Implementazione di strutture lineari e non lineari

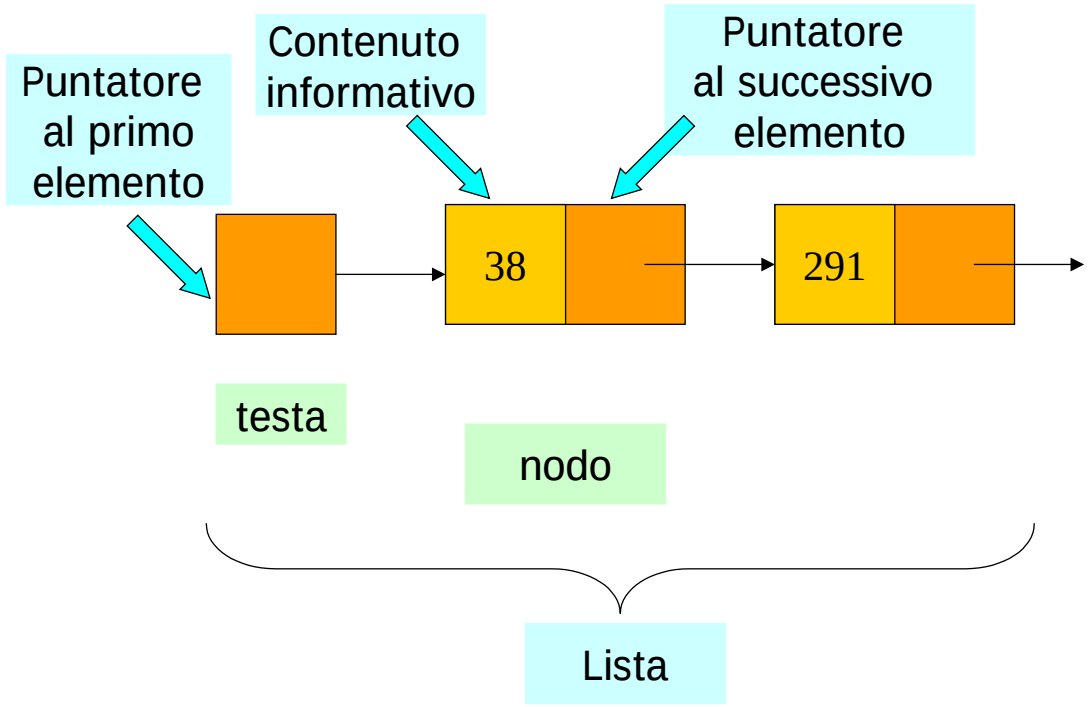
- La lista, la pila e la coda, l'albero e il grafo possono essere rappresenti nella memoria di un calcolatore utilizzando strutture gestite dinamicamente

- In C la rappresentazione farà uso di struct definite ricorsivamente in cui uno dei campi rappresenterà il contenuto informativo dell'elemento ed uno o più campi il puntatore ad altri elementi della struttura

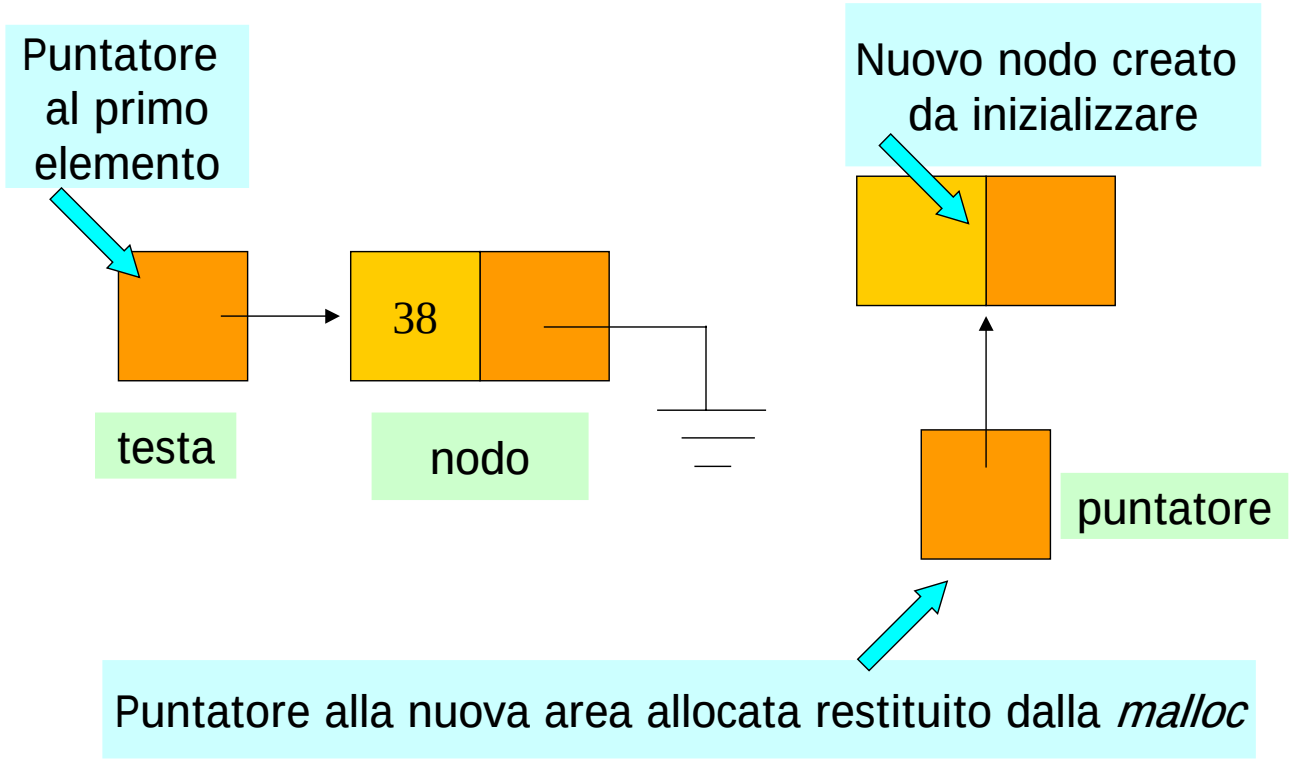
Realizzazione di una lista

Il tipo degli elementi

```
struct strutlista {
      struct strutnodo *testa;
};


struct strutnodo {
      struct strutnodo *prossimo;
      char                    *testo;
};
```



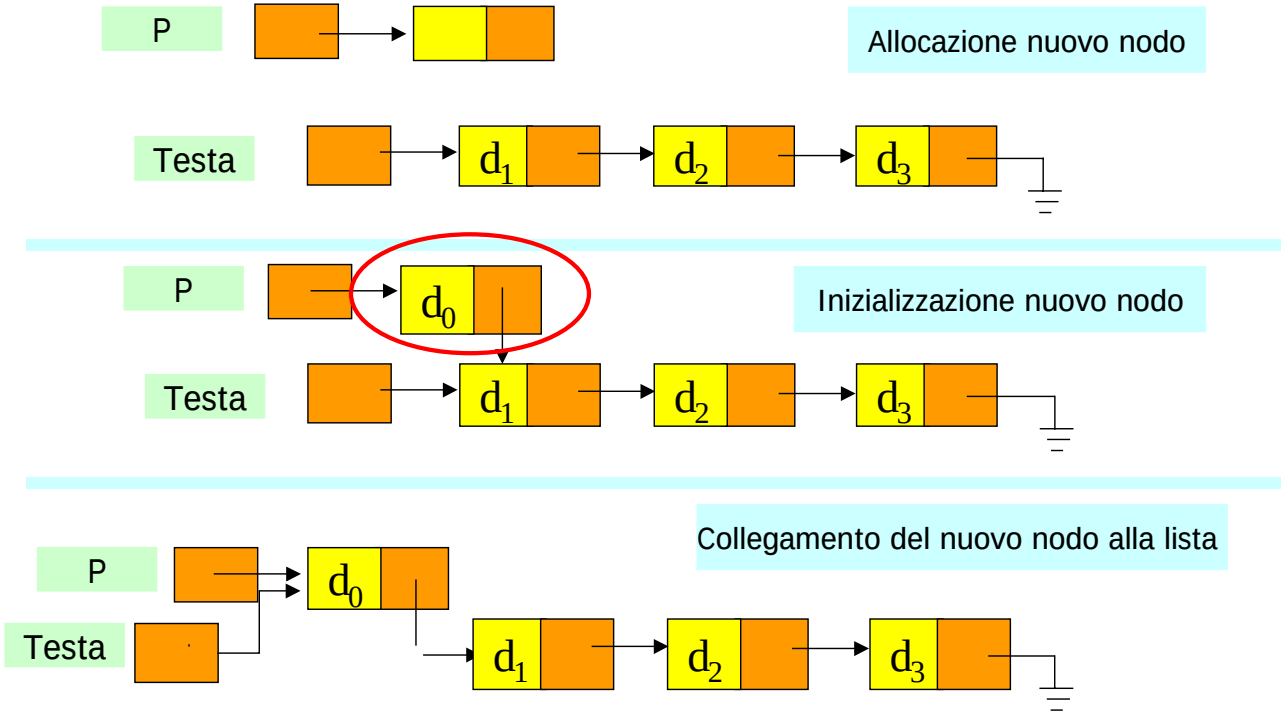
Allocazione della memoria per un nuovo nodo di una lista



Allocazione della memoria per un nuovo nodo di una lista

```
struct strutnodo *allocanodo(char *stringa) {
      struct strutnodo *sn;
      sn=malloc(sizeof(struct strutnodo));
      if (sn) {
        sn->testo=malloc(strlen(stringa)+1);
        if (sn->testo) {
              strcpy(sn->testo,stringa);
              return sn;
        } else {
              free(sn);
              return 0;
        };
      };
      return sn;
}
```

Inserimento in testa ad una lista

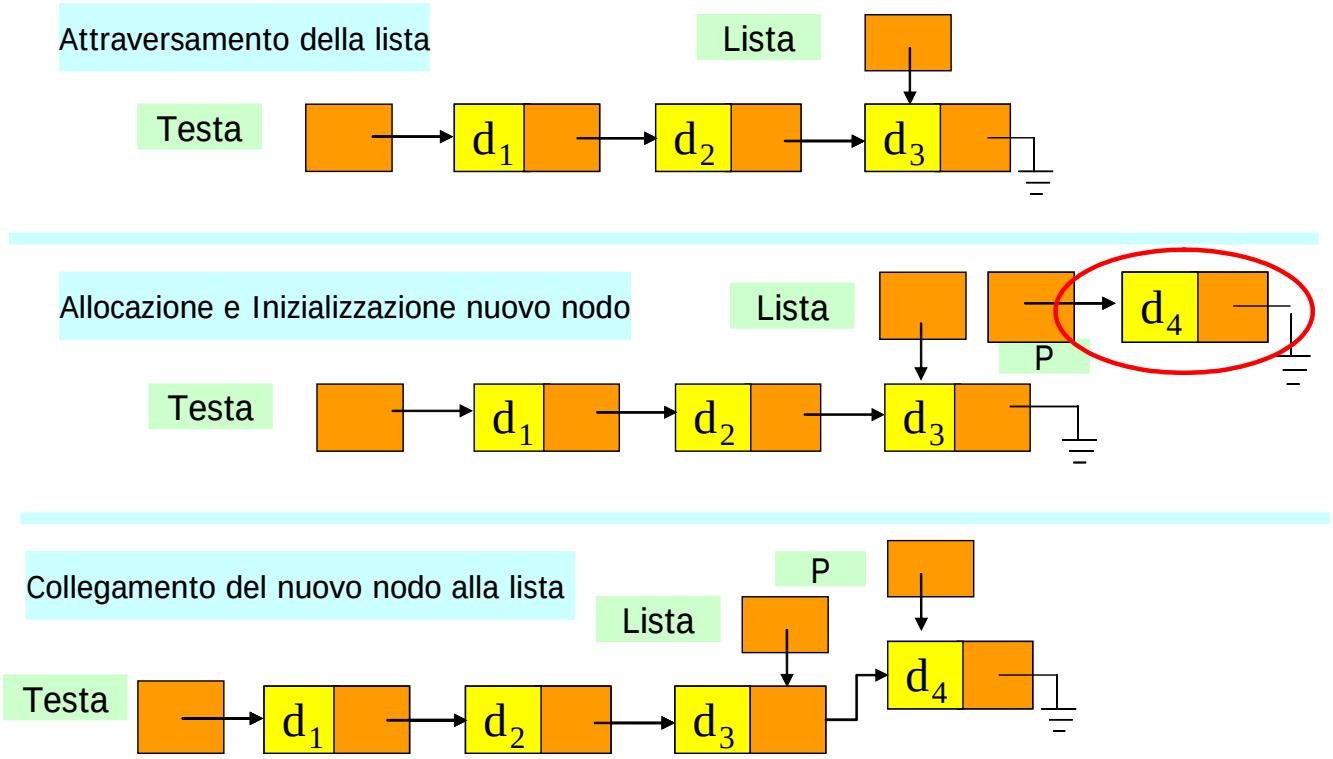


Algoritmi di inserimento di un nodo in una lista

Inserimento in testa alla lista

```
struct strutnodo *inseriscitesta(struct strutlista *ls,char *stringa) {
      struct strutnodo *sn;
      sn=allocanodo(stringa);
      if (sn) {
        sn->prossimo=ls->testa;
        ls->testa=sn;
      };
      return sn;
}
```

Inserimento in coda ad una lista



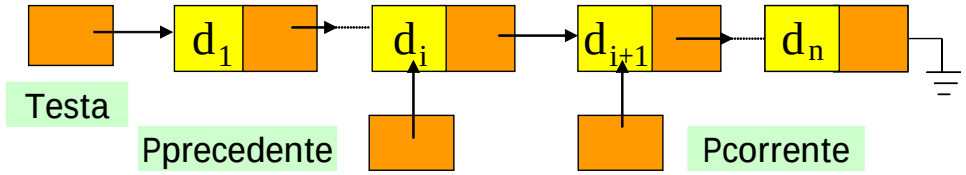
Algoritmi di inserimento di un nodo in una lista

Inserimento in coda alla lista

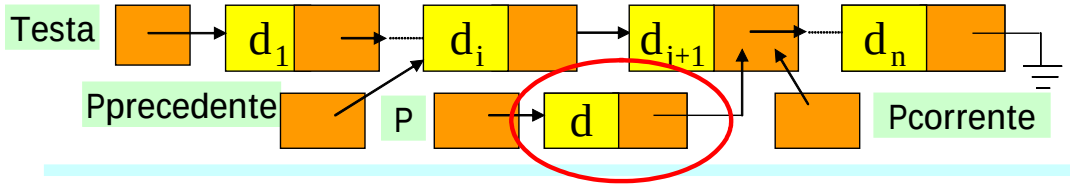
```
struct strutnodo *inseriscicoda(struct strutlista *ls,char *stringa) {
      struct strutnodo *sn,*ptr;
      sn=allocanodo(stringa);
      if (sn) {
        sn->prossimo=0;
        if (ls->testa==0) {
              ls->testa=sn;
        } else {
              for(ptr=ls->testa;ptr->prossimo!=0;ptr=ptr->prossimo);
              ptr->prossimo=sn;
        };
      };
      return sn;
}
```

Inserimento in ordine in una lista

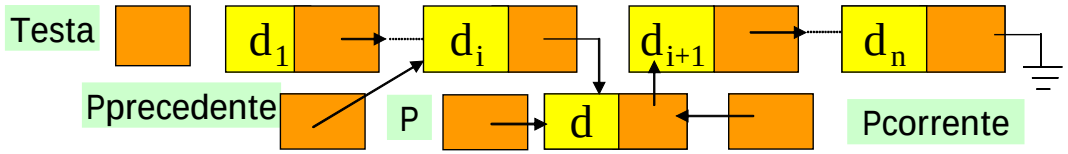
Individuazione della posizione in cui inserire il dato $d_i<d<d_{i+1}$



Allocazione e Inizializzazione del nuovo nodo



Collegamento del nuovo nodo alla lista

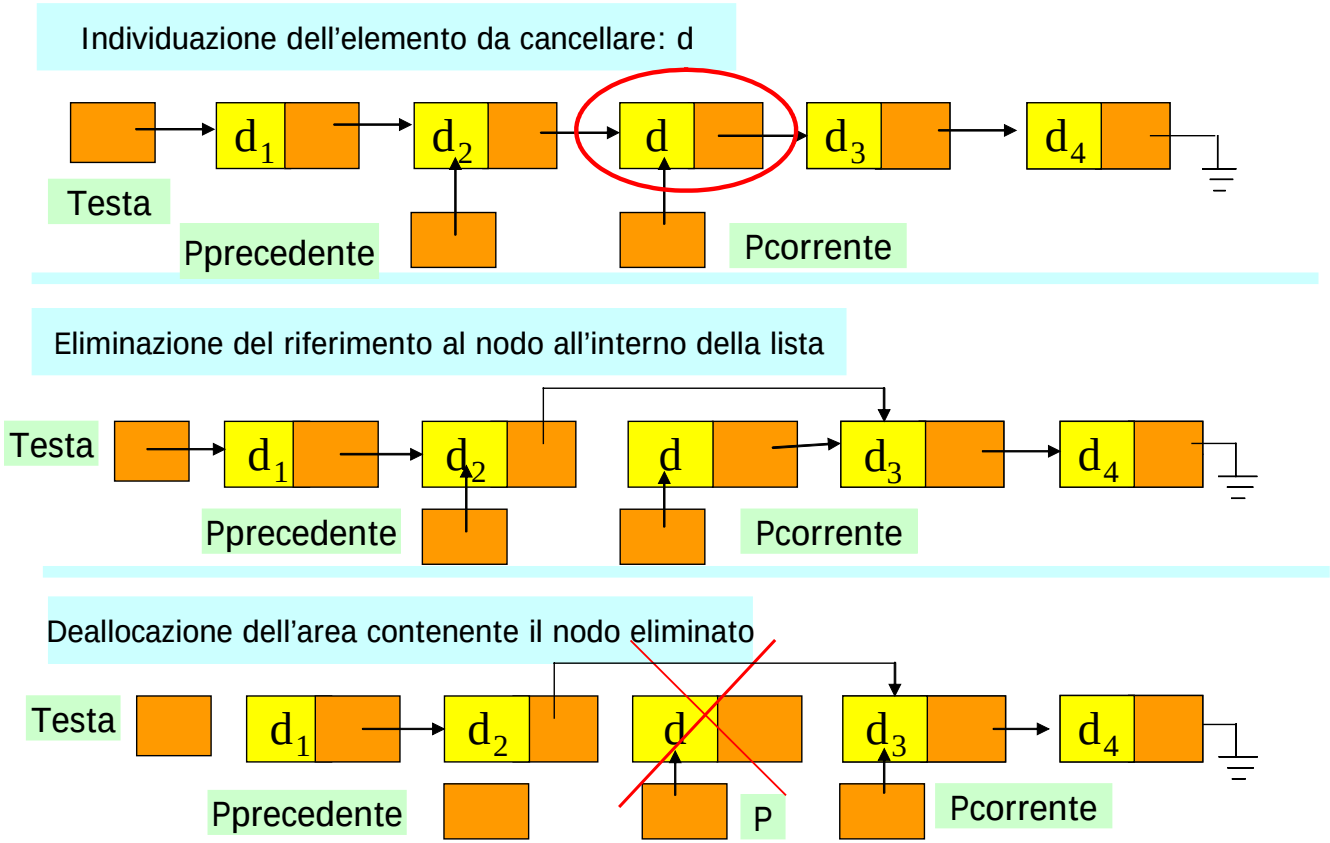


Algoritmi di inserimento di un nodo in una lista

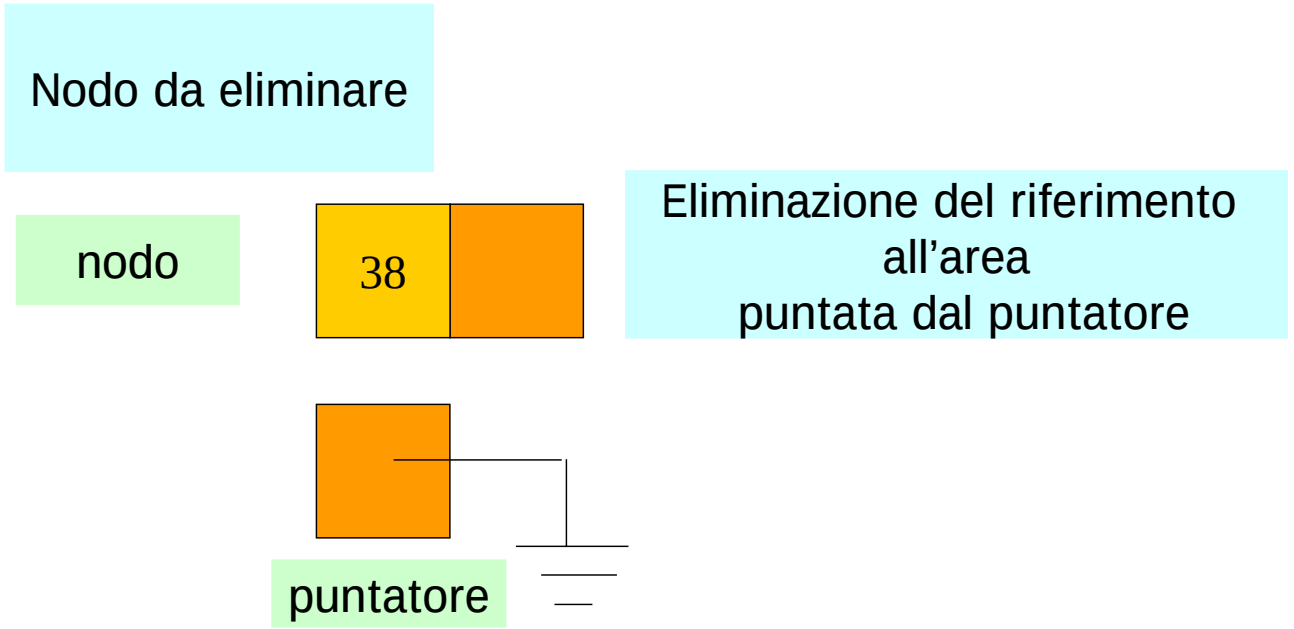
Inserimento in ordine nella lista

```
struct strutnodo *inseriscidopo(struct strutlista *ls,struct strutnodo *snprec,char *stringa) {
      struct strutnodo *sn;
      sn=allocanodo(stringa);
      if (sn) {
        sn->prossimo=snprec->prossimo;
        snprec->prossimo=sn;
      };
      return sn;
}
```

Cancellazione di un nodo in una lista



Rilascio della memoria di un nodo




```
void disallocanodo(struct strutnodo *sn) {
  if (sn->testo) free(sn->testo);
  free(sn);
}
```

Algoritmo di cancellazione di un nodo da una lista

```
void rimuovinodo(struct strutlista *ls,struct strutnodo *sn) {
     struct strutnodo *ptr,*prec;

     prec=0;
     ptr=ls->testa;
  if (ptr==sn) {
     ls->testa=0;
     disallocanodo(sn);
     return;
  };
     while((ptr!=sn)&&(ptr!=0)) {
       prec=ptr;
       ptr=ptr->prossimo;
     };
     if (ptr) {
       prec->prossimo=ptr->prossimo;
  };
     disallocanodo(sn);
}
```

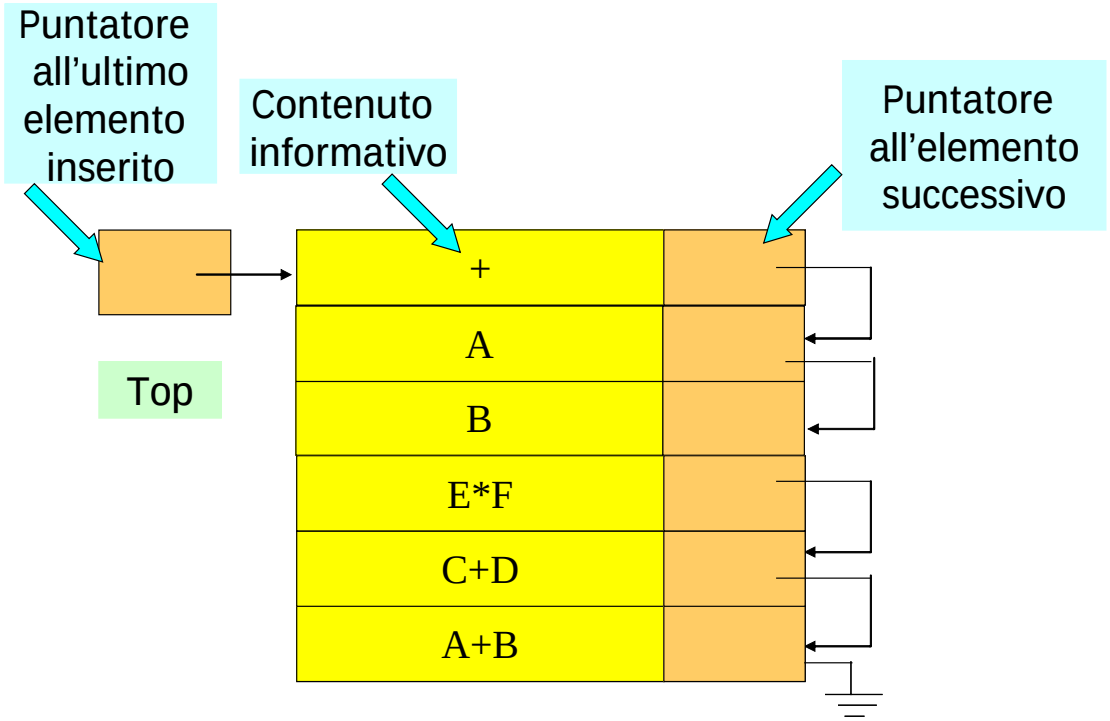
Algoritmo di ricerca in una lista

```
struct strutnodo *cerca(struct strutnodo *sn,char *stringa) {
      if (sn) {
        if (strcmp(stringa,sn->testo)==0) return sn;
        sn=cerca(sn->prossimo,stringa);
      };
      return sn;
}

struct strutnodo *scorri(struct strutnodo *sn) {
      if (sn) {
        printf("%s\n",sn->testo);
        sn=scorri(sn->prossimo);
      };
      return sn;
}
```

Realizzazione di una pila

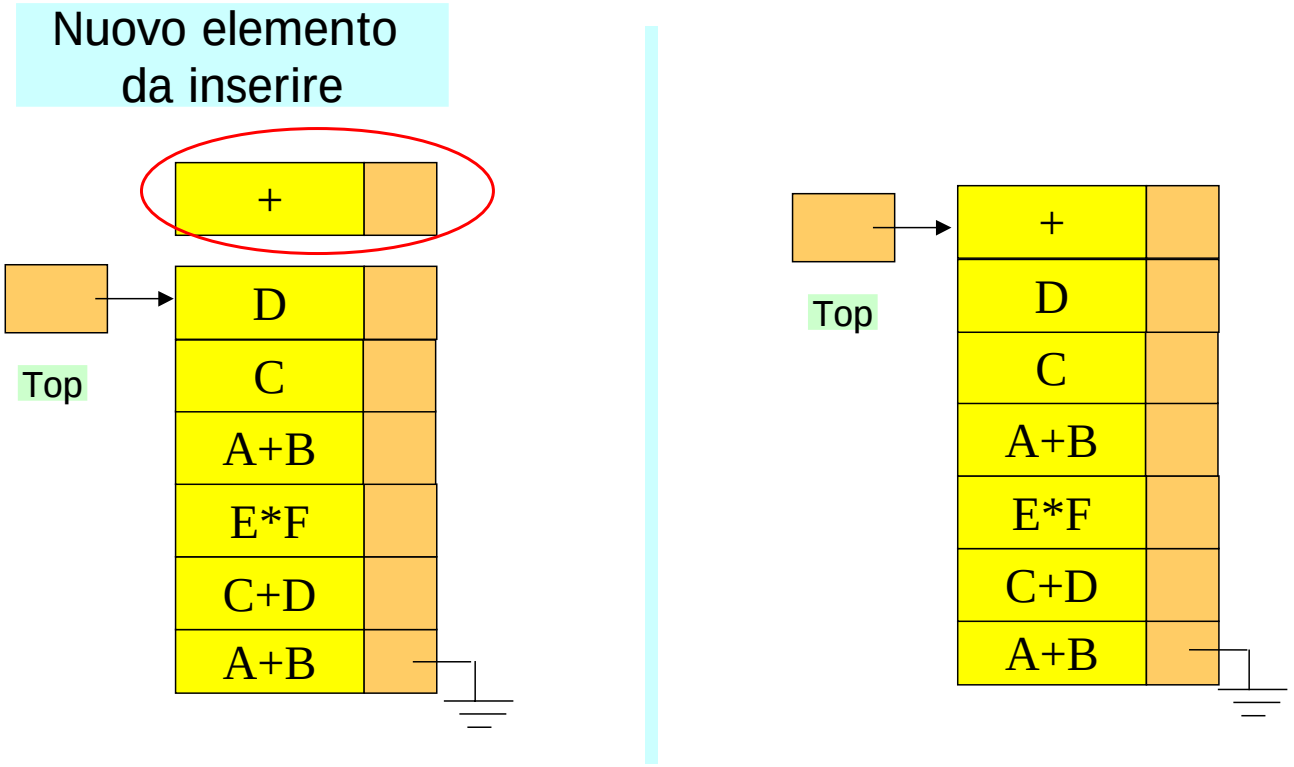
Il tipo degli elementi

```
struct strutpila {
        struct strutnodo *top;
};


struct strutnodo {
        struct strutnodo *prossimo;
        char                    *testo;
};
```



Inserimento di un nuovo elemento in una pila

PUSH



Allocazione della memoria per un nuovo elemento di una pila

PUSH

```
struct strutnodo *allocanodo(char *stringa) {
      struct strutnodo *sn;
      sn=malloc(sizeof(struct strutnodo));
      if (sn) {
        sn->testo=malloc(strlen(stringa)+1);
        if (sn->testo) {
              strcpy(sn->testo,stringa);
              return sn;
        } else {
              free(sn);
              sn=0;
        };
      };
      return sn;
}
```

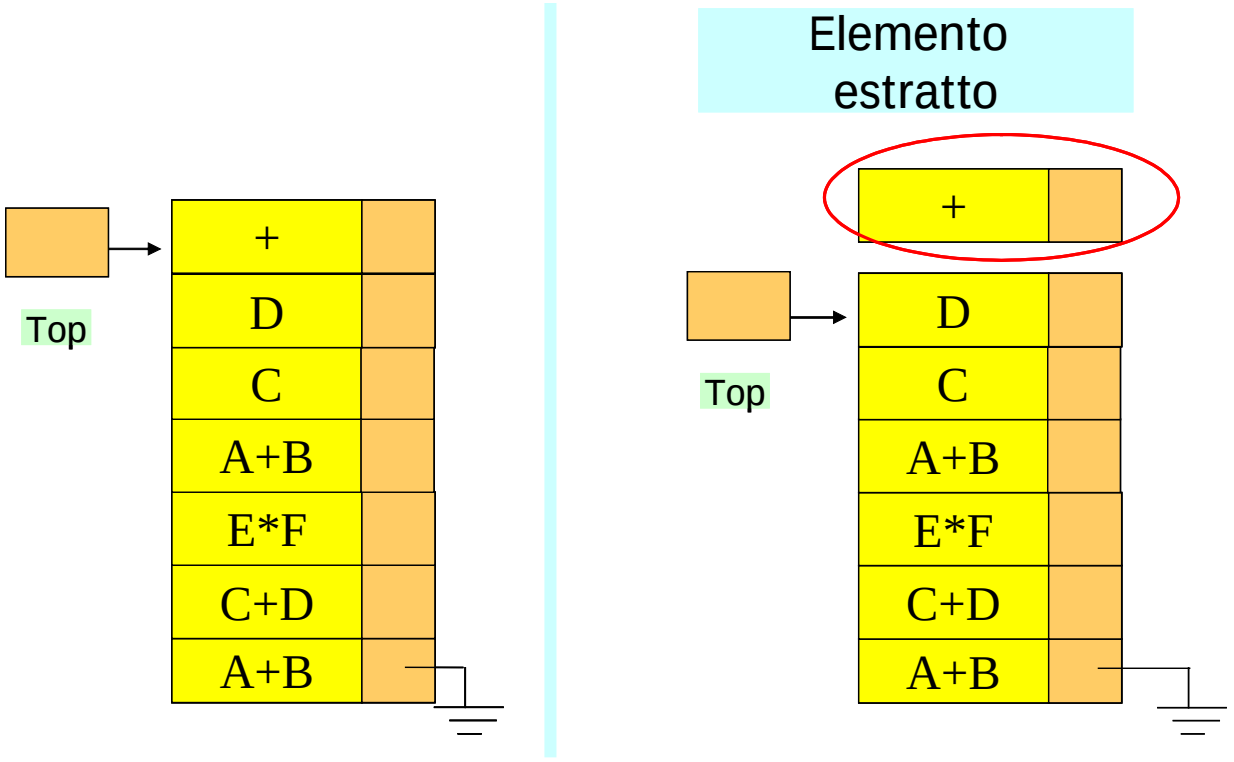
Algoritmo di inserimento in una pila

PUSH

```
void push(struct strutpila *pl,struct strutnodo *sn)
      {
      if (sn) {
        sn->prossimo=pl->top;
        pl->top=sn;
      };
}
```

Estrazione di un elemento da una pila

POP



Algoritmo di estrazione da una pila

POP

```
struct strutnodo *pop(struct strutpila *pl) {
   struct strutnodo *ptr;
   ptr=pl->top;
   if (ptr) pl->top=pl->top->prossimo;
   return ptr;
}
```

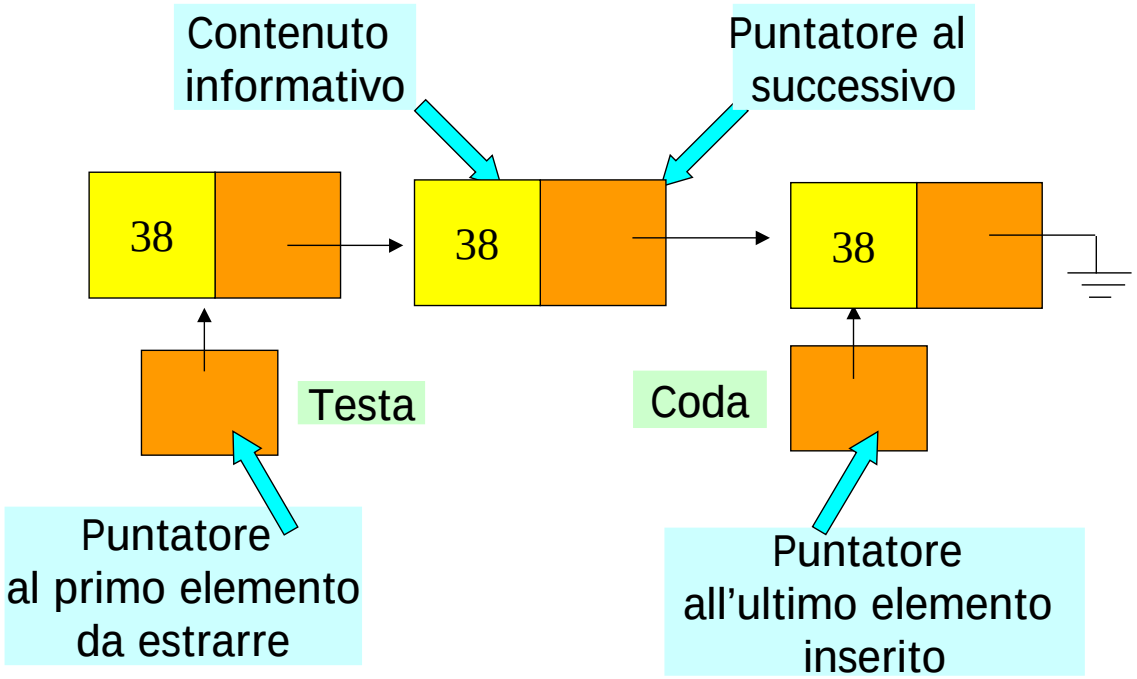
Deallocazione della memoria per estrazione

```
void disallocanodo(struct strutnodo *sn) {
   if (sn->testo) free(sn->testo);
   free(sn);
}
```

Realizzazione di una coda

Il tipo degli elementi

```
struct strutcoda {
      struct strutnodo *testa;
      struct strutnodo *coda;
};

struct strutnodo {
      struct strutnodo *prossimo;
      char                    *testo;
};
```



Allocazione della memoria per un elemento di una coda

```
struct strutnodo *allocanodo(char *stringa) {
      struct strutnodo *sn;
      sn=malloc(sizeof(struct strutnodo));
      if (sn) {
        sn->testo=malloc(strlen(stringa)+1);
        if (sn->testo) {
              strcpy(sn->testo,stringa);
              return sn;
        } else {
              free(sn);
              sn=0;
        };
      };
      return sn;
}
```

Algoritmo di inserimento di un elemento in una coda

```
void accoda(struct strutcoda *cd,struct strutnodo *sn) {
     if (sn) {
      if (cd->coda) cd->coda->prossimo=sn;
      sn->prossimo=0;
      cd->coda=sn;
      if (cd->testa==0) cd->testa=sn;
     };
}
```

Algoritmo di estrazione di un elemento da una coda

```
struct strutnodo *preleva(struct strutcoda *cd) {
      struct strutnodo *ptr;

   ptr=cd->testa;
   if (ptr) cd->testa=ptr->prossimo;
   return ptr;
}
```

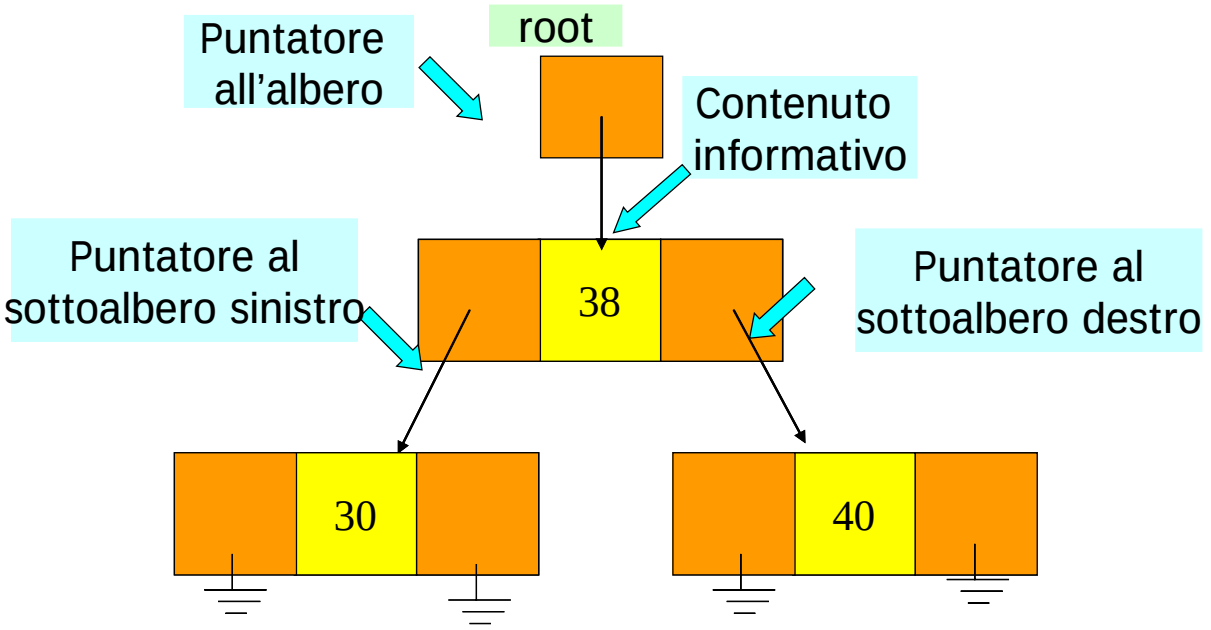
Deallocazione della memoria per estrazione

```
void disallocanodo(struct strutnodo *sn) {
   if (sn->testo) free(sn->testo);
   free(sn);
}
```

Realizzazione di un aLbero binario

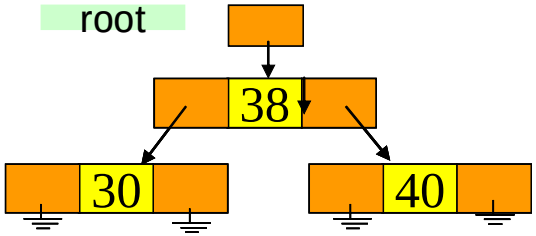
Il tipo degli elementi

```
struct strutnodo {
    struct strutnodo *destro;
    struct strutnodo *sinistro;
    char             *testo;
};
```

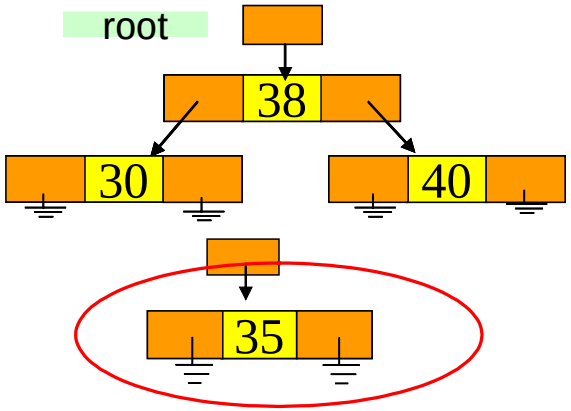


Inserimento di un nuovo nodo in un albero binario

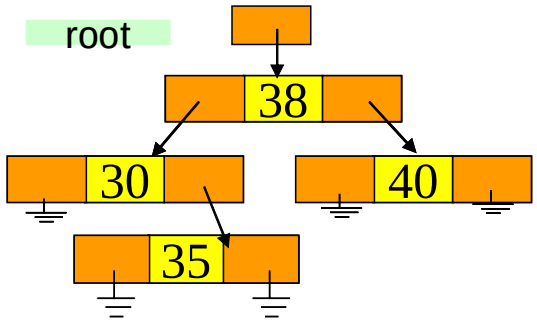
Ricerca del livello in cui inserire il nuovo nodo: 30<**35**<38



Allocazione del nuovo nodo



Collegamento del nuovo nodo



Allocazione della memoria per un nuovo nodo di un albero binario

```
struct strutnodo *allocanodo(char *stringa) {
      struct strutnodo *sn;
      sn=malloc(sizeof(struct strutnodo));
      if (sn) {
        sn->testo=malloc(strlen(stringa)+1);
        if (sn->testo) {
              strcpy(sn->testo,stringa);
                        sn->destro=0;
              sn->sinistro=0;
              return sn;
        } else {
              free(sn);
              sn=0;
        };
      };
      return 0;
}
```

Algoritmo di inserimento di un nodo in un albero binario

```
void inserisci(struct strutnodo **albero,struct strutnodo *nodo) {
     if (*albero==0) {
      *albero=nodo;
     } else {
      if (strcmp(nodo->testo,(*albero)->testo)>0) {
            inserisci(&((*albero)->destro),nodo);
      } else {
            inserisci(&((*albero)->sinistro),nodo);
      };
     };
}
```

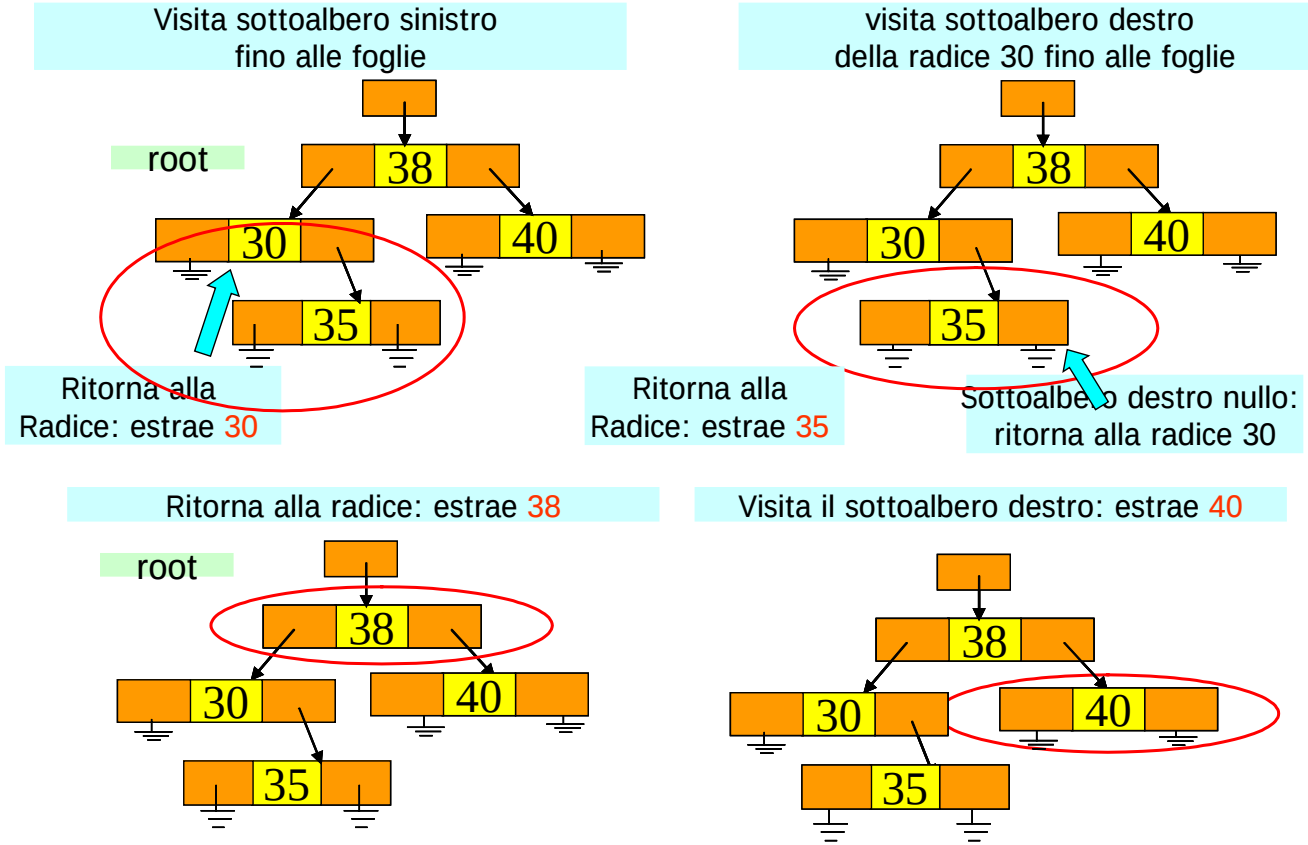
Algoritmi di cancellazione di un nodo e dell'intero albero

```
void rimuovinodo(struct strutnodo **albero,struct strutnodo *nodo)
{    if (*albero!=0)
   {   if (*albero==nodo)
         {   *albero=0;
             if (nodo->destro) inserisci(albero,nodo->destro);
             if (nodo->sinistro) inserisci(albero,nodo->sinistro);
             return;
         };
       rimuovinodo(&((*albero)->sinistro),nodo);
       rimuovinodo(&((*albero)->destro),nodo);
     };
 }

void distruggialbero(struct strutnodo **nodo)
{   if (*nodo)
   {  distruggialbero(&((*nodo)->sinistro));
      distruggialbero(&((*nodo)->destro));
      free(*nodo);
      *nodo=0;
     };
}
```

Deallocazione della memoria

```
void disallocanodo(struct strutnodo *sn) {
    if (sn->testo) free(sn->testo);
    free(sn);
}
```

Visita in-ordine di un albero binario



Algoritmi di visita in-ordine di un albero binario

```
void scorri_pre(struct strutnodo *nodo) {
     if (nodo) {
       printf("%s \n",nodo->testo);
       scorri_pre(nodo->sinistro);
       scorri_pre(nodo->destro);
     };
}
```

```
void scorri_in(struct strutnodo *nodo) {
     if (nodo) {
       scorri_in(nodo->sinistro);
       printf("%s \n",nodo->testo);
       scorri_in(nodo->destro);
     };
}
```

```
void scorri_post(struct strutnodo *nodo) {
     if (nodo) {
       scorri_post(nodo->sinistro);
       scorri_post(nodo->destro);
       printf("%s \n",nodo->testo);
     };
}
```

Algoritmo di ricerca in un albero

```
struct strutnodo *cercanodo(struct strutnodo *nodo,char *stringa) {
      struct strutnodo *sn;
      if (nodo) {
        if (strcmp(nodo->testo,stringa)==0) return nodo;
        if (sn=cercanodo(nodo->sinistro,stringa)) return sn;
        if (sn=cercanodo(nodo->destro,stringa)) return sn;
      };
      return 0;
}
```