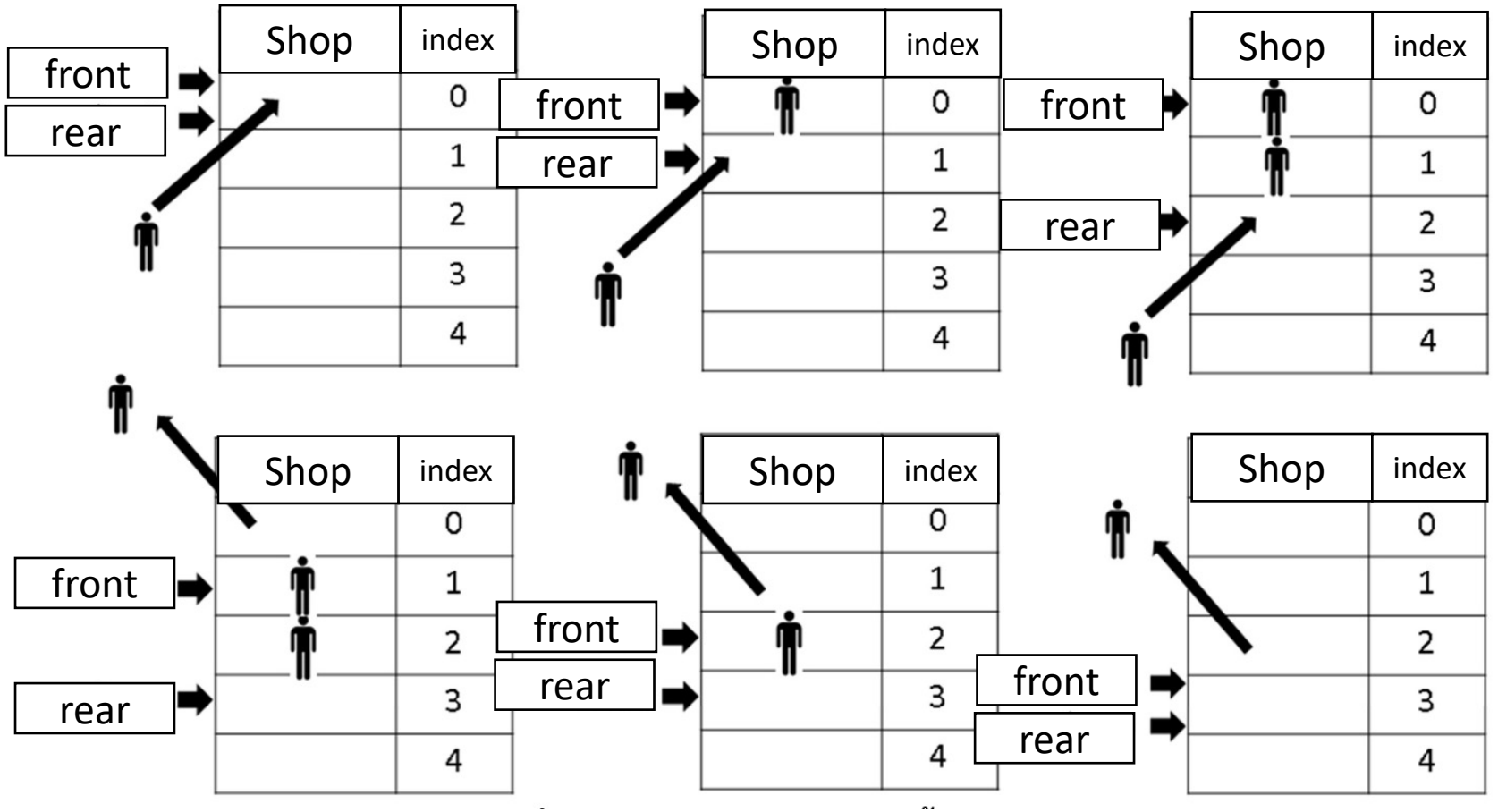
Queue

Outline

- Queue
- Array Implementation
- Applications

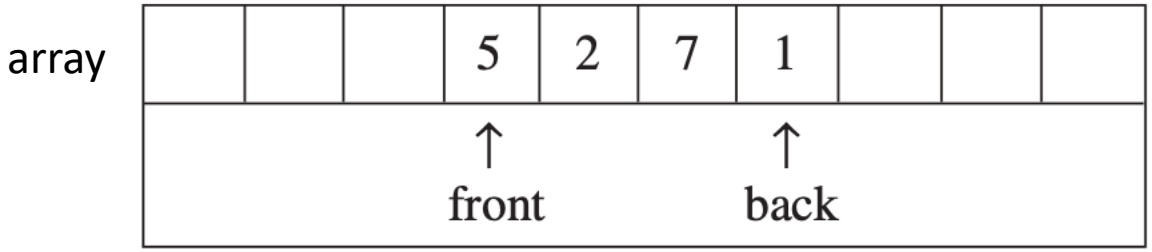
What is Queue?



What is Queue?

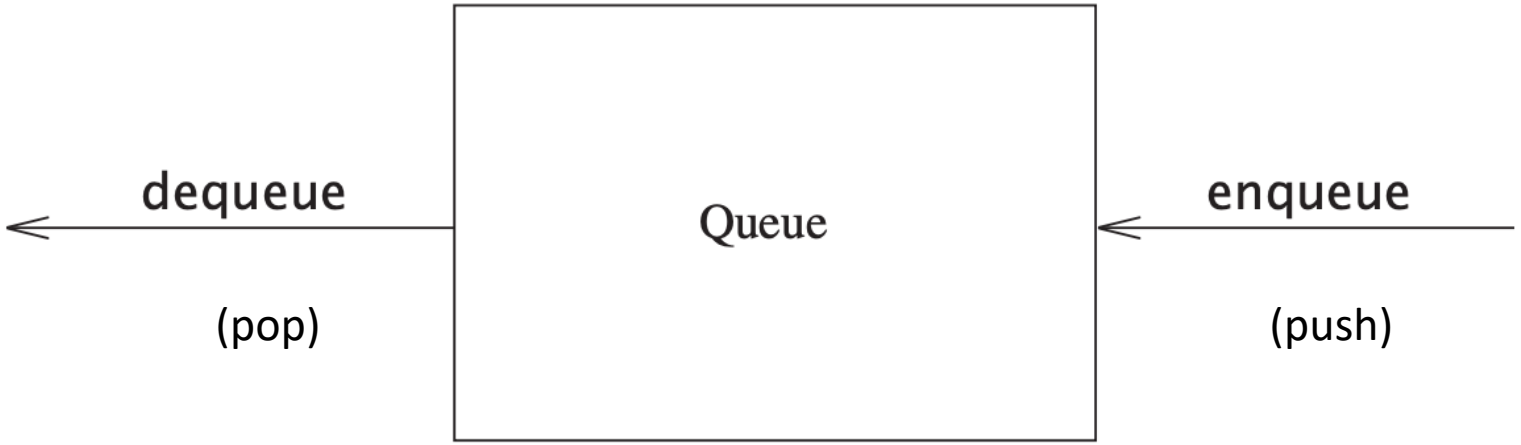
- A **Queue Data Structure** is a fundamental concept in computer science used for storing and managing data in a specific order.
- It follows the principle of "**First in, First out**" **(FIFO)**, where the first element added to the queue is the first one to be removed.

Queue



- Data members in queue class
  - array
  - front/head
  - Back/tail

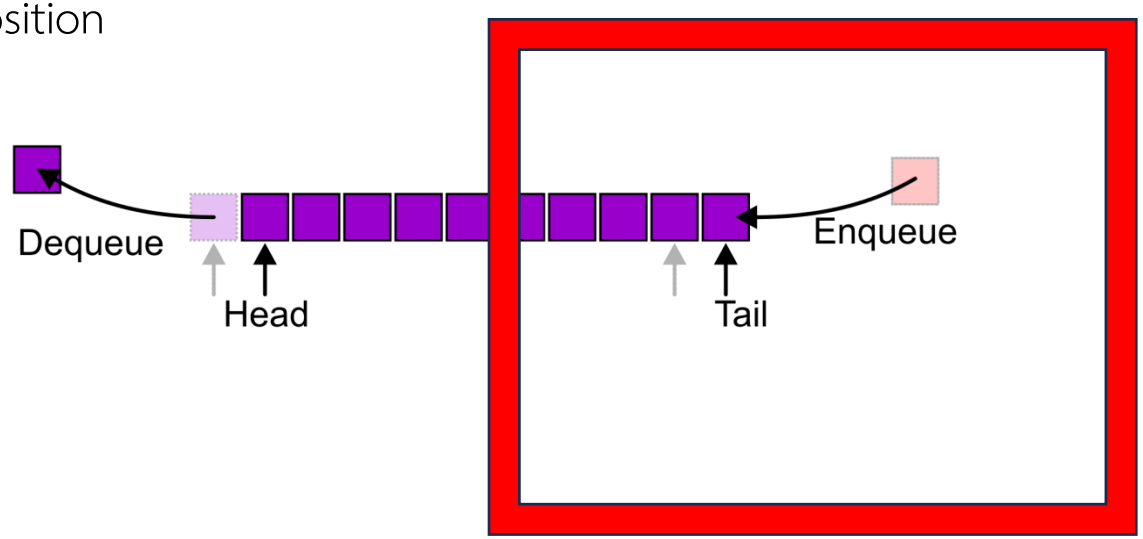
Model of a queue



- Operations of queue
  - enqueue (push)
  - dequeue (pop)

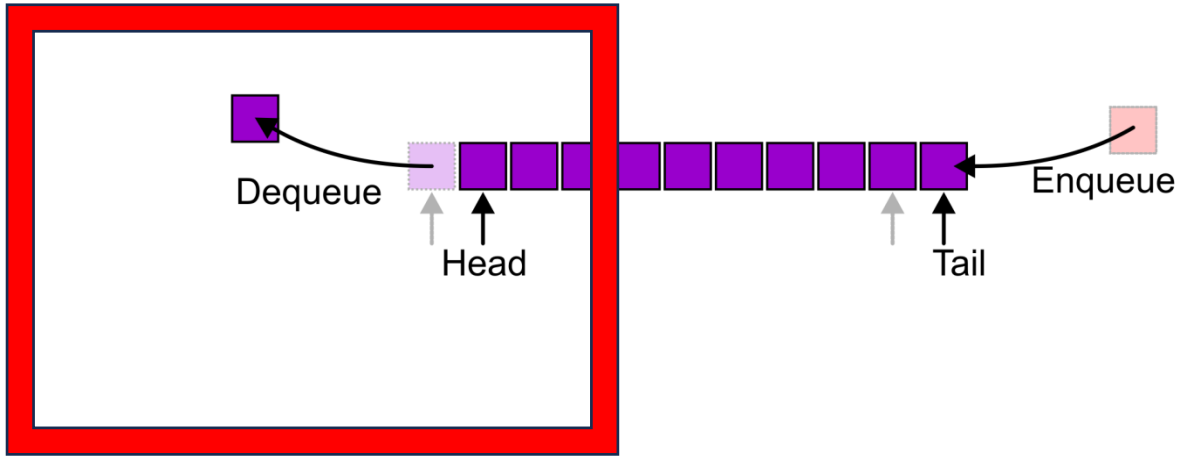
Queue: enqueue

- Enqueue operation: Add data into queue
  - increase tail/rear
  - Add data at tail/rear position

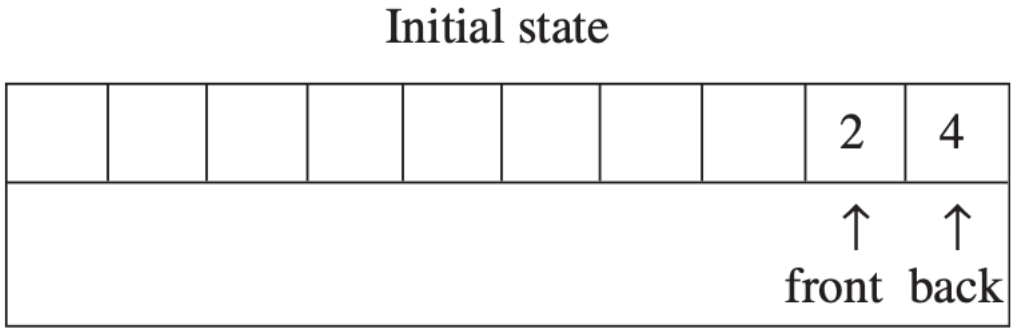


Queue: dequeue

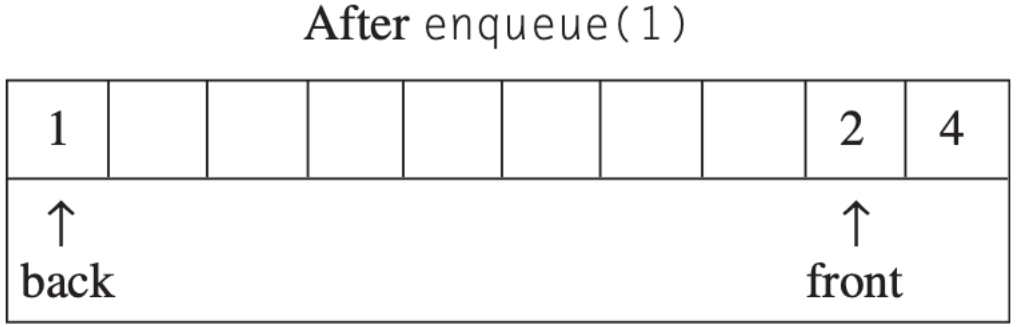
- dequeue operation: remove data from queue
  - Read/return data at head/front position
  - increase head/front



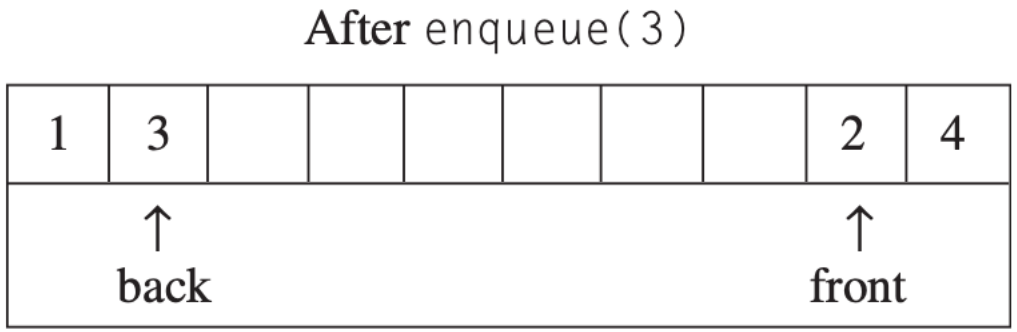
Example of queue



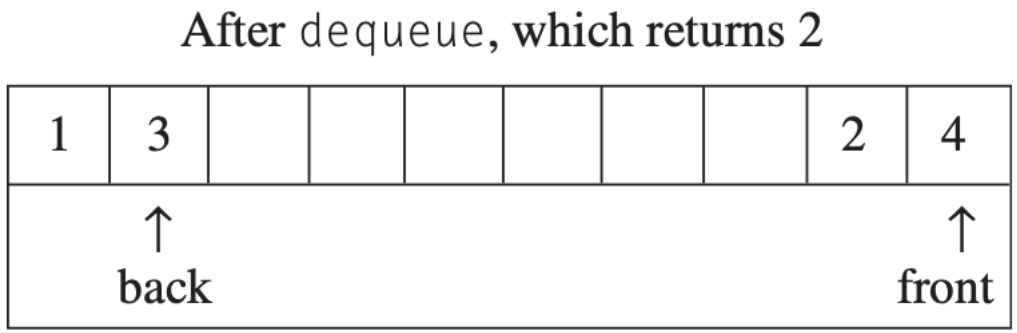
Example of queue



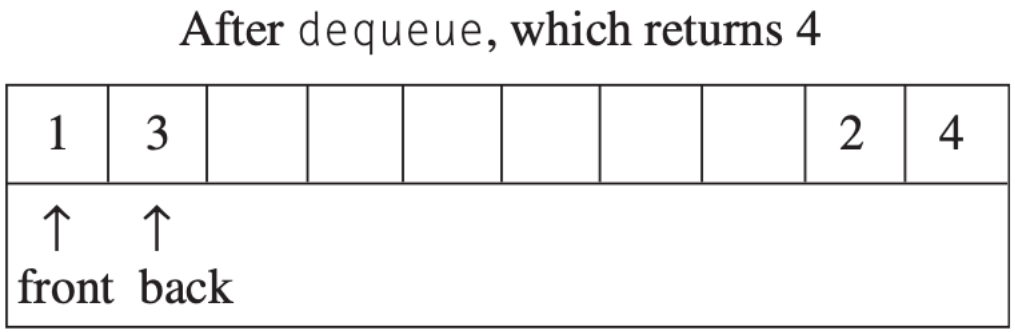
Example of queue



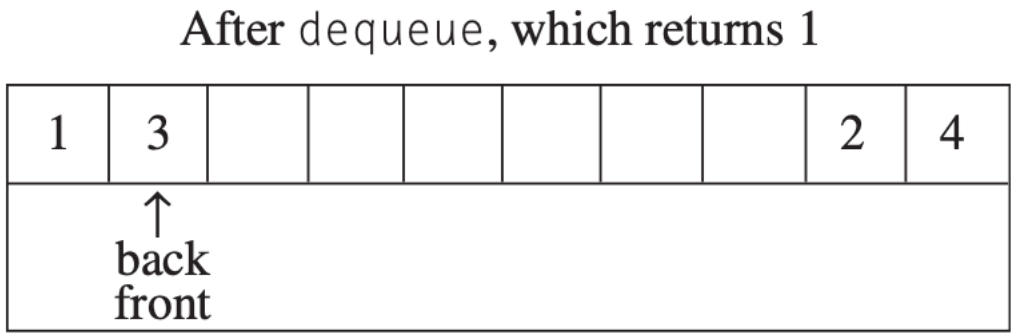
Example of queue



Example of queue

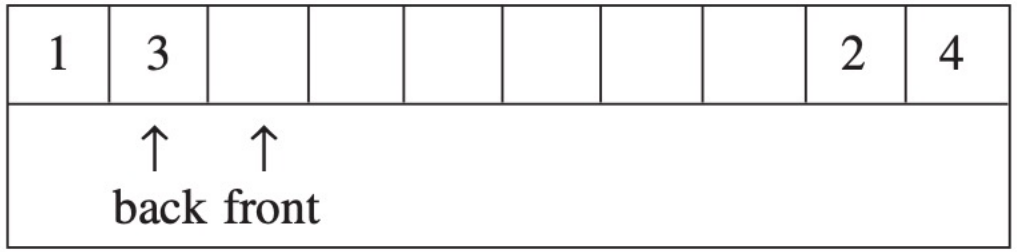


Example of queue



Example of queue

After `dequeue`, which returns 3
and makes the queue empty

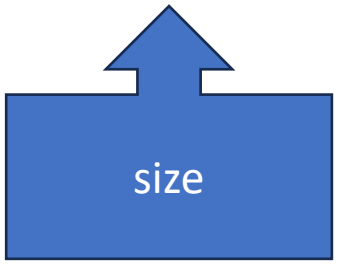


Array Implementation

We need to store an array:

- In C++, this is done by storing the address of the first entry

int array[6];



Queue-as-Array Class

The class definition is similar to that of the Stack:

```
class Queue{
  public:
      int Array[6];
      int size = 4;
      int front = 1;
      int rear = 0;

      bool empty();
      bool full();
      void enqueue(int data);
      int dequeue();
};
```

empty()

Queue is empty when front is greater than rear

```
bool empty(){
   if(front > rear)
            return 1;
   else
            return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
class Queue
{
   public:
   int Array[6];
   int size = 4;
   int front = 1;
   int rear = 0;
   bool empty()
   {
      if( front > rear )
      {
         return 1;
      }
      else
      {
         return 0;
      }
   }
   bool full()
   {
      if( rear == size )
      {
         return 1;
      }
      else
      {
         return 0;
      }
   }
   void enqueue(int data)
   {
      if( !full() )
      {
         rear++;
         Array[rear] = data;
      }
   }
   int dequeue()
   {
      if( !empty() )
      {
         int temp = Array[front];
         front++;
         return temp;
      }
      return -1;
   }
   void print()
   {
      cout<<"Queue ";
      for (int i = front ; i <= rear; i++)
      {
         cout << Array[i] << " ";
      }
      cout<<endl;
   }
};
```

full()

Queue is full when rear is equal to size.

//This code store the first data at index 1.

```
bool full(){
   if(rear == size)
            return 1;
   else
            return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
class Queue
{
   public:
   int Array[6];
   int size = 4;
   int front = 1;
   int rear = 0;
   bool empty()
   {
      if( front > rear )
      {
         return 1;
      }
      else
      {
         return 0;
      }
   }
   bool full()
   {
      if( rear == size )
      {
         return 1;
      }
      else
      {
         return 0;
      }
   }
   void enqueue(int data)
   {
      if( !full() )
      {
         rear++;
         Array[rear] = data;
      }
   }
   int dequeue()
   {
      if( !empty() )
      {
         int temp = Array[front];
         front++;
         return temp;
      }
      return -1;
   }
   void print()
   {
      cout<<"Queue ";
      for (int i = front ; i <= rear; i++)
      {
         cout << Array[i] << " ";
      }
      cout<<endl;
   }
};
```

enqueue()

Add object at rear of the queue.

```
void enqueue(int data){
        if( !full() ) {
                rear++;
                Array[rear] = data;
        }
}
```

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  class Queue
4  {
5     public:
6     int Array[6];
7     int size = 4;
8     int front = 1;
9     int rear = 0;
10    bool empty()
11    {
12       if( front > rear )
13       {
14          return 1;
15       }
16       else
17       {
18          return 0;
19       }
20    }
21    bool full()
22    {
23       if( rear == size )
24       {
25          return 1;
26       }
27       else
28       {
29          return 0;
30       }
31    }
32    void enqueue(int data)
33    {
34       if( !full() )
35       {
36          rear++;
37          Array[rear] = data;
38       }
39    }
40    int dequeue()
41    {
42       if( !empty() )
43       {
44          int temp = Array[front];
45          front++;
46          return temp;
47       }
48       return -1;
49    }
50    void print()
51    {
52       cout<<"Queue ";
53       for (int i = front ; i <= rear; i++)
54       {
55          cout << Array[i] << " ";
56       }
57       cout<<endl;
58    }
59 };
```

dequeue()

remove object at front of the queue.

```
int dequeue(){
        if( !empty() ) {
                int temp = Array[front];
                front++;
                return temp;
        }
        return -1;
}
```

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  class Queue
4  {
5     public:
6     int Array[6];
7     int size = 4;
8     int front = 1;
9     int rear = 0;
10    bool empty()
11    {
12       if( front > rear )
13       {
14          return 1;
15       }
16       else
17       {
18          return 0;
19       }
20    }
21    bool full()
22    {
23       if( rear == size )
24       {
25          return 1;
26       }
27       else
28       {
29          return 0;
30       }
31    }
32    void enqueue(int data)
33    {
34       if( !full() )
35       {
36          rear++;
37          Array[rear] = data;
38       }
39    }
40    int dequeue()
41    {
42       if( !empty() )
43       {
44          int temp = Array[front];
45          front++;
46          return temp;
47       }
48       return -1;
49    }
50    void print()
51    {
52       cout<<"Queue ";
53       for (int i = front ; i <= rear; i++)
54       {
55          cout << Array[i] << " ";
56       }
57       cout<<endl;
58    }
59 };
```

Example In C++

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  class Queue
4  {
5      public:
6      int Array[6];
7      int size = 4;
8      int front = 1;
9      int rear = 0;
10     bool empty()
11     {
12         if( front > rear )
13         {
14             return 1;
15         }
16         else
17         {
18             return 0;
19         }
20     }
21     bool full()
22     {
23         if( rear == size )
24         {
25             return 1;
26         }
27         else
28         {
29             return 0;
30         }
31     }
32     void enqueue(int data)
33     {
34         if( !full() )
35         {
36             rear++;
37             Array[rear] = data;
38         }
39     }
40     int dequeue()
41     {
42         if( !empty() )
43         {
44             int temp = Array[front];
45             front++;
46             return temp;
47         }
48         return -1;
49     }
50     void print()
51     {
52         cout<<"Queue ";
53         for (int i = front ; i <= rear; i++)
54         {
55             cout << Array[i] << " ";
56         }
57         cout<<endl;
58     }
59 };
```

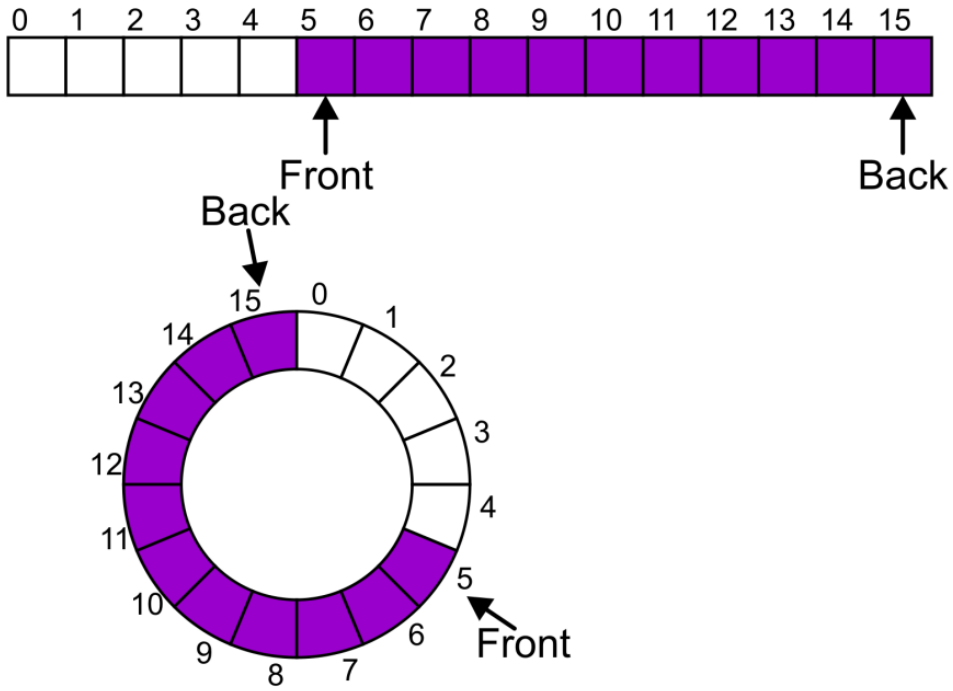
```
60 int main()
61 {
62     Queue q;
63     q.enqueue(1);       cout<<"Enqueue 1\t";
64     q.print();
65     q.enqueue(2);       cout<<"Enqueue 2\t";
66     q.print();
67     q.enqueue(3);       cout<<"Enqueue 3\t";
68     q.print();
69     q.enqueue(4);       cout<<"Enqueue 4\t";
70     q.print();
71     q.enqueue(5);       cout<<"Enqueue 5\t";
72     q.print();
73     cout<<"Dequeue\t";  cout<<q.dequeue()<<"\t";
74     q.print();
75     cout<<"Dequeue\t";  cout<<q.dequeue()<<"\t";
76     q.print();
77     cout<<"Dequeue\t";  cout<<q.dequeue()<<"\t";
78     q.print();
79     cout<<"Dequeue\t";  cout<<q.dequeue()<<"\t";
80     q.print();
81     cout<<"Dequeue\t";  cout<<q.dequeue()<<"\t";
82     q.print();
83     return 0;
84 }
```

Member Functions

Instead of viewing the array on the range 0, …, 15, consider the indices being cyclic:

…, 15, 0, 1, …, 15, 0, 1, …, 15, 0, 1, …

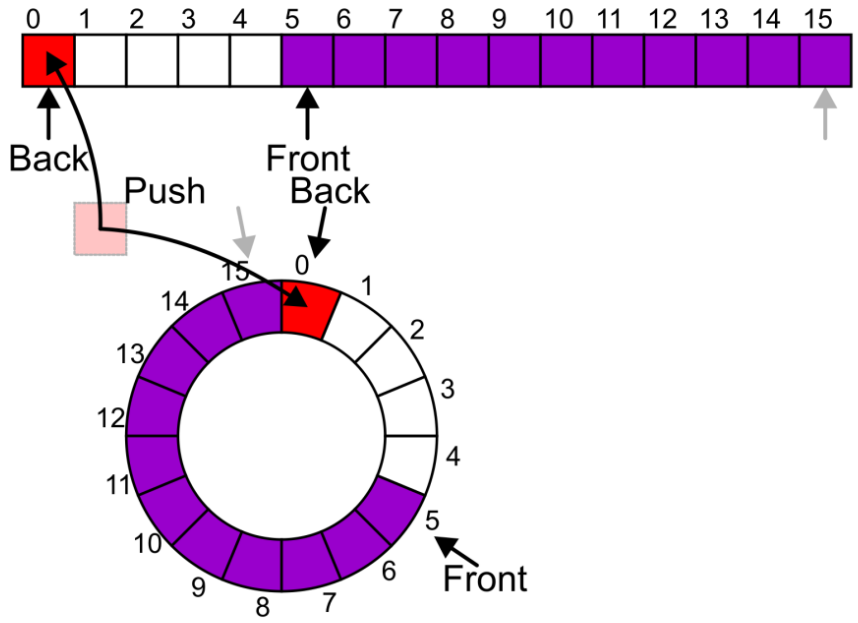
This is referred to as a *circular array*



Member Functions

Now, the next push may be performed in the next available location of the circular array:

```
++iback;
if ( iback == capacity() ) {
    iback = 0;
}
```



Exampl

In C++

```
#include <bits/stdc++.h>
using namespace std;
class Queue
{
    public:
    int array[5];
    int size = 4;
    int front = 0;
    int rear = 0;
    bool empty()
    {
        if( front == 0 && rear == 0 )
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
    bool full()
    {
        if( (front == 1) && (rear == size) ||
        (front == rear + 1) && (rear < size) )
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
    void enqueue(int data)
    {
        if( !full() )
        {
            if( empty() )
            {
                front = 1;
                rear = 1;
            }
            else
            {
                if( rear == size )
                {
                    rear = 1;
                }
                else
                {
                    rear++;
                }
            }
            array[rear] = data;
        }
    }
    int dequeue()
    {
        if( !empty() )
        {
            int temp = a[front];
            if( front == rear)
            {
                rear = 0;
                front = 0;
            }
            else if(front == size )
            {
                front = 1;
            }
            else
            {
                front++;
            }
            return temp;
        }
        return -1;
    }
    void print()
    {
        cout<<"Queue ";

        if(front > 0 && rear > 0)
        {
        if(front <= rear)
        {
            for (int i = front ; i <= rear; i++)
            {
                cout << array[i] << " ";
            }
        }
        else
        {
            for (int i = front ; i <= size; i++)
            {
                cout << array[i] << " ";
            }
            for (int i = 1 ; i <= rear; i++)
            {
                cout << array[i] << " ";
            }
        }
        }
        cout<<endl;
    }
};
```

Example

In C++

```
106  int main()
107  {
108      Queue q;
109      q.enqueue(1);          cout<<"Enqueue 1\t";
110      q.print();
111      q.enqueue(2);          cout<<"Enqueue 2\t";
112      q.print();
113      q.enqueue(3);          cout<<"Enqueue 3\t";
114      q.print();
115      q.enqueue(4);          cout<<"Enqueue 4\t";
116      q.print();
117      q.enqueue(5);          cout<<"Enqueue 5\t";
118      q.print();
119      cout<<"Dequeue\t";     cout<<q.dequeue()<<"\t";
120.     q.print();
121      q.enqueue(6);          cout<<"Enqueue 6\t";
122      q.print();
123      cout<<"Dequeue\t";     cout<<q.dequeue()<<"\t";
124      q.print();
125      cout<<"Dequeue\t";     cout<<q.dequeue()<<"\t";
126      q.print();
127      cout<<"Dequeue\t";     cout<<q.dequeue()<<"\t";
128      q.print();
129      cout<<"Dequeue\t";     cout<<q.dequeue()<<"\t";
130      q.print();
131      q.enqueue(7);          cout<<"Enqueue 7\t";
132      q.print();
133      return 0;
134  }
```

Applications

The most common application is in client-server models

- Multiple clients may be requesting services from one or more servers
- Some clients may have to wait while the servers are busy
- Those clients are placed in a queue and serviced in the order of arrival

Grocery stores, banks, and airport security use queues

The SSH Secure Shell and SFTP are clients

Most shared computer services are servers:

- Web, file, ftp, database, mail, printers, WOW, *etc*.

Applications

Uniprocessor Schedulings

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Applications

Uniprocessor Schedulings

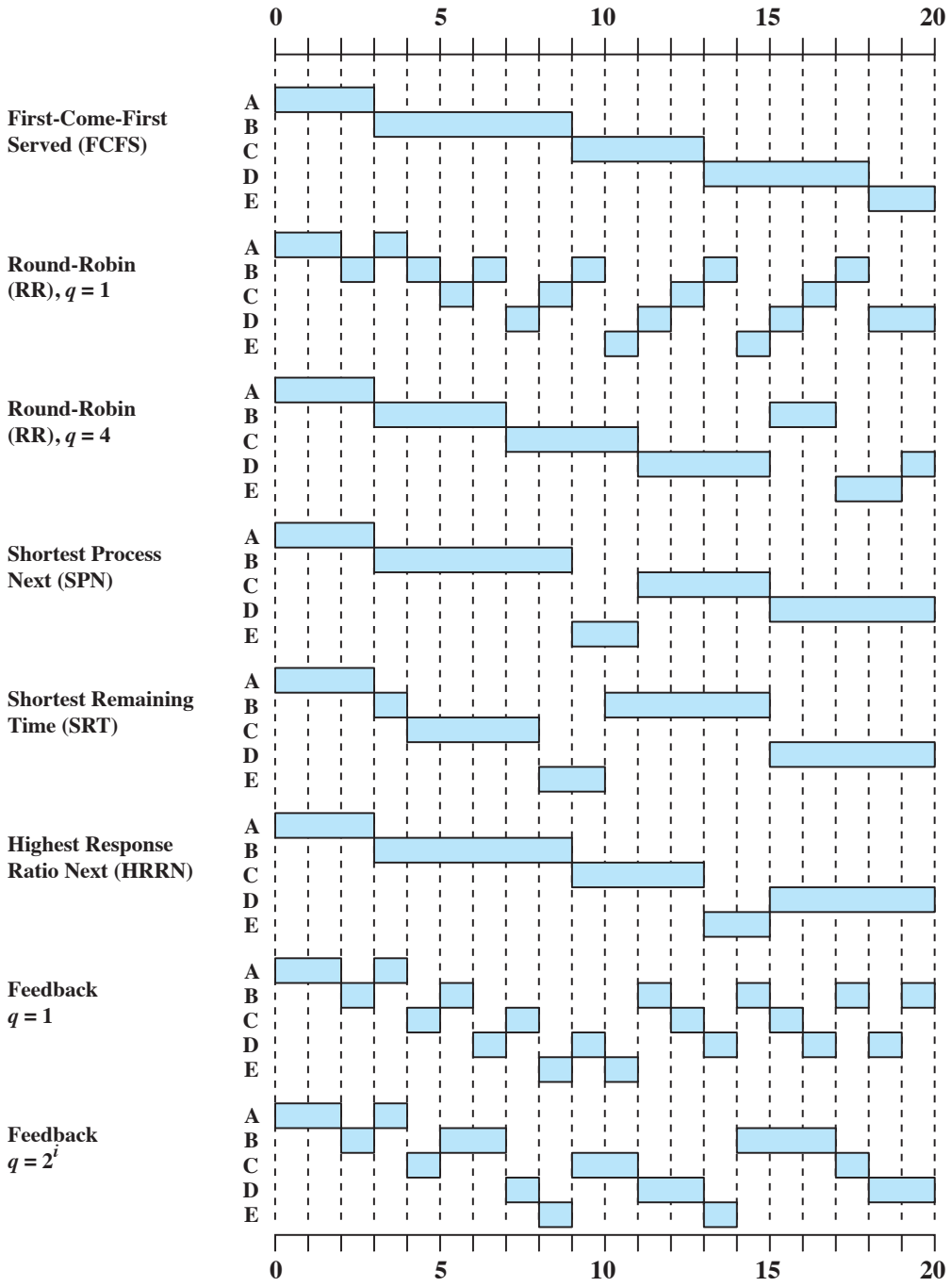
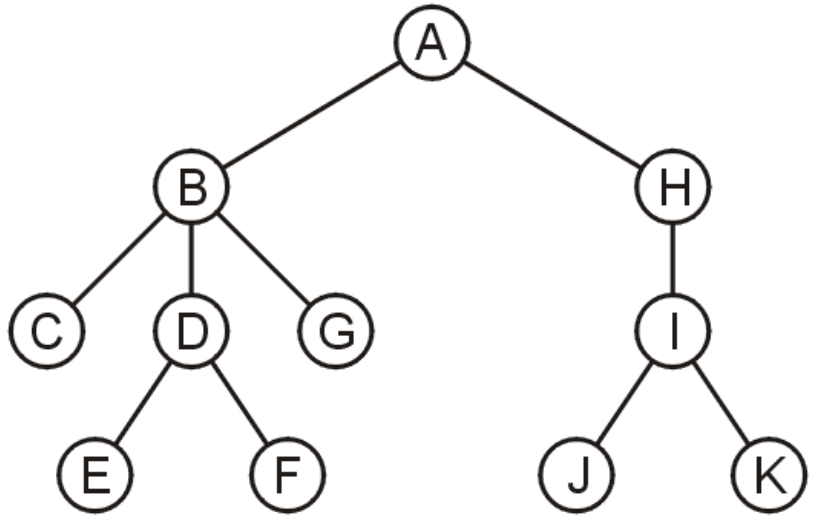


Figure 9.5 A Comparison of Scheduling Policies

Applications

Another application is performing a breadth-first traversal of a directory tree

- Consider searching the directory structure

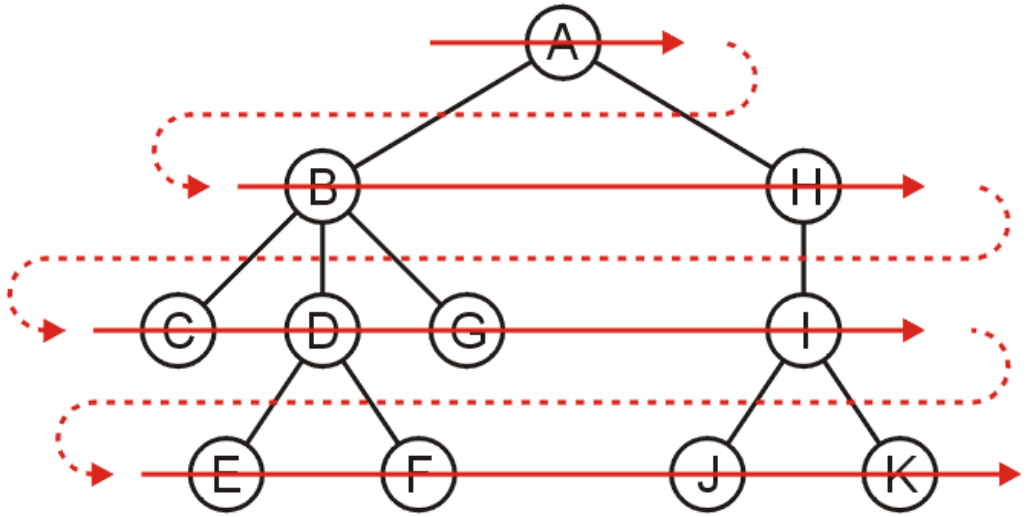


Applications

We would rather search the more shallow directories first then plunge deep into searching one sub-directory and all of its contents

One such search is called a *breadth-first traversal*

- Search all the directories at one level before descending a level



Applications

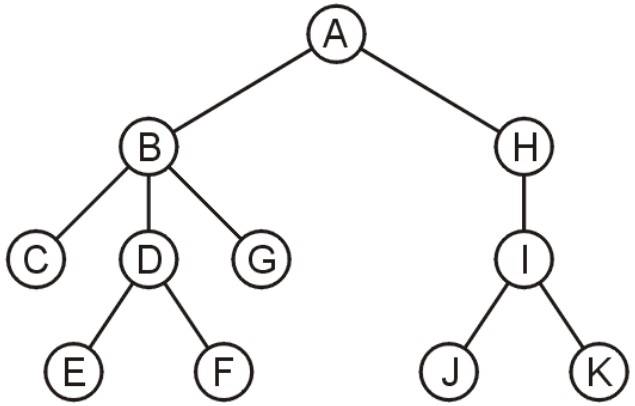
The easiest implementation is:

- Place the root directory into a queue
- While the queue is not empty:
  - Pop the directory at the front of the queue
  - Push all of its sub-directories into the queue

The order in which the directories come out of the queue will be in breadth-first order

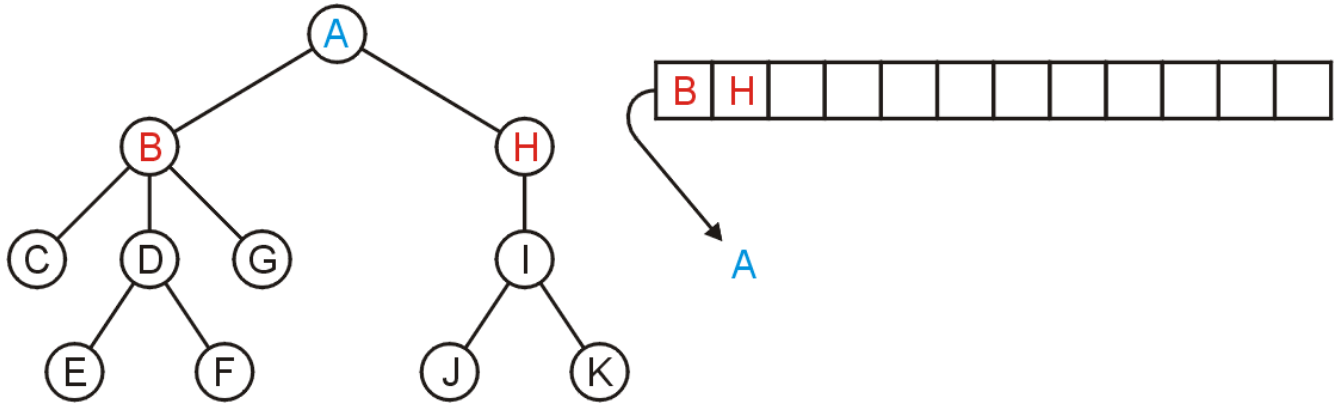
Applications

Push the root directory A



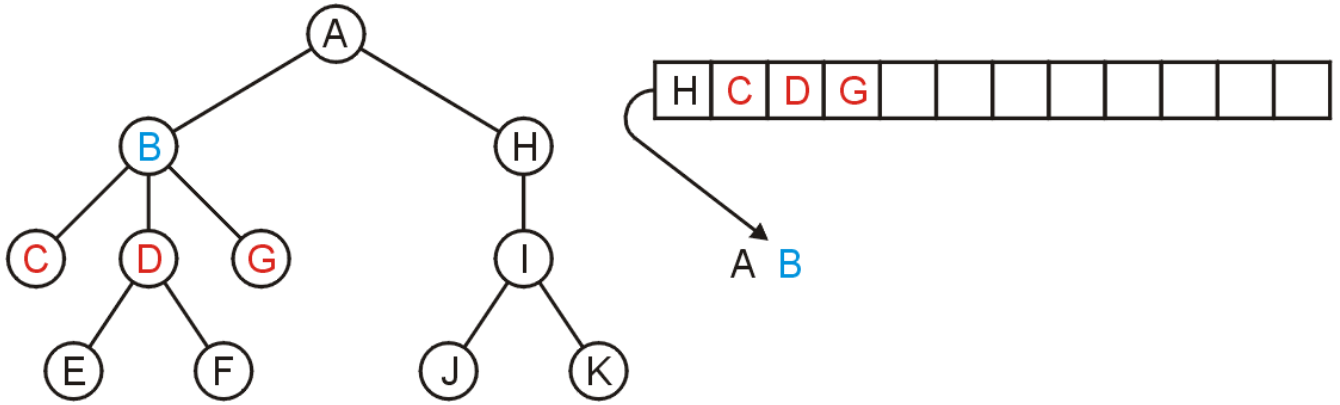
Applications

Pop A and push its two sub-directories: B and H



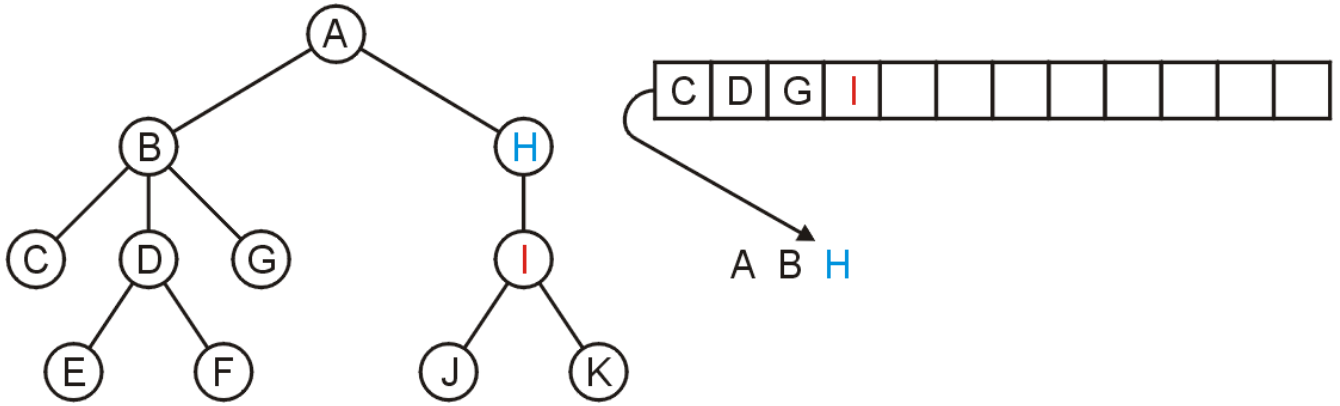
Applications

Pop B and push C, D, and G



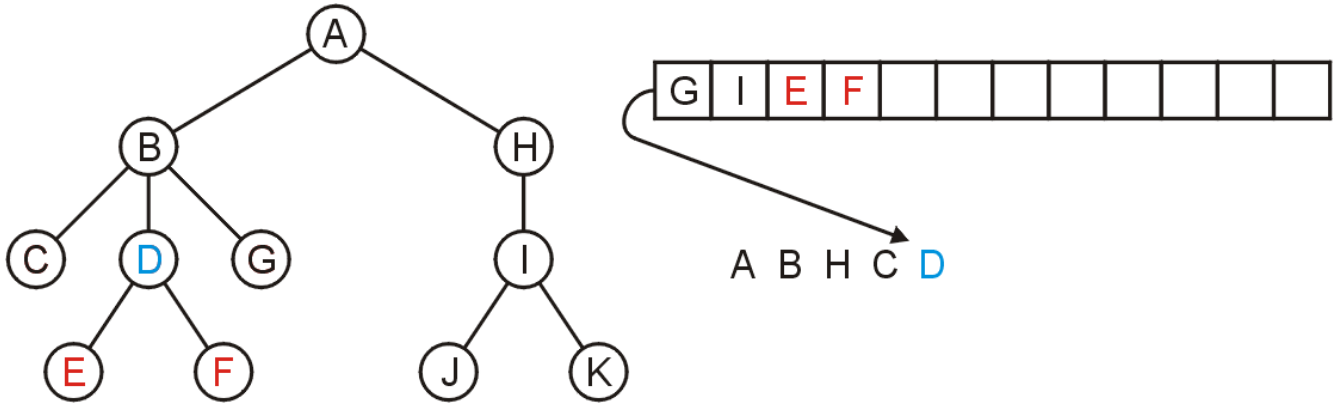
Applications

Pop H and push its one sub-directory I



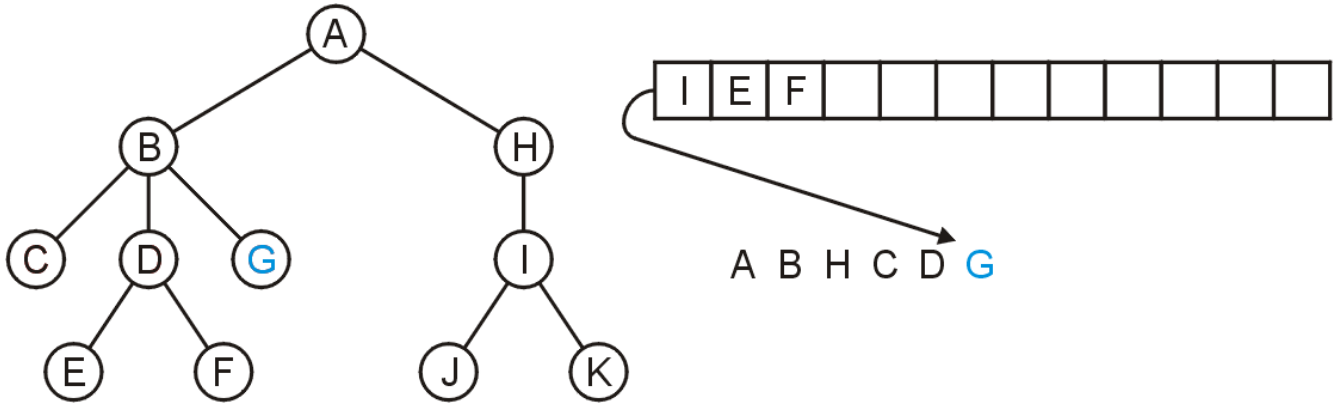
Applications

Pop D and push E and F



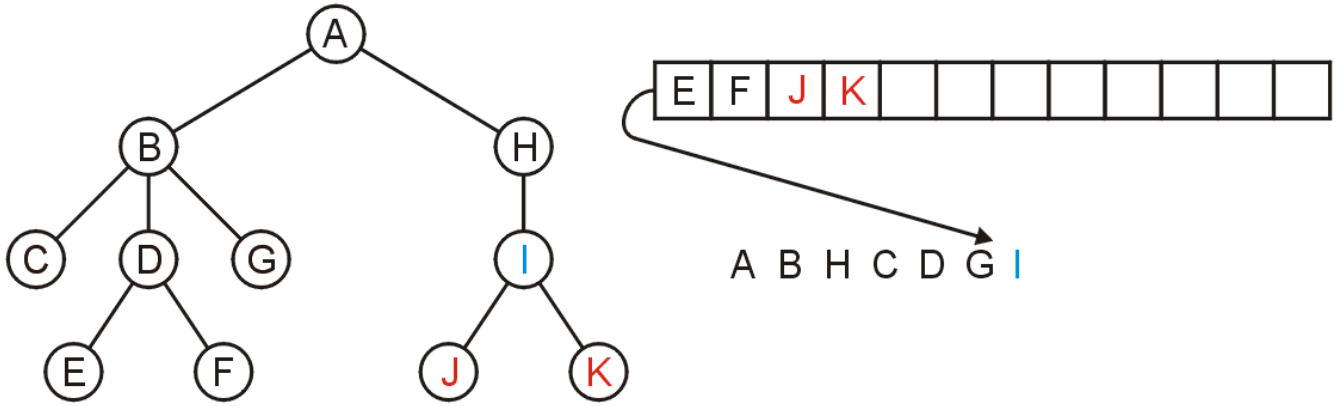
Applications

Pop G



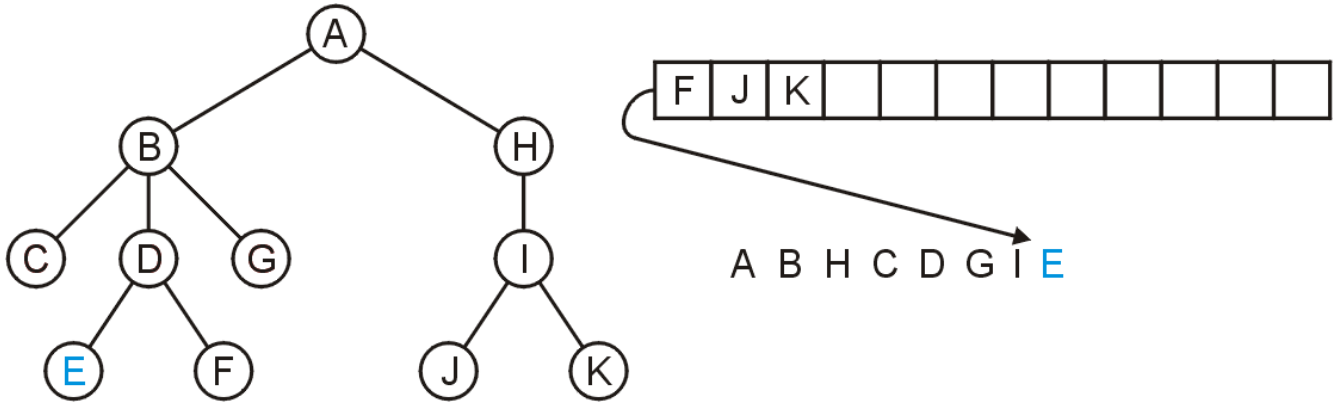
Applications

Pop I and push J and K



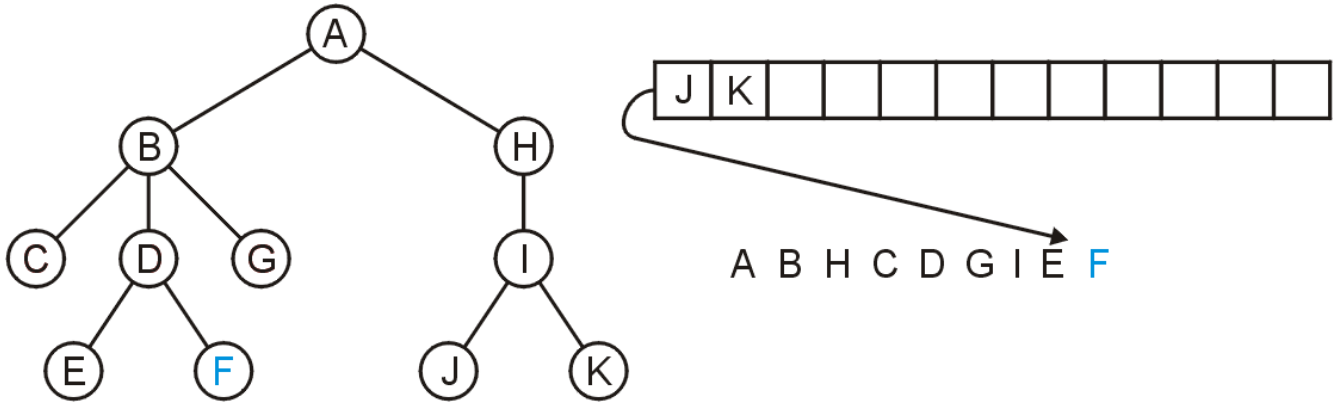
Applications

Pop E



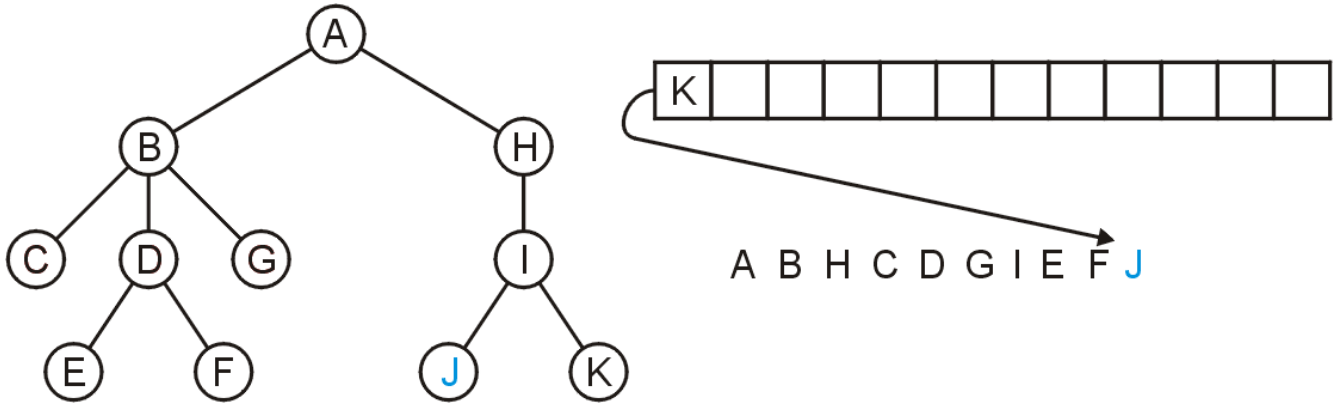
Applications

Pop F



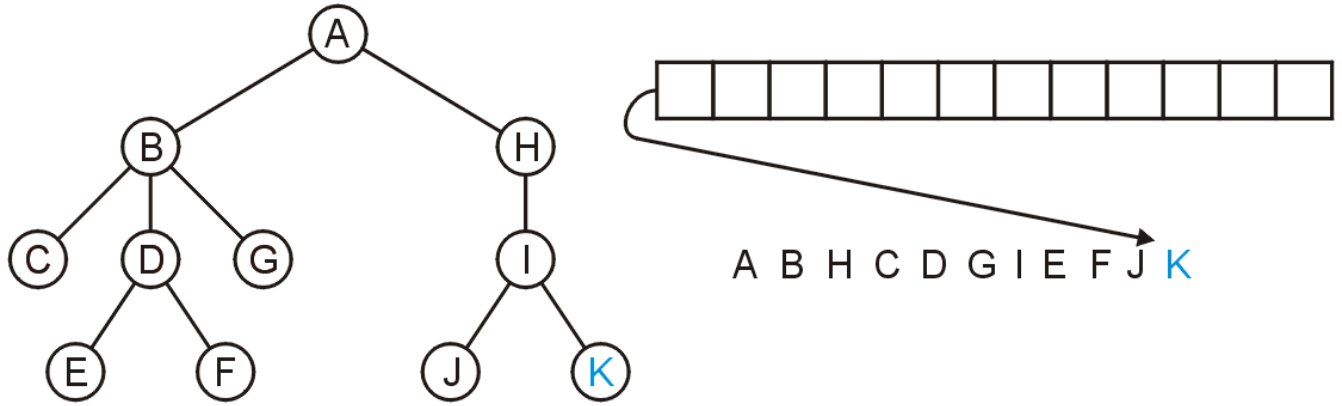
Applications

Pop J



Applications

Pop K and the queue is empty

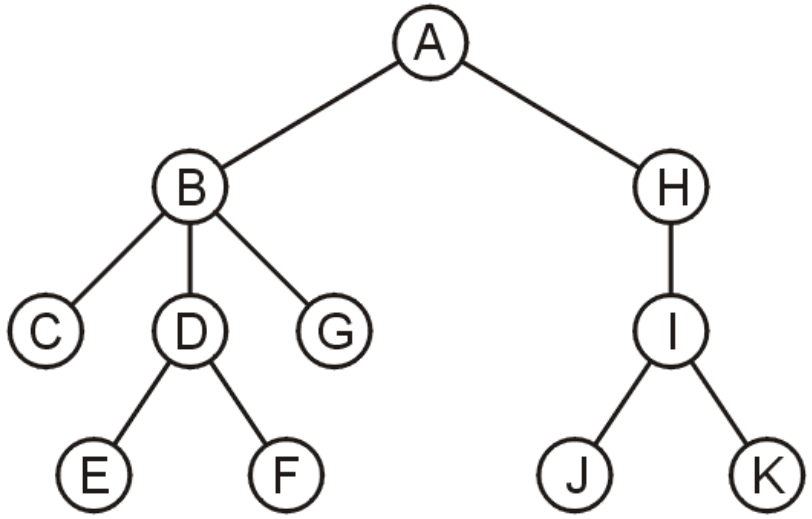


Applications

The resulting order

A B H C D G I E F J K

is in breadth-first order:



Standard Template Library

```
#include <iostream>
#include <queue>
using namespace std;
int main() {
    queue <int> iqueue;
    iqueue.push( 13 );
    iqueue.push( 42 );
    cout << "Head: " << iqueue.front() << endl;
    iqueue.pop();                    // no return value
    cout << "Head: " << iqueue.front() << endl;
    cout << "Size: " << iqueue.size() << endl;
    return 0;
}
```

Reference

Allen, W. M. (2007). *Data structures and algorithm analysis in C++*. Pearson Education India.

Nell B. Dale. (2003). *C++ plus data structures*. Jones & Bartlett Learning.

Stallings, W., & Paul, G. K. (2012). Operating systems: internals and design principles (Vol. 9). New York: Pearson.

เฉียบวุฒิ รัตนวิลัยสกุล. (2023). โครงสร้างข้อมูล. มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

https://ece.uwaterloo.ca/~dwharder/aads/Lecture_materials/