Liste, insiemi, Alberi e Ricorsione

Settimo Laboratorio

Liste e Ricorsione

SVUOTALISTA: CONSIDERAZIONI

- Per svuotare una lista si devono eliminare i singoli nodi allocati con la malloc...
- Come fare?
- Per eliminare ogni nodo, dovrò potermi muoveri sulla lista
- ... quindi se parto ad eliminare dal primo nodo, succede che...???

svuotaLista: versione Ricorsiva

- Schema Ricorsivo:
  - caso base: E([]) = []
  - caso generale: E([a|L]) = [E(L)]

```
void svuotaListaRic(lista* l){
  if((*l) == NULL) return; // lista vuota
  svuotaListaRic(&((l*)->next)); // scorro la lista
  free(*l); // elimino nodo
  (*l) = NULL; // aggiorno puntatore
}
```

Note: “Notice that this function leaves the value of *ptr* unchanged, hence it still points to the same (now invalid) location, and not to the null pointer.”

http://www.cplusplus.com/reference/clibrary/cstdlib/free/

svuotaLista: versione Iterativa

```
void svuotaListaIt(lista* l){
  if(*l == NULL) return;

  nodo * curr = *l;
  nodo * succ = (*l)->next;
  *l = NULL;

  while(curr != NULL){
       free(*curr);
       *curr = *succ;
       *succ = (*succ)->next;
  }
}
```

ESERCIZIO: INSERIMENTO IN POSIZIONE I_ESIMA

- *Inserire un nuovo elemento in posizione i_esima. Se la lista contiene meno di i elementi, inserire il nuovo elemento in coda.*

SCHEMA RICORSIVO:

- caso base:

- caso generale:

ESERCIZIO: INSERIMENTO IN POSIZIONE I_ESIMA

- *Inserire un nuovo elemento in posizione i_esima. Se la lista contiene meno di i elementi, inserire il nuovo elemento in coda.*

```
lista InserisciPosizione(lista li, int pos, int i, int v);
```

- Argomenti:
  - li: puntatore al primo nodo della lista;
  - pos: posizione corrente nella lista;
  - i: posizione dove aggiungere il nuovo elemento;
  - v: valore del nuovo elemento.

ESERCIZIO: INSERIMENTO IN POSIZIONE I_ESIMA

- *Inserire un nuovo elemento in posizione i_esima. Se la lista contiene meno di i elementi, inserire il nuovo elemento in coda.*

```
lista InserisciPosizione(lista li, int pos, int i, int v);
```

- SCHEMA RICORSIVO:
  - caso base: II([], pos, i, v) = [v]
  - caso base: II([a|L], pos, i, v) = [v|a|L] // se i = pos
  - caso generale:
    - II([a|L], pos, i, v) = [a|II(L, pos, i + 1, v)] // se i < pos

ESERCIZIO: INSERIMENTO IN POSIZIONE I_ESIMA

```
lista InserisciPosizione(lista li, int pos, int i, int v) {
   if(li == NULL){
      lista pn = malloc(sizeof(nodo));
      pn->val = v;
      pn->next = NULL;
      return pn;
   }
   if(pos == i){
      return InserisciinTesta(li, v);
   }
   li->next = InserisciPosizione(li->next, pos, i + 1, v);
   return li;
}
```

ESERCIZIO: ELIMINA NODO

- Scrivere un programma che elimini tutti i nodi di una lista che contengono il valore v.

- SCHEMA RICORSIVO:
  - caso base:
  - caso generale:

ESERCIZIO: ELIMINA NODO

- Scrivere un programma che elimini tutti i nodi di una lista che contengono il valore v.

- SCHEMA RICORSIVO:
  - caso base: E([], v) = []
  - caso generale: $E([a|L], v) = \begin{cases} E([L], v) & \text{// se a = v} \\ [a|E(L)] & \text{// se a != v} \end{cases}$

ESERCIZIO: ELIMINA NODO

```
lista eliminaNodo(lista li, int v) {
  if(li == NULL) return li;

  nodo * succ = li->next;

  if(li->value == v){
    li->next = NULL;
    free(li);
    return eliminaNodo(succ, v);
  }
  li -> next = eliminaNodo(succ, v);
  return li;
  }
```

ESERCIZIO: STAMPA NODI IN POSIZIONE DISPARI

- Scrivere un programma che stampa solamente i valori contenuti nei nodi in posizione dispari

- SCHEMA RICORSIVO:
  - caso base:
  - caso generale:

ESERCIZIO: STAMPA NODI IN POSIZIONE DISPARI

- Scrivere un programma che stampa solamente i valori contenuti nei nodi in posizione dispari

- SCHEMA RICORSIVO:
  - caso base: S([], i) = stampa[]
  - caso generale: $S([a|L], i) = \begin{cases} \text{stampa}[a]\ S([L]) & \text{se i è dispari} \\ S([L]) & \text{se i è pari} \end{cases}$

ESERCIZIO: STAMPA NODI IN POSIZIONE DISPARI

```
void stampadispari(lista li, int i) {
  if(li == NULL) return;
  if(i % 2 != 0) printf("%d ", li->value);
  stampadispari(li->next, i + 1);
  }
```

ESERCIZIO: CONTA VALORI

- *Scrivere una funzione ricorsiva che conta il numero di elementi contenuti in una lista che hanno valore minore di un valore v*

```
int contaMinore(lista li, int v);
```

- SCHEMA RICORSIVO:
  - caso base:
  - caso generale:

ESERCIZIO: CONTA VALORI

- *Scrivere una funzione ricorsiva che conta il numero di elementi contenuti in una lista che hanno valore minore di un valore v*

```
int contaMinore(Lista li, int v);
```

- SCHEMA RICORSIVO:
  - caso base: C([]) = 0
  - caso generale: $C([a|L]) = \begin{cases} 1+C(L) & \text{// se a < v} \\ 0+C(L) & \text{// se a >= v} \end{cases}$

ESERCIZIO: CONTA VALORI

```
int contaValori(lista li, int v) {
  if(li == NULL) return 0;
  if(li->value == v)
    return 1 + contaValori(li->next, v);
  return contaValori(li->next, v);
  }
```

ESERCIZIO: TROVA VALORE

- *Scrivere una funzione ricorsiva che ritorna 1 se il valore v è contenuto nella lista li, 0 altrimenti.*

```
int trovaValore(lista li, int v);
```

- SCHEMA RICORSIVO:
  - caso base:
  - caso generale:

ESERCIZIO: TROVA VALORE

- *Scrivere una funzione ricorsiva che ritorna 1 se il valore v è contenuto nella lista li, 0 altrimenti.*

```
int trovaValore(Lista li, int v);
```

- SCHEMA RICORSIVO:
  - caso base: T([], v) = 0
  - caso generale: $T([a|L]) = \begin{cases} 1 & \text{// se a = v} \\ T([L], v) & \text{// se a != v} \end{cases}$

ESERCIZIO: TROVA VALORE

```
int trovaValore(lista li, int v) {
  if(li == NULL) return 0;
  if(li->value == v) return 1;
    return trovaValori(li->next, v);
  }
```

Insiemi e Ricorsione

INSIEME

- *Lista con due proprietà*
  - *Ordinata*
  - *Senza repliche*

```
typedef Lista Insieme;
```

- *Ogni operazione che viene eseguita su un oggetto di tipo insieme, deve rispettare le due proprietà.*

Funzione Inserimento

- *L'inserimento di un nuovo elemento all'interno di un insieme, richiede di individuare la corretta posizione di inserimento, per mantenere la proprietà di ordinamento, e dovrà controllare che l'elemento non sia già presente.*
- *Si può fare sia ricorsivamente sia iterativamente*

- Schema Ricorsivo
  - caso base: I([], v) = [v]
  - caso base: I([a|L], v) = $\begin{cases} [v\mid a \mid L] & \text{se } a > v \\ [a \mid L] & \text{se } a = v \end{cases}$
  - caso ricorsivo: I([a|L], v) = $[a\mid I(L, v)]$ se $a < v$

FUNZIONE INSERIMENTO

```
Insieme InsiemeIns(Insieme li, int el) {
  if (!li || li->val > el)
   return inserisciListaTesta(li,el);
  if (li->val==el) return li;

  if (li->val<el) {
    Insieme pi=InsiemeIns(li->next, el);
    li->next=pi;
    return li;
  }
}
```

ESERCIZIO: INTERSEZIONE

- *Scrivere la funzione che esegue l'intersezione di due insiemi.*

```
Insieme InsiemeIntersezione(Insieme l1, Insieme l2);
```

- SCHEMA RICORSIVO:
  - Caso base:
  - Caso generale:

ESERCIZIO: INTERSEZIONE

- Scrivere la funzione che esegue l'intersezione di due insiemi.

```
Insieme InsiemeIntersezione(Insieme l1, Insieme l2);
```

- SCHEMA RICORSIVO:
  - Caso base:
    $$\begin{cases} I([],[]) = [] \\ I([],[a|L]) = [] \\ I([a|L],[]) = [] \end{cases}$$
  - Caso generale:
    $$I([a|L1],[b|L2]) = \begin{cases} I(L1,[b|L2]) \text{ se } a<b \\ I([a|L1],L2) \text{ se } a>b \\ [a|I(L1,L2)] \text{ se } a=b \end{cases}$$

ESERCIZIO: INTERSEZIONE

```
Insieme InsiemeIntersezione(Insieme l1, Insieme l2) {

  if (!l1 && !l2) return NULL;
  if (l1 && !l2)  return InsiemeIntersezione(l1->next,l2);
  if (!l1 && l2) return InsiemeIntersezione(l1,l2->next);

  if (l1->val==l2->val)
    return ListInsInTesta(InsiemeIntersezione(l1->next,
  l2->next), l1->val);

  if (l1->val<l2->val)
    return InsiemeIntersezione(l1->next,l2);

  if (l1->val>l2->val)
    return InsiemeIntersezione(l1,l2->next);
}
```

ESERCIZIO: DIFFERENZA

- *Scrivere la funzione che esegue la differenza tra due insiemi.*

```
Insieme InsiemeDifferenza(Insieme l1, Insieme l2);
```

- SCHEMA RICORSIVO:
  - Caso base:
    $$\begin{cases} D([],[])=[] \\ D([],[a|L])=[] \\ D([a|L],[])=[a|D(L,[])] \end{cases}$$
  - Caso generale:
    $$D([a|L1],[b|L2]) = \begin{cases} [a|D(L1,[b|L2]) \text{ se } a<b \\ D([a|L1],L2) \text{ se } a>b \\ D(L1,L2) \text{ se } a=b \end{cases}$$

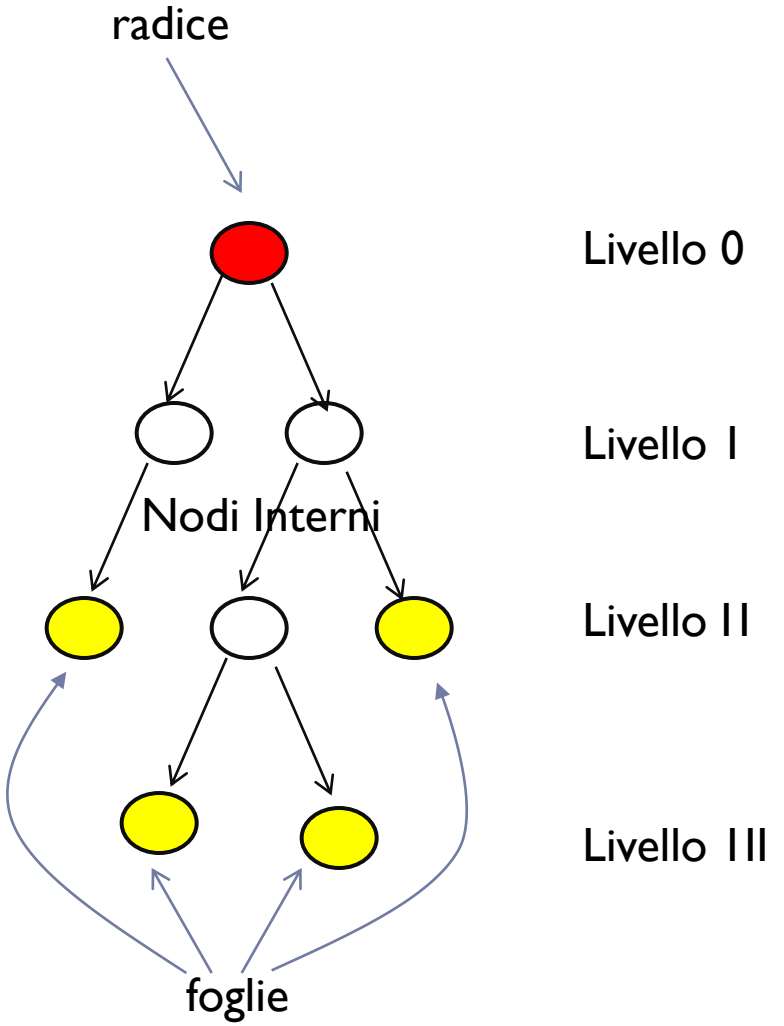
ESERCIZIO: DIFFERENZA

```
Insieme InsiemeDifferenza(Insieme l1, Insieme l2) {
  if (!l1 && !l2) return NULL;
  if (l1 && !l2)
    return ListInsInTesta(InsiemeDifferenza(l1->next,l2),
  l1->val);
  if (!l1 && l2) return InsiemeDifferenza(l1,l2->next);

  if (l1->val==l2->val)
    return InsiemeDifferenza(l1->next, l2->next);

  if (l1->val<l2->val)
    return ListInsInTesta(InsiemeDifferenza(l1->next,l2),
  l1->val);

  if (l1->val>l2->val)
    return InsiemeDifferenza(l1,l2->next);
}
```

Alberi e Ricorsione

Definizione: Albero

```
typedef struct Nodo {
   int val;
   struct Nodo* left;
   struct Nodo* right;

} Nodo;

typedef Nodo* Albero;
```



ESERCIZIO: CREO UN ALBERO VUOTO

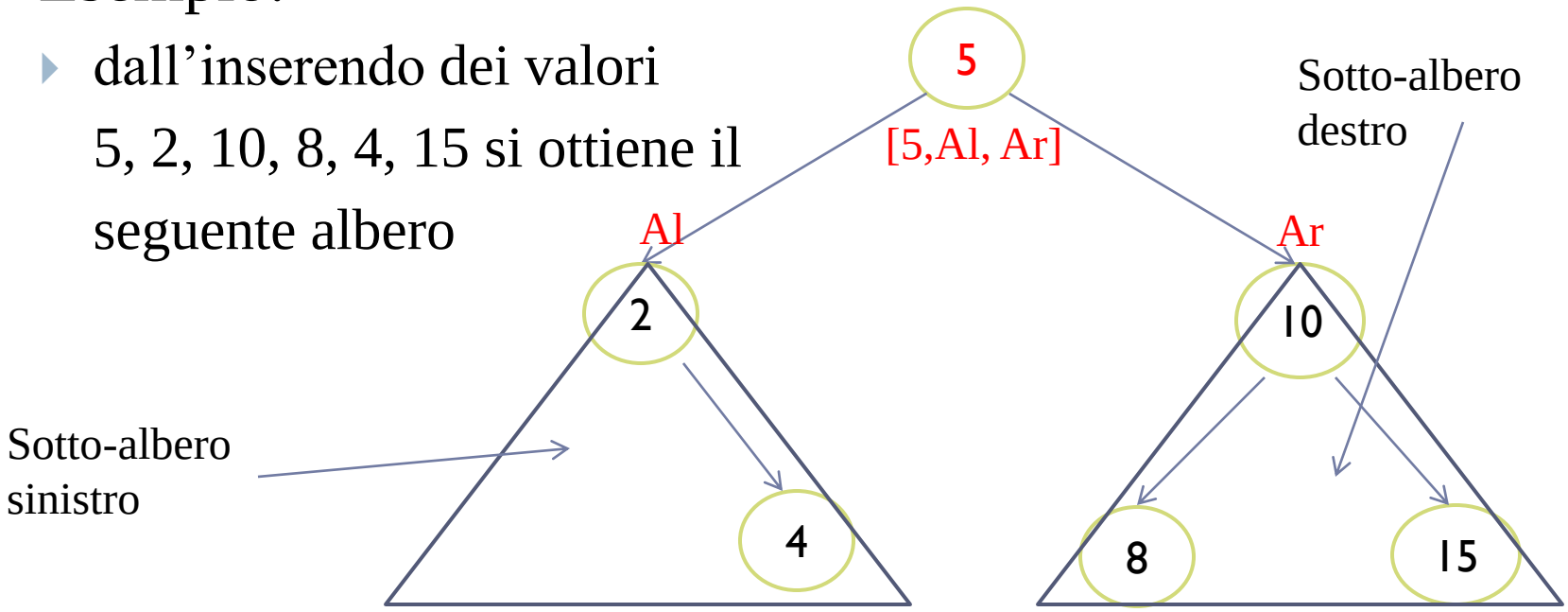
```
Albero CreaAlberoVuoto(){
  return NULL;
}
```

MAIN

```
typedef struct Nodo {
  int val;
  struct Nodo* left;
  struct Nodo* right;
} Nodo;

typedef Nodo* Albero;

Albero CreaAlberoVuoto(){
  return NULL;
}

main(){
  ...
}
```

ESERCIZIO: ALBERO ORDINATO

- *Creare un albero binario ordinato: ogni nodo contiene nell'albero di sinistra tutti i valori minori o uguali al valore del nodo e nell'albero di destra tutti i valori maggiori al valore del nodo.*
- Esempio:
  - dall'inserendo dei valori 5, 2, 10, 8, 4, 15 si ottiene il seguente albero



ESERCIZIO: INSERIMENTO DI UN ELEMENTO

- *Creare una funzione per inserire un elemento all'interno dell'albero, mantenendo la proprietà di "ordinamento" dell'albero.*

```
Albero Inserisci(Albero alb,  int val);
```

- SCHEMA RICORSIVO:
  - Caso base:
  - Caso generale:

ESERCIZIO: INSERIMENTO DI UN ELEMENTO

- *Creare una funzione per inserire un elemento all'interno dell'albero, mantenendo la proprietà di "ordinamento" dell'albero.*

```
Albero Inserisci(Albero alb,  int val);
```

- SCHEMA RICORSIVO:
  - Caso base: I([], v) = [v]
  - Caso generale:

$$I(A=[val|Al|Ar], v) = \begin{cases} [val|I(Al, v)|Ar] & \text{// se } v <= val \\ [val|Al|I(Ar, v)] & \text{// se } v > val \end{cases}$$

ESERCIZIO: INSERIMENTO DI UN ELEMENTO

```
Albero Inserisci(Albero al, int v){
  if(al == NULL){
    Albero pn=malloc(sizeof(Nodo));
    pn->val=v;
    pn->left=NULL;
    pn->right=NULL;
    return pn;
  }

  if(al->val >= v){
    al->left = Inserisci(al->left, v);
  }else{
    al->right = Inserisci(al->right, v);
  }
  return al;
}
```

ESERCIZIO: STAMPA ALBERO

- *Scrivere una funzione che stampa il contenuto dell'albero.*

```
void Stampa(Albero alb);
```

- SCHEMA RICORSIVO:
  - Caso base: S([]) = []
  - Caso generale: S(A=[val|Al|Ar]) = val S(Al) S(Ar)

ESERCIZIO: STAMPA ALBERO

```
// stampa in ordine di “inserimento”
void Stampa(Albero al){
  if(!al) return;
  printf("%d - ", al->val);
  Stampa(al->left);
  Stampa(al->right);
}
```

ESERCIZIO: STAMPA ALBERO

```
// variante ordine crescente
void StampaOrdinata(Albero al){
  if(!al) return;
  StampaOrdinata(al->left);
  printf("%d - ", al->val);
  StampaOrdinata(al->right);
}
// variante ordine decrescente
void StampaInversa(Albero al){
  if(!al) return;
  StampaInversa(al->right);
  printf("%d - ", al->val);
  StampaInversa(al->left);
}
```

ESERCIZIO: CERCA VALORE

- *Scrivere una funzione ricorsiva che cerca il nodo contenente un valore "v" all'interno dell'albero.*

```
Albero CercaValore(Albero alb,  int val);
```

- SCHEMA RICORSIVO:
  - Caso base:
  - Caso generale:

ESERCIZIO: CERCA VALORE

- *Scrivere una funzione ricorsiva che cerca il nodo contenente un valore "v" all'interno dell'albero.*

```
Albero CercaValore(Albero alb,  int val);
```

- SCHEMA RICORSIVO:
  - Caso base:
    $$\begin{cases} C([], v) = [] \\ C(A=[val, Al, Ar], v) = A \quad \text{se } v = val \end{cases}$$
  - Caso generale:
    $$C(A=[val, Al, Ar], v) = \begin{cases} C(Al, v) \ \text{// se } v < val \\ C(Ar, v) \ \text{// se } v > val \end{cases}$$

ESERCIZIO: CERCA VALORE

```
Albero Cerca(Albero al, int v){
  if(!al) return NULL;
  if(v == al->val) return al;

  if(v < al->val)
    return Cerca(al->left, v);
  else
    return Cerca(al->right, v);
}
```

CONTA NUMERO NODI

- Scrivere una funzione che ritorna il numero di nodi presenti in un albero

- Schema ricorsivo
  - caso base:

  - caso ricorsivo:

CONTA NUMERO NODI

- Scrivere una funzione che ritorna il numero di nodi presenti in un albero

- Schema ricorsivo
  - caso base: C([]) = 0
  - caso ricorsivo: C([a|Al|Ar]) = 1 + C(Al) + C(Ar)

CONTA NUMERO NODI

```
int contaNumeroNodi(Albero al){
  if(!al) return 0;
  return 1 + contaNumeroNodi(al->left) +
    contaNumeroNodi(al->right);
}
```

ELIMINA FOGLIE

- Scrivere una funzione che elimini tutte le foglie dell'albero passato e ritorni il numero di elementi eliminati

- Schema ricorsivo
  - caso base;

  - caso ricorsivo:

ELIMINA FOGLIE

- Scrivere una funzione che elimini tutte le foglie dell'albero passato e ritorni il numero di elementi eliminati

- Schema ricorsivo
  - caso base: E([]) = [], 0
  - caso base: E([a|Al|Ar]) = [], 1 // se Al = Ar = NULL

  - caso ricorsivo = E([a|Al|Ar]) = [a|E(Al) |E(Ar)]

Sesta Consegna

Sesta Consegna

- Consegna dovrà essere eseguita entro il giorno 26 gennaio 2011
- Le consegne effettuate successivamente non verranno considerate...

- Comando per la consegna:

consegna consegna6

ESERCIZIO 1

- Creare un file Amici che contenga per ogni riga i seguenti dati:

  Nome spazio Cognome spazio età spazio squadra del cuore

- Realizzare un programma che implementi le seguenti funzioni:
  - leggere il file creando una lista dinamica di tipo amico (un tipo definito come sopra);
  - data la lista di amici e un intero, crea una nuova lista di tutti gli amici che hanno quell'età e la visualizza;
  - data la lista di amici e una squadra di calcio, elimina dalla lista tutti gli amici che tifano quella squadra di calcio e visualizza la lista.

ESERCIZIO 2

- Si consideri il seguente problema di matematica ricreativa, dovuto al matematico russo Boris Kordemsky.

  “ Il gatto Purrer ha deciso di schiacciare un pisolino, e subito sogna di essere circondanto da 13 topolini, 12 grigi e uno bianco disposti a cerchio; nel sogno il padrone gli dice: "Purrer, devi mangiare tutti i topi ma quello bianco per ultimo".

  Dato che non li mangia in sequenza ma saltando ogni volta n posizioni, da quale topo deve partire, ovvero in quale posizione si deve trovare il topolino bianco partendo dal primo mangiato? “

- Realizzare un programma in C che, fornito in ingresso il passo n che il gatto fa tra due topi mangiati (n>=2), tramite l'utilizzo di una lista dica in quale posizione si deve trovare il topo bianco rispetto al primo mangiato.