LISTA

PRIMITIVA (lista = elem*)	DESCRIZIONE
tipo_inf head(lista l)	Restituisce la testa di *l* (valore dentro a *inf*)
lista tail(lista l)	Restituisce la coda di *l* (indirizzo contenuto in *pun*)
lista insert_elem(lista l, elem e)*	Aggiunge *e* ad *l* e ritorna la lista aggiornata
lista delete_elem(lista l, elem e)*	Elimina *e* da *l* e ritorna la lista aggiornata
lista search(lista l, tipo_inf v)	Cerca in *l* il valore *v* e restituisce il puntatore all'elemento che contiene *v*, se esiste, *NULL* altrimenti
lista copy(lista l1)	Per copiare una lista, ritorna una lista uguale alla lista passata in ingresso

```
struct elem {
int inf;
elem* pun;
};
```

LISTA DOPPIA

PRIMITIVA (lista = elem*)	DESCRIZIONE
tipo_inf head(lista l)	Restituisce la testa di *l* (valore dentro a *inf*)
lista tail(lista l)	Restituisce la coda di *l* (indirizzo contenuto in *pun*)
lista prev(lista l)	Restituisce l'indirizzo dell'elemento precedente
lista insert_elem(lista l, elem e)*	Aggiunge *e* ad *l* e ritorna la lista aggiornata
lista delete_elem(lista l, elem e)*	Elimina *e* da *l* e ritorna la lista aggiornata
lista search(lista l, tipo_inf v)	Cerca in *l* il valore *v* e restituisce il puntatore all'elemento che contiene *v*, se esiste, *NULL* altrimenti
lista copy(lista l1)	Copia una lista, ritorna una lista uguale alla lista passata in ingresso

```
struct elem {
int inf;
elem* pun; // punt. al prossimo elem
elem* prec; // punt. al precedente elem
} ;
```

ALBERO

PRIMITIVA (tree = node*)	DESCRIZIONE
node new_node(tipo_inf i)*	Crea un nuovo nodo con valore informativo *i*
void insert_child(tree p, tree c)	Aggiorna *p* inserendo il sottoalbero radicato in *c* come primo figlio di *p*
void insert_sibling(node n, tree t)*	Aggiorna *n* inserendo il sottoalbero radicato in *t* come fratello successivo di *n*
tipo_inf get_info(node n)*	Restituisce il contenuto informativo del nodo *n*
tipo_inf get_parent(node n)*	Restituisce il padre del nodo *n*
tipo_inf get_firtsChild(node n)*	Restituisce il primo figlio del nodo *n*, se esiste
tipo_inf get_nextSibling(node n)*	Restituisce il fratello successivo del nodo *n*, se esiste

```
struct node {
    tipo_inf inf;
    node* parent; //opzionale
    node* firstChild;
    node* nextSibling;
};
```

CODA-BFS

PRIMITIVA	DESCRIZIONE
codaBFS enqueue(codaBFS c, tipo_inf i)	Crea ed inserisce un nuovo elemento *i* in fondo alla coda, ritorna coda aggiornata
tipo_inf dequeue(codaBFS& c)	Rimuove e ritorna l'elemento in testa dalla coda
tipo_inf first(codaBFS c)	Ritorna il valore *inf* dell'elemento in testa
bool isEmpty(codaBFS c)	Verifica se la coda è vuota (*TRUE* vuota, *FALSE* non vuota)
codaBFS newQueue()	Inizializza la coda settando a *NULL* i campi *inf* e *pun*
static elemBFS new_elem(tipo_inf i)*	Crea un nuovo elemento con valore informativo *i*, usata all'interno di *enqueue()*

```
struct elemBFS
{
      tipo_inf inf;
      elemBFS* pun ;
};

typedef elemBFS* lista;

typedef struct{
      lista head;
      elemBFS* tail;} codaBFS;
```

ALBERO BINARIO DI RICERCA (BST)

PRIMITIVA (bnode* = bst)	DESCRIZIONE
bnode bst_newNode(tipo_key, tipo_inf)*	Crea un nuovo nodo con chiave e valore informativo dati in ingresso
tipo_key get_key(bnode)*	Restituisce la chiave del nodo in ingresso
tipo_inf get_value(bnode)*	Restituisce il valore del nodo in ingresso
bst get_left(bst)	Restituisce il sottoalbero sinistro dell'albero in ingresso
bst get_right(bst)	Restituisce il sottoalbero destro dell'albero in ingresso
bnode get_parent(bnode*)*	Restituisce il padre dell'albero in ingresso
void bst_insert(bst&, bnode)*	Aggiunge un nodo all'albero di ricerca
void bst_delete(bst&, bnode)*	Cancella un nodo dall'albero di ricerca
bnode bst_search(bst, tipo_key)*	Restituisce il nodo associato alla chiave in ingresso, se esiste

```
struct bnode {
      tipo_key key;
      tipo_inf inf;
      bnode* left;
      bnode* right;
      bnode* parent;};
typedef bnode* bst;
```

GRAFO CON LISTA DI ADIACENZA	
PRIMITIVA	**DESCRIZIONE**
graph new_graph(int n)	Restituisce la rappresentazione di un grafo in *n* vertici identificati univocamente da 1 a *n* attraverso *n* liste di adiacenza
void add_arc(graph& g, int s, int d, float w)	Aggiunge l'arco orientato *(s,d)* con peso *w* alla lista di adiacenza del nodo *s*
void add_edge(graph& g, int s, int d, float w)	Aggiunge l'arco non orientato *(s,d)* con peso *w* alla lista di adiacenza del nodo *s* e del nodo *d*
int get_dim(graph)	Restituisce il numero *n* dei nodi del grafo
adjlist get_adjlist(graph, int)	Restituisce la *testa* della lista di adiacenza del nodo con identificativo in ingresso
int get_adjnode(adj_node)*	Restituisce l'identificativo del nodo contenuto nell'elemento della lista di adiacenza
adj_list get_nextadj(adj_list)	Restituisce il prossimo elemento della lista di adiacenza

```
struct adj_node{
  int node;
  float weight;
  struct adj_node* next;
};

typedef adj_node* adj_list;

typedef struct{
  adj_list* nodes;
  int dim;
} graph;
```