1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

CORSO DI LAUREA IN
INFORMATICA APPLICATA
SCUOLA DI
SCIENZE TECNOLOGIE E FILOSOFIA DELL'INFORMAZIONE

RELAZIONE PROGETTO ALGORITMI E STRUTTURE DATI

Anno Accademico 2020/2021

Sessione d'esame Invernale

Studente: Cossi Simone

Nr. Matricola 290796

s.cossi2@campus.uniurb.it

Corso tenuto da: **Valerio Freschi**

1) Specifica del problema:

Si supponga di dover progettare un programma per la gestione dei dati delle CPU di un datacenter.
Il sistema di monitoraggio permette di rilevare, una volta all'ora per ogni CPU:

- La potenza richiesta in [W]
- La temperatura del chip in [°C]
- Il numero dei processi attivi
- La quantità di memoria RAM occupata in [MB]

L'intero log viene poi scritto (una volta al giorno) su un file in formato testo, secondo il seguente formato (si assumano campi separati da tabulazione o spazio):

- **Tempo:** un numero intero $\in [1, 24]$ che rappresenta l'ora del giorno a cui si riferisce il rilevamento.
- **CPU:** un codice alfanumerico che risulta dalla concatenazione di tre lettere e tre numeri che identificano la singola CPU.
- **Potenza:** un numero reale che rappresenta il consumo di potenza (medio) nell'ora di rilevamento
- **Temperatura:** un numero reale che rappresenta la temperatura (media) nell'ora di rilevamento
- **Processi:** un numero reale che rappresenta il numero (medio) di processi attivi nell'ora di rilevamento
- **Memoria:** un numero reale che rappresenta la quantità (media) di memoria RAM richiesta nell'ora di rilevamento

Esempio

Tempo	CPU	Potenza	Temperatura	Processi	Memoria
3	ABC020	8.36	32.70	23.7	10455.7
3	ABC176	9.24	22.43	3.70	12465.8
3	AAF184	10.11	32.14	11.20	20425.9
...	...	...	...	...	...
12	ABC020	4.60	30.43	20.1	12855.7
12	ABC176	12.24	24.55	3.70	265.8
12	AAF184	10.34	22.83	15.2	17523.8
...	...	...	...	...	...
21	ABC176	13.32	42.21	30.4	22585.5
...	...	...	...	...	...

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 i dati relativi alle rilevazioni di una data CPU nel giorno. Ad esempio: se l'utente chiede i dati relativi alla CPU ABC176, il programma deve restituire le informazioni contenute nelle righe corrispondenti:

Esempio

Tempo	CPU	Potenza	Temperatura	Processi	Memoria
3	ABC176	9.24	22.43	3.70	12465.8
12	ABC176	12.24	24.55	3.70	265.8
21	ABC176	13.32	42.21	30.4	22585.5
...	...	...	...	...	...

Il programma deve inoltre prevedere una modalità che implementa le stesse funzionalità utilizzando un array al posto di un albero.
Per quanto riguarda l'analisi teorica si deve fornire la complessità dell'algoritmo di ricerca per la versione basata su albero e per quella basata su array.
Oltre all'analisi teorica della complessità si deve effettuare uno studio sperimentale della stessa per le operazioni di ricerca.
Come suggerimento si può operare generando un numero *N* di rilevazioni casuali (dove *N* rappresenta il numero di CPU monitorate).
L'analisi sperimentale deve quindi valutare la complessità al variare di *N* e confrontare l'algoritmo di ricerca che lavora su albero con il corrispondente che lavora su array.

2) Analisi del problema:

Input:

L'input è composto da un file che contiene le rilevazioni delle prestazioni per ogni ora di *N* CPU differenti e da alcuni comandi impartiti dall'utente nella fase di generazione del file e nella fase di esecuzione del programma.

I comandi che l'utente può impartire sono:

- Inserire il numero di CPU
- Scegliere tra una struttura ad albero o un array
- Scegliere il nome della CPU di cui si vorranno vedere le rilevazioni
- Scegliere se andare a vedere o meno le rilevazioni di un'altra CPU dopo aver visualizzato quelle di quella precedentemente scelta

Output:

L'output è composto dai dati che il generatore andrà a scrivere su file e da delle stampe a video in fase di esecuzione del programma, tra cui:

- Avvisi di eventuali errori di incompatibilità tra le scelte possibili e quelle effettuate dall'utente
- Avvisi di eventuali errori avvenuti durante l'apertura del file contenente i dati
- Indicazioni su cosa e come l'utente deve inserire per interagire col programma
- Stampa di una lista contenente tutti i nomi delle CPU contenute nel file
- Stampa di tutte le informazioni riguardanti la CPU scelta
- Stampa del conteggio dei passi eseguiti dall'algoritmo per ottenere la stampa delle informazioni

Relazioni input-output:

Partendo dal generatore, quest'ultimo genera e sovrascrive (se già dovessero essere presenti) dati riguardanti un numero *N* di CPU scelte dall'utente su di un file.
Il programma invece, utilizzando il file, mostrerà all'utente una lista contenente tutti i nomi delle CPU di cui sono state effettuate le rilevazioni, per poi andare a mostrare le informazioni riguardanti una o più CPU scelte.
Per fare ciò utilizza una struttura ad albero o un array.

L'utente è dunque responsabile di scegliere:

- Il numero di CPU di cui fare le rilevazioni e dunque anche il numero di CPU che verranno stampate nella lista al momento di scegliere
- Se andare a utilizzare una struttura ad albero o un array per la memorizzazione delle informazioni prese da file e per la ricerca di quest'ultime.
- Se vedere o meno le rilevazioni di altre CPU

3) Progettazione dell'algoritmo:

Strutture dati:

Le strutture utilizzate sono quelle previste dalla specifica del progetto:

- **Albero**: le informazioni vengono memorizzate in una struttura ad albero binario di ricerca, quest'ultimo prevede che ogni nodo abbia al massimo due figli e che data una chiave *c* appartenente ad un nodo, ad ogni nodo del suo sottoalbero destro appartiene una chiave di valore ≥ *c*, mentre ad ogni nodo del suo sottoalbero sinistro appartiene una chiave di valore < *c*.
  La struttura ad albero utilizzata è ordinata in base ai nomi delle CPU, è consentito, anche se molto improbabile, l'inserimento di una CPU con lo stesso nome di un'altra, quando ciò avviene quella rilevata successivamente viene inserita come figlio destro.

- **Array:** questa struttura è un insieme di elementi disposti linearmente, questo tipo di struttura viene utilizzato sia nel programma principale come richiesto dalla specifica di progetto sia nel programma che genera i dati nel file .txt.

Scelte di progetto:

La generazione dei nomi e dei dati di *N* CPU scelte dall'utente è stata affidata ad un programma separato dal resto. Questa scelta è stata effettuata per evitare conflitti tra dati e semplificare il lavoro in fase di sviluppo oltre al fatto di ritrovarsi con un generatore di dati su file separato.

Il resto del progetto è invece contenuto in un altro programma che utilizza lo stesso file del programma descritto precedentemente e utilizza i dati contenuti per delle ricerche.
Ricerche che vengono effettuate o con una struttura ad albero binario di ricerca (scelta per le sue buone prestazioni in fase di inserimento e ricerca) o con un array come previsto dalle specifiche di progetto.
Il programma chiede all'utente di scegliere il tipo di struttura da utilizzare per effettuare le ricerche all'interno del file, la quale non potrà essere cambiata durante l'esecuzione del programma.
Successivamente viene stampata una lista contenente tutti i nomi delle CPU contenuti nel file.
L'utente procede con la scelta di una delle CPU e il programma restituirà tutti i dati riguardanti la CPU scelta.
Inoltre, è stata inserita una funzione che permette all'utente di cercare altre CPU dopo averne già cercate alcune. Questa funzione è stata implementata poiché nonostante ci siano delle validazioni strette sugli input l'utente potrebbe inserire un nome di CPU accettabile ma non presente all'interno del file, in questa situazione l'utente si ritroverebbe senza alcun dato mentre, inserendo questa funzione si da la possibilità di cercare altre CPU senza dover eseguire dall'inizio il programma.

Ad ogni input inserito da tastiera viene effettuata una validazione stretta.

Algoritmi:

Inserimento: l'algoritmo di inserimento viene utilizzato per l'inserimento dei dati, contenuti nel file, all'interno dell'albero binario di ricerca. Ogni volta che è necessario inserire un nuovo elemento bisogna scorrere tutto l'albero fino ad arrivare ad un nodo foglia. Durante lo scorrimento dell'albero avvengono anche dei confronti tra i dati già inseriti e quello da inserire, poiché se il nuovo elemento ha valore superiore di quello già inserito lo scorrimento sarà vero il sottoalbero destro, viceversa con il sinistro.
Le comparazioni avvengono tramite due funzioni: una verifica se i valori sono uguali mentre l'altra verifica quale dei due è minore.
Ovviamente se l'albero dovesse essere vuoto il primo inserimento corrisponderà alla radice.

Ricerca: per la ricerca basta partire dalla radice e proseguire verso destra o sinistra in base al risultato del confronto tra il valore scelto e quello della chiave contenuta nel nodo.
Se il valore scelto è maggiore di quello della chiave del nodo si procederà verso destra altrimenti verso sinistra.

4) Implementazione dell'algoritmo

GeneratoreFileASD.c

Inizialmente viene cambiato il seme della funzione rand() in modo tale da cambiare ogni volta la generazione randomica dei nomi e dei dati.

Viene richiesto all'utente quante CPU si vogliono analizzare.
Viene allocata la memoria necessaria per l'acquisizione dei nomi delle CPU che vengono successivamente calcolati randomicamente.

Viene generato/aperto il file in modalità scrittura su cui andranno inseriti tutti i dati.

Vengono generati randomicamente all'interno di un intervallo tutti i dati mancanti delle CPU e vengono così scritte tutte le informazioni su file, alla fine della scrittura si procede con la chiusura del file.

```c
int main (void)
{
    // dichiarazione variabili locali
    int numero_cpu;     // numero di cpu

    char nome_cpu[7];   // valore alfanumerico che caratter

    array *lista_cpu;   // struttura che contine tutti i no

    FILE *dati = NULL;  // puntatore al file

    srand(time(NULL));  // cambio il seme del rand ogni vol

    // acquisizione del numero di cpu da voler utilizzare
    numero_cpu = ValidazioneCpu();

    // allocazione della memoria in base al numero di cpu
    lista_cpu = (array *)malloc(numero_cpu*sizeof(array));

    // calcolo dei nomi per le varie cpu
    GeneraNomi(numero_cpu, nome_cpu, lista_cpu);

    // apertura del file per la scrittura e verifica della
    dati = fopen("dati.txt", "w");
    if (dati == NULL)
        printf("Errore nell'apertura del file 'dati'\nContr

    // se il file viene aperto correttamente si procede con
    else
        GeneraDati (numero_cpu, dati, lista_cpu);

    // chiusura del file
    fclose(dati);

    printf("END");

    return 0;
}// END 'main'
```

Funzione che chiede all'utente il numero di CPU, acquisisce la risposta e la verifica.
Nel caso l'input inserito dall'utente non dovesse essere compatibile con il tipo richiesto verrà comunicato e effettuata nuovamente la richiesta dell'input.

```c
// funzione 'ValidazioneCpu'
int ValidazioneCpu()
{
    // dichiarazione variabili locali
    int num,
        temp;

    printf("Inserire numero di cpu da analizzare: \n");
    // aquisizione e verifica valori
    do
    {
        temp = scanf("%d", &num);
        if((num <= 0) || (temp == 0))
            printf("\nIl valore inserito non e' accetta
        while (getchar() != '\n');
    }while((num <= 0) || (temp == 0));

    // restituisco il primo valore valido
    return num;
}// END 'ValidazioneCpu'
```

Funzione che genera randomicamente tutti i nomi delle CPU,
nel formato richiesto dalla specifica.
I nomi generati saranno formati da tre lettere maiuscole e tre numeri.

```c
// funzione 'GeneraNomi'
void GeneraNomi(int numero_cpu, char nome_cpu[7], array *lista_cpu)
{
    for( int i = 0; i < numero_cpu ; i++ )
    {
        for( int j = 0; j < 6; j++)
        {
            // per fare ciò bisogna consultare la tabella ascii
            // per i primi 3 valori vengono inserite delle lettere m
            if( j < 3)
                nome_cpu[j] = 65 + rand() % (90-65);
            // per gli ultimi 3 valori vengono inseriti dei numeri t
            else if(j == 6)
                nome_cpu[j] = 0;
            else
                nome_cpu[j] = 48 + rand() % (57-48);
        }

        // copio il valore ottenuto nella struttura
        strcpy(lista_cpu[i].nome_cpu, nome_cpu);
    }
}// END 'GeneraNomi'
```

Funzione che genera i valori randomici e scrive su file tutti i dati in ordine:
ora, nome, potenza, temperatura, processi e memoria.
Per il calcolo randomico dei valori utilizza un'altra funzione.

```c
void GeneraDati(int numero_cpu, FILE *dati, array *lista_cpu)
{
    // dichiarazione variabili locali
    int     ora;         // numero intero tra 0 e 24 -- andrà ad indicare l'orario delle r
    double  potenza,     // valore medio di consumo di potenza in 'W'
            temperatura, // valore medio della temperatura in '°C'
            processi,    // valore medio di processi attivi
            memoria;     // quantità media di RAM richiesta
    // calcolo dei dati per ogni ora e inserimento nel file
    for ( ora = 1; ora <= 24; ora++)
    {
        // calcolo dati casuali all'interno di un range dato
        for ( int i = 0; i < numero_cpu; i++)
        {
            potenza     = ValoreRandom (5.00, 15.00);
            temperatura = ValoreRandom (30.00, 90.00);
            processi    = ValoreRandom (1.00, 40.00);
            memoria     = ValoreRandom (10.00, 2000.00);

            // inserimento dei dati nel file
            fprintf(dati, "%d\t%s\t\t%.2f\t\t%.2f\t\t%.2f\t\t%.2f\n",
                    ora, lista_cpu[i].nome_cpu, potenza, temperatura, processi, memoria);
        }
    }
}// END 'GeneraDati'
```

Funzione che contiene l'espressione per calcolare i valori randomici delle CPU utilizzando un intervallo.

```c
double ValoreRandom (double min, double max)
{
    // dichiarazione variabili locali
    double ritorno, // variabile di ritorno
           i = rand();

    // espressione per ottenere un numero ra
    ritorno = (i/RAND_MAX)*(max-min) + min;

    return ritorno;
}// END 'ValoreRandom'
```

ProgettoASD.c

Acquisizione della scelta tra albero o array da parte dell'utente con validazione stretta.

```
do
{
    esito = scanf("%d", &opzione);
    if((esito == 0) || ((opzione != 1) && (opzione != 2)))
        printf("Scelta non consentita\nPerfavore scegliere tra '1'  e  '2'\n");
    while (getchar() != '\n');
} while ((esito == 0) || ((opzione != 1) && (opzione != 2)));
```

Nel caso l'utente dovesse scegliere di eseguire il programma utilizzando la struttura di tipo albero si procede con l'apertura del file in lettura e relativa verifica di corretta apertura.
Nel caso il file sia stato aperto correttamente si procede con l'acquisizione e l'inserimento di tutti i dati nell'albero.
Stampa di tutti i nomi delle CPU.
Scelta della CPU e stampa di tutte le informazioni relative alla CPU scelta con relative verifiche riguardanti il nome.
Si conclude con la richiesta di voler effettuare un'altra ricerca o meno.

```
case 1: // situazione in cui è stato scelto l'albero

    printf("E' stato scelto l'albero\n");

    // dichiarazione variabili utilizzate solo in questo case
    albero *radice = NULL, // nodo radice
           *nodo   = NULL; // nodo per l'inserimento

    controllo = ApriFile(&dati);   // apro il file in lettura

    if(controllo == 1)   // caso in cui sia stato aperto corretamente il file
    {
        do  // aquisizione dei dati dal file
        {
            // creazione di un nuovo nodo
            nodo = (albero*)malloc(sizeof(albero));
            nodo -> dx = nodo -> sx = NULL;

            // lettura dei valori del nodo successivo
            esito = fscanf(dati, "%d  %s  %lf  %lf  %lf  %lf",
                           &(nodo -> ora),          (nodo -> nome_cpu), &(nodo -> potenza),
                           &(nodo -> temperatura), &(nodo -> processi), &(nodo -> memoria));
            if(esito == 6 )
                if(!Inserimento(&radice, nodo))
                    printf("\nInserimento non riuscito");
        } while(esito != EOF);

        // finita l'aquisizione dei dati si procede con la chiusura del file
        fclose(dati);

        // visualizzazione dei nomi delle cpu
        printf("\nLista dei nomi delle cpu che sono state analizzate\n");

        StampaNomiAlbero(radice);       // stampa i nomi delle cpu contenuti nel file

        do
        {
            passi = 0;

            VerificaNome(nome_cpu);     // acquisizione del nome della cpu scelto

            // ricerca dei dati relativi alla cpu scelta
            if (!CercaCpuAalbero(radice, nome_cpu, &passi))
                printf("\nNon e' stata trovata alcuna CPU, (attenzione inserire lettere maiu
            printf("\nI passi compiuti dall'algoritmo per trovare e stampare i dati della CP

            scelta = AltraRicerca();

        } while (scelta == 1);
    }
    else    // caso in cui non sia stato possibile aprire il file
        printf("\nNon e' stato possibile aprire corretamente il file...\nPerfavore controllar

    break; // END 'case 1'
```

Funzione che verifica la corretta apertura del file.
Se il file non dovesse venir aperto correttamente ritornerà un valore che verrà utilizzato nel main per avvertire l'utente di un problema con il file.

```
int ApriFile(FILE **file)
{
    // apro il file in lettura
    *file = fopen("dati.txt", "r");
    // verifico la corretta apertura del file
    if(*file == NULL)
        return 0;
    else
        return 1;
}// END 'ApriFile'
```

Funzione che inserisce i valori letti da file all'interno dell'albero scorrendolo e andando a decidere la posizione dei dati da inserire.
Per decidere la posizione dei dati da inserire si avvale di due funzioni che controllano se il nome della CPU sia minore o maggiore di quello già presente.
Lo scorrimento avviene come già detto verso destra se dovesse essere maggiore altrimenti verso sinistra.

```
int Inserimento(albero **radice, albero *nodo)
{
    // dichiarazione variabili locali
    int     esito;          // variabile di ritorno
    albero  *attuale,       // nodo attuale
            *padre;         // nodo padre

    // scorrimento dell'albero per il posizionamento della foglia
    for(attuale = padre = *radice;
        ((attuale != NULL) && (!NodiUguali(attuale, nodo)));
        padre = attuale, attuale = NodoMinore(nodo, attuale) ? attuale -> sx : attuale -> dx);

    // verifica che non ci siano ripetizioni
    if(attuale != NULL)
        esito = 0;
    else
    {
        esito = 1;

        if(attuale == *radice)  // nel caso l'albero sia vuoto inserimento dell'elemento come r
            *radice = nodo;
        else                    // altrimenti l'inserimento avviene a sinistra se l'elemento è
        {
            if(NodoMinore(nodo, padre))
                padre -> sx = nodo;
            else
                padre -> dx = nodo;
        }
    }
    return esito;
}// END 'Inserimento
```

Funzioni che mi permettono di confrontare due nodi e comunicare quale sia il maggiore o il minore e se dovessero essere uguali.

```
// funzione 'NodiUguali'
int NodiUguali(albero *n1, albero *n2)
{
    // dichiarazione variabili locali
    int esito = 0;  // valore di ritorno

    // controllo se i valori tra i nodi sono uguali
    if((stricmp(n1 -> nome_cpu, n2 -> nome_cpu) == 0) && (n1 -> ora == n2 -> ora))
        esito = 1;
    return esito;
}// END 'NodiUguali'


// funzione 'NodoMinore'
int NodoMinore(albero *n1, albero *n2)
{
    //dichiarazione variabili locali
    int esito = 0;  // variabile di ritorno

    if(strcmp(n1 -> nome_cpu, n2 -> nome_cpu) < 0)
        esito = 1;
    return esito;
}// END 'NodoMinore'
```

Funzione che Stampa a video tutti i nomi delle CPU utilizzando la struttura ad albero.
Sviluppata in modo da non ripetere nomi già presenti ed è ricorsiva per andare ad analizzare tutti i sottoalberi.

```
// funzione 'StampaNomiAlbero'
void StampaNomiAlbero(albero *nodo)
{
    if(nodo != NULL)
    {
        // analizzo l'albero partendo dal sotto
        StampaNomiAlbero(nodo -> sx);
        if(nodo -> ora == 1)
            printf("\t%s\n", nodo -> nome_cpu);
        StampaNomiAlbero(nodo -> dx);
    }
}// END 'StampaNomiAlbero'
```

Funzione che chiede all'utente di inserire il nome di una CPU di cui si vogliono vedere le informazioni.
Esegue anche tutti i controlli possibili sul nome.

```c
// funzione 'VerificaNome'
void VerificaNome(char nome_cpu[7])
{
    // dichiarazione variabili locali
    int esito,
        lunghezza_nome;     // variabile utilizzata p

    // scelta della cpu da parte dell'utente
    printf("\nInserire il nome di una cpu sopra-visua
    do
    {
        esito = (scanf("\n%s", nome_cpu));
        lunghezza_nome = strlen(nome_cpu);

        if((esito != 1) || (lunghezza_nome != 6))
            printf("\nErrore nell'aquisizione del nom
        while (getchar() != '\n');
    } while ((esito != 1) || (lunghezza_nome != 6));
}// END 'VerificaNome'
```

Funzione ricorsiva che serve per scorrere l'albero finché non viene trovata la CPU richiesta, trovata la CPU ne stampa tutti i valori.

Successivamente richiama se stessa e va ad analizzare il sottoalbero destro poiché per come è stato sviluppato l'albero deve trovarsi per forza lì.

Ha un valore di ritorno perché nel caso non dovesse essere stato inserito un nome presente nella lista lo comunicherà all'utente.

```c
// funzione 'CercaCpuAlbero'
int CercaCpuAlbero(albero *radice, char nome_cpu[6], int *passi)
{
    // dichiarazione variabili locali
    albero  *attuale;       // nodo attuale
    int     esito = 0;      // variabile di ritorno

    // scorrimento dell'albero per trovare il nodo passato alla funzione
    for (attuale = radice;
        (attuale != NULL) && (strcmp(attuale -> nome_cpu, nome_cpu) != 0);
        attuale = (strcmp(nome_cpu, attuale -> nome_cpu) < 0) ? attuale -> sx: attuale -> dx)
    *passi += 1;

    // caso in cui abbia trovato una cpu con lo stesso nome
    if(attuale != NULL)
    {
        esito = 1;
        // stampa del nodo
        printf("%d\t\t%s\t\t%.2f\t\t%.2f\t\t%.2f\t\t%.2f\n",
                attuale -> ora,             attuale -> nome_cpu,    attuale -> potenza,
                attuale -> temperatura,     attuale -> processi,    attuale -> memoria);
        *passi += 1;

        // per trovare il nodo relativo all'ora successiva si procede con la ricerca nel sott
        CercaCpuAlbero(attuale -> dx, nome_cpu, passi);
    }
    return esito;
}// END 'CercaCpuAlbero
```

Funzione che acquisisce una scelta tra 2 possibilità con verifica dell'input e comunica la scelta.

```c
// funzione 'AltraRicerca'
int AltraRicerca()
{
    // dichiarazione variabili locali
    int num,
        temp;
    printf("Si disidera cercare un'altra CPU?\nPremere
    do
    {
        temp = scanf("%d", &num);
        if(((num <= 0) || (num > 2)) || (temp == 0))
            printf("\nRisposta non accettabile...\nPerf
        while (getchar() != '\n');
    }while(((num != 1) && (num != 2)) || (temp == 0));

    // restituisco il primo valore valido
    return num;
}// END 'AltraRicerca'
```

Nel caso in cui l'utente scegli di eseguire le ricerche utilizzando l'array come nell'altro caso si procede con l'apertura del file e con il controllare la corretta apertura.

Nel caso in cui venga aperto correttamente si procede con il conteggio delle righe del file per poi andare ad allocare la memoria necessaria.

Viene poi chiuso e riaperto il file per ritornare all'inizio del file, qui non viene fatto un controllo aggiuntivo poiché non dovrebbe essere necessario data la corretta apertura precedente.

Una volta riaperto si procede con l'acquisizione delle informazioni. A fine acquisizione si chiude il file.

Si esegue la stampa dei nomi delle CPU e con l'acquisizione del nome della CPU da analizzare con tutte le verifiche come nel caso precedente.

Si esegue la ricerca della CPU scelta e la stampa delle informazioni riguardanti quest'ultima.

Si conclude come nell'altro caso chiedendo all'utente se vuole effettuare un'altra ricerca.

```c
case 2: // situazione in cui è stato scelto l'array

    printf("E' stato scelto l'array\n");

    // dichiarazione varibili utilizzate solamente in questo case
    int     ora,          // numero intero tra 0 e 24 -- andrà ad indica
            righe,        // contatore delle righe del file
            c = 0;        // contatore utilizzato per l'inserimento
    double  potenza,      // valore medio di consumo di potenza in 'W'
            temperatura,  // valore medio della temperatura in '°C'
            processi,     // valore medio di processi attivi
            memoria;      // quantità media di RAM richiesta

    Array *array;       // creazione di un nuovo array
    controllo = ApriFile(&dati);    // apertura del file

    if (controllo == 1)
    {
        // conteggio del numero di righe del file aperto
        do
        {
            esito = fscanf(dati, "%d  %s  %lf  %lf  %lf  %lf",
                           &ora,          nome_cpu,   &potenza,
                           &temperatura,  &processi,  &memoria);
            righe += 1;
        } while (esito != EOF);

        array = (Array*)malloc(righe*sizeof(Array));

        // chiudo e riapro il file per poterlo leggere di nuovo dall'iniz
        fclose(dati);
        ApriFile(&dati);

        // lettura e inserimento dei dati
        do
        {
            esito = fscanf(dati, "%d  %s  %lf  %lf  %lf  %lf",
                           &ora,          nome_cpu,   &potenza,
                           &temperatura,  &processi,  &memoria);
            array[c].ora         = ora;
            array[c].potenza     = potenza;
            array[c].temperatura = temperatura;
            array[c].processi    = processi;
            array[c].memoria     = memoria;
            strcpy(array[c].nome_cpu, nome_cpu);
            c += 1;
        } while (esito != EOF);

        fclose(dati);   // chiusura del file
        printf("\nLista dei nomi delle cpu che sono state analizzate\n");

        StampaNomiArray(array, righe);      // stampa i nomi delle cpu co

        do  // acquisizione dei dati dal file
        {
            passi = 0;

            VerificaNome(nome_cpu);     // acquisizione del nome della cp

            // ricerca dei dati relativi alla cpu scelta
            if (!CercaCpuArray(array, righe, nome_cpu, &passi))
                printf("\nNon e' stata trovata alcuna CPU, (attenzione in
            printf("\nI passi compiuti dall'algoritmo per trovare e stamp

            scelta = AltraRicerca();

        } while (scelta == 1);
    }
    else    // caso in cui non sia stato possibile aprire il file
        printf("\nNon e' stato possibile aprire correttamente il file...\n

    break;  // END 'case 2'
}
```

Funzione che scorre l'array e ne stampa e nomi senza ripeterli.

```c
// funzione 'StampaNomiArray'
void StampaNomiArray(Array *array, int righe)
{
    for(int i = 0; i < righe; i++)
    {
        if(array[i].ora == 1)
            printf("\t%s\n", array[i].nome_cpu);
    }
}// END 'StampaNomiArray'
```

Funzione che cerca e stampa tutte le informazioni riguardanti la CPU scelta.

```c
// funzione 'CercaCpuArray
int CercaCpuArray(Array* array, int righe, char *nome_cpu, int *passi)
{
    // dichiarazione variabili locali
    int esito = 0;      // variabile di ritorno

    for (int i = 0; i < righe; i++)
    {
        if (strcmp(nome_cpu, array[i].nome_cpu) == 0)
        {
            esito = 1;
            printf("%d\t\t%s\t\t%.2f\t\t%.2f\t\t%.2f\t\t%.2f\n",
                    array[i].ora,           array[i].nome_cpu,  array[i].potenza,
                    array[i].temperatura,   array[i].processi,  array[i].memoria);
        }
        *passi += 1;
    }
    return esito;
}// END 'CercaCpuArray
```

5) Testing del programma:

Testing ‘GeneratoreFileASD.c’

Viene chiesto dal programma quante CPU si vogliono analizzare.

Quando vengono inseriti dei valori non accettabili il programma ce lo comunica e ci chiede nuovamente di inserire dei valori.

Al primo valore accettabile che inseriamo il programma procede con l’esecuzione e termina.

```
Inserire numero di cpu da analizzare:
a

Il valore inserito non e' accettabile
Perfavore inserire un numero naturale diverso da 0
-45

Il valore inserito non e' accettabile
Perfavore inserire un numero naturale diverso da 0
0

Il valore inserito non e' accettabile
Perfavore inserire un numero naturale diverso da 0
45
END
```

Testing ‘ProgettoASD.c’

Inizialmente vien chiesto il tipo di struttura che si vuole utilizzare.

Se si vanno a inserire risposte non accettabili viene comunicato e va inserita nuovamente una risposta.

Al primo input accettabile si procede e viene visualizzata una stampa di tutti i nomi delle CPU disponibili.
Anche qui se vengono inseriti input non accettabili ci viene comunicato e va reinserito il nome di una CPU.

Quando inseriamo il nome di una CPU accettabile ma non presente il programma eseguirà le ricerche e non trovando alcuna CPU con il nome richiesto ci verrà comunicato.

Successivamente ci viene chiesto se si vuole effettuare un’altra ricerca e anche qui quando vengono inseriti input non accettabili ci viene comunicato e dovremmo inserire nuovamente un’altra risposta.
Alla prima risposta accettabile il programma procede con l’esecuzione.

```
Quale tipo di struttura si vuole utilizzare?
1) Albero
2) Array?
Digitare '1' o '2'
a
Scelta non consentita
Perfavore scegliere tra '1'  e  '2'
-2
Scelta non consentita
Perfavore scegliere tra '1'  e  '2'
1
E' stato scelto l'albero

Lista dei nomi delle cpu che sono state analizzate
        DYD650
        MLY621
        PDF435
        SJY732
        YVV751

Inserire il nome di una cpu sopra-visualizzata di cui si vogliono visualizzare le prestazioni
a

Errore nell'aquisizione del nome della cpu scelta...
Controllare la lunghezza del nome inserito
-484

Errore nell'aquisizione del nome della cpu scelta...
Controllare la lunghezza del nome inserito
qqqqqq

Non e' stata trovata alcuna CPU, (attenzione inserire lettere maiuscole)

I passi compiuti dall'algoritmo per trovare e stampare i dati della CPU richiesta: 25
Si disidera cercare un'altra CPU?
Premere '1' per rispondere'si'
Premere '2' per rispondere 'no'
a

Risposta non accettabile...
Perfavore selezionare '1' o '2'
-2

Risposta non accettabile...
Perfavore selezionare '1' o '2'
1
```

All’inserimento di un nome accettabile e presente tra le opzioni il programma eseguirà la ricerca e stamperà a video tutte le informazioni possedute.

Ci verrà richiesto nuovamente se si vorrà effettuare un’altra ricerca, se vogliamo chiudere il programma ci basterà rispondere di non voler cercare altro.

Ci viene anche stampato il numero di passi che l’algoritmo esegue per darci ciò che abbiamo chiesto che ci servirà per analizzare gli algoritmi.

```
Inserire il nome di una cpu sopra-visualizzata di cui si vogliono visualizzare le prestazioni
DYD650
1               DYD650          8.09            38.42           39.35           1633.42
2               DYD650          12.29           40.96           30.38           795.63
3               DYD650          13.57           43.79           31.45           840.08
4               DYD650          9.61            51.11           10.17           1473.88
5               DYD650          14.70           47.86           11.02           543.29
6               DYD650          10.49           83.92           34.97           1756.95
7               DYD650          12.76           69.73           14.06           1921.90
8               DYD650          8.29            50.02           23.27           1535.34
9               DYD650          10.89           72.85           34.01           1978.68
10              DYD650          10.66           60.43           14.68           1315.98
11              DYD650          5.53            59.00           33.43           1694.76
12              DYD650          9.11            74.31           37.80           550.51
13              DYD650          14.06           49.77           19.13           502.54
14              DYD650          14.80           34.16           2.57            193.23
15              DYD650          8.84            75.55           24.63           957.90
16              DYD650          10.67           68.62           6.90            1527.20
17              DYD650          7.32            51.99           39.04           1720.21
18              DYD650          9.68            42.63           4.69            1258.83
19              DYD650          8.53            84.04           38.27           732.16
20              DYD650          9.46            75.23           5.33            792.10
21              DYD650          5.75            57.36           1.64            637.66
22              DYD650          13.09           44.75           24.41           1820.72
23              DYD650          5.49            77.38           30.89           518.69
24              DYD650          6.17            61.55           19.92           179.26

I passi compiuti dall'algoritmo per trovare e stampare i dati della CPU richiesta: 27
Si disidera cercare un'altra CPU?
Premere '1' per rispondere'si'
Premere '2' per rispondere 'no'
2

FINE PROGRAMMA
```

Il funzionamento del programma con l’array e non con l’albero per quanto riguarda l’utilizzo utente è il medesimo.

6) Valutazione della complessità del programma:

Analisi teorica della complessità:

Algoritmo di ricerca con albero binario:

Per trovare la CPU scelta dall'utente viene utilizzata la funzione ricorsiva 'CercaCpuAlbero'.
Quest'ultima scorre l'albero grazie ad un ciclo, fin quando non trova l'elemento scelto o arriva alla fine dell'albero senza trovarne alcuno.
Quando ciò che cerca viene trovato vengono stampate le informazioni di quel nodo, tuttavia saranno le informazioni riguardanti solo un'ora, perciò la funzione chiama sé stessa.
La complessità di questo algoritmo è:

In un caso ottimale (albero vuoto) risulta:

$T(n) = O(1)$ complessità costante

In un caso pessimo (albero non vuoto e cerca un elemento non presente, deve perciò essere percorso interamente) risulta:

$T(n) = O(n)$ complessità lineare

Algoritmo di visita dell'array:

Per trovare la CPU scelta dall'utente viene utilizzata la funzione 'CercaCpuArray'.
Quest'ultima attraversa gli elementi dell'array linearmente, ovvero nell'ordine in cui sono stati memorizzati, mentre li controlla (verifica se quello che sta cercando e quello che sta controllando siano uguali).
In questa funzione viene utilizzata un'istruzione di selezione *if* (β) $\mathcal{S}_1$ *else* $\mathcal{S}_2$, la cui complessità è:
N.B. l'istruzione else non è esplicitamente presente ma il funzionamento è il medesimo.

Caso Ottimo (array nullo, viene subito eseguita l'istruzione S_2):

$T(n) = O(1)$ complessità costante

Caso pessimo (array non nullo, viene eseguita l'istruzione S_1 fino a scorrere tutto l'array):

$T(n) = O(n)$ complessità lineare

Analisi sperimentale della complessità

Con l'analisi sperimentale andremo a verificare se quanto detto nell'analisi teorica risulti corretto meno.
In particolare, vedremo l'andamento dei due algoritmi all'aumentare delle CPU e li confronteremo.

CPU	ALBERO	ARRAY
10	26	241
100	27	2401
500	28	12001
1000	30	24001
2500	30	60001
5000	33	120001
10000	31	240001