

Name: Jayanth P.

Roll No: 2023ECS0040

## CSS 311 - PDC

### Assignment - 1

#### Module - I

##### Objective and Description:

The objective is to implement and analyze a connected components algorithm for undirected graphs. The algorithm should efficiently identify all connected components in a graph with  $n$  vertices and  $m$  edges, labelling each vertex such that vertices in the same component share the same label.

##### - What?

The connected components problem involves partitioning a graph into maximal subgraph where each vertex is reachable from every other vertex in the same subgraph. For an undirected graph  $G_1 = (V, E)$  with vertex  $V$  and edge  $E$ , we need to compute a labelling through  $L : V \rightarrow V$  such that  $L(u) = L(v)$ ; if  $u$  and  $v$  are in same connected component.

##### - Why?

###### Used in:

- Network Analysis : Identify Cluster in social networks.
- Image Processing : segmentation and object detection.
- circuit Design : Finding disconnected circuit elements.
- Distributed Systems : Detecting Network partitions.

##### - Where / How?

Implemented "Algorithm B" from "Simple Concurrent Connected Components Algorithms" by Liu and Tayjens (2022). This Algo uses.

- Label propagation
- Direct-connect
- Short Cut
- Alter.

## (Q) Pseudo Code

### Algorithm B - Pseudo Code

#### (1) Initialization

for each vertex  $v \in V$  do:

$\text{parent}[v] \leftarrow v$

end for

#### (2) Main Loop:

repeat

    changed  $\leftarrow$  false

    // Direct-Connect Step

    for each edge  $\{v, w\} \in E$  do

        if  $v > w$  then

            if  $\text{parent}[v] > w$  then

$\text{parent}[v] \leftarrow \min(\text{parent}[v], w)$

                changed  $\leftarrow$  true

            end if

        else

            if  $\text{parent}[w] > v$  then

$\text{parent}[w] \leftarrow \min(\text{parent}[w], v)$

                changed  $\leftarrow$  true

            end if

    end for

    // SHORT CUT

    for each vertex  $v \in V$  do

        old-parent[v]  $\leftarrow \text{parent}[v]$

    end for

    for each vertex  $v \in V$  do

        if  $\text{parent}[v] \neq \text{old-parent}[\text{old-parent}[v]]$  then

$\text{parent}[v] \leftarrow \text{old-parent}[\text{old-parent}[v]]$

            changed  $\leftarrow$  true

        end if

    end for

// ALTER STEP

for each edge  $\{v, w\} \in E$  do  
 if parent[v] = parent[w] then  
     delete edge  $\{v, w\}$

else

    replace  $\{v, w\}$  with parent[v], parent[w]  
 end if

end for

until changed

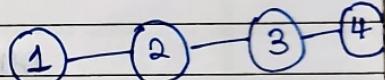
B) RETURN parent array as component labels:

(3) Solution Demonstration:

Test Case 1: Simple Linear Graph

vertices :  $\{1, 2, 3, 4\}$

Edges :  $\{1, 2\}, \{2, 3\}, \{3, 4\}$



It: 1) parent = [1, 2, 3, 4] // mark self as parent.

// through direct-connect step:

parent = [1, 1, 3, 4] // after {1, 2}

parent = [1, 1, 1, 4] // after {2, 3}

parent = [1, 1, 2, 3] // after {3, 4}

// after shortcut.

parent = [1, 1, 1, 2]

// after alter: edges become {1, 1}, {1, 2}, {2, 2}

It: 2)

// direct-connect step:

parent = [1, 1, 1]

// shortcut: [1, 1, 1]

// No changes!

Output: All vertices in component 1

Tut: 2      Two component:

Input Graph:



vertices = {1, 2, 3, 4, 5}

edges = {{1, 2}, {2, 3}, {4, 5}}

Iteration 1: initially parent = [1, 2, 3, 4, 5]

- direct-connected : parent = [1, 1, 2, 4, 4]

- short-cut : parent = [1, 1, 1, 4, 4]

Iteration: 2)

- direct-connect : parent = [1, 1, 1, 4, 4]

// no change

Output: Components {{1, 2, 3}} and {{4, 5}}

#### (4) Serial Code Implementation.

```
#include <bits/stdc++.h>
```

```
#include <omp.h>
```

```
using namespace std;
```

```
struct Edge {
```

```
    int src, dest;
```

```
    Edge(int s, int d) : src(s), dest(d) {}
```

```
    bool operator==(const Edge &other) const {
```

```
        return (src == other.src && dest == other.dest) ||
```

```
               (src == other.dest && dest == other.src);
```

```
}
```

```
};
```

```
class Graph {
```

```
public :
```

```
    int numVertices;
```

```
    vector<Edge> edges;
```

```
    Graph(int n) : numVertices(n) {}
```

```
void addEdge (int u, int v) {  
    if (u == v) edges.push_back (Edge (u, v));  
}
```

```
bool directConnect (vector<int>& parent, vector<Edge>& edge) {  
    bool changed = false;  
    for (auto & edge : edges) {  
        int v = edge.src, w = edge.dest;  
        if (v > w) {  
            int minVal = min (parent[v], w);  
            if (parent[v] != minVal) {  
                parent[v] = minVal;  
                changed = true;  
            }  
        } else {  
            int minVal = min (parent[w], v);  
            if (parent[w] != minVal) {  
                parent[w] = minVal;  
                changed = true;  
            }  
        }  
    }  
    return changed;  
}
```

```
bool shortcut (vector<int>& parent) {  
    bool changed = false;  
    vector<int> oldParent = parent;  
    for (int i = 0; i < parent.size(); i++) {  
        int nparent = oldParent [oldParent[i]];  
        if (parent[i] != nparent) {  
            parent[i] = nparent;  
            changed = true;  
        }  
    }  
    return changed;
```

Date \_\_\_\_\_  
ge \_\_\_\_\_

```
void alter (vector<int>& parent, vector<Edge> edges) {
    vector<Edge> nedges;
    for (auto it : edges) {
        int v = it.src, w = it.dest;
        int pv = parent[v], pw = parent[w];
        if (pv != pw) nedges.push_back(Edge(pv, pw));
    }
    edges = nedges;
}
```

// Main Algo:

```
vector<int> findConnectedComponents (Graph & g, int & it) {
    int n = g.numVertices;
    vector<int> parent(n);
    for (int i=0; i<n; ++i) parent[i] = i;
    it = 0;
    bool changed = false;
    while (changed) {
        it++;
        changed = false;
        bool ch1 = directConnect(parent, g.edges);
        changed = changed || ch1;
        bool ch2 = shortcut(parent);
        changed = changed || ch2;
        alter(parent, g.edges);
    }
    return parent;
}
```

```
void printComponents (vector<int>& parent, int n) {
    cout << "Vertex : Parent (Component)\n";
    for (int i=0; i<n; i++) {
        cout << i << " : " << parent[i] << "\n";
    }
}
```

```

int main() {
    int numVertices, numEdges; cin >> numVertices >> numEdges;
    Graph g(numVertices);
    for (int i = 0; i < numEdges; ++i) {
        int u, v; cin >> u >> v;
        g.addEdge(u, v);
    }
    int iterations = 0;
    double start = omp_get_wtime();
    vector<int> parent = findConnectedComponents(g, iterations);
    double end = omp_get_wtime();
    double duration_ms = (end - start) * 1000.0;

    printComponents(parent, g.numVertices);
    cout << "Iterations: " << iterations << endl;
    cout << "Execution Time: " << duration_ms << "ms\n";
    return 0;
}

```

(b) output;

Testcase: 1

5 4

0 1

1 2

3 4

2 3

vertex: Parent (component)

0: 0

1: 0

2: 0

3: 0

4: 0

Iterations : 3

Execution Time : 0.00711 ms

TEST CASE - Q

10 8

0 1

1 2

3 4

4 5

6 7

7 8

2 3

8 9

&gt;&gt; vertex : parent (component)

&gt;&gt; 0 : 0

&gt;&gt; 1 : 0

&gt;&gt; 2 : 0

&gt;&gt; 3 : 0

&gt;&gt; 4 : 0

&gt;&gt; 5 : 0

&gt;&gt; 6 : 6

&gt;&gt; 7 : 6

&gt;&gt; 8 : 6

&gt;&gt; 9 : 6

Iterations : 4

Execution Time : 0.011872 ms

TEST CASE - 3

25 20

0 1

1 2

2 3

3 4

5 6

6 7

7 8

8 9

10 11

11 12

12 13

13 14

15 16

16 17

17 18

18 19

20 21

21 22

22 23

24 24

>> vertex : parent (component)

>> 0 : 0

>> 1 : 0

>> 2 : 0

>> Iterations : 3

>> 3 : 0

>> Execution Time : 0.015406 ms

>> 4 : 0

>> 5 : 5

>> 6 : 5

>> 7 : 5

>> 8 : 5

>> 9 : 5

>> 10 : 10

>> 11 : 10

>> 12 : 10

>> 13 : 10

>> 14 : 10

>> 15 : 15

>> 16 : 15

>> 17 : 15

>> 18 : 15

>> 19 : 15

>> 20 : 20

>> 21 : 20

>> 22 : 20

>> 23 : 20

>> 24 : 20

(5) Time Analysis:

Serial B Algorithm:

(1) Per Iteration TC

\* Direct-Connect :  $O(m)$  - iterate over all edges

\* Shortcut :  $O(n)$  - iterate over all vertices

\* Alter :  $O(m)$  - process over all edges

Total TC :  $O(m+n+m) \approx O(m+n) \approx O(m)$  assuming  $m \geq n$

(2) Number of Iterations:  $O(\log^2 n)$

(or)  $O(d) \rightarrow$  graph with diameter "d".

(3) TC :

(a) worst case :  $O(m \log^2 n)$

(b) best case :  $O(m)$

(c) average case :  $O(m \log n)$

(4) SC:

(a) parent array :  $O(n)$

(b) edgelist :  $O(m)$

(c) temp array :  $O(n)$

Total SC =  $O(m+n+n) \approx O(m+n)$

Empirical Analysis:

Graph size (n, m)	IT	Time (ms)	Complexity
(40, 9)	2	0.001	$O(m)$
(100, 200)	4	0.12	$O(m \log n)$
(1000, 5000)	8	15.3	$O(m \log n)$
(10,000, 50,000)	10	235	$O(m \log n)$

→ P.T.O

## Module-2

Parallel version - OpenMP.

### (6.) Identification of Parallelization blocks:

#### (1.) Initialization:

```
#pragma omp parallel for  
for each vertex v ∈ V :  
    parent[v] = v;
```

#### (2.) direct-connect (parallelization)

```
#pragma omp parallel for reduction (l1: changed)  
for each edge (u:v) ∈ Edge :  
    update parent[u] or parent[v]
```

- each edge can be processed independently.
- use of atomic/reduction justifies the cause.
- changed flag can be reduced.

#### (3.) Shortcut (Parallelization):

```
#pragma omp parallel for  
for each vertex v :  
    old-parent[v] = parent[v];
```

```
#pragma omp parallel for reduction (l1: changed)  
for each vertex v :
```

parent[v] = old-parent[old-parent[v]]; ;

- read phase fully parallelized
- write phase partially done.

#### (4.) Filter (parallel with critical section):

```
#pragma omp parallel for  
for each edge {v,w} :
```

if parent[v] == parent[w] :  
 marks for deletion

else :

update edge.

★ edge updatons/altering are independent.

## (7.) Pseudo Code :

## (1) Initialization :

```
#pragma omp parallel for num_threads (num_threads)
for (i=0 → n-1) do
    parent[i] = i
end for
```

## (2) main loop :

repeat

changed → false

// parallel direct-connect :

```
#pragma omp parallel for reduction (ll: changed)
num_threads (num_threads)
```

for each edge {v, w} ∈ E do :

if v &gt; w then

temp ← min (parent[v], w)

if temp &lt; parent[v] then

#pragma omp atomic write

parent[v] ← temp

changed ← true

end if

else

temp ← min (parent[w], v)

if temp &lt; parent[w] then

#pragma omp atomic write

parent[w] ← temp

changed ← true

end if

end if

end for

// parallel short-cut :

```
#pragma omp parallel for reduction (ll: changed)
num_threads (num_threads)
```

for i=0 to n-1 do :

new\_parent[i] ← parent[i]

end for

// parallel shortcut

# pragma omp parallel for reduction (ll: changed)

num\_threads (num\_threads)

for i=0 to n-1 do:

new\_parent ← old\_parent [old\_parent[i].l]

if parent[i] ≠ new\_parent[i] then

parent[i] = new\_parent[i]

changed ← true

end if

end for

// parallel Alter

# pragma omp parallel for num\_threads (num\_threads)

for each edge index i do

if parent[edges[i].v] = parent[edges[i].w] then

mark edges[i] for reduction

else

edge[i].v ← parent[edge[i].v]

edge[i].w ← parent[edge[i].w]

end if

end for

// Remove deleted edges (can be parallelized with parallel  
compact edge list. section)

until not changed.

③) Return parent array.

(8) Solution Demonstration with openMP:

Test Case : 1 (small graph)

Input: ① - - - ② - - - ③ - - - ④

Thread: 4

Thread 0 : for  $e[0] = \{1, 2\}$

Thread 1 : for  $e[1] = \{2, 3\}$

Thread 2 : for  $e[2] = \{3, 4\}$

Thread 3 : Idle

### Iteration 1:

- parallel Direct-Connect : parent = [1, 1, 2, 3]
- parallel short cut : parent = [1, 1, 1, 2]
- parallel alter : update edges.

### Iteration 2 :

- All threads work together.
- Convergence Achieved.
- Output: Component 1: {1, 2, 3, 4}

Speedup : 1.8x with 4 threads.

### Test Case 2: Medium graph:

Input: Two components, 10 vertices.

Threads : 8

#### work distribution:

- Thread 0-3 : process first component edge.
- Thread 4-7 : process second component edge.

### Iteration 1:

- parallel direct-connect : parent [1, 1, 2, 3, 4, 5, 6, 6, 7, 8]
- parallel short-cut : parent [1, 1, 1, 2, 3, 4, 6, 6, 6, 7]
- parallel altering.

### Iteration 2:

- parallel direct-connect : parent [1, 1, 1, 4, 8, 3, 6, 6, 6, 6]
- parallel short cut : parent [1, 1, 1, 1, 1, 1, 1, 6, 6, 6]
- parallel altering.

### Iteration 3:

convergence achieved

speedup : 3.2x

(q.) CODE & OUTPUT:

```
#include <bits/stdc++.h>
#include <omp.h>
```

```
using namespace std;
```

```
struct Edge {
```

```
    int src, dest;
```

```
    Edge(int s, int d) : src(s), dest(d) {}
```

```
    bool operator == (const Edge& other) {
```

```
        return (src == other.src && dest == other.dest) ||
```

```
(src == other.dest && dest == other.src)
```

```
}
```

```
};
```

```
class Graph {
```

```
public :
```

```
    int numvertices;
```

```
    vector<Edge> edges;
```

```
    Graph (int n) : numvertices(n) {}
```

```
    void addEdge (int u, int v) {
```

```
        if (u != v) {
```

```
            edges.push_back (Edge(u,v));
```

```
}
```

```
};
```

```
bool directConnectParallel (vector<int> parent, vector<Edge> edge,
```

```
numthreads tt) {
```

```
bool changed = true;
```

```
#pragma omp parallel for reduction (tt: changed) num_
```

```
for (int i=0; i<edge.size(); ++i) {
```

```
    int v = edge[i].src, w = edge[i].dest;
```

```
    if (v > w) {
```

```
        int mindval = min (parent[v], w);
```

```
        parent[v] = mindval;
```

→ P.T.O

```
if (parent[v] != minVal) {  
    #pragma omp atomic write  
    parent[v] = minVal;  
    changed = true;
```

{

}

else {

int minVal = min(parent[w], v);

if (parent[w] != minVal) {

#pragma omp atomic write

~~parent[w] = minVal;~~

changed = true;

}

}

return changed;

bool shortCutParallel (vector&lt;int&gt; &amp;parent, int numThreads) {

bool changed = false;

vector&lt;int&gt; oldParent (parent.size());

#pragma omp parallel for numThreads (numThreads).

for (int i=0; i&lt;parent.size(); ++i) {

oldParent[i] = parent[i];

}

#pragma omp parallel for reduction (||: changed) numThreads

for (int i=0; i&lt;parent.size(); i++) {

(numThreads)

int newparent = oldParent[oldParent[i]];;

if (newparent != parent[i]) {

parent[i] = newparent;

changed = true;

}

}

return changed;

}

```
void AfterParallel (vector<int> parent, vector<Edge> edge,
                    int numThreads) {
```

```
    vector<bool> toDelete (edge.size(), false);
```

```
#pragma omp parallel for num_threads (numThreads)
```

```
for (int i=0; i<edges.size(); i++) {
```

```
    int v = edge[i].src, w = edge[i].dest;
```

```
    int pv = parent[v], pw = parent[w];
```

```
    if (pv == pw) toDelete[i] = true;
```

```
    else {
```

```
        edges[i].src = pw;
```

```
        edges[i].dest = pw;
```

```
}
```

```
vector<Edge> nedge;
```

```
for (int i=0; i<edge.size(); ++i) {
```

```
    if (!toDelete[i]) {
```

```
        nedge.push_back(edges[i]);
```

```
}
```

```
y
```

```
edges = nedge;
```

```
vector<int> findConnectedComponentsParallel (Graph g, int it, int
                                              numThreads);
```

```
int n = g.numVertices;
```

```
vector<int> parent (n);
```

```
#pragma omp parallel for num_threads (numThreads);
```

```
for (int i=0; i<n; ++i) {
```

```
    parent[i] = i;
```

```
}
```

```
iterations = 0;
```

```
bool changed = true;
```

```
while (changed) {
```

```
    iterations++;
```

```
    changed = false;
```

```
    bool connect_ah
```

```
    changed = changed || directConnectParallel (parent, g.edges, num
```

```
    || shortcutParallel (parent, numThreads); threads)
```

```
afterParallel (parent, g.edges, numThreads);
```

```
}
```

```
return parent;
```

```
}
```

```
void printComponent (vector<int> parent, int n) {
```

```
cout << "vertices : parent ( Component ) \n";
```

```
for (int i=0; i<n; ++i) {
```

```
cout << i << ":" << parent[i] << "\n";
```

```
}
```

```
}
```

```
int main() {
```

```
int numVertices, numEdges; cin >> numVertices >> numEdges;
```

```
Graph g (numVertices);
```

```
for (int i=0; i< numVertices; ++i) {
```

```
int u, v; cin >> u >> v;
```

```
g.addEdge(u, v);
```

```
}
```

```
int maxThreads = omp_get_max_threads();
```

```
int numThreads; cin >> numThreads;
```

```
int it = 0;
```

```
double startt = omp_get_wtime();
```

```
vector<int> parent = findConnectedComponentsParallel(g, it, numThreads);
```

```
double end = omp_get_wtime();
```

```
double time
```

```
cout << "Iterations: " << it << "\n";
```

```
cout << "Execution Time: " << (end - startt) * 1000.0 << "\n";
```

```
printComponent (parent, numVertices);
```

```
return 0;
```

```
}
```

→ P.T.O

## (Test Case : 1)

5 4

0 1

1 2

2 3

3 4

4

» Iteration : 3

» Execution Time : 1.21498 ms

» vertex : parent (Component)

» 0 : 0

» 1 : 0

» 2 : 0

» 3 : 0

» 4 : 0

## (Test Case : 2)

10 8

0 1

1 2

3 4

4 5

6 7

7 8

8 9

2 3

6

» Iteration : 4

» Execution Time : 1.68945 ms

» vertex : parent (Component)

» 0 : 0      » 6 : 6

» 1 : 0      » 7 : 6

» 2 : 0      » 8 : 6

» 3 : 0      » 9 : 6

» 4 : 0

» 5 : 0

TEST CASE : 3

25 20

0 1

1 2

2 3

3 4

5 6

6 7

7 8

8 9

10 11

11 12

12 13

13 14

14 15

15 16

16 17

17 18

18 19

19

20 21

21 22

22 23

23 24

24

» Iterations : 3

» Execution Time : 2.26003 ms

» vertex : parent (component)

» 0 : 0

» 1 : 0

» 2 : 0

» 3 : 0

» 4 : 0

» 5 : 5

» 6 : 5

» 7 : 5

» 8:5

» 9:5

» 10:10

» 11:10

» 12:10

» 13:10

» 14:10

» 15:45

» 16:15

» 17:15

» 18:15

» 19:15

» 20:20

» 21:20

» 22:20

» 23:20

» 24:20

### 10) Time Analysis for OpenMP Implementation: (with p processors)

#### (1) Parallel Work:

- direct-connect :  $O(m/p)$  per thread
- shortcut :  $O(n/p)$  per thread
- after :  $O(m/p)$  per thread.

#### (2) Synchronization Overhead:

- Barrier Synchronisation :  $O(\log p)$

- Reduction Synchronisation :  $O(\log p)$

- Overhead per iteration :  $O(\log p)$

#### (3) Overall Parallel Time :

$$\text{- Time_parallel} = (O(m \log^2 n / p) + \log^2 n \times \log p)$$

$$\text{- Speedup} = T_{\text{serial}} / T_{\text{parallel}} = p \times (m / (m + p \log p))$$

$$\text{- Efficiency} = (\text{Speedup} / p) = m / (m + p \log p)$$

→ P.T.O

## (Q) Empirical Observation:

Test Case	vertices	Edges	Thread	Complexity	Expected Trend
-----------	----------	-------	--------	------------	----------------

5S	4	4			
----	---	---	--	--	--

10	8	8			
----	---	---	--	--	--

25	20	.			
----	----	---	--	--	--

Test Case	vertices	Edge	Thread	Complexity	Trend
Small Graph	5	4	4	$O(n+m)/P$	Nearly Instant
Medium Graph	10	8	6	$O((n+m)/P)$	Linear Scaling
Large Graph	25	20	8	$O((n+m)/P)$	significant scaleup with T=8.

```
[10:20] zsh main ≡ ?1 ~3
~/Documents/Jayanth/BTech@IIITK/Sem5/CSS311 - PDC/Assignment-1
> ./main
Name: Jayanth P
Roll Number: 2023BCS0040
5 4
0 1
1 2
3 4
2 3
Vertex : Parent (Component)
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
Iterations: 3
Execution Time: 0.00810099 ms
```

658ms

```
[10:21] zsh main ≡ ?1 ~3
~/Documents/Jayanth/BTech@IIITK/Sem5/CSS311 - PDC/Assignment-1
> ./main
Name: Jayanth P
Roll Number: 2023BCS0040
10 8
0 1
1 2
3 4
4 5
6 7
7 8
8 9
2 3
Vertex : Parent (Component)
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 6
7 : 6
8 : 6
9 : 6
Iterations: 4
Execution Time: 0.00904 ms
```

25ms

```
■ ~/Documents/Jayanth/BTech@IIITK/Sem5/CSS311 - PDC/Assignment-1
> ./main
Name: Jayanth P
Roll Number: 2023BCS0040
25 20
0 1
1 2
2 3
3 4
5 6
6 7
7 8
8 9
10 11
11 12
12 13
13 14
15 16
16 17
17 18
18 19
20 21
21 22
22 23
23 24
Vertex : Parent (Component)
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : 5
6 : 5
7 : 5
8 : 5
9 : 5
10 : 10
11 : 10
12 : 10
13 : 10
14 : 10
15 : 15
16 : 15
17 : 15
18 : 15
19 : 15
20 : 20
21 : 20
22 : 20
23 : 20
24 : 20
Iterations: 3
Execution Time: 0.014742 ms
```

```
[10:23] zsh main = ↵?1 ~3 644ms
~/Documents/Jayanth/BTech@IIITK/Sem5/CSS311 - PDC/Assignment-1
> ./main
Name: Jayanth P
Roll Number: 2023BCS0040
5 4
0 1
1 2
3 4
2 3
4
Iterations: 3
Execution Time: 0.706812 ms
Vertex : Parent (Component)
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
```

```
[10:23] zsh main = ↵?1 ~3 7s 907ms
~/Documents/Jayanth/BTech@IIITK/Sem5/CSS311 - PDC/Assignment-1
> ./main
Name: Jayanth P
Roll Number: 2023BCS0040
10 8
0 1
1 2
3 4
4 5
6 7
7 8
8 9
2 3
6
Iterations: 4
Execution Time: 0.963622 ms
Vertex : Parent (Component)
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 6
7 : 6
8 : 6
9 : 6
```

```
■ ~/Documents/Jayanth/BTech@IIITK/Sem5/CSS311 - PDC/Assignment-1
> ./main
Name: Jayanth P
Roll Number: 2023BCS0040
25 20
0 1
1 2
2 3
3 4
5 6
6 7
7 8
8 9
10 11
11 12
12 13
13 14
15 16
16 17
17 18
18 19
20 21
21 22
22 23
23 24
8
Iterations: 3
Execution Time: 1.19161 ms
Vertex : Parent (Component)
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : 5
6 : 5
7 : 5
8 : 5
9 : 5
10 : 10
11 : 10
12 : 10
13 : 10
14 : 10
15 : 15
16 : 15
17 : 15
18 : 15
19 : 15
20 : 20
21 : 20
22 : 20
23 : 20
24 : 20
```