Quadtree

Nome: MASSIMILIANO
Cognome: GIORDANO ORSINI
Matricola: 0124002214

Descrizione del problema:

Il quesito richiede la progettazione e l'implementazione di un quadtree per insiemi di punti (Point-Sets quadtree), in cui la capacità della regione associata univocamente ad un nodo sia di al più due punti. Tali punti sono circoscritti all'interno di un quadrato di dimensione 128x128.
In particolare, è richiesta l'implementazione di un menù, da cui sia possibile richiamare le seguenti operazioni: l'inserimento dei punti da file di testo e il metodo NEIGHBOR() che consente di ottenere, dato un nodo e una direzione, il nodo adiacente nella direzione richiesta, se esiste.

Descrizione della struttura dati:

Un quadtree è un albero radicato in cui ogni nodo interno ha quattro figli. Ogni nodo corrisponde ad uno spazio quadrato e dispone di un contenitore di uno o più punti. Se un nodo ν ha figli, i loro quadrati corrispondenti sono i quattro sotto-quadranti del quadrato di ν, da qui il nome di quadtree. Questo implica che i quadrati delle foglie formano insieme una suddivisione del quadrato della radice.

I figli di un nodo sono etichettati con NE, NW, SW, SE ad indicare a quale quadrante corrispondono; NE sta per il quadrante a Nord-Est, NW per il quadrante a Nord-Ovest, e così via. Le regioni suddivise in un quadtree possono essere quadrate o rettangolari, oppure possono avere forme arbitrarie.

I quattro vertici agli angoli del quadrato sono chiamati "corner vertices", in italiano spigoli. I segmenti che connettono due spigoli consecutivi vengono detti "sides of the square", in italiano lati. I bordi dei quadrati ottenuti dopo la suddivisione, che sono contenuti nei confini di un quadrato sono chiamati "edges of the square", in italiano bordi. Quindi, un lato di un sotto-quadrato contiene almeno un bordo ma è possibile che ne contenga più di uno. Due quadrati sono detti "neighbors", in italiano vicini, se condividono un bordo.

I quadtree possono essere utilizzati per memorizzare diversi tipi di dati. Nello specifico caso, si descrive la variante che memorizza un insieme di punti nel piano, la suddivisione ricorsiva dei quadrati continua fino a quando ci sono più di due punti in un quadrato associato ad una foglia. Infatti, tali punti devono trovarsi solo nelle foglie e non nei nodi interni.

$$\sigma := [x_\sigma : x'_\sigma] \times [y_\sigma : y'_\sigma].$$

- If $\text{card}(P) \leqslant 2$ then the quadtree consists of a single leaf where the set P and the square σ are stored.
- Otherwise, let σ_{NE}, σ_{NW}, σ_{SW}, and σ_{SE} denote the four quadrants of σ. Let $x_{\text{mid}} := (x_\sigma + x'_\sigma)/2$ and $y_{\text{mid}} := (y_\sigma + y'_\sigma)/2$, and define

$$\begin{aligned}
P_{\text{NE}} &:= \{p \in P : p_x > x_{\text{mid}} \text{ and } p_y > y_{\text{mid}}\},\\
P_{\text{NW}} &:= \{p \in P : p_x \leqslant x_{\text{mid}} \text{ and } p_y > y_{\text{mid}}\},\\
P_{\text{SW}} &:= \{p \in P : p_x \leqslant x_{\text{mid}} \text{ and } p_y \leqslant y_{\text{mid}}\},\\
P_{\text{SE}} &:= \{p \in P : p_x > x_{\text{mid}} \text{ and } p_y \leqslant y_{\text{mid}}\}.
\end{aligned}$$

Sia v la radice dell'albero, se v ha quattro figli:

1. *Il figlio NE è la radice del quadtree per il set P-NE all'interno del quadrato σ-NE,*
2. *Il figlio NW è la radice del quadtree per il set P-NW all'interno del quadrato σ-NW,*
3. *Il figlio SW è la radice del quadtree per il set P-SW all'interno del quadrato σ-SW,*
4. *Il figlio SE è la radice del quadtree per il set P-SE all'interno del quadrato σ-SE.*

La profondità d di un quadtree per un insieme di punti P nel piano è al più

$$log\left(\frac{s}{c}\right) + \frac{1}{2}$$

dove c è la più piccola distanza tra due punti in P ed s è la lunghezza del lato del quadrato iniziale che contiene P.
Ciò è dato dal fatto che quando si scende da un nodo ad uno dei suoi figli, la dimensione del quadrato corrispondente si dimezza. Quindi, la lunghezza del lato del quadrato di un nodo alla profondità i è $\frac{s}{2^i}$. La massima distanza tra due punti all'interno di un quadrato è data dalla lunghezza della sua diagonale, che è pari a $\frac{s\sqrt{2}}{2^i}$ per il quadrato di un nodo alla profondità i. Considerando quindi un nodo interno ha almeno due punti nel suo quadrato associato, e la minima distanza tra due punti è c, la profondità i di un nodo interno deve soddisfare

$$\frac{s\sqrt{2}}{2^i} \geq c$$

che implica

$$i \leq \log\frac{s\sqrt{2}}{c} = log\left(\frac{s}{c}\right) + \frac{1}{2}$$

Poiché la profondità di un quadtree è esattamente uno più la massima profondità di un qualsiasi nodo interno, questa risulta essere

$$\log\left(\frac{s}{c}\right) + \frac{3}{2}$$

In alternativa la notazione utilizzata è $d + 1$.

Formato dei dati input/output:

All'avvio del file eseguibile, viene stampato a video il menù con cui l'utente può interagire, da tastiera, digitando un numero intero positivo, corrispondente all'operazione che si vuole eseguire. Nel caso in cui l'utente dovesse inserire una cifra non valida, il programma fa ripetere l'inserimento della cifra per la scelta tra le opzioni disponibili fino a quando questa non risulta essere corretta.

Il menù numerato mostra tre opzioni: l'inserimento da file di testo, in cui l'utente è chiamato ad inserire la stringa corrispondente al nome del file di testo in cui sono memorizzati i punti, il metodo NEIGHBOR(), per il quale l'utente deve fornire un numero intero positivo corrispondente all'identificatore di un nodo e la direzione, indicata con un numero intero positivo da selezionare e digitare tra quelli proposti, e l'uscita, che avviene digitando la cifra corrispondente.

Il file di testo, presente all'interno della directory del codice sorgente, contiene un insieme di punti, uno per ogni riga; le coordinate x e y di un punto sono separate da una virgola. Ogni coordinata è rappresentata da un numero intero o decimale, positivo, compreso tra 1 e 128, come definito nelle assunzioni della traccia.

L'output quindi è rappresentato dalla stampa a video della riuscita o meno dell'inserimento dei punti all'interno della struttura dati, e dell'identificatore del nodo adiacente, restituito dal metodo NEIGHBOR() per il nodo e la direzione immessi, qualora questo esista o meno.

Descrizione dell'algoritmo:

SUBDIVIDE():

La procedura subdivide() si occupa della suddivisione del quadrante in sotto-quadranti.

Partendo dagli attributi forniti dal nodo, rispettivamente le coordinate del centro, la larghezza e l'altezza del quadrato partendo dal centro, crea i nuovi sotto-quadranti, che risulteranno essere i figli del nodo di input.

```
procedura subdivide()
      crea il sotto-quadrante NE
      crea un nuovo nodo a cui viene assegnato il sotto-quadrante NE
      assegna il nodo appena creato come figlio NE

      crea il sotto-quadrante NW
      crea un nuovo nodo a cui viene assegnato il sotto-quadrante NW
      assegna il nodo appena creato come figlio NW

      crea il sotto-quadrante SE
      crea un nuovo nodo a cui viene assegnato il sotto-quadrante SE
      assegna il nodo appena creato come figlio SE

      crea il sotto-quadrante SW
      crea un nuovo nodo a cui viene assegnato il sotto-quadrante SW
      assegna il nodo appena creato come figlio SW
```

INSERT():

In primo luogo, si effettua un controllo sull'appartenenza del punto al quadrante associato al nodo, che alla prima iterazione corrisponde alla radice del quadtree. In caso di esito negativo, il punto non può essere inserito nella lista di quel nodo.

Successivamente si verifica la possibilità dell'inserimento del punto nella lista dei punti del nodo; ciò è possibile se la capacità dei punti per quel nodo non è stata superata e se il nodo risulta ancora essere una foglia poiché non suddiviso. In caso di esito affermativo, il punto viene inserito correttamente all'interno del quadtree. Nel caso in cui la capacità massima di punti del nodo venga raggiunta, il punto non può essere inserito nella lista corrispondente ma è necessario suddividere il quadrante associato in sotto-quadranti attraverso l'apposita funzione subdivide(). Qualora invece il nodo sia suddiviso significa che è ancora possibile scendere in profondità, fino alle foglie, ed inserire il punto in una regione più ristretta.

Infine la funzione insert() viene richiamata ricorsivamente per scendere in profondità nei sotto-quadranti già esistenti oppure appena creati.

Una volta inserito il punto, si verifica se l'ultimo inserimento ha prodotto una suddivisione di un nodo in sotto-quadranti. In caso di esito affermativo, è necessario "spostare" i punti appartenenti al nodo appena suddiviso, nelle foglie del quadtree, in quanto solo le foglie possono contenere i punti; pertanto viene richiamata una procedura di "fix-up" su tali punti, ossia questi vengono estratti e re-inseriti mediante la funzione insert().

```
procedura insert(p)
      splittedNode ← NIL
      if (Wrapper_insert(p, root))
            print "Insert for point XY: success!"
      else
            print "Insert for point XY: failed!"

      if (splittedNode ≠ NIL)
            while (splittedNode.points ≠ ∅)
                  point ← splittedNode.points.pop()
```

```
            if (Wrapper_insert(p, root) == FALSE)
                  return
```

```
function Wrapper_insert(p, node, splittedNode)
      if (node.boundary.containsPoint(p) == FALSE)
            return FALSE
      if (node.points.size < QT_NODE_CAPACITY && node.divided == FALSE)
            node.points.append(p)
            return TRUE
      if (node.divided == FALSE)
            splittedNode ← node
            subdivide(node)
      if (Wrapper_insert(p, node.northWest, splittedNode)) return TRUE
      if (Wrapper_insert(p, node.northEast, splittedNode)) return TRUE
      if (Wrapper_insert(p, node.southWest, splittedNode)) return TRUE
      if (Wrapper_insert(p, node.southEast, splittedNode)) return TRUE

      return FALSE

function FixUp_insert(p, node)
      if (node.boundary.containsPoint(p) == FALSE)
            return FALSE
      if (node.points.size < QT_NODE_CAPACITY && node.divided == FALSE)
            node.points.append(p)
            return TRUE
      if (node.divided == FALSE)
            subdivide(node)
      if (FixUp_insert(p, node.northWest)) return TRUE
      if (FixUp_insert(p, node.northEast)) return TRUE
      if (FixUp_insert(p, node.southWest)) return TRUE
      if (FixUp_insert(p, node.southEast)) return TRUE

      return FALSE
```

SEARCH():

La funzione search() è una funzione booleana, ricorsiva, che permette la ricerca di un nodo all'interno del quadtree tramite le coordinate di un punto; è necessaria per la chiamata al metodo NEIGHBOR().

Sia p un punto qualsiasi, il metodo prevede due casi base:

- se p non è contenuto all'interno del quadrato associato al nodo, restituisce FALSE.
- se p è contenuto in un quadrato che risulta essere una foglia, il nodo associato al quadrato viene memorizzato e restituisce TRUE.

La ricerca continua ricorsivamente nei sotto-alberi NE, NW, SE, SW fino a quando il punto è contenuto in un quadrato di un nodo-foglia. In caso di esito negativo alla prima iterazione, significa che le coordinate del punto inserito non risultano essere valide in quanto questo non è contenuto all'interno del quadrato iniziale, pertanto viene restituito FALSE.

```
function search(p, node, result)
      if (node.boundary.containsPoint(p) == FALSE)
            return FALSE
      if (node.divided == FALSE)
            result ← node
            return TRUE
```

```
    if (search(p, node.northWest, result)) return TRUE
    if (search(p, node.northEast, result)) return TRUE
    if (search(p, node.southWest, result)) return TRUE
    if (search(p, node.southEast, result)) return TRUE

    return FALSE
```

NEIGHBOR():

Il metodo NEIGHBOR() prende come parametri di input un punto, che corrisponderà ad un nodo dell'albero tramite la funzione search(), e una direzione; restituisce il nodo adiacente nella direzione richiesta.

Quindi, avvia la ricerca del nodo all'interno del quadtree. In caso di esito negativo, viene restituito NIL.

In base alla direzione immessa, chiama il metodo corrispondente per la ricerca del "neighbor", che si occuperà di restituire il nodo adiacente nella direzione richiesta, se questo esiste.

Sia ***v*** un nodo esistente all'interno del quadtree, supponiamo che l'utente voglia conoscere il nodo adiacente a ***v*** nella direzione nord. Il metodo prevede tre casi base:

- se ***v*** è la radice, restituisce NIL poiché questa non ha nodi adiacenti.
- se ***v*** è figlio SW rispetto al padre di ***v***, allora il nodo adiacente è il figlio NW rispetto al padre di ***v***.
- se ***v*** è figlio SE rispetto al padre di ***v***, allora il nodo adiacente è il figlio NE rispetto al padre di ***v***.

Se il nodo ***v*** non ricade in questi tre casi, a partire dal padre di ***v***, si risale l'albero ricorsivamente cercando di ricondursi a questi. Sia ***u*** il prodotto delle chiamate ricorsive, il metodo prevede tre casi:

- se ***u*** risulta essere NIL oppure ***u*** è una foglia, viene restituito ***u***. Nel primo caso, significherebbe che ***v*** non ha nodi adiacenti a nord, nel secondo che il nodo ***u*** sia effettivamente il nodo adiacente a nord più in profondità nell'albero.
- se ***v*** risulta essere il figlio NW rispetto al padre di ***v***, viene restituito il figlio SW di ***u***, in quanto questo risulta essere comunque un nodo adiacente nella direzione nord che abbia al più la stessa profondità di ***v***.
- se ***v*** risulta essere il figlio NE rispetto al padre di ***v***, viene restituito il figlio SE di ***u***, in quanto questo risulta essere comunque un nodo adiacente nella direzione nord che abbia al più la stessa profondità di ***v***.

L'algoritmo viene applicato per le altre direzioni, con le opportune modifiche.

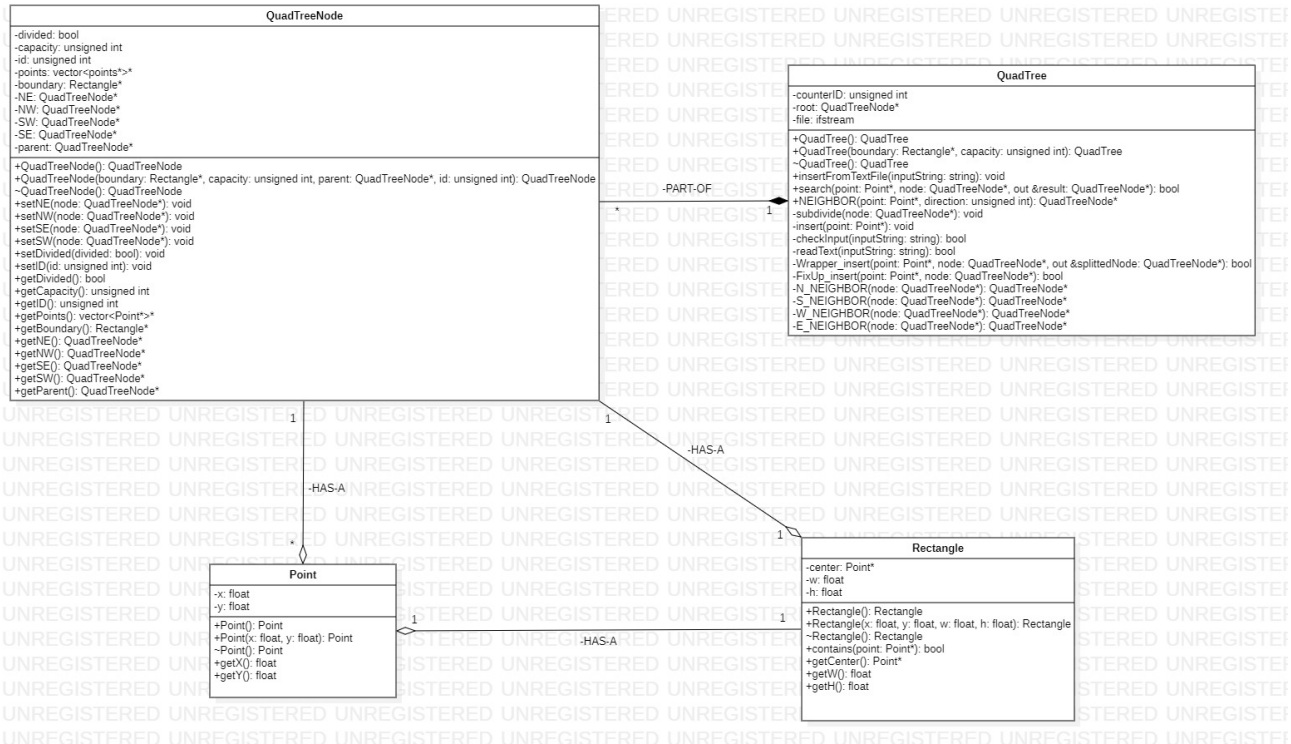
```
function NEIGHBOR(node, direction)
    result ← NIL
    if ((search(node, root, result) == FALSE)
        print "error: node not found!"
        return result

    print "node found!!
    switch (direction)
        case 1:    //se la direzione corrisponde a nord
            return N_NEIGHBOR(result)
            break
        case 2:    //se la direzione corrisponde a sud
            return S_NEIGHBOR(result)
            break
        case 3:    //se la direzione corrisponde a ovest
            return W_NEIGHBOR(result)
            break
```

```
case 4:    //se la direzione corrisponde a est
      return E_NEIGHBOR(result)
      break
default:
      print "invalid input"
      return NIL
      break
```

Algorithm NORTHNEIGHBOR($v, \mathcal{T}$)
Input. A node v in a quadtree $\mathcal{T}$.
Output. The deepest node v' whose depth is at most the depth of v such that $\sigma(v')$ is a north-neighbor of $\sigma(v)$, and **nil** if there is no such node.

1. **if** $v = root(\mathcal{T})$ **then return nil**
2. **if** $v =$ SW-child of $parent(v)$ **then return** NW-child of $parent(v)$
3. **if** $v =$ SE-child of $parent(v)$ **then return** NE-child of $parent(v)$
4. $\mu \leftarrow$ NORTHNEIGHBOR($parent(v), \mathcal{T}$)
5. **if** $\mu =$ **nil or** μ is a leaf
6. **then return** μ
7. **else if** $v =$ NW-child of $parent(v)$
8. **then return** SW-child of μ
9. **else return** SE-child of μ

Class Diagram:



La classe QuadTreeNode ha una relazione di composizione di tipo "part-of" con la classe QuadTree in quanto i nodi compongono l'albero, da qui motivata anche la scelta della molteplicità *…1, poiché più nodi compongono un albero e un albero è composto da più nodi.

La classe QuadTreeNode, inoltre, ha una relazione di aggregazione di tipo "has-a" con la classe Rectangle poichè utilizza un'istanza della classe Rectangle, a differenza di una relazione "is-a" che viene di solito

utilizzata in caso di ereditarietà delle classi, tra classe derivata e superclasse. Ad un nodo corrisponde un quadrato.

La relazione di aggrezione di tipo “has-a” tra la classe QuadTreeNode e la classe Point è motivata dal fatto che un nodo ha un contenitore di punti poichè questo può contenere fino a 2 punti.

La classe Rectangle ha una relazione di aggregazione di tipo “has-a” con la classe Point poiché, nel caso specifico, un oggetto Rectangle ha uno ed unico centro e quindi un punto.

Studio della complessità:

Un quadtree di profondità d che memorizza un insieme di n punti ha al più $(d+1)n$ nodi e può essere costruito in tempo $O\big((d+1)n\big)$.

Ogni nodo interno in un quadtree ha quattro figli, quindi il numero totale di foglie è uno più tre volte il numero di nodi interni. Ciò rappresenta il limite superiore del numero di nodi interni.

I quadrati dei nodi alla stessa profondità in un quadtree sono disgiunti e coprono esattamente il quadrato iniziale. Questo significa che il numero totale di nodi interni ad una data profondità qualsiasi è al più n. Da qui deriva il limite superiore.

L’operazione che impiega la maggior parte del tempo nell’algoritmo di costruzione ricorsiva è la distribuzione dei punti sui quadranti del quadrato corrente. Tuttavia, l’ammontare del tempo speso in un nodo interno è lineare nel numero di punti che non appartengono più al quadrante associato. Ciò rispetta comunque il limite superiore proposto in precedenza.

SUBDIVIDE():

La procedura subdivide() crea i quattro sotto-quadranti associati ai nuovi figli del nodo appena suddiviso. La creazione di un quadrato prevede l’assegnazione delle coordinate del centro del quadrante, la larghezza e l’altezza agli attributi del quadrato appena creato, quindi possiamo ritenere che questa sia svolta in un tempo costante c_1. Allo stesso modo, la creazione di un nodo prevede l’inizializzazione dei suoi attributi, attraverso l’assegnazione del quadrante, del padre e del suo identificatore, realizzata quindi in un tempo costante c_2.

Possiamo concludere che la procedura subdivide() ha complessità $\Theta(c_1 + c_2)$, quindi $\Theta(k)$ dove k è una costante.

INSERT():

Nel caso migliore, ossia all’inserimento del primo punto, il costo dell’operazione è pari ad un tempo costante $O(1)$, mentre, nel caso peggiore, cioè in seguito all’inserimento di punti che si trovano in un intorno relativamente piccolo, che quindi tendono a sbilanciare l’albero e a mostrarlo come una sorta di lista, il costo dell’operazione per l’inserimento di un punto, appartenente ancora a quell’intorno ristretto, è asintoticamente pari ad $O(n)$. Nel caso medio, ossia quando l’albero risulta essere sommariamente bilanciato, l’operazione di inserimento viene eseguita in tempo $O(d+1)$ dove d è la profondità del cammino più lungo. La complessità della chiamata alla procedura subdivide() all’interno della funzione insert() risulta essere asintoticamente trascurabile poiché costante.

SEARCH():

La funzione search() lavora tramite le coordinate di un punto che identificano un certo quadrato all’interno del quadtree.

Nel caso migliore, ossia quando il quadtree contiene al più due punti, la ricerca avviene in tempo costante $O(1)$, in quanto l'albero è formato dal quadrato iniziale che corrisponde alla radice, mentre, nel caso peggiore, cioè in seguito all'inserimento di punti che si trovano in un intorno relativamente piccolo, che quindi tendono a sbilanciare l'albero e a mostrarlo come una sorta di lista, il costo dell'operazione per la ricerca di un nodo identificato dalle coordinate di un punto, appartenente ancora a quell'intorno ristretto, è asintoticamente pari ad $O(n)$.

Nel caso medio, ossia quando l'albero risulta essere sommariamente bilanciato, l'operazione di ricerca viene eseguita in tempo $O(d+1)$ dove d è la profondità del cammino più lungo. Questa complessità risulta essere ottimale per la struttura del quadtree implementata, a differenza di una ricerca tramite identificatore del nodo, riconducibile alla complessità di una visita anticipata dell'albero.

NEIGHBOR():

Il metodo NEIGHBOR() sviluppato si divide in due passi: la ricerca del nodo all'interno dell'albero e la chiamata al metodo corrispondente alla direzione richiesta.

Il metodo richiamato in base alla direzione immessa non necessariamente restituisce un nodo foglia. L'algoritmo impiega tempo costante ad ogni chiamata ricorsiva. Ad ogni chiamata la profondità del nodo passato come argomento decrementa di uno. Quindi, la complessità di tempo è lineare rispetto alla profondità del quadtree. Pertanto il "neighbor" di un certo nodo v in una certa direzione può essere trovato in tempo $O(d+1)$ dove d è la profondità dell'albero.

Nel caso migliore, ossia quello in cui l'albero sia formato solamente dalla radice, il tempo impiegato dalla funzione search() è costante così come quello impiegato dal metodo richiamato per la direzione richiesta, per cui il metodo NEIGHBOR() ha complessità pari a $O(1)$.

Nel caso peggiore, ossia in cui il quadtree sia sbilanciato completamente verso una direzione, la ricerca eseguita dalla funzione search() avviene in tempo $O(n)$. Nella direzione più svantaggiosa, il metodo richiamato per restituire il nodo adiacente, a partire dal nodo, risalirebbe l'albero percorrendo quindi tutti i nodi, impiegando quindi un tempo asintoticamente pari ad $O(n)$, che non è altro che la complessità del metodo NEIGHBOR().

Nel caso in cui l'albero sia sommariamente bilanciato, la funzione search() impiega un tempo $O(d+1)$ così come quella del metodo richiamato per la direzione richiesta; possiamo concludere quindi che il metodo NEIGHBOR() ha complessità pari a $O(d+1)$.

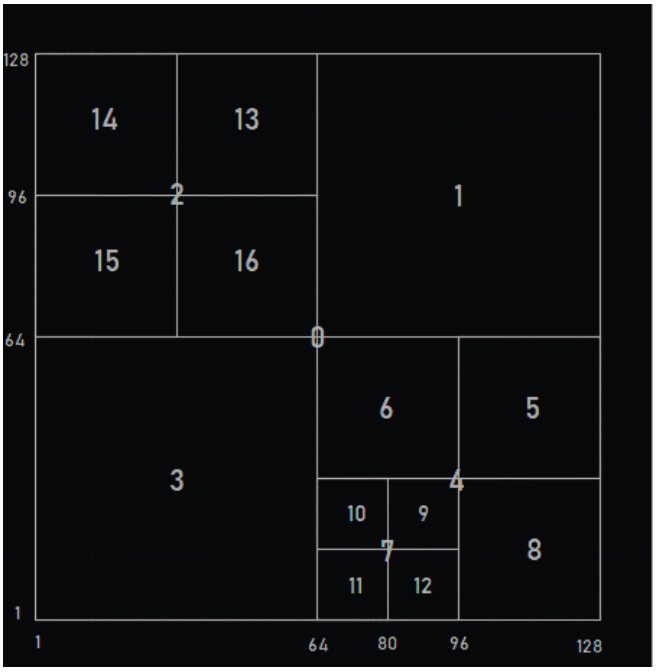
Test e risultati:

Test #1:

```
----------------QUADTREE----------------
1. Inserisci i punti dal file di testo
2. NEIGHBOR()
3. Esci
1
Inserisci il nome del file di testo:
input.txt

Inserimento dei punti nel file di testo

FILE: input.txt
Status: Successfully opened!

Insert for (112,121): success!
Insert for (24,97): success!
Insert for (26,35): success!
Insert for (66,23): success!
Insert for (67,25): success!
Insert for (68,24): success!
Insert for (25,98): success!
Insert for (42,99): success!

Insert from TextFile: success!
1. Inserisci i punti dal file di testo
2. NEIGHBOR()
3. Esci
2
Inserisci le coordinate del punto (una dopo l'altra) che identificano il nodo di cui vuoi conoscere il NEIGHBOR():
120
120

Seleziona una direzione:
1. N
2. S
3. W
4. E
3
ID_NODO: 1      ID_NEIGHBOR: 2
```

L'obiettivo di questo test è inserire i punti all'interno del quadtree e chiamare il metodo NEIGHBOR(). All'avvio del file eseguibile, l'utente ha davanti a sé il menù, ossia l'interfaccia che gli permette di interagire tramite tastiera.

L'utente quindi digita l'opzione 1 e consente l'inserimento dei punti dal file di testo all'interno del quadtree, pertanto immette la stringa corrispondente al nome del file.



Viene stampato a video la corretta apertura del file o meno e successivamente l'esito dell'inserimento per i punti presenti all'interno del file.

Successivamente l'utente digita l'opzione 2 per chiamare il metodo NEIGHBOR(), deve quindi immettere le coordinate di un punto.

Di fianco è riportata una rappresentazione grafica del quadtree generato dai punti inseriti tramite file.

L'utente quindi sceglie un punto a sua discrezione che identificherà il quadrante, e quindi il nodo, e in seguito la direzione. Vengono stampati a video l'ID del nodo, se il punto è valido, e l'ID del "neighbor", se questo esiste.

Nel caso specifico l'utente inserisce il punto (120,120), identificato dall'ID_NODO pari ad 1. Il "neighbor" corrispondente per la direzione selezionata (ovest) ha ID_NEIGHBOR pari a 2.

L'output risulta essere corretto sia per la ricerca del nodo, come mostra la foto, sia per il nodo restituito dal metodo NEIGHBOR(), in quanto, come detto in precedenza, viene restituito quello che risulta avere al più la stessa profondità del nodo passato come parametro, ovviamente nella direzione richiesta.

Test #2:

Questo test ha l'obiettivo di lavorare con punti diversi rispetto al precedente e quindi un quadtree diverso, e mostrare il caso in cui il punto inserito per identificare un nodo non risulta essere valido e in seguito il caso in cui il "neighbor" non esista.

```
----------------QUADTREE----------------
1. Inserisci i punti dal file di testo
2. NEIGHBOR()
3. Esci
1
Inserisci il nome del file di testo:
input.txt

Inserimento dei punti nel file di testo

FILE: input.txt
Status: Successfully opened!

Insert for (127,89): success!
Insert for (36,78): success!
Insert for (78,99): success!
Insert for (13,22): success!
Insert for (101,98): success!

Insert from TextFile: success!
1. Inserisci i punti dal file di testo
2. NEIGHBOR()
3. Esci
2
Inserisci le coordinate del punto (una dopo l'altra) che identificano il nodo di cui vuoi conoscere il NEIGHBOR():
157
44

Seleziona una direzione:
1. N
2. S
3. W
4. E
1
ID non trovato
        NEIGHBOR NON TROVATO
1. Inserisci i punti dal file di testo
2. NEIGHBOR()
3. Esci
2
Inserisci le coordinate del punto (una dopo l'altra) che identificano il nodo di cui vuoi conoscere il NEIGHBOR():
111
99

Seleziona una direzione:
1. N
2. S
3. W
4. E
1
ID_NODO: 5      NEIGHBOR NON TROVATO
```

L'utente permette l'inserimento dei punti correttamente e in seguito immette le coordinate di un punto non presente all'interno del quadrato iniziale, di conseguenza il "neighbor" non può essere trovato. Viene quindi stampato a video "ID non trovato" "NEIGHBOR NON TROVATO".

L’utente immette nuovamente un punto, questa volta valido, e richiede il nodo adiacente nella direzione nord. Il punto si trova nel quadrante associato al nodo che ha ID pari a 5.

Come risulta dalla rappresentazione grafica, non esiste nodo adiacente in quella direzione, pertanto viene stampato a video “NEIGHBOR NON TROVATO” correttamente.

Poiché non è possibile mostrare a schermo la rappresentazione grafica dell’albero, questa dev’essere sviluppata autonomamente, tenendo conto dell’ordine delle suddivisioni, determinante nell’assegnazione dell’identificatore del nodo.