'invariante si mantiene

- poiché T_{MAR} è un albero, se eliminiamo (x, z) otteniamo un grafo non connesso, costituito da due alberi T_1 e T_2
- quindi se consideriamo $T_1 + T_2 + (y, u)$ otteniamo:
  - nuovamente un albero
  - che contiene tutti i nodi di G, quindi è un albero ricoprente
  - con somma pesi degli archi $\leq T_{MAR}$, in quanto differiscono solo per (y, u) al posto di (x, z), e $c_{y,u} \leq c_{x,z}$
- quindi $T_1 + T_2 + (y, u)$ è un minimo albero ricoprente, contro l'ipotesi per assurdo
- come in Dijkstra, u è ora nero, quindi occorre ripristinare l'invariante (2), controllando se per qualche v non nero adiacente a u l'arco (u, v) ha costo minore del precedente arco che univa v a un nodo nero

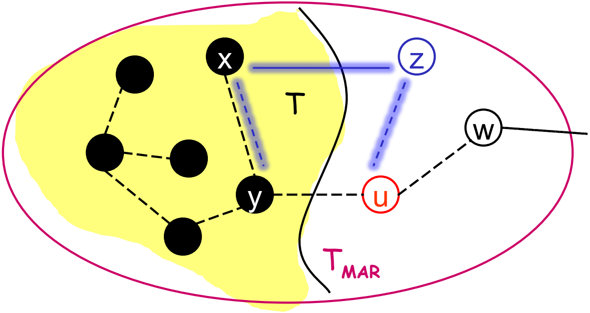
Graficamente

nodi neri $= T \subseteq T_{MAR}$ per un certo T_{MAR} minimo albero ricoprente di G



viene estratto u con `dist`[u] minima, cioè connesso a un nodo nero y da un arco di costo minimo tra tutti quelli che uniscono un nodo nero a un nodo non nero

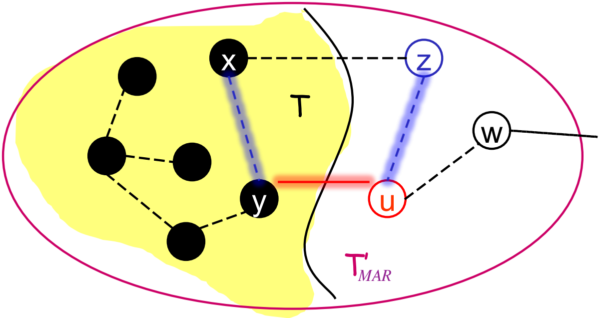
Graficamente



per assurdo: supponiamo $T + (y, u)$ non sia parte di nessun minimo albero ricoprente di G

quindi in T_{MAR} non c'è l'arco (y, u), ma allora ci deve essere un altro cammino da y a u, che contiene un (primo) arco (x, z) tra un nodo di T e un nodo non di T

Graficamente



se eliminiamo (x, z) e aggiungiamo (y, u) otteniamo un albero ricoprente con somma pesi degli archi $\leq T_{MAR}$, in quanto $c_{y,u} \leq c_{x,z}$
quindi un minimo albero ricoprente che contiene (y, u), contro l'ipotesi per assurdo

Postcondizione e terminazione

- all'uscita dal ciclo tutti i nodi sono neri, quindi T connette tutti i nodi, cioè è un albero ricoprente di G
- per l'invariante (1), $T \subseteq T_{MAR}$ per qualche T_{MAR} minimo albero ricoprente, quindi $T = T_{MAR}$
- funzione di terminazione: numero di nodi nello heap (non neri)

Osservazioni

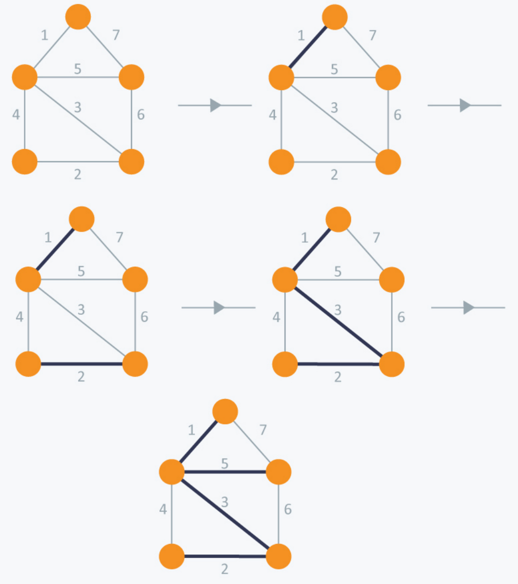
- come in Dijkstra, per dimostrare che il ciclo mantiene l'invariante (1) è necessario dimostrare che mantiene anche l'invariante (2)
- genera un albero radicato di radice s, ma il minimo albero ricoprente è un albero non radicato, quindi è possibile ottenere lo stesso a partire da nodi diversi

  si può dimostrare che se i pesi degli archi sono tutti distinti è unico
- l'analisi della complessità è esattamente la stessa dell'algoritmo di Dikstra, quindi $O((n+m)\log n)$.

Algoritmo di Kruskal

- anche questo algoritmo risolve il problema del minimo albero ricoprente
- questo viene costruito mantenendo una foresta alla quale si aggiunge a ogni iterazione un nuovo arco che unisce due sottoalberi distinti
- quindi, a differenza di quello di Prim, non a partire da un nodo scelto come radice, ma come insieme di archi che alla fine risulta essere un grafo connesso aciclico, quindi un albero libero

Esempio



Pseudocodice

```
Kruskal(G)
  s = sequenza archi di G in ordine di costo crescente
  T = foresta formata dai nodi di G e nessun arco
  while (s non vuota)
    estrai da s il primo elemento (u,v)
    if (u,v non connessi in T) T = T + (u,v)
  return T
```

non si fa nessun reinserimento o variazione di ordine quindi è una semplice sequenza ordinata

Ottimizzazione

Ottimizzazione: un albero di n nodi ha $n-1$ archi, quindi si può interrompere il ciclo dopo aver aggiunto all'albero $n-1$ archi

```
Kruskal(G)
  s = sequenza archi di G in ordine di costo crescente
  T = foresta formata dai nodi di G e nessun arco
  counter = 0
  while (counter < n-1)
    estrai da s il primo elemento (u,v)
    if (u,v non connessi in T) T = T + (u,v)
    counter++
  return T
```

Correttezza (simile a Prim)

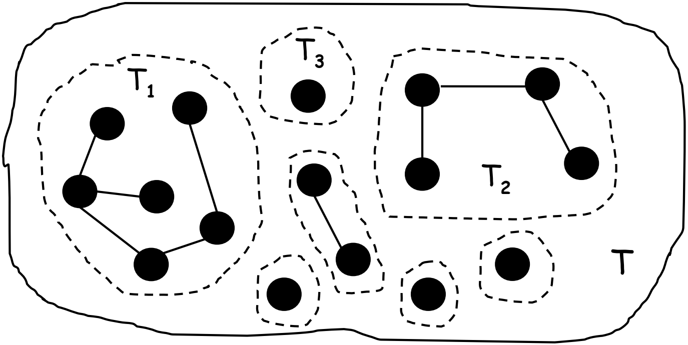
- invariante: $T \subseteq T_{MAR}$ per T_{MAR} minimo albero ricoprente di G
- all'inizio vale banalmente perché in T non ci sono archi
- si mantiene, analogamente a Prim:
  - estraggo (u, v) di peso minimo tra quelli rimanenti
  - se u e v appartengono a uno stesso albero in T, aggiungendo (u, v) si avrebbe un ciclo, quindi T rimane uguale e l'invariante vale ancora
  - se u e v appartengono a due alberi distinti T_u e T_v in T, dimostriamo che $T + arco$ è ancora una foresta contenuta in un minimo albero ricoprente
  - si ragiona per assurdo in modo analogo a Prim

Ragionamento per assurdo (simile a Prim)

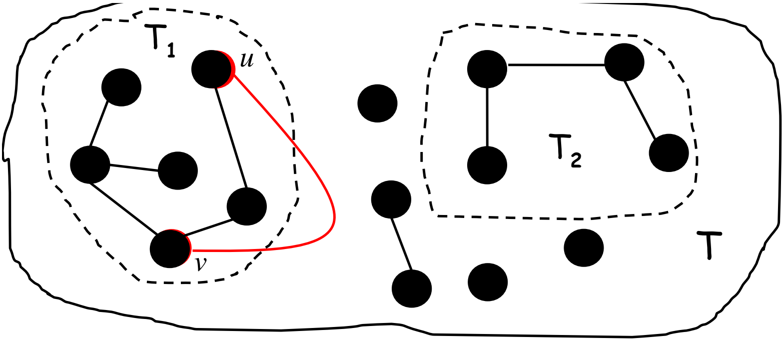
- assumiamo $T+(u, v)$ non parte di un minimo albero ricoprente di G
- in T_{MAR} non c'è l'arco (u, v), ma allora ci deve essere un altro cammino da u a v
- questo dovrà contenere un arco (x, y) con x nodo in T_u e y no
- (x, y) non può appartenere alla foresta corrente T, perché dovrebbe essere un arco di T_u
- quindi (x, y) è ancora in s, quindi $c_{u,v} \leq c_{x,y}$
- quindi, come in Prim, se eliminiamo (x, y) da T_{MAR}, e aggiungiamo invece l'arco (u, v) otteniamo:
  - nuovamente un albero ricoprente
  - in cui la somma dei pesi degli archi è minore o uguale di quella di T_{MAR}
- quindi è un minimo albero ricoprente che contiene l'arco (u, v), contro l'ipotesi per assurdo

Graficamente

foresta corrente $T \subseteq T_{MAR}$ per un certo T_{MAR} minimo albero ricoprente di G

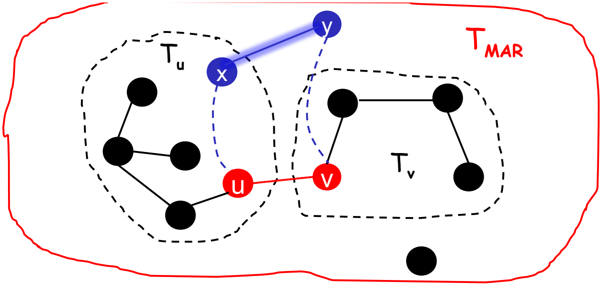


Graficamente



- estraggo (u, v) di peso minimo tra quelli rimanenti
- se u e v appartengono a uno stesso albero in T, aggiungendo (u, v) si avrebbe un ciclo, quindi T rimane uguale e l'invariante vale ancora

Graficamente



- se u e v appartengono a due alberi distinti T_u e T_v in T, dimostriamo che $T + (u, v)$ è ancora una foresta contenuta in un minimo albero ricoprente, per assurdo
- in T_{MAR} non c'è l'arco (u, v), ma allora ci deve essere un altro cammino da u a v ... etc

Postcondizione e terminazione

- postcondizione: l'algoritmo ha esaminato tutti gli archi unendo ogni volta due alberi di T non ancora connessi (ossia, ha inserito $n - 1$ archi, come si vede direttamente nella versione ottimizzata)
- quindi alla fine T è un unico albero, sottoalbero di un minimo albero ricoprente di G, contenente tutti i nodi di G, quindi è un minimo albero ricoprente di G
- funzione di terminazione: numero archi non estratti
  oppure `counter` - (n-1)

Complessità

- punto chiave: controllare se due nodi sono già connessi
- farlo in modo banale richiede tempo $O(n)$ nel caso peggiore, quindi si ha $O(m \cdot n)$ (controllo fatto nel caso peggiore su tutti gli archi)
- per migliorare l'efficienza possiamo implementare la foresta con una struttura detta union-find (argomento del terzo modulo del corso)
- complessità nella versione con union-find:
  - ordinamento degli archi: $O(m \log m)$
  - ciclo: complessità "quasi" $O(n + m)$ (complessità ammortizzata)
- essendo il grafo connesso si ha $m \geq n - 1$, quindi $O(n + m) = O(m)$, quindi complessivamente $O(m \log m)$