Unità Didattica 6

Linguaggio C

Strutture dati non lineari: Alberi e Grafi

Definizione di Albero

- Un insieme A di elementi si dice <u>albero</u> se:

  $A = \varnothing$;

  oppure:

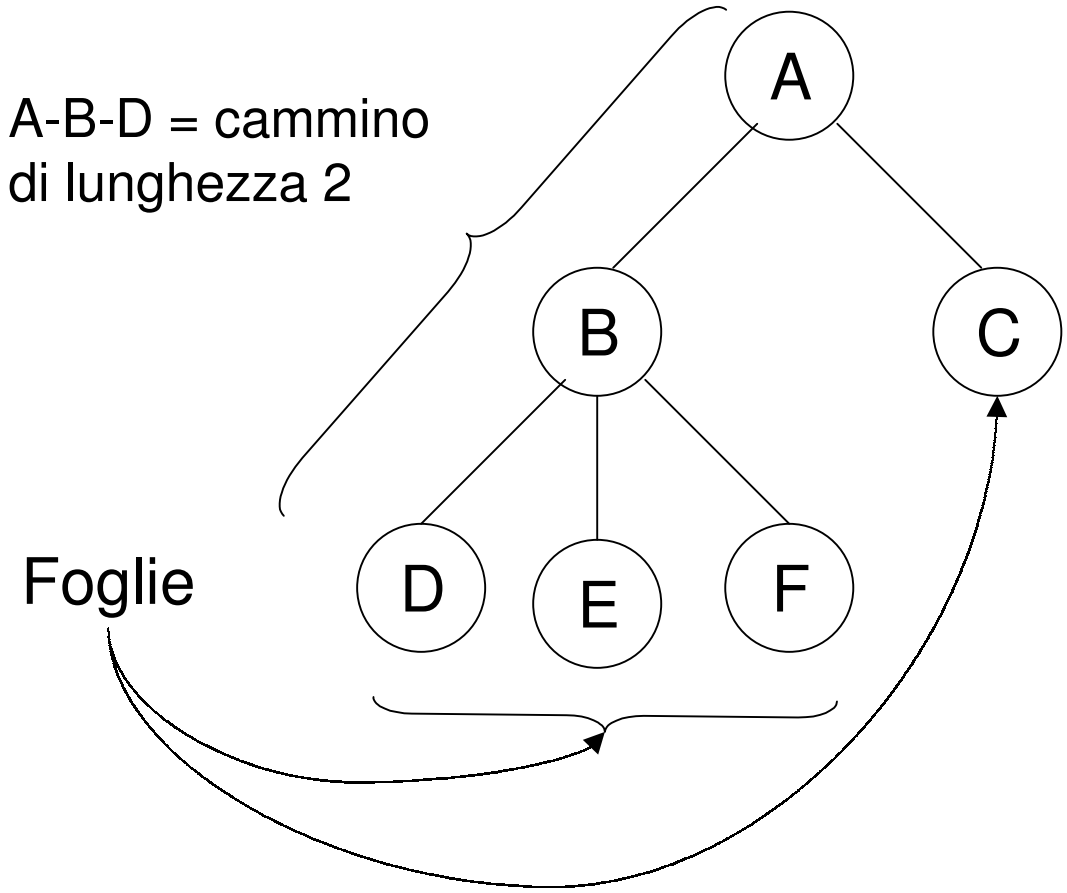
  $A = A(r, A_0, A_1, \ldots, A_{n-1})$; r = radice;

  A_i = albero,
  detto “sottoalbero i-esimo di A”;

- E’ una definizione ricorsiva:
  Struttura dati vuota, oppure costituita da un elemento detto radice e da altri elementi che formano insiemi disgiunti chiamati sottoalberi della radice, ciascuno dei quali è ancora un albero;

Nomenclatura sugli alberi

- Grado di un nodo: n° di sottoalberi di quel nodo;
- Foglia o nodo terminale: nodo di grado 0;
- Padre: nodo non terminale da cui discendono i suoi sottoalberi;
- Figli: nodi che discendono da uno stesso padre;
- Radice: nodo che non discende da altri nodi.
- X è discendente di Y se $\exists\ x_0, x_1, x_{n-1}$: $Y = x_0\ x_1\ x_{n-1} = X$ e x_i è discendente diretto di x_{i-1}, tale successione si chiama cammino;
- Lunghezza del cammino = (n° nodi del cammino – 1);
- Livello di un nodo X: è *i* se il livello del padre di X è *i-1*, con la convenzione che la radice ha livello 0;
- Altezza = (livello massimo delle foglie + 1);



A = Radice,
Nodo di grado 2
e livello 0

C = Nodo di grado 0
e livello 1

B = Nodo di grado 3
e livello 1

D,E,F = Nodi di grado 0
e livello 2

Altezza dell'albero = (Livello max delle foglie + 1) = 3

Definizione di albero binario

- Un insieme A di elementi si dice albero binario se:

  $A = \varnothing$;

  oppure:

  $A = A(r, A_l, A_r)$; r = radice;

  A_l = albero binario detto “sottoalbero sinistro”;

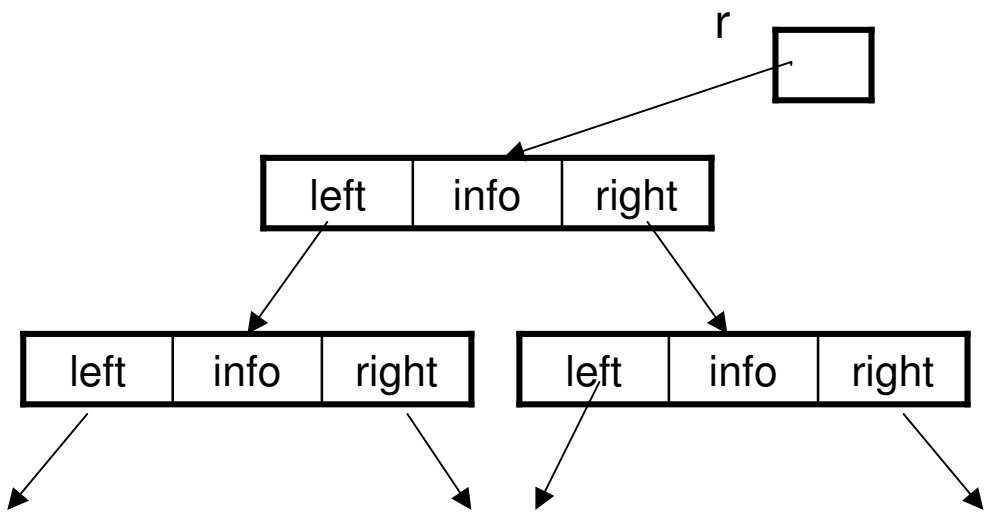
  A_r = albero binario detto “sottoalbero destro”;

- Struttura dati vuota, oppure costituita da un elemento detto radice e da due elementi disgiunti che sono a loro volta alberi binari detti sottoalbero sinistro e destro;
- Albero binario completo: albero nel quale le foglie sono distribuite al più su due livelli consecutivi *k-1* e *k*, e il numero di nodi a livello *k-1* deve essere 2^{k-1};

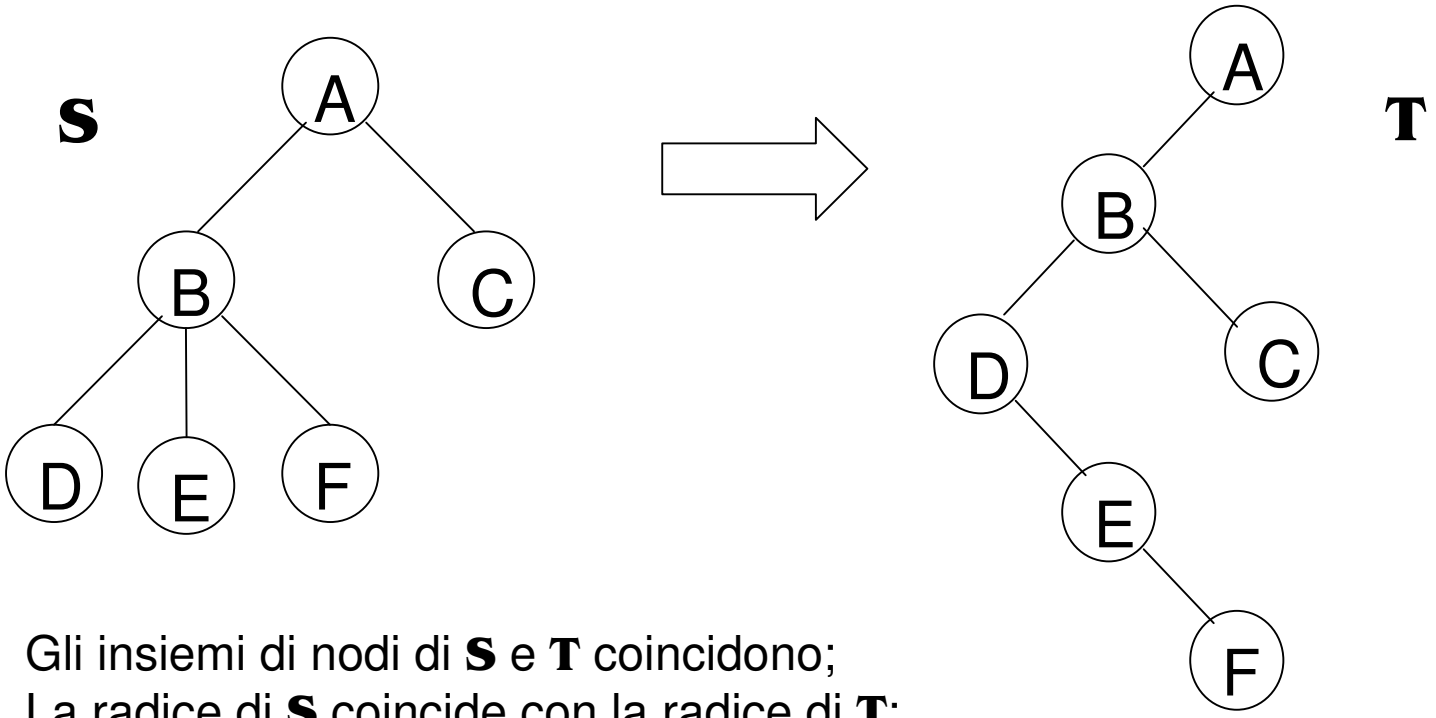
Realizzazione concreta di un albero binario

- Mediante strutture concatenate contenenti
  - Un campo informazione;
  - Due campi puntatore ai nodi radice del sottoalbero sinistro e destro.

```
struct nodo
{
   tipoinfo info;
   struct nodo* left;
   struct nodo* right;
}
struct nodo *r;
```



Trasformazione di alberi in alberi binari



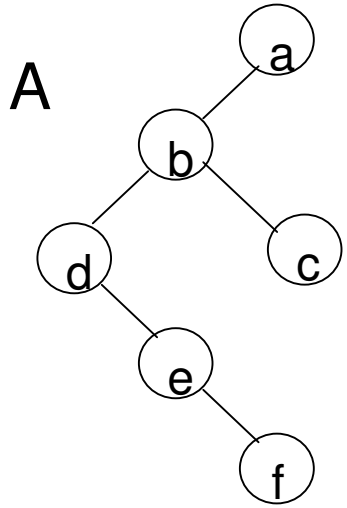
1) Gli insiemi di nodi di **S** e **T** coincidono;
2) La radice di **S** coincide con la radice di **T**;
3) Ogni nodo *n* di **T** ha come radice del sottoalbero sinistro il primo figlio di *n* in **S** e come radice del sottoalbero destro il fratello successivo a destra di *n* in **S**. Se *n* non ha i figli (fratelli a destra) in **S**, il sottoalbero sinistro (destro) di *n* in **T** è vuoto.

Visita di alberi binari

- Attraversamento di tutti i nodi di un albero binario, secondo particolari criteri:

1. Visita anticipata (radice, sottoalb. sinistro, sottoalb. destro);
2. Visita simmetrica (sottoalb.sinistro, radice, sottoalb. destro);
3. Visita differita (sottoalb. sinistro, sottoalb. destro, radice).

Visita Anticipata



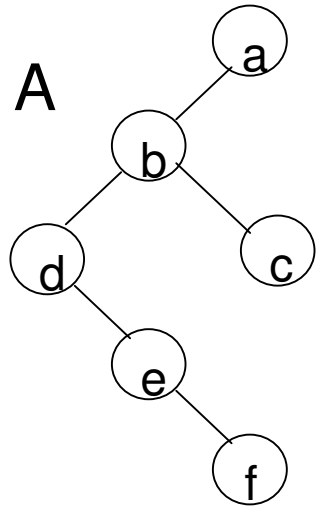
- visita_anticipata(A)
  1. Passo base:
     - se A = $\varnothing \Rightarrow$ esci;
  2. Passo di induzione:
     - si visita la radice;
     - visita_anticipata(A_l) //A_l sottoalbero sinistro;
     - visita_anticipata(A_r) //A_r sottoalbero destro;

- Risultato visita anticipata: *abdefc*

Visita Anticipata (C)

```
void visita_anticipata(struct nodo *r)
{
   if (r == NULL)
       return;
    printf("%s\n", r->info);
    visita_anticipata(r->left);
    visita_anticipata(r->right);
}
```

Visita Simmetrica



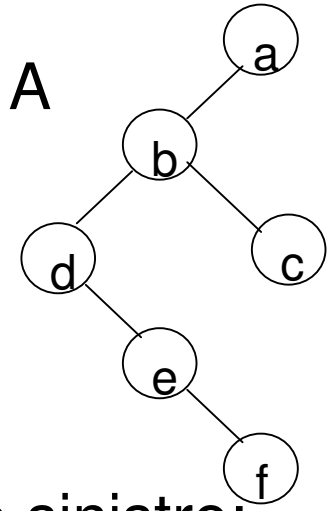
- visita_simmetrica(A)
  1. Passo base:
     - se A = $\varnothing \Rightarrow$ esci;
  2. Passo di induzione:
     - visita_simmetrica(A_l) //A_l sottoalbero sinistro;
     - si visita la radice;
     - visita_simmetrica(A_r) //A_r sottoalbero destro;

- Risultato visita simmetrica: *defbca*

Visita Simmetrica (C)

```
void visita_simmetrica(struct nodo *r)
{
   if (r == NULL)
       return;
    visita_simmetrica(r->left);
    printf("%s\n", r->info);
    visita_simmetrica(r->right);
}
```

Visita Differita



- visita_differita(A)
  1. Passo base:
     - se A = $\varnothing \Rightarrow$ esci;
  2. Passo di induzione:
     - visita_differita(A_l) //A_l sottoalbero sinistro;
     - visita_differita(A_r) //A_r sottoalbero destro;
     - si visita la radice;

- Risultato visita differita: *fedcba*

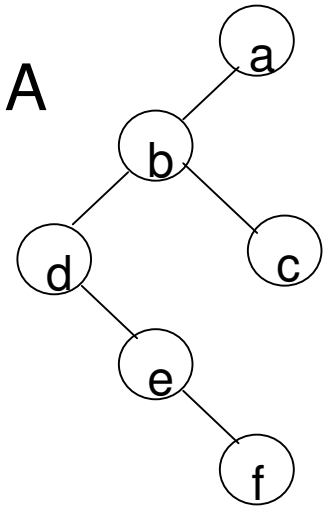
Visita Differita (C)

```
void visita_differita(struct nodo *r)
{
   if (r == NULL)
       return;
    visita_differita(r->left);
    visita_differita(r->right);
    printf("%s\n", r->info);
}
```

Altezza albero binario

- altezza_albero (A)
  1. Passo base:
     - se A = $\varnothing \Rightarrow$ restituisci 0;
  2. Passo di induzione:
     - restituisci

$$\text{altezza} = 1 + \max(\text{altezza\_albero}(A_r), \text{altezza\_albero}(A_l));$$



- altezza_albero (A) $\Leftrightarrow$ 5

Altezza albero binario (C)

```
int altezza_albero(struct nodo *r)
{
   if (r == NULL)            //Passo base
       return 0;

                        //Passo induzione
   return (1 + max(altezza_albero(r->left),
                   altezza_albero(r->right));
}
```

Albero binario di ricerca

Definizione:

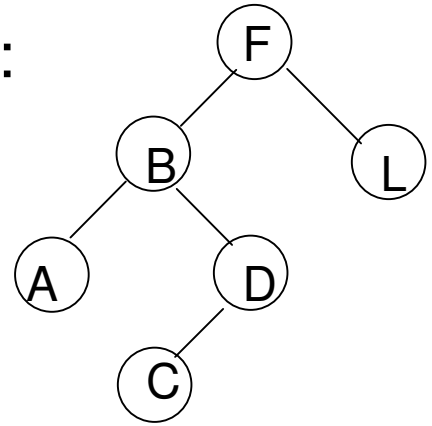
Un albero binario si dice di ricerca se è vuoto

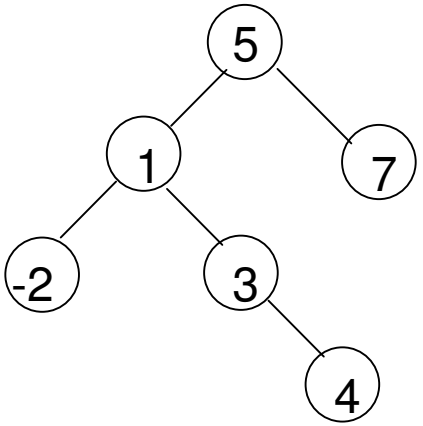
oppure:

è costituito da una radice e due sottoalberi sinistro e destro tali:

- gli elementi del sottoalbero sinistro sono “minori” della radice;
- gli elementi del sottoalbero destro sono “maggiori” della radice;

Es:





Costruzione di un albero binario

```
struct nodo
{
      tipoinfo info;
      struct nodo *left;
      struct nodo *right;
};
typedef struct nodo *p_n;

p_n inserisci_nodo(p_n root, tipoinfo x)
{
   if (root == NULL)
   {
          root = (struct nodo *)malloc(sizeof(struct nodo));
          root->info = x;
          root->left = NULL; root->right = NULL;
   }
   else
   {
      if (x < root->info)
          root->left = inserisci_nodo(root->left, x);
      else
          root->right = inserisci_nodo(root->right, x);
    }
   return root;
}
```

Grafi

- Grafo diretto o orientato:
  - Insieme N di elementi detti nodi e insieme di archi. Un arco è una coppia ordinata (v,w) di nodi (v,w $\in$ N);
- Grafo non orientato:
  - Insieme N di nodi e insieme di archi. Un arco è una coppia non ordinata (v,w) di nodi (v,w $\in$ N);

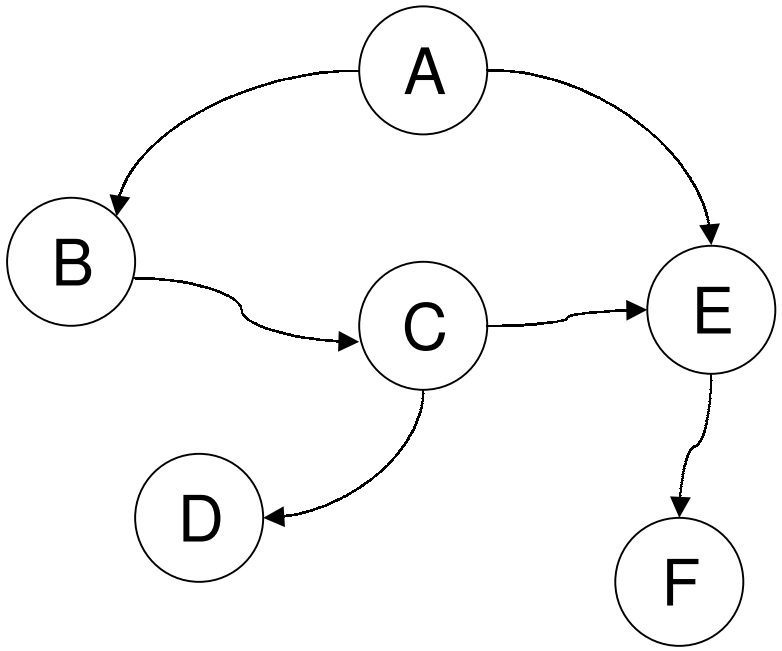
Nomenclatura sui grafi

- Se $(v,w) \in N$ sono collegati da un arco, si dice che <u>v è adiacente a w</u>;

- <u>Cammino di un grafo</u>: sequenza di nodi $v_0, v_1, \ldots, v_{n-1}$, tali che $(v_0, v_1), (v_1, v_2), \ldots, (v_{n-2}, v_{n-1})$ sono archi;

- <u>Lunghezza del cammino</u>: (n° di archi del cammino), oppure (n° nodi del cammino – 1);

- <u>Cammino semplice</u>: cammino in cui tutti i nodi, eccetto eventualmente il primo e l'ultimo, sono distinti;



ABCD = cammino semplice

Lungh(ABCD) = 3

Realizzazioni concrete di un grafo

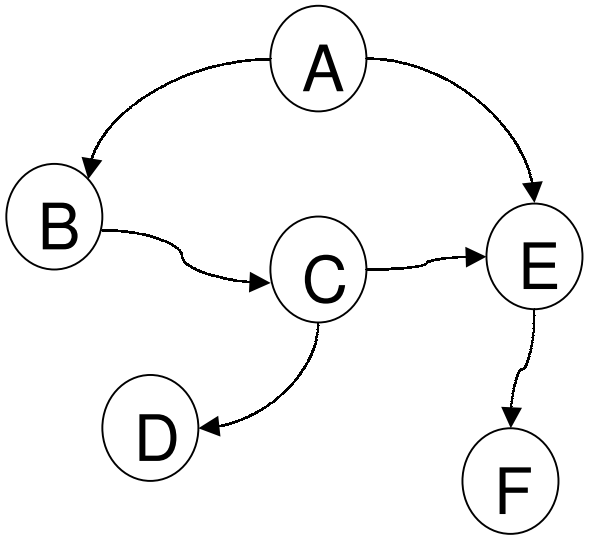
- Matrice di adiacenza $A = A(a_{ij})$
  - $a_{ij} = 1 \Leftrightarrow$ nodo *i* è connesso a nodo *j*
  - $a_{ij} = 0 \Leftrightarrow$ nodo *i* non è connesso a nodo *j*

  Proprietà:

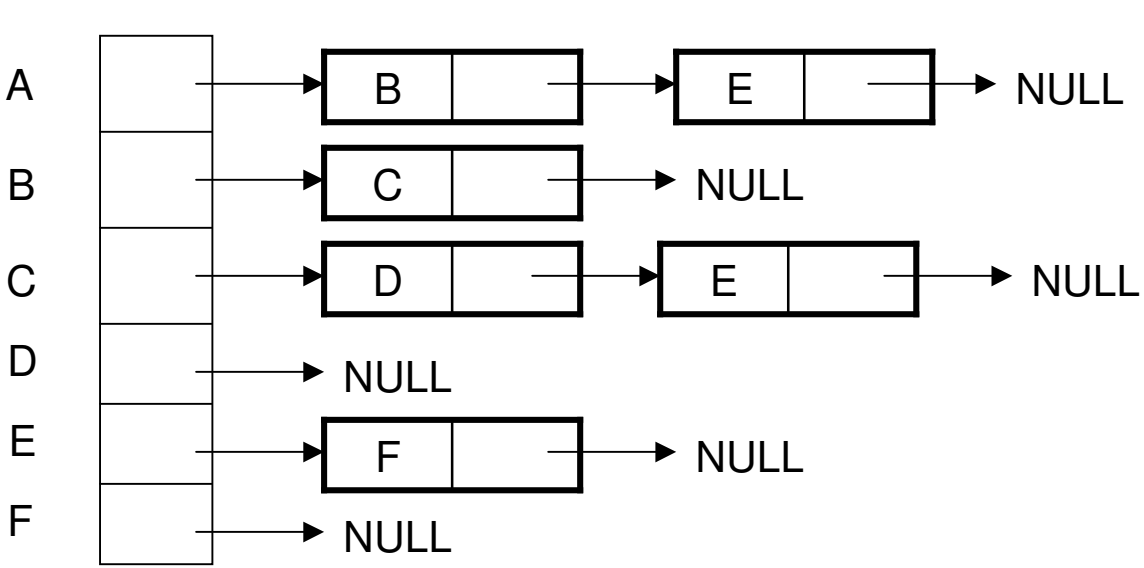
  la matrice di adiacenza di un grafo non diretto è simmetrica
- Liste di adiacenza:
  - Ogni nodo è descritto da una lista di elementi contenenti il nome dei nodi ad esso adiacenti;
- Plessi:
  - Struttura concatenata in cui ogni elemento ha un campo informazione e un puntatore ad una lista di puntatori ai nodi adiacenti*;*

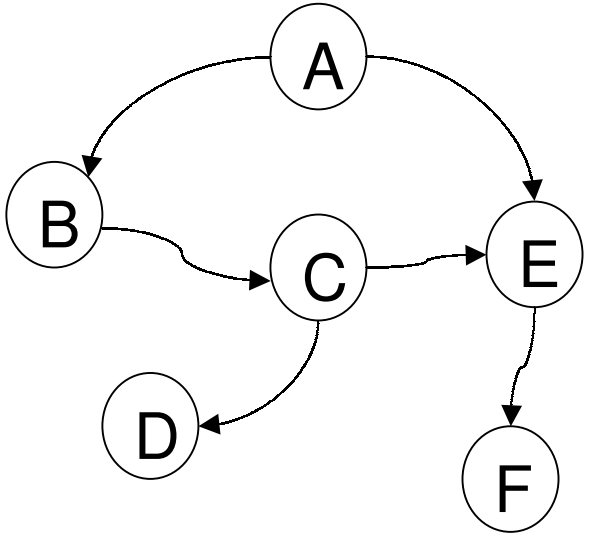
Realizzazione mediante Matrice di Adiacenza

	A	*B*	*C*	*D*	*E*	*F*
A		1			1	
B			1			
C				1	1	
D						
E						1
F						



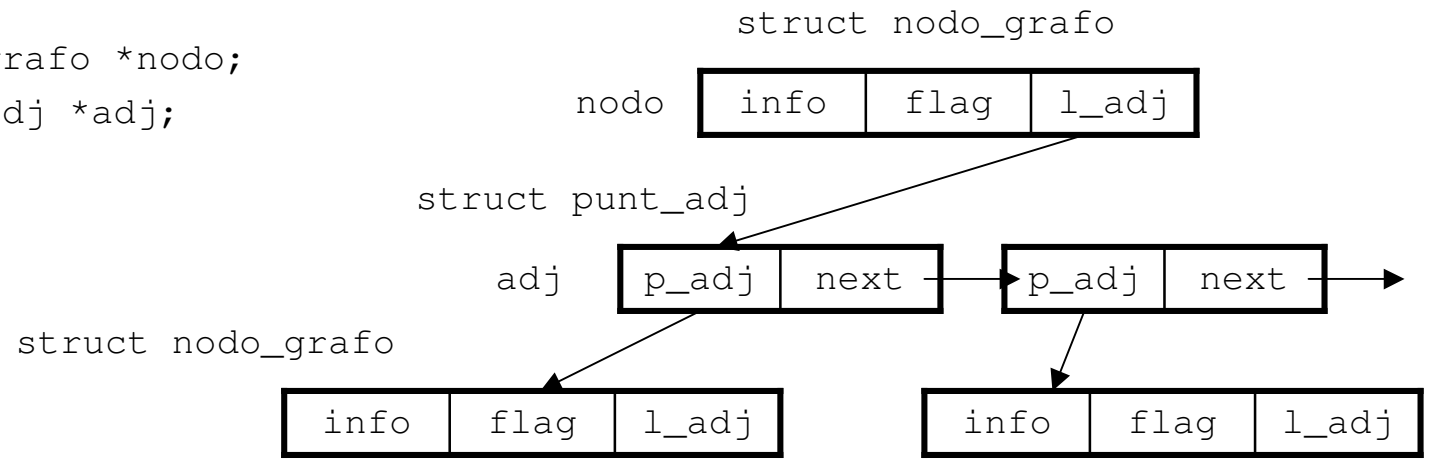
Realizzazione mediante Liste di Adiacenza





Realizzazione mediante plessi

```
struct nodo_grafo    //elemento nodo di un grafo
{
   tipoinfo info;
   int flag;
   struct punt_adj *l_adj;
};
struct punt_adj     //elemento lista
                    //puntatori a nodo di un grafo
{
   struct nodo_grafo *p_adj;
   struct punt_adj *next;
};
struct nodo_grafo *nodo;
struct punt_adj *adj;
```



Algoritmi di visita per i grafi

- Si dice visita di un grafo un insieme di cammini con origine in uno stesso vertice;

  ovvero:

  Attraversamento di tutti i nodi di un grafo secondo un particolare criterio;

- Si distinguono due tipi di visita di un grafo:
  - In profondità;
  - In ampiezza.

Grafi: visita in profondità

- visita_profondità(G)
  1. Passo base
     se G = Ø ⇒ esci;
  2. Passo di induzione
     visita il nodo G se non è stato visitato
     per ogni nodo adiacente G->*adj*
     visita_profondità(G->*adj*);

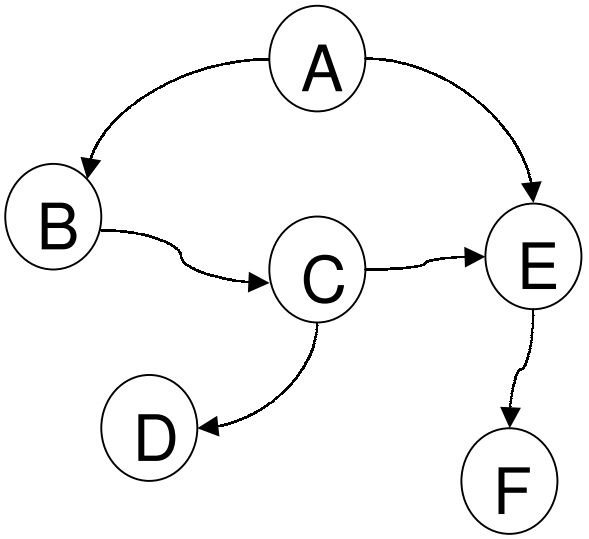
- *Osservazione:* la visita in profondità è strettamente legata alla politica di gestione di uno stack (come tutti gli algoritmi ricorsivi);

Visita in profondità (C)

```
void visita_profondita(struct nodo_grafo *nodo)
{

  if (nodo == NULL) return;

  if (nodo->flag == 0)
  {
    printf("%s\n", nodo->info);
    nodo->flag = 1;
    adj = nodo->l_adj;
    while(adj != NULL)
    {
          visita_profondita(adj->p_adj);
          adj = adj->next;
    }
  }
  return;

}
```

Risultato visita in profondità

ABCDEF



Altra formulazione della visita in profondità

- visita_profondità(G)

  per ogni nodo non visitato:

  - si inserisce il nodo in uno stack;
  - si estrae dallo stack un nodo e si visita;
  - si inseriscono nello stack tutti gli elementi adiacenti al nodo corrente.

Grafi: visita in ampiezza

- visita_ampiezza(G)

  per ogni nodo non visitato:
  - si inserisce il nodo in una coda;
  - si estrae dalla coda un nodo e si visita;
  - si inseriscono in coda tutti gli elementi adiacenti al nodo corrente.

- *Osservazione:* la visita in ampiezza è strettamente legata alla politica di gestione di una coda;

Visita in ampiezza (C)

```
void visita_ampiezza(struct nodo_grafo *nodo)
{
  struct punt_adj *adj;
  static struct p_coda *first_el;

  coda_ins(nodo);

  while(first_el != NULL)  //finché ci sono elementi nella coda
  {
    nodo = coda_extr();                  //Estrazione dalla coda di un nodo
    printf("nodo->info %c\n", nodo->info);  //Visita del nodo
    nodo->flag = 1;
    adj = nodo->l_adj;

    while(adj != NULL)   //Inserimento in coda dei nodi adiacenti
    {
          if (adj->nodo_adj->flag == 0)
                    coda_ins(adj->p_adj);
          adj = adj->next;
    }
  }
}
```

Risultato visita in ampiezza

ABECFD