Incicco Matteo [MAT. 261716]

Algoritmi e Strutture Dati

Docente: Prof. Valerio Freschi

Relazione sul progetto della sessione autunnale
2013/2014

Specifica del problema

Si supponga di elaborare i dati relativi a delle prove d'esame a risposta multipla. Le informazioni associate al problema sono contenute in un file di testo datiesame.txt che contiene nelle prime tre righe il numero di studenti (N_s), il numero di domande della prova (N_q) e una stringa di caratteri con le risposte corrette. Ogni linea seguente del file contiene un intero che rappresenta l'identificativo del singolo studente e, dopo uno spazio, una stringa di N_q caratteri che rappresenta le risposte dello studente.

Scrivere un programma ANSI C che esegue le seguenti elaborazioni:
1. Acquisisce da file le informazioni relative alla prova. Il formato del file `e del tipo:

<Numero totale degli studenti>
<Numero delle domande della prova>
<Stringa risposte corrette>
<ID1> <stringaID1>
<ID2> <stringaID2>
...
<IDNs> <stringaIDNs>

Ad esempio:

3
5
dbbac
121 babac
107 baadc
104 baaac

2. Produce in output a monitor un report d'esame contenente la stringa delle risposte corrette, gli ID dei vari studenti seguiti dai punteggi percentuali ottenuti e, infine, le informazioni su quanti studenti hanno sbagliato un dato quesito. Ad esempio:

Report d'esame

Domanda 1 2 3 4 5

Risposta b a a d c

ID	Punteggio (%)
121	60
107	100
104	80

Domanda 1 2 3 4 5

Sbagliata da 0 0 1 2 0

3. Produce, su richiesta, una versione del report ordinata in base all'identificativo studente oppure calcola la media e la mediana dei voti ottenuti.

Per quanto riguarda l'analisi teorica si devono studiare le complessità degli algoritmi di calcolo e output del report, di calcolo del report ordinato e di calcolo di media e mediana.

Si deve anche verificare sperimentalmente la complessità dell'ordinamento e del calcolo di media e mediana, generando casualmente una sequenza di ID studente e stringhe di risposte (la cui lunghezza si può ipotizzare costante al variare di N_S) da fornire come input all'algoritmo per valori crescenti di N_S.

Analisi del Problema

L'input è costituito da:
- Numero degli studenti,
- Numero delle domande della prova,
- Stringa risposte corrette,
- Id di ogni studente,
- Risposte di ogni studente.

I dati in input vengono acquisiti da un file chiamato "datiesame.txt".

L'output consiste nel comunicare all'utente i dati acquisiti dal file calcolando il punteggio espresso in percentuale di ogni studente, il numero degli studenti che hanno sbagliato ogni domanda e, su richiesta, un report riordinato in base agli id o il calcolo di media e mediana sui punteggi espressi in percentuale.

Per calcolare il punteggio espresso in percentuale è necessario ottenere il numero delle risposte esatte di ogni utente e applicare la proporzione:

a : b = c : d

Per fornire un report riordinato in base agli id è necessario implementare un algoritmo di ordinamento. L'algoritmo scelto in questo caso è il "quicksort".

Per il calcolo della media si può far fede alla formula:

media = somma dati / numero dati

Per il calcolo della mediana si devono osservare delle regole:

1. si devono disporre i valori in ordine crescente e contare il numero totale n di dati;
2. se il numero (n) di dati è dispari, la mediana corrisponde al valore numerico del dato centrale, quello che occupa la posizione (n + 1) / 2;
3. se il numero (n) di dati è pari, la mediana si calcola effettuando la media tra i due valori centrali che occupano le posizioni n / 2 e n / 2 + 1.

Progettazione dell’algoritmo

Panoramica generale della risoluzione del problema.

Per arrivare agli output richiesti è necessario soffermarsi sulle fasi, ovvero i punti richiesti, del problema. Si possono perciò dividere in 3 macroargomenti:

1. Acquisizione e validazione dati da file;
2. Stampa del report;
3. Ordinamento o calcolo media e mediana.

Analizziamo la questione punto per punto.

---

1. Acquisizione e validazione dati da file

La prima parte del programma si concentra sull’acquisizione e la validazione dei dati da file.
Possiamo perciò suddividere questa prima parte in altri sottoparti.

Il file in questione, chiamato “datiesame.txt”, deve essere disposto come segue:

Prima riga:	< Numero totale degli studenti >
Seconda riga:	< Numero delle domande della prova >
Terza riga:	< Stringa risposte corrette >
Quarta riga:	< ID1 > < Stringa risposte ID1 >
.	. .
.	. .
N-esima riga:	< IDn > < Stringa risposte IDn >

Esempio:

Prima riga:	3
Seconda riga:	4
Terza riga:	abcd
Quarta riga:	121 bacd
Quinta riga:	122 bcde
Sesta riga:	123 bdea

È quasi scontato quindi dire che il programma deve segnalare un errore qualora il file non sia organizzato come sopra illustrato.

Si può quindi dividere l’acquisizione e la validazione in altre tre parti:

- Acquisizione e validazione del numero totale degli studenti e del numero delle domande della prova;

- Acquisizione e validazione della stringa delle risposte corrette;

- Acquisizione e validazione degli id e delle risposte degli studenti.

Questa divisione è necessaria anche per valorizzare la leggibilità e la comprensibilità del codice.

Qualora ci fosse un errore non verranno acquisiti i dati successivi alla posizione dell'errore: se è presente un dato sbagliato al posto del numero degli studenti, verrà stampato un preciso messaggio e non verranno acquisiti i dati successivi; di conseguenza l'esecuzione del programma termina.

Nell'acquisizione da file c'è da tener conto anche la questione della terminazione del file; quindi, non si deve solo controllare la correttezza sintattica dei dati.

La variabile "eof" avrà valore 1 quando si è acquisito il dato (non validato) e -1 quando il file è terminato e di conseguenza l'acquisizione non è riuscita.
Il messaggio di errore in questo caso è:
- "Errore: file vuoto." quando non è riuscita l'acquisizione nemmeno del numero degli studenti;
- "Errore: file terminato prima dell'acquisizione…" negli altri casi.

Per validare un intero è necessario acquisirlo come stringa e controllare che in ogni posizione sia presente un numero da '0' a '9'. Nella prima posizione non dovrà essere presente lo '0' in quanto il programma non accetta lo 0; il numero minimo è 1. Non avrebbe senso fare un report d'esame con 0 studenti o con 0 domande.

Per fare ciò si è creata una apposita funzione "valida_intero".
"Valida_intero" restituirà 1 se il numero è errato o 0 se il numero è corretto.

Si hanno delle variabili specifiche per ogni errore.
Es. "errore_nstud" per segnalare un errore nel numero studenti.

Il numero massimo accettato è un numero a 5 cifre, quindi 99999.

Se il numero degli studenti è acquisito correttamente si procede con l'acquisizione del secondo numero ovvero il numero delle domande richiamando di nuovo la funzione "valida_intero".

Ora si passa all'acquisizione e alla validazione della stringa delle risposte corrette.
Prima di acquisire la stringa si dovrà allocare la variabile con la lunghezza contenuta nella variabile "n_dom".

Una volta che l'acquisizione è andata a buon fine viene chiamata la funzione "valida_stringa" che si occupa di validare la stringa in base ad un dizionario precedentemente dichiarato; il dizionario "caratteri".

Il dizionario “caratteri” è un vettore che contiene le seguenti lettere consentite:

a	b	c	d	e	f	g
0	1	2	3	4	5	6

La funzione restituirà 1 se la stringa è errata o 0 se è corretta.

“Valida_stringa” controllerà anche che la lunghezza della stringa acquisita sia della dimensione scritta nella seconda riga; in quanto non ci può essere una prova con più risposte del dovuto, ma nemmeno una prova senza una risposta.

Una volta che è stata acquisita e validata la stringa “risp_cor” si proseguirà con gli id e le risposte di ogni singolo studente.

Prima di fare un ciclo per l’acquisizione si dovranno allocare le variabili restanti:
- la “riga_intera”;
- il vettore “id” allocato in base al numero degli studenti;
- e la matrice “risp_cor” allocata con tante colonne quante sono le risposte corrette e tante righe quanti sono gli studenti.

Il vettore di caratteri “riga_intera” conterrà l’acquisizione dell’intera riga che andrà poi suddivisa.

Es. “riga_intera”:

1	2	1		a	b	c	d
0	1	2	3	4	5	6	7

Chiamando la funzione “divisione_stringa” si otterrà il seguente risultato:

“id”

121

“risp[]”

a	b	c	d
0	1	2	3

L’id acquisito e le risposte di ogni studenti saranno ovviamente validati con le funzione sopracitate.

Per acquisire tutti gli id e tutte le risposte si effettuerà un ciclo “for” che concluderà o quando si è trovato un errore o quando si è acquisiti tutti i dati.

La variabile “errore_idrisp” avrà valore 0 quando è tutto corretto, valore 1 quando c’è un errore sintattico su un acquisizione (id o risposte), valore 2 quando il file termina prima del previsto.

Usciti dal ciclo si stamperò l’errore specifico.

Eventuali righe superflue non verranno considerate.

---

2. Stampa del report

Una volta acquisiti e validati tutti i dati si procede con il secondo macroargomento.

Per stampare il report viene chiamata la funzione “report”, dopo aver allocato i vettori che conterranno la percentuale delle risposte esatte e il numero degli studenti che hanno sbagliato ogni singola domanda.

La funzione “report” avrà al suo interno diverse chiamate a funzioni specifiche:

- una funzione “risposte” che si occupa di stampare la stringa delle risposte corrette come segue:

  es.

Domanda	1	2	3	4	5
Risposta	a	b	c	d	e

- Una funzione “punteggio” che si occupa di calcolare e stampare l’id e il punteggio in percentuale di ogni studente.

  Per fare ciò la funzione punteggio si avvale della funzione “conta_carattere” che confronta le risposte dello studente con la stringa delle risposte corrette e conta quanti caratteri nella stessa posizione sono uguali ai caratteri della stringa “risp_cor”.

  Una volta che si ha il valore intero delle risposte esatte date dallo studente si effettua la proporzione:

  a : b = c : d

  es.

  4 : 5 = x : 100

  dove 4 è il numero di risposte corrette su 5, quindi

(4*100)/5 = 80

80 % di risposte esatte.

Calcolato il punteggio in percentuale la funzione chiamerà un altra funzione (“stampa_punt”) che stamperà ben incolonnati i seguenti dati:

es.

ID	Punteggio (percentuale)
121	80
122	90
etc.	

- Ed infine la funzione “errori” che conta e stampa il numero degli studenti che hanno sbagliato ogni singola domanda.

  Ci saranno due cicli “for” annidati. Il primo scandisce gli studenti il secondo confronta l’array delle risposte dello studente con l’array delle risposte corrette. Verrà quindi incrementata il vettore di interi numero errori ogni qualvolta viene trovata una domanda sbagliata.

  Di conseguenza la funzione stamperà i risultati da essa calcolati come segue:

  es.

Domanda	1	2	3	4	5
Sbagliata da	1	0	3	2	0

---

3. Ordinamento o calcolo media e mediana

Una volta stampato il report verrà stampato a video all’utente il seguente messaggio:

“Digitare ‘o’ se si vuole un report ordinato in base all’identificativo utente.
Digitare ‘m’ se si vuole calcolare la media e la mediana dei voti ottenuti dagli studenti.
Digitare ’t’ se si vuole terminare l’esecuzione del programma.”

Quindi l’utente dovrà inserire ‘o’ o ‘m’ o ’t’. Qualora scriverà un carattere diverso verrà stampato il seguente messaggio di errore:

“Errore: non si è immesso un carattere corretto.
N.B. il software è case sensitive.”

Una volta acquisito correttamente il carattere verranno effettuate le operazioni adeguate.

- Se la risposta dell'utente è 'o' :

  In un primo luogo viene chiamata la funzione "quicksort" per ordinare il report in base all'id.

  L'algoritmo quicksort è stato opportunamente modificato in quanto ogni valore che si scambia all'array id deve essere scambiato anche all'array delle percentuali.

  Successivamente chiamerà la funzione "stampa_punt" per stampare gli array riordinati.

- Se la risposta dell'utente è 'm':

  Verrà chiamato di nuovo la funzione "quicksort" ma quasta volta l'array primario che viene ordinato è quello delle pervenutali, di conseguenza a quest'ultimo l'array id.

  Per calcolare poi la media si fa un "for" che sommerà tutte le percentuali in "somma_media"; successivamente viene diviso il valore di "somma_media" con il numero degli studenti.

  es.

  Percentuali
  100
  80
  60

  "somma_media" = 240 —> 240 / 3 = 80

  La media sarà quindi 80, e verrà stampato il seguente messaggio:
  "La media delle percentuali è: 80.000000."

  Per calcolare la mediana invece si deve verificare se il numero degli studenti è dispari o pari; per fare ciò si usa l'operatore "%" il quale è utile per calcolare il resto.

  es.
  resto = < numero degli studenti > % 2

  "resto" avrà valore 1 se il numero degli studenti è dispari o valore 0 se il numero degli studenti è pari.

  Se il numero degli studenti è dispari la mediana corrisponde al valore numerico del dato centrale, quindi quello che occupa la posizione (n + 1) / 2.

  es.
  Percentuali
  60
  70

90

La mediana è 70 e verrà stampato il seguente messaggio:
“La mediana delle percentuali è: 70.000000.”

Se il numero degli studenti è pari la mediana corrisponde alla media dei due valori del centro, ossia quelli in posizione n / 2 e n / 2 + 1.

es.
Percentuali
60
80
90
90

La mediana è 85 e verrà stampato il seguente messaggio:
“La mediana delle percentuali è: 85.000000.”

La media e la mediana vengono calcolati come “float" per aumentare la precisione del risultato.

- Se la risposta dell’utente è ’t’ semplicemente non viene effettuata nessuna operazione e di conseguenza termina l’esecuzione del programma.

Implementazione dell'algoritmo

Di seguito è riportato il codice sorgente, commentato, dell'implementazione dell'algoritmo in C:

```
/*****************************************/
/* inclusione delle librerie standard del c */
/*****************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


/*********************************/
/* dichiarazione delle funzioni */
/*********************************/

int valida_intero(const char *intero_stringa,
                  int *intero_validato);
int valida_stringa(const char *str,
                   const char *dizionario,
                   const int len_str);
void divisione_stringa(char *stringa_iniziale,
                       int inizio,
                       int fine,
                       char *stringa_finale);
void report(const char *r_corrette,
            const int *id_stud,
            char **risp_stud,
            int *perc_stud,
            const int num_stud,
            int *num_errori);
void risposte(const char *risposte);
void punteggio(const char *r_corrette,
               const int *id_stud,
               char **risposte_stud,
               int *percentuale_stud,
               const int numero_stud);
int conta_carattere(const char *str1,
                    const char *str2);
void stampa_punt(const int *id_stud,
                 const int *percentuale_stud,
                 const int numero_stud);
void errori(const char *risposte_corrette,
            char **risposte_studenti,
            const int numero_studenti,
            int *numero_errori);
void quicksort(int *a_primario,
```

```
                int sx,
                int dx,
                int *a_secondario);


/**********************************/
/* definizione della funzione main */
/**********************************/

int main(void)
{
  /* dichiaro le variabili locali alla funzione */
  int n_stud = 0,                       /* i.: numero studenti */
      n_dom = 0;                        /* i.: numero domande */

  int *id;                              /* i.: id studente */
  char **risp;                          /* i.: risposte studente */

  char *risp_cor;                       /* i.: risposte corrette */
  int *n_err;                           /* o.: numero risposte errate */
  int *perc;                            /* o.: percentuale risposte corrette */

  const char caratteri[7] = {'a',       /* l.: caratteri consentiti */
                             'b',
                             'c',
                             'd',
                             'e',
                             'f',
                             'g'};

  FILE *file_datiesame;                 /* l.: puntatore al file di input */

  int i;                                /* l.: contatore */

  char intero_validare[6];              /* l.: intero da validare */

  int eof;                              /* l.: controllo fine file */

  int len_riga;                         /* l.: lunghezza riga intera */

  char *res;                            /* l.: esito fgets */

  char *riga_intera;                    /* i./l.: riga intera da suddividere */

  int errore_nstud = 0,                 /* l.: indica se n_stud è errata */
      errore_ndom = 0,                  /* l.: indica se n_dom è errata */
      errore_rispcor = 0,               /* l.: indica se risp_cor è errata */
      errore_idrisp = 0,                /* l.: acquisizione id e risp err. */
      esito_acquisizione = 0;           /* l.: esito complessivo acquisizione */

  char risposta_utente;                 /* i.: risposta utente della domanda */
  char newline;                         /* l.: acquisizione newline */
```

```
float f_nstud;                          /* l.: float n_stud */
float somma_media;                      /* l.: somma percentuali per la media */
float media;                            /* o.: media delle percentuali */

int resto;                              /* l.: resto per calcolo mediana */
float mediana_1 = 0;                    /* l.: mediana prima della metà */
float mediana_2 = 0;                    /* l.: mediana dopo la metà */
float mediana;                          /* o.: mediana delle percentuali */

/* apre il file */
file_datiesame = fopen ("datiesame.txt",
                                "r");

/* verifica errori in apertura */
if (file_datiesame == NULL)
{
  perror ("Errore in apertura del file.\n");
  exit(1);
}


/* acquisisce il numero degli studenti dalla prima riga */
eof = fscanf (file_datiesame,
                "%s",
                intero_validare);

/* controllo che il file non sia finito per continuare */
if (eof != 1)
  printf("\nErrore: file vuoto.\n");
else
{
  /* validazione n_stud */
  errore_nstud = valida_intero(intero_validare,
                                    &n_stud);

  /* stampa il messaggio di errore opportuno */
  if (errore_nstud == 1)
  {
    printf ("\nErrore: il numero degli studenti contenuto nella\n"
            "       prima riga non è corretto.\n"
            "       Modificare il file.\n"
            "\n N.B. Il numero massimo accettato è 99999.\n");
  }

  /* acquisisce il numero delle domande dalla seconda riga */
  eof = fscanf (file_datiesame,
                "%s",
                intero_validare);

  /* controllo che il file non sia finito per continuare */
  if (eof != 1)
```

```
    printf("\nErrore: file terminato prima dell'acquisizione\n"
           "        della seconda riga.\n");
  else if (errore_nstud != 1)
  {
    /* validazione n_dom */
    errore_ndom = valida_intero(intero_validare,
                                &n_dom);

    /* stampa il messaggio di errore opportuno */
    if (errore_ndom == 1)
    {
      printf("\nErrore: il numero delle domande contenuto nella\n"
             "        seconda riga non è corretto.\n"
             "        Modificare il file.\n"
             "\n N.B. Il numero massimo accettato è 99999.\n");
    }
  }
}

/* continua controllando che tutto sia corretto,
   altrimenti non esegue le prossime operazioni */
if (eof == 1)
{
  if (errore_nstud != 1)
  {
    if (errore_ndom != 1)
    {
      /* allocazioni */
      risp_cor = (char *) malloc((n_dom + 1) * sizeof (char));

      /* acquisisce la stringa delle risposte corrette */
      eof = fscanf(file_datiesame,
                   "%s",
                   risp_cor);

      /* controllo che non sia terminato il file */
      if (eof != 1)
          printf("\nErrore: file terminato prima dell'acquisizione\n"
                 "        della stringa delle risposte corrette\n"
                 "        nella terza riga.\n");
      else
      {
        /* validazione risp_cor */
        errore_rispcor = valida_stringa(risp_cor,
                                        caratteri,
                                        n_dom);
        /* stampa il messaggio di errore opportuno */
        if (errore_rispcor == 1)
        {
          printf("\nErrore: la stringa delle risposte esatte contenuta\n"
                 "        nella terza riga non è corretta.\n"
                 "        Modificare il file.\n"
```

```
                "\n N.B. I caratteri accettati sono: a, b, c, d, e, f, g.\n"
                " N.B. La lunghezza deve essere quella contenuta nella\n"
                "      seconda riga del file.\n");
      }
     }
    }
   }
}

/* continua controllando che tutto sia corretto,
   altrimenti non esegue le prossime operazioni */
if (eof == 1)
{
  if (errore_nstud != 1)
  {
    if (errore_ndom != 1)
    {
      if (errore_rispcor != 1)
      {
        /* allocazioni */
        riga_intera = (char *) malloc((6 + 1 + n_dom + 1) * sizeof (char));
        id = (int *) malloc(n_stud * sizeof (int));

        /* viene allocata la matrice "risp" con una malloc
           in modo da poterla utilizzare all'interno delle
           funzioni */
        risp = (char **) malloc(sizeof(char *) * n_stud);
        for(i = 0;
            i < n_stud;
            i++)
          risp[i] = (char *) malloc(sizeof(char) * n_dom + 1);

        /* acquisizione riga intera */
        for (i = 0, errore_idrisp = 0;
             i < n_stud && errore_idrisp == 0;
             i ++)
        {
          /* salto new line rimanente da fscanf */
          if (i == 0)
          {
            res = fgets(riga_intera,
                        2,
                        file_datiesame);

            if (res == NULL)
            {
              errore_idrisp = 2;
            }
            else
            {
              res = fgets(riga_intera,
                          6 + 1 + n_dom +1,
```

```
                    file_datiesame);

        if (res == NULL)
        {
          errore_idrisp = 2;
        }
        else
        {
          /* divisione e validazione */
      len_riga = strlen (riga_intera);

          /* rimozione \n */
          riga_intera[len_riga - 1] = '\0';
        len_riga = strlen(riga_intera);

          if (len_riga < 1 + 1 + n_dom)
      {
        errore_idrisp = 1;
      }
      else
      {
        /* divide la stringa per acquisire le risposte */
        divisione_stringa (riga_intera,
                           len_riga - n_dom,
                           len_riga - 1,
                           risp[i]);

        /* valida le risposte */
        errore_idrisp = valida_stringa (risp[i],
                                        caratteri,
                                       n_dom);

        if (errore_idrisp == 0)
        {
          /* divide la stringa per acquisire l'id */
          divisione_stringa (riga_intera,
                             0,
                             len_riga - n_dom - 1 - 1,
                             intero_validare);

          /* valida l'id */
          errore_idrisp = valida_intero (intero_validare,
                                         &id[i]);
        }
      }
          }
    }
}
else
{
  res = fgets(riga_intera,
              6 + 1 + n_dom + 1,
```

```
                file_datiesame);

      if (res == NULL)
    {
        errore_idrisp = 2;
      }
      else
      {
          /* divisione e validazione */
     len_riga = strlen(riga_intera);

          /* rimozione \n */
     if (riga_intera[len_riga - 1] == '\n')
     {
           riga_intera[len_riga - 1] = '\0';
         len_riga = strlen(riga_intera);
          }

          if (len_riga < 1 + 1 + n_dom)
     {
      errore_idrisp = 1;
     }
     else
     {
      /* divide la stringa per acquisire le risposte */
      divisione_stringa (riga_intera,
                         len_riga - n_dom,
                 len_riga - 1,
                           risp[i]);

      /* valida le risposte */
      errore_idrisp = valida_stringa (risp[i],
                                        caratteri,
                                      n_dom);

      if (errore_idrisp == 0)
      {
       /* divide la stringa per acquisire l'id */
       divisione_stringa (riga_intera,
                          0,
                          len_riga - n_dom - 1 - 1,
                          intero_validare);

       /* valida l'id */
       errore_idrisp = valida_intero (intero_validare,
                                        &id[i]);
      }
     }
     }
  }
}
 }
```

```
    }
  }
}

/* se c'è un errore nell'acquisizione degli id e delle
   risposte stampa il messaggio di errore */
if (errore_idrisp == 2)
{
  printf("\nErrore: file terminato prima di acquisire\n"
         "        tutti gli id e tutte le risposte.\n");
}
else if (errore_idrisp == 1)
{
  printf("\nErrore: id e risposte degli studenti non sono corretti.\n"
        "        Modificare il file.\n"
        "\n N.B. l'id massimo accettato è 99999.\n"
          " N.B. I caratteri accettati sono: a, b, c, d, e, f, g.\n"
          " N.B. La lunghezza deve essere quella contenuta nella\n"
        "      seconda riga del file.\n");
}

/* assegna esito di tutte le acquisizioni
   a esito_acquisizione */
if (eof == 1 &&
    errore_nstud == 0 &&
    errore_ndom == 0 &&
    errore_rispcor == 0 &&
    errore_idrisp == 0)
  esito_acquisizione = 1;
else
  esito_acquisizione = 0;


/* se non si sono verificati errori durante
   l'acquisizione fa il report d'esame e chiede
   all'utente cosa si intende fare a questo
   punto dell'esecuzione */
if (esito_acquisizione == 1)
{
  /* allocazione perc e n_err */
  perc = (int *) malloc(n_stud * sizeof (int));
  n_err = (int *) malloc(n_dom * sizeof (int));

  /* viene inizializzato l'array "n_err" */
  for (i = 0;
       i < n_dom;
       i ++)
  {
    n_err[i] = 0;
  }

  /* chiamata alla funzione report */
```

```
report(risp_cor,
       id,
       risp,
       perc,
       n_stud,
       n_err);

/* effettua domanda all'utente */
printf("\nDigitare 'o' se si vuole un report ordinato\n"
         "in base all'identificativo studente.\n\nDigitare"
         " 'm' se si vuole calcolare la media e\nla mediana"
         " dei voti ottenuti dagli studenti.\n\nDigitare 't'"
         " se si vuole terminare l'esecuzione\ndel programma.\n\n");

/* acuisisce risposta dell'utente */
do
{
  scanf("%c/%c",
          &risposta_utente,
          &newline);

  /* se non si è immesso un carattere di quelli indicati
       stampa il relativo messaggio di errore */
  if (risposta_utente != 'o' &&
        risposta_utente != 'm' &&
        risposta_utente != 't')
  {
    printf("Errore: non si è immesso un carattere corretto.\n\n"
             "N.B. il software è case sensitive.\n");
  }
}
while (risposta_utente != 'o' &&
         risposta_utente != 'm' &&
         risposta_utente != 't');

/* procede in base alla risposta dell'utente */
if (risposta_utente == 'o')
{
  /* chiama quicksort per ordinare in base all'id */
  quicksort(id,
              0,
              n_stud - 1,
              perc);

  /* stampa id e punteggio */
  stampa_punt(id,
                perc,
                n_stud);
}
else if (risposta_utente == 'm')
{
  /* chiama quicksort per ordinare in base alla percentuale */
```

```
    quicksort(perc,
              0,
              n_stud - 1,
              id);

    /* calcola e stampa la media delle percentuali */
    for (i = 0, somma_media = 0;
         i < n_stud;
         i++)
    {
      somma_media = somma_media + perc[i];
    }

    f_nstud = n_stud;

    media = somma_media / f_nstud;

    printf("\nLa media delle percentuali è:\t%f.\n",
           media);

    /* calcola e stampa la mediana delle percentuali */
    resto = n_stud % 2;

    if (resto == 1)
    {
      mediana = perc[((n_stud + 1) / 2) - 1];
    }
    else
    {
      mediana_1 = perc[(n_stud / 2) - 1];
      mediana_2 = perc[((n_stud / 2) + 1) - 1];
      mediana = (mediana_1 + mediana_2) / 2;
    }

    printf("\nLa mediana delle percentuali è:\t%f\n\n",
           mediana);
   }
  }

  /* chiudo il file aperto */
  fclose(file_datiesame);

  return(0);
}


/*****************************/
/* definizione delle funzioni */
/*****************************/


/* definizione della funzione che valida
```

```
   un intero, restituirà 1 se è errato o
   0 se è corretto */
int valida_intero(const char *intero_stringa,       /* i.: stringa intero da validare */
                  int *intero_validato)             /* o.: intero validato */
{
  /* dichiarazione delle variabili locali alla funzione */
  int i = 0,                                        /* l.: indice */
      len;                                          /* l.: lunghezza stringa intero */

  int errore = 0;                                   /* o.: errore che rest. la fun. */

  len = strlen(intero_stringa);

  /* controlla che len non sia più grande di 5 */
  if (len >= 6)
    errore = 1;
  else
  {
    /* controlla che nella prima posizione non ci
       sia uno 0 */
    if (intero_stringa[i] != '0')
    {
      /* controlla che in ogni posizione dell'array ci
         sia una cifra da 0 a 9 */
      while (i < len)
      {
        if (errore == 1)
          i = len;
        else
        {
          /* se l'elemento nella posizione i è diverso da un
             numero da 0 a 9 assegna 1 a "errore" */
          if (intero_stringa[i] != '0' &&
              intero_stringa[i] != '1' &&
                intero_stringa[i] != '2' &&
              intero_stringa[i] != '3' &&
                intero_stringa[i] != '4' &&
              intero_stringa[i] != '5' &&
                intero_stringa[i] != '6' &&
              intero_stringa[i] != '7' &&
                intero_stringa[i] != '8' &&
              intero_stringa[i] != '9')
            errore = 1;
        }

        i ++;
      }
    }
    else
      errore = 1;
  }
```

```
  /* se non ha trovato nessun errore usa la funzione atoi
     per convertire la stringa in intero e assegnarlo a
     "intero_validato" */
  if (errore != 1)
    *intero_validato = atoi(intero_stringa);

  return errore;
}


/* definizione della funzione che valida una stringa,
   restituirà 1 se è errata o 0 se è corretta */
int valida_stringa(const char *str,          /* i.: stringa da validare */
                   const char *dizionario,   /* i.: caratteri consentiti */
                   const int len_str)        /* i.: lunghezza corretta stringa */
{
  /* dichiarazione delle variabili locali alla funzione */
  int i,                                     /* l.: indice */
      j;                                     /* l.: indice */

  int len,                                   /* l.: lunghezza "str" */
      dim_diz;                               /* l.: dimensione dizionario */

  int errore;                                /* o.: errore */

  len = strlen(str);
  dim_diz = strlen(dizionario);

  /* controlla che la lunghezza della stringa
     acquisita sia della lunghezza giusta */
  if (len != len_str)
    errore = 1;
  else
  {
    /* ciclo che scandisce la stringa da validare */
    for (i = 0, errore = 0;
         i < len;
         i ++)
    {
      if (errore == 1)
        i = len;
      else
      {
        errore = 1;

        /* ciclo che scandisce e confronta il dizionario */
        for (j = 0;
             j < dim_diz;
             j ++)
        {
          if (str[i] == dizionario[j])
          {
```

```
            errore = 0;
            j = dim_diz;
          }
        }
      }
    }
  }

  return errore;
}


/* definizione della funzione necessaria per suddividere una stringa,
   essa copia il contenuto di una stringa in un altra stringa da un
   indice di inizio ad un indice di fine */
void divisione_stringa(char *stringa_iniziale,          /* i.: str. da cop. */
                       int inizio,              /* i.: indice inizio */
                       int fine,                /* i.: indice fine */
                       char *stringa_finale)          /* o.: str. risultante */
{
  /* dichiarazione delle variabili locali alla funzione */
  int i;                                        /* l.: variabile contatore */

  /* ciclo utilizzato per copiare valori da 'stringa_iniziale' a
     'stringa_finale' confrontando 'inizio' e 'fine' */
  for (i = 0;
       inizio <= fine;
       inizio ++, i ++)
  {
    stringa_finale[i] = stringa_iniziale[inizio];
  }

  /* assegno '\0' all'ultima posizione di stringa_finale */
  stringa_finale[i] = '\0';
}


/* definizione della funzione che si occupa di fare un
   report dell'esame */
void report(const char *r_corrette,         /* i.: risposte corrette */
            const int *id_stud,                     /* i.: id dello studente */
            char **risp_stud,               /* i.: risposte dello studente */
            int *perc_stud,                 /* o.: perc. risp. esatte stud. */
            const int num_stud,             /* l.: numero degli stud. */
            int *num_errori)                /* o.: numero errori domande */
{
  /* stampa a video dell'intestazione "Report d'esame" */
  printf("\n\n\t\t\tReport d'esame\n");

  /* stampa a video delle risposte corrette */
  risposte(r_corrette);
```

```
  /* stampa a video del punteggio di ogni studente */
  punteggio(r_corrette,
            id_stud,
            risp_stud,
            perc_stud,
            num_stud);

  /* stampa a video degli errori commessi dagli studenti
     su ogni domanda */
  errori(r_corrette,
           risp_stud,
         num_stud,
           num_errori);
}


/* definizione della funzione che stampa a video il
   numero della domanda e la risposta corretta */
void risposte(const char *risposte)                    /* i.: risposte corrette */
{
  /* dichiarazione delle variabili locali alla funzione */
  int num_risp,                                        /* l.: numero risposte */
      i;                                  /* l.: indice */

  num_risp = strlen(risposte);


  /* stampa la riga che indica a quale domanda
     ci si riferisce */
  printf("Domanda\t\t");

  /* ciclo che stampa il numero della domanda
     in colonna */
  for(i = 1;
      i <= num_risp;
      i ++)
  {
    if (i >= 10)
    {
      if (i != num_risp)
      {
        printf("  %d",
                i);
      }
      else
      {
        printf("  %d\n",
                i);
      }
    }
    else
    {
```

```
      if (i != num_risp)
      {
        printf("   %d",
                i);
      }
      else
      {
        printf("   %d\n",
                i);
      }
    }
  }

  /* stampa la riga che indica la risposta corretta
     per ogni domanda */
  printf("Risposta\t");

  /* ciclo che stampa la risposta corretta
     incolonnata con il numero della domanda */
  for(i = 0;
      i < num_risp;
      i ++)
  {
    if(i != num_risp - 1)
    {
      printf("   %c",
              risposte[i]);
    }
    else
    {
      printf("   %c\n",
              risposte[i]);
    }
  }
}


/* definizione della funzione che stampa l'id di
   ogni studente e il suo relativo punteggio
   espresso in percentuale */
void punteggio(const char *r_corrette,              /* i.: risposte corrette */
               const int *id_stud,                  /* i.: id studente */
               char **risposte_stud,              /* i.: risposta studente */
               int *percentuale_stud,             /* o.: perc. risp. cor. */
               const int numero_stud)                   /* l.: numero degli stud. */
{
  /* dichiarazione delle variabili locali alla funzione */
  int num_risp;                                         /* l.: numero risposte */
  num_risp = strlen(r_corrette);

  int i;                                    /* l.: contatore */
```

```
  int ricorrenza;                          /* l.: num. ricor. carattere */

  /* ciclo per calcolare la percentuale di risposte
     corrette di ogni studente */
  for(i = 0;
      i < numero_stud;
      i ++)
  {
    /* chiama la funzione conta_carattere per contare
       quanti carattere uguali sono presenti nella
       stessa posizione */
    ricorrenza = conta_carattere(risposte_stud[i],
                                 r_corrette);

    /* applica la formula matematica per calcolare
       la proporzione a : b = c : d */
    percentuale_stud[i] = (ricorrenza * 100) / num_risp;
  }

  /* chiama la funzione che si occupa di
     stampare il punteggio */
  stampa_punt(id_stud,
              percentuale_stud,
              numero_stud);
}


/* definizione della funzione che conta quanti caratteri
   sono uguali e nella stessa posizione */
int conta_carattere(const char *str1,          /* i.: stringa 1 */
                    const char *str2)          /* i.: stringa 2 */
{
  /* dichiarazione delle variabili locali alla funzione */
  int i,                                   /* l.: indice */
      len;                                 /* l.: lungh. stringhe */

  int c = 0;                               /* o.: contatore ric. */

  len = strlen(str1);

  /* ciclo che conta le ricorrenze dei caratteri
     uguali nella stessa posizione */
  for (i = 0;
       i < len;
       i ++)
  {
    if (str1[i] == str2[i])
      c++;
  }

  return c;
}
```

```
/* definizione della funzione che stampa il
   punteggio calcolato */
void stampa_punt(const int *id_stud,                /* i.: id studenti */
                 const int *percentuale_stud,    /* i.: percentuale studenti */
                     const int numero_stud)          /* i./l.: numero studenti */
{
  /* dichiarazione delle variabili locali alla funzione */
  int i;                                    /* l.: indice */

  /* stampa a video l'id di ogni studente con il
     relativo punteggio */
  printf("\tID\t\tPunteggio (percentuale)\n");
  for(i = 0;
      i < numero_stud;
      i ++)
  {
    printf("\t%d\t\t   %d\n",
           id_stud[i],
           percentuale_stud[i]);
  }
}


/* definizione della funzione che conta da quanti
   studenti è stata sbagliata ogni singola domanda */
void errori(const char *risposte_corrette, /* i.: risposte corrette */
            char **risposte_studenti,          /* i.: risposte studenti */
            const int numero_studenti,                /* l.: numero degli studenti */
            int *numero_errori)                /* o.: numero errori commessi */
{
  /* dichiarazione delle variabili locali alla funzione */
  int i,                                    /* l.: indice */
      j,                                    /* l.: indice */
      numero_domande;                                /* l.: numero delle domande */

  numero_domande = strlen(risposte_corrette);

  /* ciclo che scandisce l'array risposte_correte
     e lo confronta con studenti.risp */
  for(i = 0;
      i < numero_studenti;
      i ++)
  {

    /* ciclo interno che scandisce gli stundenti */
    for (j = 0;
         j < numero_domande;
         j ++)
    {
      if (risposte_studenti[i][j] != risposte_corrette[j])
```

```
      numero_errori[j] ++;
  }
}

/* stampo gli errori nel report d'esame */
printf("Domanda\t\t");

/* ciclo che stampa il numero della domanda
   in colonna */
for(i = 1;
    i <= numero_domande;
    i ++)
{
  if (i >= 10)
  {
    if (i != numero_domande)
    {
      printf("  %d",
             i);
    }
    else
    {
      printf("  %d\n",
             i);
    }
  }
  else
  {
    if (i != numero_domande)
    {
      printf("   %d",
             i);
    }
    else
    {
      printf("   %d\n",
             i);
    }
  }
}

/* stampa la riga che indica da quanti studenti
   è stata sbagliata la risposta di ogni domanda */
printf("Sbagliata da\t");

/* ciclo che stampa il numero di domande sbagliate */
for(i = 0;
    i < numero_domande;
    i ++)
{
  if (numero_errori[i] >= 10)
  {
```

```
      if (i != numero_domande - 1)
      {
        printf("  %d",
                numero_errori[i]);
      }
      else
      {
        printf("  %d\n",
                numero_errori[i]);
      }
    }
    else
    {
      if (i != numero_domande - 1)
      {
        printf("   %d",
                numero_errori[i]);
      }
      else
      {
        printf("   %d\n",
                numero_errori[i]);
      }
    }
  }
}


/* definizione della funzione necessaria al riordinamento degli
   array */
void quicksort(int *a_primario,
                int sx,
                int dx,
                int *a_secondario)
{
  int pivot,
      tmp,
      i,
      j;

  /* crea la tripartizione */
  for (pivot = a_primario [(sx + dx) / 2], i = sx, j = dx;
       (i <= j);
      )
  {
    while (a_primario[i] < pivot)
      i++;
    while (a_primario[j] > pivot)
      j--;
    if (i <= j)
    {
      if (i < j)
```

```
      {
        /* scambio valori */
        tmp = a_primario[i];
          a_primario[i] = a_primario[j];
        a_primario[j] = tmp;

        tmp = a_secondario[i];
          a_secondario[i] = a_secondario[j];
          a_secondario[j] = tmp;
      }
      i++;
      j--;
    }
  }

  /* ordina la prima e la terza parte se contenenti più di un elemento
     chiamando ricorsivamente quicksort */
  if  (sx < j)
    quicksort(a_primario,
                sx,
              j,
                a_secondario);
  if  (i < dx)
    quicksort(a_primario,
                i,
                dx,
                a_secondario);
}
```

Di seguito il codice del Makefile:

```
progetto: progetto.c Makefile
        gcc -ansi -Wall -O progetto.c -o progetto
pulisci:
        rm -f progetto.o
pulisci_tutto:
        rm -f progetto progetto.o
```

N.B. Il codice è copiato e incollato dal file in c. Per questo può differire nell'indentazione.

Testing del programma

Di seguito sono riportati i test effettuati per gli errori più significativi:

Input da file:

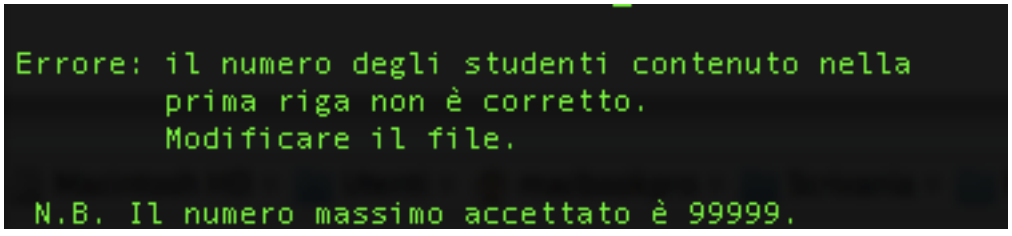
- file vuoto -

Output a video:

Input da file:
c
5
abcde
121 abcdd
122 acdee
123 acdee

- errore prima riga -

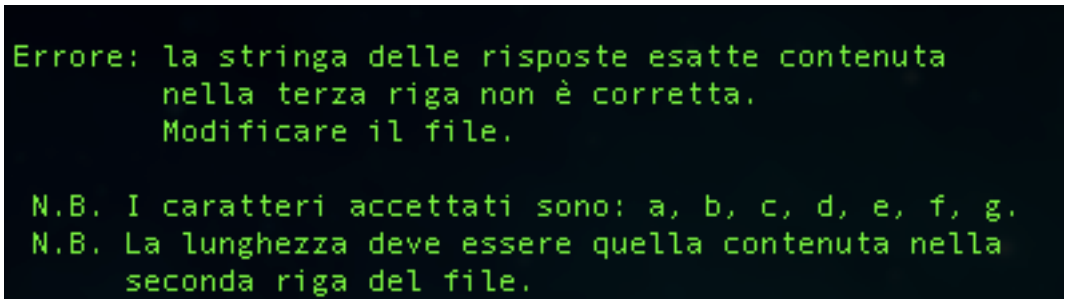
Output a video:



Input da file:
3
5
abcdef
121 abcdd
122 acdee
123 acdee

- stringa terza riga più lunga del previsto -

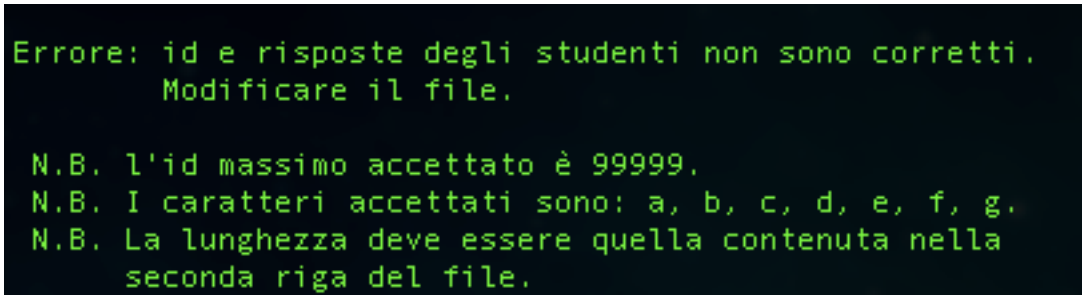
Output a video:



Input da file:
3
5
abcde
121 abcdde
122 acdee
123 acdee

- stringa risposte dello studente della quarta riga troppo lunga -

Output a video:



Input da file:
3
5
abcde
121 abcdd
122 acdee

- meno studenti del previsto -

Output a video:

```
Errore: file terminato prima di acquisire
        tutti gli id e tutte le risposte.
```

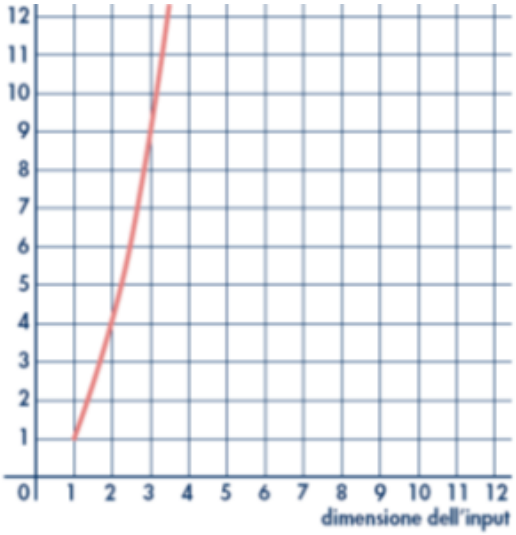
Valutazione della complessità del programma

Per la valutazione della complessità del programma ho realizzato un programmino supplementare che calcola i passi base generando id e percentuali casuali.

Di seguito viene riportato l'output di quest'ultimo che evidenzia la complessità del quicksort da me implementato (e modificato).

```
I passi effettuati dal quicksort per 1 id sono : 13.
I passi effettuati dal quicksort per 10 id sono : 217.
I passi effettuati dal quicksort per 20 id sono : 472.
I passi effettuati dal quicksort per 30 id sono : 833.
```

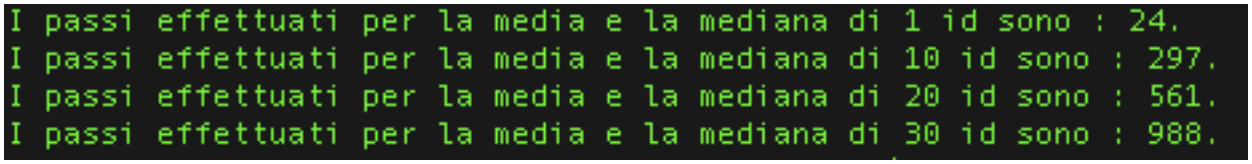
Possiamo quindi notare che il quicksort ha complessità quadratica all'aumentare dell'input (n^2).



Si è scelto il quicksort per l'ordinamento in quanto in media è più efficiente di selectsort, insertsort e bubblesort. Il mergesort è più efficiente del quicksort solo nel caso pessimo.

Ora è necessario calcolare anche la complessità nel calcolo della media e della mediana. Considerando che, come primo passaggio, si ha il riordino dell'array in base alle percentuali, ai passi base del quicksort andranno aggiunti i passi per il calcolo di media e mediana.

Di seguito gli output riportati dal programma.



Possiamo quindi notare che il calcolo della media e della mediana ha complessità lineare che, sommata alla complessità quadratica del quicksort, lo fa diventare di complessità quadratica.