Università di Pisa
Dipartimento di Ingegneria dell'informazione

Ingegneria Informatica

Algoritmi & Strutture dati

Raccolta delle diapositive dell'ing.Alfeo

A.A 2019-2020 - Secondo semestre

File realizzato da Gabriele Frassi, disponibile presso la copisteria *OneCent*

Indice

Algoritmi e Strutture Dati

Lezione 1

http://mlpi.ing.unipi.it/alfeo

Antonio Luca alfeo

luca.alfeo@ing.unipi.it

“Ma io so già programmare!”

Fondamenti I	Fondamenti II
Sia dato un array contenente delle **frasi**. Scrivere un programma che prenda in input una **stringa** e restituisca tutte le frasi che la contengono.	Realizzare un motore di ricerca che abbia complessità O(log n) sulle operazioni di lettura.

Algoritmi e **Strutture Dati**

Informazioni

- Lezioni teorico-pratiche
- Esercizi assegnati per casa
- Esame in laboratorio

Pre Requisiti

- Fondamenti I
- Utilizzo compilatore
- Comandi base unix

Sommario

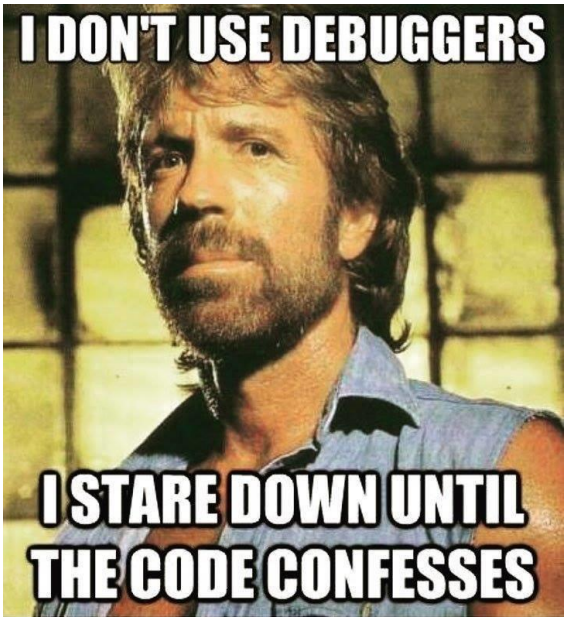
- Debug Triviale
- Gestione Dinamica Input e Liste
- Debug Assistito
- Soluzioni della Standard Template Library
- Gestione Stringhe e Vector

Debugging

“ If **debugging** is the process of **removing** software bugs,

then **programming** must be the process of **putting** them in „

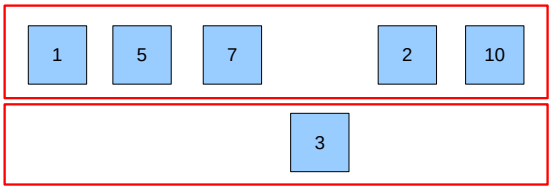
E. Dijkstra



Tecniche

- Testo
- Visuale
- “Debugger” (es **GDB**)
- Compilatore
- Analisi Memoria (**Valgrind**)

Debug Visuale



Debug Visuale

```
1  // stampa con buco
2  void stampaArray( int arr[] , int len , int buco)
3
4
5          1     5     7           2     10
6
7
8
9
10
11
12
13
14 // stampa "segno" in "posizione"
15 void stampaSegno( int posizione , int segno )
16
17
18                        3
19
20
```


Debug Visuale

```
1  // stampa con buco
2  void stampaArray( int arr[] , int len , int buco)
3
4          // PER OGNI ELEMENTO
5
6          // SE SONO IN POSIZIONE buco, SALTO
7
8          // ALTRIMENTI STAMPO ELEMENTO
9
10
11
12
13
14 // stampa "segno" in "posizione"
15 void stampaSegno( int posizione , int segno )
16
17         // SALTO TUTTI GLI ELEMENTI FINO A posizione
18
19         // STAMPO IL SEGNO
20
```

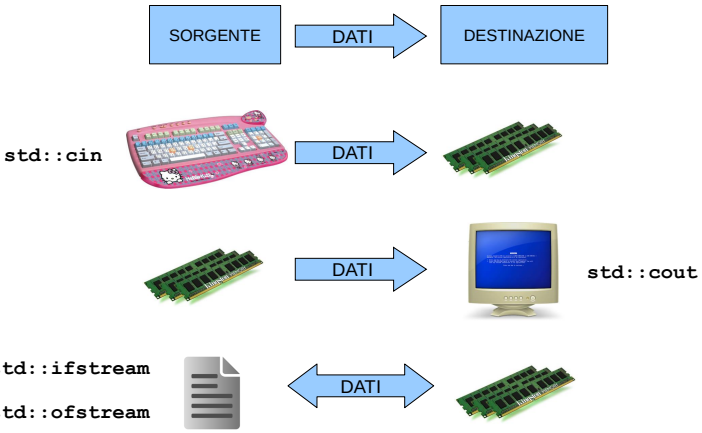

Debug Visuale

```
1  // stampa con buco
2  void stampaArray( int arr[] , int len , int buco)
3  {
4      for( int i=0 ; i < len ; ++i )
5      {
6          if(i==buco)
7              cout << "\t";
8          else
9              cout << arr[i] << "\t" ;
10     }
11     cout << endl;
12 }
13
14 // stampa "segno" in "posizione"
15 void stampaSegno( int posizione , int segno )
16 {
17     for( int i = 0 ; i < posizione ; ++i )
18         cout << "\t";
19     cout << segno << "\n";
20 }
```

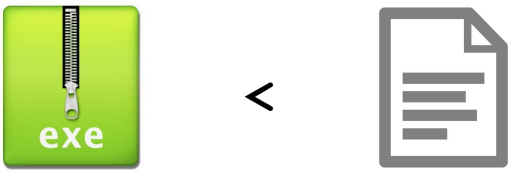

Tipo Accessi

VS

Lettura Input



Redirezione DA File




```
./stlSort < reqFile
```


Lettura Input

```
1
2
3       leggiInput( )
4
5
6         // 1) LEGGO PRIMO VALORE (numero elementi)
7
8
9         // 2) ALLOCAZIONE MEMORIA
10
11
12        // 3) LETTURA CARATTERE PER CARATTERE
13
14
15
16
17
18
```

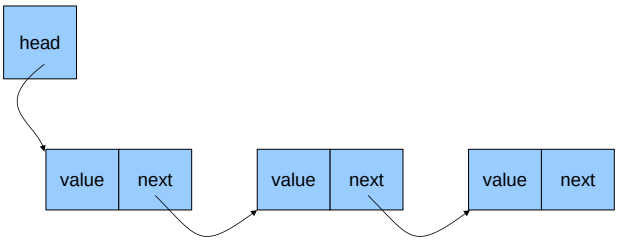

Lettura Input

```
1
2   int * leggiInput( )
3   {
4
5       cin >> len;
6
7       int * arr = new int[len];
8
9
10      for( int i = 0 ; i < len ; ++i )
11          cin >> arr[i];
12
13
14      return arr;
15  }
16
17
18
```


Memoria dinamica

- Quantità di dati **NON** nota a tempo di compilazione
- Quantità di dati **VARIABILE** durante l'esecuzione

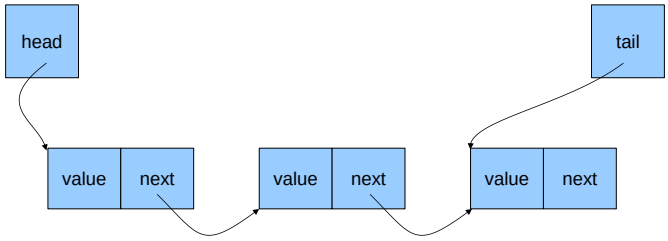
LISTE



Operazioni su Lista

- Inserimento in testa
- **Inserimento in coda**
- Scorrimento
- Estrazione

liste



Lettura su Lista

```
1  Obj * leggiInput()
2  {
3      // LEGGO LUNGHEZZA
4
5
6      // VARIABILI DI APPOGGIO
7
8      // PER TUTTA LA LUNGHEZZA
9      {
10         // LEGGO VALORE
11
12         // CREO E INIZIALIZZO OGGETTO
13
14
15         // AGGIORNO TESTA
16     }
17     // RITORNO TESTA
18 }
```


Lettura su Lista

```
1  Obj * leggiInput()
2  {
3      int value , l;
4      cin >> l;
5
6      Obj * head , * newObj;
7
8      for( int i = 0 ; i < l ; ++i )
9      {
10         cin >> value;
11         newObj = new Obj();
12         newObj->next_ = head;
13         newObj->value_ = value;
14
15         head = newObj;
16     }
17     return head;
18 }
```


Stampa Lista

```
1  void stampaLista( Obj * head )
2  {
3      Obj * pointer = head;
4      while( pointer != NULL )
5      {
6          cout << pointer->value_ << endl ;
7          pointer = pointer->next_;
8      }
9      cout << endl;
10 }
11
12
```

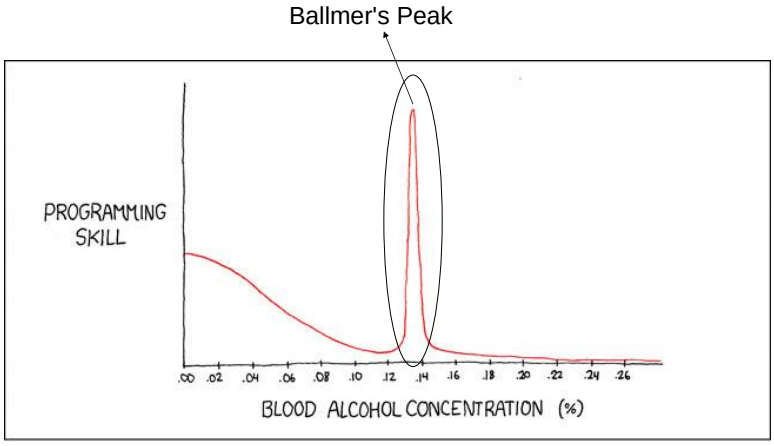

Stampa Lista

```
18 void stampaLista( Obj * head )
19 {
20     Obj * pointer = head;
21     while( pointer != NULL )
22     {
23         cout << pointer->value_ << endl ;
24         pointer = pointer->next_;
25     }
26     cout << endl;
27 }
28
29
```

Lettura su Lista

```
1  Obj * leggiInput()
2  {
3      int value , l;
4      cin >> l;
5
6      Obj * head , * newObj;
7
8      for( int i = 0 ; i < l ; ++i )
9      {
10         cin >> value;
11         newObj = new Obj();
12         newObj->next_ = head;
13         newObj->value_ = value;
14
15         head = newObj;
16     }
17     return head;
18 }
```


Birra!



Fonte: https://xkcd.com/323/


```
kruviser@ilMioComputer:~/Dropbox/lezioni algoritmi/lezione 2$ ./testList < listFile
letto 10
5       1       26      8       12      78      6       2       18      3
3
18
2
6
78
12
8
26
1
5
Segmentation fault (core dumped)
```

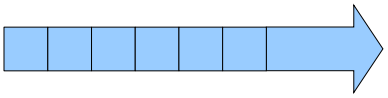

Valgrind

- Babysitter Memoria
- Controlla accessi
- Conta accessi

```
12                                                    valgrind ./eseguibile
8
26
1
5
==3307== Conditional jump or move depends on uninitialised value(s)
==3307==    at 0x8048719: stampaLista(Obj*) (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==    by 0x804883D: main (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==
==3307== Use of uninitialised value of size 4
==3307==    at 0x80486E5: stampaLista(Obj*) (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==    by 0x804883D: main (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==
==3307== Invalid read of size 4
==3307==    at 0x80486E5: stampaLista(Obj*) (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==    by 0x804883D: main (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==  Address 0xffff is not stack'd, malloc'd or (recently) free'd
==3307==
==3307==
==3307== Process terminating with default action of signal 11 (SIGSEGV)
==3307==  Access not within mapped region at address 0xFFFF
==3307==    at 0x80486E5: stampaLista(Obj*) (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==    by 0x804883D: main (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==  If you believe this happened as a result of a stack
==3307==  overflow in your program's main thread (unlikely but
==3307==  possible), you can try to increase the size of the
==3307==  main thread stack using the --main-stacksize= flag.
==3307==  The main thread stack size used in this run was 8388608.
==3307==
==3307== HEAP SUMMARY:
==3307==     in use at exit: 80 bytes in 10 blocks
==3307==   total heap usage: 10 allocs, 0 frees, 80 bytes allocated
==3307==
==3307== LEAK SUMMARY:
==3307==    definitely lost: 0 bytes in 0 blocks
==3307==    indirectly lost: 0 bytes in 0 blocks
==3307==      possibly lost: 0 bytes in 0 blocks
==3307==    still reachable: 80 bytes in 10 blocks
```

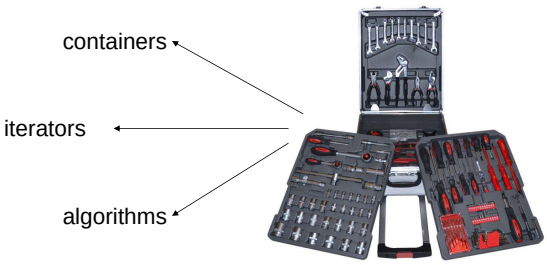
```
12
8       g++ -g -o eseguibile eseguibile.cpp
26                    valgrind ./eseguibile
1
5
==3288== Conditional jump or move depends on uninitialised value(s)
==3288==    at 0x8048719: stampaLista(Obj*) (testList.cpp:21)
==3288==    by 0x804883D: main (testList.cpp:58)
==3288==
==3288== Use of uninitialised value of size 4
==3288==    at 0x80486E5: stampaLista(Obj*) (testList.cpp:23)
==3288==    by 0x804883D: main (testList.cpp:58)
==3288==
==3288== Invalid read of size 4
==3288==    at 0x80486E5: stampaLista(Obj*) (testList.cpp:23)
==3288==    by 0x804883D: main (testList.cpp:58)
==3288==  Address 0xffff is not stack'd, malloc'd or (recently) free'd
==3288==
==3288==
==3288== Process terminating with default action of signal 11 (SIGSEGV)
==3288==  Access not within mapped region at address 0xFFFF
==3288==    at 0x80486E5: stampaLista(Obj*) (testList.cpp:23)
==3288==    by 0x804883D: main (testList.cpp:58)
==3288==  If you believe this happened as a result of a stack
==3288==  overflow in your program's main thread (unlikely but
==3288==  possible), you can try to increase the size of the
==3288==  main thread stack using the --main-stacksize= flag.
==3288==  The main thread stack size used in this run was 8388608.
==3288==
==3288== HEAP SUMMARY:
==3288==     in use at exit: 80 bytes in 10 blocks
==3288==   total heap usage: 10 allocs, 0 frees, 80 bytes allocated
==3288==
==3288== LEAK SUMMARY:
==3288==    definitely lost: 0 bytes in 0 blocks
==3288==    indirectly lost: 0 bytes in 0 blocks
==3288==      possibly lost: 0 bytes in 0 blocks
==3288==    still reachable: 80 bytes in 10 blocks
```


Vettore



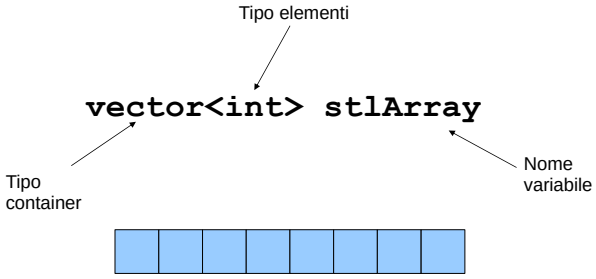
- Struttura dati di dimensione estendibile
- Accesso efficiente
- Algoritmi
- Magari già pronta?!?

Standard Template Library (STL)



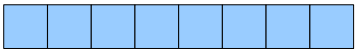
Documentazione:
www.cplusplus.com/reference/stl/

Uso Vector



Uso Vector

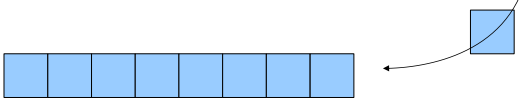
`vector<int> stlArray`



Uso Vector

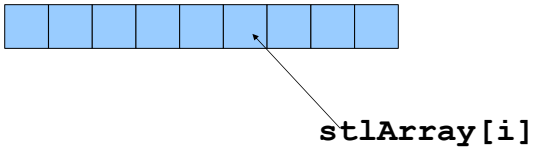
`vector<int> stlArray`

`stlArray.push_back(val)`



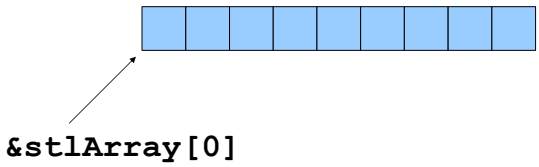
Uso Vector

vector<int> stlArray



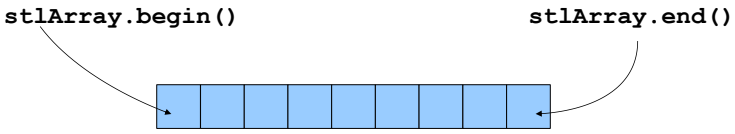
Uso Vector

vector<int> stlArray



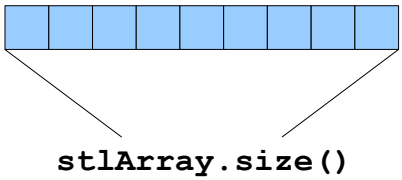
Uso Vector

vector<int> stlArray



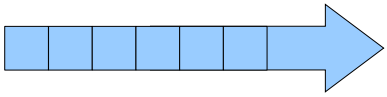
Uso Vector

vector<int> stlArray



Uso Vector

- Dinamico
  - Allocazione dinamica dimensione



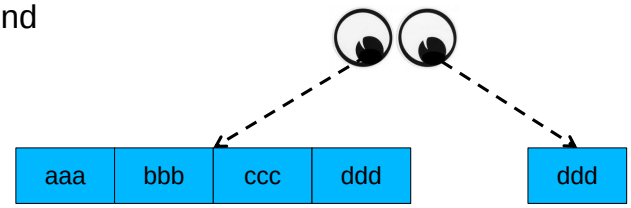
- Contiguo
  - Accesso Random con costo costante
  - Gestione array-like ← **(con prudenza)**

File ⇨ Vector

```
1   #include <vector>
2
3   void leggiInput( std::vector<int> & arr )
4   {
5
6       cin >> len;
7
8       int val;
9       for( int i = 0 ; i < len ; ++i )
10      {
11          cin >> val;
12          arr.push_back(val);
13      }
14
15      return;
16  }
17
18
```

Stringhe

- Creazione
- Concatenazione
- Compare
- Find



Stringhe

```
1  #include <string>
2
3  String parola = "liste";
4
5
6
7
8
9
10
11
12
13
14
```


Stringhe

```
1  #include <string>
2
3  String parola = "liste";
4
5  String frase = "mi piacciono le liste";
6
7
8
9
10
11
12
13
14
```


Stringhe

```
1  #include <string>
2
3  String parola = "liste";
4
5  String frase = "mi piacciono le liste";
6
7  String parola2 = "non ";
8
9  String frase2 = parola2 + frase;
10
11
12
13
14
```


Stringhe

```
1  #include <string>
2
3  String parola = "liste";
4
5  String frase = "mi piacciono le liste";
6
7  String parola2 = "non ";
8
9  String frase2 = parola2 + frase;
10
11 frase.find(parola);
12 // se fallisce -> string::npos
13
14
```


Stringhe

```
1  #include <string>
2
3  String parola = "liste";
4
5  String frase = "mi piacciono le liste";
6
7  String parola2 = "non ";
8
9  String frase2 = parola2 + frase;
10
11 frase.find(parola);
12 // se fallisce -> string::npos
13
14 parola.compare(parola2);
```

http://www.cplusplus.com/reference/string/string/

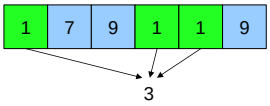
Stringhe

```
1   #include <string>
2   int main ()
3   {
4   string str = "There are two needles in this
5   haystack with needles.";
6
7   string str2 = "needle";
8
9   int found = str.find(str2);
10
11    if (found!=string::npos)
12  cout << "first 'needle' found at: " <<
13  found << '\n';
14  }
```

first 'needle' found at: 14

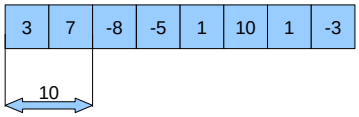
ESERCIZI:

K interi più frequenti

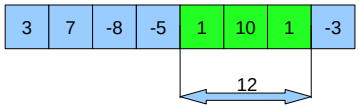


- Input: elementi array , intero k
- Output: Occorrenza valore più frequente

Esercizio: Somma Massima



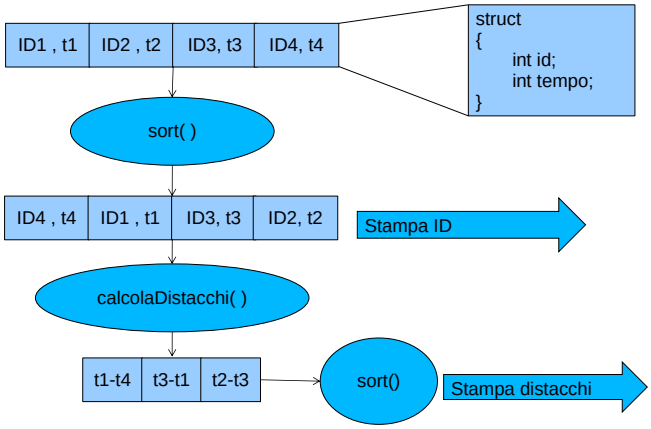
- Input: array
- Output:somma massima di elementi consecutivi
- Esempio



Gara

- Ad una gara partecipano N concorrenti
- Ogni concorrente e' caratterizzato da:
  - Un ID intero
  - Un tempo di arrivo espresso in secondi
- Calcolare:
  - Classifica
  - K distacchi più ampi di utenti **consecutivi**

Gara



Come Esercitarsi

Input: `input.txt`

```
3
1
9
15
```

Output: `output.txt`

```
25
135
yes
```

Input

- Il primo carattere indica il numero di valori da leggere
- Un valore per riga

Output

- Somma dei valori
- Prodotto dei valori
- I valori sono positivi? Rispondere yes o no

Come Esercitarsi

- Input: input.txt Output: output.txt

LETTURA
```
cin >> valore;
```

INPUT
```
./eseguibile < input.txt
```

GENEZIONE OUTPUT
```
cout << uscita;
```

VERIFICA
```
./eseguibile < input.txt | diff - output.txt
```

Algoritmi e Strutture Dati

Lezione 2

http://mlpi.ing.unipi.it/alfeo

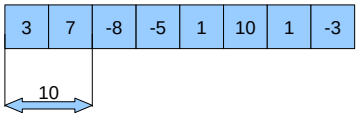
Antonio Luca alfeo

luca.alfeo@ing.unipi.it

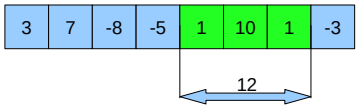
Sommario

- Soluzione esercizi e relativa complessità
- Insertion Sort
- Merge Sort
- Esercizi

Esercizio: Somma Massima



- Input: array
- Output:somma massima di elementi consecutivi
- Esempio



Soluzione 1

```
1  int somme1(int a[] , int size )
2  {
3
4
5
6      for(i=0; i<size; i++)                  // n
7      {
8
9
10
11
12
13
14
15
16
17     }
18     return max;
   }
```


Soluzione 1

```
1  int somme1(int a[] , int size )
2  {
3
4
5
6      for(i=0; i<size; i++)                  // n
7      {
8          for(j=i; j<size; j++)              // n
9          {
10
11
12
13
14
15
16         }
17     }
18     return max;
   }
```


Soluzione 1

```
1  int somme1(int a[] , int size )
2  {
3      int somma;
4      int i,j,k;
5      int max=a[0];
6      for(i=0; i<size; i++)                  // n
7      {
8          for(j=i; j<size; j++)              // n
9          {
10             somma=0;
11             for(k=i; k<=j; k++)            // n
12             {
13                 somma+=a[k];
14             }
15             if(somma > max) max=somma;
16         }
17     }
18     return max;
   }
```

Soluzione 1

```
1  int somme1(int a[] , int size )
2  {
3      int somma;
4      int i,j,k;
5      int max=a[0];
6      for(i=0; i<size; i++)                  // n
7      {
8          for(j=i; j<size; j++)              // n
9          {
10             somma=0;
11             for(k=i; k<=j; k++)            // n
12             {
13                 somma+=a[k];
14             }
15             if(somma > max) max=somma;
16         }
17     }
18     return max;
}
```

$\Theta\left(n^3\right)$

Soluzione 2

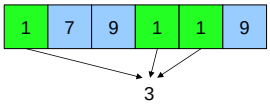
```
1  int somme2(int a[] , int size )
2  {
3      int somma;
4      int i,j;
5      int max=a[0];
6      for(i=0; i<size; i++)
7      {
8          somma=0;
9          for(j=i; j<size; j++)
10         {
11             somma+=a[j];
12             if(somma > max) max=somma;
13         }
14     }
15     return max;
16 }
17
18
```


Soluzione 2

```
1  int somme2(int a[] , int size )
2  {
3      int somma;
4      int i,j;
5      int max=a[0];
6      for(i=0; i<size; i++)                  // n
7      {
8          somma=0;
9          for(j=i; j<size; j++)              // n
10         {
11             somma+=a[j];
12             if(somma > max) max=somma;
13         }
14     }
15     return max;
16 }
17
18
```

$\Theta\left(n^2\right)$

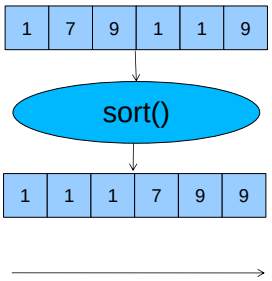
Occorrenza valore più frequenti



- Input: elementi array, intero k
- Output: Occorrenza valore più frequenti

Occorrenza valore più frequenti

```
int main() {
// inserimento -> O(n)

// ordinamento -> O(n lon(n))

// calcolo occorrenze -> O(?)

// stampa

}
```



Occorrenza valore più frequenti

```
int maxOcc(int arr[], int size) {
countMax = 0;
tmpCount = 1;
   for(i=0; i<size-1; i++) {
        if (arr[i]==arr[i+1])
            tmpCount += 1;
        else {
             tmpCount > countMax ? countMax = tmpCount : countMax = countMax;
             tmpCount = 1;
        }
   }
return countMax;
}
```

$\Theta\left(n\right)$

Occorrenza valore più frequenti

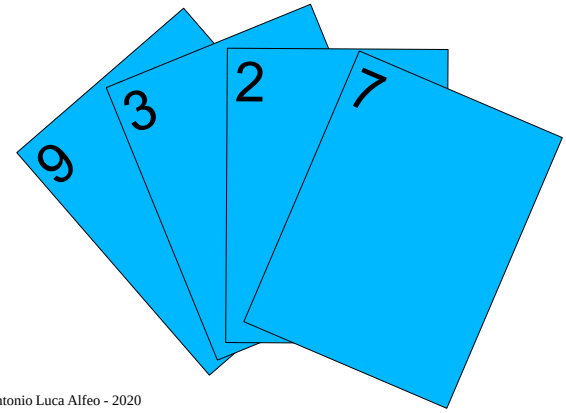
```
int main() {
// inserimento -> O(n)

// ordinamento -> O(n lon(n))

// calcolo occorrenze con maxOcc -> O(n)

// stampa

}
```

$$\Theta(n \log n)$$

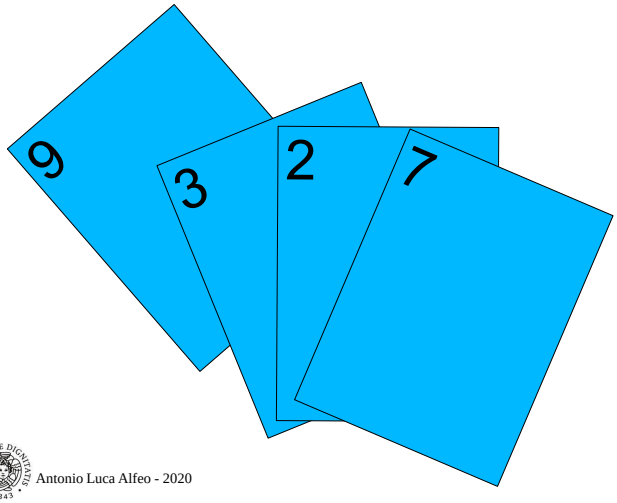
ORDINAMENTO

INSERTION SORT

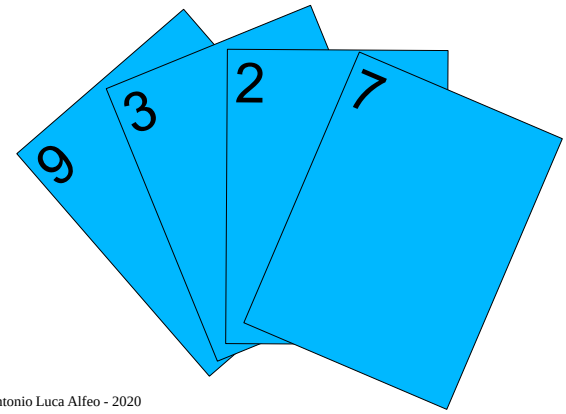
Ordinare Carte da Gioco



Ordinare Carte da Gioco

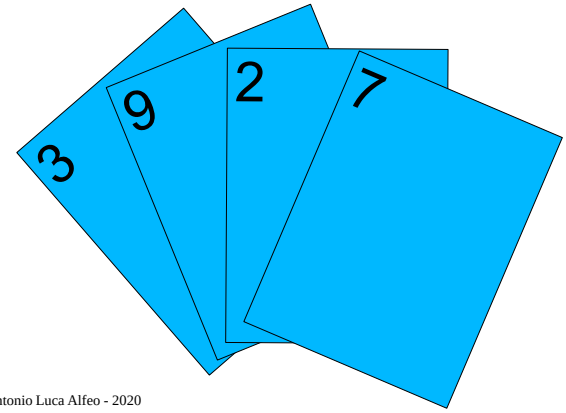


Ordinare Carte da Gioco

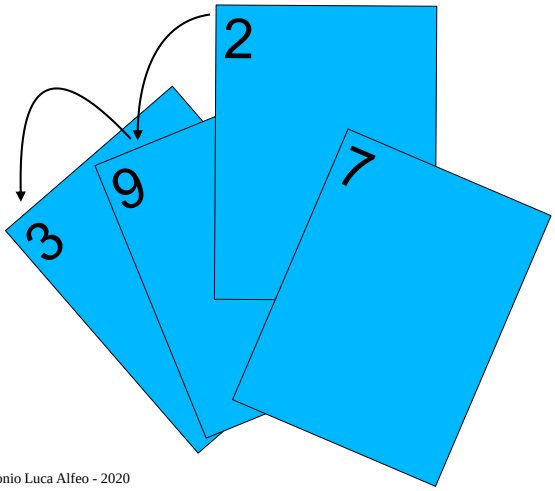


Ordinare Carte da Gioco

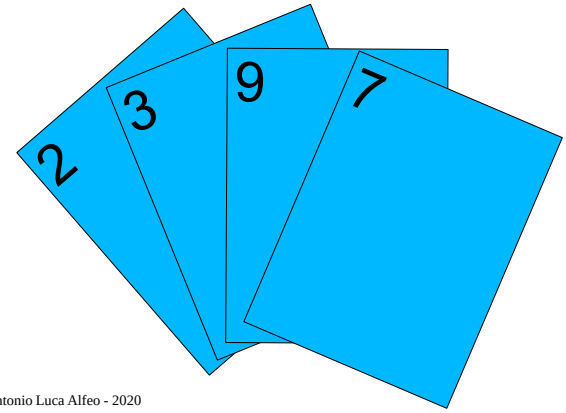
Ordinare Carte da Gioco



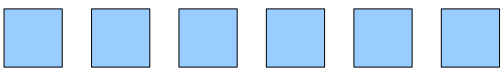
Ordinare Carte da Gioco



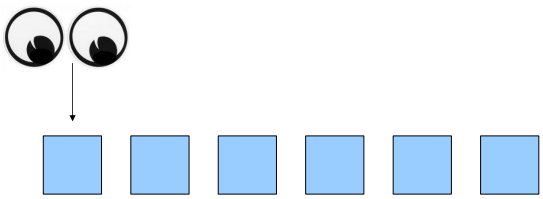
Ordinare Carte da Gioco



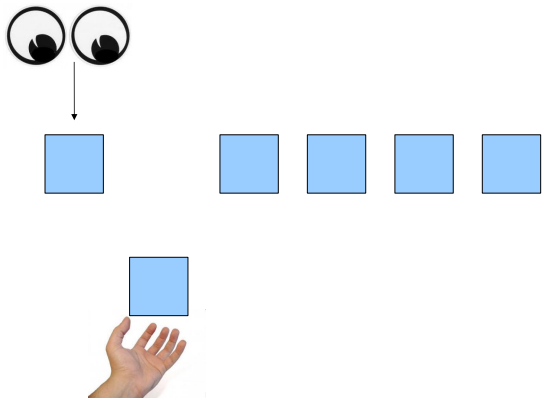
Insertion Sort: rappresentazione



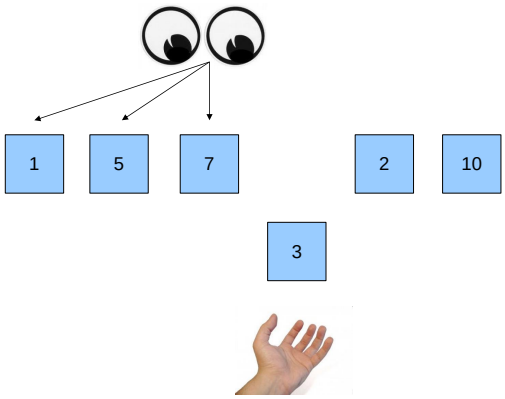
Insertion Sort: rappresentazione



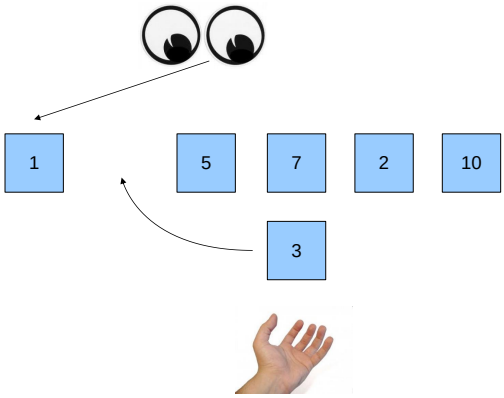
Insertion Sort: rappresentazione



Insertion Sort: rappresentazione



Insertion Sort: rappresentazione



Sorting

```
1  void sortArray( int arr[] , int len )
2  {
3
4
5      for( per ogni elemento della fila )
6      {
7          // inizializzo mano e occhio
8
9
10         while( trovo posizione corretta )
11         {
12             // sposto oggetto
13             // sposto occhio
14         }
15
16         // libero mano
17     }
18 }
```


Sorting

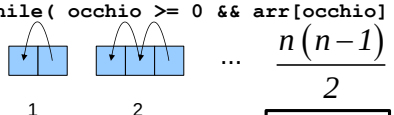
```
1  void sortArray( int arr[] , int len )
2  {
3      int mano = 0;
4      int occhio = 0;
5      for(          )
6      {
7
8
9
10         while(       )
11         {
12
13
14         }
15
16
17     }
18 }
```


Sorting

```
1  void sortArray( int arr[] , int len )
2  {
3      int mano = 0;
4      int occhio = 0;
5      for( int iter = 1 ; iter < len ; ++iter )
6      {
7          mano = arr[iter];
8          occhio = iter-1;
9
10         while( occhio >= 0 && arr[occhio] > mano )
11         {
12             arr[occhio+1] = arr[occhio];
13             --occhio;
14         }
15
16         arr[occhio+1] = mano;
17     }
18 }
```

Analisi InsertionSort

```
1  void sortArray( int arr[] , int len )
2  {
3
4
5      for( int iter = 1 ; iter < len ; ++iter )
6      {
7
8
9
10         while( occhio >= 0 && arr[occhio] > mano )
11         {
12
13
14         }
15
16
17     }
18 }
```



$$\frac{n(n-1)}{2}$$

$$\Theta(n^2)$$

CONFRONTO COMPLESSITA'

STL : Sort()

```
sort( stlArray.begin(),stlArray.end() );
```

restituisce un iteratore

$$\Theta(n \log n)$$

Confronto

Insertion Sort	vs	STL Sort
$\Theta(n^2)$		$\Theta(n \log n)$

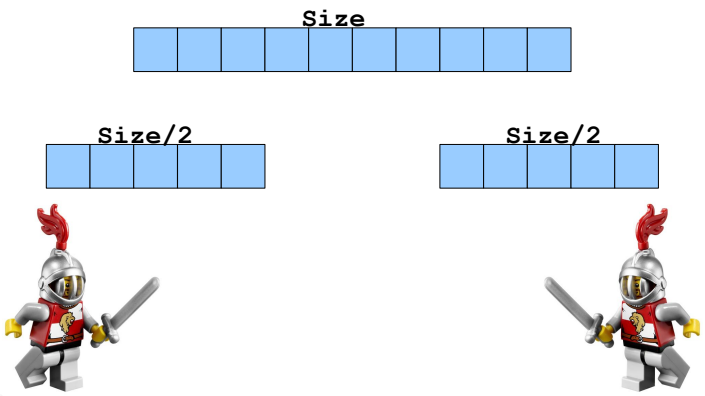
```
time ./stlSort
```


Qualche esperimento per casa…

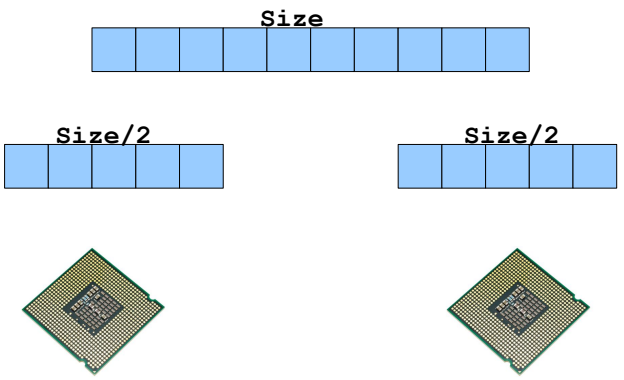
- Casi limite inputs
  - Tutti uguali
  - Ordine crescente (gia' ordinato)
  - Ordine decrescente
- Valori random
  - srand(seed) rand()%maxVal
- Comando time
  - time nomeEseguibile

MERGE SORT

Merge Sort

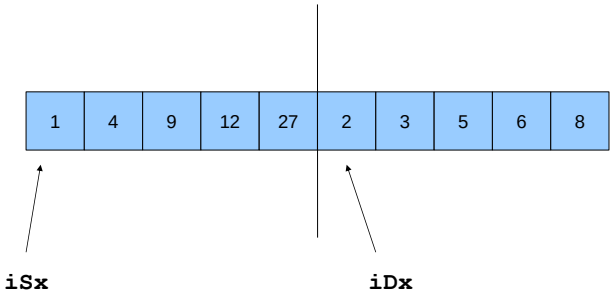


Vantaggi



Unire gli array

Strategia? Divide , Conquer ...& Combine



combine

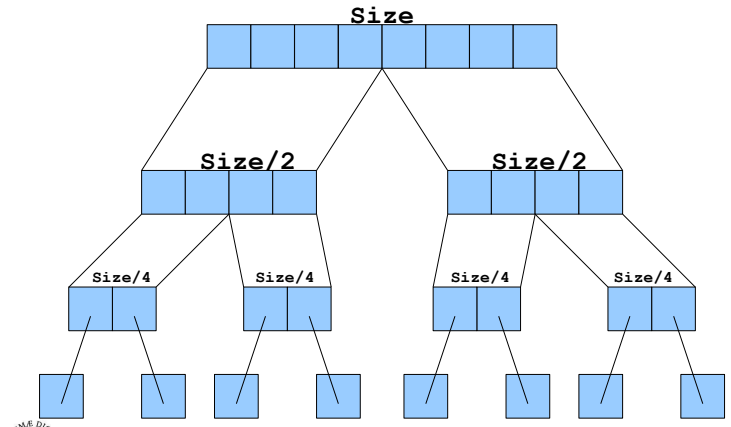
```
1  void combine( int arr[] , int start , int mid , int end )
2  {
3      // init Variabili di stato + buffer appoggio
4
5      while(1)
6      {
7          // se arr[iSx] più piccolo
8          {
9              // Inserisco arr[iSx]
10
11         }
12         // se arr[iDx] più piccolo
13         {
14             // Inserisco arr[iDx]
15
16         }
17     }
18
19
20 }
```


combine

```
1  void combine( int arr[] , int start , int mid , int end )
2  {
3      int iSx = start , iDx = mid; // stato
4      std::vector<int> tempResult; // buffer
5      while(1)
6      {
7          if(arr[iSx] < arr[iDx])
8          {
9              tempResult.push_back(arr[iSx++]);
10             // CONDIZIONE USCITA
11         }
12         else
13         {
14             tempResult.push_back(arr[iDx++]);
15             // CONDIZIONE USCITA
16         }
17     }
18     // GESTISCO ULTIMI
19     // RICOPIO DA BUFFER A ARR
}
```

```
1       2       5       6       8       12      18      26      78
3       6       9       99      100     120     150     168     300
sx 1
sx 2
dx 3
sx 5
dx 6
sx 6
sx 8
dx 9
sx 12
sx 18
sx 26
sx 78
1       2       3       5       6       6       8       9       _ 12
18      26      78      99      100     120     150     168     300
```

divide



Divide , Conquer , Combine

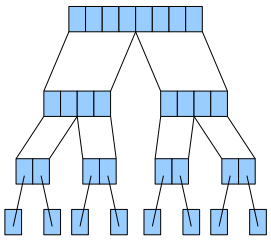
```
1   conquer ( int * arr , int start , int end )
2   {
3       int mid;
4       if( start<end )
5       {
6           mid = (start+end)/2; // DIVIDE
7           conquer( arr , start , mid ); // CONQUER
8           conquer( arr , mid+1 , end ); // CONQUER
9           combine( arr , start , mid+1 , end );
10      }
11  }
12
```


Divide , Conquer , Combine ovvero Split, Sort, Merge

```
1   void mergeSort( int * arr , int start , int end )
2   {
3       int mid;
4       if( start<end )
5       {
6           mid = (start+end)/2; // DIVIDE
7           mergeSort( arr , start , mid ); // CONQUER
8           mergeSort( arr , mid+1 , end ); // CONQUER
9           merge( arr , start , mid+1 , end ); // COMBINE
10
11      }
    }
```


Complessità mergesort

- Elementi n
- Livelli $log(n)+1$
- Costo livello n



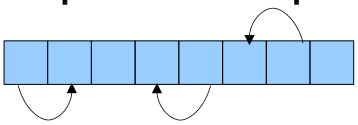
$$n(log(n)+1) \longrightarrow n\,log(n)+n$$

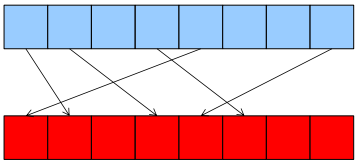
Complessità Mergesort

$$\Theta(n\,log(n))$$

Complessità?

- Tempo di esecuzione: **worst** vs **best** vs **avg**
- Memoria: **in-place** or **not in-place**?





BEST CASE

WORST CASE

	Worst Case	Best Case	Average Case
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Insertion Sort	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$

Esercizio Complessita' Merge e Insertion

- Dato un array di numeri interi, individuare il numero di scambi di elementi effettuati dalle procedure di Insertion Sort e di merge della procedura Merge Sort. Testare nei casi:
  - [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  - [1, 2, 3, 7, 9, 11, 14, 19, 20, 25, 29, 30]
  - [30, 9, 29, 10, 1, 9, 2, 20, 15, 2, 13, 4, 28, 1]

Esempio Insertion Sort

- Input in ordine decrescente

10 , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1


```
1  void sortArray( int arr[] , int l )
2  {
3      // init total e counter
4
5      for( int iter = 1 ; iter < len ; ++iter )
6      {
7
8          counter = 0;
9
10         while( occhio >= 0 && arr[occhio] > mano )
11         {
12             counter++;
13
14         }
15
16         total += counter;
17     }
18 }
```



```
start
         9
10       8       7       6       5       4       3       2       1
         10      8       7       6       5       4       3       2       1
9        10      8       7       6       5       4       3       2       1
=========================================
================  1  ====================
                 8
9        10              7       6       5       4       3       2       1
9                10      7       6       5       4       3       2       1
         9       10      7       6       5       4       3       2       1
8        9       10      7       6       5       4       3       2       1
=========================================
================  2  ====================
                         7
8        9       10              6       5       4       3       2       1
8        9                10      6       5       4       3       2       1
8                9       10      6       5       4       3       2       1
         8       9       10      6       5       4       3       2       1
7        8       9       10      6       5       4       3       2       1
=========================================
================  3  ====================
                                 6
7        8       9       10              5       4       3       2       1
7        8       9                10      5       4       3       2       1
7        8                9       10      5       4       3       2       1
7                8       9       10      5       4       3       2       1
         7       8       9       10      5       4       3       2       1
6        7       8       9       10      5       4       3       2       1
=========================================
================  4  ====================
                                         5
6        7       8       9       10              4       3       2       1
6        7       8       9                10      4       3       2       1
6        7       8                9       10      4       3       2       1
6        7                8       9       10      4       3       2       1
6                7       8       9       10      4       3       2       1
         6       7       8       9       10      4       3       2       1
5        6       7       8       9       10      4       3       2       1
=========================================
================  5  ====================
```


45

Esercizio chiamate ordinamento

Scrivere una funzione che, dati un insieme di interi positivi ricevuti da tastiera, li ordina utilizzando il Merge Sort implementato iterativamente, stampando inoltre a video le informazioni sugli step operativi dell'algoritmo, ovvero:

- le chiamate ricorsive alla procedura MergeSort, con i parametri che le vengono passati;
- le chiamate alla procedura di fusione Merge, con i parametri che le vengono passati.

Dopo ogni chiamata a MergeSort, il programma deve stampare a video il risultato parziale ottenuto, ovvero la sottosequenza ordinata ottenuta.

Ad esempio con gli elementi [3, 1, 2, 5, 4] potrei ottenere un output tipo…


```
Chiamo MergeSort(A,1,5)
   Chiamo MergeSort(A,1,3)
      Chiamo MergeSort(A,1,2)
         Chiamo MergeSort(A,1,1)
         -Sequenza ordinata: 3
         Chiamo MergeSort(A,2,2)
         -Sequenza ordinata: 1
         Chiamo Merge(A,1,1,2)
      -Sequenza ordinata: 1,3
      Chiamo MergeSort(A,3,3)
      -Sequenza ordinata: 2
      Chiamo Merge(A,1,2,3)
   -Sequenza ordinata: 1,2,3
   Chiamo MergeSort(A,4,5)
      Chiamo MergeSort(A,4,4)
      -Sequenza ordinata: 5
       Chiamo MergeSort(A,5,5)
       -Sequenza ordinata: 4
       Chiamo Merge(A,4,4,5)
    -Sequenza ordinata: 4,5
    Chiamo Merge(A,1,3,5)
 -Sequenza ordinata: 1,2,3,4,5
```

Algoritmi e Strutture Dati

Lezione 3

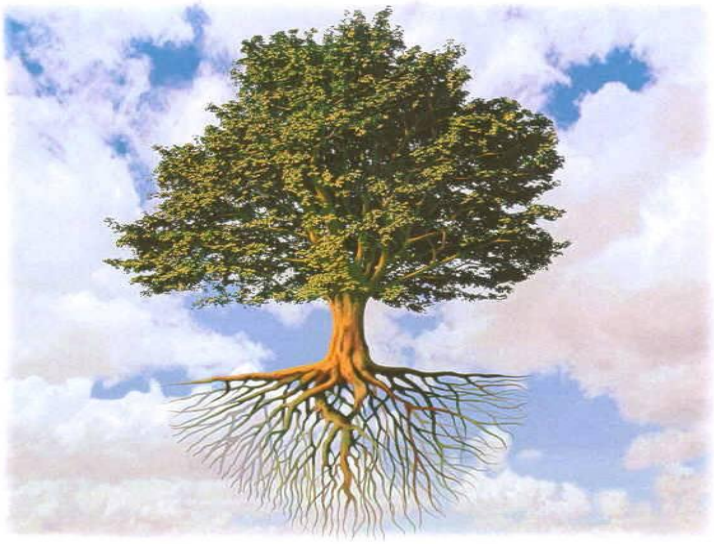
http://mlpi.ing.unipi.it/alfeo

Antonio Luca alfeo

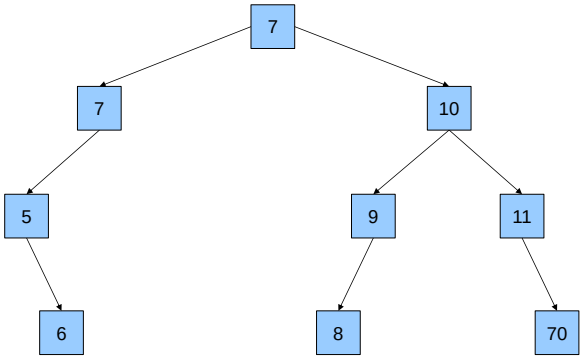
luca.alfeo@ing.unipi.it

Sommario

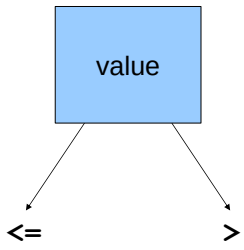
- Alberi Binari di Ricerca
- Progettazione e Inizializzazione
- Esempi Funzioni con Alberi Binari di Ricerca
- Alberi binari con etichette complesse
- Esercizi



Alberi Binari



Alberi Binari di Ricerca



binTree

```
1  struct Node
2  {
3      int value;
4      Node * left;
5      Node * right;
6
7      Node(int val):
8          value(val) , left(NULL) , right(NULL) {}
9  };
10
11
12
13
14
15
16
17
18
19
20
```

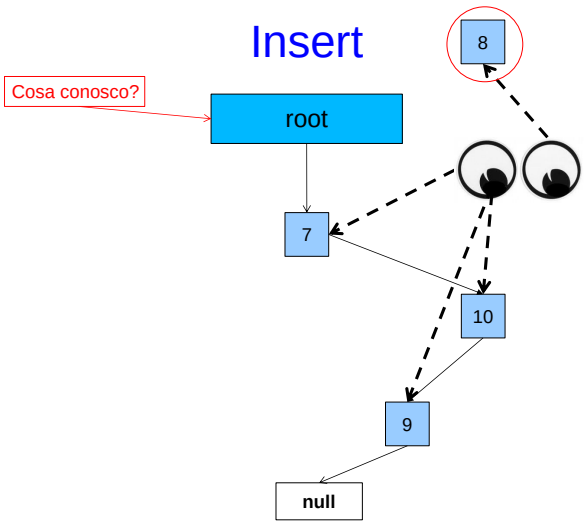
N.B. = inizializzare i puntatori a NULL

binTree

```
1  struct Node
2  {
3      int value;
4      Node * left;
5      Node * right;
6
7      Node(int val):
8          value(val) , left(NULL) , right(NULL) {}
9  };
10
11 class BinTree
12 {
13     Node * root_;
14
15 public:
16
17     BinTree() { root_ = NULL; }
18
19     Node * getRoot() { return root_; }
20 }
```

N.B. = inizializzare i puntatori a NULL

Insert



Insert

```
1  void insert( int val )
2  {
3      // inizializzo nuovo elemento
4      // inizializzo variabili appoggio
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```


Insert

```
1  void insert( int val )
2  {
3      // inizializzo nuovo elemento
4      // inizializzo variabili appoggio
5
6
7      // finchè non arrivo ad una foglia
8      {
9          // aggiorno variabili
10         // se <=
11             // vado a sinistra
12         // altrimenti
13             // vado a destra
14     }
15
16
17
18
19
20
21 }
```


Insert

```
1  void insert( int val )
2  {
3      // inizializzo nuovo elemento
4      // inizializzo variabili appoggio
5
6
7      // finchè non arrivo una foglia
8      {
9          // aggiorno variabili
10         // se <=
11             // vado a sinistra
12         // altrimenti
13             // vado a destra
14     }
15     // se albero vuoto
16         // aggiorno radice
17
18     // decido se diventare figlio left o right
19
20
21 }
```

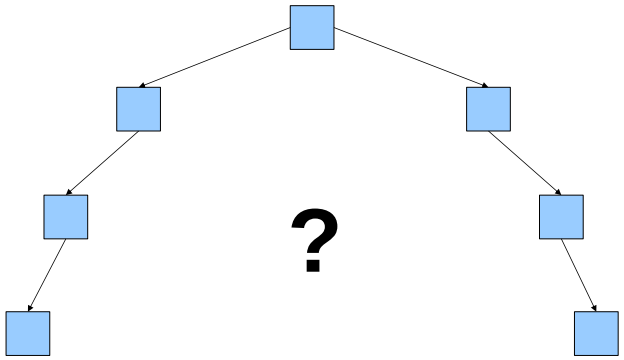
INIZIALIZZAZIONE

INDIVIDUO
POSIZIONE

INSERIMENTO

Esempio: trova Min e Max

min & MAX



Min & Max

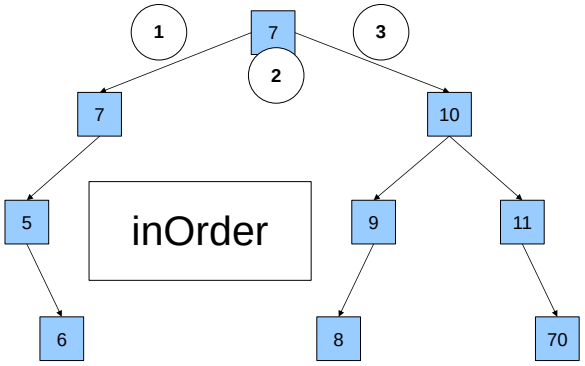
```
1  Node * min()
2  {
3      Node * temp = root_;
4      while( temp->left != NULL )
5          temp = temp->left;
6      return temp;
7  }
8
9
10
11
12
13
14
15
16
17
18
19
20
```


Min & Max

```
1  Node * min()
2  {
3      Node * temp = root_;
4      while( temp->left != NULL )
5          temp = temp->left;
6      return temp;
7  }
8
9
10
11 Node * max()
12 {
13     Node * temp = root_;
14     while( temp->right != NULL )
15         temp = temp->right;
16     return temp;
17 }
18
19
20
```


Esempio: visita l'albero

Visita l'albero



In-Order

```
1
2
3
4   void inOrder( Node * tree )
5   {
6       // se l'albero non e' terminato
7       {
8
9
10
11
12
13
14
15      }
16  }
17
18
19
20
```


In-Order

```
1
2
3
4   void inOrder( Node * tree )
5   {
6       // se l'albero non e' terminato
7       {
8
9           // visito verso left
10
11          // stampo questo valore
12
13          // visito verso right
14
15      }
16  }
17
18
19
20
```

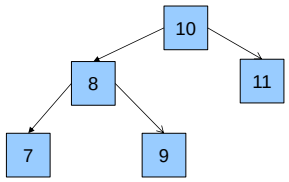

In-Order

```
1
2
3
4   void inOrder( Node * tree )
5   {
6       if(tree!=NULL)
7       {
8
9           inOrder(tree->left);
10
11          cout << tree->value << "\t";
12
13          inOrder(tree->right);
14
15      }
16  }
17
18
19
20
```


Sort vs BinTree

AVG CASE

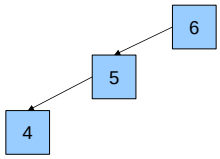
- Albero alto: **log(n)**
- Inserimento: **n * log(n)**
- Sort/visita: **n**



$$\Theta(n \cdot \log n)$$

WORST CASE

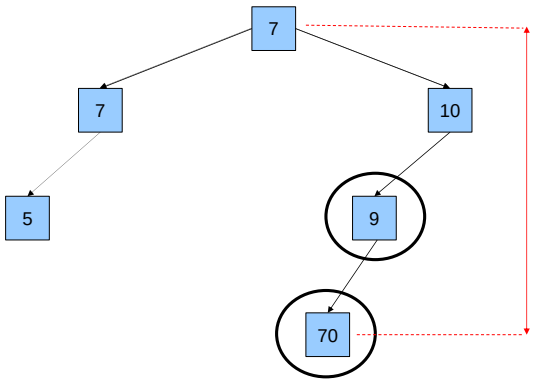
- Albero alto: **n**
- Inserimento: **n * n**
- Sort/visita: **n**



$$\Theta(n^2)$$

Esempio: calcolare altezza albero

Altezza albero



Altezza albero

```
1  int height( Node * tree )
2  {
3      int hLeft;
4      int hRight;
5
6      if( tree == NULL )
7          return 0;
8
9
10
11
12
13
14
15
16
17
18 }
19
20
```


Altezza albero

```
1  int height( Node * tree )
2  {
3      int hLeft;
4      int hRight;
5
6      if( tree == NULL )
7          return 0;
8
9
10     hLeft = height(tree->left);
11
12     hRight = height(tree->right);
13
14
15
16
17 }
18
19
20
```

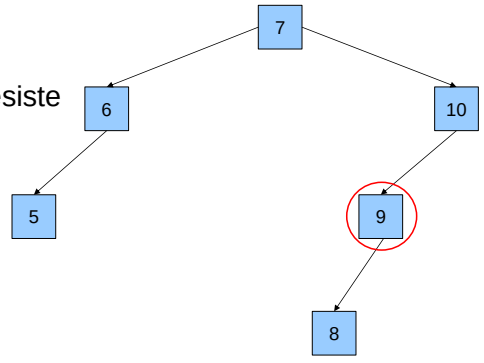

Altezza albero

```
1  int height( Node * tree )
2  {
3      int hLeft;
4      int hRight;
5
6      if( tree == NULL )
7          return 0;
8
9
10     hLeft = height(tree->left);
11
12     hRight = height(tree->right);
13
14
15     return 1 +  max(hLeft,hRight);
16 }
17
18
19
20
```


Esempio: trova un elemento

Trova elemento

- Dato
  - Un albero binario con valori distinti
  - Un valore K
- Trovare
  - Se il valore esiste



Search

```
1  bool search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return false;
5
6
7
8
9
10
11
12
13
14
15
16
17
18 }
19
20
```

Search

```
1  bool search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return false;
5
6     bool found;
7
8      if( tree->value == val )
9          return true;
10
11
12
13
14
15
16
17
18
19 }
20
```


Search

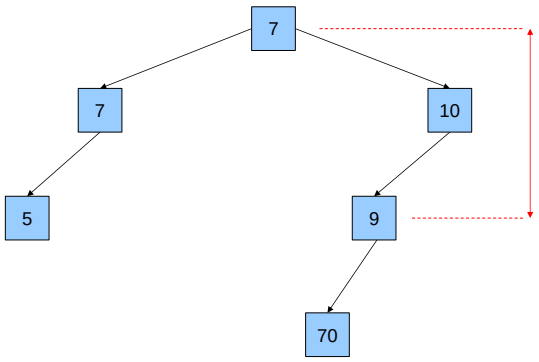
```
1  bool search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return false;
5
6     bool found;
7
8      if( tree->value == val )
9          return true;
10
11
12     else if( val <= tree->value )
13        found = search( tree->left , val );   INDIVIDUO
14     else                                     DIREZIONE
15        found = search( tree->right , val );
16
17
18    return found;
19 }
20
```


Esempio: altezza di un elemento

Trovare altezza chiave K

- Input
  - Una sequenza di N interi positivi
  - Chiave K
- Output
  - L'altezza della chiave K dentro l'albero (se esiste)

K=9



Altezza elemento

```
1  int search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return 0;
5
6      int cont = 0;
7      if( tree->value == val )
8          return 1;
9
10
11
12
13
14
15
16
17
18
19 }
20
```

Altezza elemento

```
1  int search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return 0;
5
6      int cont = 0;
7      if( tree->value == val )
8          return 1;
9
10     else if( val <= tree->value )
11         cont = search( tree->left , val );
12     else
13         cont = search( tree->right , val );
14
15
16
17
18
19 }
20
```


Altezza elemento

```
1  int search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return 0;
5
6      int cont = 0;
7      if( tree->value == val )
8          return 1;
9
10     else if( val <= tree->value )
11         cont = search( tree->left , val );
12     else
13         cont = search( tree->right , val );
14
15     if( cont != 0 )
16         return cont+1;
17
18     else return 0;
19 }
20
```

SEARCH

HEIGHT

ESERCIZI

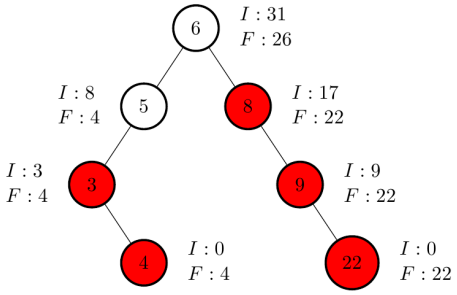
Esercizio: Somma Nodi

- Input:
  - Un intero N
  - N interi
- Operazioni:
  - Inserire gli N interi in un albero binario di ricerca
  - Per ogni nodo *u*, calcolare *I(u)* e *F(u)* (vedi slide seguente)
- Output:
  - Stampare le etichette dei nodi tali che *I(u) <= F(u)*

Somma Nodi

- ***I(u)*:** somma delle chiavi dei nodi interni del sottoalbero radicato in *u*, compresa la radice se e solo se questa non è anche foglia
- ***F(u)*:** somma delle chiavi delle foglie del sottoalbero radicato in *u*

STAMPARE ETICETTE DEI NODI IN CUI:
I(u) <= F(u)



Calcolo I(u) e F(u)

- Devo visitare tutto l'albero.
- I valori di I(u) e F(u) di un nodo padre, dipendono dagli stessi valori calcolati per i nodi figli.
- Di quali nodi posso calcolare I(u) e F(u) "al volo"?

- **Suggerimento:** come facevamo a calcolare l'altezza di un nodo? (relazione padre/figli)

Esercizio: Altezza Figli

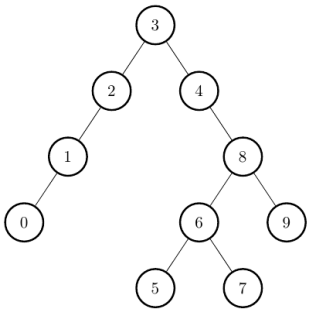
Input:

- N interi da inserire in un albero binario di ricerca

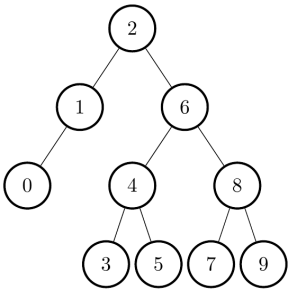
Verificare che :

- Per ciascun nodo, l'altezza dei suoi sottoalberi sinistro e destro deve differire al massimo di uno

Outcome dell'esercizio Altezza figli



no



ok


```
1  bool isOk( Node * tree, int & maxH )
2  {
3     // Controllo se ho raggiunto una foglia
4        return true;
5
6     // Controllo i figli sinistro e destro
7     bool propL = isOk(tree->left,hl);
8     bool propR = isOk(tree->right,hr);
9
10    // ottengo l'altezza del nodo corrente
11    // .. il massimo tra quella sx e dx
12
13    // se la proprietà e' verificata da:
14    // nodo corrente, nodo sx, nodo dx
15       return true;
16    else
17       return false;
18 }
```

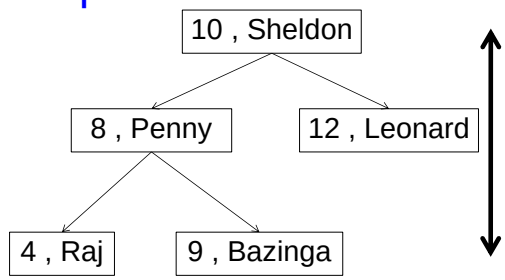
Esercizio: Albero Binario a etichette complesse

- Input:
  - Un intero **N**
  - Un intero **H**
  - N coppie **[intero,stringa]**
- Operazioni:
  - Inserire le N coppie in un albero binario di ricerca (usando il valore intero come chiave)
- Output:
  - stringhe che si trovano in nodi ad altezza H, stampate in ordine lessicografico

Albero Binario a etichette complesse

input

```
5
2
10
Sheldon
12
Leonard
8
Penny
4
Raj
9
Bazinga
```



output

```
Bazinga
Raj
```


Analisi Traccia

- Input:
  - Un intero N
  - Un intero H
  - N coppie [intero,stringa]
- Operazioni:
  - Inserire le N coppie in un albero binario di ricerca
- Output:
  - stringhe che si trovano in nodi ad altezza H, stampate in ordine lessicografico

Analisi Traccia

Implementare struttura dati che supporti
- Albero binario
- Etichette multi valore

```
struct node
{
        int key;
        string str;
        struct node* right;
        struct node* left;
};
```

Funzioni
- Insert su albero binario → insert()
- Trovare nodi ad altezza H → visita+altezza
- Sort su string → sort+compare

Esercizio: nodi “concordi” e “discordi”

Si consideri un sistema di memorizzazione che legga una sequenza di N interi unici e non negativi e li inserisca dentro un albero binario di ricerca (ABR). Si definiscono:

- Concordi i nodi c caratterizzati da altezza del nodo pari (dispari) e etichetta pari (dispari).
- Discordi i nodi d con altezza pari e etichetta dispari, o viceversa.

L’ altezza del nodo x corrisponde al numero di nodi compresi tra x e la radice dell’albero, x escluso. La radice ha altezza 0. Scrivere un programma che:

- legga da tastiera N, il numero di interi da memorizzare nell’albero;
- legga da tastiera N interi, ovvero le etichette dei nodi da inserire nell’ABR. I valori devono essere inseriti nello stesso ordine con cui vengono letti (le etichette ≤ vanno inserite a sinistra);
- stampi la differenza tra la sommatoria S delle etichette dei nodi discordi e la stessa dei nodi concordi

$$S = \sum label(d) - \sum label(c)$$

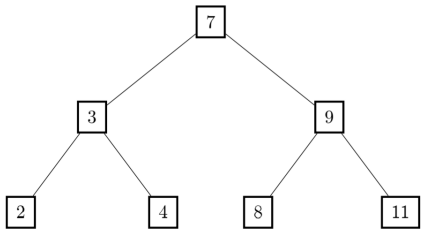
Outcome: “concordi” e “discordi”

- L’input è formattato nel seguente modo: la prima riga contiene l’intero N. Seguono N righe contenenti un’etichetta ciascuna.
- L’output contiene la soluzione su un unica riga.

Esempio

Input

```
7
7
3
9
2
4
8
11
```



Output

```
-8
```


Possibile Soluzione: nodi “concordi” e “discordi”

```
#include <iostream>      // std::cout
#include <algorithm>     // std::sort
#include <vector>        // std::vector
#include <fstream>
#include <math.h>        /* floor */
#include <stdlib.h>
#include <cmath>         /* pow  */

using namespace std;


struct Node
{
    int value;
    Node * left;
    Node * right;

    Node( int i ): value(i) , left(NULL) , right(NULL) {}
};
```



```
class BinTree
{
    Node * root_;
public:
    BinTree() { root_ = NULL ; }

    Node * getRoot() { return root_; cout << "getRoot" << endl;}

    void insert( int i )
    {
        Node * node = new Node(i);

        Node * pre = NULL;
        Node * post = root_;
        while( post != NULL)
        {
            pre = post;
            if( i <= post->value )
            {
                post = post->left;
            }
            else
            {
                post = post->right;
            }
        }

        if( pre == NULL )
            root_ = node;
        else if( i <= pre->value )
        {
            pre->left = node;
        }
        else
        {
            pre->right = node;
        }
        return;
    }
};
```



```
int layerTree( Node * tree, int altezza)
{
    // Nodo non trovato
    if( tree == NULL)  {
       return 0;
    }

    if (altezza++%2 != tree->value%2)  {
          //cout<<tree->value<<" : "<<altezza-1<<" discorde"<<endl;
          return layerTree( tree->left , altezza) + layerTree( tree->right , altezza) + tree->value;
    }
    else   {
          //cout<<tree->value<<" : "<<altezza-1<<" concorde"<<endl;
          return layerTree( tree->left , altezza) + layerTree( tree->right , altezza) - tree->value;
    }
}


int main()
{
    int N ;
    int x ;
    BinTree albero ;

    cin >> N ;

    // riempio l' albero
    for(int i=0 ; i<N ; ++i )   {
        cin >> x;
        albero.insert(x);
    }

    cout<<layerTree(albero.getRoot(), 0)<<endl;
}
```


Esercizio: nodi “completi”

Si consideri un sistema di memorizzazione che legga una sequenza di N interi unici e non negativi e li inserisca dentro un albero binario di ricerca (ABR). Si definiscono completi i nodi c aventi due nodi figli, incompleti i nodi i restanti. Scrivere un programma che:

- legga da tastiera N, il numero di interi da memorizzare nell’albero;
- legga da tastiera N interi, ovvero le etichette dei nodi da inserire nell’ABR. I valori devono essere inseriti nello stesso ordine con cui vengono letti (le etichette ≤ vanno inserite a sinistra);
- stampi la differenza tra la sommatoria S delle etichette dei nodi completi e la stessa dei nodi incompleti

$$S = \sum label(c) - \sum label(i)$$

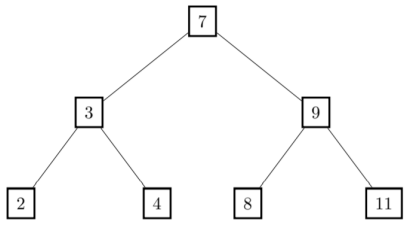
Outcome: “completi”

- L’input è formattato nel seguente modo: la prima riga contiene l’intero N. Seguono N righe contenenti un’etichetta ciascuna.
- L’output contiene la soluzione su un unica riga

Esempio

Input

```
7
7
3
9
2
4
8
11
```



Output

```
-6
```


Possibile soluzione: nodi “completi”

```
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector
#include <fstream>
#include <math.h>       /* floor */
#include <stdlib.h>
#include <cmath>        /* pow  */

using namespace std;


struct Node
{
    int value;
    Node * left;
    Node * right;

    Node( int i ): value(i) , left(NULL) , right(NULL) {}
};
```



```
class BinTree
{
    Node * root_;
public:
    BinTree() { root_ = NULL ; }

    Node * getRoot() { return root_; cout << "getRoot" << endl;}

    void insert( int i )
    {
        Node * node = new Node(i);

        Node * pre = NULL;
        Node * post = root_;
        while( post != NULL)
        {
            pre = post;
            if( i <= post->value )
            {
                post = post->left;
            }
            else
            {
                post = post->right;
            }
        }

        if( pre == NULL )
            root_ = node;
        else if( i <= pre->value )
        {
            pre->left = node;
        }
        else
        {
            pre->right = node;
        }
        return;
    }
};
```



```
int complete( Node * tree)
{
    // Nodo non trovato
    if( tree == NULL)  {
       return 0;
    }

    if (tree->left != NULL && tree->right != NULL)  {
        return complete( tree->left ) + complete( tree->right ) + tree->value;
    }
    else   {
        return complete( tree->left ) + complete( tree->right ) - tree->value;
    }
}


int main()
{
    int N ;
    int x ;
    BinTree albero ;

    cin >> N ;

    // riempio l' albero
    for(int i=0 ; i<N ; ++i )   {
        cin >> x;
        albero.insert(x);
    }

    cout<<complete(albero.getRoot())<<endl;
}
```

Algoritmi e Strutture Dati

Lezione 4

http://mlpi.ing.unipi.it/alfeo

Antonio Luca alfeo

luca.alfeo@ing.unipi.it

Sommario

- Esercizi ultimo laboratorio
- Heap
- Ordinamento tramite Heap
- Hashing
- Hashing e tipi di input
- Esercizi

Esercizio: nodi “concordi” e “discordi”

Si consideri un sistema di memorizzazione che legga una sequenza di N interi unici e non negativi e li inserisca dentro un albero binario di ricerca (ABR). Si definiscono:

- Concordi i nodi c caratterizzati da altezza del nodo pari (dispari) e etichetta pari (dispari).
- Discordi i nodi d con altezza pari e etichetta dispari, o viceversa.

L’ altezza del nodo x corrisponde al numero di nodi compresi tra x e la radice dell’albero, x escluso. La radice ha altezza 0. Scrivere un programma che:

- legga da tastiera N, il numero di interi da memorizzare nell’albero;
- legga da tastiera N interi, ovvero le etichette dei nodi da inserire nell’ABR. I valori devono essere inseriti nello stesso ordine con cui vengono letti (le etichette ≤ vanno inserite a sinistra);
- stampi la differenza tra la sommatoria S delle etichette dei nodi discordi e la stessa dei nodi concordi

$$S = \sum label(d) - \sum label(c)$$

Possibile Soluzione: nodi “concordi” e “discordi”

```
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector
#include <fstream>
#include <math.h>       /* floor */
#include <stdlib.h>
#include <cmath>        /* pow  */

using namespace std;


struct Node
{
    int value;
    Node * left;
    Node * right;

    Node( int i ): value(i) , left(NULL) , right(NULL) {}
};
```



```
class BinTree
{
    Node * root_;
public:
    BinTree() { root_ = NULL ; }

    Node * getRoot() { return root_; cout << "getRoot" << endl;}

    void insert( int i )
    {
        Node * node = new Node(i);

        Node * pre = NULL;
        Node * post = root_;
        while( post != NULL)
        {
            pre = post;
            if( i <= post->value )
            {
                post = post->left;
            }
            else
            {
                post = post->right;
            }
        }

        if( pre == NULL )
            root_ = node;
        else if( i <= pre->value )
        {
            pre->left = node;
        }
        else
        {
            pre->right = node;
        }
        return;
    }
};
```



```
int layerTree( Node * tree, int altezza)
{
    // Nodo non trovato
    if( tree == NULL)  {
       return 0;
    }

    if (altezza++%2 != tree->value%2)  {
        //cout<<tree->value<<" : "<<altezza-1<<" discorde"<<endl;
        return layerTree( tree->left , altezza) + layerTree( tree->right , altezza) + tree->value;
    }
    else   {
        //cout<<tree->value<<" : "<<altezza-1<<" concorde"<<endl;
        return layerTree( tree->left , altezza) + layerTree( tree->right , altezza) - tree->value;
    }
}


int main()
{
    int N ;
    int x ;
    BinTree albero ;

    cin >> N ;

    // riempio l' albero
    for(int i=0 ; i<N ; ++i )   {
        cin >> x;
        albero.insert(x);
    }

    cout<<layerTree(albero.getRoot(), 0)<<endl;
}
```

Esercizio: nodi “completi”

Si consideri un sistema di memorizzazione che legga una sequenza di N interi unici e non negativi e li inserisca dentro un albero binario di ricerca (ABR). Si definiscono completi i nodi c aventi due nodi figli, incompleti i nodi i restanti. Scrivere un programma che:

- legga da tastiera N, il numero di interi da memorizzare nell’albero;
- legga da tastiera N interi, ovvero le etichette dei nodi da inserire nell’ABR. I valori devono essere inseriti nello stesso ordine con cui vengono letti (le etichette ≤ vanno inserite a sinistra);
- stampi la differenza tra la sommatoria S delle etichette dei nodi completi e la stessa dei nodi incompleti

$$S = \sum label(c) - \sum label(i)$$

Possibile soluzione: nodi “completi”

```
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector
#include <fstream>
#include <math.h>       /* floor */
#include <stdlib.h>
#include <cmath>        /* pow  */

using namespace std;


struct Node
{
    int value;
    Node * left;
    Node * right;

    Node( int i ): value(i) , left(NULL) , right(NULL) {}
};
```



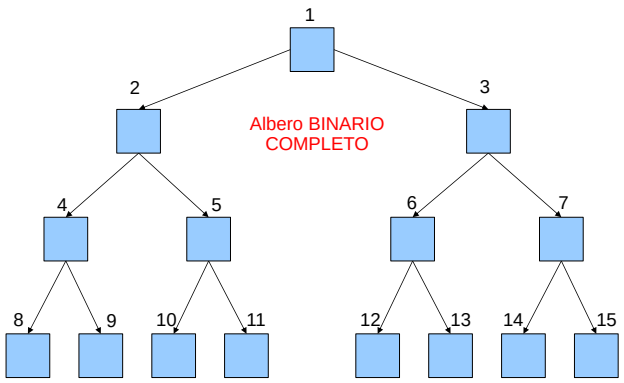
```
class BinTree
{
    Node * root_;
public:
    BinTree() { root_ = NULL ; }

    Node * getRoot() { return root_; cout << "getRoot" << endl;}

    void insert( int i )
    {
        Node * node = new Node(i);

        Node * pre = NULL;
        Node * post = root_;
        while( post != NULL)
        {
            pre = post;
            if( i <= post->value )
            {
                post = post->left;
            }
            else
            {
                post = post->right;
            }
        }

        if( pre == NULL )
            root_ = node;
        else if( i <= pre->value )
        {
            pre->left = node;
        }
        else
        {
            pre->right = node;
        }
        return;
    }
};
```



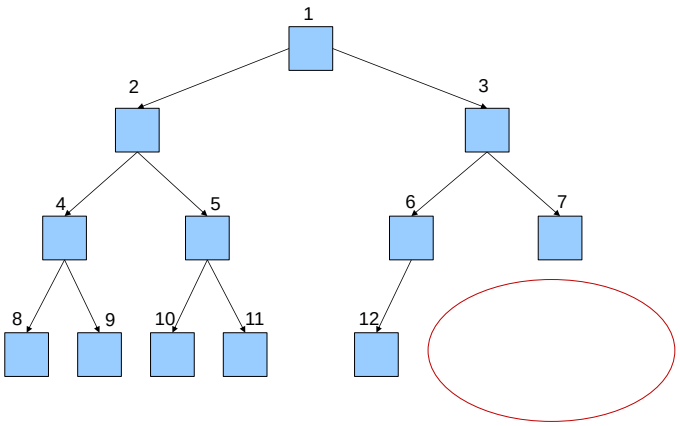
```
int complete( Node * tree)
{
    // Nodo non trovato
    if( tree == NULL)  {
       return 0;
    }

    if (tree->left != NULL && tree->right != NULL)  {
        return complete( tree->left ) + complete( tree->right ) + tree->value;
    }
    else   {
        return complete( tree->left ) + complete( tree->right ) - tree->value;
    }
}


int main()
{
    int N ;
    int x ;
    BinTree albero ;

    cin >> N ;

    // riempio l' albero
    for(int i=0 ; i<N ; ++i )   {
        cin >> x;
        albero.insert(x);
    }

    cout<<complete(albero.getRoot())<<endl;
}
```


HEAP

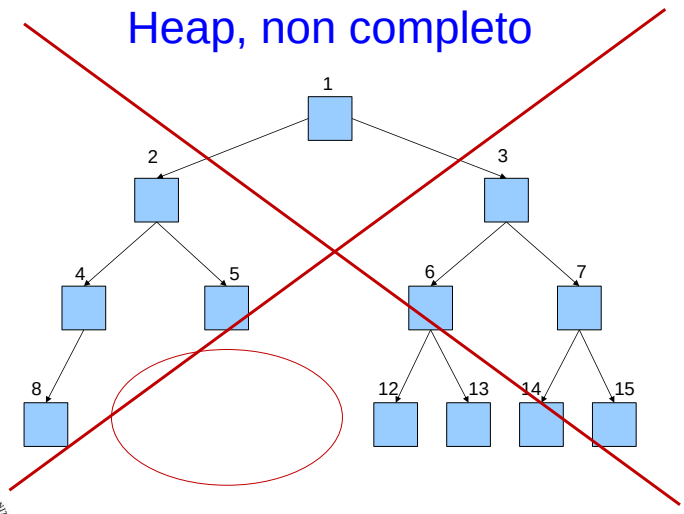
Proprieta’ strutturali Heap



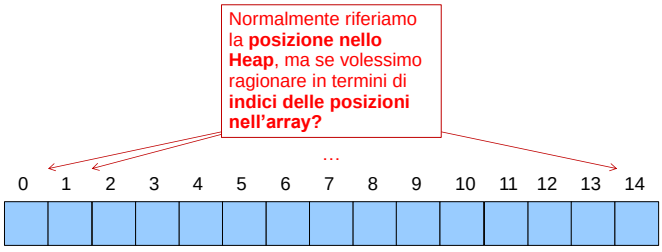
Heap, non completo



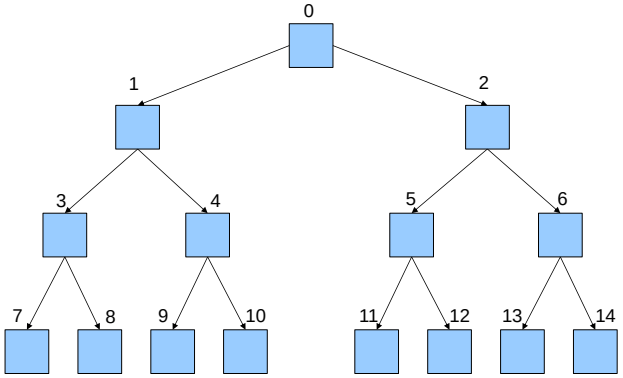
Heap, non completo



Heap <-> Array



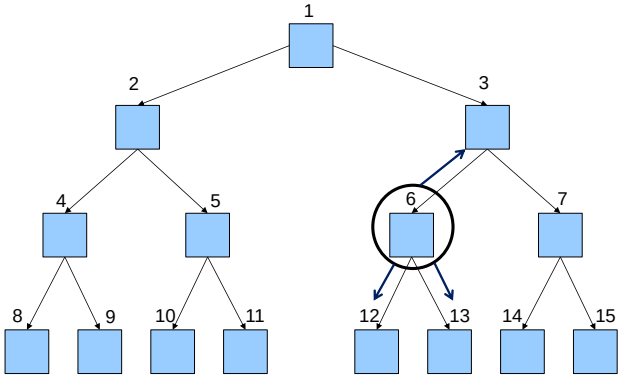
Heap <-> Array



Heap

```
1  class Heap
2  {
3      std::vector<int> data_;
4
5      int length_;    // lunghezza Array
6      int size_;   // dimensione Heap
7
8  public:
9      Heap() {};
10
11     void fill( int l );
12     void printVector();
13
14    ...
15 }
```

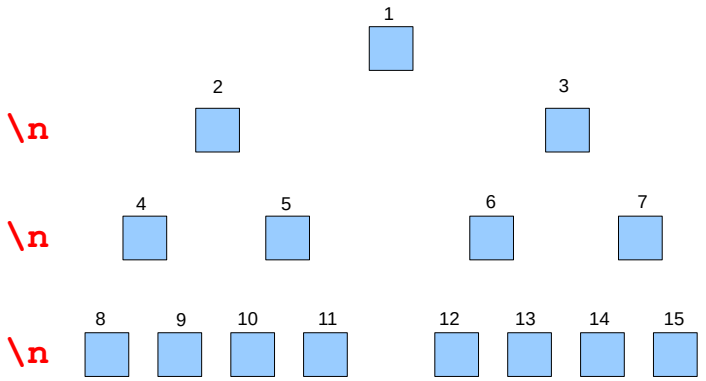

Navigare lo Heap grazie alle sue proprieta' strutturali



Navigare lo Heap

```
1  int parent(int i)
2  {
3      return floor((i-1)/2);    // floor(i/2)
4  }
5
6  int getLeft(int i)
7  {
8      return (i*2) + 1;         // i*2
9  }
10
11 int getRight(int i)
12 {
13     return (i*2)+2;           // (i*2)+1
14 }
15
16
17
18
```


Stampare il contenuto dello Heap



Quando andare a capo?

2	0010	-1 →	1	0001
4	0100	-1 →	3	0011
8	1000	-1 →	7	0111

8	1000	
7	0111	&
0	0000	

Print

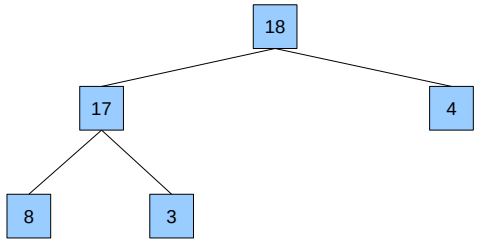
```
1  bool isFirstChild( int i )
2  {
3      if( ( i!=0 ) && ( (i&(i-1)) == 0) )
4          return true;
5      else
6          return false;
7  }
8
9
10
11
12
13
14
15
16
17
18
```

Quando andare a capo

Print

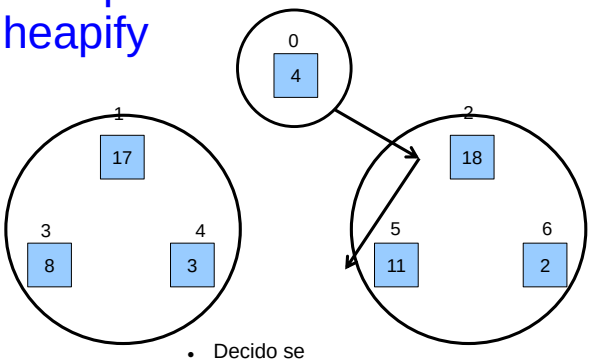
```
1  bool isFirstChild( int i )
2  {
3      if( ( i!=0 ) && ( (i&(i-1)) == 0) )
4          return true;
5      else
6          return false;
7  }
8
9  void print()
10 {
11     for( int i=0 ; i < length_ ; ++i )
12     {
13         if( isFirstChild(i+1) )
14             cout << endl;
15         cout << data_[i] << "\t";
16     }
17     cout << endl;
18 }
```


Proprieta' valori di un Heap



Consideriamo un max-heap:
ogni nodo padre deve avere etichetta > delle etichette dei suoi figli!

Esempio heapify



- Decido se
  - già ok?
  - andare a destra
  - andare a sinistra

heapify

```
1  void maxHeapify(int i)
2  {
3      // ottengo left e right
4
5
6
7      // (se ho figlio left) AND (left > i)
8          // left é più grande
9      // altrimenti
10         // i é più grande
11
12     // (se ho figlio right) AND (right > largest)
13         // right è più grande
14
15     // se i viola la proprietà di max-heap
16     {
17         // scambio i e il più grande
18         // controllo se l'albero che ho cambiato va bene
19     }
20 }
```


heapify

```
1  void maxHeapify(int i)
2  {
3      int left = getLeft(i);
4      int right = getRight(i);
5      int largest;
6
7      if((left < size_)&&(data_[left] > data_[i]))
8          largest = left;
9      else
10         largest = i;
11
12     if((right < size_)&&(data_[right] > data_[largest]))
13         largest = right;
14
15     if( largest != i )
16     {
17         scambia(i,largest);
18         maxHeapify(largest);
19     }
20 }
```

inizializzazione

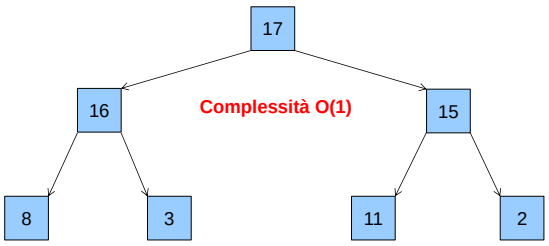
Identifico + grande

Aggiorno albero

Build Heap

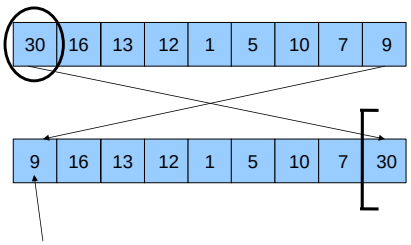
```
1  void buildMaxHeap()
2  {
3      size_ = length_;
4
5      int i = floor(length_/2)-1
6
7      for( ; i>=0 ; --i )
8      {
9          maxHeapify(i);
10         print();
11     }
12 }
13
14
15
16
17
18
19
20
```


Trovare il massimo?



Complessità O(1)

Heapsort



Max-heapify

Ad ogni passo:

- Scambio l'elemento max (la radice) con l'ultimo elemento
- Riduco la dimensione dello heap (size)
- Applico la max-heapify sulla radice

Heapsort

```
1  void heapSort()
2  {
3
4      int i = length_-1
5
6      for( ; i>0 ; --i)
7      {
8
9
10         scambia(0,i);
11
12
13
14         --size_;
15
16         maxHeapify(0);
17     }
18
19 }
20
```


Programma completo

```
1  int main()
2  {
3      Heap hp;
4
5      hp.fill();
6      hp.print();
7
8
9      hp.buildMaxHeap();
10     hp.print();
11
12     hp.heapSort();
13     hp.printArray();
14
15     return 0;
16
17 }
18
```


Heap STL

- **#include <algorithm>**
- **make_heap(inizio , fine)**
- **pop_heap(inizio , fine)**

#include <queue>

priority_queue<int> prioQ

prioQ.push(val)

prioQ.top()

prioQ.pop()

Algorithms

```
1  #include <vector>
2  #include <algorithm>
3
4
5  vector<int> vect;
6
7  for( int i = 0 ; i<quanti ; ++i )
8  {
9      cin >> val;
10     vect.push_back(val);
11 }
12
13 make_heap(vect.begin(),vect.end());
14
15 while(!vect.empty())
16 {
17     cout << "top " << *vect.begin() << endl;
18     pop_heap(vect.begin(),vect.end());
19     vect.pop_back();
20 }
```


priority_queue

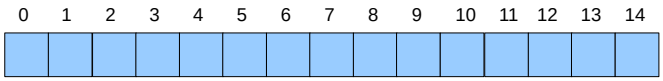
```
1  #include <queue>     // std::priority_queue
2
3  priority_queue<int> prioQ;
4
5  for( int i = 0 ; i<quanti ; ++i )
6  {
7      cin >> val;
8      prioQ.push(val);
9  }
10
11 while(!prioQ.empty())
12 {
13     cout << "top " << prioQ.top() << endl;
14     prioQ.pop();
15 }
16
17
18
19
20
```


Esercizi

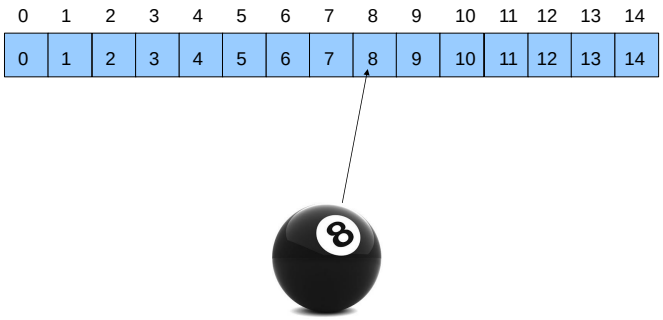
- Esercizi
  - Aggiunta nodo ad un heap
  - Eliminazione nodo da un heap
  - Aumento Valore di un nodo di un heap
- Esperimenti
  - Utilizzo Heap fatto a mano
  - Heapsort VS MergeSort
  - Priority_queue

HASH

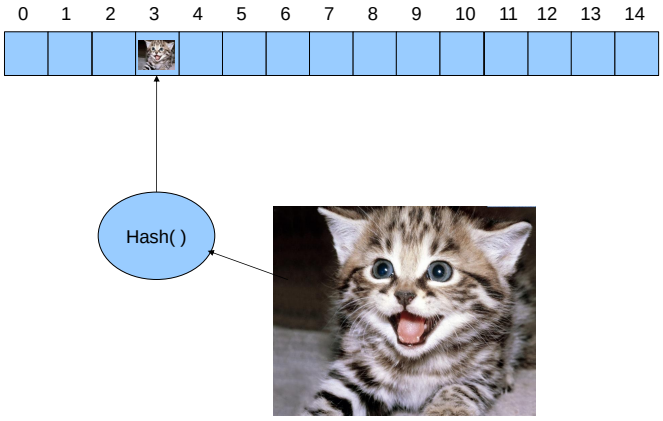
Array ad indirizzamento diretto



Array ad indirizzamento diretto

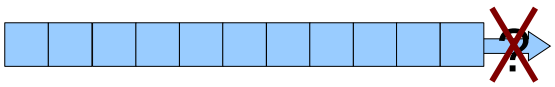


Hashing



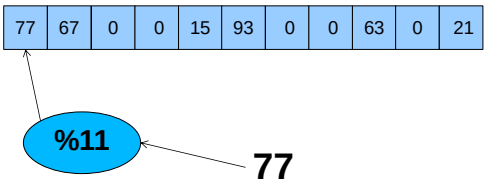
Strutture Dati

- Array
- Vector
- ...



Simple Hash Table

- Dati in input: interi positivi
- Chiave coincide con valore
- La funzione HASH e' la funzione modulo
- Convenzione: 0 per locazione vuota



Class

```
1  class HashTable
2  {
3      int * table_;
4      int size_;
5
6  public:
7      HashTable( int size );
8
9      bool insert( int key );
10
11     void print();
12
13     int hash( int key );
14 };
15
16
17
18
```


Costruttore

```
1  HashTable::HashTable( int size )
2  {
3      table_ =  new int[size];
4
5      size_ = size;
6
7
8
9  }
10
11
12
13
14
15
16
17
18
```

memset(address , value , size)

Example

```
1  /* memset example */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main ()
6  {
7    char str[] = "almost every programmer should know memset!";
8    memset (str,'-',6);
9    puts (str);
10   return 0;
11 }
```

Output:

```
------ every programmer should know memset!
```


Hashing

```
1  int HashTable::hash( int key )
2  {
3
4
5      return key % size_;
6
7  }
8
9
10
11
12
13
14
15
16
17
18
```


Insert

```
1  bool HashTable::insert( int key )
2  {
3      // trova indice tramite hashing
4
5
6      // se posizione già occupata
7      {
8          // non posso inserire
9
10     }
11
12
13
14     // inserisco
15
16
17 }
18
```


Insert

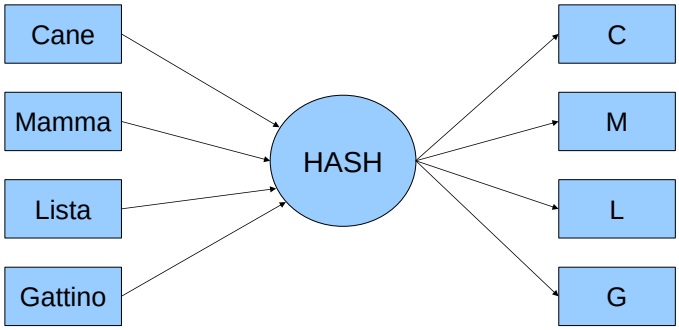
```
1  bool HashTable::insert( int key )
2  {
3      int index = hash(key);
4
5      if( table_[index] != 0 )
6      {
7          cout << "already occupied" << endl;
8          return false;
9      }
10     table_[index] = key;
11
12     cout << "key stored" << endl;
13     return true;
14 }
15
```

Esempio Hashing Stringhe: Prima lettera

```
int hash(string key)
{
    int index = key[0] % size_;
}
```

?

Esempio Hashing Stringhe: Prima lettera



Esempio Hashing Stringhe: Somma caratteri

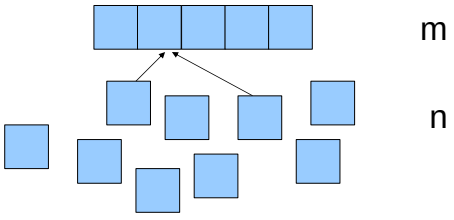
```
for( int i = 0 ; i < key.length() ; ++i )
{
  index = ( index + key[i] )% size_;
}
```

?

- Dipende fortemente dal tipo di applicazione
- Applicazioni con crittografia richiedono (tra le altre cose) di essere **difficilmente invertibili**.
- Per applicazioni di indexing e' fondamentale l'**uniformità**
- Lavorano sulla **rappresentazione binaria**
- E.g. MurmurHash, CityHash, FarmHash …

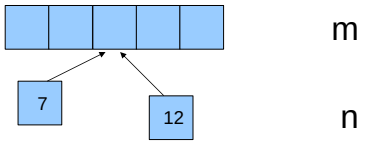
Google

Already occupied? -> Collisione



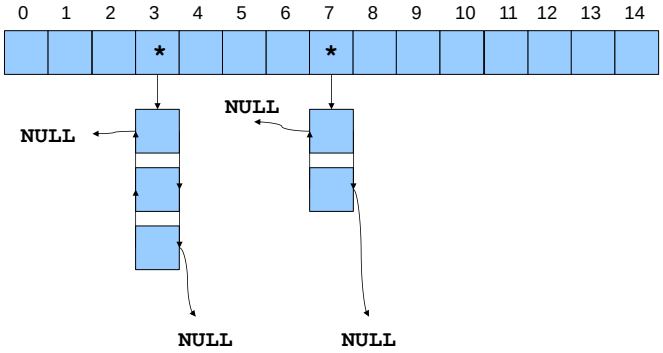
m<n

Gestione delle Collisioni



- Liste di trabocco
- Indirizzamento aperto

Array di puntatori



Elem

```
1  struct Elem
2  {
3     int key;
4     Elem * next;
5     Elem * prev;
6
7     Elem(): next(NULL) , prev(NULL) {}
8  };
9
10
11
12
13
14
15
16
17
18
```

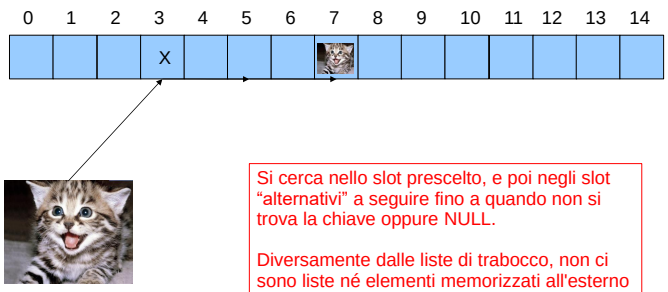

Hash con trabocco

```
1  class HashTable
2  {
3      Elem * * table_;
4      int size_;
5
6  public:
7
8
9
10
11
12
13
14
15
16
17
18 };
```


Implementazione

- Insert / Print / Find
- Stiamo trattando liste con inserimento in testa

Indirizzamento Aperto



Si cerca nello slot prescelto, e poi negli slot "alternativi" a seguire fino a quando non si trova la chiave oppure NULL.

Diversamente dalle liste di trabocco, non ci sono liste né elementi memorizzati all'esterno della tavola.

stl::map

```
std::map < key_T , obj_t > table;
```

Tipo chiave — Tipo dati

```
table['uno']="valore uno";
table.find('uno');
```

E infine…

Esperimenti

- Provare implementazioni variando:
  - Test dimensione table
  - Test tipi di hash
- Effettuare Confronto map vs hash
- Effettuare Confronto tempi di esecuzione Sorting e tabelle hash
- Interrogazioni:
  - Conoscere i primi K
  - Conoscere posizione di uno specifico elemento

Esercizio

Si consideri un sistema di memorizzazione dei dati relativi a dei veicoli. Ogni veicolo è rappresentato da un valore *targa* univoco, intero e positivo, e da un valore intero *categoria* compreso tra 0 e $C-1$. Il sistema legge i dati relativi a N veicoli e li inserisce dentro una *tabella hash*, utilizzando la *targa* come etichetta. La tabella hash è realizzata con il metodo di concatenazione.

Per ogni indice i della tabella hash si definisce $M(i)$ come la categoria con più veicoli in i, e $V(i)$ come il numero di veicoli in i appartenenti a $M(i)$. A parità di numero di veicoli, si considerino le *categorie* in ordine crescente. Scrivere un programma che:

- legga da tastiera una sequenza di N coppie di interi {*targa*,*categoria*} e le inserisca nella tabella hash all'indirizzo dato dalla seguente funzione hash:

$$h(x) = \{[(\ a \times x\ ) + \ b\ ]\ \%\ p\ \}\ \%\ (C)$$

  dove p=999149 , a=1000 e b=2000;
- Stampi a video i primi K indirizzi i della tabella hash in ordine di $V(i)$ decrescente. A parità di $V(i)$, si considerino gli indirizzi in ordine crescente.

L'**input** è formattato nel seguente modo: la prima riga contiene gli interi N, K e C separati da uno spazio. Seguono N righe contenenti una coppia di interi ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

Esempio

Input	Output
11 6 7 170 1 96 2 577 0 692 0 873 6 446 6 732 1 394 1 845 1 715 1 456 3	0 3 1 2 4 6

Possibile soluzione

```
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector
#include <fstream>
#include <math.h>       /* floor */
#include <stdlib.h>
#include <cmath>        /* pow  */

using namespace std;


struct Veicolo
{
    int targa;
    int categoria;

    Veicolo(): targa(-1) , categoria(-1) {};
    Veicolo( int t , int c ): targa(t) , categoria(c) {}
};

// conterrà le info relative a ciascuna entrata della tabella hash
struct Info
{
    int indirizzo;
    int veicoli;
};
```



```
class Concatenazione
{
    vector<int> categorie_;
    vector<Veicolo> veicoli_;

public:
    Concatenazione( int C )
    {
        categorie_.resize(C, 0);
    }

    // inserisci e aggiorna l'utente max
    void inserisci( Veicolo v )
    {
        veicoli_.push_back(v);

        ++categorie_[v.categoria];
    }

    int getMaxV()
    {
        int vMax = -1;
        int c;

        for( c = 0 ; c < categorie_.size(); ++c )
        {
//            cout << "\t" << c << "," << categorie_[c] << endl;
            if( categorie_[c] > vMax )
            {
                vMax = categorie_[c];
            }
        }
        return vMax;
    }
};
```

```
bool compare( Info a , Info b )
{
    if( a.veicoli > b.veicoli )
        return true;
    else if( a.veicoli == b.veicoli)
        return a.indirizzo < b.indirizzo;
    else
        return false;
}

int main()
{
    unsigned int N , K , C;
    int x , c ;

    cin >> N >> K >> C;
    Concatenazione cBase(C);
    vector<Concatenazione> hashTable(C, cBase);

    int pos;

    Veicolo tempVeicolo;
    // riempio le liste di concatenazione
    for(unsigned int i=0 ; i<N ; ++i )
    {
        // leggo e creo un nuovo utente
        cin >> x >> c;
        tempVeicolo.targa = x;
        tempVeicolo.categoria = c;

        pos = hashFun( x , C );

//          cout << x << "," << c << " -> " << pos << endl;

        hashTable[pos].inserisci(tempVeicolo);
    }
```



```
    // calcolo insieme I
    vector <Info> info;
    Info tempInfo;
    for( unsigned int i=0 ; i<C ; ++i )
    {
        tempInfo.indirizzo = i;

//          cout << endl << "============= " << i << " =============" << endl;
        tempInfo.veicoli = hashTable[i].getMaxV();
        info.push_back( tempInfo );
    }
    sort( info.begin() , info.end() , compare );

    for( unsigned int i=0 ; i < K && i<info.size() ; ++i )
    {
        cout << info[i].indirizzo << endl;
    }
}
```