CFG e Test Suite

Software Engineering for Embedded Systems

Agata Parietti

14/02/2023

Scopo del progetto

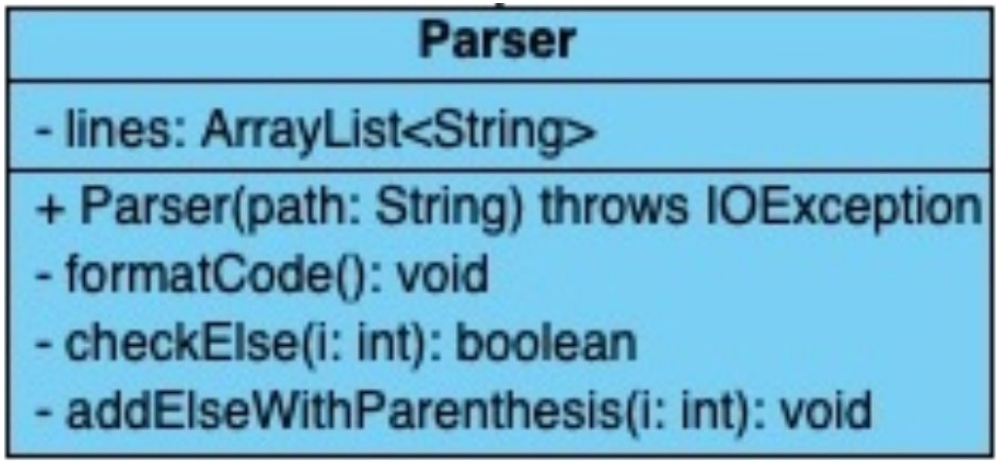
A partire da del codice C:

1. Generare il Control Flow Graph
2. Creare una Test Suite che contenga tutti i possibili test del codice secondo il metodo All Paths
3. Testare il programma su degli algoritmi di ordinamento

Implementazione

Il programma è stato implementato in Java ed è composto dalle seguenti classi:

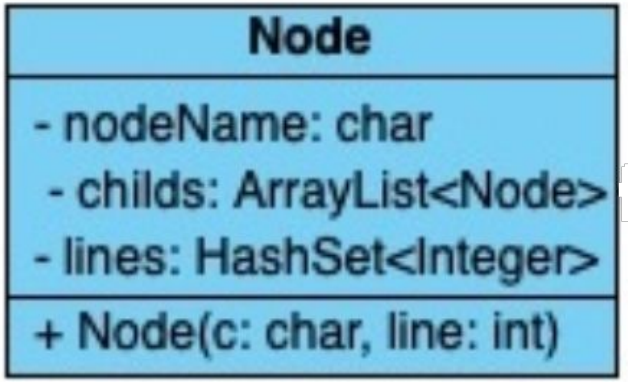
- Parser: interpreta il codice sorgente e applica regole di formattazione per trasformarlo in una struttura dati comprensibile dal programma.



Implementazione

Il programma è stato implementato in Java ed è composto dalle seguenti classi:

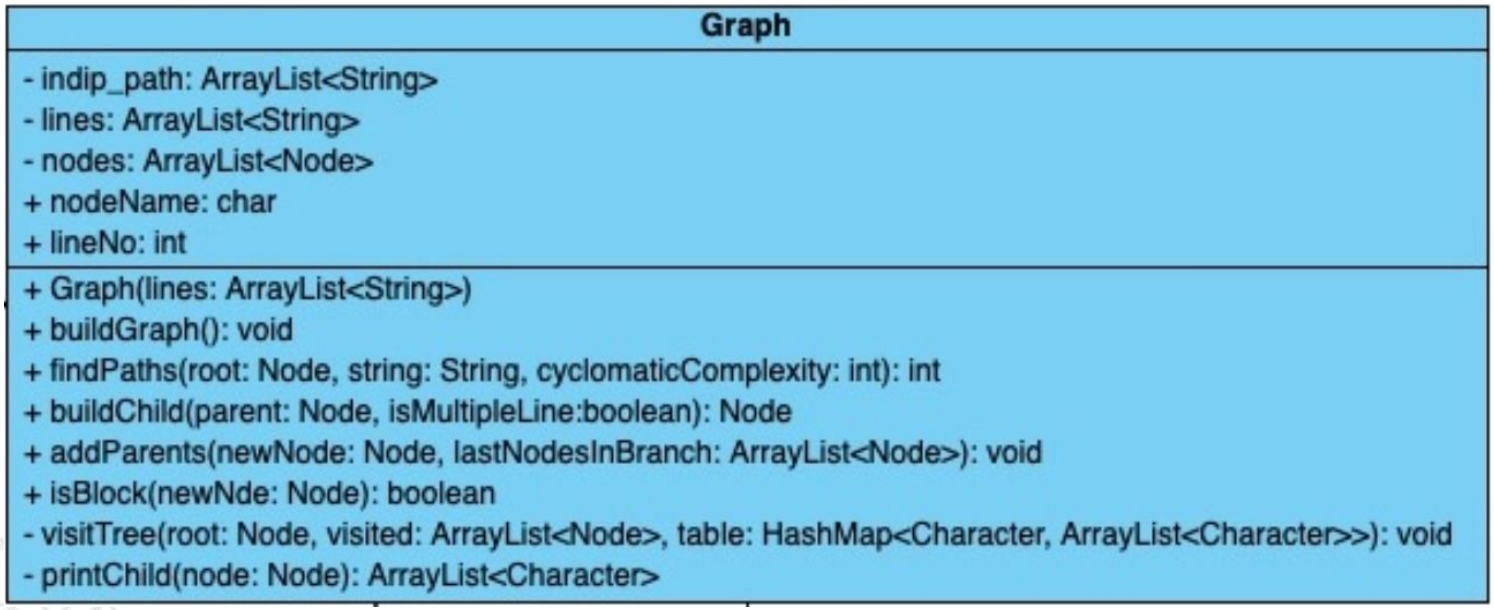
- Node: rappresenta un nodo del CFG



Implementazione

Il programma è stato implementato in Java ed è composto dalle seguenti classi:

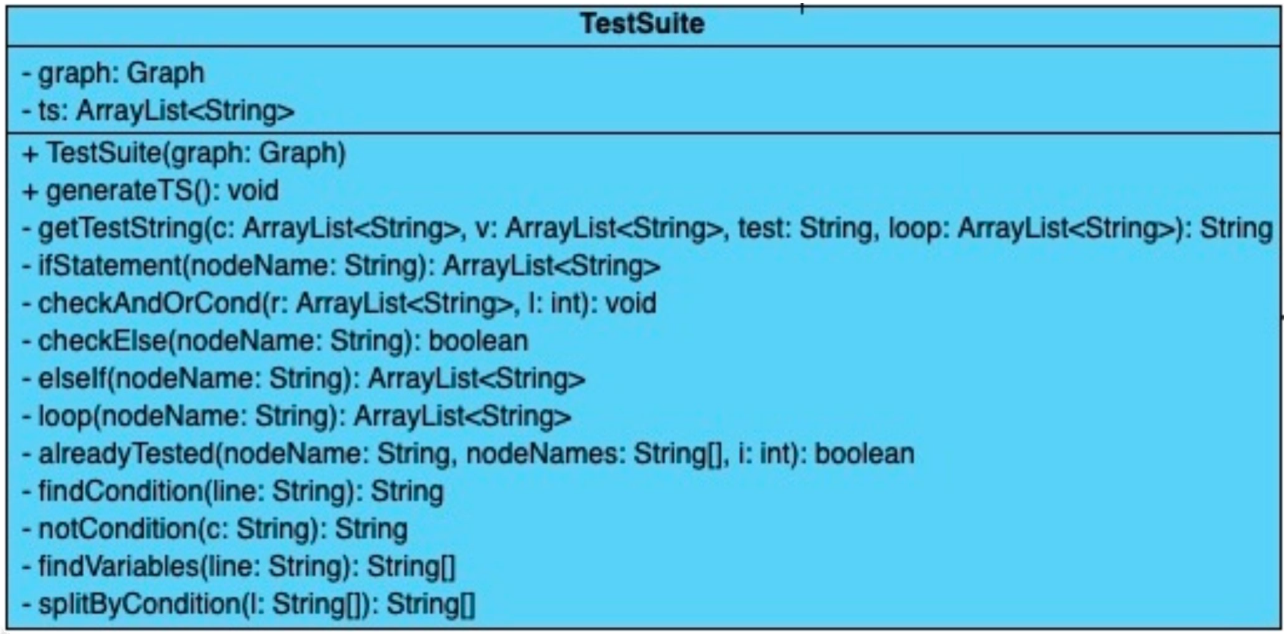
- Graph: responsabile della costruzione e della gestione del CFG. Utilizzando oggetti Node, il Grafo viene creato in base alla struttura del codice sorge



Implementazione

Il programma è stato implementato in Java ed è composto dalle seguenti classi:

- TestSuite: è responsabile della generazione dei test a partire dai percorsi indipendenti del CFG.



Implementazione

Il programma è stato implementato in Java ed è composto dalle seguenti classi:

- Main

```
public class Main {
    no usages
    public static void main(String[] args) throws IOException {
        String path = "quickSort.txt";
        Parser p = new Parser(path);
        Graph graph = new Graph(p.lines);
        System.out.println();
        System.out.println("The graph:");
        graph.buildGraph();
        System.out.println();
        System.out.println("The Test Suite:");
        TestSuite ts = new TestSuite(graph);
        ts.generateTS();
    }
}
```

Implementazione

Inoltre è stato creato uno script Python per la raffigurazione grafica dei CFG e dei vari percorsi indipendenti trovati.

Librerie Utilizzate:

- *NetworkX*: Per la creazione e la manipolazione del grafo.
- *Matplotlib*: Per il disegno del grafo.

Esempio

Facciamo un esempio per capire meglio il funzionamento del programma

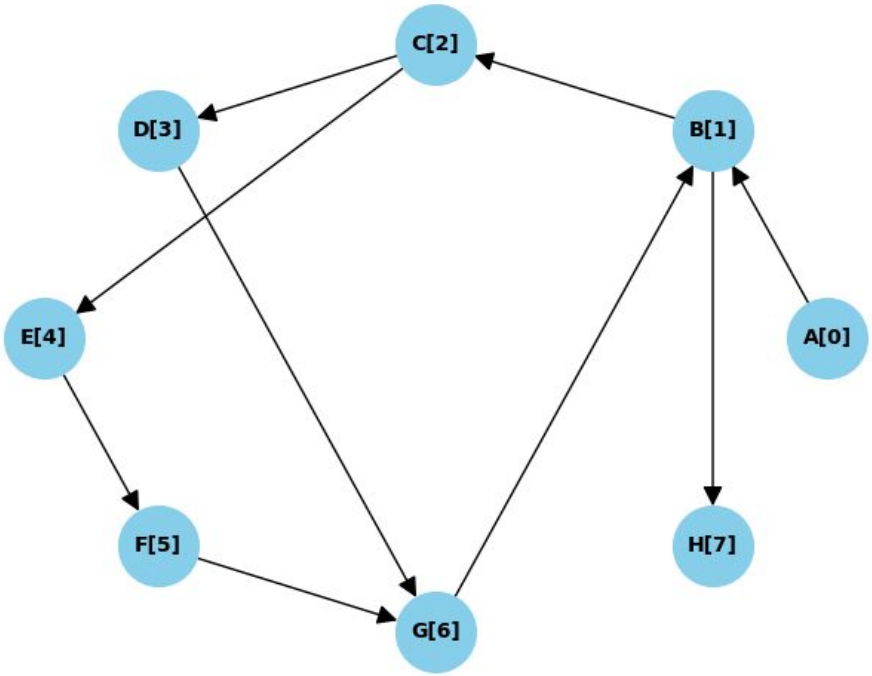
```
1  for (int i = 0; i < 10; i++) {
2       if (i % 2 == 0)
3            printf("%d is even", i);
4       else
5            printf("%d is odd", i);
6  }
```

Come vediamo ci sono 3 percorsi:

1. Attraversa il ciclo "for" e soddisfa la condizione dell'istruzione "if".
2. Segue lo stesso ciclo "for", ma non soddisfa la condizione dell'istruzione "if".
3. Non coinvolge il ciclo "for".

Esempio

Vediamo il CFG generato dal programma




```
Indepent paths:
A-->B-->C-->D-->G-->B-->H
A-->B-->C-->E-->F-->G-->B-->H
A-->B-->H
```

```
The graph:
A-->(B)
A[0]
B-->(C,H)
B[1]
C-->(D,E)
C[2]
D-->(G)
D[3]
G-->(B)
G[6]
E-->(F)
E[4]
F-->(G)
F[5]
H-->()
H[7]
```

Esempio

La Test Suite generata segue questi 3 percorsi.

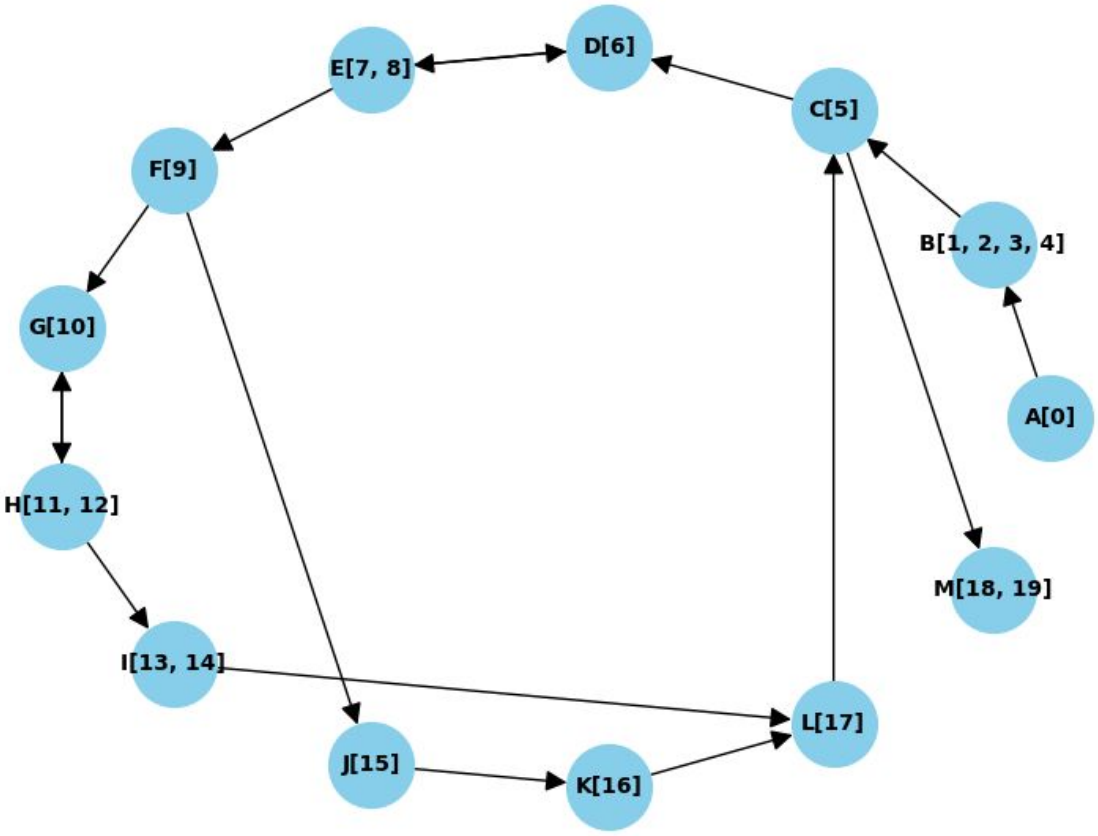
```
1  for (int i = 0; i < 10; i++) {
2      if (i % 2 == 0)
3          printf("%d is even", i);
4      else
5          printf("%d is odd", i);
6  }
```

```
Indepent paths:
A-->B-->C-->D-->G-->B-->H
A-->B-->C-->E-->F-->G-->B-->H
A-->B-->H
```

```
The Test Suite:
0.  i  <  10; i % 2  ==  0;
1.  i  <  10; i % 2  !=  0;
2.  i  >=  10;
```

Risultati - Quick Sort

Vediamo il CFG dell'algoritmo di Quick Sort



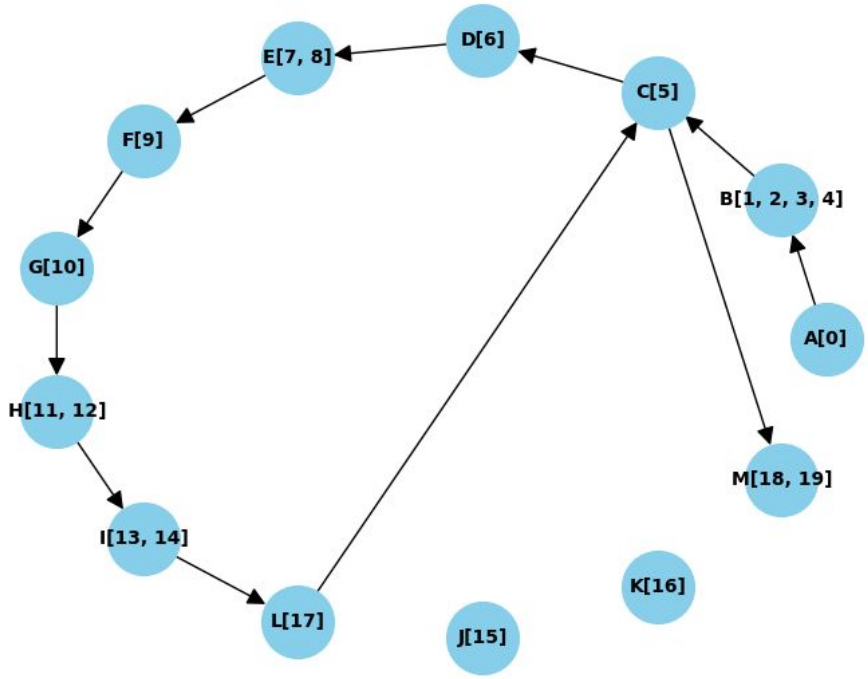

```
The graph:
A-->(B)
A[0]
B-->(C)
B[1, 2, 3, 4]
C-->(D,M)
C[5]
D-->(E)
D[6]
E-->(D,F)
E[7, 8]
F-->(G,J)
F[9]
G-->(H)
G[10]
H-->(G,I)
H[11, 12]
I-->(L)
I[13, 14]
L-->(C)
L[17]
J-->(K)
J[15]
K-->(L)
K[16]
M-->()
M[18, 19]
```

Risultati - Quick Sort

Analizziamo i singoli percorsi trovati

1. A-->B-->C-->D-->E-->F-->G-->H-->I-->L-->C-->M

```
1   int l, r, pivot;
2   pivot = V[0];
3   l = 0;
4   r = N;
5   while ( l < r ) {
6         do {
7             r--;
8         }while ( V[r] > pivot && r > l );
9         if ( r != l ) {
10              do {
11                  l++;
12              } while ( V[l] <= pivot && l < r );
13              swap(V, l, r);
14        }
15  }
16  swap(V, l, 0);
```

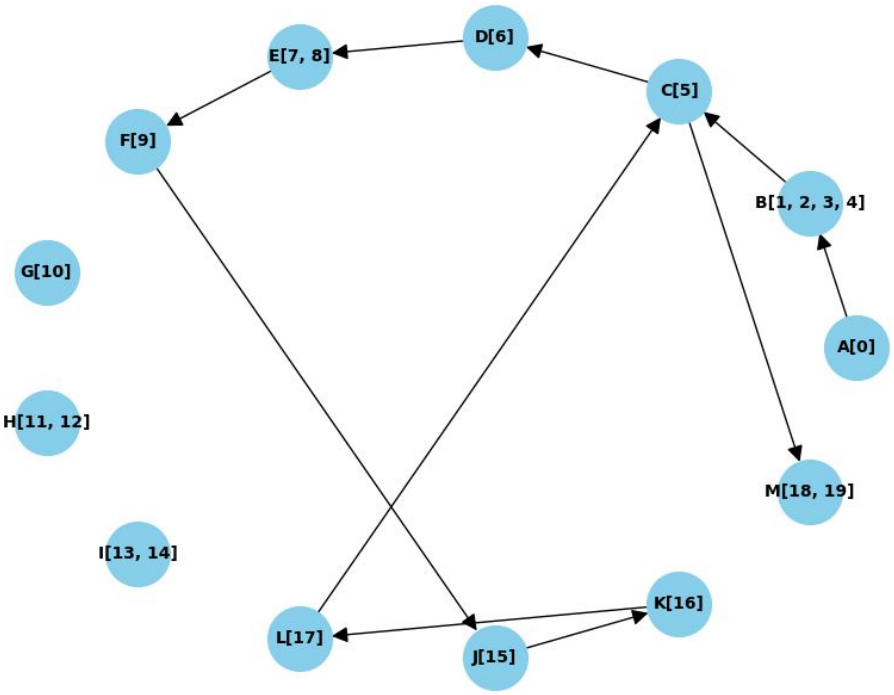


Risultati - Quick Sort

Analizziamo i singoli percorsi trovati

1. A-->B-->C-->D-->E-->F-->J-->K-->L-->C-->M

```
1   int l, r, pivot;
2   pivot = V[0];
3   l = 0;
4   r = N;
5   while ( l < r ) {
6         do {
7             r--;
8         }while ( V[r] > pivot && r > l );
9         if ( r != l ) {
10            do {
11                l++;
12            } while ( V[l] <= pivot && l < r );
13            swap(V, l, r);
14        }
15  }
16  swap(V, l, 0);
```

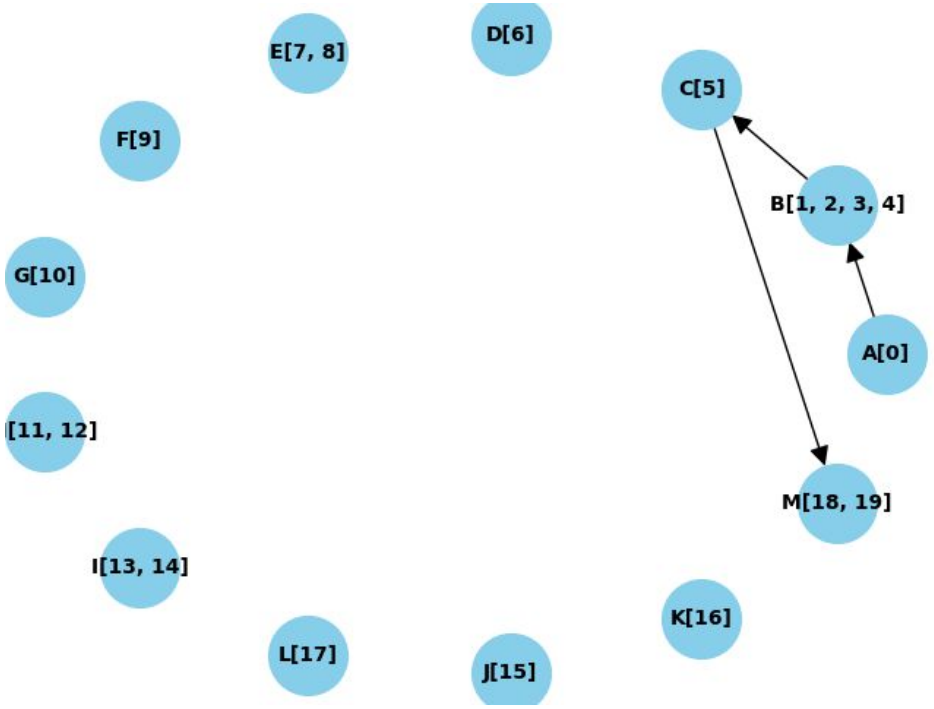


Risultati - Quick Sort

Analizziamo i singoli percorsi trovati

1. A-->B-->C-->M

```
1   int l, r, pivot;
2   pivot = V[0];
3   l = 0;
4   r = N;
5   while ( l < r ) {
6         do {
7             r--;
8         }while ( V[r] > pivot && r > l );
9         if ( r != l ) {
10            do {
11                l++;
12            } while ( V[l] <= pivot && l < r );
13            swap(V, l, r);
14        }
15  }
16  swap(V, l, 0);
```



Risultati - Quick Sort

I percorsi indipendenti trovati e la Test Suite generata

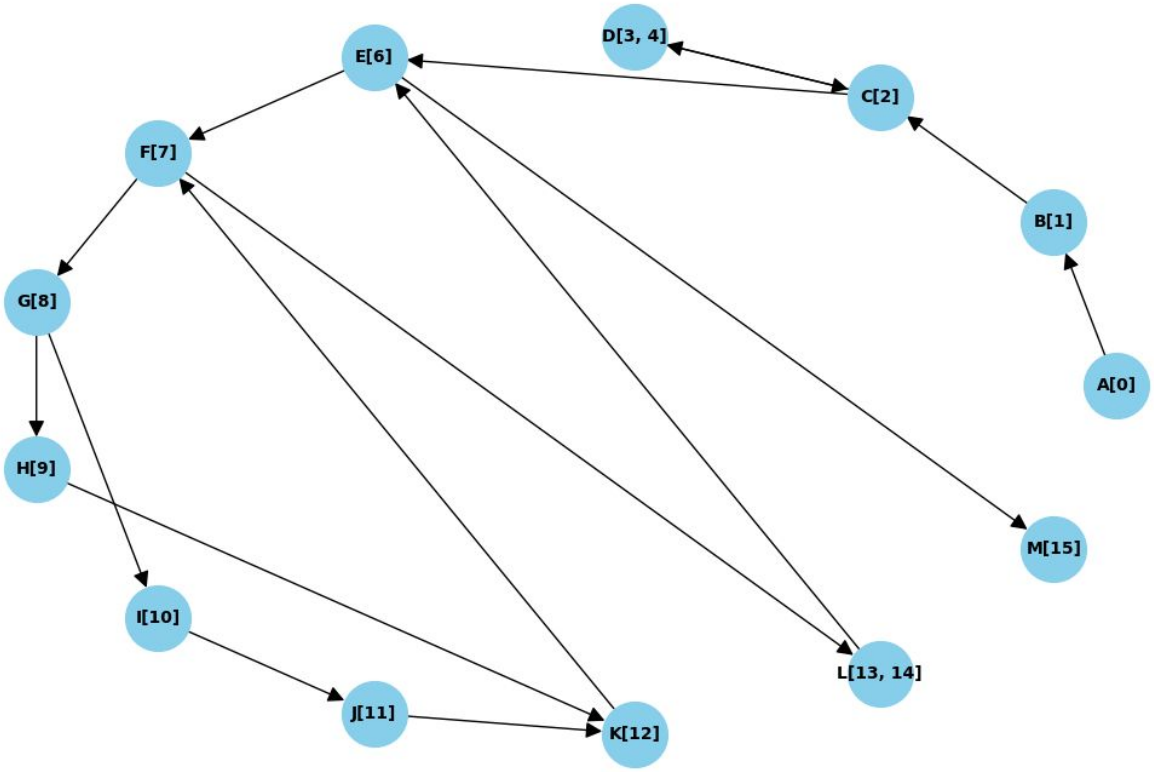
```
1       int l, r, pivot;
2       pivot = V[0];
3       l = 0;
4       r = N;
5       while ( l < r ) {
6             do {
7                 r--;
8             }while ( V[r] > pivot && r > l );
9             if ( r != l ) {
10                do {
11                    l++;
12                } while ( V[l] <= pivot && l < r );
13                swap(V, l, r);
14            }
15      }
16      swap(V, l, 0);
```

The Test Suite:

0. l < r ; V[r] > pivot ; r > l ; r != l ; V[l] < pivot ; l < r ;
1. l < r ; V[r] > pivot ; r > l ; r == l ; V[l] >= pivot ;
2. l >= r ;

Risultati - Selection Sort

Vediamo il CFG dell'algoritmo di Selection Sort




```
The graph:
A-->(B)
A[0]
B-->(C)
B[1]
C-->(D,E)
C[2]
D-->(C)
D[3, 4]
E-->(F,M)
E[6]
F-->(G,L)
F[7]
G-->(H,I)
G[8]
H-->(K)
H[9]
K-->(F)
K[12]
I-->(J)
I[10]
J-->(K)
J[11]
L-->(E)
L[13, 14]
M-->()
M[15]
```

Risultati - Selection Sort

I percorsi indipendenti trovati e la Test Suite generata

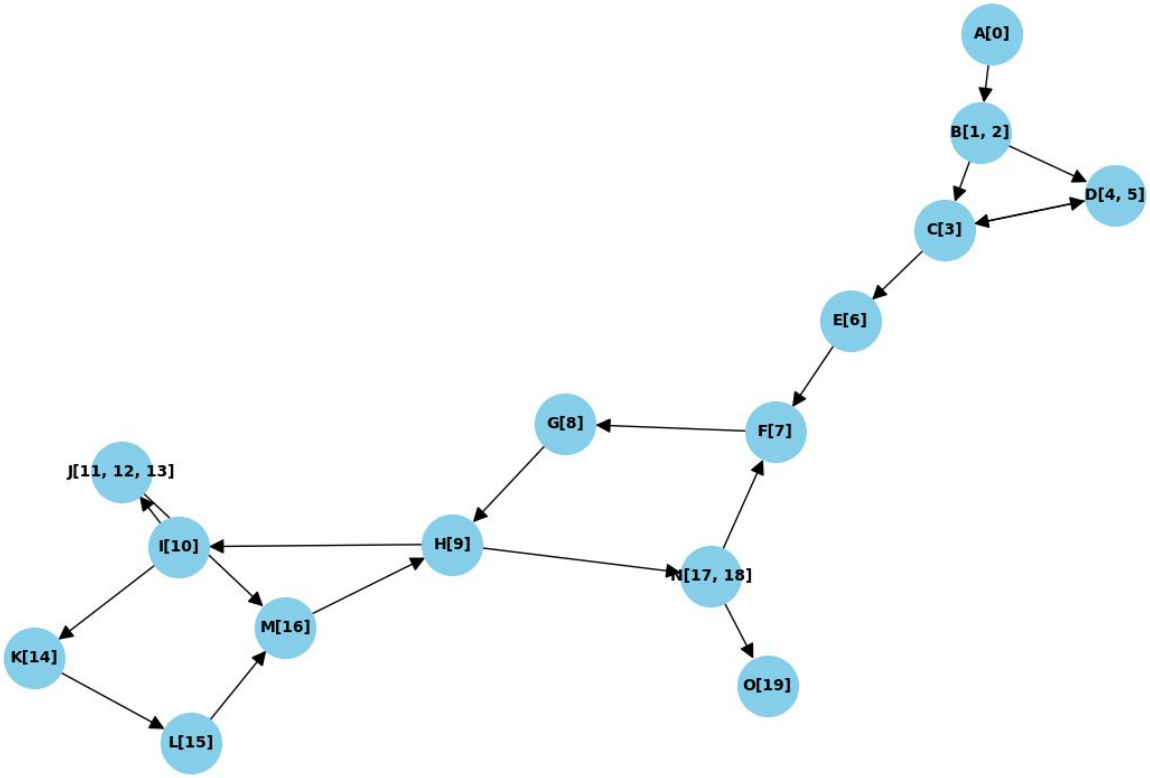
```
int iter, count, count_of_max;
for ( count=0; count<N; count++ ) {
   Perm[count] = count;
}

for ( iter=0; iter<N-1; iter++ ) {
    for (count=1, count_of_max=0; count<N-iter; count++) {
        if (V[Perm[count]] > V[Perm[count_of_max]])
            count_of_max = count;
    }
    swap(Perm,count_of_max,N-iter-1);
}
```

```
The Test Suite:
0.  count < N;  iter < N-1;  count < N-iter; V[Perm[count]]  >  V[Perm[count_of_max]];
1.  count < N;  iter < N-1;  count < N-iter; V[Perm[count]]  <=  V[Perm[count_of_max]];
2.  count < N;  iter < N-1;  count >= N-iter;
3.  count < N;  iter >= N-1;
4.  count >= N;  iter < N-1;  count < N-iter; V[Perm[count]]  >  V[Perm[count_of_max]];
5.  count >= N;  iter < N-1;  count < N-iter; V[Perm[count]]  <=  V[Perm[count_of_max]]; V[Perm[count]]  <=  V[Perm[count_of_max]];
6.  count >= N;  iter < N-1;  count >= N-iter;
7.  count >= N;  iter >= N-1;
```

Risultati - Bubble Sort

Vediamo il CFG dell'algoritmo di Bubble Sort




```
The graph:
A-->(B)
A[0]
B-->(C)
B[1, 2]
C-->(D,E)
C[3]
D-->(C)
D[4, 5]
E-->(F)
E[6]
F-->(G)
F[7]
G-->(H)
G[8]
H-->(I,N)
H[9]
I-->(J,K)
I[10]
J-->(M)
J[11, 12, 13]
M-->(H)
M[16]
K-->(L)
K[14]
L-->(M)
L[15]
N-->(F,O)
N[17, 18]
O-->()
O[19]
```

Risultati - Bubble Sort

I percorsi indipendenti trovati e la Test Suite generata

```
1    int iter, count;
2    boolean swap_found;
3    for (count=0; count<N; count++) {
4       Perm[count] = count;
5    }
6    iter = 0;
7    do {
8        swap_found=FALSE;
9        for (count=0; count < N-iter-1; count++) {
10           if (V[Perm[count]] > V[Perm[count+1]]) {
11               swap(Perm, count, count+1);
12               swap_found = TRUE;
13           }
14       }
15       iter ++;
16   }while(swap_found==TRUE);
```

The Test Suite:

0. count < N; count < N-iter-1; V[Perm[count]] > V[Perm[count+1]]; swap_found == TRUE;
1. count < N; count < N-iter-1; V[Perm[count]] <= V[Perm[count+1]];swap_found != TRUE;
swap_found == TRUE;
2. count < N; count >= N-iter-1; swap_found == TRUE;
3. count >= N; count < N-iter-1; V[Perm[count]] > V[Perm[count+1]]; swap_found == TRUE;
4. count >= N; count < N-iter-1; count >= N; V[Perm[count]] <= V[Perm[count+1]];
swap_found != TRUE; count >= N; swap_found != TRUE; count >= N; swap_found != TRUE;
V[Perm[count]] <= V[Perm[count+1]]; swap_found != TRUE; swap_found == TRUE;
5. count >= N; count >= N-iter-1; swap_found == TRUE;

Grazie per l’attenzione