Data Structures and Algorithms

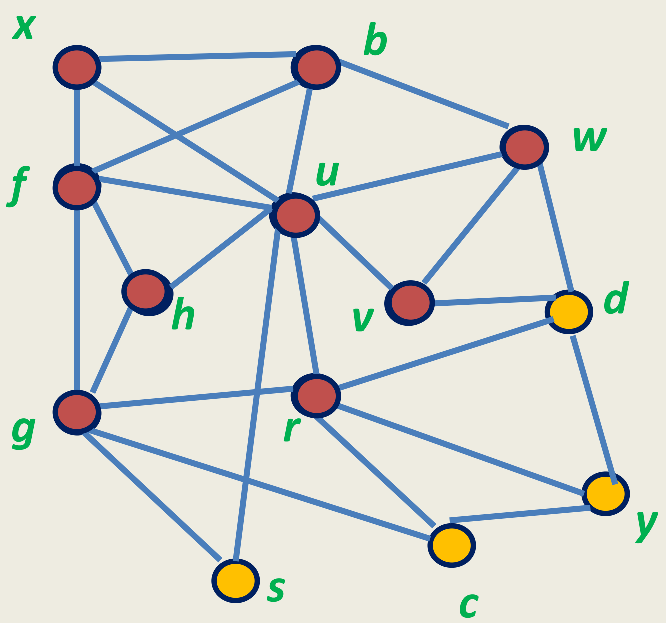
(ESO207)

Lecture 24

- BFS traversal (proof of correctness)
- BFS tree
- An important application of BFS traversal

Breadth First Search traversal

BFS Traversal
in Undirected Graphs



BFS traversal of G from a vertex x

```
BFS(G, x)        //Initially for each v, Distance(v) ← ∞  , and Visited(v) ← false.
{  CreateEmptyQueue(Q);
   Distance(x) ← 0;
   Enqueue(x,Q);     Visited(x) ← true;
   While(Not IsEmptyQueue(Q))
   {   v ←Dequeue(Q);
       For each neighbor w of v
       {
           if (Distance(w) = ∞)
           {   Distance(w) ← Distance(v) +1 ;  Visited(w) ← true;
               Enqueue(w, Q);
           }
       }
   }
```

Observations about BFS(x)

Observations:

- Any vertex v enters the queue at most once.

- Before entering the queue, **Distance**(v) is updated.

- When a vertex v is dequeued, v **processes** all its **unvisited** neighbors as follows
  - its distance is **computed**,
  - It is **enqueued**.

- A vertex v in the queue **is surely removed** from the queue during the algorithm.

Correctness of BFS traversal

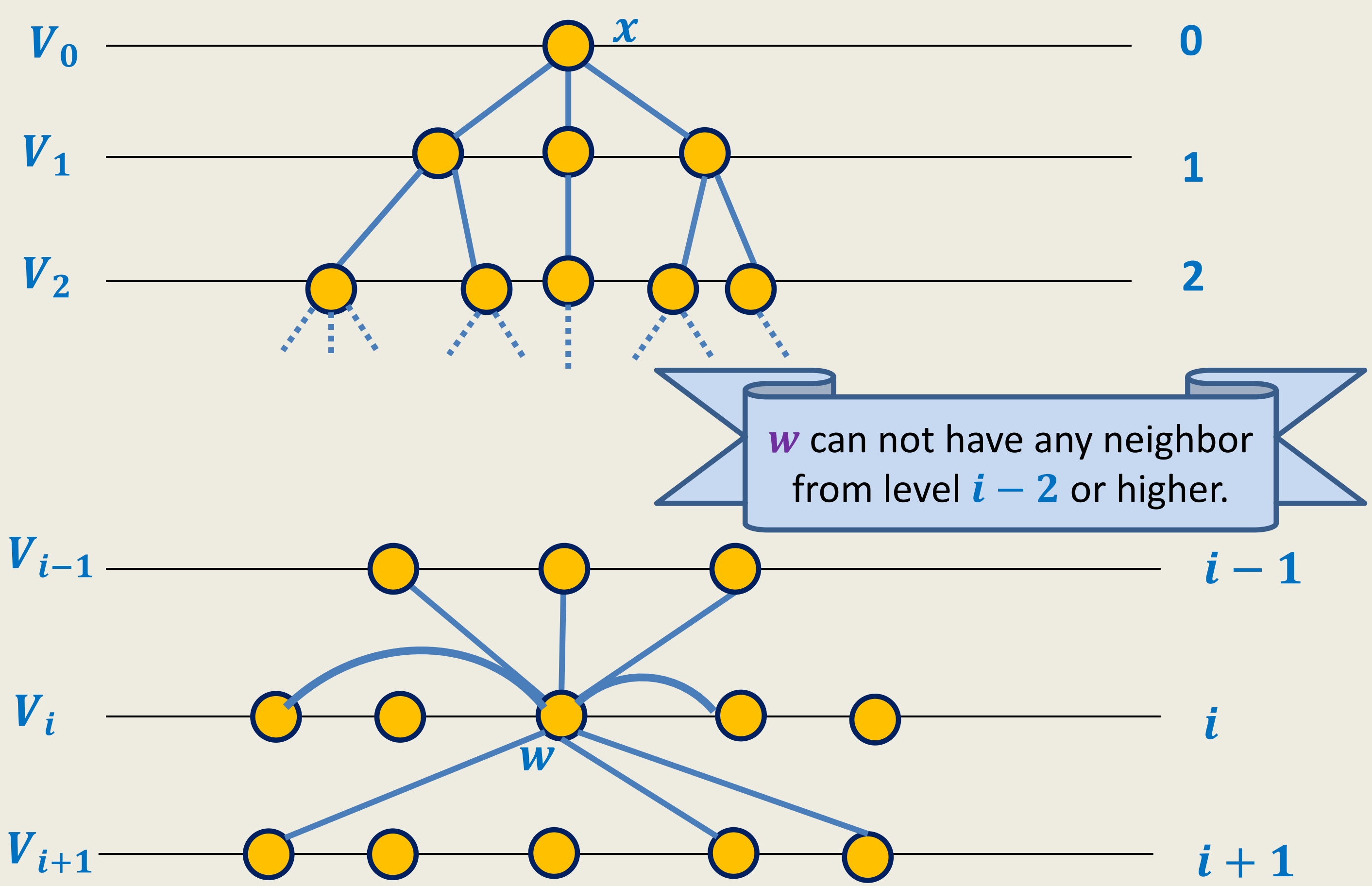
Question: What do we mean by correctness of **BFS**(**G**, x) ?

Answer:

- **All vertices** reachable from x **get visited**.
- Vertices are visited in **non-decreasing** order of distance from x (Try as exercise).
- At the end of the algorithm, **Distance**(v) **is the distance** of vertex v from x (Try as exercise).

The key idea

Partition the vertices according to their distance from x.



Correctness of BFS(x) traversal

Part 1

All vertices reachable from x get visited

Proof of Part 1

Theorem: Each vertex v reachable from x gets visited during **BFS**(G, x).

Proof:

(By **induction** on distance from x)

Inductive Assertion A(i) :

Every vertex v at distance i from x get visited.

Base case: $i = 0$.

x is the only vertex at distance **0** from x.

Right in the beginning of the algorithm **Visited**(x) ← **true**;

Hence the assertion **A**(**0**) is true.

Induction Hypothesis: **A**(j) is true for all $j < i$.

Induction step: To prove that **A**(i) is true.

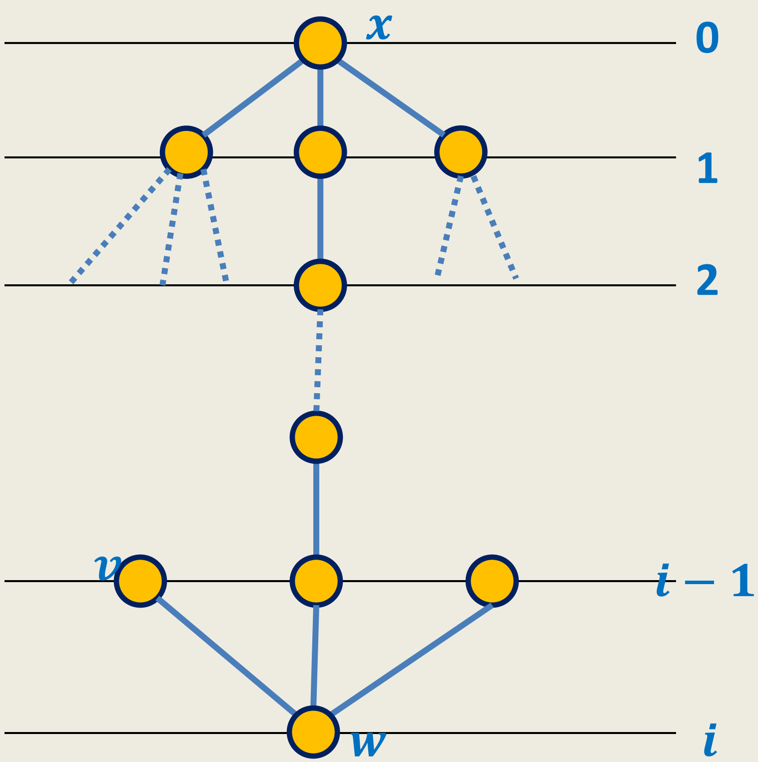
Let $w \in V_i$.

BFS traversal of G from a vertex x

```
BFS(G, x)        //Initially for each v, Distance(v) ← ∞  , and Visited(v) ← false.
{  CreateEmptyQueue(Q);
   Distance(x) ← 0;
   Enqueue(x,Q);    Visited(x) ← true;
   While(Not IsEmptyQueue(Q))
   {   v ←Dequeue(Q);
       For each neighbor w of v
       {
           if (Distance(w) = ∞)
           {   Distance(w) ← Distance(v) +1 ;   Visited(w) ← true;
               Enqueue(w, Q);
           }
       }
   }
```

Induction step:

To prove that **w** ϵ V_i is visited during **BFS**(x)



Let v ϵ V_{i-1} be any neighbor of w.

By induction hypothesis,

v gets visited during **BFS**(x).

So v gets **Enqueued**.

Hence v gets **dequeued**.

Focus on the moment when v is **dequeued**,

➔

v scans all its neighbors and

marks all its unvisited neighbors as **visited**.

Hence w gets **visited** too

This proves the induction step.

if not already visited.

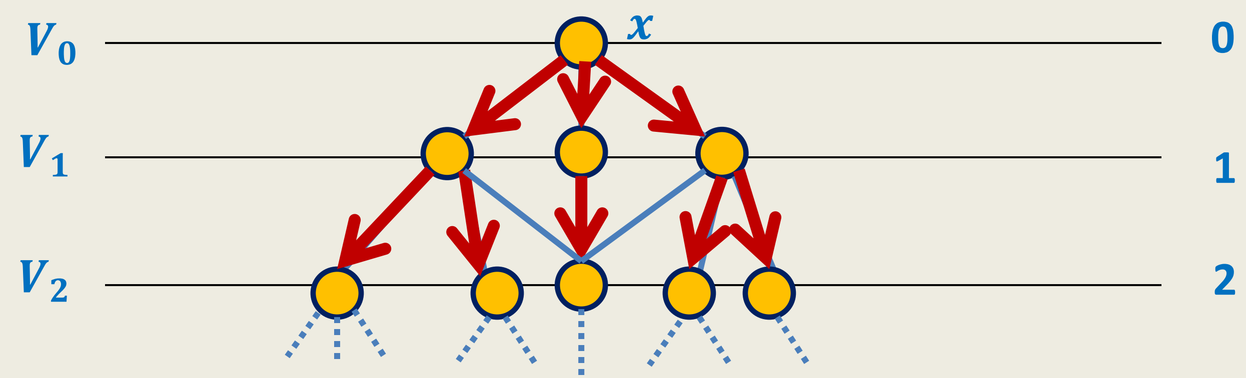
Hence by the principle of mathematical induction, **A**(i) holds for each i.

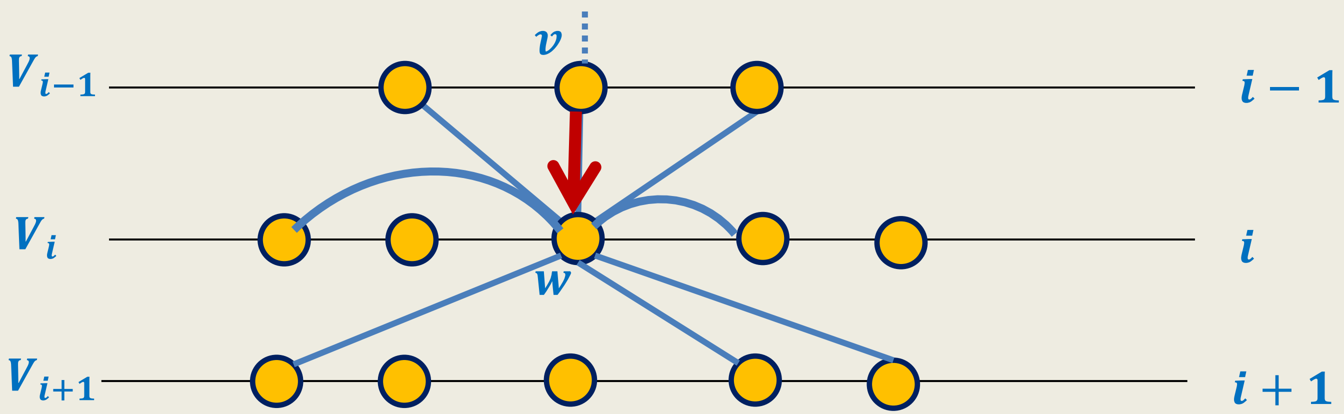
This completes the proof of **part 1**.

BFS tree

BFS traversal gives a tree

Perform BFS traversal from x.





A nontrivial application of BFS traversal

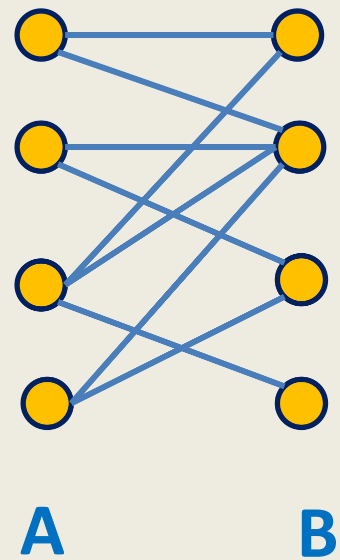
Determining

if a graph is bipartite

Bipartite graph

Definition: A graph **G**=(**V**,**E**) is said to be bipartite
if its vertices can be partitioned into two sets **A** and **B**
such that every edge in **E** has one endpoint in **A** and another in **B**.

Is this graph bipartite ?

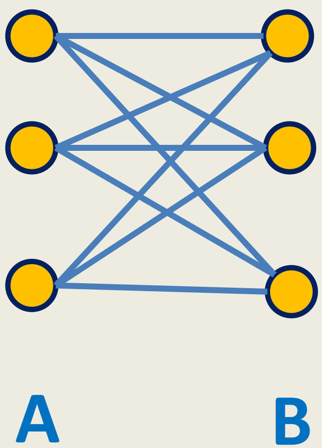


YES

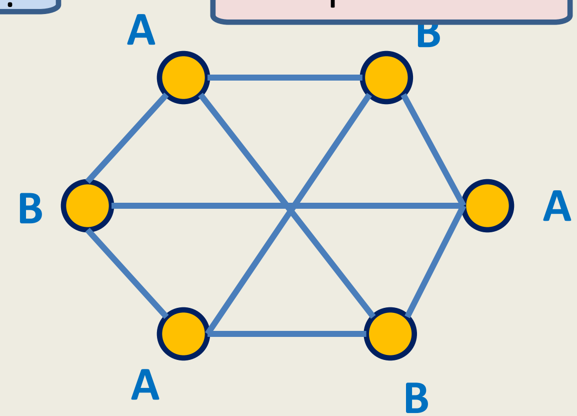
Nontriviality
in determining whether a graph is bipartite

Both are same graph but drawn in different ways. Can you see it ?

Is this graph bipartite ?



=

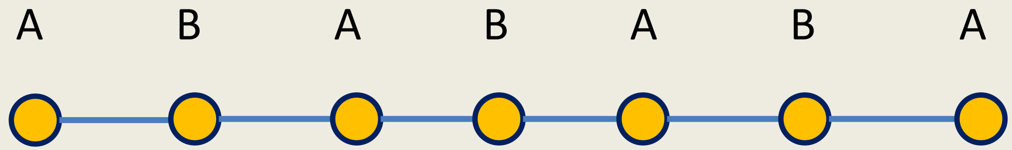


bipartite

bipartite

Bipartite graph

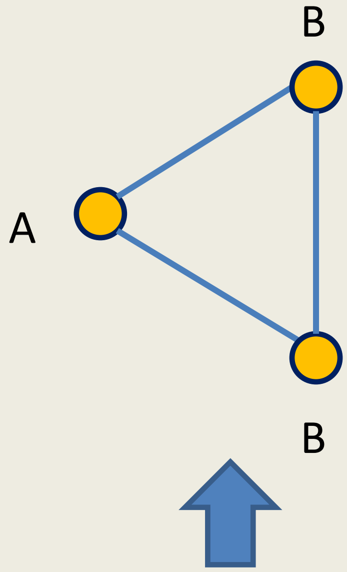
Question: Is a path bipartite ?



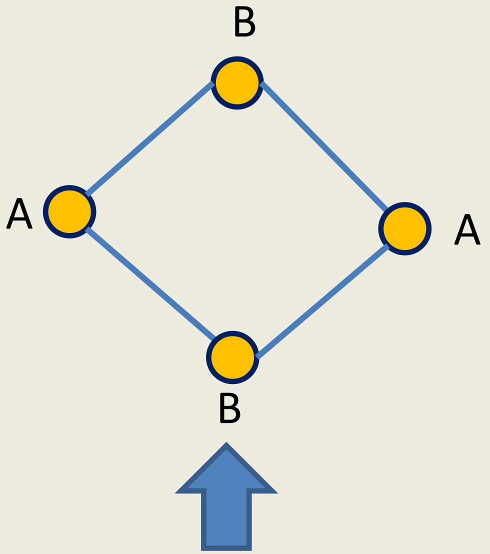
Answer: Yes

Bipartite graph

Question: Is a cycle bipartite ?



non-bipartite

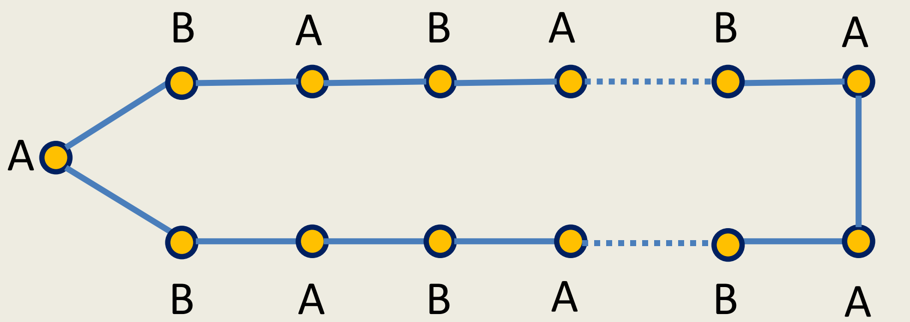


bipartite

Bipartite graph

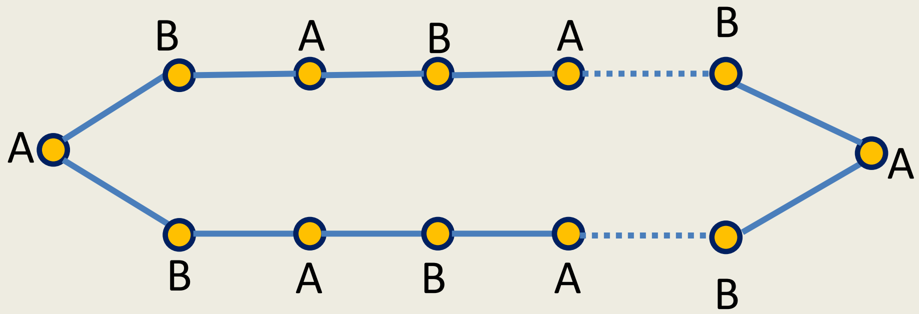
Question: Is a cycle bipartite ?

Odd length cycle



non-bipartite

Even length cycle



bipartite

Subgraph

A subgraph of a graph **G**=(**V**,**E**)

is a graph **G'**=(**V'**,**E**') such that

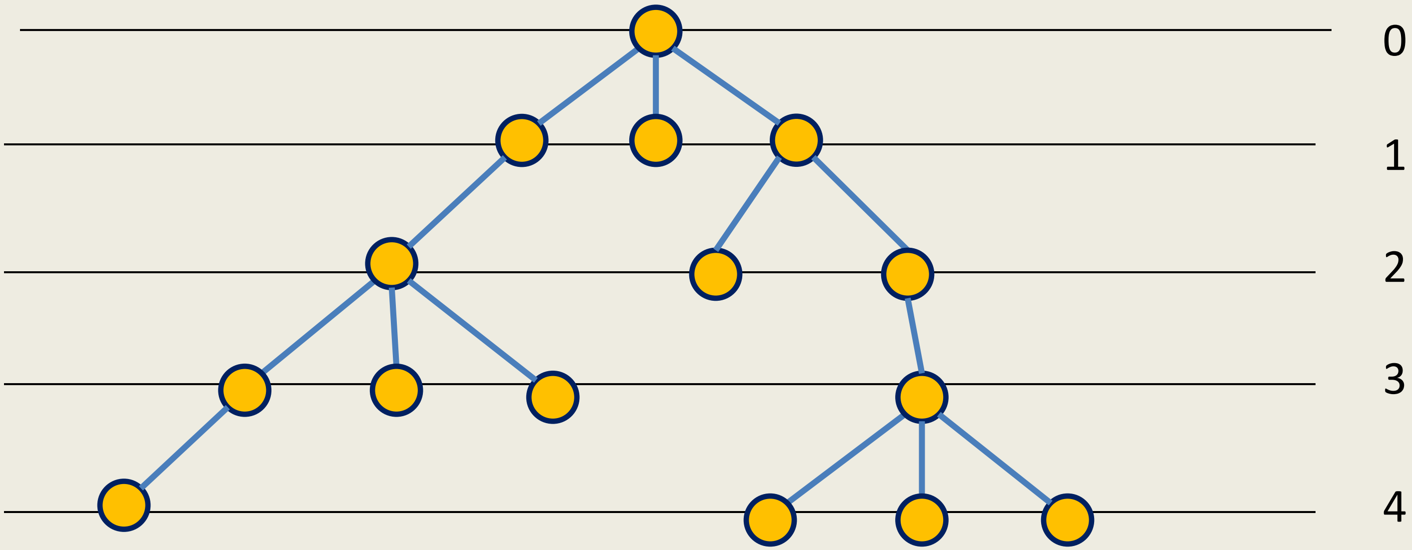
- $V' \subseteq V$
- $E' \subseteq E \cap (V' \times V')$

Question: If **G** has a subgraph which is **an odd cycle**, is **G** bipartite ?

Answer: **No**.

Bipartite graph

Question: Is a tree bipartite ?



Answer: Yes

Even level vertices: **A**

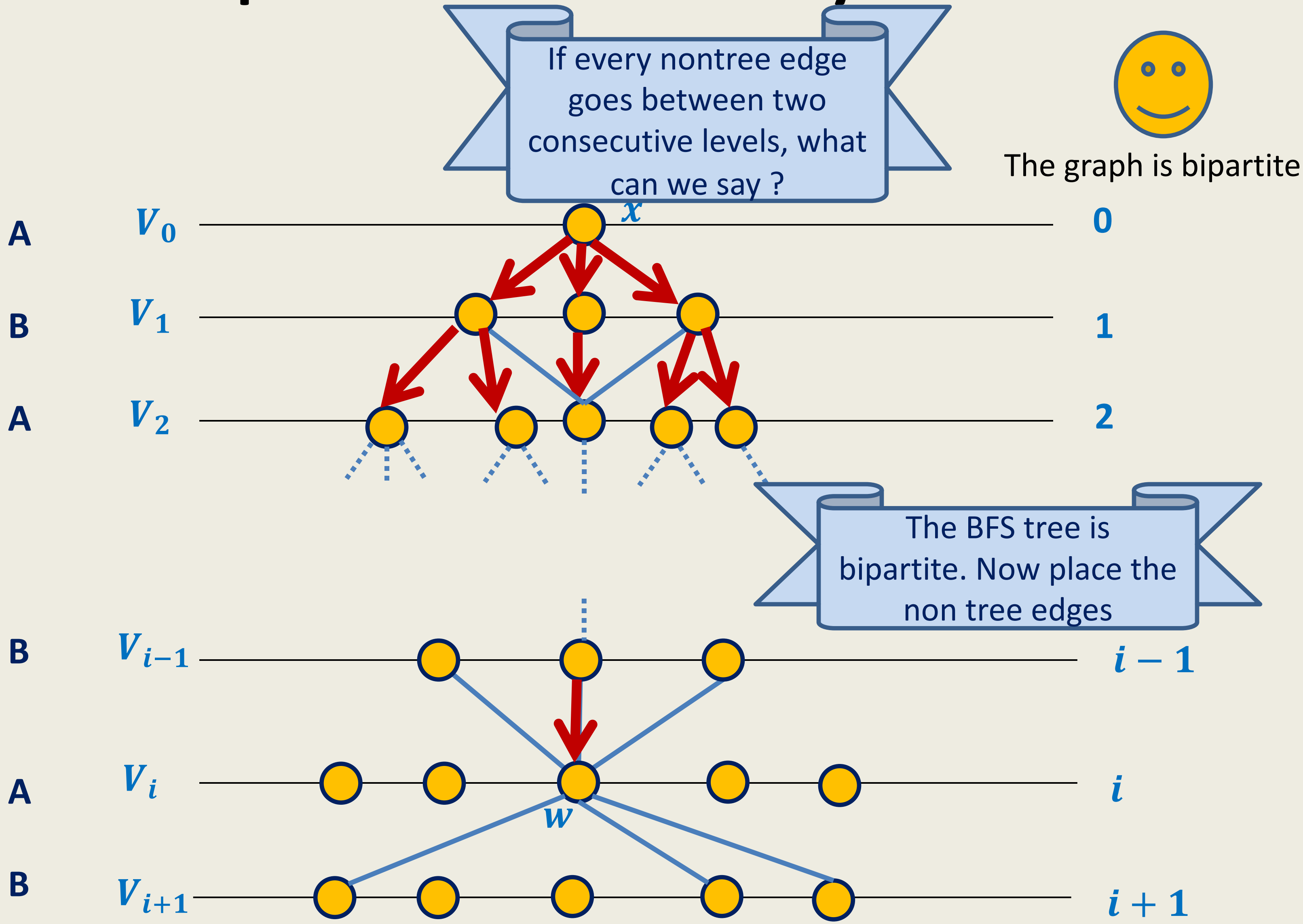
Odd level vertices: **B**

An algorithm for determining if a given graph is bipartite

Assumption:

the graph is a single connected component

Compute a BFS tree at any vertex x.



Observation:

If every non-tree edge goes between two consecutive levels of **BFS** tree, then the graph is bipartite.

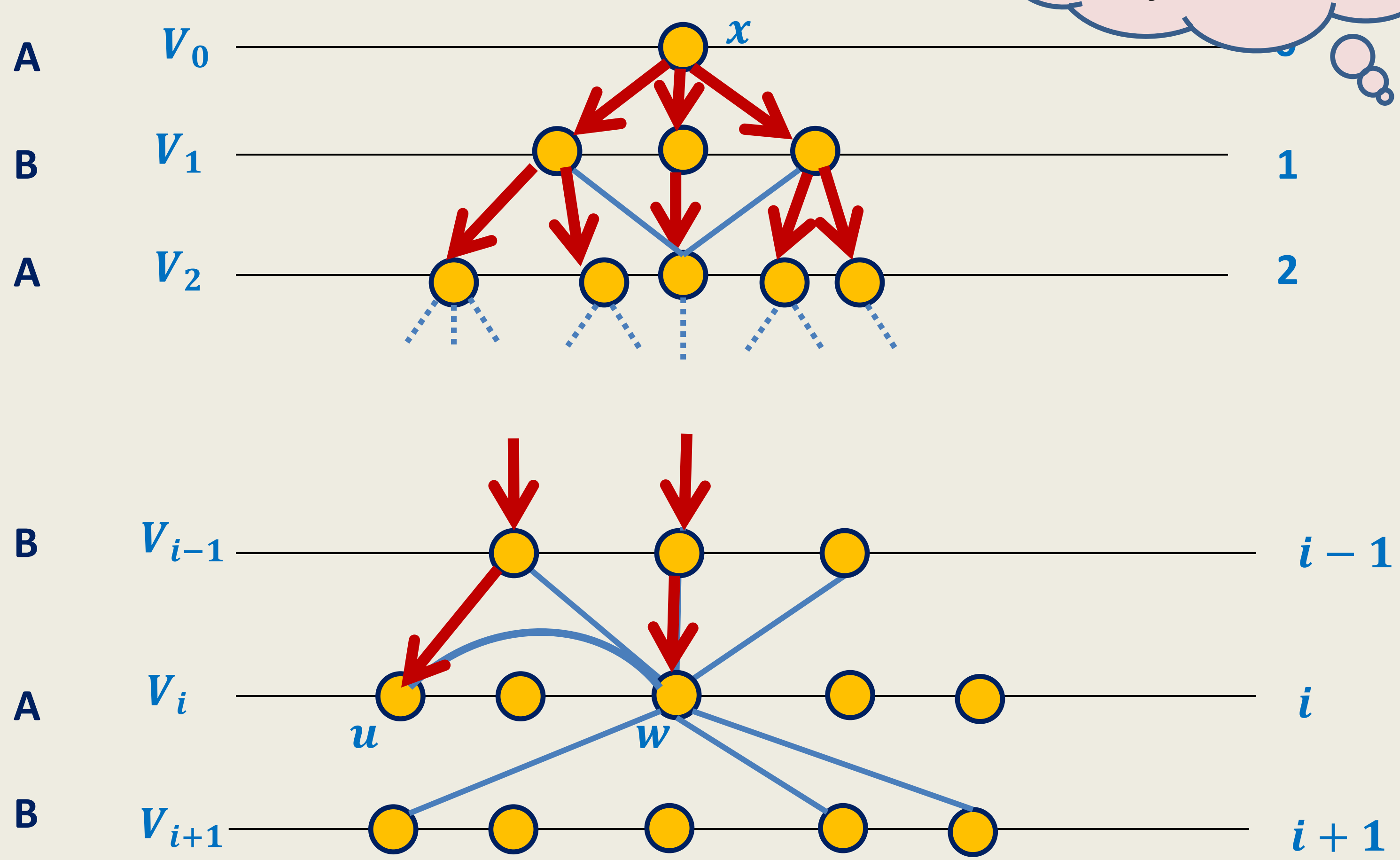
Question:

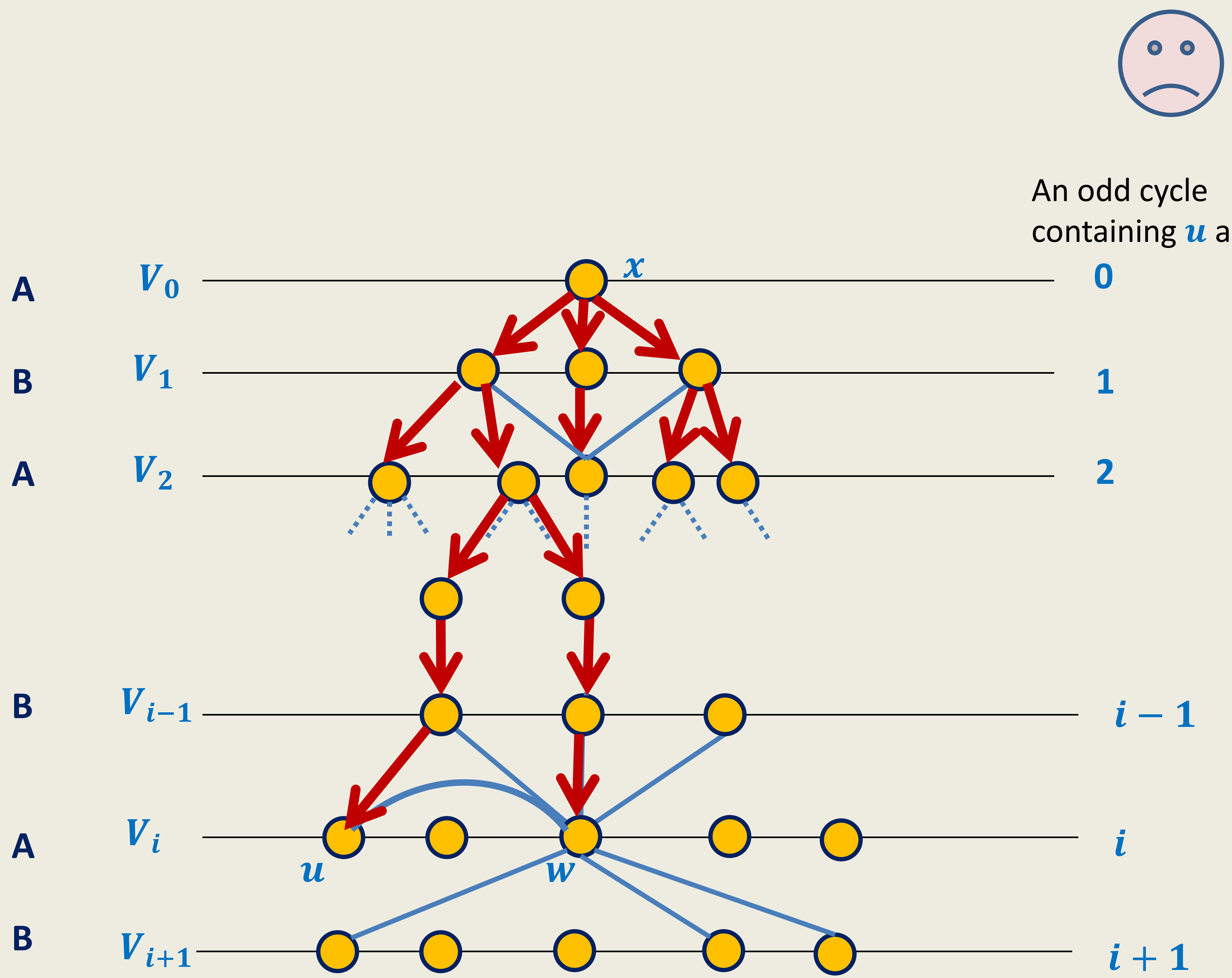
What if there is an edge with both end points at same level ?

What if there is an edge
with both end points at <u>same</u> level ?

Keep following parent pointer from u and w simultaneously until we reach a common ancestor. What do we get ?

Can you spot an **odd length cycle** here ?




An odd cycle
containing u and w
A
B
A
B
A
B
V_0
V_1
V_2
V_{i-1}
V_i
V_{i+1}
x
u
w
0
1
2
$i-1$
i
$i+1$

Observation:

If there is **any** non-tree edge with both endpoints at the same level

then the graph has **an odd length cycle**.

Hence the graph is **not** bipartite.

Theorem:

There is an **O**($\boldsymbol{n} + \boldsymbol{m}$) time algorithm to determine
if a given graph is **bipartite**.

In the next 3 lectures, we are going to discus **Depth First Traversal**

: the most nontrivial, elegant graph traversal technique with wide applications.