Relazione di laboratorio

Corso di Algoritmi Avanzati
Laurea Magistrale in Informatica
A.A. 2019-2020

Magarotto Francesco - 1236594
Muraro Enrico - 1238899
Piva Giulio - 1242455

1 Introduzione

Lo scopo del laboratorio è valutare le prestazioni dell'algoritmo di Karger per il problema del minimum cut rispetto a quattro parametri:

- Il tempo impiegato dalla procedura di Full Contraction
- Il tempo impiegato dall'algoritmo completo per ripetere la contrazione un numero sufficientemente alto di volte
- Il discovery time, ossia il momento in cui l'algoritmo trova per la prima volta il taglio di costo mimimo
- L'errore nella soluzione trovata rispetto al risultato ottimo

Il linguaggio di programmazione scelto dal nostro gruppo è Java.

1.1 Esecuzione del programma

Gli algoritmi sono stati sviluppati come progetto Maven. All'interno della cartella é presente la versione portable di Maven, pertanto non è necessario averlo installato. É richiesto almeno il JDK 11 installato nel sistema. Per eseguire i tre algoritmi utilizzare i seguenti comandi:
Linux:

```
./mvnw install
./mvnw exec:java
```

Windows:

```
mvnw.cmd install
mvnw.cmd exec:java
```

L'esecuzione del main genera automaticamente dei file csv nella directory del progetto contenenti i tempi registrati. L'algoritmo di Karger viene interrotto dopo 60 secondi.

1.2 Strutture dati utilizzate

Per rappresentare il grafo abbiamo utilizzato una lista di archi, dove ogni arco è rappresentato da un oggetto `Edge`. Ogni arco è da intendersi come indiretto, quindi un singolo arco rappresenta un collegamento in entrambi i versi.

1.3 Lettura di un grafo da file

Per caricare un grafo in memoria, abbiamo implementato una classe GraphReader, che si occupa della lettura del file tramite la libreria *nio* di Java. La classe infine ritorna un'istanza della classe `Graph`.

1.4 Implementazione di Karger

L'algoritmo di Karger contiene un metodo `solve` che ha come parametri: il grafo, il numero di ripetizioni da effettuare, e infine il timeout, ovvero la durata massima dell'esecuzione dell'algoritmo. Per eseguire le misurazioni dei tempi di esecuzioni si fa uso dell'istruzione `System.currentTimeMillis()` che ritorna il tempo attuale in ms. Essendo il programma single thread non c'è il rischio che l'esecuzione venga interrotta e quindi permette di ottenere misure affidabili. Infine, per quanto riguarda la codifica l'algoritmo si occupa di richiamare `k`-volte il metodo Full-Contraction calcolando il valore del taglio minimo.

1.4.1 Implementazione del metodo Full Contraction

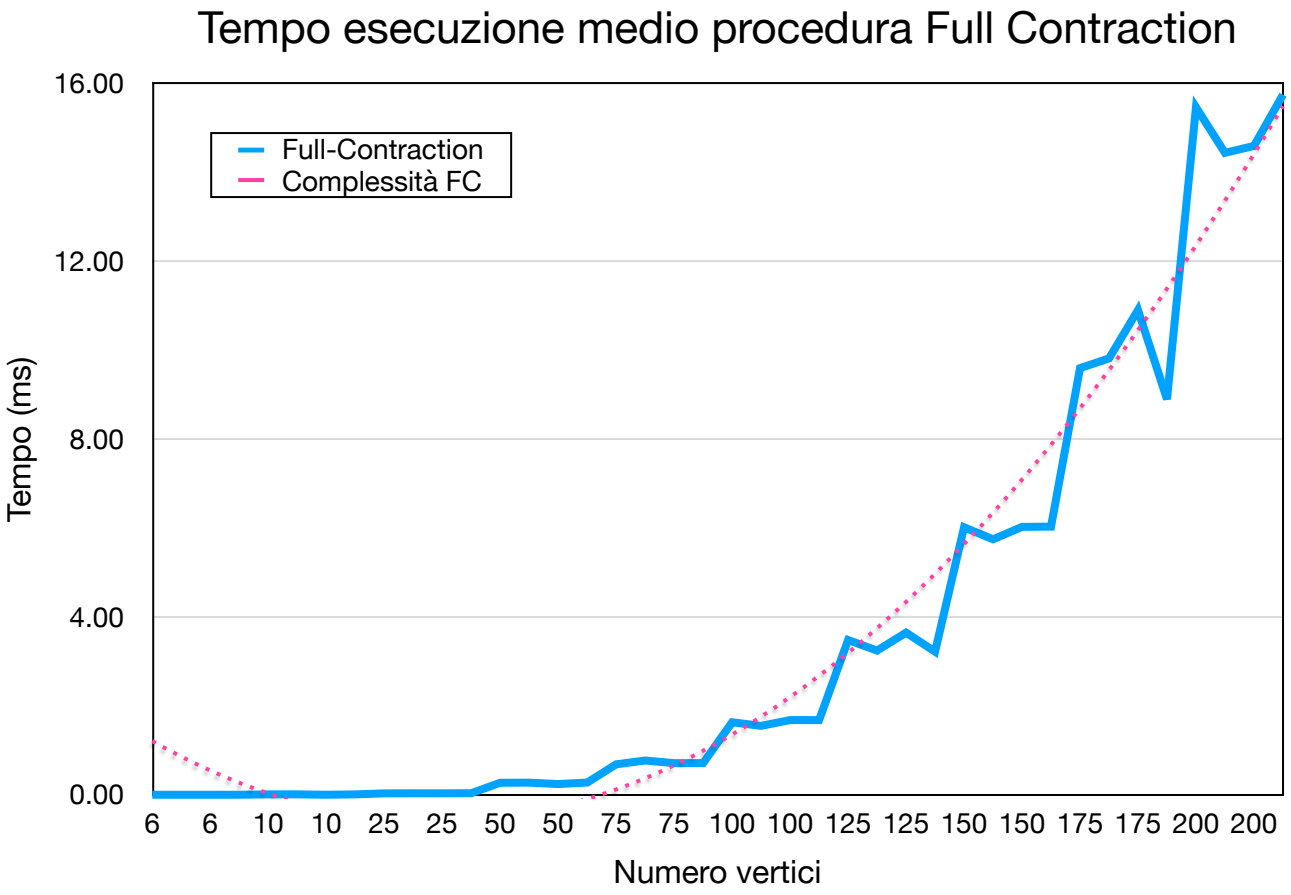
Il metodo Full Contraction sceglie un arco in maniera casuale dalla lista degli archi del grafo, e rimuove tutti gli archi tra le due estremità dell'arco scelto. Per la contrazione utilizziamo una sottoprocedura `Contraction`, che oltre alla rimozione si occupa di cambiare quegli archi che avevano come estremità uno dei nodi cancellati, cambiandola con il nuovo nodo che rimpiazza i due nodi cancellati. Gli archi da aggiornare vengono trovati semplicemente scorrendo la lista di archi del grafo. I nuovi nodi vengono etichettati in ordine progressivo, aumentando il contatore ogni volta che viene fatta una contrazione.

2 Tabella dei risultati

Indice	Numero vertici	Numero ripetizioni	Risultato atteso	Tempo impiegato	Discovery time	Tempo medio Full contraction	Errore	Taglio minimo
1	6	32	2	0.00E+00	0.00E+00	0.00	0	2
2	6	32	1	0.00E+00	0.00E+00	0.00	0	1
3	6	32	3	0.00E+00	0.00E+00	0.00	0	3
4	6	32	4	0.00E+00	0.00E+00	0.00	0	4
5	10	115	4	1.00E+00	0.00E+00	0.01	0	4
6	10	115	3	1.00E+00	0.00E+00	0.01	0	3
7	10	115	2	0.00E+00	0.00E+00	0.00	0	2
8	10	115	1	1.00E+00	0.00E+00	0.01	0	1
9	25	1004	7	3.10E+01	0.00E+00	0.03	0	7
10	25	1004	6	3.30E+01	0.00E+00	0.03	0	6
11	25	1004	8	3.20E+01	0.00E+00	0.03	0	8
12	25	1004	9	3.50E+01	0.00E+00	0.03	0	9
13	50	4890	15	1.32E+03	1.00E+00	0.27	0	15
14	50	4890	16	1.33E+03	2.00E+00	0.27	0	16
15	50	4890	14	1.18E+03	1.00E+00	0.24	0	14
16	50	4890	10	1.34E+03	0.00E+00	0.27	0	10
17	75	12140	19	8.32E+03	9.00E+00	0.69	0	19
18	75	12140	15	9.36E+03	5.30E+01	0.77	0	15
19	75	12140	18	8.64E+03	6.00E+00	0.71	0	18
20	75	12140	16	8.65E+03	7.00E+00	0.71	0	16
21	100	23025	22	3.76E+04	1.90E+01	1.63	0	22
22	100	23025	23	3.57E+04	3.00E+00	1.55	0	23
23	100	23025	19	3.87E+04	1.30E+01	1.68	0	19
24	100	23025	24	3.87E+04	3.30E+01	1.68	0	24
25	125	37718	34	6.00E+04	6.60E+01	3.48	0	34
26	125	37718	29	6.00E+04	6.70E+01	3.24	0	29
27	125	37718	36	6.00E+04	1.40E+01	3.65	0	36
28	125	37718	31	6.00E+04	6.20E+01	3.22	0	31
29	150	56369	37	6.00E+04	1.36E+02	6.02	0	37
30	150	56369	35	6.00E+04	1.40E+01	5.75	0	35
31	150	56369	41	6.00E+04	3.24E+02	6.03	0	41
32	150	56369	39	6.00E+04	1.10E+02	6.03	0	39
33	175	79083	42	6.00E+04	2.10E+01	9.60	0	42
34	175	79083	45	6.00E+04	3.17E+02	9.82	0	45
35	175	79083	53	6.00E+04	3.33E+02	10.91	0	53
36	175	79083	43	6.00E+04	7.40E+01	8.90	0	43
37	200	105966	54	6.00E+04	3.30E+01	15.47	0	54
38	200	105966	52	6.00E+04	9.90E+01	14.45	0	52
39	200	105966	51	6.00E+04	2.57E+02	14.60	0	51
40	200	105966	61	6.00E+04	8.10E+01	15.75	0	61

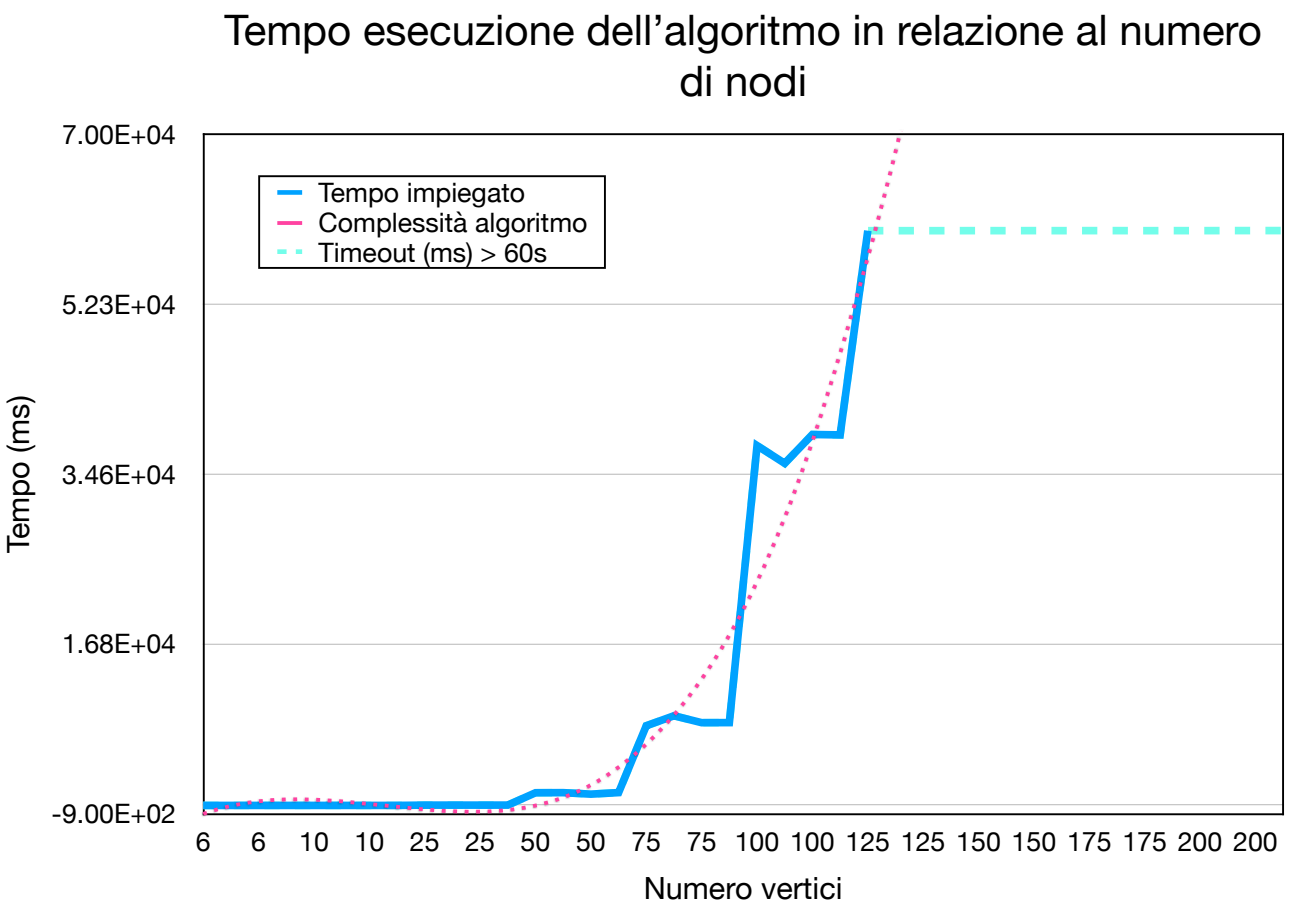
Come si può vedere dalla tabella, l'errore è zero, quindi il taglio calcolato dall'algoritmo è proprio il minimo, mentre il numero di ripetizioni di Karger non corrisponde esattamente al k calcolato, infatti i nodi che hanno tempo impiegato pari a $6.00E+04$ sono andati in timeout e quindi l'algoritmo sarà stato ripetuto $k' \leq k$ volte.

3 Domanda 1



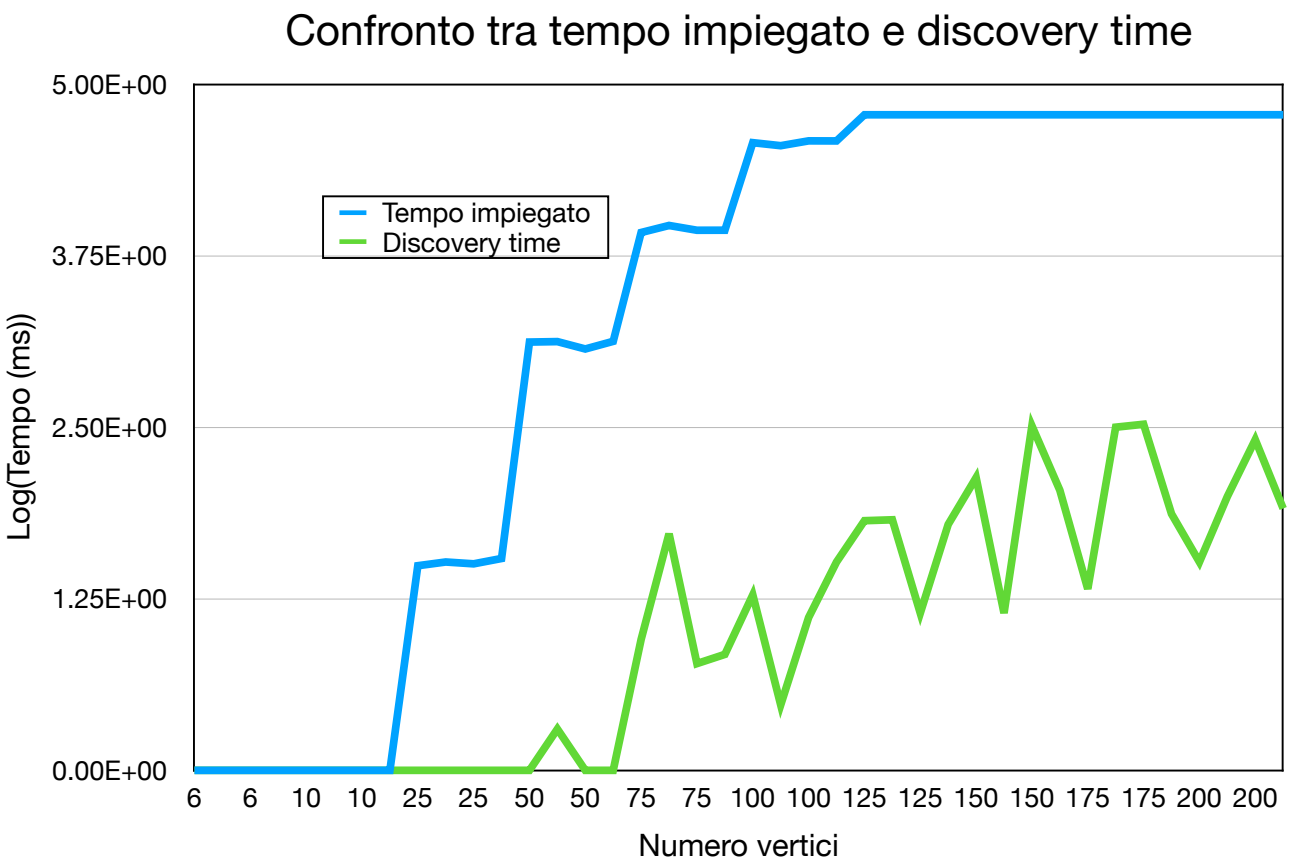
Il tempo medio per eseguire una Full-Contraction in base al numero di vertici rimane abbastanza vicino alla sua complessità asintotica.

4 Domanda 2



Il numero di ripetizioni scelto è $k = (n^2)/2 * log(n)$ dove n è il numero di vertici, in modo da garantire che la probabilità di sbagliare sia minore o uguale a $1/n$. Il tempo di esecuzione dell'algoritmo segue l'andamento della sua complessità asintotica, ma per i grafi più grandi è stato necessario introdurre un timeout di 60 secondi per mantenere un tempo di esecuzione totale ragionevole.

5 Domanda 3



Il grafico mette in relazione il discovery time vs il tempo impiegato applicando la scala logaritmica se il valore del tempo è maggiore di 0, altrimenti lasciamo zero. Come si può evincere dal grafico, fino a 10 nodi il tempo di esecuzione è brevissimo, e le due misurazioni sono equivalenti. È importante evidenziare che all'aumentare del numero di vertici le due misurazioni aumentano. Come evidenziato dal discovery time, nonostante il timeout, l'algoritmo riesce comunque a trovare il taglio minimo molto prima, ovviamente l'interruzione dell'esecuzione non permette di eseguire tutti i confronti necessari, ma il fattore d'interesse è garantire un'alta probabilità di successo in relazione al risultato ritornato.

6 Domanda 4

Come si può vedere nella tabella dei risultati nella sezione 2 il taglio ottenuto dall'algoritmo è esattamente il taglio minimo atteso. Questo perché, anche nelle istanze più grandi del dataset, l'algoritmo ha trovato la soluzione ottima prima del timeout di 60 secondi che abbiamo imposto. L'errore relativo è quindi 0 per ogni grafo.

7 Conclusioni

L'algoritmo risulta molto efficace a risolvere il problema del minimum cut nonostante la sua semplicità. È interessante notare che il discovery time in genere è molto più breve del tempo di esecuzione totale, e che le iterazioni aggiuntive servono semplicemente a garantire che il risultato sia giusto con alta probabilità. Infatti, anche le istanze da 200 vertici hanno ritornato il minimum cut nonostante siano state interrotte molto prima che l'algoritmo completasse tutte le iterazioni. Ovviamente imporre un timeout troppo piccolo riduce drasticamente la probabilità di successo, al punto di rendere l'algoritmo inaffidabile e poco utile per i grafi più grandi.