

Studente: Luca Grasso

Matricola: 294612

Corso di Algoritmi e Strutture Dati
Progetto per la sessione estiva 2018/2019

Docente: Ing. Valerio Freschi

Specifica del problema.

Specifica del problema:

Si supponga di dover progettare un programma per l'analisi un sistema di riscaldamento di un edificio. Il sistema rileva il consumo di gas e di energia elettrica (relativi ad ogni ora) e scrive i dati relativi su un file di log in formato testo, secondo il seguente formato (si assumano campi separati da tabulazione o spazio):

- *Tempo*: un codice numerico che risulta dalla concatenazione di anno, mese, giorno e ora della rilevazione.
- *Gas*: un numero reale che rappresenta il consumo in m³ nell'ora di rilevamento.
- *Elettricità*: un numero reale che rappresenta il consumo in KWh nell'ora di rilevamento.

Ad esempio:

Tempo	Gas	Elettricità
2019042908	1.3	2.70
2019042909	0.3	4.26
2019042910	8.3	5.14
...

Si scriva un programma ANSI C che esegue le seguenti elaborazioni:

1. Acquisisce il file e memorizza le relative informazioni in una struttura dati di tipo albero.
2. Ricerca e restituisce il dato relativo ai consumi di una determinata rilevazione. Ad esempio: se l'utente chiede quali consumi sono stati effettuati il giorno 29/04/2019 dalle ore 09 alle ore 10, il programma deve restituire le informazioni contenute nella riga 3 (2019042910, 8.3, 5.14).
3. Permette la cancellazione del dato corrispondente ad una determinata rilevazione (tempo, gas, elettricità) sulla base della richiesta dell'utente (sempre tramite indicazione del tempo).

Il programma deve inoltre prevedere una modalità che implementa le stesse funzionalità utilizzando un array al posto dell'albero. Per quanto riguarda l'analisi teorica si deve fornire la complessità corrispondente ad ognuna delle seguenti operazioni: ricerca; cancellazione. Oltre all'analisi teorica della complessità si deve effettuare uno studio sperimentale della stessa per le operazioni di ricerca e cancellazione. Come suggerimento si può operare generando un numero N di rilevazioni casuali. L'analisi sperimentale deve quindi valutare la complessità al variare di N e confrontare l'algoritmo di ricerca e quello di cancellazione che lavorano su albero con i corrispondenti che lavorano su array.

Analisi del problema.

Analisi del problema:

Analisi degli input:

Per prima cosa, il programma chiede all'utente quale struttura dati vuole utilizzare per memorizzare i dati forniti dalle rilevazioni. Sono disponibili 3 opzioni:

- T. Le informazioni verranno memorizzate in un albero ordinato
- A. Le informazioni verranno memorizzate in un array ordinato
- Q. l'utente esce dal programma

Se l'utente non inserisce una opzione valida (le opzioni valide sono quindi T, A, Q), il programma chiede nuovamente di scegliere una opzione, finché non viene inserita una opzione valida.

Successivamente, il programma chiede all'utente di inserire il nome del file in cui sono contenuti i dati sulle rilevazioni. Il file dovrà contenere i dati nel formato

Tempo	Gas	Elettricità
2019042908	1.3	5.7

La lettura del file è stata implementata in modo che:

- l'intestazione che riporta i titoli delle colonne (Tempo, Gas, Elettricità) può essere presente oppure no. Infatti, se il programma legge una riga che non contiene una cifra come primo elemento, ignora i dati presenti nella riga.
- in ogni riga sono riportati tempo, gas ed elettricità di una singola rilevazione.
- le righe che contengono rilevazioni devono iniziare con una stringa di 10 caratteri che riporta la data e l'ora della rilevazione nel formato aaaammddhh e ci devono essere due stringhe che riportano i valori float di gas ed elettricità. Le tre stringhe possono essere separate da uno o più spazi.

A mano a mano che il programma legge i dati, questi vengono memorizzati nella struttura dati scelta dall'utente (albero o array). Per entrambe le strutture il programma è stato implementato in modo da ignorare una rilevazione che ha un valore di tempo uguale ad una rilevazione già inserita.

Al termine della lettura, l'array viene ordinato con l'algoritmo mergesort.

Successivamente, viene presentato all'utente un menu, le operazione che l'utente può scegliere sono

- R. ricerca di un insieme di rilevazioni comprese tra due valori di tempo specificati dall'utente
- D. cancellazione di un insieme di rilevazioni comprese tra due valori di tempo specificati dall'utente
- Q. uscita dal programma

Dopo ogni operazione, viene ripresentato il menu per consentire all'utente di svolgere più operazioni, finché l'utente non seleziona Q.

Se l'utente inserisce una opzione non valida, il programma chiede di reinserire una opzione.

Se l'utente seleziona ricerca o cancellazione, il programma chiede di inserire data e ora di inizio e fine delle rilevazioni da cercare/cancellare. Sia per l'inizio che per la fine, l'utente deve inserire una stringa di 10 cifre nel formato aaaammddhh. Se la stringa inserita non è valida (cioè non è composta da 10 cifre), il programma chiede di reinserire una stringa valida.

Analisi degli output:

- Se l'utente seleziona l'opzione R, è richiesta la stampa a video dei dati di tempo, gas ed elettricità delle rilevazioni che si sono verificate in un tempo successivo o uguale alla data di inizio e precedente alla data di fine, come da specifica del problema.
- Se l'utente seleziona l'opzione D, è richiesta la cancellazione dei dati di tempo, gas ed elettricità delle rilevazioni che si sono verificate in un tempo successivo o uguale alla data di inizio e precedente alla data di fine, come da specifica del problema.

Relazioni intercorrenti fra input e output:

L'utente inserisce i tempi di inizio e fine nel formato aaaammddhh, cioè nello stesso formato previsto per il file di input. Per trovare e restituire le letture che cadono nel range specificato, si confrontano le stringhe usando l'ordine lessicografico.

Progettazione dell'algoritmo.

Progettazione dell'algoritmo:

Strutture Dati

Come previsto dalle specifiche, il problema è stato risolto utilizzando

- un albero ordinato
- un array ordinato

Sono stati definiti questi tipi di dato

- NodoLettura_t : rappresenta un nodo dell'albero, è una struttura che contiene 5 campi: tempo (stringa), gas, elettricità (float), destro, sinistro (puntatori a strutture di tipo NodoLettura_t)
- Lettura_t : rappresenta un elemento dell'array, , è una struttura che contiene 3 campi: tempo (stringa), gas, elettricità (float).

Funzioni per la struttura Albero

```
int inserisci_nodo_albero(NodoLettura_t **radice, char *tempo, float gas, float elettricità)
```

Questa funzione inserisce un nodo nell'albero, se nell'albero non è già presente un nodo con lo stesso tempo. Inizialmente si percorre l'albero con un ciclo for per cercare la posizione in cui inserire il nuovo nodo e per verificare se è già presente un nodo con lo stesso tempo. Se il ciclo for non individua un nodo già presente nell'albero, si crea un nuovo nodo e lo si inserisce nell'albero.

Se l'albero è vuoto al momento della chiamata alla funzione, il nuovo nodo diventa la radice dell'albero.

```
int ricercaAlbero(NodoLettura_t **radice)
```

Questa funzione chiede all'utente di inserire una data di inizio e una data di fine rilevazioni e verifica la validità delle stringhe inserite (controlla se sono stringhe di 10 cifre).

La funzione effettua una visita simmetrica all'albero e stampa i nodi il cui tempo cade nel range inserito. Si è scelta la visita simmetrica per avere la stampa dei nodi in ordine di tempo.

```
int cancellaDaAlbero(NodoLettura_t **radice)
```

Questa funzione chiede all'utente di inserire una data di inizio e una data di fine rilevazioni da cancellare, verifica la validità delle stringhe inserite (controlla se sono stringhe di 10 cifre). Si effettua una visita posticipata e si cancellano i nodi il cui valore di tempo cade nel range specificato. Si è scelta la visita posticipata affinché i figli vengano analizzati prima dei rispettivi padri.

Funzioni per la struttura array

```
int inserisci_lettura_array(Lettura_t *arrayLetture, int contaLetture, int *dimensione_array, char *tempo, float gas, float elettricità)
```

Questa funzione inserisce una nuova lettura nell'array passato come primo argomento. Riceve come argomenti un puntatore al primo elemento dell'array, il numero di letture già inserite nell'array (contaLetture), la dimensione dell'array (dimensione_array) e i dati della lettura da inserire.

Il nuovo elemento viene inserito alla fine dell'array. Se si supera la dimensione dell'array, l'array viene riallocato in modo da aumentarne la dimensione. La funzione non ordina l'array e non tiene conto di eventuali duplicati. Una volta inseriti tutte le letture, l'ordinamento e la cancellazione dei duplicati vengono fatti nel main.

```
int ricercaArray(Lettura_t *arrayLetture, int contaLetture)
```

Questa funzione chiede all'utente di inserire una data di inizio e una data di fine rilevazioni e verifica la validità delle stringhe inserite (controlla se sono stringhe di 10 cifre).

La funzione scorre l'array ordinato e stampa tutti gli elementi che cadono nel range specificato.

```
int cancellaDaArray(Lettura_t *arrayLetture, int contaLetture)
```

Questa funzione chiede all'utente di inserire una data di inizio e una data di fine rilevazioni da cancellare e verifica la validità delle stringhe inserite (controlla se sono stringhe di 10 cifre). Riceve come argomenti un puntatore al primo elemento dell'array e un intero contaLetture, che rappresenta il numero di elementi nell'array. Restituisce il numero di elementi nell'array dopo la cancellazione.

La funzione scorre l'array ordinato, contando gli elementi che cadono nel range specificato. Questi elementi vengono poi soprascritti e viene restituita la variabile contaLetture diminuita del numero di letture cancellate.

Implementazione dell'algoritmo

Implementazione dell'algoritmo:

Di seguito viene riportato il Makefile che prevede la compilazione del file sorgente principale secondo lo standard ANSI con segnalazioni di warning:

progetto: AnalisiRiscaldamento.c Makefile

```
gcc -ansi -Wall -O AnalisiRiscaldamento.c -o progetto
```

pulisci:

```
rm -f progetto.o
```

pulisci_tutto:

```
rm -f progetto progetto.o Risultati.txt
```

esegui:

```
./progetto
```

Di seguito viene riportato il file sorgente, AnalisiRiscaldamento.c, che costituisce l'intero programma.

```
1  *****/
2  /* Progetto di Algoritmi e Strutture Dati */
3  /* Sessione Estiva 2018/2019 */
4  /* Gestione di un sistema di riscaldamento */
5  /* Studente: Luca Grasso */
6  /* Matricola 294612 */
7  *****/
8
9  /* Inclusione delle librerie */
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13 #include <time.h>
14
15 /* Definizione delle costanti */
16 #define MAX_LENGTH 20      /* massima lunghezza della stringa che rappresenta il tempo */
17 #define INITIAL_SIZE 20    /* dimensione iniziale dell'array che contiene le letture */
18 #define MULTIPLY_FACTOR 2 /* fattore che definisce di quanto si incrementa l'array delle
19                         letture quando si raggiunge il numero massimo di elementi */
20
21 /* Definizione del tipo di dato che contiene il nodo dell'albero che rappresenta una singola lettura */
22 typedef struct NodoLettura{
23     char tempo[MAX_LENGTH];
24     float gas;
25     float elettricita;
26     struct NodoLettura *destro;
27     struct NodoLettura *sinistro;
28 } NodoLettura_t;
29
30 /* Definizione del tipo di dato struttura che rappresenta una lettura da inserire nell'array */
31 typedef struct Lettura {
32     char tempo[MAX_LENGTH];
33     float gas;
34     float elettricita;
35 } Lettura_t;
36
37 *****/
38 /* Dichiarazione delle funzioni implementate in questo file */
39 *****/
40 /* Sezione albero */
41 int inserisci_nodo_albero(NodoLettura_t **radice, char *tempo, float gas, float elettricita);
42 int ricercaAlbero(NodoLettura_t **radice);
43 int cancellaDaAlbero(NodoLettura_t **radice);
44 void cerca_simm(NodoLettura_t *nodo, char *inizio, char *fine);
45 void stampaAlbero(NodoLettura_t *nodo);
46
47 /*sezione array */
48 int inserisci_lettura_array(Lettura_t *arrayLetture, int size, int *dimensione_array, char *tempo, float gas, float elettricita);
49 int ricercaArray(Lettura_t *arrayLetture, int contaLettture);
50 int cancellaDaArray(Lettura_t *arrayLetture, int contaLettture);
51 int ordina_array(Lettura_t *arrayLetture, int inizio, int fine);
52 void fondi(Lettura_t a[], int sx, int mx, int dx);
53 void cancella_nodi(NodoLettura_t **radice, NodoLettura_t *nodo, NodoLettura_t *padre, char *inizio, char *fine);
54 int controllaData(char data[]);
55
56 /* sezione Test sperimentale */
57 int test(NodoLettura_t **radice);
58 int generaTempo(char *tempo);
```

```

59 /*****
60 /* Definizione della funzione main */
61 *****/
62 int main(void){
63     /* Dichiarazione delle variabili della funzione */
64
65     /* creazione della radice dell'albero */
66     NodoLettura_t **radice = (NodoLettura_t **) malloc(sizeof(NodoLettura_t *));
67     /* si chiede all'utente di inserire il nome del file */
68     printf("\nInserisci il nome del file di tipo nome.estensione (es. Lettura.txt):\n");
69
70     char *nomeFile = malloc(MAX_LENGTH*sizeof(char));    /* Input: Puntatore alla variabile di lettura nome file */
71     /* Leggo l'input del nome del file */
72     scanf("%s", nomeFile);
73
74     FILE *file;      /* Lavoro: variabile di tipo file */
75
76     /* apro il file */
77     file = fopen(nomeFile, "r");
78
79     /* se il file indicato dall'utente non è valido, il programma stampa un messaggio di errore e termina */
80     if(file == NULL){
81         printf("Non hai inserito un nome valido\n FINE DEL PROGRAMMA\n\n");
82         return 1;
83     }
84
85     /* variabili utilizzate per contenere le stringhe lette dal file */
86     char *nChar;           /* Lavoro: puntatore a char */
87     char buffer[100];      /* Lavoro: buffer della riga */
88
89     int contaLetture = 0;   /* Lavoro variabile che rappresenta il numero di letture inserite nell'array */
90
91
92     /* si legge il file una riga alla volta, si individuano all'interno della stringa tempo, gas ed elettricità
93      se nella struttura scelta non è già presente una lettura con il valore di tempo letto, la si inserisce
94      se invece la struttura dati scelta (albero o array) contiene già una lettura con lo stesso tempo,
95      la nuova stringa viene ignorata */
96     do{
97         nChar = fgets(buffer, 100, file);    /* leggo la riga */
98         if(nChar != NULL && nChar[0]>= '0' && nChar[0] <= '9'){    /* verifico che abbia solo numeri o se il riga sia vuota */
99
100            char tempo[20],           /* Input: variabile stringa contenente la data e ora */
101                s_gas[20],          /* Input: variabile stringa contenente la lettura del gas*/
102                s_elettricità[20]; /* Input: variabile stringa contenente la lettura dell'elettricità */
103
104            /* si copiano i primi 10 caratteri nella variabile tempo */
105            strncpy(tempo, buffer, 10); /* copio i primi 10 caratteri */
106            tempo[10] = 0;
107
108            /* si ricerca l'indice che corrisponde all'inizio del valore del gas tenendo conto dello spazio */
109            int i = 10;
110            while(buffer[i]== ' ')
111                i++;
112
113            /* si legge il valore del gas e lo si memorizza nella stringa s_gas tenendo conto dello spazio*/
114            int j=0;
115            while(buffer[i]!= ' '){
116                s_gas[j++] = buffer[i++];
117            }
118            s_gas[j] = 0;
119
120            /* si ricerca l'indice che corrisponde all'inizio del valore dell'elettricità tenendo conto dello spazio */
121            j = 0;
122            while(buffer[i]== ' ')
123                i++;
124
125        }
126    }
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172

```

```

173     /* si legge il valore dell'elettricità e lo si memorizza nella stringa s_elettricità */
174     while(buffer[i]!=' ' && buffer[i]!=0){
175         s_elettricità[j++] = buffer[i++];
176     }
177     s_elettricità[j] = 0;
178
179     /* conversione di s_gas e s_elettricità in float */
180     float gas = atof(s_gas);           /* variabile contenente il valore di gas in float */
181     float elettricità = atof(s_elettricità);    /* variabile contenente il valore di elettricità in float */
182
183     /* inserimento della nuova lettura nell'albero oppure nell'array */
184     if(strutturaDati==1){
185         inserisci_nodo_albero(radice, tempo, gas, elettricità); /* sottoprogramma di inserimento dati per l'albero */
186     } else {
187         /* sottoprogramma di inserimento dati per l'array */
188         inserisci_lettura_array(arrayLettura, contaLettura++, &dimensione_array, tempo, gas, elettricità);
189     }
190 }
191 } while(nChar != NULL); /* ripeto finché non trovo una riga nulla */
192
193 /* chiusura del file */
194 fclose(file);
195
196 /* Nel caso in cui è stato scelto la struttura dati array lo riordino */
197 if(strutturaDati == 2){
198     ordina_array(arrayLettura, 0, contaLettura-1); /* ordino l'array con l'algoritmo mergesort */
199
200     /* controllo duplicati */
201     int i; /* Lavoro: contatore del ciclo for */
202     for(i=0; i<contaLettura-1; i++){
203         if(strcmp(arrayLettura[i].tempo, arrayLettura[i+1].tempo)==0){
204             int j;
205             for(j=i+2; j < contaLettura; j++)
206                 arrayLettura[j-1] = arrayLettura[j];
207             contaLettura--;
208         }
209     }
210
211     /* stampo i valori dell'array ordinati*/
212     for(i = 0; i < contaLettura; i++){
213         printf("%s ", arrayLettura[i].tempo);
214         printf("%f ", arrayLettura[i].gas);
215         printf("%f\n", arrayLettura[i].elettricità);
216     }
217 }
218
219 /* si chiede all'utente quale operazione vuole fare
220 si ripete la richiesta finché l'utente non seleziona Q (uscita dal programma) */
221 while(1){
222
223     printf("\nSelezionare una opzione (R/D/Q):\n");
224     printf("*****\n");
225     printf("R. Ricerca\n");
226     printf("D. Cancellazione\n");
227     printf("Q. Esci dal Programma\n\n");
228
229     char stringaLetta[MAX_LENGTH];
230     scanf("%s", stringaLetta);

```

```

232
233     switch(stringaLetta[0]){
234         case 'r':
235             /* l'utente vuole fare una ricerca nell'albero o nell'array */
236             if(strutturaDati == 1)
237                 ricercaAlbero(radice);
238             else {
239                 ricercaArray(arrayLettura, contaLettura);
240             }
241             break;
242         case 'd':
243             /* l'utente vuole fare una cancellazione nell'albero o nell'array */
244             if(strutturaDati == 1)
245                 cancellaDaAlbero(radice);
246             else {
247                 contaLettura = cancellaDaArray(arrayLettura, contaLettura);
248             }
249             break;
250         case 'q':
251         case 'Q':      /* l'utente vuole uscire dal programma */
252             return 0;
253         default:        /* l'utente non ha selezione un'opzione valida e viene comunicato a video */
254             printf("Hai selezionato una opzione non valida, riprova\n");
255     }
256 }
257
258 }

260 ****
261 /* Sottoprogramma per inserisce un nuovo NodoLettura_t nell'albero, */
262 /* si è usato l'algoritmo di inserimento di un nodo in un albero      */
263 /* ordinato visto a lezione                                              */
264 ****
265 int inserisci_nodo_albero(NodoLettura_t **radice, char *tempo, float gas, float elettricita){
266     int inserito;
267     NodoLettura_t *nodo, *padre, *nuovo;
268
269     /* per confrontare due lettura, confrontiamo i rispettivi valori della variabile tempo con la funzione strcmp
270     strcmp prende come argomento due stringhe,
271     - restituisce 0 se le due stringhe sono uguali
272     - restituisce un numero positivo se la prima stringa segue la seconda in ordine lessicografico
273     - restituisce un numero negativo se la prima stringa precede la seconda in ordine lessicografico */
274     for (nodo = padre = *radice;
275         ((nodo != NULL) && (strcmp(tempo, nodo->tempo) != 0));
276         padre = nodo, nodo = strcmp(tempo, nodo->tempo) < 0 ? nodo->sinistro: nodo->destro);
277     if (nodo != NULL)
278         inserito = 0;
279     else {
280         inserito = 1;
281         nuovo = (NodoLettura_t *)malloc(sizeof(NodoLettura_t));
282         strncpy(nuovo->tempo, tempo, 15);
283         nuovo->gas = gas;
284         nuovo->elettricita = elettricita;
285         nuovo->sinistro = nuovo->destro = NULL;
286         if (nodo == *radice)
287             *radice = nuovo;
288         else
289             if (strcmp(tempo, padre->tempo) < 0)
290                 padre->sinistro = nuovo;
291             else
292                 padre->destro = nuovo;
293     }
294     return(inserito);
295 }

```

```

297 /*****
298 /* Sottoprogramma che chiede all'utente di inserire un intervallo */
299 /* di tempo e stampa tutte le letture nell'intervallo */
300 *****/
301 int ricercaAlbero(NodoLettura_t **radice){
302
303     char inizio[MAX_LENGTH];      /* Lavoro: variabile di inizio lettura dati */
304     char fine[MAX_LENGTH];       /* Lavoro: variabile di fine lettura dati */
305     char temp[MAX_LENGTH];       /* Input: variabile di appoggio dati in Input*/
306
307     int isValid = 0;             /* Lavoro: verifico se il dato è valido */
308
309     /* l'utente deve inserire il tempo nel formato aaaammddhh, esempio 2019042908 */
310     printf("Inserisci giorno, mese, anno, orario di inizio rilevazioni da ricercare nel formato aaaammddhh, per esempio 2019042908\n");
311
312     /* verifico se l'input è esclusivamente numerico a prescindere della validità posizionale */
313     do{
314         scanf("%s", temp);
315         isValid = controllaData(temp);
316         if(!isValid)
317             printf("Non hai inserito una sequenza di 10 cifre\n");
318     }while(!isValid);
319     strncpy(inizio, temp, 11);
320
321     /* l'utente deve inserire il tempo nel formato aaaammddhh, esempio 2019042908 */
322     printf("Inserisci giorno, mese, anno, orario di fine rilevazioni da ricercare nel formato aaaammddhh, per esempio 2019042910\n");
323
324     /* verifico se l'input è esclusivamente numerico a prescindere della validità posizionale */
325     do{
326         scanf("%s", temp);
327         isValid = controllaData(temp);
328         if(!isValid)
329             printf("Non hai inserito una sequenza di 10 cifre\n");
330     }while(!isValid);
331     strncpy(fine, temp, 11);
332
333     /* chiamata al sottoprogramma ricorsivo cerca_simm */
334     cerca_simm(*radice, inizio, fine);
335
336     return 0;
337 }
338 *****/
339 /* Sottoprogramma per l'inserimento dell'input per la */
340 /* cancellazione di un valore/i nell'albero */
341 *****/
342
343 int cancellaDaAlbero(NodoLettura_t **radice){
344
345     char inizio[MAX_LENGTH];      /* Lavoro: variabile con la data e ora di inizio */
346     char fine[MAX_LENGTH];       /* Lavoro: variabile con la data e ora di fine */
347     char temp[MAX_LENGTH];       /* Input: variabile che acquisisce l'input da utente */
348
349     int isValid = 0;             /* Lavoro: verifico se il dato è valido */
350
351     /* l'utente deve inserire il tempo nel formato aaaammddhh, esempio 2019042908 */
352     printf("Inserisci giorno, mese, anno, orario di inizio rilevazioni da cancellare nel formato aaaammgghh\n");
353
354     /* verifico se l'input è esclusivamente numerico a prescindere della validità posizionale */
355     do{
356         scanf("%s", temp);
357         isValid = controllaData(temp);
358         if(!isValid)
359             printf("Non hai inserito una sequenza di 10 cifre\n");
360     }while(!isValid);
361     strncpy(inizio, temp, 11);
362
363     /* l'utente deve inserire il tempo nel formato aaaammddhh, esempio 2019042908 */
364     printf("Inserisci giorno, mese, anno, orario di fine rilevazioni da cancellare\n");
365
366     /* verifico se l'input è esclusivamente numerico a prescindere della validità posizionale */
367     do{
368         scanf("%s", temp);
369         isValid = controllaData(temp);
370         if(!isValid)
371             printf("Non hai inserito una sequenza di 10 cifre\n");
372     }while(!isValid);
373     strncpy(fine, temp, 11);
374
375     /* Sottoprogramma per la cancellazione di dati nell'albero */
376     cancella_nodi(radice, *radice, *radice, inizio, fine);
377
378     return 0;
379 }
```

```

381 /*********************************************************************
382 /* Sottoprogramma che inserisce una lettura nell'array */
383 /* mediante un intervallo di tempo aaaammddhh */
384 /*********************************************************************
385 int inserisci_lettura_array(Lettura_t *arrayLetture, int contaLetture, int *dimensione_array, char *tempo, float gas, float elettricità){
386
387 if(contaLetture >= *dimensione_array){
388
389     /* ridimensiono l'array se contalettura è maggiore della dimensione_array */
390     arrayLetture = realloc(arrayLetture, MULTIPLY_FACTOR*(dimensione_array)*sizeof(Lettura_t));
391
392     /* aggiorno il valore della dimensione */
393     *dimensione_array = MULTIPLY_FACTOR*(dimensione_array);
394 }
395
396 /* creazione di un Lettura_t */
397 Lettura_t nuovo;      /* Lavoro: creo un nodo di tipo lettura_t */
398
399 strncpy(nuovo.tempo, tempo, MAX_LENGTH); /* assegno la data e ora */
400 nuovo.gas = gas;                      /* assegno il valore digas */
401 nuovo.elettricità = elettricità;       /* assegno il valore dell'elettricità */
402
403 /* la nuova lettura viene copiata alla fine dell'array */
404 arrayLetture[contaLetture] = nuovo;
405
406 return 0;
407 }

409 /*********************************************************************
410 /* Funzione per la ricerca in un array con inserimento */
411 /* di input da utente */
412 /*********************************************************************
413 int ricercaArray(Lettura_t *arrayLetture, int contaLetture){
414
415     char inizio[MAX_LENGTH];      /* Lavoro: variabile con la data e ora di inizio */
416     char fine[MAX_LENGTH];       /* Lavoro: variabile con la data e ora di fine */
417     char temp[MAX_LENGTH];        /* Input: variabile che acquisisce l'input da utente */
418
419     int isValid = 0;             /* Lavoro: verifico se il dato è valido */
420
421     /* si chiede all'utente di inserire giorno e ora di inizio lettura da ricercare nel formato aaaammddhh */
422     printf("Inserisci giorno, mese, anno, orario di inizio rilevazioni da ricercare\n");
423
424     /* controllo della validità dell'input se è interamente numerico */
425     do{
426         scanf("%s", temp);
427         isValid = controllaData(temp);
428         if(!isValid)
429             printf("Non hai inserito una sequenza di 10 cifre\n");
430     }while(!isValid);
431     strcpy(inizio, temp, 11);
432
433     /* si chiede all'utente di inserire giorno e ora di fine lettura da ricercare nel formato aaaammddhh */
434     printf("Inserisci giorno, mese, anno, orario di fine rilevazioni da ricercare\n");
435
436     /* controllo della validità dell'input se è interamente numerico */
437     do{
438         scanf("%s", temp);
439         isValid = controllaData(temp);
440         if(!isValid)
441             printf("Non hai inserito una sequenza di 10 cifre\n");
442     }while(!isValid);
443     strcpy(fine, temp, 11);
444
445     int contatore = 0;           /* Lavoro: indice per l'array dei dati inseriti */
446
447     /* cerco la prima lettura successiva alla data di inizio */
448     while(strcmp(inizio, (arrayLetture[contatore]).tempo) > 0 && contatore < contaLetture)
449         contatore++;
450
451     /* si stampa la lettura corrente finché la lettura corrente è precedente alla data di fine */
452     while(strcmp(fine, (arrayLetture[contatore]).tempo) > 0 && contatore < contaLetture) {
453         printf("%s ", arrayLetture[contatore].tempo);
454         printf("%f ", arrayLetture[contatore].gas);
455         printf("%f\n", arrayLetture[contatore].elettricità);
456         contatore++;
457     }
458
459 }
```

```

461 ****
462 /* Sottoprogramma per la cancellazione da un array */
463 /* con inserimento degli input */
464 ****
465 int cancellaDaArray(Lettura_t *arrayLettture, int contaLettture){
466
467     char inizio[MAX_LENGTH];      /* Lavoro: variabile con la data e ora di inizio */
468     char fine[MAX_LENGTH];       /* Lavoro: variabile con la data e ora di fine */
469     char temp[MAX_LENGTH];        /* Input: variabile che acquisisce l'input da utente */
470
471     int isValid = 0;             /* Lavoro: verifico se il dato è valido */
472
473     /* si chiede all'utente di inserire giorno e ora di inizio rilevazioni da cancellare nel formato aaaammddhh */
474     printf("Inserisci giorno, mese, anno, orario di inizio rilevazioni da cancellare\n");
475
476     /* controllo della validità dell'input se è interamente numerico */
477     do{
478         scanf("%s", temp);
479         isValid = controllaData(temp);
480         if(!isValid)
481             printf("Non hai inserito una sequenza di 10 cifre\n");
482     }while(!isValid);
483     strncpy(inizio, temp, 11);
484
485     /* si chiede all'utente di inserire giorno e ora di fine rilevazioni da cancellare nel formato aaaammddhh */
486     printf("Inserisci giorno, mese, anno, orario di fine rilevazioni da cancellare\n");
487
488     /* controllo della validità dell'input se è interamente numerico */
489     do{
490         scanf("%s", temp);
491         isValid = controllaData(temp);
492         if(!isValid)
493             printf("Non hai inserito una sequenza di 10 cifre\n");
494     }while(!isValid);
495     strncpy(fine, temp, 11);
496
497     /* contatore per le letture cancellate */
498     int lettureCancellate = 0; /* Lavoro: contatore per indice di array */
499
500     int contatore = 0;          /* Lavoro: contatore per indice di arra */
501
502     /* si cerca l'indice della prima lettura successiva al valore del tempo di inizio scelto dall'utente */
503     while(strcmp(inizio, (arrayLettture[contatore]).tempo) > 0 && contatore < contaLettture)
504         contatore++;
505
506     /* incremento lettureCancellate finché la lettura corrente ha il valore del tempo che precede il tempo di fine scelto dall'utente */
507     while(strcmp(fine, (arrayLettture[contatore]).tempo) > 0 && contatore < contaLettture) {
508         lettureCancellate++;
509         contatore++;
510     }
511
512     /* a questo punto contatore è l'indice del primo elemento che non va cancellato
513     copio tutte letture successive al contatore corrente nelle posizioni precedenti
514     in modo da cancellare esattamente lettureCancellate elementi prima del contatore */
515     int i;           /* Lavoro: indice per lo scorrimento dell'array */
516     for(i=contatore; i<contaLettture; i++){
517         arrayLettture[i - lettureCancellate] = arrayLettture[i];
518     }
519

```

```

520 /* restituisce il numero di letture presenti nell'array dopo la cancellazione */
521 return contaLetture - lettureCancellate;
522 }
523
524 ****
525 /* Sottoprogramma per la ricerca in un albero */
526 ****
527 void cerca_simm(NodoLettura_t *nodo, char *inizio, char *fine){
528     /* se il nodo è nullo cerco a destra */
529     if (nodo != NULL) {
530         /* chiamata ricorsiva per sinistra */
531         cerca_simm(nodo->sinistro, inizio, fine);
532         /* se trovo il nodo lo stampo */
533         if( strcmp(nodo->tempo, inizio) >= 0 && strcmp(nodo->tempo, fine) < 0){
534             printf("%s ", nodo->tempo);
535             printf("%f ", nodo->gas);
536             printf("%f\n", nodo->elettricità);
537         }
538         /* ricerca nel nodo destro */
539         cerca_simm(nodo->destro, inizio, fine);
540     }
541 }
542
543 ****
544 /* Sottoprogramma per l'ordinamento dell'array con Mergesort */
545 /* split dell'array */
546 ****
547 int ordina_array(Lettura_t *arrayLetture, int inizio, int fine){
548
549     int mx; /* Lavoro: variabile di appoggio per mergesort */
550
551     if (inizio < fine) {
552         mx = (inizio + fine) / 2;      /* calcolo indice per la metà */
553
554         /* ordino da inizio a mx */
555         ordina_array(arrayLetture,inizio,mx);
556
557         /* ordino da mx + 1 a fine */
558         ordina_array(arrayLetture,mx + 1,fine);
559
560         /* unisco i due array */
561         fondi(arrayLetture,inizio,mx,fine);
562     }
563
564     return 0;
565 }
566

```

```

567 /****** */
568 /* Sottoprogramma per l'ordinamento dell'array con Mergesort */
569 /* unione dei due array
570 /***** */
571 void fondi(Lettura_t a[], int sx, int mx, int dx){
572
573     Lettura_t *b; /* Lavoro: array di appoggio */
574
575     int i, /* Lavoro: indice per la parte sinistra di a (da sx ad m) */
576         j, /* Lavoro: indice per la parte destra di a (da m + 1 a dx) */
577         k; /* Lavoro: indice per la porzione di b da sx a dx */
578
579     /* fondi ordinatamente le due parti finche' sono entrambe non vuote */
580     b = (Lettura_t *)calloc(dx + 1,sizeof(Lettura_t));
581     for (i = sx, j = mx + 1, k = 0;((i <= mx) && (j <= dx));k++){
582         if (strcmp(a[i].tempo, a[j].tempo) <= 0){
583             b[k] = a[i];
584             i++;
585         } else {
586             b[k] = a[j];
587             j++;
588         }
589         while (i<=mx) {
590             b[k] = a[i];
591             i++;
592             k++;
593         }
594         while (j<=dx) {
595             b[k] = a[j];
596             j++;
597             k++;
598         }
599         for (k=sx; k<=dx; k++)
600             a[k] = b[k-sx];
601
602         /* Libero la memoria */
603         free(b);
604     }

```

```

606 /*****
607 /* Sottoprogramma per la cancellazione dei nodi in un albero */
608 *****/
609 void cancella_nodi(NodoLettura_t **radice, NodoLettura_t *nodo, NodoLettura_t *padre, char *inizio, char *fine){
610
611     if(nodo != NULL) {
612         cancella_nodi(radice, nodo->sinistro, nodo, inizio, fine);
613         cancella_nodi(radice, nodo->destro, nodo, inizio, fine);
614     }
615     if(nodo != NULL && strcmp(nodo->tempo, inizio) >= 0 && strcmp(nodo->tempo, fine) < 0) {
616         if (nodo->sinistro == NULL){
617             if (nodo == *radice)
618                 *radice = nodo->destro;
619             else
620                 if (strcmp(nodo->tempo, padre->tempo) < 0)
621                     padre->sinistro = nodo->destro;
622                 else
623                     padre->destro = nodo->destro;
624         } else
625             if (nodo->destro == NULL){
626                 if (nodo == *radice)
627                     *radice = nodo->sinistro;
628                 else
629                     if (strcmp(nodo->tempo, padre->tempo) < 0)
630                         padre->sinistro = nodo->sinistro;
631                     else
632                         padre->destro = nodo->sinistro;
633             }
634     } else {
635         NodoLettura_t *sost = nodo;
636         for (padre = sost, nodo = sost->sinistro;(nodo->destro != NULL);padre = nodo, nodo = nodo->destro);
637         strncpy(sost->tempo, nodo->tempo, MAX_LENGTH);
638         sost->gas = nodo->gas;
639         sost->elettricita = nodo->elettricita;
640         if (padre == sost)
641             padre->sinistro = nodo->sinistro;
642         else
643             padre->destro = nodo->sinistro;
644     }
645 }
646 }
647
648 *****/
649 /* Sottoprogramma per il controllo della data che sia numerico */
650 /* e della lunghezza giusta */
651 *****/
652 int controllaData(char data[]) {
653
654     if(strlen(data) != 10)
655         return 0;
656     int i; /* Lavoro: variabile per scorrere il ciclo for */
657     for(i=0; i<10; i++){
658         if(data[i] < '0' || data[i] > '9')
659             return 0;
660     }
661
662     return 1;
663 }
664

```

```

665 /*****
666 /* Sottoprogramma per il test sperimentale */
667 */
668 *****/
669 int test(NodoLettura_t **radice) {
670
671     clock_t start_time, end_time; /* variabili utilizzate per calcolare il tempo di esecuzione di ciascuna funzione */
672     FILE *fd_test; /* i risultati dell'analisi vengono riportati in un file di nome Risultati.txt */
673     fd_test=fopen("Risultati.txt", "w");
674     int N[] = {100, 1000, 10000}; /* l'analisi viene eseguita 3 volte: con 100, 1000, 10000 elementi nell'array e nell'albero*/
675     int i;
676     for(i=0; i<3; i++){
677         int contaLettture = 0; /* numero di elementi nell'array*/
678         Lettura_t *arrayLettture = malloc(N[i]*sizeof(Lettura_t)); /* creazione array di 100, 1000, 10000 */
679         int j;
680         for(j=0; j<N[i]; j++){
681             char *tempo = malloc(MAX_LENGTH*sizeof(char));
682             generaTempo(tempo); /* creazione di un valore casuale di tempo
683             Notiamo che i valori di gas ed elettricità non influiscono sul tempo di ricerca e cancellazione, quindi
684             prendiamo valori nulli per gas ed elettricità*/
685             inserisci_lettura_array(arrayLettture, contaLettture++, &N[i], tempo, 0, 0); /* inserimento di una lettura nell'array */
686             inserisci_nodo_albero(radice, tempo, 0, 0); /* inserimento di una lettura nell'albero */
687         }
688         ordina_array(arrayLettture, 0, contaLettture-1); /* ordinamento dell'array */
689
690         char inizio[MAX_LENGTH] = "2018010109"; /* definizione dei tempi di inizio e fine della ricerca e cancellazione da effettuare*/
691         char fine[MAX_LENGTH] = "2018030109";
692
693         /* CERCA ALBERO */
694         start_time = clock(); /* si registra l'istante di tempo in cui inizia l'operazione */
695         cerca_simm(*radice, inizio, fine);
696         end_time = clock(); /* si registra l'istante di tempo in cui è terminata l'operazione */
697         /* il numero di secondi impiegati è la differenza end_time - start_time divisa per la costante CLOCKS_PER_SEC (numero di clocks per secondo
698         eseguiti dal computer, è necessario fare il cast da (end_time - start_time) a float, altrimenti viene eseguita la divisione fra interi */
699         fprintf(fd_test, "Ricerca albero di %d elementi: %f secondi\n", N[i], (float)(end_time - start_time) / CLOCKS_PER_SEC);
700
701         /* CERCA ARRAY */
702         start_time = clock(); /* si registra l'istante di tempo in cui inizia l'operazione */
703         int contatore = 0;
704         /* cerco la prima lettura successiva alla data di inizio */
705         while(strcmp(inizio, (arrayLettture[contatore]).tempo) > 0 && contatore < contaLettture)
706             contatore++;
707
708         /* si stampa la lettura corrente finché la lettura corrente è precedente alla data di fine */
709         while(strcmp(fine, (arrayLettture[contatore]).tempo) > 0 && contatore < contaLettture) {
710             printf("%s ", arrayLettture[contatore].tempo);
711             printf("%f ", arrayLettture[contatore].gas);
712             printf("%f\n", arrayLettture[contatore].elettricità);
713             contatore++;
714         }
715         end_time = clock(); /* si registra l'istante di tempo in cui è terminata l'operazione */
716         fprintf(fd_test, "Ricerca array di %d elementi: %f secondi\n", N[i], (float)(end_time - start_time) / CLOCKS_PER_SEC);
717
718         /* CANCELLA ALBERO */
719         start_time = clock(); /* si registra l'istante di tempo in cui inizia l'operazione */
720         cancella_nodi(radice, *radice, *radice, inizio, fine);
721         end_time = clock(); /* si registra l'istante di tempo in cui è terminata l'operazione */
722         fprintf(fd_test, "Cancellazione albero di %d elementi: %f secondi\n", N[i], (float)(end_time - start_time) / CLOCKS_PER_SEC);
723
724         /* CANCELLA ARRAY */
725         start_time = clock(); /* si registra l'istante di tempo in cui inizia l'operazione */
726         contatore = 0;
727         int letturaCancellate = 0;
728
729         /* si cerca l'indice della prima lettura successiva al valore del tempo di inizio scelto dall'utente */
730         while(strcmp(inizio, (arrayLettture[contatore]).tempo) > 0 && contatore < contaLettture)
731             contatore++;
732
733         /* incremento letturaCancellate finché la lettura corrente ha il valore del tempo che precede il tempo di fine scelto dall'utente */
734         while(strcmp(fine, (arrayLettture[contatore]).tempo) > 0 && contatore < contaLettture) {
735             letturaCancellate++;
736             contatore++;
737         }
738         /* a questo punto contatore è l'indice del primo elemento che non va cancellato
739         copio tutte letture successive al contatore corrente nelle posizioni precedenti
740         in modo da cancellare esattamente letturaCancellate elementi prima del contatore */
741         int k;
742         for(k=contatore; k<contaLettture; k++){
743             arrayLettture[k - letturaCancellate] = arrayLettture[k];
744         }
745         end_time = clock(); /* si registra l'istante di tempo in cui è terminata l'operazione */
746         fprintf(fd_test, "Cancellazione array di %d elementi: %f secondi\n", N[i], (float)(end_time - start_time) / CLOCKS_PER_SEC);
747
748     }
749     fclose(fd_test);
750
751     printf("\nCreato il file Risultato.txt per la ricerca sperimentale\n\n");
752     printf("Il programma è terminato\n\n");
753
754     return 0;
755 }
756
757 }
```

```

759 /*****
760 /* Genera il tempo per il test sperimentale */
761 *****/
762 int generaTempo(char *tempo){
763
764     /* rand() restituisce un numero intero casuale compreso fra 0 e il massimo intero rappresentabile */
765     /* rand()%K genera un numero casuale fra 0 e K-1 */
766     int anno = rand()%3 + 2017; /* genera un anno tra 2017, 2018, 2019 */
767     int mese = rand()%12 + 1; /* genera un numero fra 1 e 12 (mese) */
768     int giorno = 0;
769     switch(mese){
770         case 11:
771         case 4:
772         case 6:
773         case 9:
774             giorno = rand()%30 + 1;
775             break;
776         case 2:
777             giorno = rand()%28 + 1;
778             break;
779         default:
780             giorno = rand()%31 + 1;
781     }
782     int ora = rand()%10 + 9; /* le rilevazioni avvengono fra le 9 e le 18 */
783     sprintf(tempo, "%4d%02d%02d%02d", anno, mese, giorno, ora);
784     return 0;
785 }
786
787
793 *****/
794 /* Sottoprogramma di stampa dell'albero */
795 *****/
796 void stampaAlbero(NodoLettura_t *nodo) {
797
798     if(nodo != NULL) {
799         stampaAlbero(nodo->sinistro);
800         printf("%s %f %f\n", nodo->tempo, nodo->gas, nodo->elettricita);
801         stampaAlbero(nodo->destro);
802     }
803 }
804
805

```

Testing del programma

Testing del programma:

Vengono di seguito riportate alcune prove sui punti più significativi del programma. A tale scopo è stato predisposto un file Lettura.txt con diverse letture per essere inserite nella struttura dati scelta dall'utente.

Sono stati eseguiti vari test sulla conformità degli input inseriti e la loro validità.

Test sulla compilazione del sorgente:

```
MacBook-Pro-di-Luca:ProgettoASD lucagrasso$ Make progetto
gcc -ansi -Wall -O AnalisiRiscaldamento.c -o progetto
MacBook-Pro-di-Luca:ProgettoASD lucagrasso$ █
```

Test su opzione errata inserita

```
Selezionare una opzione (T/A/Q):
*****
T. Alberi
A. Array
X. Analisi Sperimentale
Q. Esci dal Programma
*****
e
Non hai selezionato una opzione valida, riprova
```

Test di inserimento errato di un file di lettura

```
Selezionare una opzione (T/A/Q):
*****
T. Alberi
A. Array
X. Analisi Sperimentale
Q. Esci dal Programma
*****
T
Inserisci il nome del file di tipo nome.estensione (es. Lettura.txt):
Lettss
Non hai inserito un nome valido
FINE DEL PROGRAMMA
```

Inserimento dati in un albero e ricerca di un intervallo se la data di ricerca è errata (non è numerica e non rispetta 10 caratteri) abbiamo un messaggio di errore e di reinserimento.

Selezionare una opzione (T/A/Q):

T. Alberi
A. Array
X. Analisi Sperimentale
Q. Esci dal Programma

t

Inserisci il nome del file di tipo nome.estensione (es. Lettura.txt):
Lettura.txt

Selezionare una opzione (R/D/Q):

R. Ricerca
D. Cancellazione
Q. Esci dal Programma

r

Inserisci giorno, mese, anno, orario di inizio rilevazioni da ricercare nel formato aaaammddhh, per esempio 2019042908
123

Non hai inserito una sequenza di 10 cifre

2019042908

Inserisci giorno, mese, anno, orario di fine rilevazioni da ricercare nel formato aaaammddhh, per esempio 2019042910
sad

Non hai inserito una sequenza di 10 cifre

2019043008

2019042908 1.300000 2.700000

2019042909 0.300000 4.260000

2019042911 8.300000 5.140000

2019042912 4.300000 3.140000

2019042913 3.300000 5.199000

2019042914 8.300000 3.150000

2019042915 5.450000 9.330000

2019042916 9.400000 15.440000

2019042917 2.300000 99.739998

2019042918 2.300000 14.940000

Test di cancellazione dei dati in un albero, nel range 2019042911 - 2019042915.

Selezionare una opzione (R/D/Q):

R. Ricerca
D. Cancellazione
Q. Esci dal Programma

d

Inserisci giorno, mese, anno, orario di inizio rilevazioni da cancellare nel formato aaaammgghh

2019042911

Inserisci giorno, mese, anno, orario di fine rilevazioni da cancellare

2019042915

Selezionare una opzione (R/D/Q):

R. Ricerca
D. Cancellazione
Q. Esci dal Programma

r

Inserisci giorno, mese, anno, orario di inizio rilevazioni da ricercare nel formato aaaammddhh, per esempio 2019042908
2019042908

Inserisci giorno, mese, anno, orario di fine rilevazioni da ricercare nel formato aaaammddhh, per esempio 2019042910
2019043008

2019042908 1.300000 2.700000

2019042909 0.300000 4.260000

2019042915 5.450000 9.330000

2019042916 9.400000 15.440000

2019042917 2.300000 99.739998

2019042918 2.300000 14.940000

Test inserimento dei dati in un array, avviene la stampa dell'array ordinato alla fine del processo.

```
Selezionare una opzione (T/A/Q):
*****
T. Alberi
A. Array
X. Analisi Sperimentale
Q. Esci dal Programma
*****
a

Inserisci il nome del file di tipo nome.estensione (es. Lettura.txt):
Lettura.txt
0.000000 0.000000
2019042908 1.300000 2.700000
2019042909 0.300000 4.260000
2019042911 8.300000 5.140000
2019042912 4.300000 3.140000
2019042913 3.300000 5.199000
2019042914 8.300000 3.150000
2019042915 5.450000 9.330000
2019042916 9.400000 15.440000
2019042917 2.300000 99.739998
2019042918 2.300000 14.940000
2019043008 1.300000 2.700000
2019043009 0.300000 4.260000
2019043010 0.300000 4.220000
2019043011 8.300000 5.140000
2019043012 4.300000 3.140000
2019043013 3.300000 5.199000
2019043014 8.300000 3.150000
2019043015 5.450000 9.330000
2019043016 9.400000 0.000000
2019043018 2.300000 14.940000
```

Test di ricerca di un range all'interno dell'array.

```
Selezionare una opzione (R/D/Q):
*****
R. Ricerca
D. Cancellazione
Q. Esci dal Programma

r
Inserisci giorno, mese, anno, orario di inizio rilevazioni da ricercare
2019042908
Inserisci giorno, mese, anno, orario di fine rilevazioni da ricercare
2019
Non hai inserito una sequenza di 10 cifre
2019042915
2019042908 1.300000 2.700000
2019042909 0.300000 4.260000
2019042911 8.300000 5.140000
2019042912 4.300000 3.140000
2019042913 3.300000 5.199000
2019042914 8.300000 3.150000

Selezionare una opzione (R/D/Q):
*****
R. Ricerca
D. Cancellazione
Q. Esci dal Programma
```

Test di cancellazione in un array del range 2019042911 - 2019042915.

Selezionare una opzione (R/D/Q):

R. Ricerca

D. Cancellazione

Q. Esci dal Programma

d

Inserisci giorno, mese, anno, orario di inizio rilevazioni da cancellare
2019042911

Inserisci giorno, mese, anno, orario di fine rilevazioni da cancellare
2019042915

Selezionare una opzione (R/D/Q):

R. Ricerca

D. Cancellazione

Q. Esci dal Programma

r

Inserisci giorno, mese, anno, orario di inizio rilevazioni da ricercare
2019042908

Inserisci giorno, mese, anno, orario di fine rilevazioni da ricercare
2019043008

2019042908 1.300000 2.700000

2019042909 0.300000 4.260000

2019042915 5.450000 9.330000

2019042916 9.400000 15.440000

2019042917 2.300000 99.739998

2019042918 2.300000 14.940000

Come Test finale ho creato un file con i dati non in ordine temporale e con duplicati.

Le date sono in ordine e non contengono duplicati in un albero

Selezionare una opzione (T/A/Q):

T. Alberi

A. Array

X. Analisi Sperimentale

Q. Esci dal Programma

T

Inserisci il nome del file di tipo nome.estensione (es. Lettura.txt):

Lettura.txt

2019042908 1.300000 2.700000

2019042909 0.300000 4.260000

2019042911 8.300000 5.140000

2019042912 4.300000 3.140000

2019042913 3.300000 5.199000

2019042914 8.300000 3.150000

2019042915 5.450000 9.330000

2019042916 9.400000 15.440000

2019042917 2.300000 99.739998

2019042918 2.300000 14.940000

2019043008 1.300000 2.700000

2019043009 0.300000 4.260000

2019043010 0.300000 4.260000

2019043011 8.300000 5.140000

2019043012 4.300000 3.140000

2019043013 3.300000 5.199000

2019043014 8.300000 3.150000

2019043015 5.450000 9.330000

2019043016 9.400000 15.440000

2019043017 2.300000 99.739998

2019043018 2.300000 14.940000

Le date sono in ordine e non contengono duplicati in un array

```
Selezionare una opzione (T/A/Q):
*****
T. Alberi
A. Array
X. Analisi Sperimentale
Q. Esci dal Programma
*****
a

Inserisci il nome del file di tipo nome.estensione (es. Lettura.txt):
Lettura.txt
2019042908 1.300000 2.700000
2019042909 0.300000 4.260000
2019042911 8.300000 5.140000
2019042912 4.300000 3.140000
2019042913 3.300000 5.199000
2019042914 8.300000 3.150000
2019042915 5.450000 9.330000
2019042916 9.400000 15.440000
2019042917 2.300000 99.739998
2019042918 2.300000 0.000000
2019043008 1.300000 2.700000
2019043009 0.300000 4.260000
2019043010 0.300000 4.260000
2019043011 8.300000 5.140000
2019043012 4.300000 3.140000
2019043013 3.300000 5.199000
2019043014 8.300000 3.150000
2019043015 5.450000 9.330000
2019043016 9.400000 15.440000
2019043017 2.300000 99.739998
2019043018 2.300000 14.940000
```

Test della Ricerca Sperimentale e riporto il risultato al file Risultato.txt

```
2018022415 0.000000 0.000000
2018022415 0.000000 0.000000
2018022415 0.000000 0.000000
2018022417 0.000000 0.000000
2018022509 0.000000 0.000000
2018022511 0.000000 0.000000
2018022512 0.000000 0.000000
2018022512 0.000000 0.000000
2018022514 0.000000 0.000000
2018022515 0.000000 0.000000
2018022518 0.000000 0.000000
2018022609 0.000000 0.000000
2018022610 0.000000 0.000000
2018022610 0.000000 0.000000
2018022611 0.000000 0.000000
2018022611 0.000000 0.000000
2018022611 0.000000 0.000000
2018022613 0.000000 0.000000
2018022614 0.000000 0.000000
2018022615 0.000000 0.000000
2018022616 0.000000 0.000000
2018022618 0.000000 0.000000
2018022710 0.000000 0.000000
2018022711 0.000000 0.000000
2018022712 0.000000 0.000000
2018022712 0.000000 0.000000
2018022712 0.000000 0.000000
2018022714 0.000000 0.000000
2018022714 0.000000 0.000000
2018022718 0.000000 0.000000
2018022718 0.000000 0.000000
2018022718 0.000000 0.000000
2018022718 0.000000 0.000000
2018022809 0.000000 0.000000
2018022811 0.000000 0.000000
2018022815 0.000000 0.000000
2018022816 0.000000 0.000000
2018022817 0.000000 0.000000
2018022817 0.000000 0.000000
2018022818 0.000000 0.000000
2018022818 0.000000 0.000000
```

Creata il file Risultato.txt per la ricerca sperimentale

Il programma è terminato

```
MacBook-Pro-di-Luca:ProgettoASD lucagrasso$
```

Risultato all'interno del file Risultato.txt

Ricerca albero di 100 elementi: 0.000035 secondi

Ricerca array di 100 elementi: 0.000016 secondi

Cancellazione albero di 100 elementi: 0.000004 secondi

Cancellazione array di 100 elementi: 0.000003 secondi

Ricerca albero di 1000 elementi: 0.000188 secondi

Ricerca array di 1000 elementi: 0.000128 secondi

Cancellazione albero di 1000 elementi: 0.000038 secondi

Cancellazione array di 1000 elementi: 0.000006 secondi

Ricerca albero di 10000 elementi: 0.000804 secondi

Ricerca array di 10000 elementi: 0.001103 secondi

Cancellazione albero di 10000 elementi: 0.000262 secondi

Cancellazione array di 10000 elementi: 0.000027 secondi

*Valutazione della complessità delle
funzioni*

Inserimento di nodi nell'albero

La funzione di inserimento di un nodo nell'albero dapprima effettua una ricerca nell'albero per individuare la posizione in cui inserire il nuovo nodo e controlla se il nuovo nodo non è già presente nell'albero. Nel caso pessimo, la ricerca ci fa percorrere tutto il percorso dalla radice ad una foglia e, sempre nel caso pessimo, la lunghezza del cammino tra la radice e la foglia è pari alla dimensione dell'albero (ogni padre, tranne la foglia, ha esattamente un figlio). Nel caso pessimo il tempo di esecuzione è $O(n)$.

Nel caso migliore, il nuovo nodo viene inserito alla radice e si verifica quando l'albero è vuoto. Il tempo di esecuzione nel caso ottimo è $O(1)$.

Come si è visto nel corso, il tempo di esecuzione nel caso medio è $O(\log n)$.

Ricerca nell'albero

```
void cerca_simm(NodoLettura_t *nodo, char *inizio, char *fine){  
    if (nodo != NULL) {  
        cerca_simm(nodo->sinistro, inizio, fine);  
        if( strcmp(nodo->tempo, inizio) >= 0 && strcmp(nodo->tempo, fine) < 0){  
            printf("%s ", nodo->tempo);  
            printf("%f ", nodo->gas);  
            printf("%f\n", nodo->elettricità);  
        }  
        cerca_simm(nodo->destro, inizio, fine);  
    }  
}
```

La funzione `cerca_simm` è ricorsiva. Il tempo di esecuzione $T(n)$ è la somma di

- valutazione della condizione dell'if : $O(1)$
- esecuzione di `cerca_simm` sul sottoalbero sinistro : $T(k)$, dove k è la dimensione del sottoalbero sinistro
- valutazione della condizione dell'if : $O(1)$
- eventuale stampa dei valori di tempo, gas, elettricità del nodo corrente : $O(1)$
- esecuzione di `cerca_simm` sul sottoalbero destro : $T(n-k-1)$, dove k è la dimensione del sottoalbero sinistro e quindi $n-k-1$ è la dimensione del sottoalbero destro

Nel caso ottimo si ha che il nodo passato alla funzione è `NULL`, quindi il tempo di esecuzione è $O(1)$. In generale vale

$$T(0) = 1$$

$$T(n) = 1 + T(k) + T(n-k-1) + d$$

dove d è il costo delle operazioni fatte per elaborare ciascun nodo (controllo se il tempo è compreso nel range assegnato, stampa dei valori del nodo). Come visto nel corso, si ha che

$$T(n) = (d+2)n + 1.$$

Dunque, $T(n) = O(n)$.

Cancellazione dall'albero

```
void cancella_nodi(NodoLettura_t **radice, NodoLettura_t *nodo, NodoLettura_t *padre, char *inizio, char *fine) {
    if(nodo != NULL) {
        cancella_nodi(radice, nodo->sinistro, nodo, inizio, fine);
        cancella_nodi(radice, nodo->destro, nodo, inizio, fine);
    }
    if(nodo != NULL && strcmp(nodo->tempo, inizio) >= 0 && strcmp(nodo->tempo, fine) < 0) {
        if (nodo->sinistro == NULL){
            if (nodo == *radice)
                *radice = nodo->destro;
            else
                if (strcmp(nodo->tempo, padre->tempo) < 0)
                    padre->sinistro = nodo->destro;
                else
                    padre->destro = nodo->destro;
        } else
            if (nodo->destro == NULL){
                if (nodo == *radice)
                    *radice = nodo->sinistro;
                else
                    if (strcmp(nodo->tempo, padre->tempo) < 0)
                        padre->sinistro = nodo->sinistro;
                    else
                        padre->destro = nodo->sinistro;
            }
        else {
            NodoLettura_t *sost = nodo;
```

```

        for (padre = sost, nodo = sost->sinistro;(nodo->destro != NULL);padre =
nodo, nodo = nodo->destro);

        strncpy(sost->tempo, nodo->tempo, MAX_LENGTH);

        sost->gas = nodo->gas;

        sost->elettricità = nodo->elettricità;

        if (padre == sost)

            padre->sinistro = nodo->sinistro;

        else

            padre->destro = nodo->sinistro;

    }

}

}

```

La funzione cancella_nodi è ricorsiva. Il tempo di esecuzione è la somma di

- esecuzione sul sottoalbero sinistro: $T(k)$, dove k è il numero dei nodi del sottoalbero sinistro
- esecuzione sul sottoalbero destro: $T(n-k-1)$
- controllo se il nodo corrente ha il valore di tempo che cade nel range: $O(1)$
- nel caso il nodo corrente debba essere cancellato, il tempo di cancellazione è la somma di
 - controllo se il figlio sinistro è NULL: $O(1)$
 - controllo se il nodo da cancellare è la radice: $O(1)$
 - se il nodo da cancellare è la radice, riassegno la radice: $O(1)$
 - se invece il nodo da cancellare non è la radice, controllo se il figlio destro del nodo da cancellare ha il valore di tempo maggiore o minore del tempo del padre del nodo da cancellare: $O(1)$
 - in entrambi i casi, assegno il figlio destro al padre : $O(1)$
- se il figlio sinistro non è NULL, controllo se il figlio destro è NULL: $O(1)$
- controllo se il nodo da cancellare è la radice: $O(1)$
- se il nodo da cancellare è la radice, riassegno la radice: $O(1)$
- se invece il nodo da cancellare non è la radice, controllo se il figlio sinistro del nodo da cancellare ha il valore di tempo maggiore o minore del tempo del padre del nodo da cancellare: $O(1)$
- in entrambi i casi, assegno il figlio destro al padre : $O(1)$

- se il nodo da cancellare ha entrambi i figli non NULL, cerco con un ciclo for nel sottoalbero sinistro l'elemento più grande, il numero di iterazioni dipende dalla struttura dell'albero sinistro. Nel caso ottimo devo fare una sola iterazione per trovare il nodo che sostituisce il nodo corrente. Nel caso ottimo il numero di iterazioni è uguale alla dimensione k del sottoalbero sinistro.
- una volta trovato il nodo che sostituisce il nodo da cancellare, copio i valori del nodo trovato nel nodo da cancellare: $O(1)$
- controllo se l'elemento che sostituisce è il figlio sinistro del nodo da cancellare: $O(1)$
- riassegno il puntatore al figlio (sinistro o destro) del padre del nodo da cancellare: $O(1)$

Con un calcolo analogo a quello fatto per la ricerca si vede che $T(n) = O(n)$.

Inserimento nell'array

Poiché le letture vengono inserite sempre in coda, il tempo di inserimento è $O(1)$. Dopo aver inserito tutte le letture, viene ordinato l'array con mergesort. Bisogna quindi sommare al tempo di inserimento di tutte le letture un tempo $O(n \log n)$ necessario a riordinare l'array e un tempo $O(n)$ necessario a cancellare i duplicati.

Ricerca nell'array

```
int contatore = 0;  
  
// cerco la prima lettura successiva alla data di inizio  
  
while(strcmp(inizio, (arrayLetture[contatore]).tempo) > 0 && contatore < contaLetture)  
    contatore++;  
  
// si stampa la lettura corrente finché la lettura corrente è precedente alla data di fine  
  
while(strcmp(fine, (arrayLetture[contatore]).tempo) > 0 && contatore < contaLetture) {  
    printf("%s ", arrayLetture[contatore].tempo);  
    printf("%f ", arrayLetture[contatore].gas);  
    printf("%f\n", arrayLetture[contatore].elettricità);  
    contatore++;  
}
```

Il tempo di esecuzione su un array ordinato di n elementi è la somma di

- inizializzazione del contatore: $O(1)$
- ciclo while che scorre l'array per trovare la prima lettura che cade nel range, all'interno del ciclo while viene eseguito il controllo sul valore del tempo della lettura corrente, il controllo sul valore di contatore e viene incrementato il contatore. Il tempo di esecuzione è pari al numero di iterazioni per 3. Il numero di iterazioni è k , dove k è il numero di elementi nell'array che sono minori del tempo di inizio: $3k$
- ciclo while che scorre gli elementi da stampare. Ad ogni iterazione del ciclo viene eseguito il controllo sul valore del tempo della lettura corrente, il controllo sul valore di contatore, vengono stampati i valori di tempo, gas, elettricità del nodo corrente e viene incrementato il contatore. Il tempo di esecuzione è proporzionale al numero di elementi trovati, indichiamo con t il numero di elementi trovati e con c il costo di ciascuna iterazione.

Complessivamente, il tempo di esecuzione è $1 + 3k + ct$.

Nel caso ottimo, si trova un unico elemento da stampare all'inizio dell'array, il tempo di esecuzione nel caso ottimo è $O(1)$.

Nel caso pessimo, viene attraversato l'intero array (ciò corrisponde alla situazione in cui l'ultimo elemento dell'array cade nel range). Nel caso pessimo il tempo di esecuzione è O(n).

Cancellazione dall'array

```
int lettureCancellate = 0;  
int contatore = 0;  
  
// si cerca l'indice della prima lettura successiva al valore del tempo di inizio scelto dall'utente  
while(strcmp(inizio, (arrayLetture[contatore]).tempo) > 0 && contatore < contaLetture)  
    contatore++;  
  
// incremento lettureCancellate finché la lettura corrente ha il valore del tempo che precede il tempo di fine  
// scelto dall'utente  
  
while(strcmp(fine, (arrayLetture[contatore]).tempo) > 0 && contatore < contaLetture) {  
    lettureCancellate++;  
    contatore++;  
}  
  
// a questo punto contatore è l'indice del primo elemento che non va cancellato  
// copio tutte letture successive al contatore corrente nelle posizioni precedenti  
// in modo da cancellare esattamente lettureCancellate elementi prima del contatore  
for(int i=contatore; i<contaLetture; i++){  
    arrayLetture[i - lettureCancellate] = arrayLetture[i];  
}
```

Il tempo di esecuzione su un array ordinato di n elementi è la somma di

- inizializzazione del contatore: O(1)
- inizializzazione del contatore delle letture da cancellare: O(1)
- ciclo while che scorre l'array per trovare la prima lettura che cade nel range, all'interno del ciclo while viene eseguito il controllo sul valore del tempo della lettura corrente, il controllo sul valore di contatore e viene incrementato il contatore. Il tempo di esecuzione è pari al numero di iterazioni per 3. Il numero di iterazioni è k, dove k è il numero di elementi nell'array che sono minori del tempo di inizio: 3k
- ciclo while che scorre gli elementi da cancellare. Ad ogni iterazione del ciclo viene eseguito il controllo sul valore del tempo della lettura corrente, il controllo sul valore di contatore, viene incrementato il contatore e il contatore delle letture da cancellare. Il tempo di esecuzione è pari a 4 per il numero di letture da cancellare, indichiamo con t il numero di letture da cancellare.
- ciclo for che sovrascrive le letture da cancellare. Ad ogni iterazione del ciclo viene copiata una lettura in una posizione diversa dell'array, viene incrementato il contatore,

viene controllata la condizione di uscita. Il tempo di esecuzione è pari a 3 per il numero di iterazioni. Il numero di iterazioni è uguale al numero di elementi che seguono gli elementi da cancellare, cioè $n - k - t$

Complessivamente, il tempo di esecuzione è $1 + 1 + 3k + 4t + 3(n-k-t) = 2 + t + 3n$, cioè $O(n)$.

Nel caso ottimo, si trova che non c'è nessun elemento da cancellare, il tempo di esecuzione nel caso ottimo è $O(1)$.

Nel caso pessimo, ci sono elementi da cancellare. Nel caso pessimo il tempo di esecuzione è $O(n)$.

Analisi Sperimentale

La funzione "Analisi Sperimentale" selezionabile nel menu del programma esegue una analisi del tempo di esecuzione delle operazioni di ricerca e cancellazione su array e alberi aventi N=100, 1000, 10000 elementi.

I tempi ottenuti vengono scritti in un file Risultati.txt e sono mostrati nella tabella seguente.

	N=100	N=1000	N=10000
Ricerca su albero	0.000035 s	0.000200 s	0.0001354 s
Cancellazione su albero	0.000004 s	0.00034 s	0.000347 s
Ricerca su array	0.000015 s	0.000132 s	0.000732 s
Cancellazione su array	0.000002 s	0.000005 s	0.000041 s

Notiamo che i tempi di cancellazione sono molto piccoli.

Il tempo di ricerca dà valori leggermente più piccoli per l'albero rispetto all'array, tuttavia i tempi crescono con la stessa velocità, in accordo col fatto che il tempo di esecuzione teorico è $O(n)$ per entrambe le strutture.

Ricordiamo che l'array è ordinato, pertanto i tempi di ricerca e di cancellazione sono inferiori rispetto a quanto si avrebbe nel caso di un array non ordinato. Tuttavia, la creazione di un array ordinato ha un costo aggiuntivo dovuto all'ordinamento dell'array con mergesort.