Strutture Dati

Lezione 1
Presentazione del corso
Introduzione

Obiettivi del corso

- Definire formalmente la nozione di algoritmo
- Organizzare le informazioni in strutture dati
- Progettare algoritmi corretti ed efficienti
  - risolvendo il problema il più velocemente possibile
  - usando il minor spazio di memoria possibile

Descrizione del corso

- Analisi e complessità degli algoritmi
- Algoritmi ricorsivi e relazioni di ricorrenza
- Strutture dati elementari
- Algoritmi di ordinamento

Programma

- Introduzione, algoritmi [2 ore]
- Ricorsione, complessità [4 ore]
- Tipo di dati astratto (ADT). Array, matrice [2 ore]
- Stringa, Problema del pattern matching [2 ore]
- Coda, stack, liste [6 ore]
- Alberi [4 ore]
- Code con priorità, heap [2 ore]
- Alberi binari di ricerca [2 ore]
- Grafi [8 ore]

Programma

- Ordinamento per inserzione e per selezione [2 ore]
- Ordinamento rapido [2 ore]
- Ordinamento per fusione [2 ore]
- Ordinamento con heap e con radice [2 ore]
- Esercitazioni [8 ore]

Prerequisiti

- E' richiesta la conoscenza:
  - Analisi matematica
  - Algebra lineare (vettori e matrici)
  - Buone capacità di programmazione in C
- E' quindi preferibile che lo studente abbia seguito e superato con successo l'esame:
  - Programmazione + Laboratorio di Programmazione

Orario

- Lezioni
  - Lunedì dalle 9:00 alle 11:00
  - Mercoledì dalle 9:00 alle 11:00
- Tutoraggio
  - da definire
- Ricevimento
  - martedì 10:00-12:00
  - on demand

Calendario

- Secondo semestre (12 settimane)
  - Inizio: 1 marzo
  - Fine: 31 maggio
  - Prima verifica: 3-13 aprile
  - Seconda verifica: a fine corso
- Appelli d'esame
  - 15 giugno
  - 13 luglio
  - seconda metà settembre

Come funziona…

- La frequenza al corso è obbligatoria
- Durante il corso vengono effettuate 2 prove valutative (con voto in 30esimi): una intermedia ed una finale che coprono circa il 50% del programma ciascuna
- La partecipazione ai compiti di valutazione intermedia e finale non è obbligatoria
- Alla fine del corso verrà consegnato un progetto applicativo

Come funziona…

- Il voto finale è dato dalla media delle due prove di valutazione (si può partecipare alla seconda con voto $\geq$16 nella prima)
- La valutazione del progetto si basa su:
  - funzionalità, compattezza, eleganza ed originalità della soluzione
- Il corso di Strutture Dati è coordinato con quello di Laboratorio di Strutture Dati
- E' necessario aver svolto tutti gli esercizi proposti durante il corso di Laboratorio di Strutture Dati

Coordinate del docente

- Cecilia Di Ruberto
  - Telefono: 070 6758507
  - E-mail: `dirubert@unica.it`
  - Ricevimento: martedì 10:00-12:00

Materiale didattico

- Di base:
  - E. Horowitz, S. Sahni, S. Anderson-Freed, S*trutture Dati in C*, McGraw-Hill, Milano, 1993

    (in alternativa)
  - R. Sedgewick, *Algoritmi in C*, Addison-Wesley Masson, Milano 1990
- Per approfondimento:
  - F. Luccio, *La struttura degli algoritmi*, Bollati-Boringhieri, Torino, 1982 Rogers, Adams
- Verranno distribuite (sul sito del corso) le slides delle lezioni (PPT)

Suggerimenti…

- Seguire attentamente tutte le lezioni
- Quando non si hanno le idee chiare: interrompere, fare domande e chiedere di ripetere
- Ripassare la lezione (ogni volta!)
- Svolgere le esercitazioni di Laboratorio di Strutture Dati (ogni volta!)
- Fare le prove di valutazione
- Sfruttare i ricevimenti e il tutoraggio per risolvere i problemi quando si manifestano, senza accumularli

Introduzione

"Cosa si intende per algoritmo?"

Cos'è un algoritmo

- Dato un problema per arrivare ad un programma che lo risolva dobbiamo:
  - individuare gli input
  - definire gli output
  - trovare un metodo di risoluzione
    - un algoritmo
    - strutture dati
  - codificare l'algoritmo e strutture dati in un linguaggio comprensibile dalla macchina (**C**)
- Un algoritmo è un insieme di regole (non ambigue) che ci permette di calcolare i risultati voluti a partire dai dati di input

Come analizzare un algoritmo

- Correttezza
  - dimostrazione formale
  - ispezione informale
- Utilizzo delle risorse
  - tempo di esecuzione
  - utilizzo della memoria
- Semplicità
  - comprensibilità
  - manutenzione

Problema dell'ordinamento

- Input: $\langle a_1,a_2,...,a_n \rangle$
- Output: $\langle a'_1,a'_2,...,a'_n \rangle$ permutazione di $\langle a_1,a_2,...,a_n \rangle$ tale che $a'_1 \leq a'_2 \leq ... \leq a'_n$.

Ordinamento per selezione

```
for i=0; i<n; i++ {
  Esamina lista[i] fino a lista[n-1] e
  trova il numero intero più piccolo
  lista[min];
  Scambia lista[i] con lista[min];
}
```

Ordinamento per selezione

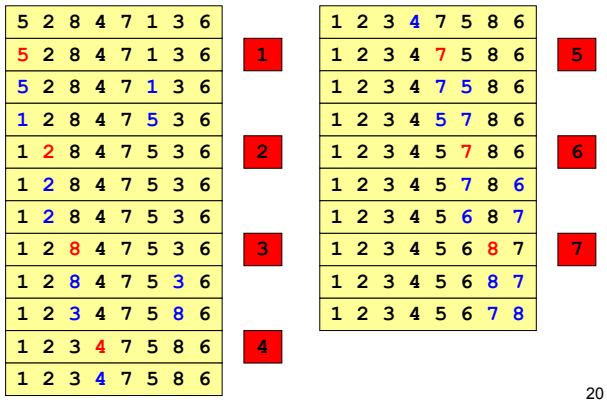
```
void sort(int lista[], int n)
{
 int i,j,min;

 for(i=0; i<n-1; i++)
  {
    min = i;
    for(j=i+1; j<n; j++)
       if(lista[j] < lista[min])
             min = j;
    SWAP(&lista[i], &lista[min]);
  }
}
```

```
void SWAP(int *x, int *y)
{
 int temp;

 temp = *x;
 *x = *y;
 *y = temp;
}
```


Ordinamento per selezione



Analisi: correttezza

- Quando il ciclo `for` esterno completa la sua iterazione per $i=k$, abbiamo che $lista[k] \leq lista[r]$ con $k<r<n$
- Nelle iterazioni successive $i>k$, $lista[0],\ldots, lista[k]$ rimangono invariati
- Dopo l'ultima iterazione del `for` esterno ($i=n-1$), abbiamo $lista[0] \leq lista[1] \leq lista[n-1]$

Ricerca binaria

- Input: $\langle a_1,a_2,\ldots,a_n\rangle$ numeri distinti ordinati
- Output: Verificare se *numric* appartiene all'insieme:
  - i se $numric = a_i$
  - -1 altrimenti

Ricerca binaria

- Supponiamo che *primo* e *ultimo* indichino il primo e l'ultimo numero della lista
- Inizialmente $primo=0$ e $ultimo=n-1$
- Sia $mezzo=(primo+ultimo)/2$ la posizione centrale della lista
- Confrontare *numric* con $lista[mezzo]$
  - $numric<lista[mezzo]$
  - $numric=lista[mezzo]$
  - $numric>lista[mezzo]$

Ricerca binaria

- Se $numric<lista[mezzo]$
  - *numric* va ricercato tra le posizioni 0 e $mezzo-1$
- Se $numric=lista[mezzo]$
  - il programma fornisce il valore *mezzo*
- Se $numric>lista[mezzo]$
  - *numric* va ricercato tra le posizioni $mezzo+1$ e $n-1$

Ricerca binaria

```
while (esistono altri numeri confrontare) {
   mezzo = (primo+ultimo)/2;
   if (numric<lista[mezzo)    ultimo = mezzo-1;
   else if (numric==lista[mezzo]) return(mezzo);
        else  primo = mezzo+1;
}
```


Ricerca binaria

```
int ricbin(int lista[], int numric, int n)
{
 int primo=0, ultimo=n-1, mezzo;

 while(primo<=ultimo) {
   mezzo = (primo + ultimo)/2;
   switch(CONFRONTA(lista[mezzo], numric)){
        case -1:  primo = mezzo + 1;
                  break;
        case 0:   return mezzo;
        case 1:   ultimo = mezzo -1;
       }
 }
 return -1;
}
```

```
int CONFRONTA(int x,int y)
{
 if(x<y) return -1;
 else  if(x==y) return 0;
       else return 1;
}
```


Ricerca binaria

numric=6

0	1	2	3	4	5	6	7	8	
2	4	6	8	10	12	18	20	24	
2	4	6	8	10	12	18	20	24	primo=0 ultimo=8 mezzo=4
2	4	6	8	10	12	18	20	24	primo=0 ultimo=3 mezzo=1
2	4	6	8	10	12	18	20	24	primo=2 ultimo=3 mezzo=2

Trovato! mezzo=2

Ricerca binaria

numric=7

0	1	2	3	4	5	6	7	8	
2	4	6	8	10	12	18	20	24	
2	4	6	8	10	12	18	20	24	primo=0 ultimo=8 mezzo=4
2	4	6	8	10	12	18	20	24	primo=0 ultimo=3 mezzo=1
2	4	6	8	10	12	18	20	24	primo=2 ultimo=3 mezzo=2
2	4	6	8	10	12	18	20	24	primo=3 ultimo=3 mezzo=3

primo=3>ultimo=2!!!!!