

Corso di laurea in Informatica
Progettazione d'algoritmi
Didattica blended

Gli algoritmi greedy: esercizi

Angelo Monti



SAPIENZA
UNIVERSITÀ DI ROMA

ESERCIZIO1:

Ci sono n case, indicate con gli interi $i = 0, 1, \dots, n - 1$, ognuna delle quali necessita di una fornitura d'acqua.

La costruzione di un pozzo nella casa i costa $P[i]$ e la costruzione di una tubazione fra le case i e j costa $M[i][j]$ ($M[i][j] = -1$ se non c'è possibilità di stendere una tubazione tra i e j).

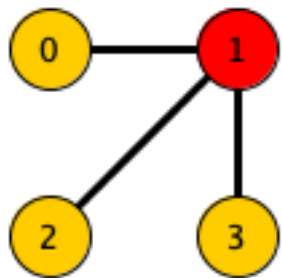
Una casa risulta fornita d'acqua se vi è stato costruito un pozzo o l'acqua vi giunge da un pozzo di un'altra casa tramite tubazioni.

Dare lo pseudo-codice di un algoritmo che determina le case in cui costruire i pozzi e le tubazioni da costruire per dare una fornitura d'acqua a tutte le case minimizzando il costo totale.

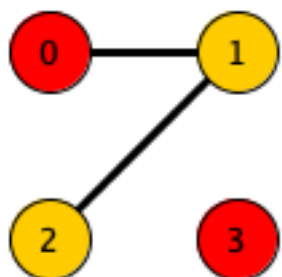
Il costo totale è dato dalla somma dei costi dei pozzi e delle tubature costruiti. L'algoritmo deve avere complessità $O(n^2 + m \log n)$.

Motivare bene la correttezza dell'algoritmo.

Per l'istanza con 4 case dove $P = [4, 5, 6, 3]$ e $M = \begin{bmatrix} -1 & 2 & 4 & 6 \\ 2 & -1 & 1 & 7 \\ 4 & 1 & -1 & 8 \\ 6 & 7 & 8 & -1 \end{bmatrix}$



è una soluzione di costo $5 + 2 + 1 + 7 = 15$

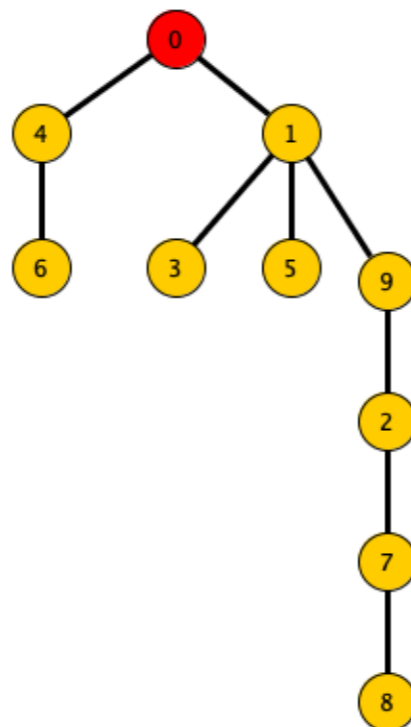


è la soluzione ottima di costo $4 + 3 + 2 + 1 = 10$

Per l'istanza con 10 case dove $P = [1, 8, 6, 5, 2, 3, 3, 2, 8, 4]$ e

$$M = \begin{bmatrix} -1 & 2 & 2 & 2 & 1 & 3 & -1 & 5 & 6 & 7 \\ 2 & -1 & 5 & 1 & -1 & 2 & -1 & 5 & 9 & 1 \\ 2 & 5 & -1 & -1 & -1 & 4 & 5 & 1 & -1 & 1 \\ 2 & 1 & -1 & -1 & 4 & 3 & 4 & 4 & 9 & 1 \\ 1 & -1 & -1 & 4 & -1 & 4 & 1 & 4 & 8 & 4 \\ 3 & 2 & 4 & 3 & 4 & -1 & 3 & 4 & 3 & 4 \\ -1 & -1 & 5 & -1 & 1 & 3 & -1 & 5 & 9 & -1 \\ 5 & 5 & 1 & 4 & 4 & 4 & 5 & -1 & 1 & 1 \\ 6 & 9 & -1 & 9 & 8 & 3 & 9 & 1 & -1 & 9 \\ 7 & 1 & 1 & 1 & 4 & 4 & -1 & 1 & 9 & -1 \end{bmatrix}$$

Una soluzione ottima è



Il costo della soluzione è $1 + 1 + 2 + 1 + 1 + 2 + 1 + 1 + 1 + 1 = 12$

Esercizio.

La rete viaria della città *Universa* consiste di varie piazze collegate tra loro da strade. La particolarità di questa città è che tutte le strade sono a senso unico. La piazza *Unica* è la più importante della città e si sa che è raggiungibile da qualunque altra piazza.

Il sindaco vuole selezionare un insieme di strade da riasfaltare in modo tale che da ogni piazza sia possibile poi raggiungere la piazza *Unica* percorrendo solo strade asfaltate. Il costo di riasfaltare le varie strade dipende dalle loro lunghezze ed il sindaco vuole spendere il meno possibile. Gli urbanisti comunali pensano di poter trovare questo insieme di strade tramite il seguente algoritmo greedy che lavora sul grafo pesato G in cui ogni nodo è una piazza di *Universa* e un arco da x a y di costo c sta a significare che c'è una strada che porta da x a y e riasfaltarla costa c .

def *seleziona*(G) :

Sol = *set*()

A = l'insieme con tutti i nodi di G tranne piazza *Unica*

E = la lista con tutti gli archi di G

while $A \neq \text{set}()$:

 prendi in E arco (x, y) con x in A e y non in A di costo minimo

Sol.add((x, y))

A .remove(x)

return *Sol*

provare o confutare che:

1. le strade selezionate permettono di raggiungere la piazza *Unica* da qualunque altra piazza
2. la soluzione prodotta è di costo minimo

def *seleziona*(*G*) :

Sol = *set*()

A = *l'insieme con tutti i nodi di G tranne Piazza Unica*

E = *la lista con tutti gli archi di G*

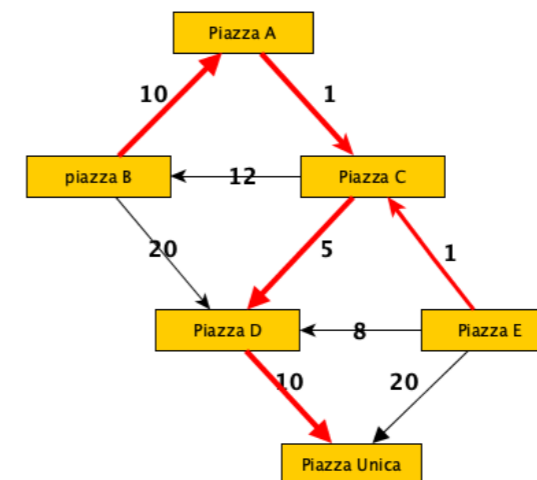
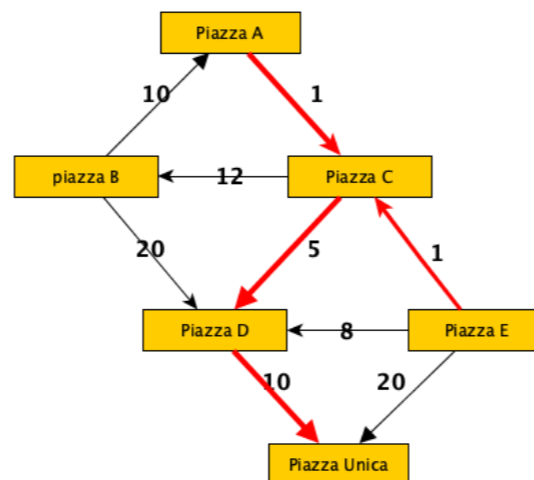
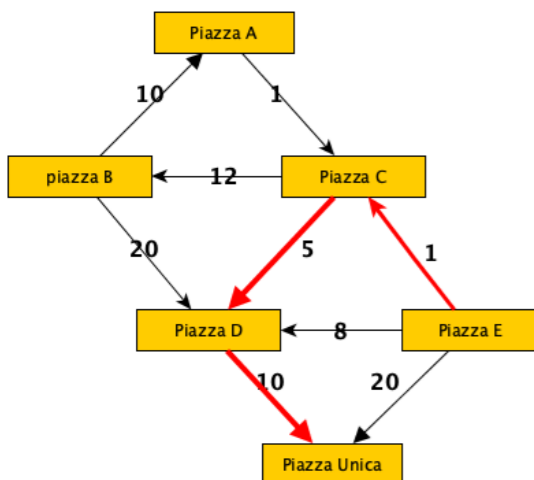
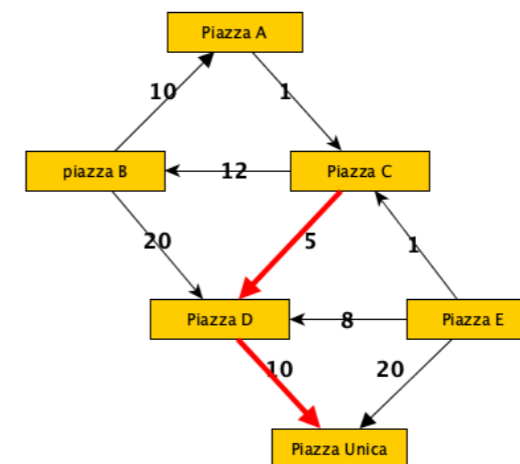
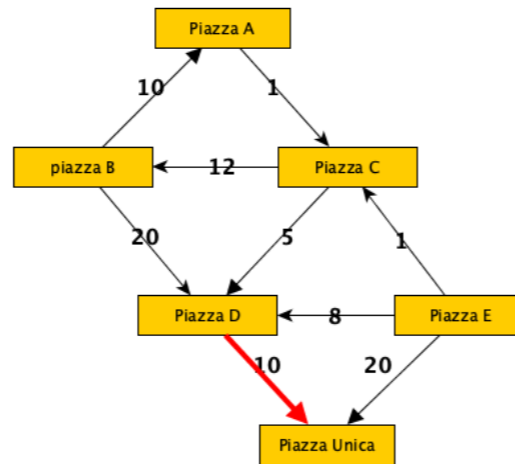
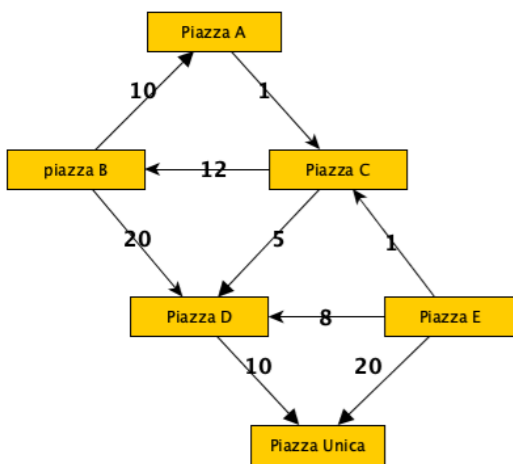
while *A* \neq *set*() :

prendi in E arco (x,y) con x in A e y non in A di costo minimo

Sol.add((*x,y*))

A.remove(*x*)

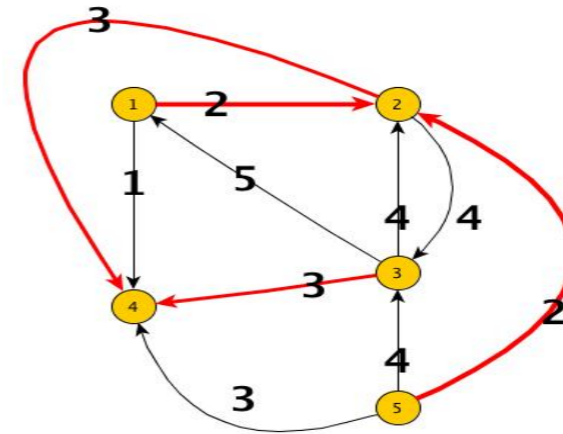
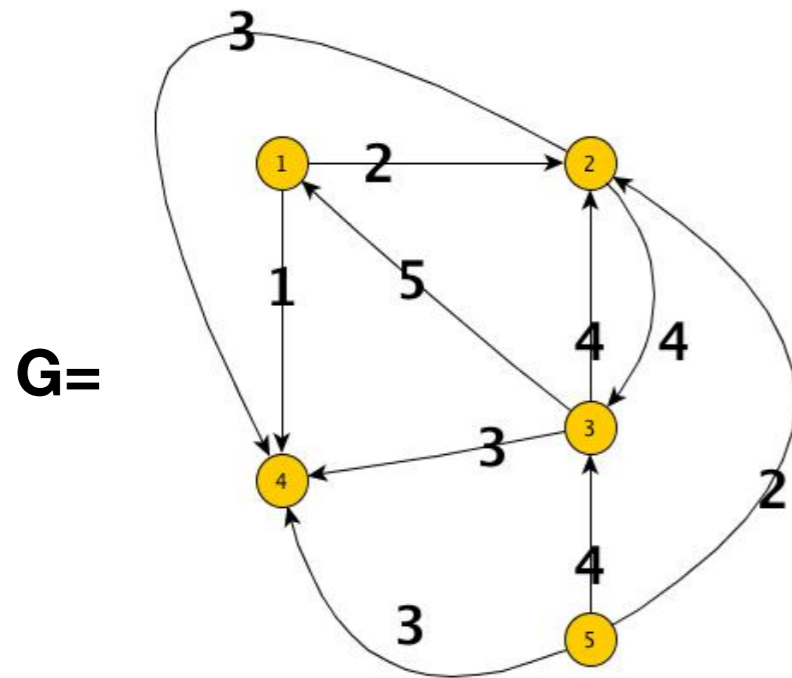
return *Sol*



La soluzione prodotta costa 27.

Esercizio.

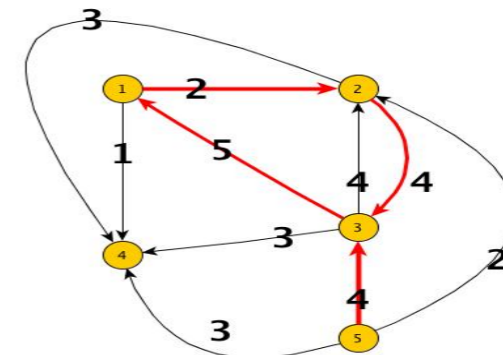
In un grafo diretto e pesato G un sottoinsieme A degli archi è detto **univoco** se non contiene 2 o più archi uscenti dallo stesso nodo. Il peso dell'insieme univoco è dato dalla somma dei pesi dei suoi archi.



In rosso un insieme univoco di G di peso **10**

Problema. Progettare un algoritmo *greedy* che, dato un grafo G diretto e pesato, in tempo $O(n + m)$ restituisce il peso e gli elementi di un insieme univoco di peso massimo di G . **Motivare la correttezza e la complessità dell'algoritmo proposto.**

Insieme univoco di peso massimo **15** per G



ESERCIZIO: All'acquisto degli n biglietti per un'importante prima teatrale sono interessati m gruppi di persone.

Ciascun gruppo ha intenzione di acquistare i biglietti solo se questi sono a sufficienza per tutti i membri del gruppo.

Vogliamo selezionare i gruppi a cui vendere i biglietti in modo da massimizzare il numero di biglietti venduti.

Per risolvere il problema ci viene proposto il seguente algoritmo che

- prende in input il numero n di posti disponibili e la lista $lista$ di m elementi dove in $lista[i]$ c'è un numero minore o uguale a n che rappresenta i biglietti richiesti dal gruppo i
- restituisce il numero massimo di biglietti che è possibile vendere e l'insieme dei gruppi a cui venderli.

```
def seleziona(n, lista):
    gruppi=[(l,i) for i,l in enumerate(lista)]
    gruppi.sort(reverse=True)
    tot, Sol = 0, set()
    for l,i in gruppi:
        if tot + l <= n:
            Sol.add(i)
            tot+= l
    return tot, Sol
```

1. provare che l'algoritmo non è corretto

2. provare che l'algoritmo ha un rapporto d'approssimazione limitato da 2

ESERCIZIO: Dato un grafo connesso e pesato G , un suo nodo s ed un intero $k \leq n$ vogliamo trovare un sottografo aciclico di G che comprende il nodo s e contiene k nodi e sia di costo minimo. (nota che quando $n = k$ il problema diviene quello di trovare il minimo albero di copertura di G)

Viene proposto il seguente algoritmo *greedy*:

```
def copertura( $G, s, k$ ) :
```

```
     $A, T = \{s\}, \{x : [ ]\}$ 
```

```
    for _ in range( $k$ ) :
```

```
        sia  $(x, y)$  un arco di costo minimo con  $x$  in  $A$  e  $y$  not in  $A$ 
```

```
         $T[x].add(y)$ 
```

```
         $T[y] = [x]$ 
```

```
         $A.add(y)$ 
```

```
return  $T$ 
```

Provare che:

1. l'algoritmo sbaglia
2. l'algoritmo non ha un rapporto d'approssimazione costante

ESERCIZIO:

Definiamo **sottosequenza** di una sequenza S una qualunque sequenza che si ottiene da S eliminando da S un numero arbitrario di elementi.

Ad esempio, per $S = 1011001$

- 01101 e 10001 sono sottosequenze di S (ottenute da $_011_01$ e 10_001)
- 00110 e 01010 non sono sottosequenze di S

Problema: date due sequenze binarie A e B vogliamo trovare una sottosequenza di lunghezza massima comune ad entrambe.

Ad esempio:

per $A=01101$ e $B=110100$ una sottosequenza comune di lunghezza massima è **1101**

Viene proposto il seguente algoritmo dove si assume che le sequenze sono rappresentate tramite liste.

```
def comune(A,B):  
    za, zb=A.count(0), B.count(0)  
    ua, ub=len(A)-za, len(B)-zb  
    z=min(za, zb)  
    u=min(ua, ub)  
    if z>u: return [0]*z  
    return [1]*u
```

1. trovare un controesempio che fa sbagliare l'algoritmo con un rapporto d'approssimazione di 2
2. dimostrare che il rapporto d'approssimazione dell'algoritmo è 2.