Travel Network
Project 7

CS 211 – Fall 2018

Project 7

Project 6 with the addition of 1 command

- s <int1> <int2>

- Display the list of Airports for the Shortest Path from Airport <int1> to Airport <int2>
- Call function breadthFirstSearch which returns a list of Airports containing the Short Path

Shortest Path Algorithm

- Needs a Queue (for Breadth First Search) and a Stack (to build Path)

- You can create two new classes or just add the following operations to the MyList class from Project 6
  - addToFrontOfList ( )
  - addToBackOfList ( )
  - getValueAtFrontOfList ( )
  - removeValueFromFrontOfList ( )
  - isListEmpty ( )
- These methods may already exist in your MyList code

Shortest Path Algorithm

- Needs each Airport to have a “previous location”
- In Depth First Search, we just needed to know if an Airport has been visited or not
- In Breadth First Search (and when building the Path), we need to know which Airport the flight came from when we visit an Airport
- Add a “previousLocation” member to the Airport class

```
list breadthFirstSearch (int x, int y)
{
  mark all airports as unvisited (set all previousLocation to -1);
  set the list AirportQueue to be empty
  add x to the end of the AirportQueue
  if ( bfs ( y, AirportQueue ) = = FALSE )
    print (“You can NOT get from airport “ + x + “ to airport “ + y + “ in one or more flights”);
    return an empty list
  else
    print (“You can get from airport “ + x + “ to airport “ + y + “ in one or more flights”);
    pathList is set to an empty list
    set currentAirport to y
    add currentAirport to front of pathList
    do
        currentAirport = previousLocation for airport nodeValue
        add currentAirport to front of pathList
    while ( currentAirport != x )
    return pathList
}
```

```
list breadthFirstSearch (int x, int y)
{
  mark all airports as unvisited (set all previousLocation to -1);
  set the list AirportQueue to be empty
  add x to the end of the AirportQueue
  if ( bfs ( y, AirportQueue ) = = FALSE )
     print ("You can NOT get from airport " + x + " to airport " + y + " in one or more flights");
     return an empty list
  else
     print ("You can get from airport " + x + " to airport " + y + " in one or more flights");
     . . .
}
```

If AirportQueue is a data member of TravelNetwork, no need for it as a parameter!

```
list TravelNetwork::breadthFirstSearch (int x, int y)
{
  mark all airports as unvisited (set all previousLocation to -1);
  set the list AirportQueue to be empty
  add x to the end of the AirportQueue
  if ( bfs ( y ) = = FALSE )
     print (“You can NOT get from airport “ + x + “ to airport “ + y + “ in one or more flights”);
     return an empty list
  else
     print (“You can get from airport “ + x + “ to airport “ + y + “ in one or more flights”);
     . . .
}
```

If AirportQueue is a data member of TravelNetwork, no need for it as a parameter!

```
boolean bfs ( int b, AirportQueue )
{
  while ( the AirportQueue is not empty )
    {
      Set a to be the Airport at the front of the AirportQueue
      Remove Airport at the front of the AirportQueue

      for (each airport c that can be reached from a in one flight)
          if ( airport c is unvisited (previousLocation is still -1 )  )
              mark airport c as visited (set previousLocation to a);
              if ( c == b )
                  return TRUE
              add c to the end of the AirportQueue
     }
  return FALSE
}
```

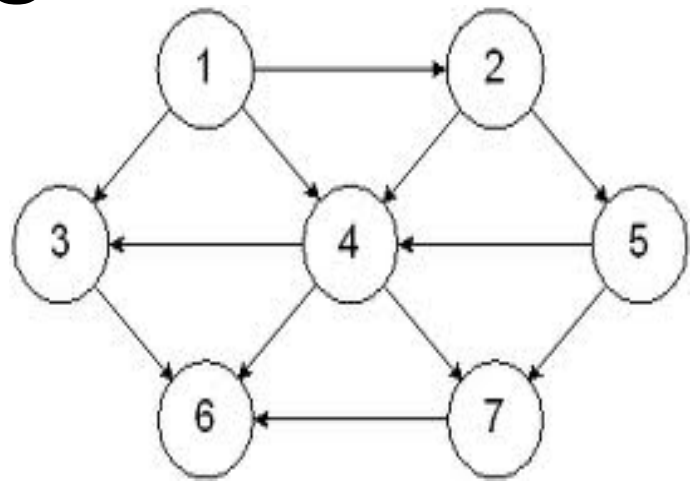
```
boolean TravelNetwork::bfs ( int b )
{
  while ( the AirportQueue is not empty )
    {
      Set a to be the Airport at the front of the AirportQueue
      Remove Airport at the front of the AirportQueue

      for (each airport c that can be reached from a in one flight)
          if ( airport c is unvisited (previousLocation is still -1 )  )
              mark airport c as visited (set previousLocation to a);
              if ( c == b )
                  return TRUE
              add c to the end of the AirportQueue
     }
  return FALSE
}
```

```
list TravelNetwork::breadthFirstSearch (int x, int y)
{
  mark all airports as unvisited (set all previousLocation to -1);
  set the list AirportQueue to be empty
  add x to the end of the AirportQueue
  if ( bfs ( y ) = = FALSE )
     print (“You can NOT get from airport “ + x + “ to airport “ + y + “ in one or more flights”);
     return an empty list
  else
     print (“You can get from airport “ + x + “ to airport “ + y + “ in one or more flights”);
     . . .
}
```

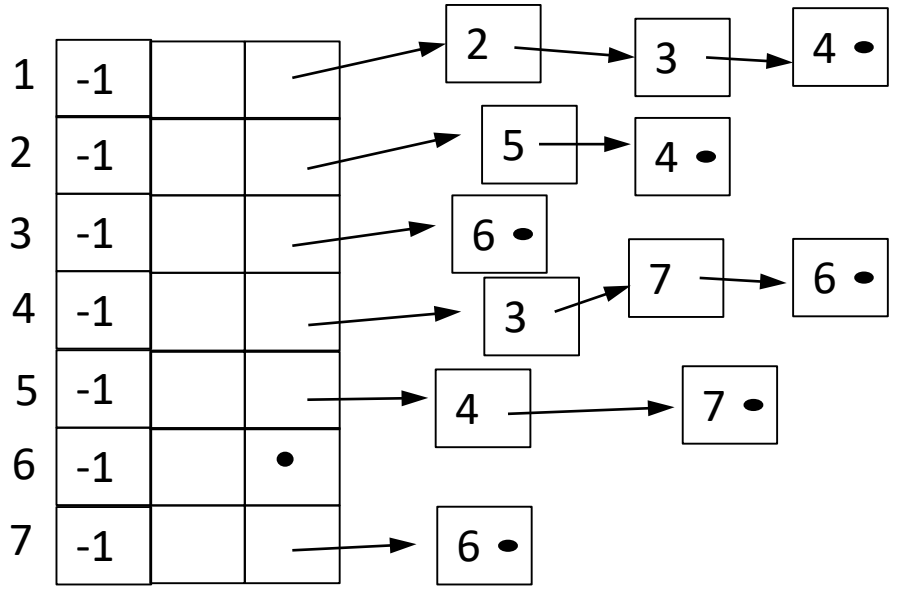
If AirportQueue is a data member of TravelNetwork, no need for it as a parameter!

Breadth First Search Walk Through

Shortest Path from 2 to 6



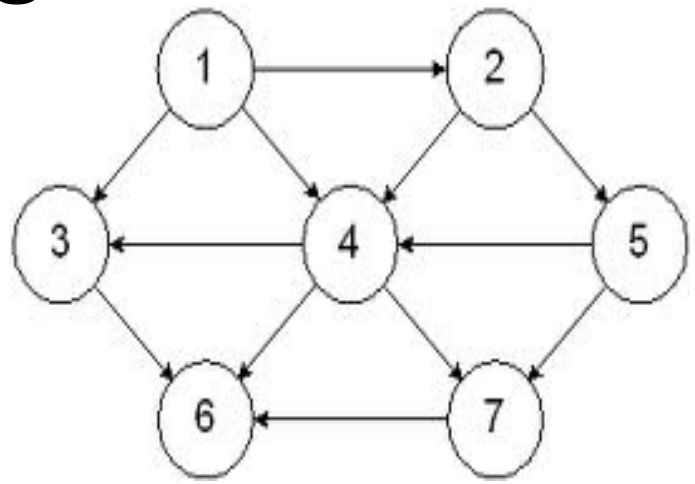
- Mark all Airport as “Unvisited”
  - -1 in the previousLocation data member
- Empty out the AirportQueue
- Add 2 to end of AirportQueue
- Call bfs (6)



AirportQueue: 2

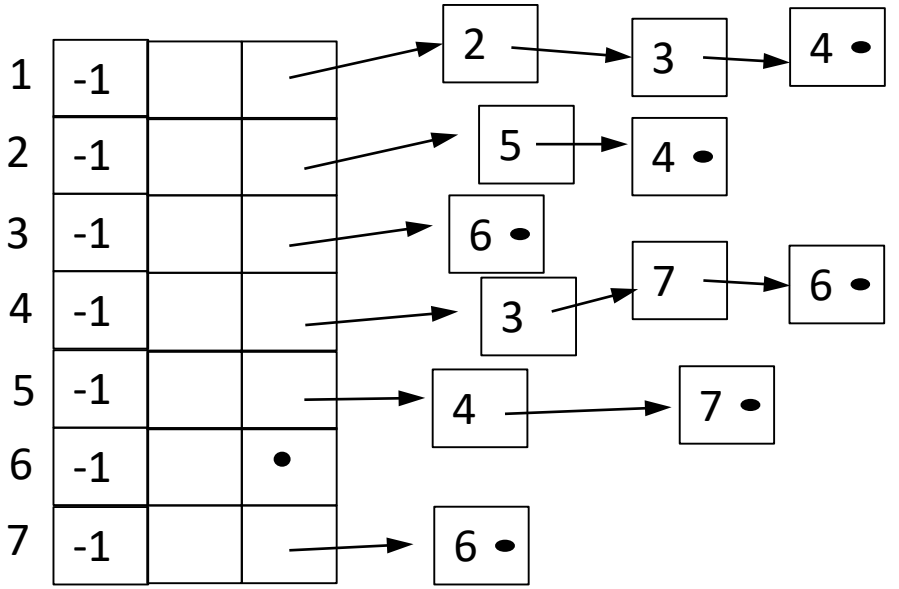
Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



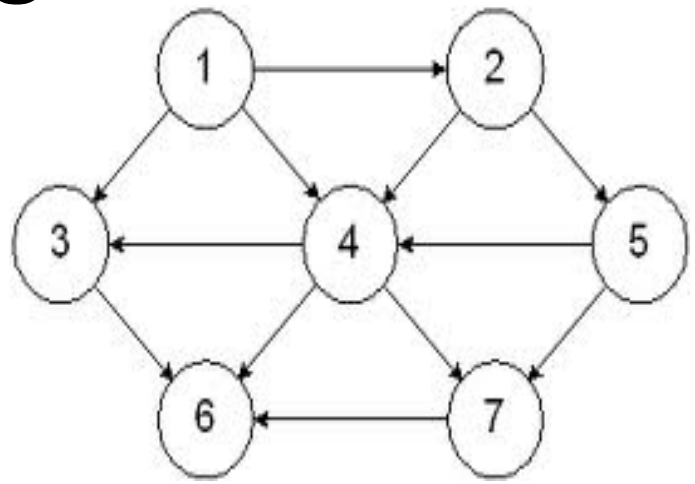
- AirportQueue is not Empty
  - Remove value at front and set to A (A = 2)
- For each Airport C that A can reach
  - 5 and 4
  - If (C is unvisited)
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE
    - Add C to end of Airport Queue

AirportQueue: ~~2~~



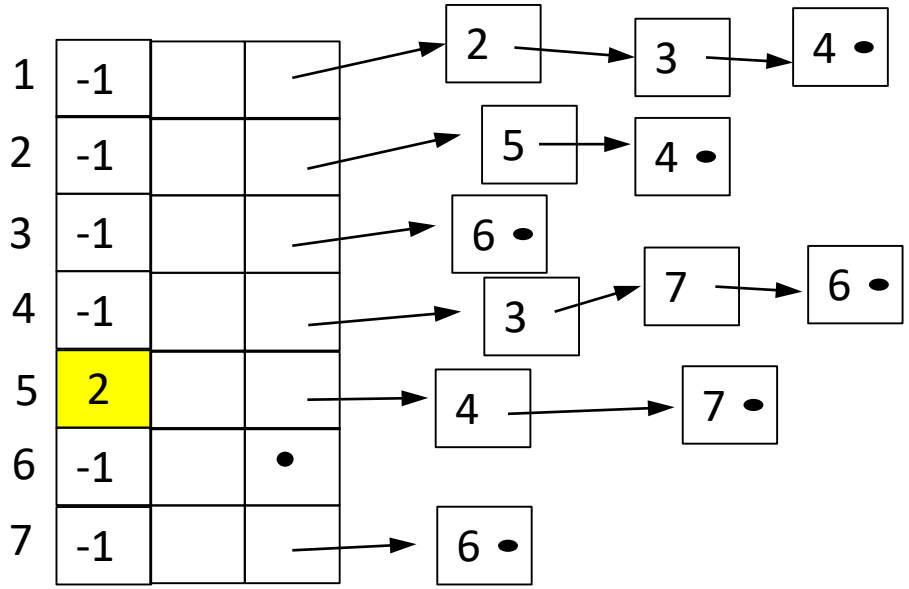
Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



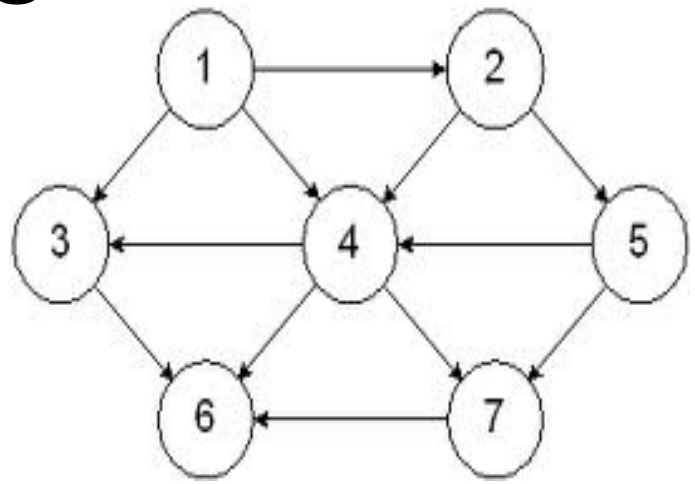
- AirportQueue is not Empty
  - Remove value at front and set to A (A = 2)
- For each Airport C that A can reach
  - 5 and 4
  - If (C is unvisited)
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE
    - Add C to end of Airport Queue

AirportQueue: ~~2~~ 5



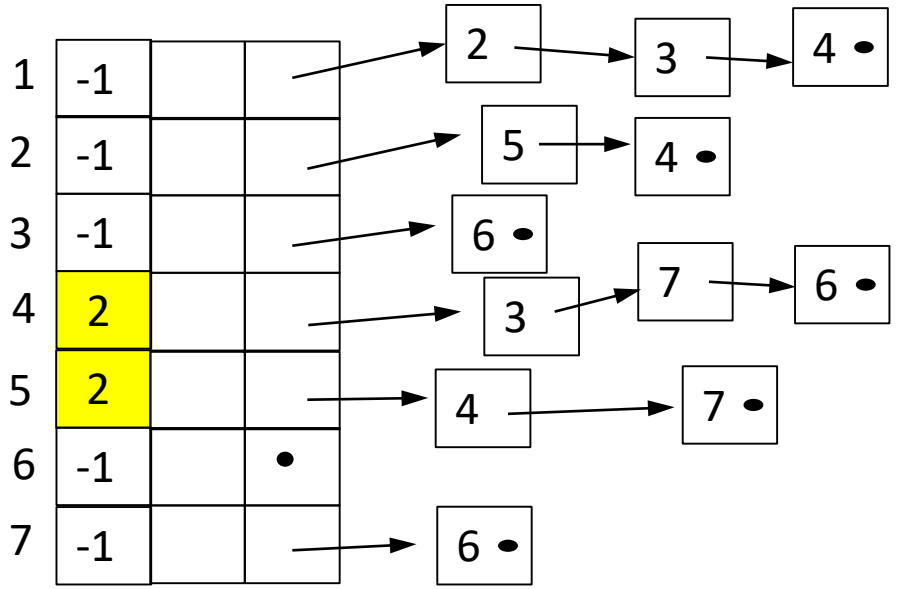
Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



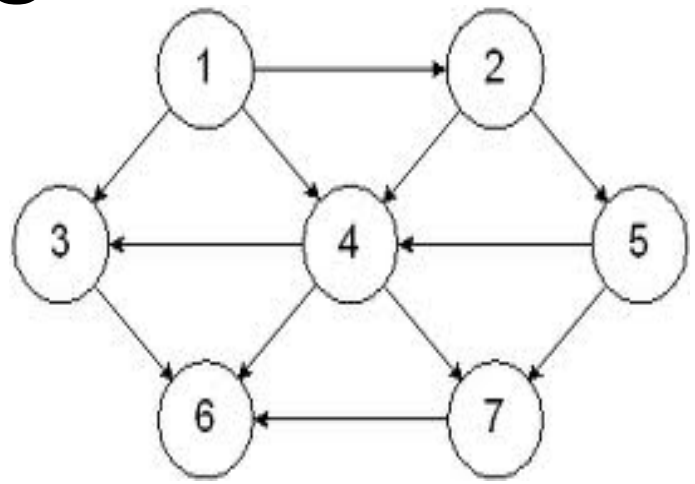
- AirportQueue is not Empty
  - Remove value at front and set to A (A = 2)
- For each Airport C that A can reach
  - 5 and 4
  - If (C is unvisited)
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE
    - Add C to end of Airport Queue

AirportQueue: ~~2~~ 5 4



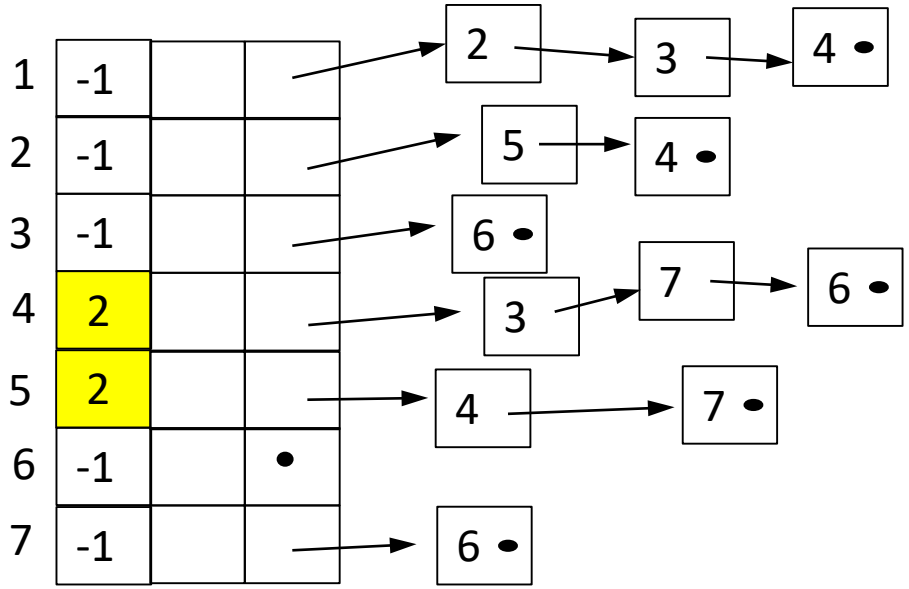
Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



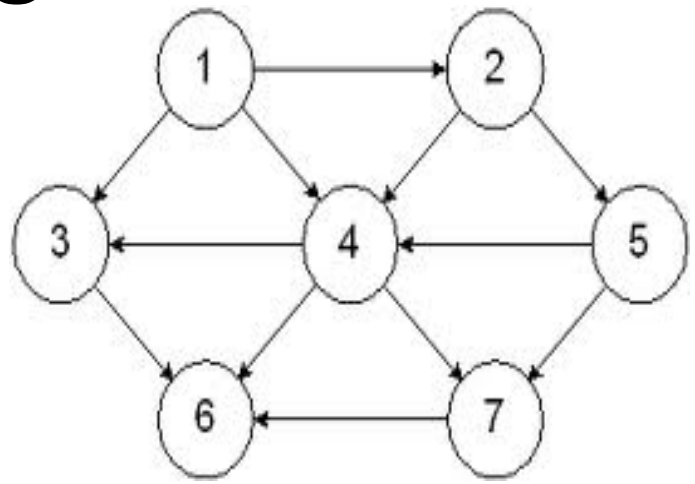
- AirportQueue is not Empty
  - Remove value at front and set to A (A = 5)
- For each Airport C that A can reach
  - 4 and 7
  - If (C is unvisited)
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE
    - Add C to end of Airport Queue

AirportQueue: ~~2~~ ~~5~~ 4



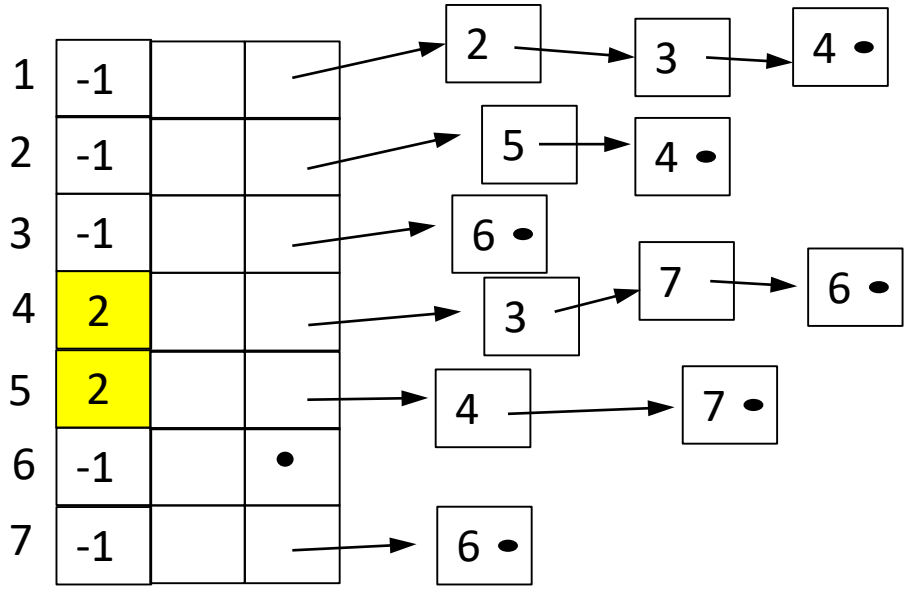
Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



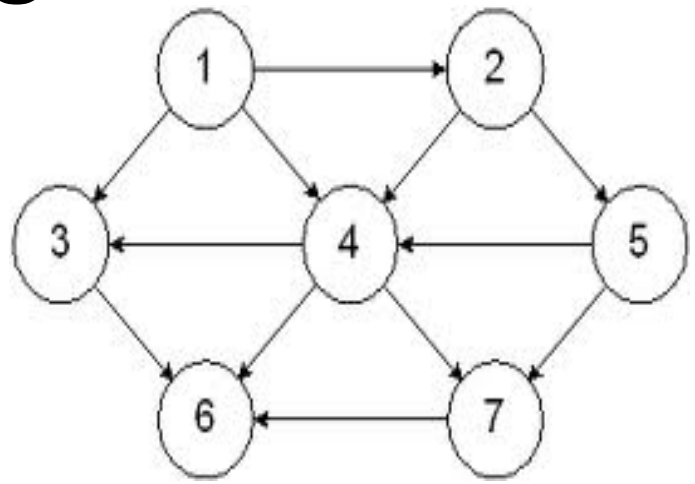
- AirportQueue is not Empty
  - Remove value at front and set to A (A = 5)
- For each Airport C that A can reach
  - 4 and 7
  - If (C is unvisited) // 4 is visited
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE
    - Add C to end of Airport Queue

AirportQueue: ~~2~~ ~~5~~ 4



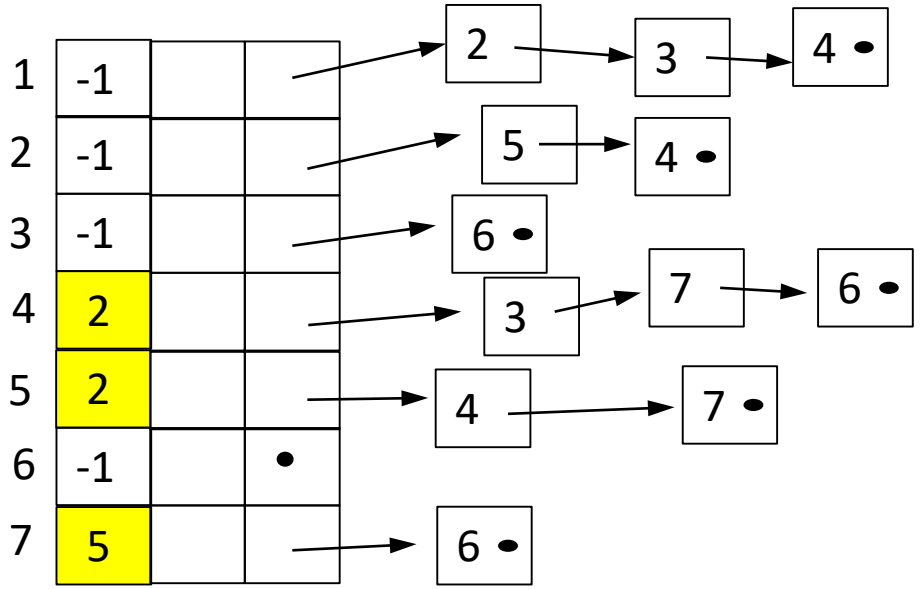
Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



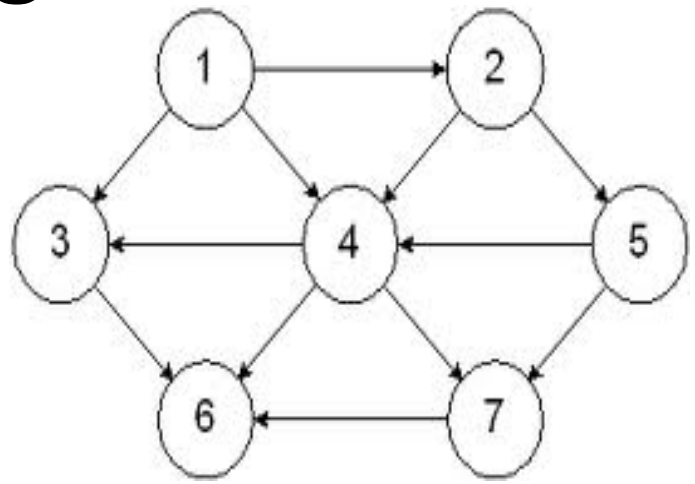
- AirportQueue is not Empty
  - Remove value at front and set to A (A = 5)
- For each Airport C that A can reach
  - 4 and 7
  - If (C is unvisited)
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE
    - Add C to end of Airport Queue

AirportQueue: ~~2~~ ~~5~~ 4 7

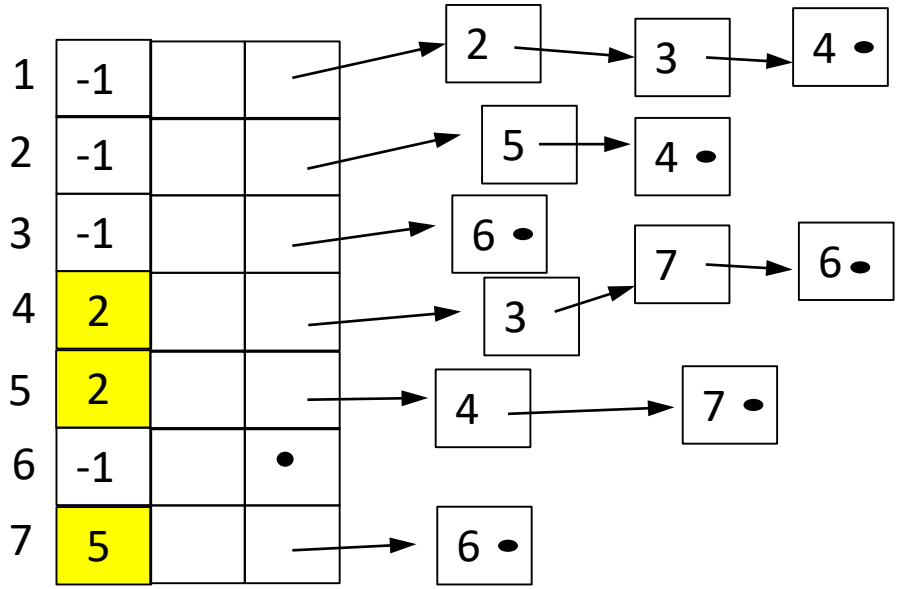


Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



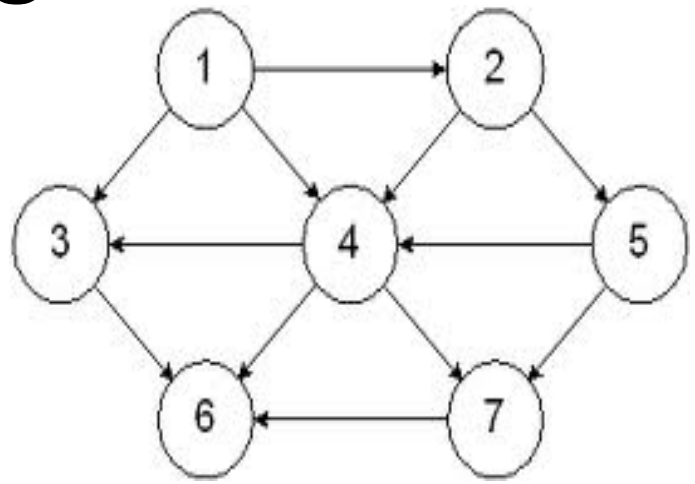
- AirportQueue is not Empty
  - Remove value at front and set to A (A = 4)
- For each Airport C that A can reach
  - 3, 7 and 6
  - If (C is unvisited)
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE
    - Add C to end of Airport Queue



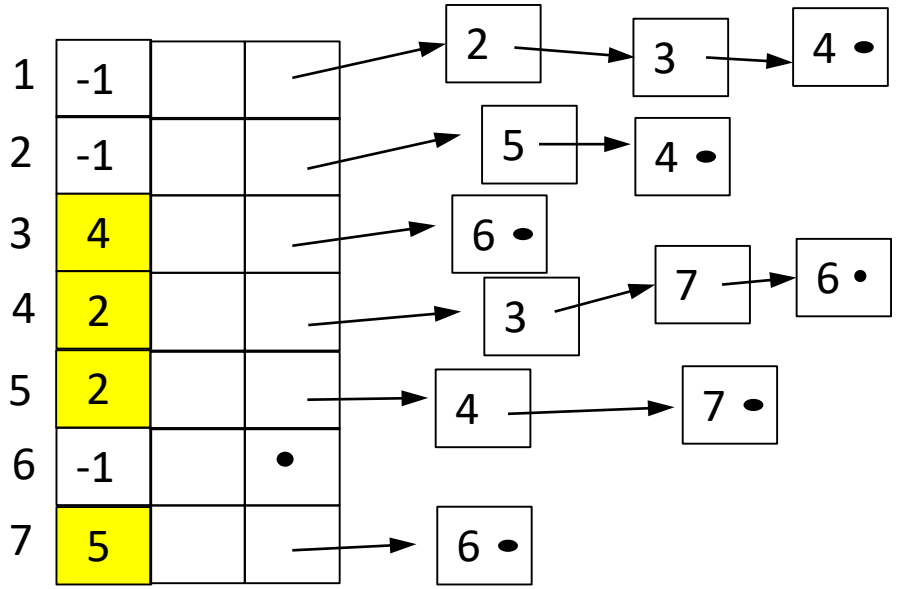
AirportQueue: ~~2~~ ~~5~~ ~~4~~ 7

Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



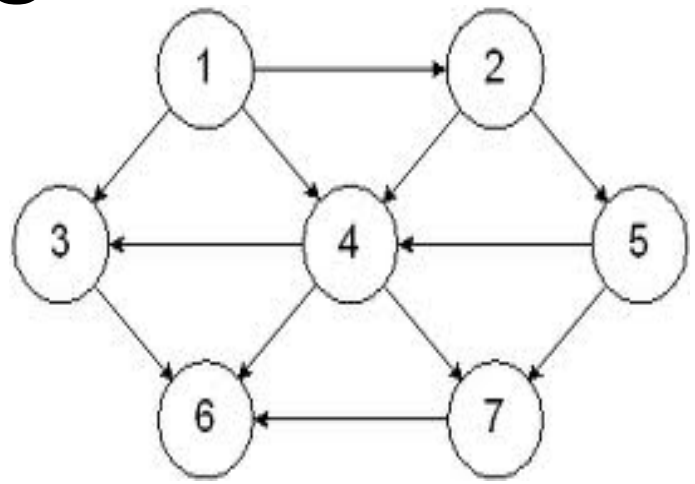
- AirportQueue is not Empty
  - Remove value at front and set to A (A = 4)
- For each Airport C that A can reach
  - 3, 7 and 6
  - If (C is unvisited)
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE
    - Add C to end of Airport Queue



AirportQueue: ~~2~~ ~~5~~ ~~4~~ 7 3

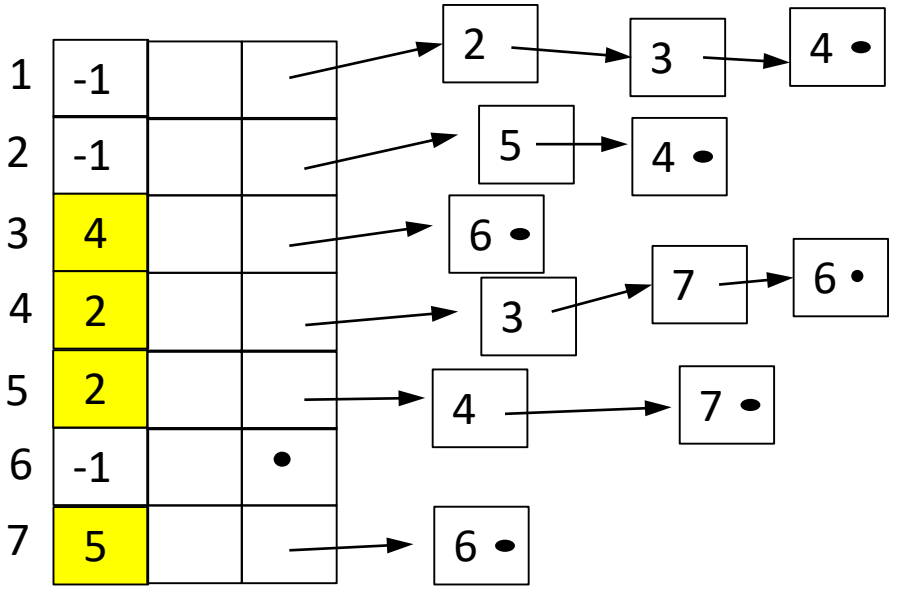
Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



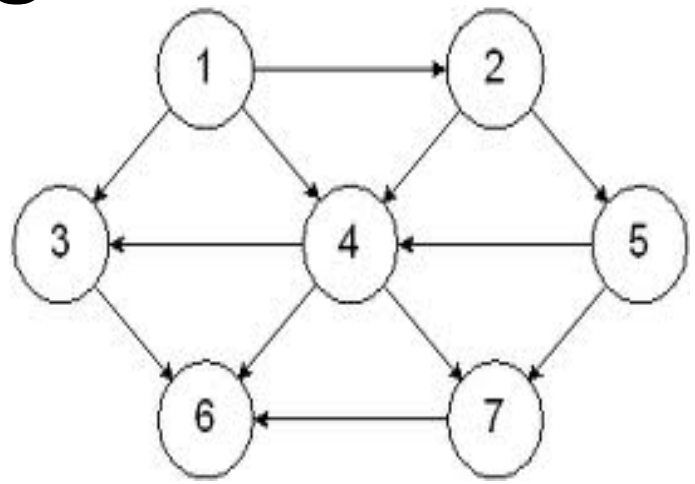
- AirportQueue is not Empty
  - Remove value at front and set to A (A = 4)
- For each Airport C that A can reach
  - 3, 7 and 6
  - If (C is unvisited) // 7 is visited
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE
    - Add C to end of Airport Queue

AirportQueue: ~~2~~ ~~5~~ ~~4~~ 7 3

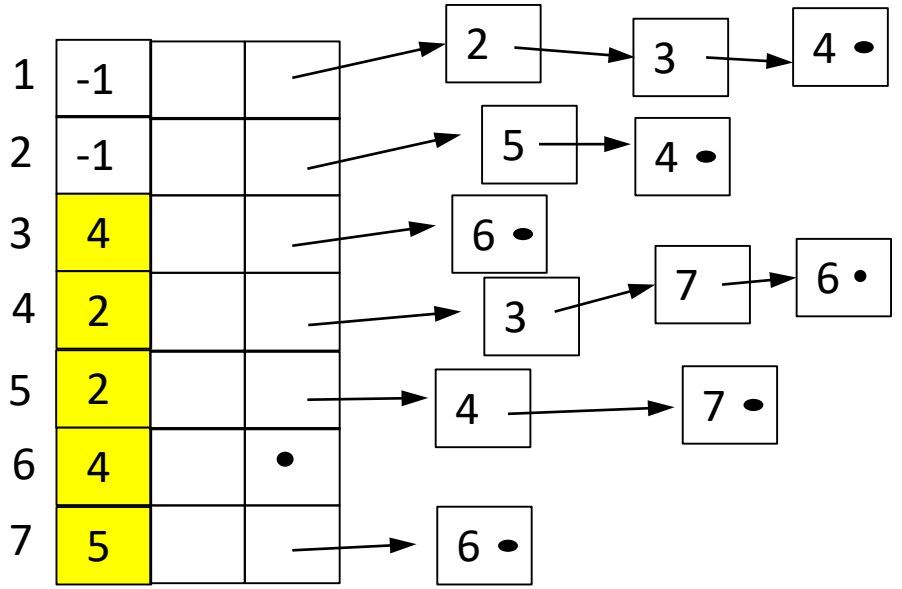


Breadth First Search Walk Through

Shortest Path from 2 to 6: in bsf (6)



- AirportQueue is not Empty
  - Remove value at front and set to A (A = 4)
- For each Airport C that A can reach
  - 3, 7 and 6
  - If (C is unvisited)
    - Mark C as visited (set previousLocation to A)
    - If (C == B) FOUND TARGET, RETURN TRUE

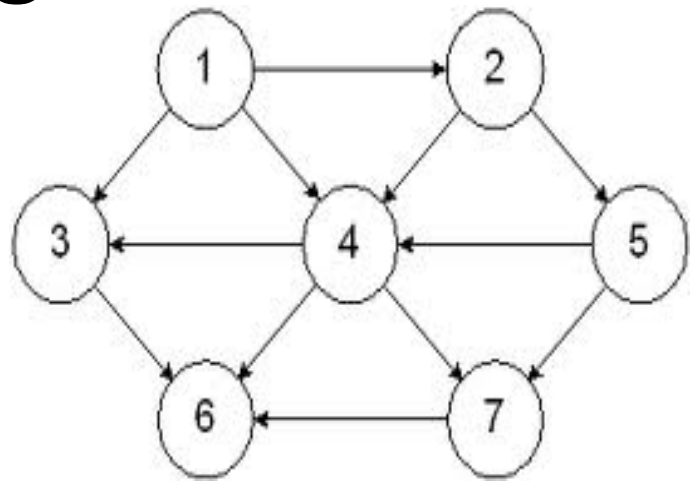


AirportQueue: ~~2~~ ~~5~~ ~~4~~ 7 3

```
list breadthFirstSearch (int x, int y)
{
  mark all airports as unvisited (set all previousLocation to -1);
  set the list AirportQueue to be empty
  add x to the end of the AirportQueue
  if ( bfs ( y, AirportQueue ) = = FALSE )
    print (“You can NOT get from airport “ + x + “ to airport “ + y + “ in one or more flights”);
    return an empty list
  else
    print (“You can get from airport “ + x + “ to airport “ + y + “ in one or more flights”);
    pathList is set to an empty list
    set currentAirport to y
    add currentAirport to front of pathList
    do
        currentAirport = previousLocation for airport nodeValue
        add currentAirport to front of pathList
    while ( currentAirport != x )
    return pathList
}
```

Breadth First Search Walk Through

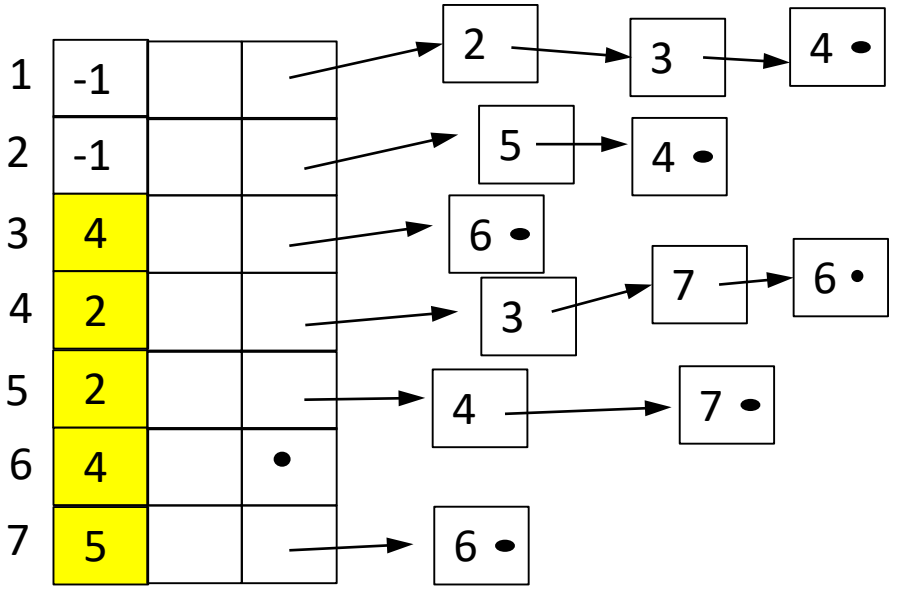
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

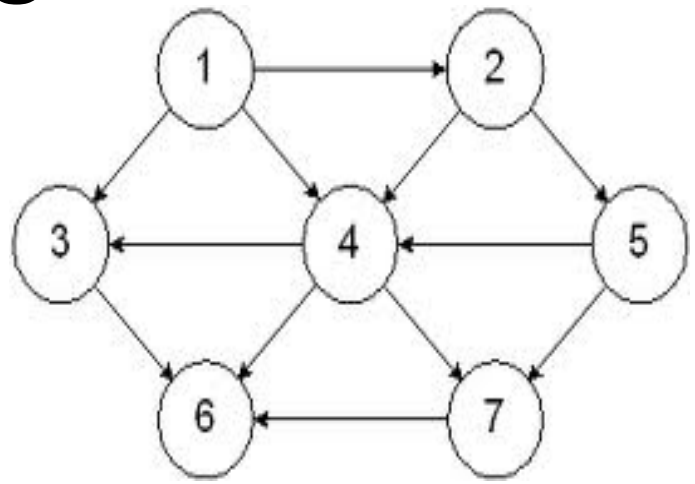
currentAirport:

pathList:



Breadth First Search Walk Through

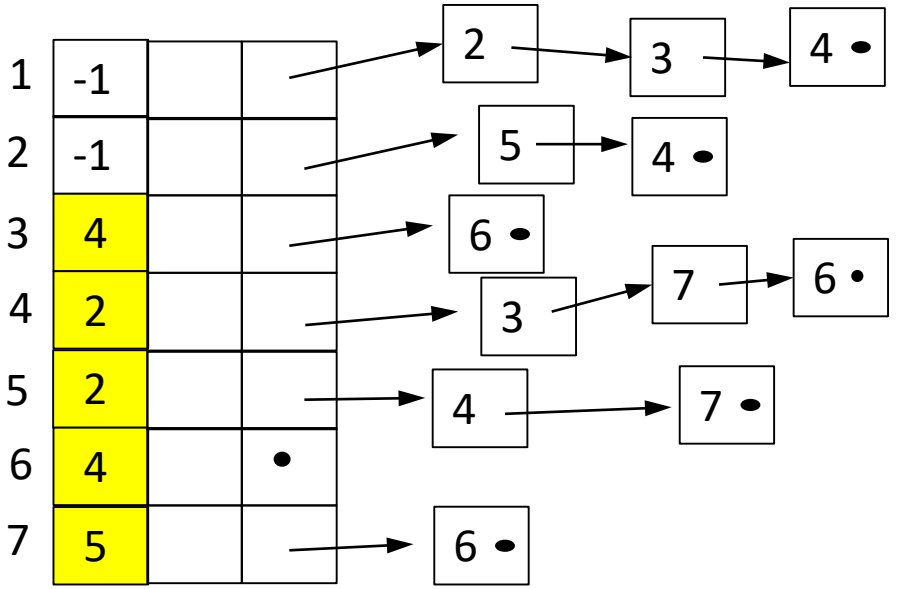
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

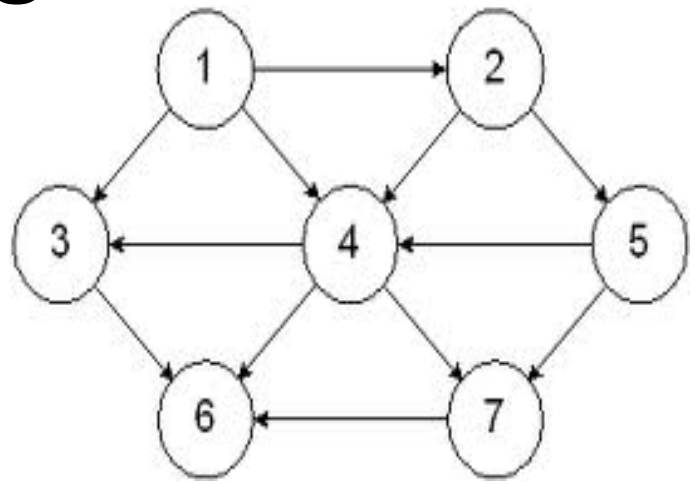
currentAirport:

pathList: •



Breadth First Search Walk Through

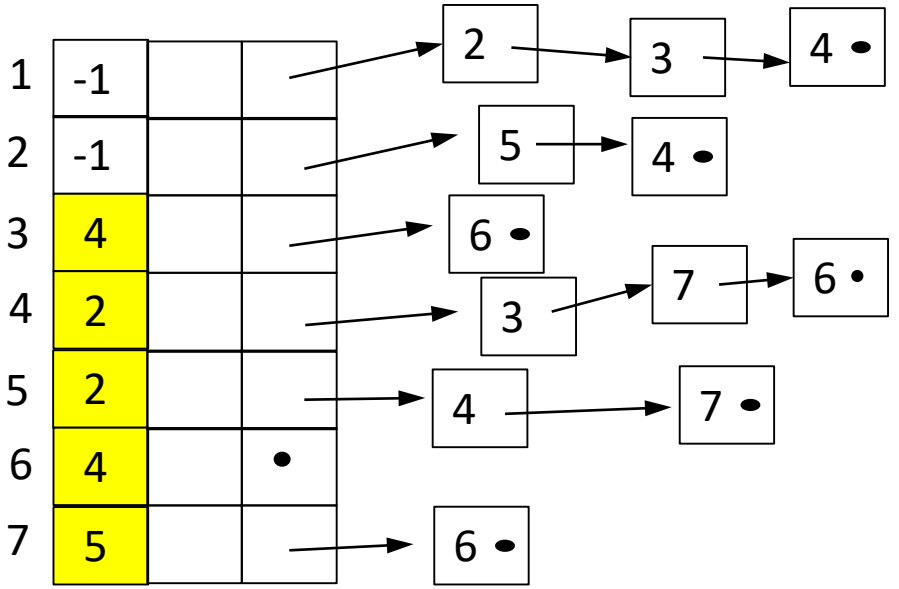
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

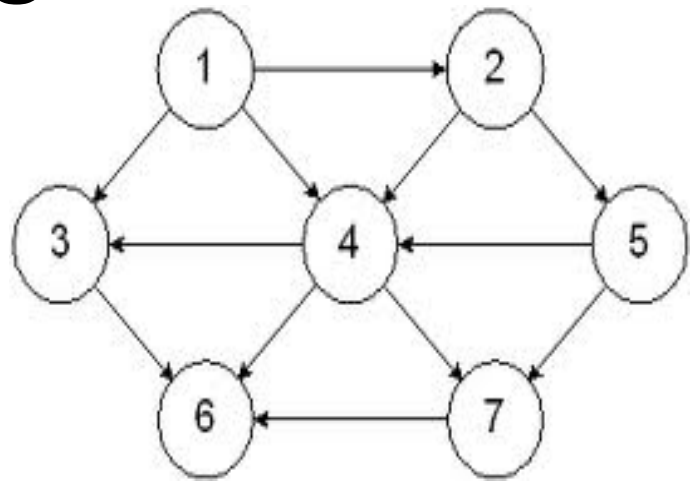
currentAirport: 6

pathList: •



Breadth First Search Walk Through

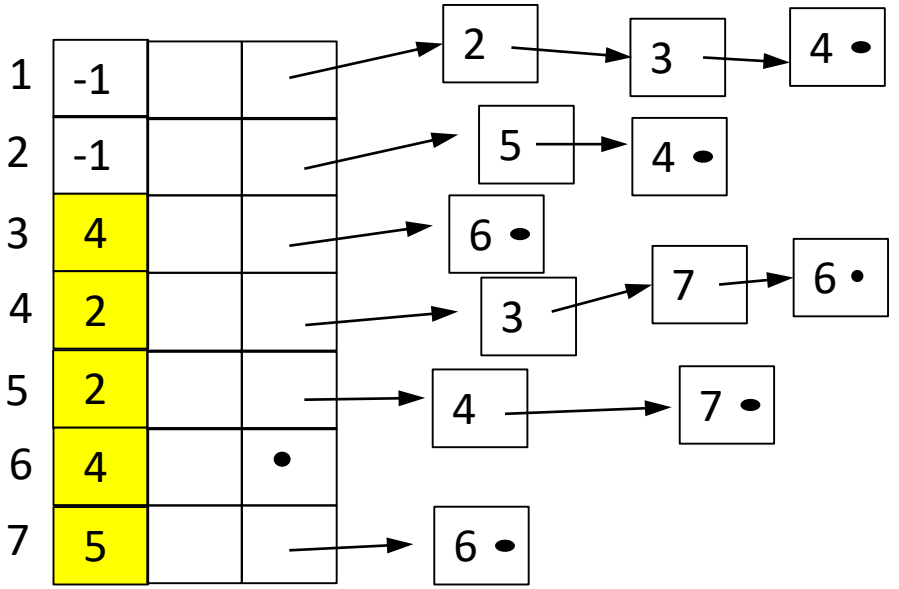
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

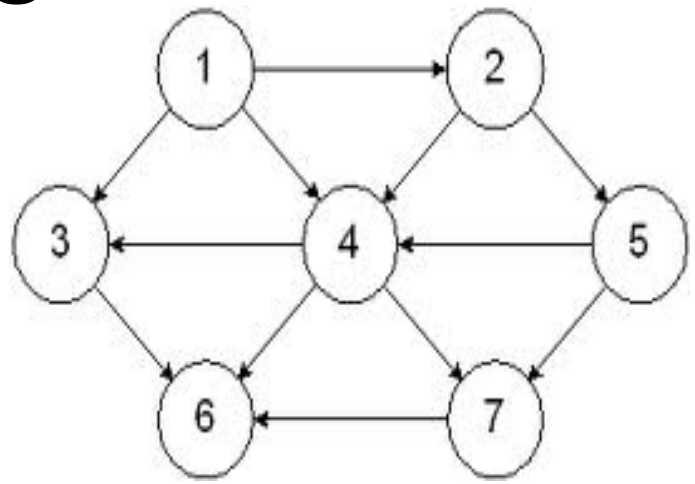
currentAirport: 6

pathList: → [6 •]



Breadth First Search Walk Through

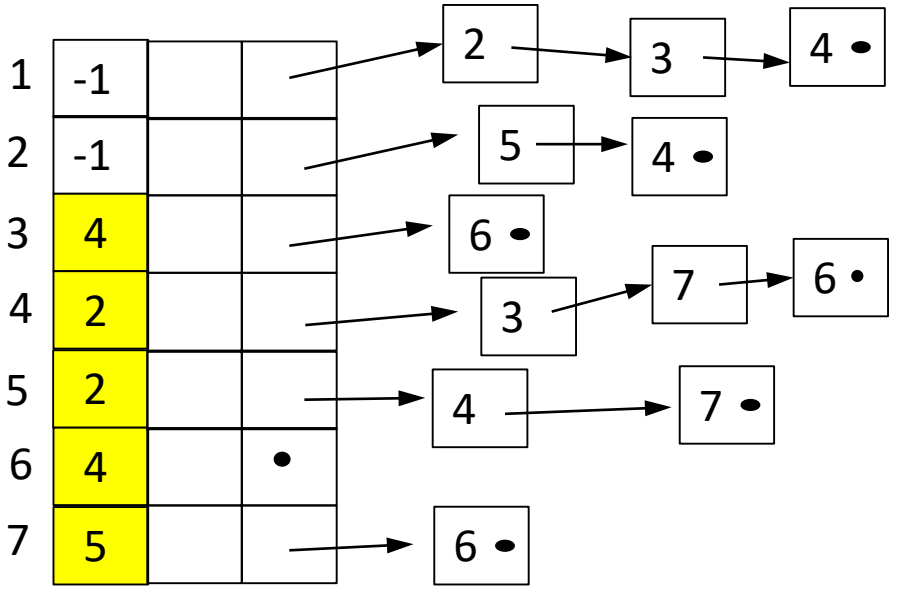
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

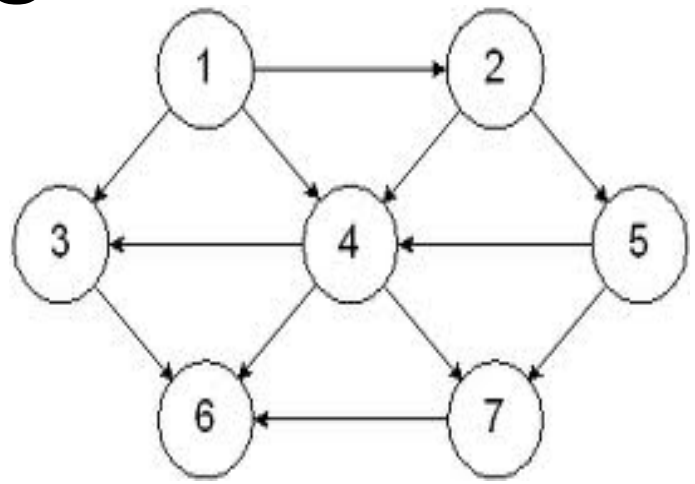
currentAirport: 6

pathList: → [6 •]



Breadth First Search Walk Through

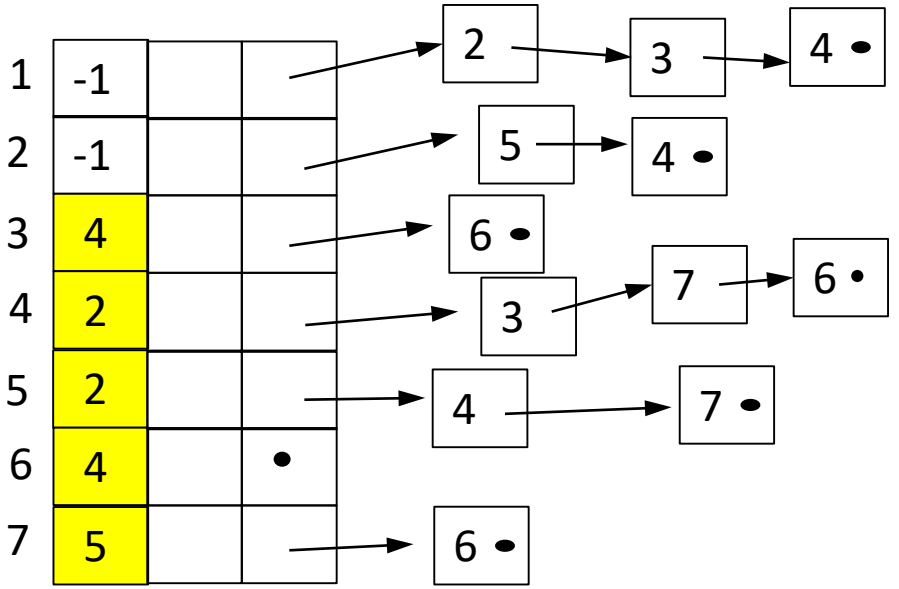
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

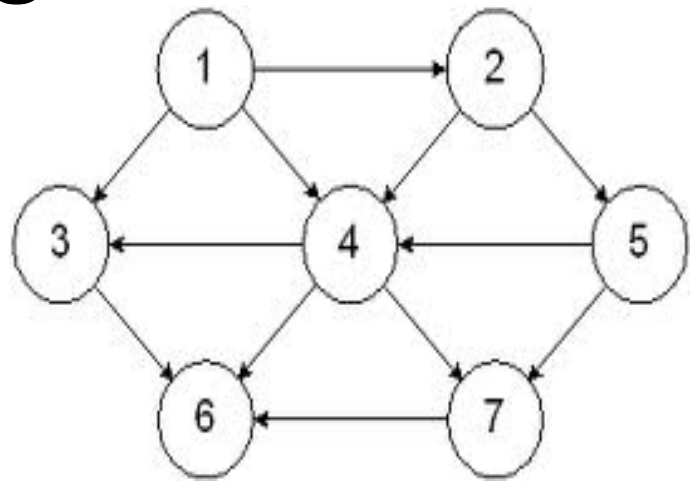
currentAirport: 4

pathList: → 6 •



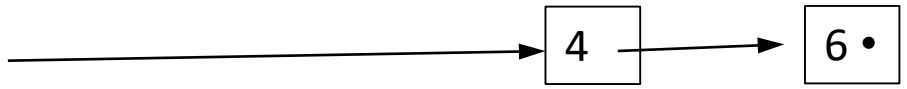
Breadth First Search Walk Through

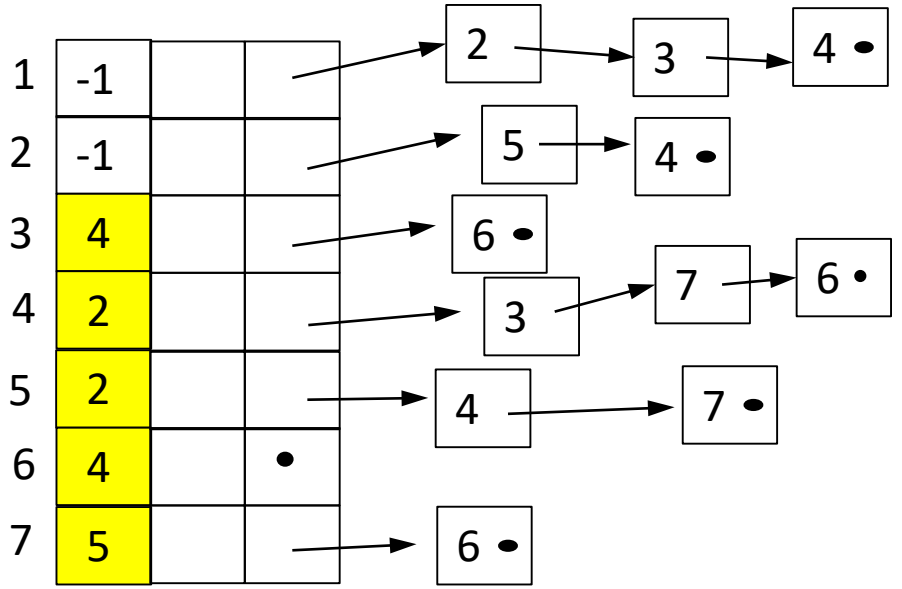
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

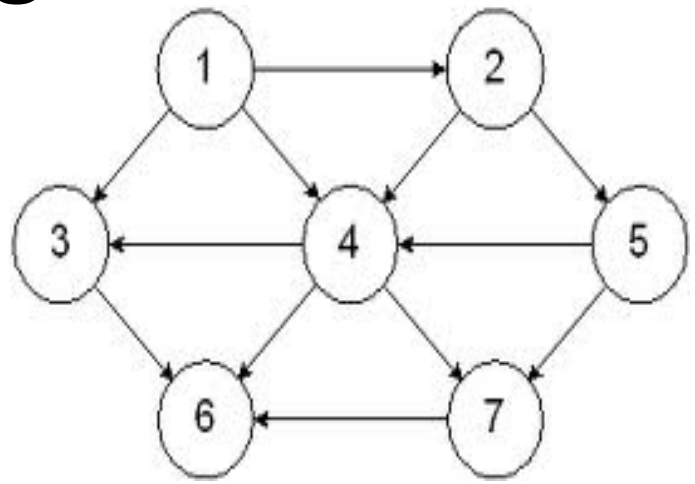
currentAirport: 4

pathList: 



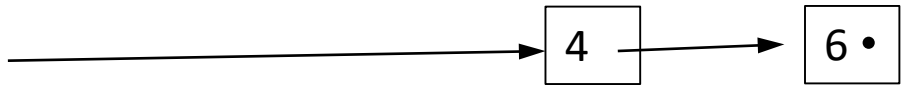
Breadth First Search Walk Through

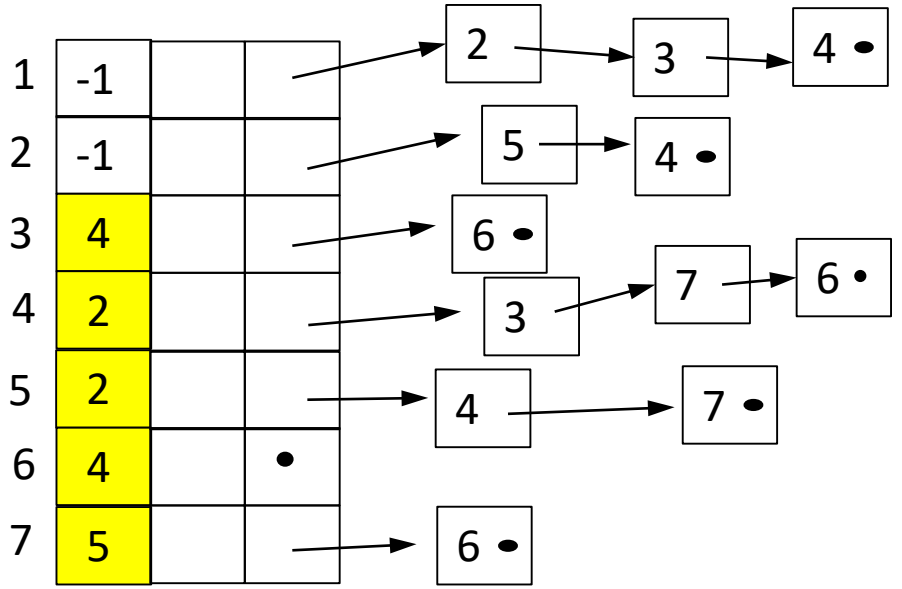
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

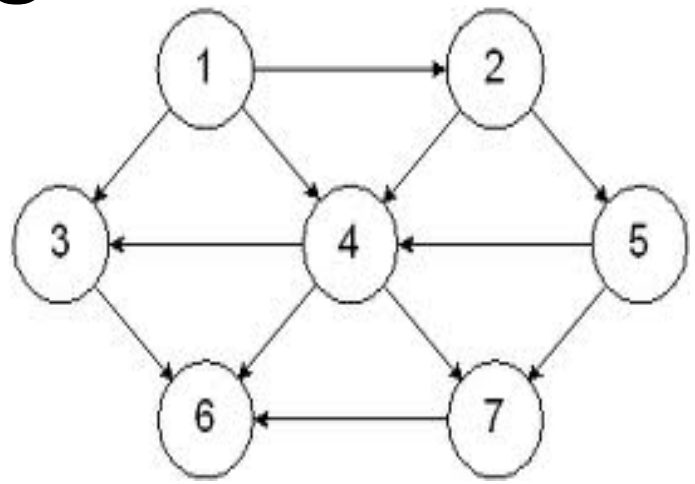
currentAirport: 4

pathList: 



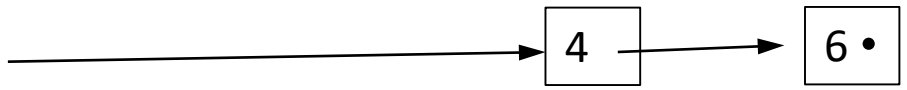
Breadth First Search Walk Through

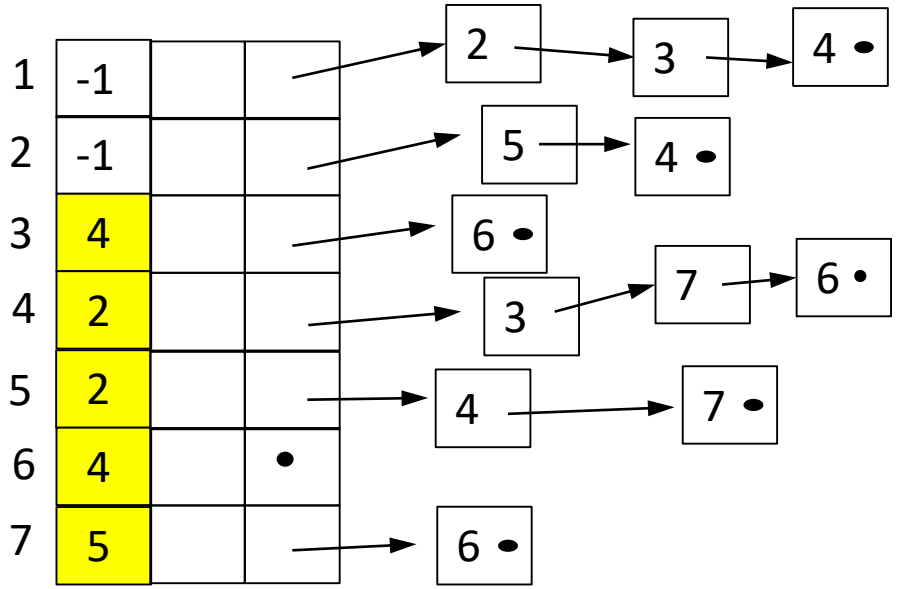
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

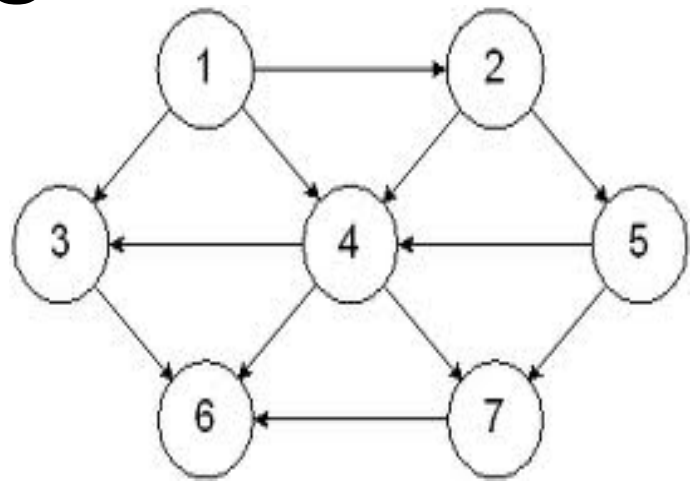
currentAirport: 2

pathList: 



Breadth First Search Walk Through

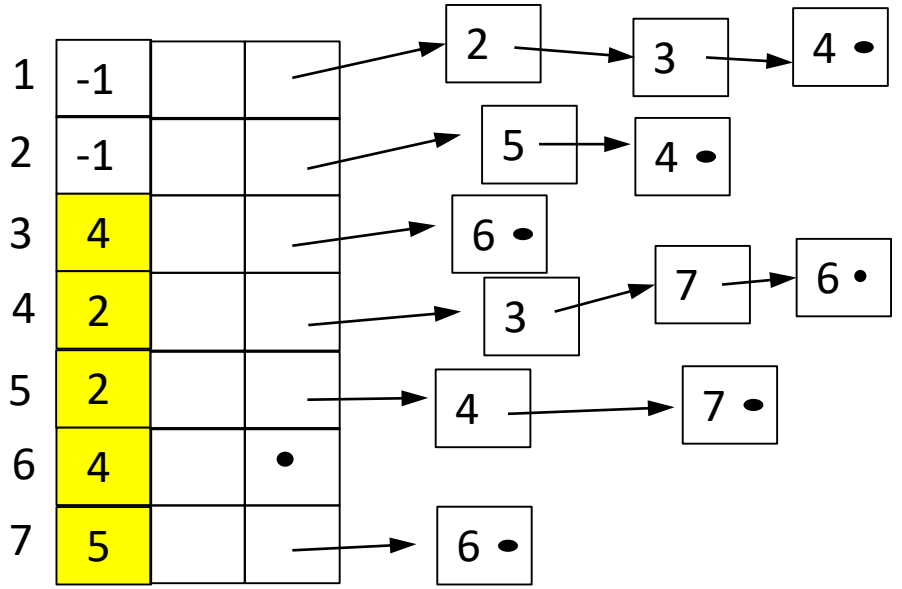
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

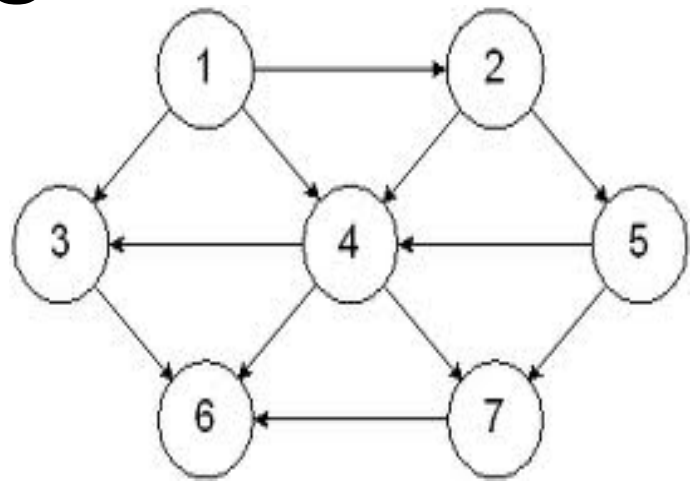
currentAirport: 2

pathList: 



Breadth First Search Walk Through

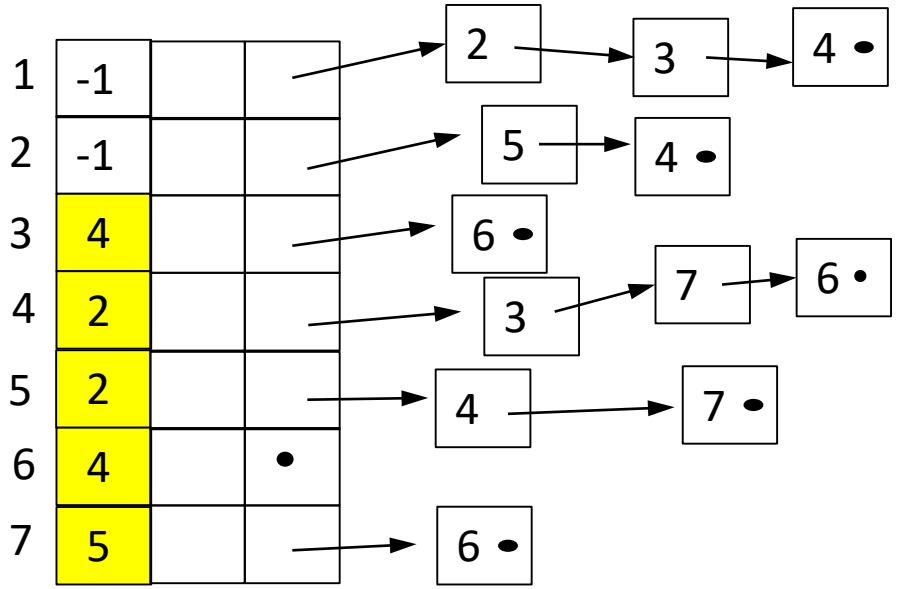
Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

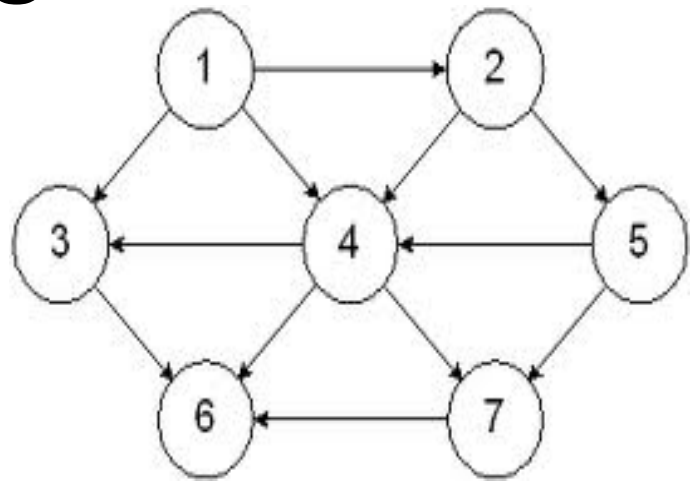
currentAirport: 2

pathList: 



Breadth First Search Walk Through

Build the Path from 2 to 6



- pathList is set to Empty
- Set currentAirport to Y (end target)
- Add currentAirport to front of pathList
- do
  - currentAiport = previousLocation for cA
  - Add currentAirport to front of pathList
- While (currentAirport != X)

currentAirport: 2

pathList: