#Visita DFS

```
void serialize(tree t){
   cout<<"(";
   print(get_info(t));
   tree t1 = get_firstChild(t);
   while(t1!=NULL){
      serialize(t1);
      t1 = get_nextSibling(t1);
   }
   cout<<")";
}
```

#Visita BFS

```
int dimensione(tree t){
   int count=0;
   codaBFS c = newQueue();
   c=enqueue(c,t);
   while(!isEmpty(c)){
      node* n=dequeue(c);
      count++;                          #visita nodo
      tree t1 = get_firstChild(n);
      while(t1!=NULL){
            c=enqueue(c,t1);
            t1 = get_nextSibling(t1);
        }
   }
   return count;                        #facoltativo

}

void bst_insert(bst& b, bnode* n){
   bnode* x;
   bnode* y=NULL;
   if(b==NULL)
        b=n;
   else{
        x=b;
        while (x != NULL) {
            y=x;
            if (compare_key(get_key(n),get_key(x))<0)
                 x = get_left(x);
            else
                 x = get_right(x);
         }
         n->parent = y;
         if (compare_key(get_key(n), get_key(y))<0)
              y->left = n;
          else
              y->right = n;
}}



bnode* bst_search(bst b,tipo_key k){
      (if b==NULL return 0 ric)
       while (b != NULL) {
         if (k ==get_key(b))
            return b;
         if  (k<get_key(b) )
            b = get_left(b);
         else
            b = get_right(b);
       }
       return NULL;
}
```

```
void print_BST(bst b){
   if(get_left(b)!= NULL)
      print_BST(get_left(b));

   print_key(get_key(b));
   cout<<"  ";
   print(get_value(b));
   cout<<endl;

   if(get_right(b)!= NULL)
       print_BST(get_right(b));

}



lista sposta(lista& l, int soglia){
   lista l_temp = l;
   lista precedente = NULL;
   lista precedente_pari = NULL;
   lista l2 = NULL;

   while(l_temp != NULL){
      if(head(l_temp) < soglia){
         if(precedente == NULL)
            l = tail(l);
         else
            precedente->pun = tail(l_temp);
         if(l2 == NULL)
            l2 = precedente_pari = l_temp;
         else{
            precedente_pari->pun = l_temp;
            precedente_pari = l_temp;
         }
      }
      else
         precedente = l_temp;

      l_temp = tail(l_temp);
   }
   precedente_pari->pun = NULL;
   return l2;
}



bool same(btree b1, btree b2){
   if(b1== NULL || b2 == NULL)
      if( b1 == NULL && b2 == NULL)
            return true;
      else
            return false;
  else {
     bool left = same(b1->left,b2->left);
     bool right = same(b2->right, b2->right);
     return b1->inf == b2 -> inf && left && right;}}
```

Scorrimento adj list

```
for(i=1;i<=get_dim(g);i++){
   adj_list l=get_adjlist(g,i);
   while(l!=NULL){
        v[i-1]++;
        l=get_nextadj(l);
    }
}

bool path(node* n, tipo_inf v){
```

```
   if(compare(get_info(n),v)==0)
     return true;
   tree t1 = get_firstChild(n);
   bool ris = false;
   while(t1!=NULL&&!ris){
      ris = path(t1,v);
      if (!ris)
         t1 = get_nextSibling(t1);
   }
return ris;

int altezza(tree t){
    if(get_firstChild(t)==NULL)
        return 0;
    int max=0,max_loc;
    tree t1 = get_firstChild(t);
    while(t1!=NULL){
          max_loc=altezza(t1);
          if(max_loc>max)
               max=max_loc;
               t1 = get_nextSibling(t1);
    }
    return max+1;
}
bool even_path(graph g, int x, int y){
  adj_list l = get_adjlist(g, x);
  if(l == NULL)
    return false;
  do{
    if(get_adjnode(l) == y)
      return true;
    if(get_adjnode(l)%2 == 0){
      if(even_path(g, get_adjnode(l), y))
        return true;
    }
    l = get_nextadj(l);
  } while (l != NULL);
  return false;
}

bool path(nodo n1, user u, graph g){
    bool* raggiunto= new int[g.dim];
    for(int i=0;i<g.dim;i++)
        raggiunto[i]=false;
    coda c=newQueue();
    c=enqueue(c,n1);
    while(!isEmpty(c)){
        nodo n=dequeue(c);
        if(!raggiunto[n.id]){
          if(strcmp(n.u,u)==0)
             return true;
          raggiunto[n.id]=true;
         adj_list* app=g.nodes[n.id];
         while(app!=NULL){
             c=enqueue(c,app);
             app=app->next;
        }
    }
}
```