```
int height(kTree t) {
    if(t == NULL) return 0;
    else if(t->child == NULL) return 0;
    else{
        int ht = 0;
        kTree c = t->child;
        while(c != NULL) {
            ht = max(ht, height(c));
            c = c->sibling;
        }
        return ht+1;
    }
}
```

```
int sumLeaf(kTree t) {
    if(t == NULL) return 0;
    else if(t->child == NULL) return t->key;
    else{   // t NON è una foglia
        int sum = 0;
        kTree c = t->child;
        while (c != NULL) {
            sum += sumLeaf(c);
            c = c->sibling;
        }
        return sum;
    }
}
```

```
list fringe(kTree t) {
    list l = NULL;
    if(t == NULL) return NULL;
    else if(t->child == NULL)   // t è
        return Cons(t->key, NULL);
    else{   // t NON è una foglia
        kTree c = t->child;
        while (c != NULL) {
            l = concat(l, fringe(c));
            c = c->sibling;
        }
    }
    return l;
}
```

```c
int cardinality(kTree t) {
    if(t==NULL) return 0;
    else{
        int card = 1;
        kTree c = t->child;
        while (c != NULL) {
            card += cardinality(c);
            c = c->sibling;
        }
        return card;
    }
}
```

```c
int degree(kTree t) {
    if(t==NULL) return 0;

    int num_degree = 0;
    kTree c = t->child;
    while (c!=NULL){
        num_degree = max(num_degree, degree(c));
        c=c->sibling;
    }
    return num_degree+1;
}
```

```c
int degreeProf(kTree t) {
    if(t==NULL) return 0;
    else if(t->child == NULL) return 0;
    else {
        kTree c = t->child;
        int droot = 0;
        int dt = 0;
        while(c != NULL) {
            dt = max(dt, degree(c));
            droot = droot + 1;
            c = c->sibling;
        }
        return max(dt, droot);
    }
}
```

```
list kTreeBFS(kTree t) {
    if(t == NULL) return NULL;
    list l = NULL;
    queue q = NewQueue();
    EnQueue(t, q);
    while (!isEmptyQueue(q)) {
        kTree node = DeQueue(q);
        l = Cons(node->key, l);
        node = node->child;
        while(node != NULL) {
            EnQueue(node, q);
            node = node->sibling;
        }
    }
    return reverse(l);
}
```

```
bool sum(kTree t) {
    if(t->child == NULL) return true;
    kTree c = t->child;
    int res = 0;
    bool b = true;
    while (c != NULL){
        res += c->key;
        b = b & sum(c);
        c=c->sibling;
    }
    return t->key == res && b;
}
```

```c
void sommaCammino(kTree t, int s){
    if(t->child == NULL){
    t->child = consTree(s + t->key, NULL, NULL);
    }else{
        s+=t->key;
        kTree c = t->child;
        while (c != NULL) {
            sommaCammino(c, s);
            c=c->sibling;
        }
    }
}
void sommaRamo(kTree t){
    sommaCammino(t,0);
}
```

```c
int nodiProfondi(kTree t, int h) {
    if(h == 0) return 1;
    else{
        int n = 1;
        kTree c = t->child;
        while (c!=NULL){
            n += nodiProfondi(c, h-1);
            c = c->sibling;
        }
        return n;
    }
}
```

```c
int maxSumBranch(kTree t) {
    if(t == NULL) return 0;
    else if(t->child == NULL) return t->key;
    else{
        int maxSum = 0;
        kTree c = t->child;
        while (c != NULL) {
            maxSum = max(maxSum, maxSumBranch(c));
            c=c->sibling;
        }
        return t->key + maxSum;
    }
}
```

```c
int shortest2(kTree t) {
    if(t==NULL) return 0;
    else if(t->child == NULL) return 1;
    else{
        int corto = 10000;
        kTree c = t->child;
        while (c!=NULL){
            int pathlength =
            corto = min(corto, shortest(c));
            c=c->sibling;
        }
        return corto+1;
    }
}
```

```c
kTree completeBedi(int key, int dg, int ht) {
    if(ht==0) return consTree(key, NULL, NULL);
    kTree child= complete(key, dg, ht-1);
    kTree temp = child;
    for(int i = 1; i<dg; i++){
        temp->sibling = complete(key, dg, ht-1);
        temp = temp->sibling;
    }
    return consTree(key, child, NULL);
}
```

```
int countInternalNodes(kTree t){
    if(t == NULL) return 0;
    else if(t->child == NULL) return 0;
    int sumInternalNodes = 0;
    kTree c = t->child;
    while(c != NULL){
        sumInternalNodes += countInternalNodes(c);
        c = c->sibling;
    }
    return sumInternalNodes+1;
}
```

```
int minimo(kTree t) {
    if (t == NULL)return NULL;
    if (t->child == NULL) return t->key;
    int fm = t->key;
    kTree currentChild = t->child;
    while (currentChild != NULL) {
        fm = min(fm, minimo(currentChild));
        currentChild = currentChild->sibling;
    }
    return fm;
}
```

```
bool isOdd(kTree t) {
    if(t==NULL) return false;
    else if(t->child == NULL){
        if(t->child->key % 2 == 1) // è
            return true;
        else // t->child->key % 2 == 0
            return false;
    }else{
        bool odd = true;
        kTree c = t->child;
        while (c!=NULL){
            odd = odd && isOdd(c);
            c=c->sibling;
        }
        return odd;
    }
}
```

```
int Large(kTree t) {
    if(t == NULL) return 0;
    queue q = NewQueue();
    EnQueue(t, q);
    int max = 0;
    int count=0;
    while (!isEmptyQueue(q)) {
        kTree current = DeQueue(q);
        kTree c = current->child;
        count = 0;
        while (c != NULL) {
            EnQueue(c, q);
            count += 1;
            c = c->sibling;
        }
        if(count > max) {
            max = count;
            count=0;
        }
    }
    return max +1;
}
```