How many key comparisons does this algorithm do for finding the min and the max of n=2k data items:

```
1. for (i=0; i < n; i+=2)
      if (A[i] > A[i+1]) swap(A[i], A[i+1])
Then use a linear scan (like in MaxSort) to
      2. Find the min of
            A[0], A[2], A[4], ... , A[n-2]
      3. Find the max of
            A[1], A[3], A[5], ... A[n-1]
```

Old final Exam Question

Answer true or false and justify your answer:

Since it takes at least n-1 key comparisons to find the min of n data items and it takes at least n-1 key comparisons to find the max of n data items, it takes at least 2n-2 key comparisons to find both the min and the max.

Announcements:

Assignment #5: Programming questions:
Upload your solution to connex by 11:55pm on Saturday Nov. 25.

Assignment #6: Hand this in on paper at the beginning of class on Thursday Nov. 30.

Special Final Exam Tutorial: Sat. Dec. 2: 1pm to 4pm in Elliott 168.

Graph Traversals

Two common types of graph traversals are **Depth First Search** (DFS) and **Breadth First Search** (BFS). A preorder traversal of a binary tree is a special case of DFS and a level order traversal is a special case of BFS. DFS is implemented with a stack, and BFS with a queue.

The aim in both types of traversals is to visit each vertex of a graph exactly once. In DFS, you follow a path as far as you can go before backing up. With BFS, you visit all the neighbours of the current node before exploring further afield in the graph.

Queue (used for BFS)

http://devcentral.f5.com/weblogs/images/devcentral_f5_com/weblogs/Joe/WindowsLiveWriter/PowerShellABCsQisforQueues_919A/queue_2.jpg

http://www.ac-nancy-metz.fr/enseign/anglais/Henry/bus-queue.jpg

Queue data structure:

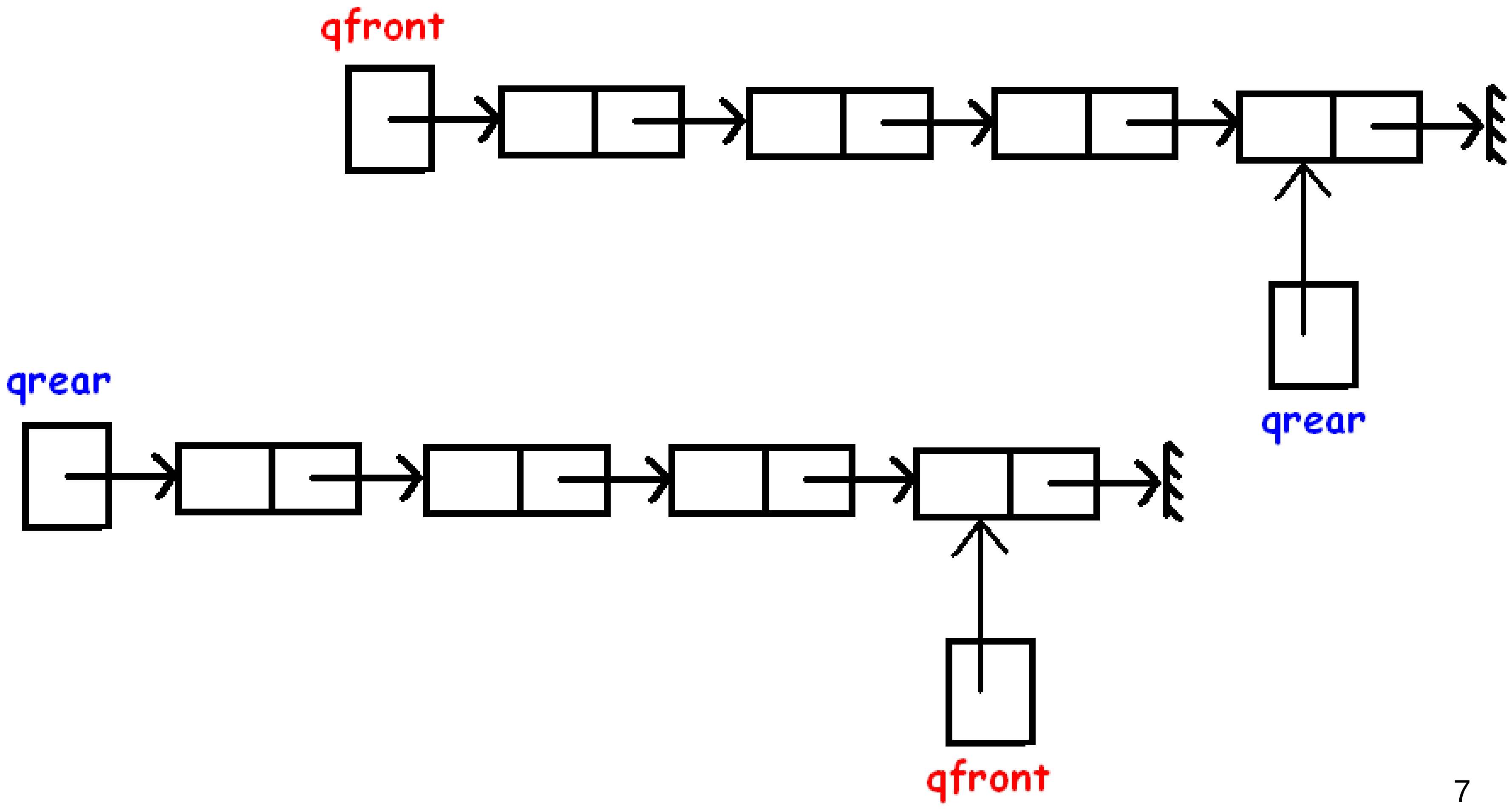
Items are:

Added to the rear of the queue.

Removed from the front of the queue.

http://cs.wellesley.edu/~cs230/assignments/lab12/queue.jpg

Queues can be implemented as a linked list. Which end of the list should we use for the queue front?

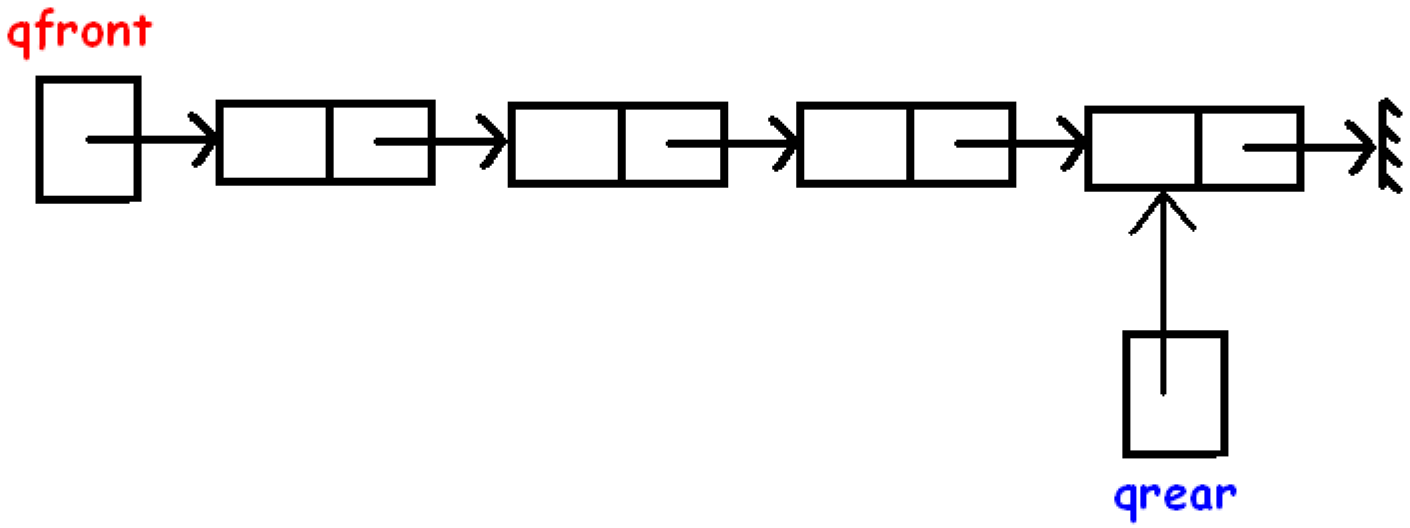


Queue data structure:

Items are:

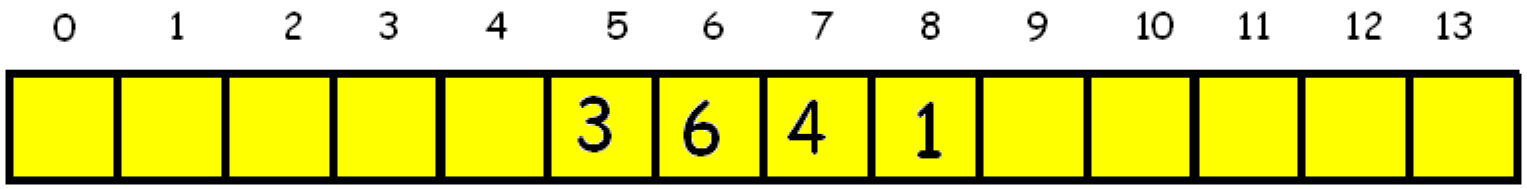
Added to the rear of the queue.

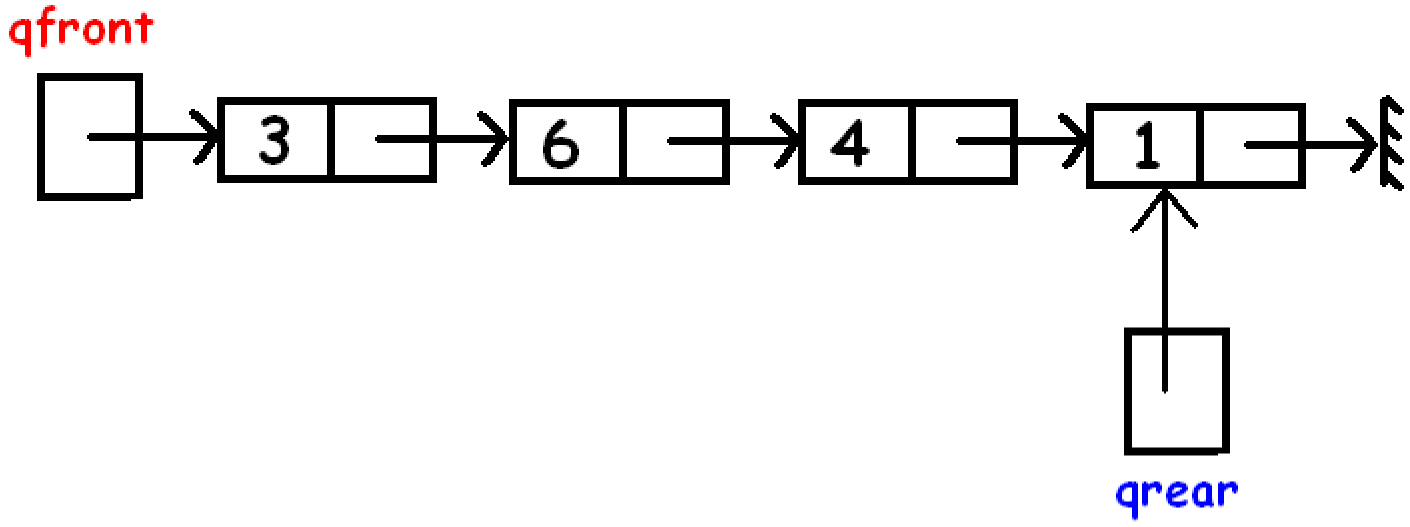
Removed from the front of the queue.



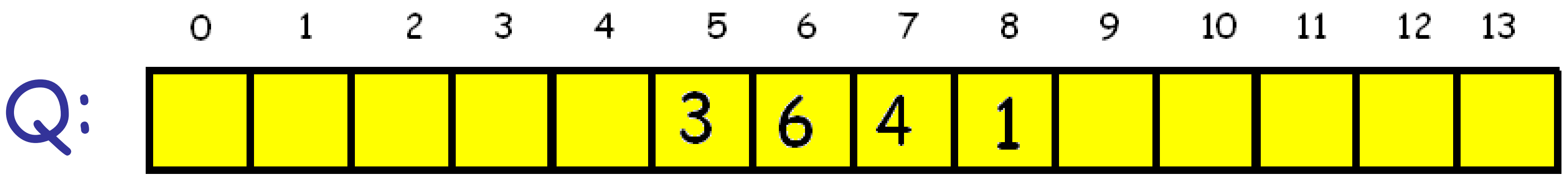
If you have an upper bound on the lifetime size of the queue then you can use an array: qfront=5, qrear=9

(qrear is next empty spot in array)





qfront=5, qrear=9



To test if there is something in the queue:

if (qfront < qrear)

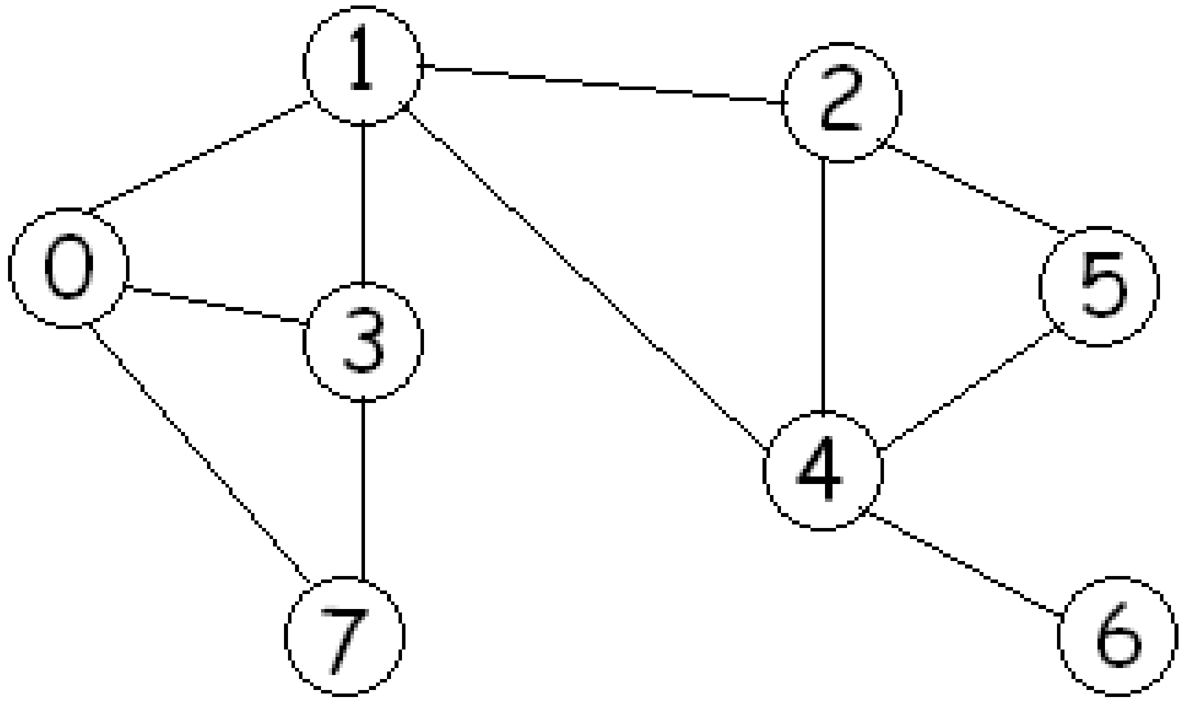
To add x to the queue:

Q[qrear]= x; qrear++;

To delete front element of the queue:

x= Q[qfront]; qfront++;

If the neighbours of each vertex are ordered according to their vertex numbers, in what order does a BFS starting at 0 visit the vertices?



BFS starting at a vertex s using an array for the queue:

Data structures:

A queue Q[0..(n-1)] of vertices, qfront, qrear.

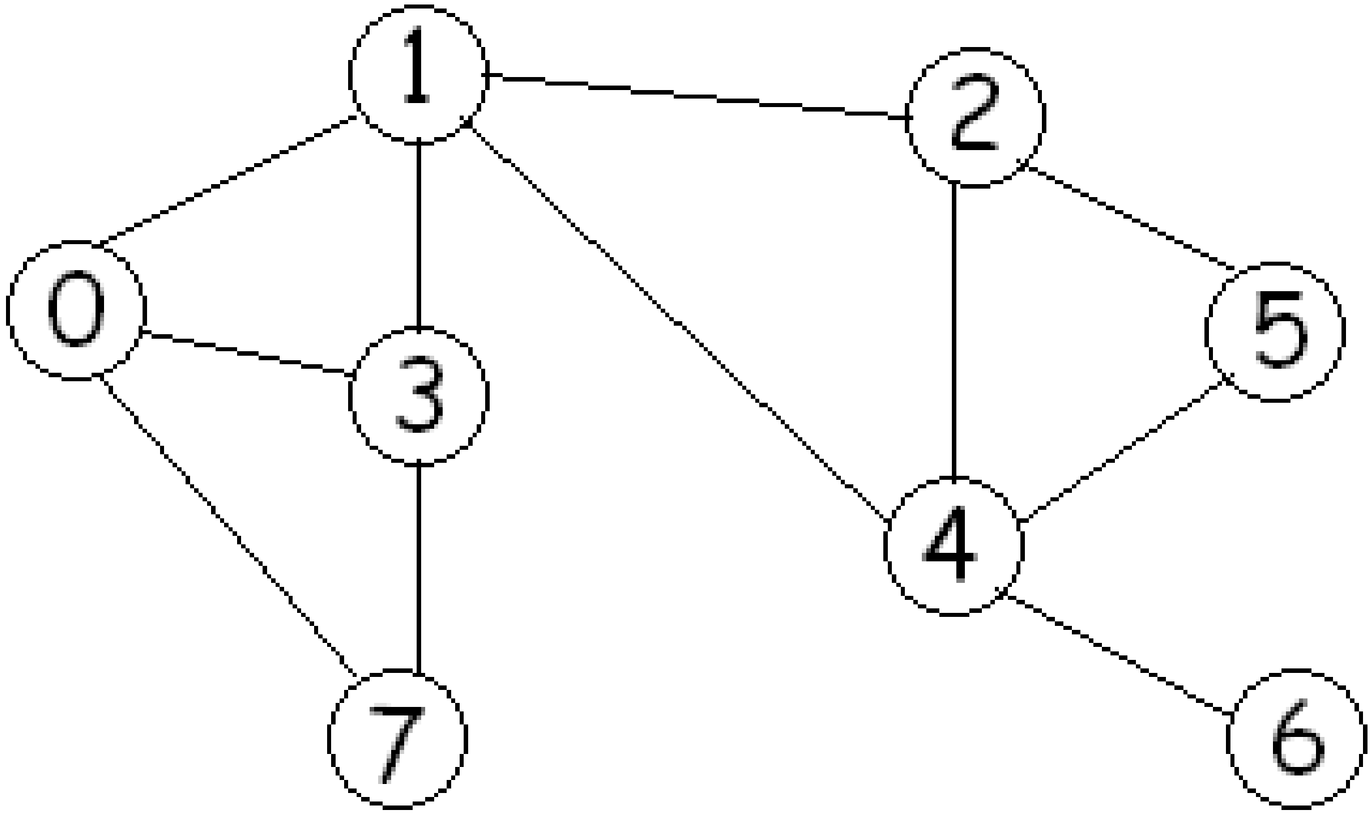
parent[i]= BFS tree parent of node i.

The parent of the root s is s.

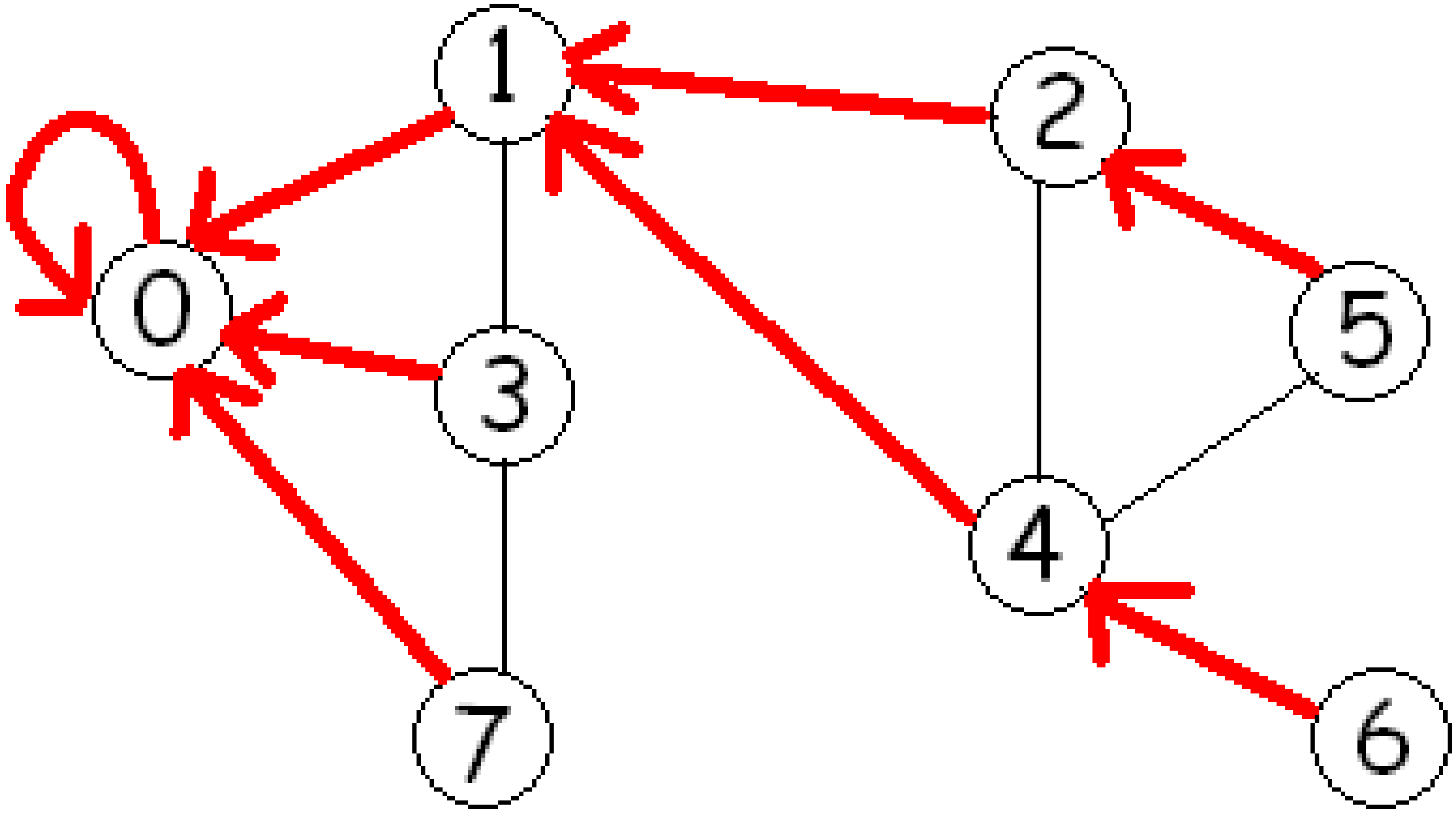
To initialize:

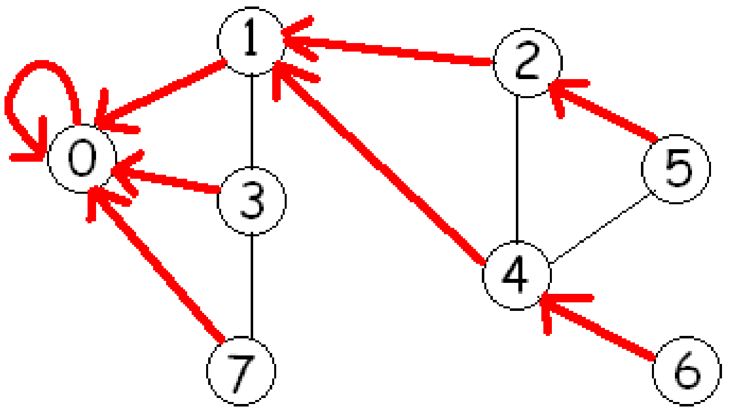
```
// Set parent of each node to be -1 to indicate
// that the vertex has not yet been visited.
for (i=0; i < n; i++) parent[i]= -1;

// Initialize the queue so that BFS starts at s
qfront=0; qrear=1; Q[qfront]= s;
parent[s]=s;
```

```
while (qfront < qrear) // Q is not empty
    u= Q[qfront]; qfront++;
    for each neighbour v of u
        if (parent[v] == -1) // not visited
            parent[v]= u;
            Q[qrear]= v; qrear++;
        end if
    end for
end while
```

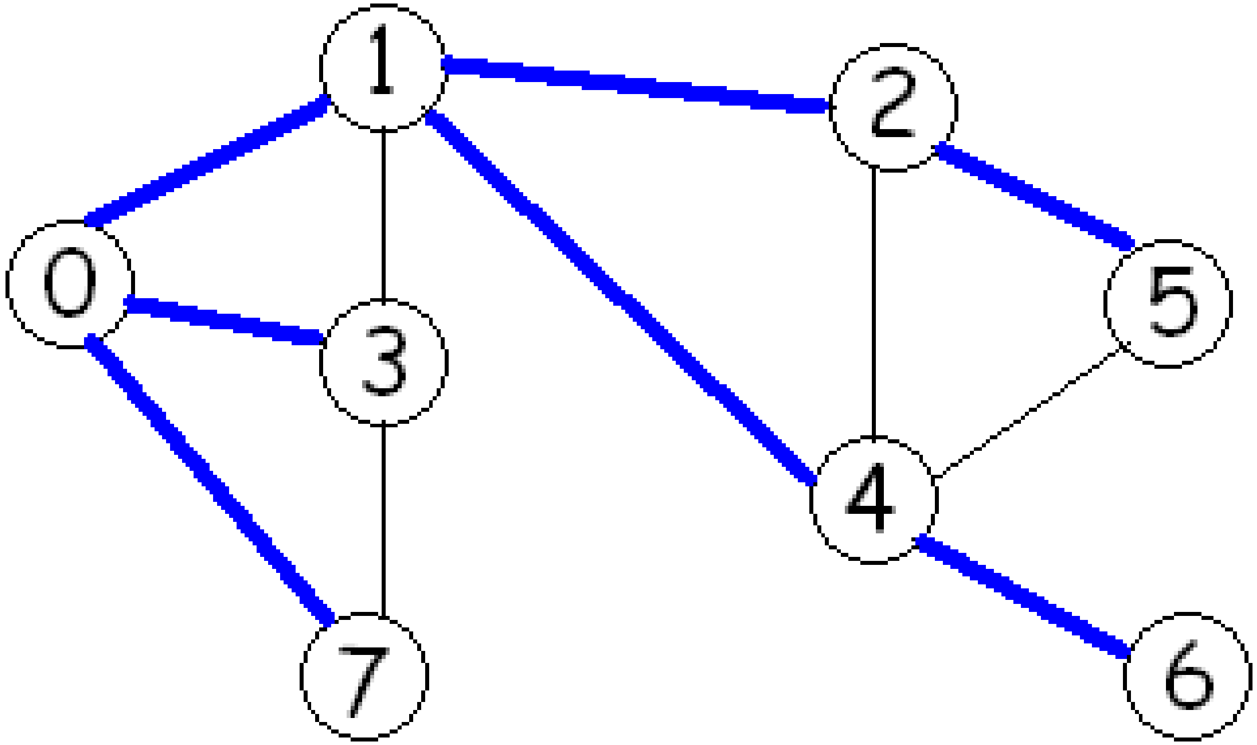

1
2
0
3
5
4
7
6

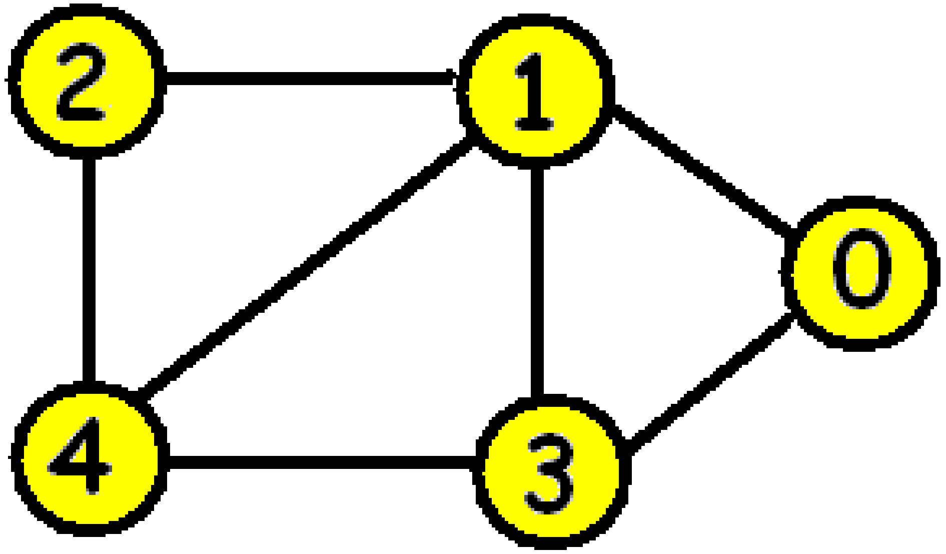
Red arcs represent parent information:





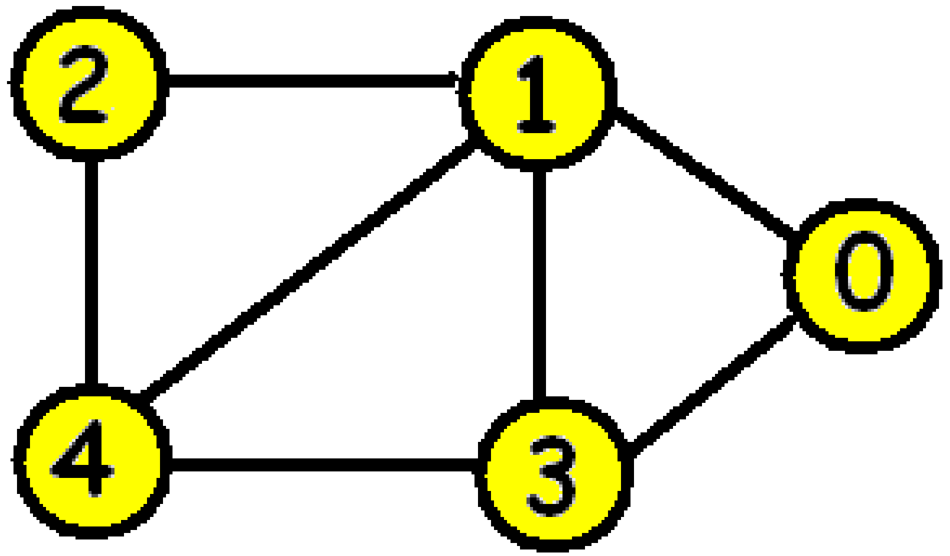
The blue spanning tree is the BFS tree.



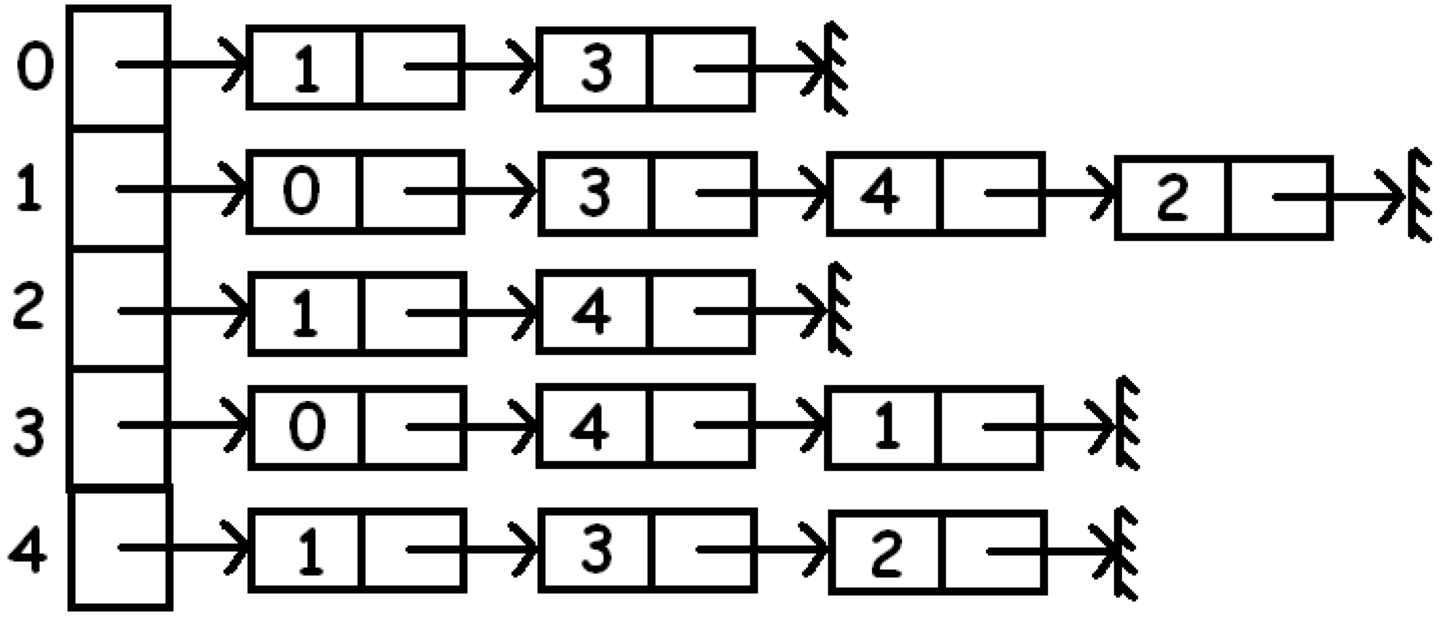


Adjacency matrix:

	0	1	2	3	4
0	0	1	0	1	0
1	1	0	1	1	1
2	0	1	0	0	1
3	1	1	0	0	1
4	0	1	1	1	0



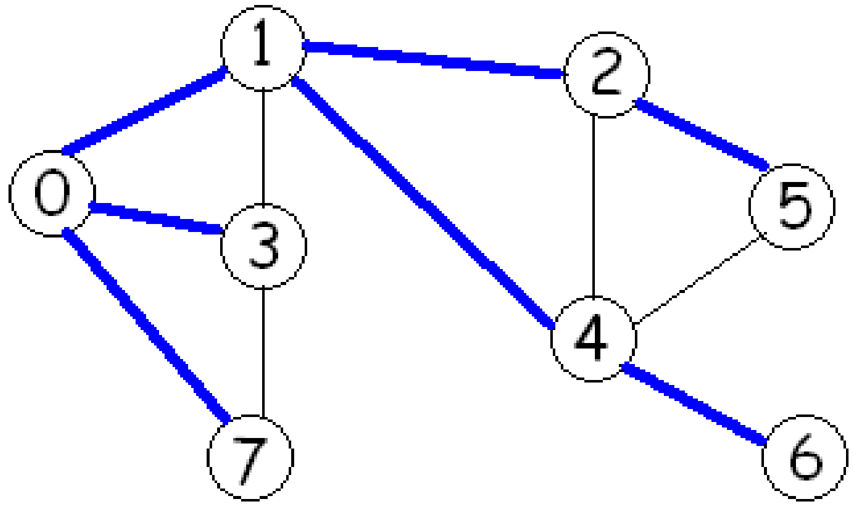
Adjacency list:



BFI[v]= Breadth first index of v

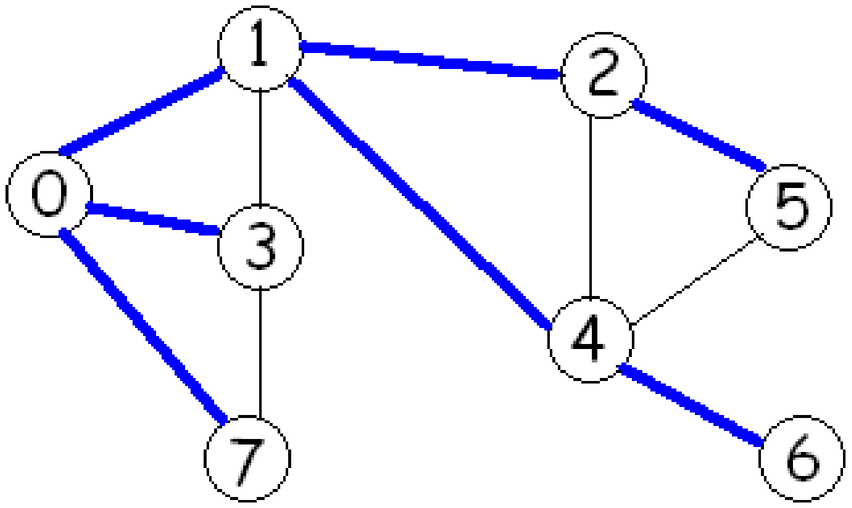
= step at which v is visited.

The BFI[v] is equal to v's position in the queue.



	0	1	2	3	4	5	6	7
Q	0	1	3	7	2	4	5	6

level[v]= distance from v to the root vertex r (number of edges on a shortest path from v to r)



v	0	1	2	3	4	5	6	7
level(v)	0	1	2	1	2	3	3	1

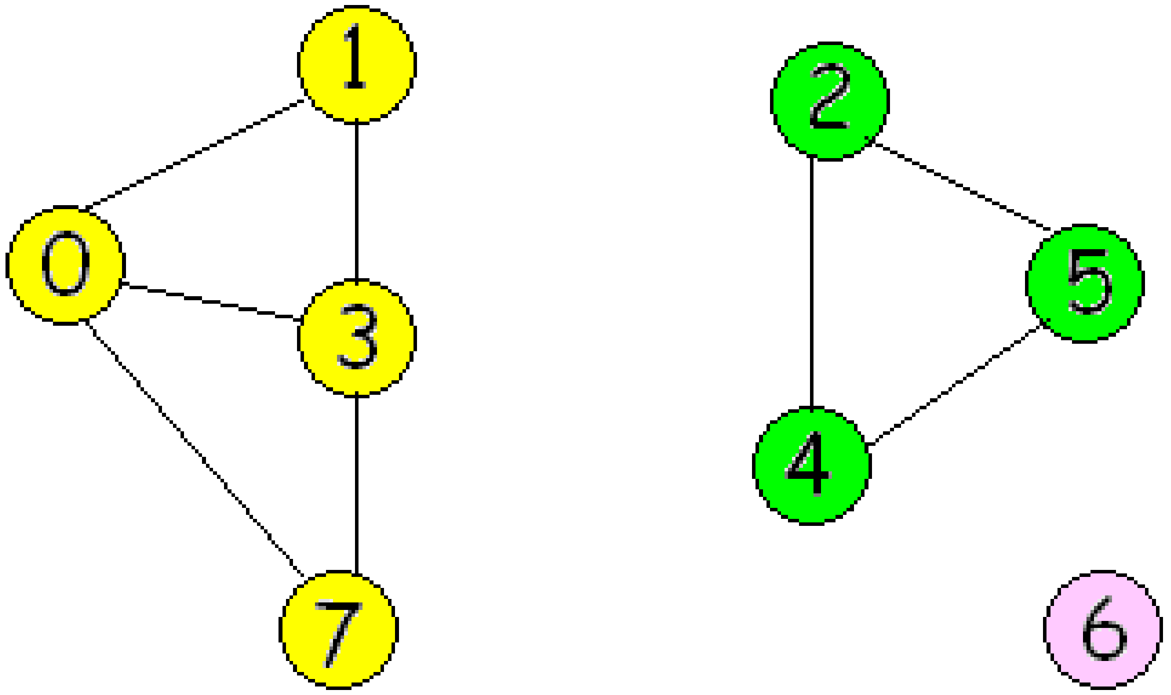
To initialize:

```
// Set parent of each node to be -1 to indicate
// that the vertex has not yet been visited.
for (i=0; i < n; i++) parent[i]= -1;

// Initialize the queue so that BFS starts at s
qfront=0; qrear=1; Q[qfront]= s;
parent[s]=s;

BFI[s]= 0;

Level[s]=0;
```

```
while (qfront < qrear) // Q is not empty
    u= Q[qfront]; qfront++;
    for each neighbour v of u
        if (parent[v] == -1) // not visited
            parent[v]= u; BFI[v]= qrear;
            level[v]= level[u]+1;
            Q[qrear]= v; qrear++;
        end if
    end for
end while
```

One application:

How many connected components does a graph have and which vertices are in each component?

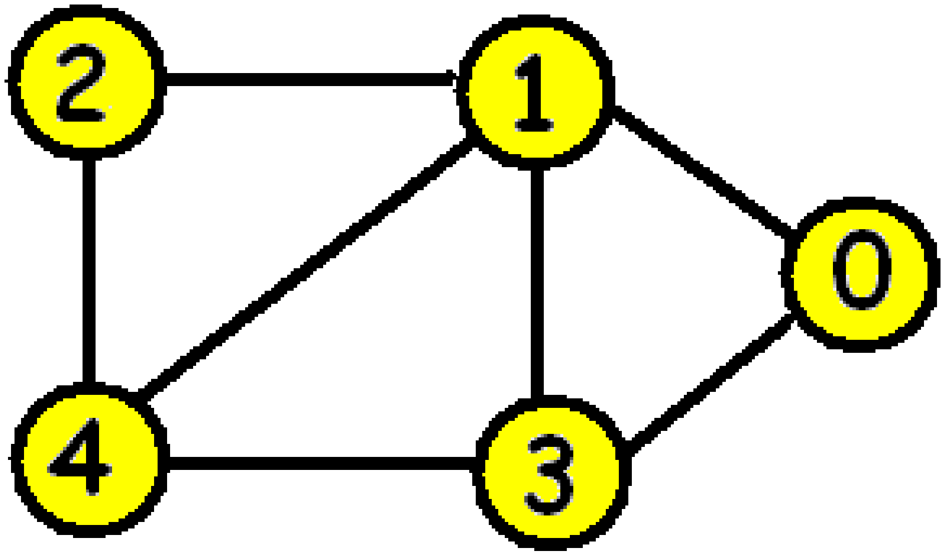


To find the connected components:

```
for (i=0; i < n; i++)
      parent[i]= -1;
nComp= 0;
for (i=0; i < n; i++)
      if (parent[i] == -1)
            nComp++;
            BFS(i, parent, component, nComp);
BFS(s, parent, component, nComp)

// Do not initialize parent.

// Initialize the queue so that BFS starts at s

qfront=0; qrear=1; Q[qfront]= s;

parent[s]=s;

component[s]= nComp;
```

```
while (qfront < qrear) // Q is not empty
    u= Q[qfront]; qfront++;
    for each neighbour v of u
        if (parent[v] == -1) // not visited
            parent[v]= u; component[v]= nComp;
            Q[qrear]= v; qrear++;
        end if
    end for
end while
```

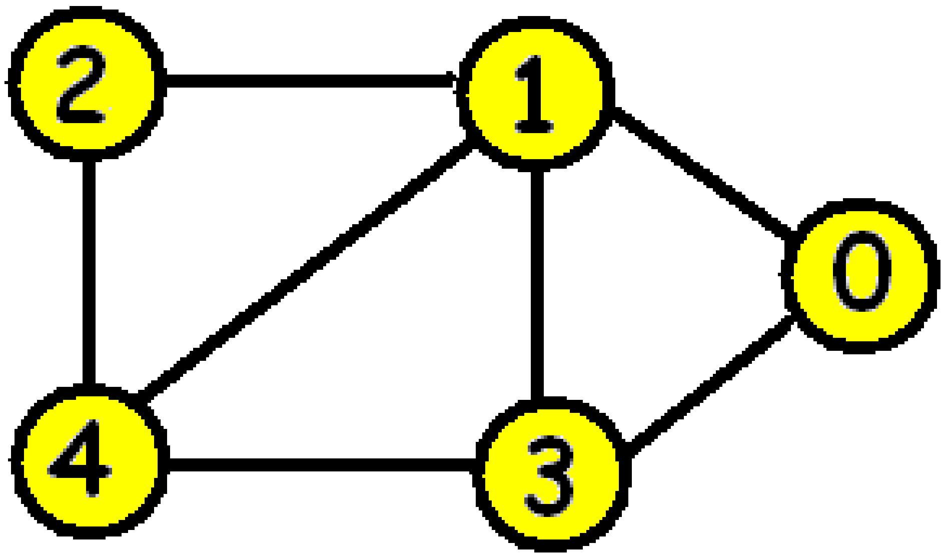
How much time does BFS take to indentify the connected components of a graph when the data structure used for a graph is an adjacency matrix?



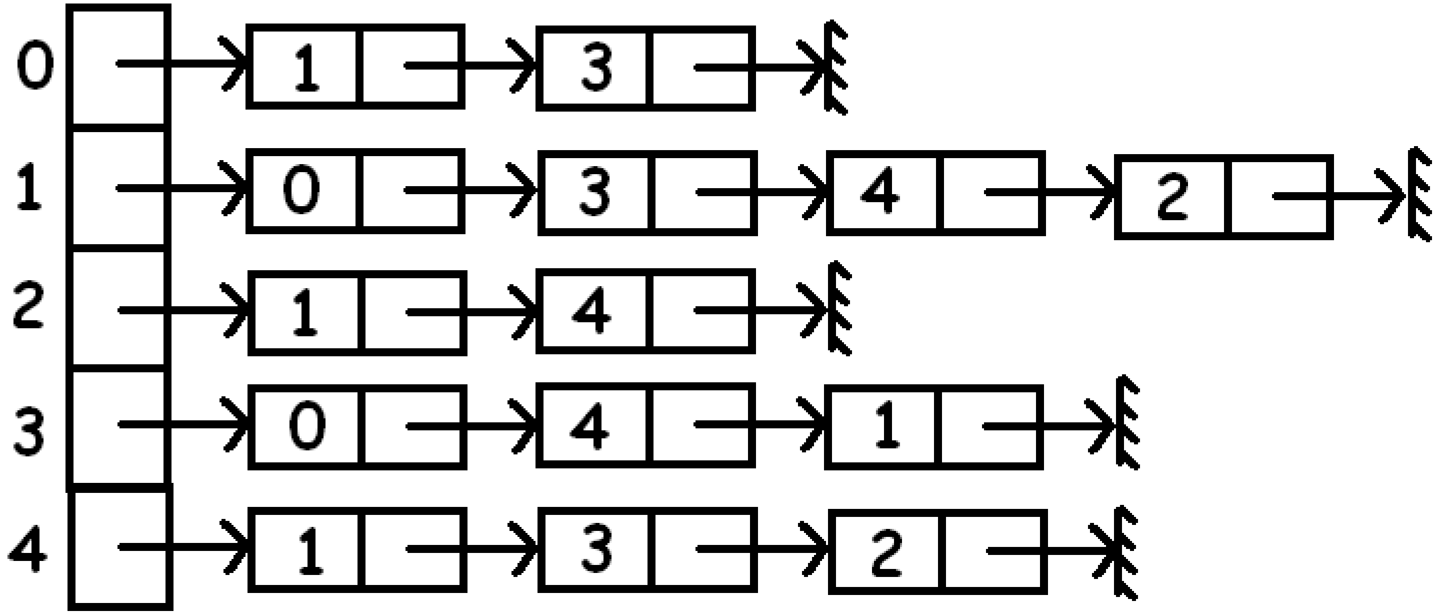
Adjacency matrix:

	0	1	2	3	4
0	0	1	0	1	0
1	1	0	1	1	1
2	0	1	0	0	1
3	1	1	0	0	1
4	0	1	1	1	0

How much time does BFS take to indentify the connected components of a graph when the data structure used for a graph is an adjacency list?



Adjacency list:


0
1
3
1
0
3
4
2
2
1
4
3
0
4
1
4
1
3
2