ale-cci

Modelli Algoritmi per il Supporto alle Decisioni

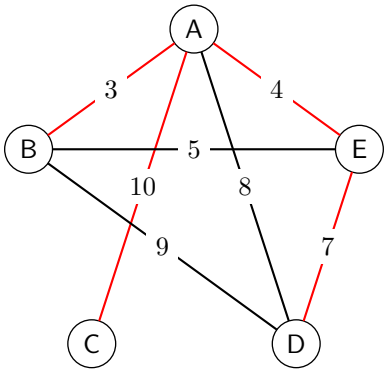
March 31, 2020

Introduction

TODO

Minimum Spanning Tree

Algoritmo di Kruskal (Greedy)




```python
from utils import num_vertices

def kruskal(edges: list, N: int) -> list:
    connected = set()
    mst = []

    edges = sorted(graph)
    for edge in edges:
        weight, lhs, rhs = edge

        # Two nodes already connected
        if lhs in connected and rhs in connected:
            continue

        mst.append(edge)
        connected.update({lhs, rhs})

        if len(mst) == N:
            break

    return mst


if __name__ == '__main__':
    graph = [(10, 'A', 'C'), (8, 'A', 'D'),
             (7, 'D', 'E'), (4, 'A', 'E'),
             (3, 'B', 'A'), (9, 'B', 'D'), (5, 'B', 'E')]
    N = num_vertices(edges=graph)

    print(kruskal(graph, N))
```

Correttezza algoritmo di Kruskal

Supponiamo per assurdo che esista un' diverso MST $T' = (V, E_{T'})$ di peso inferiore a $T = (V, E_T)$, quello restituito dall'algoritmo greedy.

Siccome i due alberi hanno costo diverso, differiscono di almeno un' arco. Indichiamo con e_h l'arco a peso minore appartenente a $\{E_T - E_{T'}\}$. Dato che T' è un MST, esiste un ciclo C in $\{e_h\} \cup E_{T'}$ contenente l'arco e_h. Siccome anche T è un albero, quindi non ha cicli, allora $C \cap E_T \neq \emptyset$. Chiamiamo e_r l'arco a peso minore appartenente a $C \cap \{E_T - E_{T'}\}$. Necessariamente $w_{e_r} \leq w_{e_h}$,

altrimenti l'algoritmo greedy applicato a T avrebbe selezionato prima e_h al posto di e_r. Sostituendo in T' l'arco e_r con e_h ottengo un nuovo albero di peso inferiore.

Questo va contro l'ipotesi T' è l'albero di supporto a peso minore.

Analisi complessità

$O(E \cdot \log(E))$, dovuta all'ordinamento degli archi in ordine di peso. Il controllo dell'esistenza di cicli è effettuato in $O(1)$.

Foresta di supporto

Viene chiamata foresta di supporto di un grafo G un grafo parziale $F = (V, E_F)$ privo di cicli. In particolare, un albero di supporto è una foresta con una sola componente connessa.

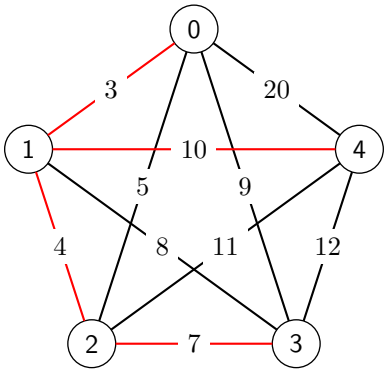
Teorema

Indichiamo con $(V_1, E_1), \ldots, (V_k, E_k)$ le componenti connesse di una foresta di supporto $F = (V, E_F)$ del grafo G. Sia inoltre (u, v) un arco a peso minimo tra quelli con un unico estremo in V_1. Allora esiste almeno un albero di supporto a peso minimo appartenente a $\bigcup_{i=1}^{k} E_i$, che contiene (v, v).

Dimostrazione

Per assurdo, l'albero a peso minimo non contiene (u, v). Ma aggiungendo (u, v) a tale albero si forma un ciclo contenente un altro arco (u', v') con un solo estremo in V_1. Se si toglie questo arco e si lascia (u, v) si ottiene un albero di peso minore, contraddicendo l'ipotesi.

Algoritmo MST-1




```python
def mst_1(w: list) -> list:
    V = set(range(len(w)))  # {0, 1, 2, 3, 4}
    c = [0] * len(V)        # [0, 0, 0, 0, 0]
    U = {0}
    mst = []

    while U != V:
        weight, u = min((w[v][c[v]], v) for v in V - U)
        U.add(u)
        mst.append((u, c[u]))

        for v in V - U:
            if w[v][u] < w[v][c[v]]:
                c[v] = u

    return mst
```

```
if __name__ == '__main__':
    w = [[ 0,  3,  5,  9, 20],
         [ 3,  0,  4,  8, 10],
         [ 5,  4,  0,  7, 11],
         [ 9,  8,  7,  0, 12],
         [20, 10, 11, 12,  0]]

    print(mst_1(w))
```

Correttezza MST-1

Inizialmente abbiamo la foresta con $V_1 \equiv U = \{v_1\}, V_i = \{v_i\} \quad i = 2 \dots n$, con tutti gli $E_i = \emptyset$.

Alla prima iterazione si inserisce l'arco (V_i, v_{j_1}), $j_1 \neq 1$, a peso minimo tra quelli con un solo estremo in $U \equiv V_1$ e quindi, per il teorema visto al paragrafo della foresta di supporto, tale arco farà parte dell'albero di supporto a peso minimo tra tutti i possibili alberi di supporto.

Con l'aggiunta di questo arco, le due componenti connesse (V_1, E_1) e (V_{j_1}, E_{j_1}) si fondono in un'unica componente connessa con nodi $U = \{v_i, v_{j_1}\}$ e l'insieme di archi $E_T = \{(v_1, v_{j_1})\}$, mentre le altre componenti connesse non cambiano. Abbiamo cioè che le componenti connesse

$$(U, E_T), \quad (V_i, \emptyset) i \in \{2, \dots n\} - \{j_1\}$$

Alla seconda iterazione andiamo a selezionare il nodo v_{j_2} e il relativo arco $(v_{j_2}, c(v_{j_2}))$ con il peso minimo tra tutti quelli con un solo estremo in U. In base al teorema, l'arco $(v_{j_2}, c(v_{j_2}))$ farà parte di un albero di supporto a peso minimo tra tutti quelli che contengono l'unione di tutti gli archi delle componenti connesse, che si riduce ad E_T.

Effettuiamo lo stesso ragionamento per tutti gli n ottenendo l'albero di supporto a peso minimo.

Complessità dell'algoritmo

Il numero di operazioni richiesto è pari a $O(V^2)$ dovuta al ciclo eseguito V volte $(O(V))$ e la ricerca del minimo in tempo lineare.

Confronto con algoritmo di Kruskal

Anche se risulta essere peggiore rispetto all'algoritmo di greedy Kruskal, è possibile dimostrare che, in caso di grafi densi ha una complessità ottima. Infatti per tali grafi non possiamo aspettarci di fare meglio di $O(V^2)$: la sola operazione di lettura dei pesi degli archi richiede $O(V^2)$.

Algoritmo MST-2

```
import utils

def mst_2(edges, N):
    shortest = [None] * N
    minimum = [None] * N
    # connected = set(range(N))

    for weight, u, v in edges:
        if minimum[u] == None or weight < minimum[u]:
            shortest[u] = (u, v)
            minimum[u] = weight

        if minimum[v] == None or weight < minimum[v]:
            shortest[v] = (u, v)
            minimum[v] = weight

    return set(shortest)
```

```
if __name__ == '__main__':
    edges = [(3, 0, 1), ( 5, 0, 2), (9, 0, 3), (20, 0, 4),
             (4, 1, 2), ( 8, 1, 3), (10, 1, 4),
             (7, 2, 3), (11, 2, 4),
             (12, 3, 4)]
    N = utils.num_vertices(edges=edges)

    print(mst_2(edges, N))
```

Dimostrazione correttezza

Lasciata per esercizio, si basa sul teorema della foresta. *"Tutti gli archi shortest aggiunti ad una certa iterazione, sono tutti archi che fanno parte ad un albero di supporto ottimo, tra tutti i possibili alberi di supporto."*

Complessità algoritmo

$O(E \cdot \log_2(V))$ derivato dal costo dell'iterazione su tutti gli archi $O(E)$, eseguita un numero massimo di $\log(|V|)$ volte.

Inizialmente il numero di componenti connesse è pari al numero di nodi. Sicuramente ad ogni iterazione, il numero di componenti connesse viene almeno dimezzato. Per cui, il primo ciclo viene eseguito al più $\log(|V|)$ volte.

Per grafi densi con $|E| = O(|V|^2)$ questa complessità è peggiore di quella di MST-1, ma se il numero di archi scende sotto l'ordine $O(|V|^2/log(|V|))$ l'algoritmo MST-2 ha prestazioni migliori.

Note

Questi tre algoritmi appena visti sono tutti e tre algoritmi costruttivi, senza revisione delle decisioni passate.

Contents