```
//=======================================
//========1° ES.; soluz. Ricorsiva .======
//=======================================
int SeqUguali (int v[], int k, int start, int end) {
//PREC.: k >= 0 e start <= end
//POSTC.: restituisce 1 se ci sono almeno k numeri uguali in sequenza in v,
//0 altrimenti
assert(k >= 0);
assert(start <= end);
//start è il punto dell'array v da cui iniziare a cercare i k elementi
//consecutivi uguali;
//end è l'indice dell'ultimo el nel vettore v
if (k == 0 || k==1) return 1;
if (start == end) return 0;
if (v[start] == v[start+1]) return SeqUguali (v, k-1, start+1, end);
else return SeqUguali (v, k, start+1, end);
}


//=======================================
//=======1° ES.; soluz. Iterativa ========
//=======================================
/*qui n è il numero degli elementi del vettore*/
int SeqUguali (int v[], int k, int n) {
//PREC.: 0 <= k <= n
//POSTC.: restituisce 1 se ci sono almeno k numeri uguali in sequenza in v,
//0 altrimenti
int cnt = 1, i = 0;
assert(k >= 0);
assert( k <= n);
if (k == 0 ) return 1;
if (n == 0) return 0;
while (cnt < k && i < n-1){
    if (v[i] == v[i+1]) cnt++; else cnt=1;
    i++;
}
if (cnt == k) return 1; return 0;
}

//=======================================
//==========2° ES.; prima soluz.==========
//=======================================
int GV(TreePtr tPtr){
/* prec: l'albero in input è un albero binario i cui nodi son gialli
(contengono 1) o verdi (contengono 0).
postc: dà vero se il numero dei nodi gialli
è maggiore di quello dei verdi in ogni sottoalbero, falso altrimenti*/
if (!tPtr) return 0;
if (GVAux(tPtr) > 0) return 1; else return 0;
}

int GVAux(TreePtr t) {
/*'prec: t != NULL,
post: restituisce la differenza
tra il numero dei nodi gialli e quelli verdi*/
int diffG_V1,diffG_V2;
if (foglia(t)) if (t->elem ==1) return 1; else return -1;
if (t->left) {
    diffG_V1= GVAux(t ->left);
    if (diffG_V1 > 0) {
        if (t->right) {
            diffG_V2 = GVAux(t ->right);
            if (diffG_V2 > 0) diffG_V1 += diffG_V2;
```

```
            else return diffG_V2;
        }
    }
    else return diffG_V1;
else diffV_G1 = GVAux(t->right); // qui il figlio destro e' non vuoto
if (t->elem == 1) return diffG_V1+1;
else return diffG_V1-1;
}


//=======================================
//==2° ES.; seconda soluz (Del medico).===
//=======================================
int G_magg_V (treeptr t){
int g, v;
return G_meno_V (t, &g, &v);
}
int G_meno_V (treeptr t, int* g, int* v) {
//g e v conterranno il numero di nodi gialli e verdi che occorrono
//nel sottoalbero radicato in t; dà 0 o 1 a seconda che il sottoalbero
//radicato in t rispetti la proprietà di avere più gialli o no
int gl, vl, gr, vr;
if (!t) {
g = v = 0; /*un sottoalbero vuoto non ha nodi */
return 1;
}
if (G_meno_V(t->left, &gl, &vl) && G_meno_V(t->right, &gr, &vr)) {
/*entra solo se per entrambi i sottoalberi la proprietà è vera*/
      *g = *gl + *gr; //somma i gialli del sottoalbero sinistro e destro di t
      *v = *vl + *vr; //somma i verdi del sottoalbero sinistro e destro di t
      if (t->col == 'g') (*g)++; else (*v)++; //considera il nodo corrente
      if (*g > *v) return 1; else return 0;
}
return 0;
}


//=======================================
//=============3° ES.====================
//=======================================

//=============File Paese.h==============
typedef struct paese* paeseptr;
//costruttore
paeseptr costr (char* n);
//PREC.: n != NULL
//POSTC.: restituisce il puntatore ad una struttura di tipo paese
//con nome inizializzato a n e gli altri attributi a valori di default

//distruttore
void distr (paeseptr p);
//PREC.: p != NULL
//POSTC.: dealloca la memoria associata a p

//estrattori
char* p_nome (paeseptr p);
//PREC.: p != NULL
//POSTC.: restituisce il nome di p
float p_sup (paeseptr p);
//PREC.: p != NULL
//POSTC.: restituisce la superficie di p
long int p_ab_m (paeseptr p);
//PREC.: p != NULL
```

```
//POSTC.: restituisce il numero di abitanti maschi in p
long int p_ab_f (paeseptr p);
//PREC.: p != NULL
//POSTC.: restituisce il numero di abitanti femmine in p

//funzionalità
void set_sup (paeseptr p, float s);
//PREC.: p != NULL && s> 0
//POSTC.: s è la superficie di p
void set_ab_m (paeseptr p, long int k);
//PREC.: p != NULL && k>=0
//POSTC.: il paese p ha k abitanti maschi
void set_ab_f (paeseptr p, long int k);
//PREC.: p != NULL && k>=0
//POSTC.: il paese p ha k abitanti femmine
long int get_ab (paeseptr p);
//PREC.: p != NULL
//POSTC.: restituisce la popolazione totale di p
float get_dens (paeseptr p);
//PREC.: p != NULL
//POSTC.: restituisce la densità di popolazione in p

//=============File Paese.c===============
typedef struct paese {
char* nome;
float sup;
long int m;
long int f;
}

//========================================
//=============4° ES.=====================
//========================================
Dati di test a scatola nera per

int SeqUguali (int v[], int k, int n) {
//PREC.: 0 <= k <= n
//POSTC.: restituisce 1 se ci sono almeno k numeri uguali in sequenza in v,
vettore di n
//elementi, 0 altrimenti

Diamo per scontato che il valore di n sia la dimensione del vettore v.
Dalle post condizioni cerchiamo di suddividere il dominio dei dati di input in
classi di elementi equivalenti rispetto al test:
n = 0 e k > 0 => ris = 0 se n = 0 il risultato è 0 perchè non essendoci
   elementi, a maggior ragione non ce sono di uguali, per qualunque k > 0.
n = 0 e k = 0 => ris = 1 se n = 0 il risultato è 1 perchè essendoci 0 elementi,
   ci sono 0 elementi uguali.
Per n > 0 dovremmo prendere un vettore con n elementi che contiene una
   sequenza di k uguali e una che non la contiene, ma se n = 1 abbiamo solo due
   possibità: k=0 e k=1, e in entrambi i casi il risultato è 1, quindi trattiamo
   questo caso come a sè stante. Prendiamo per esempio
   Per n = 1, k=1 e v=[1] il risultato è 1
   Per n = 1, k=0 e v=[1] il risultato è 1
   Per n > 0 possiamo considerare tutti i valori di k da 1 fino a n.
   Per n = 2 però, escludendo il caso banale k=1 abbiamo solo un valore
       possible, trattiamo quindi il caso a parte. k = 2 possiamo prendere
       v=[1,1] e il risultato è 1 oppure v=[1,2] e il risultato è 0.
   Prendiamo n= 4 e consideriamo i casi k=2,k=3 e k=4 e per ogni caso un esempio
       di vettore per il quale la soluzione è 1 e uno per il quale la soluzione
       è 0: Per esempio n = 4 e k = 2,3 e 4 con
         v=[1,2,3,4] il risultato è 0 per tutti i k.
         v=[1,1,2,3] con risultato 1 per k=2 e 0 per k =3,4
```

v=[1,1,1,3] con risultato 1 per k=2,3 e 0 per k =4
v=[1,1,1,1] con risultato 1 in tutti i casi.
Potremmo anche fermarci qui, ma osserviamo che in tutti i casi la sequenza cercata è presente all'inizio. Potremmo considerare equivalenti tutti questi rispetto al test, separandoli dai casi in cui la sequenza è alla fine o centrale. Prendiamo per esempio n =4 e k=2,3
v=[1,2,3,3] con risultato 1 per k=2 e 0 per k =3
v=[1,2,2,2] con risultato 1 per k=2,3
Infine tutti quelli in cui la sequenza è "centrale". Prendiamo per esempio n =4 e k=2
v=[1,2,2,3] con risultato 1 per k=2.

Glass box

```
int SeqUguali (int v[], int k, int n) {
//PREC.: 0 <= k <= n
//POSTC.: restituisce 1 se ci sono almeno k numeri uguali in sequenza in v, 0
altrimenti
int cnt = 1, i = 0;
assert(k >= 0);
assert( k <= n);
if (k == 0 ) return 1;
if (n == 0) return 0;
while (cnt < k && i < n-1){
      if (v[i] == v[i+1]) cnt++;
      else cnt=1;
      i++;
}
if (cnt == k) return 1; return 0;
}
```

asso 1, visto che le precondizioni sono trattate con l'assert controlliamole chiamando la funzione su valori che le violano per essere sicuri di non aver dimenticato qualche precondizione. Prendiamo un valore di k negativo, p.e. k = -3, e un caso in cui n < k, p.e. n=3 e k=4, o anche n=-1 e k =1.
I casi n=0 e k= 0 sono già stati considerati.
Il caso k=1 è uno di quelli che provoca la non esecuzione del ciclo, lo testiamo per un n>1, p.e. n=4. Dovremmo considerare i casi che provocano un'unica esecuzione del ciclo, ma abbiamo già dei casi di questo tipo: Per esempio per k = 2 con
    n = 4 e v=[1,2,3,4] il risultato è 0, si esce eseguendo l'else.
    n = 2 e v=[1,1] con risultato 1 si esce eseguendo l'if
Ora i casi di almeno due esecuzioni del ciclo con le diverse uscite, ma anche qui notiamo che già I casi
    v=[1,2,3,4] il risultato è 0 per tutti i k.
    v=[1,1,2,3] con risultato 1 per k=2
    v=[1,1,1,3] con risultato 1 per k=2
    v=[1,1,1,1] con risultato 1 in tutti i casi.
    v=[1,2,3,3] con risultato 1 per k=2
    v=[1,2,2,2] con risultato 1 per k=2
Consentono di verificare le uscite dal ciclo, o la permanenza nel ciclo dopo due iterazioni, per non aggiungiamo altri dti di test.