DIEGO BORSOI
N° 129804
borsoi.diego@spes.uniud.it

PROGETTO DI LABORATORIO DI ALGORITMI E STRUTTURE DATI A.A. 2016/2017

Indice:

Descrizione del Problema

Il progetto consiste nell'acquisizione di un testo come input (composto da qualunque carattere presente nell' ASCII esteso, quindi valori da 0 a 255) e la suddivisione delle parole presenti in esso: per parole si intendono le sequenze di caratteri diverse da ' '(spazio, carattere corrispondente al numero 32 in ASCII).
A questo punto si deve generare il grafo corrispondente, dove i nodi rappresentano le parole inserite senza tener conto delle ripetizioni e gli archi vengono creati secondo la seguente regola:

> esiste un collegamento da una parola X ad una Y se nell'array contenente la frequenza di ciascun carattere presente in X, esiste almeno una casella con valore maggiore alla rispettiva dell'array di Y e le restanti caselle sono maggiori od uguali

Dopodichè dal grafo bisogna calcolare la lunghezza del più lungo cammino possibile fra due suoi nodi. Infine stampare in output la lunghezza massima trovata ed il grafo creato in formato DOT.

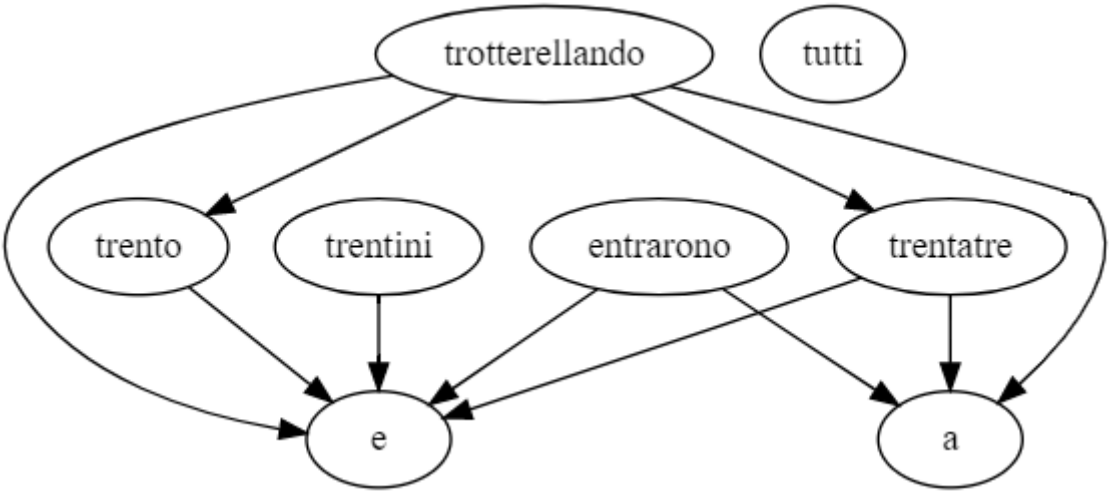
Esempio:

input: "trentatre trentini entrarono a trento tutti e trentatre trotterellando"

output:

```
2
digraph G_T {
0 [label="a"];
1 [label="e"];
2 [label="tutti"];
3 [label="trento"];
4 [label="trentini"];
5 [label="entrarono"];
6 [label="trentatre"];
7 [label="trotterellando"];
3 -> 1;
4 -> 1;
5 -> 0;
5 -> 1;
6 -> 0;
6 -> 1;
7 -> 0;
7 -> 1;
7 -> 3;
7 -> 6;
}
```

Visualizzazione:



Soluzione proposta

Il programma che risolve tale problema è diviso in 4 parti principali:

1. **Acquisizione input ed eliminazione dei doppioni**

Viene letta la sequenza di caratteri in input e generata la lista di parole secondo gli spazi inseriti. Dopodichè vengono ordinate secondo la lunghezza ed inserite in una lista di gruppi che le divide in base ad essa.
A questo punto ogni gruppo viene ordinato lessicograficamente per poi effettuare una scansione che permette di eliminare le parole che si ripetono.

2. **Creazione del grafo**

Il grafo viene creato grazie ad una classe apposita.
Per prima cosa viene creato un nodo per ogni parola, dopodichè per ciascuno di essi viene effettuata la ricerca e la creazione di tutti gli archi possibili attraverso un algoritmo apposito.

3. **Ricerca cammino più lungo**

Per effettuare la ricerca del cammino più lungo viene effettuata una visita DFS del grafo, durante la quale però viene aggiornato ad ogni passaggio l'array contenente la lunghezza del cammino più lungo fra il nodo sulla quale ci si trova ed una "foglia" (un nodo cioè che non presenta archi uscenti, e che ha quindi lunghezza del cammino massimo = 0).
Conclusa la visita il maggior valore presente nell'array corrisponde alla lunghezza del cammino massimo possibile.

4. **Stampa cammino e grafo**

Infine viene effettuata la stampa del valore del cammino di lunghezza massima trovato, seguito dal grafo scritto in formato DOT.

Complessità e correttezza del programma

Il calcolo della complessità viene effettuato relativamente ad un input generico di lunghezza n. Questo studio viene nuovamente diviso nelle 4 componenti del programma.

1. Acquisizione input ed eliminazione dei doppioni

Acquisizione input

Il testo viene letto dallo standard input tramite un ciclo while che acquisisce ogni singolo carattere inserito(appartenente all' ASCII esteso, quindi da 0 a 255) finché non viene letto l'EOF (End Of File: carattere di valore negativo, solitamente -1).

Complessità: Θ(n) dato che deve acquisire tutti gli n caratteri in input.

Ordinamento per lunghezza

Le parole vengono ordinate utilizzando *counting sort* dopo aver salvato in un array la lunghezza di ogni singola parola con la rispettiva posizione nella lista.

Complessità: Counting Sort ordina le m parole di lunghezza massima k in O(m + k), essendo però k sempre ≤ ad n ed il numero di parole anch'esso ≤ n, si ottiene una complessità di O(n + n) = O(n). A questo va aggiunto il tempo per calcolare la lunghezza massima delle parole, ma essendo calcolato in Θ(m) = O(n), non influisce sulla complessità complessiva, che rimane quindi O(n).

Raggruppamento

Avendo il nuovo ordinamento delle parole, vengono raggruppate per lunghezza in una lista di gruppi.

Complessità: Viene eseguito su tutte le m parole, quindi Θ(m) = O(n).

Ordinamento gruppi

Ogni singolo gruppo di parole viene ordinato in ordine lessicografico utilizzando *radix sort*.

Complessità: Radix sort su un input di p parole ha complessità O($k * p$), dove k è la lunghezza delle parole, questo solamente se l'algoritmo counting sort usato al suo interno ha complessità lineare.
Generalmente counting sort ha complessità O(q + h), dove q sono il numero di parole ed h il valore massimo dei valori che deve ordinare, ma essendo l'ordinamento di un array di caratteri, h vale al massimo 255, portando la sua complessità a O(q + 255) = O(q), cioè lineare.
Per calcolare la complessità di radix sort prendiamo un caso generico:
m parole distribuite fra g gruppi, dove il gruppo t_i contiene w_i parole lunghe l_i con $i = 1, ..., g$ e $\sum_{i=1}^{g} w_i = m$.
Radix sort viene quindi eseguito per ogni gruppo, avendo una complessità totale di

$$O(\sum_{i=1}^{g}(l_i * w_i))$$

questa sommatoria però rappresenta la somma dei prodotti di tutte le parole per le rispettive lunghezze, che non è altro che l'input n, togliendo gli spazi che delimitano le parole.
Perciò la complessità totale è O(n).

Eliminazione dei doppioni

Viene effettuata una scansione di tutte le parole, gruppo per gruppo, controllando che ogni parola non sia uguale alla successiva, in caso contrario la seconda viene eliminata e si procede con la successiva.

Complessità: Per ogni parola presente vengono eseguiti due confronti (uno con la precedente ed uno con la successiva), tranne per le parole ad inizio od alla fine di un gruppo o per quelle che vengono eliminate, che ne eseguono uno soltanto.
Perciò il caso peggiore si presenta quando si hanno tutte parole diverse in un unico gruppo.
Essendo che ad ogni passaggio vengono confrontati al massimo tutti i caratteri delle due parole avremo che in totale vengono eseguiti un numero di confronti $< 2 * n$.
Quindi la complessità sarà O(n).

2. Creazione del grafo

Creazione grafo ed inserimento nodi

Per la creazione del grafo vengono istanziati due array: il primo che conterrà i nodi, ed il secondo che conterrà la lista di adiacenze di ogni nodo inserito.
A questo punto viene eseguito un ciclo for sulle parole, le quali vengono inserite nel grafo tramite la loro posizione nella lista in cui sono salvate (partendo da 0).

Complessità: La creazione del grafo viene eseguita in tempo costante Θ(1).
I nodi invece vengono inseriti uno ad uno, quindi la complessità è Θ(m) (con m = numero di parole) che diventa O(n) perchè il numero di parole è sempre inferiore al numero di caratteri inseriti.

Creazione alfabeto ridotto

Per la creazione degli archi del grafo viene prima generato un array che permette di collegare la posizione di un carattere dell'alfabeto dell'ASCII esteso con quella nell'alfabeto ridotto.
Tale alfabeto viene generato per la successiva creazione degli array delle frequenze dei caratteri di ogni parola evitando quindi i caratteri che non sono presenti in nessuna parola.
Viene cioè creato un array di lunghezza 256 (con posizioni da 0 a 255) dove in ogni casella è indicata la posizione del carattere nell'array dell'alfabeto ridotto, per fare ciò viene utilizzato un array dove sono indicati i caratteri presenti in input e quali no (era stato creato all'acquisizione dell'input ed aggiornato ad ogni carattere letto).

Complessità: L'array viene creato in Θ(256) e successivamente viene eseguito un ciclo for che lo percorre completamente, quindi ancora Θ(256).
Perciò la complessità totale è di $\Theta(2 * 256)$ che essendo una costante equivale a Θ(1).

Creazione archi

La creazione degli archi viene eseguita partendo dalle parole di lunghezza maggiore, le quali una ad una vengono "confrontate" con tutte quelle appartenenti a gruppi precedenti nella

lista, cioè contenenti parole di lunghezza minore. Questa procedura viene ripetuta fino ad arrivare alle parole di lunghezza minore, che non possono avere archi uscenti.
Tutto questo è possibile grazie ai teoremi enunciati di seguito, i quali permettono di evitare confronti fra parole della stessa lunghezza e fra parole dove la seconda ha lunghezza maggiore della prima.
Come enunciato nella descrizione del problema il "confronto" fra due parole avviene attraverso l'array delle frequenze dei caratteri di queste.

Complessità: Tale procedura ha complessità O(n^2). Questo è dato dal numero di archi presenti nel caso peggiore:
ci sono due gruppi contenenti lo stesso numero di parole rispettivamente di lunghezza l_1 e l_2, con $l_1 < l_2$ ed ogni parola del secondo gruppo è in relazione con tutte quelle del primo, cioè ha un arco uscente verso ogni parola del primo gruppo. Se questi gruppi hanno un numero di elementi appartenete a Θ(n) allora il numero totale di archi sarà Θ(n^2).
Un esempio si ha quando l'input è composto da x parole diverse di lunghezza l_1 ognuna seguita da uno spazio (la diversità si può ottenere attraverso la permutazione dei caratteri), e poi altre x parole diverse di lunghezza l_1 +1 seguite anch'esse ciascuna da uno spazio(per la diversità si può usare anche in questo caso lo stratagemma delle permutazioni). L'input è quindi composto da $l_1 * x + x + x * (l_1 + 1) + x$ caratteri, quindi avremo che

$$(l_1 * x + x + x * (l_1 + 1) + x) = n$$
$$(l_1 * x + x + x * l_1 + x + x) = n$$
$$(2 * l_1 * x + 3 * x) = n$$
$$x * (2 * l_1 + 3) = n$$
quindi $x = \frac{n}{2*l_1+3}$

essendo l_1 << n , x appartiene a Θ(n).

Teorema 1:

> *Se una parola x ha lunghezza maggiore di un'altra parola y,*
> *allora non può esserci una arco da y verso x.*

Dimostrazione:
Per assurdo ipotizziamo che |y| > |x| (lunghezza di y > lunghezza di x) e che ci sia un arco da x a y.
Questo implicherebbe l'esistenza di un carattere c tale che il suo numero di ripetizioni nella parola x è maggiore di quelle nella parola y, ma questo porterebbe ad avere |x| > |y|, avendo x almeno un carattere c in più di y. Si genera quindi un ASSURDO.

Teorema 2:

> *Se una parola x ha lunghezza uguale ad un'altra parola y,*
> *allora non può esserci un arco da y verso x e nemmeno da x verso y.*

Dimostrazione:
Per assurdo ipotizziamo che |y| = |x| (lunghezza di y = lunghezza di x) e che ci sia un arco da x a y (per il caso simmetrico si procede con la stessa logica).
Questo implicherebbe l'esistenza di un carattere c tale che il suo numero di ripetizioni nella parola x è maggiore di quelle nella y, ma questo porterebbe ad avere |x| > |y|, avendo x almeno un carattere c in più di y. Si genera quindi un ASSURDO.

3. Ricerca del cammino di lunghezza maggiore

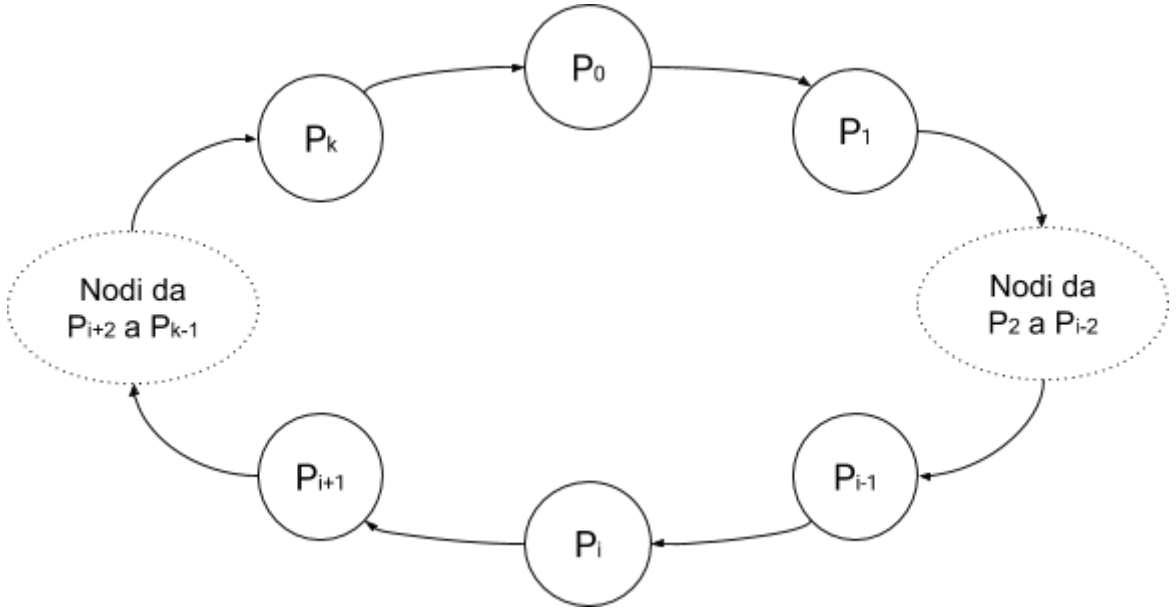
Per trovare la lunghezza del più lungo cammino percorribile nel grafo, viene utilizzata una visita DFS leggermente modificata.
Inizialmente viene creato un ulteriore array (inizializzato a 0) di lunghezza uguale al numero di nodi presenti, nel quale viene salvata la lunghezza del cammino massimo percorribile da quel nodo.
Poi in ogni nodo controllo se non ha archi uscenti, in questo caso la sua lunghezza (il cammino più lungo eseguibile da questo nodo) è 0; se invece presenta degli archi uscenti controllo il colore del nodo a cui punta ciascuno di essi : se è grigio proseguo, se invece è bianco eseguo la visita DFS su di esso. Successivamente indipendentemente dal colore, controllo se la lunghezza di tale nodo + 1 è maggiore di quella salvata nel nodo in cui ci si trova, nel caso affermativo la aggiorno a questo nuovo valore.
Alla conclusione della visita il valore massimo dell'array delle lunghezze rappresenta la lunghezza del massimo cammino percorribile.
La correttezza di questo algoritmo è data dal fatto che il grafo in questione è aciclico ed inoltre si basa sulla correttezza della visita DFS (dimostrati entrambi in seguito).

Complessità: Ha la stessa complessità di una visita DFS, quindi Θ(|V| + |E|): la cardinalità dei nodi sarà sempre O(n) , invece la cardinalità degli archi sarà sempre O(n^2) come spiegato in precedenza.
La visita ha quindi complessità O(n^2).

Teorema 3:

Qualunque grafo creato secondo tali regole è aciclico.

Dimostrazione:
Ipotizziamo per assurdo che esista un ciclo all'interno di un grafo.
Tale ciclo è composto dai nodi P_0,...,P_k e da k archi come in questa immagine:



A questo punto prendiamo un nodo P_i all'interno di essi, con i=1,...,k.
Esiste quindi un cammino che conduce da P_0 a P_i, il quale implica che la parola del nodo P_0 ha una lunghezza maggiore di quella del nodo P_i (a causa del *Teorema 1*).

Esiste però anche un cammino che conduce dal nodo P_i al nodo P_0.
Questo secondo cammino implica che la lunghezza della parola del nodo P_i è maggiore della lunghezza di quella del nodo P_0 (sempre a causa del *Teorema 1*), creando quindi un ASSURDO.

Dimostrazione correttezza ricerca cammino massimo:
La dimostrazione viene eseguita sui vari tipi di nodi presenti:

nodo "foglia": un nodo che non presenta archi uscenti.
Lunghezza del cammino massimo è impostato a 0.

nodo "interno": un nodo che presenta degli archi uscenti.
La lunghezza del cammino massimo da questo nodo viene impostato come: il massimo + 1 della lunghezza dei nodi raggiungibili tramite un arco uscente da questo nodo. Tali nodi possono essere solamente di 2 colori: bianco , un nuovo nodo non ancora visitato sul quale quindi viene effettuata la visita DFS, oppure nero, un nodo già visitato da un'altra visita, che quindi possiede già una lunghezza del cammino massimo da tale nodo. Non è possibile incontrare nodi grigi perchè un tale arco indurrebbe la presenza di un ciclo all'interno del grafo (cosa impossibile dal *Teorema 3*).

4. **Stampa del grafo**

Per ultimo viene stampata la lunghezza del cammino massimo trovato, seguita dalla stampa del grafo in formato DOT.

Complessità: La stampa della lunghezza del cammino avviene in un tempo costante Θ(1).
La stampa del grafo invece avviene in Θ(|V| + |E|) essendo che vengono stampati tutti gli archi ed i nodi. La complessità della stampa è quindi O(n^2) (vedere complessità della visita DFS discussa in precedenza).

- **Complessità totale**

Il programma completo avrà quindi una complessità totale data da:

$$\Theta(n) + O(n) + O(n) + O(n) + O(n) + O(n) + \Theta(1) + O(n^2) + O(n^2) + O(n^2)$$
$$=$$
$$O(n^2)$$

Misurazione dei tempi

Per la misurazione dei tempi effettivi impiegati dal programma e da ogni sua componente sono stati utilizzati gli algoritmi spiegati a lezione.

- **Calcolo della granularità e di *tMin***

La granuralità del sistema, cioè il più piccolo intervallo di tempo misurabile dal sistema utilizzato, viene calcolata utilizzando l'algoritmo 4 degli appunti.
A questo punto bisogna calcolare il *tMin* cioè l'intervallo di tempo minimo misurabile di un evento avendo un errore ragionevole.
In questo caso si vuole ottenere una misurazione con un errore tollerato del 5%.

Viene quindi calcolato facendo: $tMin = Granularità / 0.05$

- **Calcolo delle ripetizioni**

Per effettuare una misurazione che rispetti la precisione scelta, bisogna eseguire il programma per un certo numero di volte *rip*. Tale numero viene calcolato utilizzando l'algoritmo 5, il quale usa il tempo *tMin* calcolato in precedenza.
Questo calcolo verrà eseguito per il programma da testare, ma anche per la parte del programma che ha il compito di preparare l'input.

- **Calcolo tempo medio netto**

Questa funzione (algoritmo 7) permette di misurare il tempo impiegato da una singola parte del programma, escludendo la porzione che la precede.
Per far ciò viene inizialmente calcolato il numero *rip* della "tara",la parte di programma che precede la funzione desiderata, ed il numero *rip* del programma fino alla parte da misurare compresa.
Dopodichè vengono misurati i tempi di queste due parti del programma usando il numero di ripetizioni corrispondente, per poi sottrarre dal secondo tempo calcolato (quello comprendente la porzione reale di codice da misurare) quello impiegato dalla "tara".

- **Misurazione**

Quest'ultima funzione utilizzata (algoritmo 9) permette di calcolare il tempo impiegato dalla parte di programma in esame sapendo che tale risultato approssima il reale tempo di esecuzione, con una certa precisione scelta. Tale algoritmo fa uso della varianza campionaria per assicurarsi di rientrare nell'intervallo di confidenza voluto.
La precisione viene impostata tramite il parametro *za*, il quale viene scelto partendo da un coefficiente α : nel nostro caso α viene impostato a 0.05 e di conseguenza *za* = 1.96
Il parametro c viene invece impostato a 5, invece l'ultimo parametro di questa funzione corrisponde ad un delta (Δ) che viene usato per decidere se il valore misurato è accettabile; tale valore viene scelto eseguendo una misurazione della parte di programma in esame ed impostandolo ad un decimo del valore ottenuto.

- **Tempi ottenuti**

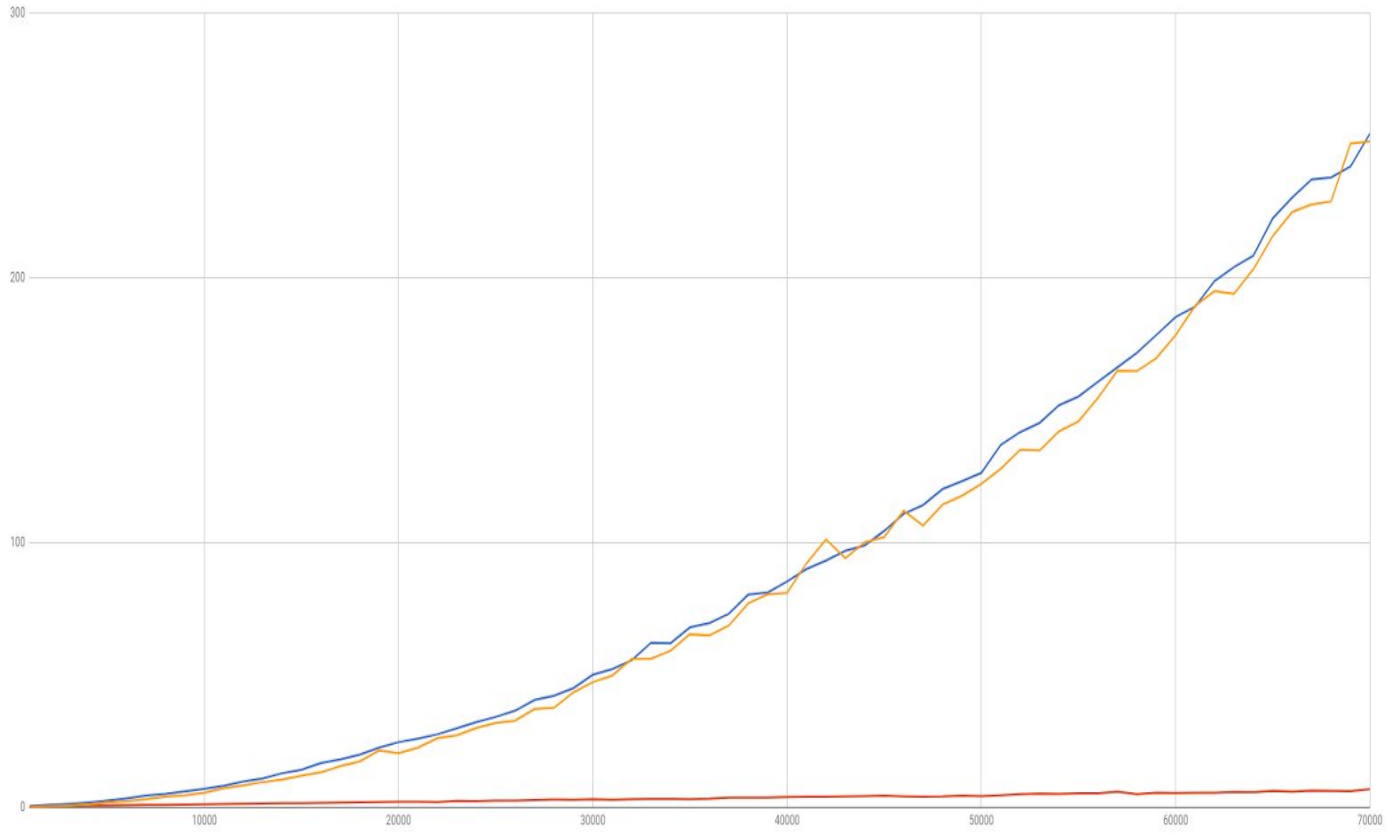
Per la creazione dell'input nei seguenti esempi viene utilizzato l'algoritmo 8 degli appunti, utilizzando come seed il numero 14081996.
Sono state eseguite le misurazioni utilizzando due tipologie di algoritmi differenti per la creazione dell'input, entrambi però eseguiti per generare stringhe da 1000 a 70000 caratteri (con un passo di 1000) :

Input composto di parole di lunghezza tra 1 e 40

In questo caso l'input è composto da parole di lunghezza massima di 40 caratteri: inizialmente viene scelta randomicamente la lunghezza della parola da inserire, e dopodichè vengono creati i caratteri che la compongono (da 0 a 255, escludendo lo spazio).

Lunghezza input (caratteri)	Tempo - totale (ms)	Delta - totale (ms)	Tempo - Input (ms)	Delta - Input (ms)	Tempo - Grafo (ms)	Delta - Grafo (ms)
1000	0.4037522949	0.008742377526	0.2606658983	0.02030075165	0.07917627402	0.008295987128
2000	0.8726217591	0.03661917875	0.3953023158	0.0238715669	0.3157661783	0.02456286162
3000	1.244196927	0.06332038348	0.4903456145	0.0183208632	0.565	0.04919560956
4000	1.773120197	0.1163596945	0.6255253517	0.03081239575	0.9946428571	0.0709300712
5000	2.513099129	0.08534069828	0.6790438133	0.04148207855	1.729761905	0.08864210186
6000	3.361070853	0.04383822227	0.8245633085	0.02914604909	2.283333333	0.05843590981
7000	4.433809771	0.3688237724	0.9152357128	0.05817720039	3.026666667	0.2125794471
8000	5.017443898	0.2311717258	0.9066395373	0.02575730607	3.935	0.04720305075
9000	5.975106044	0.1740792182	1.033542693	0.04311857983	4.52	0.03506154589
10000	6.947222222	0.1761580975	1.094282051	0.0537657002	5.4625	0.3052192654
11000	8.12275	0.2282423764	1.224080268	0.06918987014	7.125	0.4416800312
12000	9.706818182	0.2902231231	1.339891775	0.02147059432	8.175	0.2968445721
13000	10.8427381	0.5871416327	1.434570863	0.08518135917	9.525	0.2237726277
14000	12.81644737	0.5825236438	1.49744582	0.08602005574	10.42916667	0.4085945831
15000	14.09885621	0.3364050291	1.515543301	0.04656832852	11.9	0.3247007511
16000	16.68988971	0.5584832175	1.609227941	0.1028670524	13.13333333	0.4702184835
17000	18.06083333	0.3367943631	1.691458333	0.1034565505	15.5	0.5165053512
18000	19.8252381	0.44479498	1.822069597	0.08942735051	17.33333333	0.6317432671
19000	22.49990842	0.7642280155	1.992925824	0.1806571614	21.4	1.212985078
20000	24.57591575	0.5512734998	2.012728938	0.07056159013	20.36666667	1.004573033
21000	25.94313187	0.329213313	2.020687646	0.08728533985	22.48333333	0.7568521946
22000	27.54372711	0.3955195387	2.000757576	0.06042986737	26.125	1.055492302
23000	29.79204545	0.6641850262	2.364734848	0.1231009448	27.2	1.256159703
24000	32.19545455	0.210299841	2.309090909	0.1122474745	29.9	1.222842099
25000	34.10909091	0.7894882284	2.470454545	0.0770108332	31.85	1.493334792
26000	36.46181818	0.7209204215	2.505	0.1749787187	32.675	1.187378288
27000	40.54	1.649698363	2.731111111	0.1082160546	37.15	1.4901157
28000	42.11111111	1.073391519	2.858333333	0.1447444235	37.525	0.3627324083
29000	44.99055556	0.5648082516	2.85	0.200840235	43.4	2.227641533
30000	50.04097222	0.7977002984	3.036805556	0.10397321	47.25	1.926641119
31000	52.15	1.509798535	2.815972222	0.1460644095	49.75	2.195724937
32000	55.50902778	0.8174049626	3.0375	0.114128984	56.0125	3.110332655
33000	62.19375	2.114280445	3.11875	0.1312523333	56.1	1.049561242
34000	61.97678571	1.357669354	3.139285714	0.1775952561	59.125	1.084661237
35000	67.975	2.878119186	3.047321429	0.07987365022	65.375	2.597462223
36000	69.53660714	1.764182977	3.208928571	0.2149353391	64.925	1.48721256
37000	73.12321429	1.164699876	3.691666667	0.3112775111	68.625	1.58475424
38000	80.35714286	3.747243253	3.594047619	0.1789044438	77.05	3.627059305
39000	81.09285714	1.279388604	3.605952381	0.2595438734	80.475	3.2126965
40000	85.29166667	1.794999226	3.816666667	0.3336160801	80.975	2.094312966
41000	90.00833333	1.920600103	3.97202381	0.301244313	92.175	2.17551171
42000	93.19285714	2.356379256	4.008333333	0.2545066688	101.15	5.070209739
43000	96.9297619	1.424948155	4.021428571	0.3657993986	94.125	4.837866059
44000	98.94761905	2.539475913	4.15	0.25887174	100.1	4.133695296

45000	104.425	1.158492833	4.423333333	0.3515698306	102.075	5.471174207
46000	110.8666667	1.983461251	4.108333333	0.1098105439	112.05	3.620433565
47000	114.1983333	2.072773492	3.966666667	0.1209108028	106.425	0.9691600487
48000	120.21	2.229606613	4.05	0.1164143939	114.35	2.442540644
49000	123.2	1.33242364	4.416666667	0.196	117.675	2.755874308
50000	126.34	1.091882051	4.141666667	0.1940850902	122.2	2.414064456
51000	136.93	3.54945123	4.46	0.2275626683	127.95	2.324893804
52000	141.7633333	3.808349435	4.964166667	0.3692493107	135.1	1.417447283
53000	145.24	3.145589339	5.12	0.4076611534	134.85	0.7666582029
54000	151.92	2.465178264	5.01	0.2635453054	142.05	3.194860122
55000	155.2	1.459377621	5.2075	0.2900203993	145.8	2.206197453
56000	160.7925	3.80971518	5.1875	0.2477288033	154.65	3.894436647
57000	166.22	3.767648204	5.8375	0.2777050954	164.9	7.12049414
58000	171.71	3.415510194	4.96	0.2576486911	164.8	2.507727896
59000	178.4	2.649989585	5.43	0.3191852628	169.65	3.605014951
60000	185.2325	4.073073826	5.385	0.2016038889	178.35	4.766065547
61000	189.13	6.566298381	5.4775	0.3428879817	189.45	5.914850057
62000	198.84	3.84613732	5.4925	0.7018674889	195	7.983241447
63000	204.085	3.313374058	5.795	0.2280053771	193.95	3.103367719
64000	208.3975	0.6643937447	5.6175	0.154734521	203.25	3.742009086
65000	222.55	3.653606396	6.155	0.6163335233	215.85	4.273069716
66000	230.3875	2.642004391	5.8375	0.3279707304	224.95	6.875941477
67000	237.2575	2.764039575	6.2625	0.4666590833	227.8	4.82375074
68000	237.9775	2.72413031	6.175	0.2090411443	228.9	4.075197762
69000	242.0875	2.539174137	6.1125	0.1910371692	250.8	3.353261576
70000	254.575	3.3720047	6.8375	0.3929176759	251.65	7.24617063

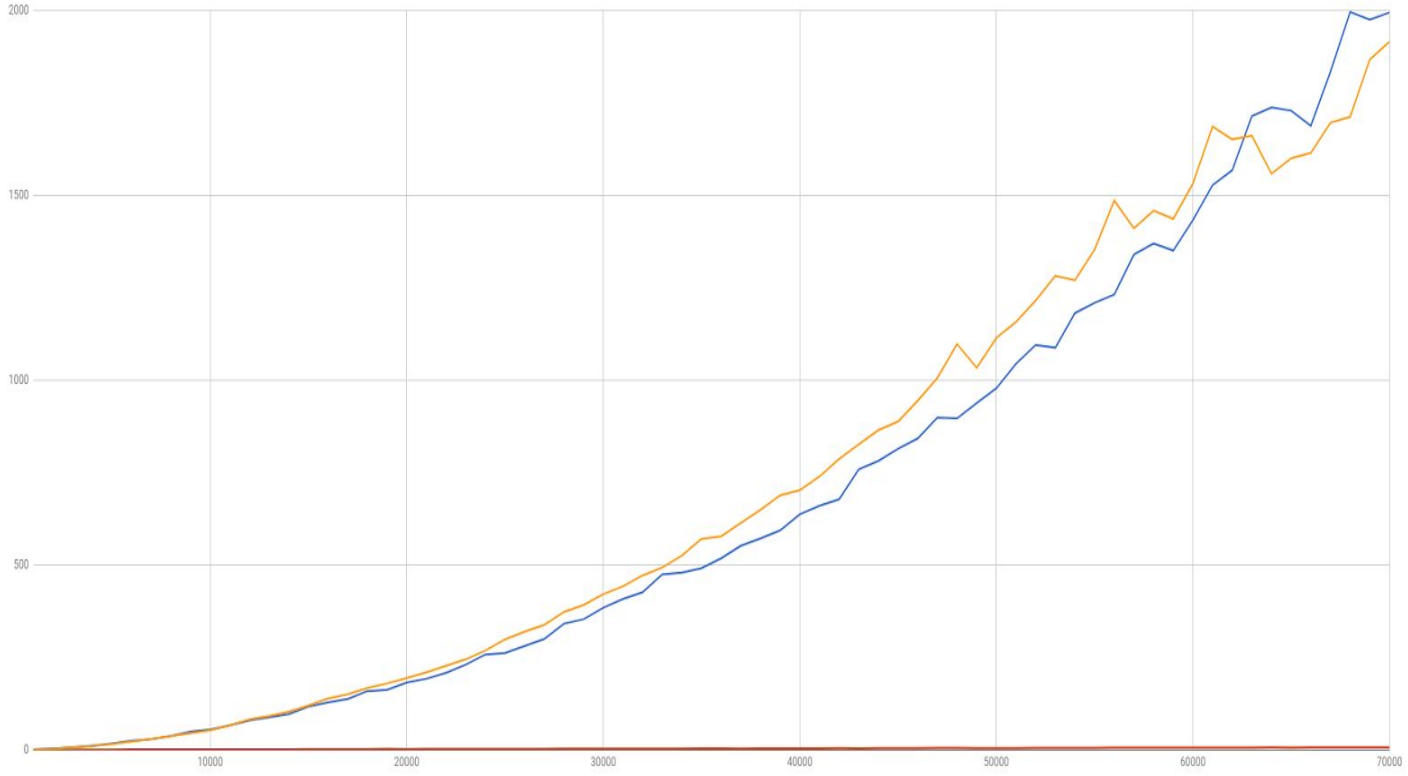


Input con spazio presente con probabilità > 18%

In questo secondo caso l'input è costruito scegliendo ogni carattere singolarmente, però lo spazio ha una probabilità di essere scelto > 18% (questo valore è stato scelto perché rappresenta la frequenza del carattere spazio in un testo normale).
L'esatta probabilità del carattere spazio è $\frac{18}{100} + \frac{1}{256}$ essendo che può essere comunque scelto anche quando il carattere non deve essere per forza uno spazio, ogni altro carattere ha probabilità di essere scelto di $\frac{1}{256}$.

Lunghezza input (caratteri)	Tempo - totale (ms)	Delta - totale (ms)	Tempo - Input (ms)	Delta - Input (ms)	Tempo - Grafo (ms)	Delta - Grafo (ms)
1000	0.9505901669	0.01468014145	0.1818426599	0.008943090319	0.6521436295	0.01700247824
2000	2.987078652	0.1317251295	0.3020488036	0.02177318134	2.651568627	0.09007086864
3000	6.355324358	0.08756276381	0.3855549483	0.01886990563	5.680705128	0.2070681811
4000	10.76843091	0.3291534408	0.4623589379	0.03405936833	10.13111111	0.4501983425
5000	16.73864967	0.8445929659	0.5679185868	0.02535878871	15.4125	0.3284279601
6000	24.10141931	1.886409229	0.6570613553	0.01598434833	22.24047619	0.5897067821
7000	28.57061265	1.05803606	0.7565217391	0.02910034931	29.19	1.557034659
8000	36.72959273	1.173343223	0.8849600564	0.06751264369	37.20166667	1.492050849
9000	49.18812887	0.9236754587	0.9913602941	0.09643063386	44.7775	1.154579754
10000	54.74916667	1.240366308	1.000089286	0.02940041666	52.3425	1.256113829
11000	65.71754579	1.814315335	1.0625	0.049	66.1625	1.052416885
12000	79.33003663	5.563513911	1.175641026	0.04675467047	82.49583333	1.843656427
13000	87.9625	2.279228737	1.273257576	0.02607979019	91.6	0.9955565278
14000	96.62727273	3.397217555	1.280833333	0.0546212772	102.85	1.974861435
15000	116.4763636	8.808657517	1.391666667	0.05770083766	119.8333333	0.798831925
16000	128.24	7.728374694	1.447361111	0.03155054381	138.7833333	1.549929203
17000	137.1272222	4.473571	1.638611111	0.1089171963	149.9666667	1.719644821
18000	158.8784722	6.626843387	1.901339286	0.128673635	166.7583333	1.10111382
19000	162.2444444	1.298731967	2.09702381	0.08105772977	179.4083333	3.119658115
20000	181.98125	7.487117424	1.969642857	0.1057961483	193.8333333	5.386187541
21000	192.0875	4.447050334	2.27202381	0.09161639531	209.7	2.481742372
22000	207.8294643	3.157659964	2.328571429	0.1506545718	227.125	3.76822425
23000	230.2916667	3.659089261	2.376785714	0.145899585	244.475	1.576247062
24000	257.7642857	8.391456693	2.381666667	0.1385501818	268.475	8.09268786
25000	261.9333333	4.915519202	2.366666667	0.09064632125	298.65	6.69649138
26000	280.6	5.296313468	2.458333333	0.1200249974	319.825	10.72587503
27000	299.6	6.425899515	2.625	0.08946135106	337.775	6.132012785
28000	341.1916667	6.150821925	2.783333333	0.1209108028	372.775	8.867348465
29000	353.12	4.600419191	2.853333333	0.1356827214	391.9	6.434280286
30000	384.24	6.897811527	2.805	0.0944061015	420.925	7.988833532
31000	407.72	11.03988508	2.945	0.2052860249	442.075	7.648584523
32000	426.08	7.674024521	3.11	0.1390080861	471.725	9.196974981
33000	474.31	17.81840301	3.21	0.2418833769	493	7.034529835
34000	479.24	11.08786396	3.055	0.110353396	525.325	10.22060601
35000	491.1625	5.792463245	3.495	0.077661522	570.725	14.96152234
36000	518.2725	17.07104361	3.5775	0.1094357711	577.775	12.32374841
37000	552.12	3.688197157	3.4425	0.1496867796	614.125	14.13002321
38000	571.8025	10.38208457	3.4975	0.1074430454	649.4	15.68710339
39000	593.7375	4.148740616	3.8	0.1359694819	688.525	12.37371766
40000	637.275	10.03840076	4.0875	0.1411676309	702.5	12.20326284

41000	660.0125	12.23668436	4	0.1428583214	739.3	5.755195496
42000	677.7375	5.412382165	4.25	0.2750118179	787.1	22.35800789
43000	759.075	23.69942634	4.1	0.02683840532	826.65	21.19139818
44000	781.625	11.37597027	4.266666667	0.2981448567	865.05	29.52042403
45000	814.525	17.41067983	4.45	0.2147072425	888.35	23.57734177
46000	842.35	35.53093095	4.545833333	0.1425779319	944.6	27.1559397
47000	898.875	24.83222153	5	0.417219966	1006.5	32.06053187
48000	896.7875	13.30535429	4.9625	0.2143341628	1097.9	66.23416838
49000	938.4041667	17.25577669	4.820833333	0.1727013479	1033.3	35.79493432
50000	978.175	22.9591904	4.816666667	0.1927056477	1114.55	11.88300009
51000	1044.516667	60.21449356	4.854166667	0.2917226613	1157.65	22.6079012
52000	1095.4	24.43800298	5.15	0.1812926425	1215.8	22.93049226
53000	1088.033333	7.849331669	5.066666667	0.2328287878	1282.25	20.72972204
54000	1181.733333	38.40381473	5.266666667	0.1691111403	1270.75	24.70066623
55000	1209.4	42.63477833	5.533333333	0.1093235768	1353.8	48.69927559
56000	1232.133333	37.94020917	5.633333333	0.1573435025	1486.05	97.10325438
57000	1340.041667	30.83693538	5.883333333	0.4043283594	1411.1	20.33758562
58000	1370.133333	92.77138482	5.75	0.2487819215	1459.1	26.76659703
59000	1351	59.20186047	5.833333333	0.3666824239	1436.7	42.66443253
60000	1433.166667	45.3286697	5.9	0.2748662705	1532.1	66.35701504
61000	1527.833333	44.01079026	5.95	0.4769780801	1686.3	127.3793731
62000	1568.2	21.04588377	6.166666667	0.3363239179	1651.8	40.3592033
63000	1714.583333	119.5265254	6.2	0.3253573762	1661.8	126.9799731
64000	1738.016667	83.47676842	6.45	0.3881701803	1558.8	47.50289578
65000	1729.55	83.38041854	6.183333333	0.1753077294	1600.65	35.39349744
66000	1688.666667	27.64665516	6.966666667	0.2196210878	1615.1	52.72101255
67000	1835.15	93.41941804	6.708333333	0.1847905722	1696.95	21.74557949
68000	1996.625	125.4121845	6.941666667	0.4269222151	1712.8	34.23060179
69000	1975.5	121.8754064	6.875	0.216686153	1868.1	124.8085001
70000	1995.075	40.57906401	6.775	0.4064342505	1916	82.10610545



Conclusione

Dai grafici ottenuti si può notare come i tempi ottenuti seguano un andamento polinomiale di secondo grado.
La gestione dell'input invece segue un andamento lineare (nei grafici ottenuti lo si nota poco a causa della differenza dei valori rispetto al tempo totale di esecuzione).
Un'incongruenza riscontrata rispetto alla teoria si ha nel fatto che i tempi del calcolo dei nodi e degli archi del grafo vengano, per alcuni valori, maggiori del tempo totale di esecuzione del programma; questo è dato dal fatto che le misurazioni effettuate possono variare a causa del carico della CPU al momento dell'esecuzione.

Compilazione

Per compilare i due progetti bisogna posizionarsi nella cartella del programma che si vuole compilare ed eseguire il comando:

- per il programma principale

posizionarsi nella cartella ProgettoASD ed eseguire
> javac mainProject\Main.java
e per eseguirlo
> java mainProject/Main
(importante anche su Windows bisogna usare /)

- per il programma del calcolo dei tempi

posizionarsi nella cartella ProgettoTempi ed eseguire
> javac progettoTempi\Main.java
e per eseguirlo
> java progettoTempi/Main
(importante anche su Windows bisogna usare /)

Problemi riscontrati

Un problema è stato riscontrato nella misurazione dei tempi di esecuzione della ricerca del cammino massimo e della stampa, infatti l'algoritmo per tale misurazione dava dei risultati negativi e quindi non sono stati inseriti nei grafici.
Questo può essere dato dal fatto che per tale calcolo viene misurato il programma fino alla creazione del grafo (tempo della tara) e successivamente fino alla stampa (tempo lordo). Essendo eseguiti in momenti diversi la CPU potrebbe avere carichi computazionali diversi, portando quindi ad avere incongruenze nelle misurazioni causate dal ridotto tempo di esecuzione della parte di programma in esame, a sua volta causato dal numero di archi presenti negli esempi utilizzati (molto piccolo rispetto alla grandezza dell'input, a causa della generazione randomica dei caratteri).
Un ulteriore problema consisteva nella stampa dei caratteri '\n' (13) e '\r' (10), infatti questi portavano ad una errata stampa in formato DOT. Come soluzione al posto della stampa di tali caratteri viene stampata la stringa "\n" e "\r" rispettivamente.

Note

Il progetto è stato scritto e testato su un PC con sistema operativo Windows 10 e processore i7-7700HQ