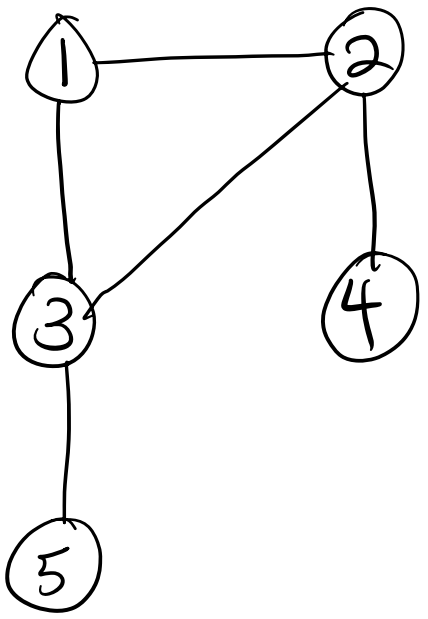
Graphs, continued.

Recall: A graph can be represented by a

Adjacency Matrix



	1	2	3	4	5
1		1	1	0	0
2	1		1	1	0
3	1	1		0	1
4	0	1	0		0
5	0	0	1	0	

↖

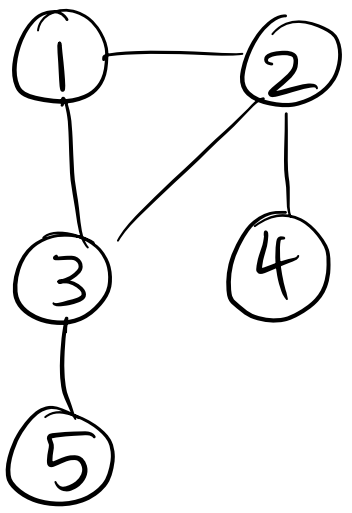
appropriate value for the diagonal may be application specific.

Pro: Fast look-up for "Is 3 adjacent to 4?"

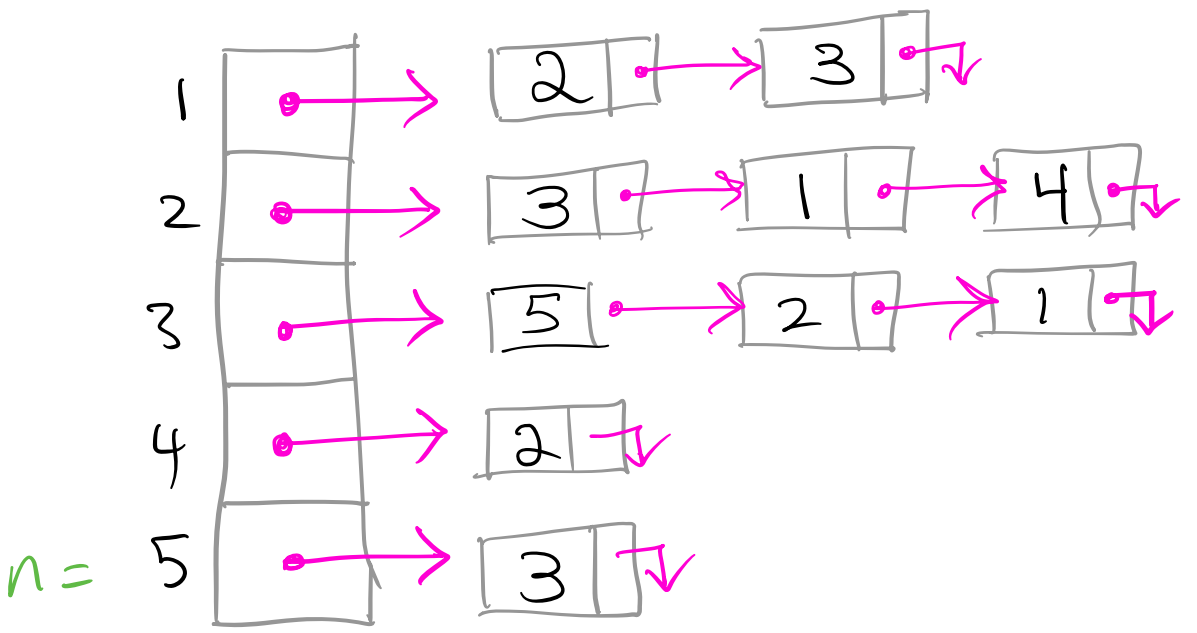
Con: - $\Theta(n^2)$ space even if m is $\Theta(n)$

- $\Theta(n)$ to process all neighbours of a vertex v even if v has no neighbours

Adjacency List Representation.



Logically, we can organize it as follows:



Size is $\Theta(n+m)$

$2m$ of these records

Add an edge $(4,5)$:

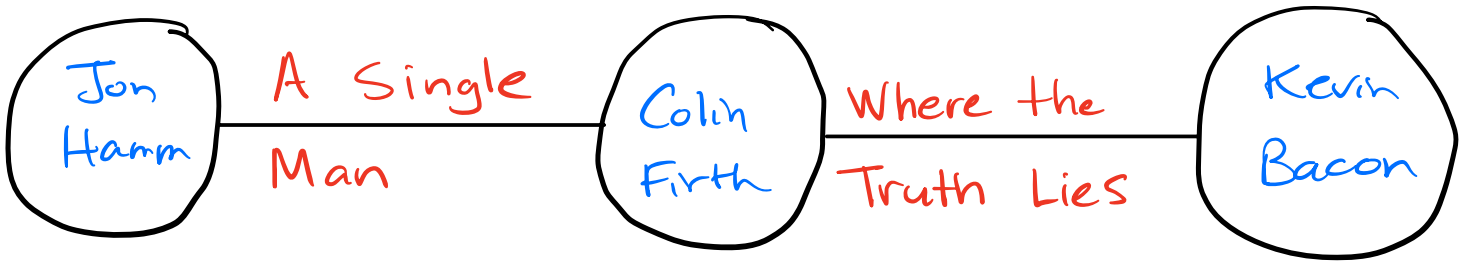
- insert [5] into (head of) 4's list
- insert [4] into (head of) 5's list

Definition: The **degree** of a vertex is the number of edges **incident** with the vertex

eg. vertex 1 has degree 2.

Also, u is adjacent to vertex v if $(u,v) \in E$.

Important Application: Bacon number.

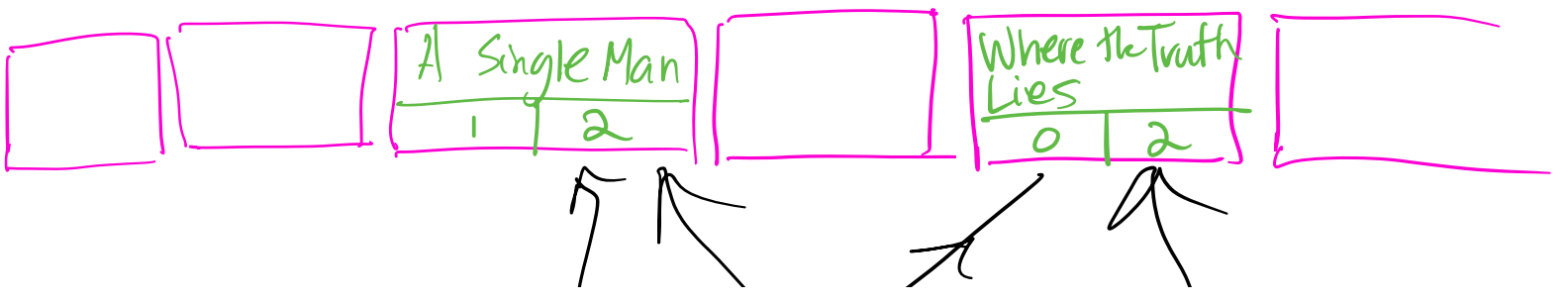


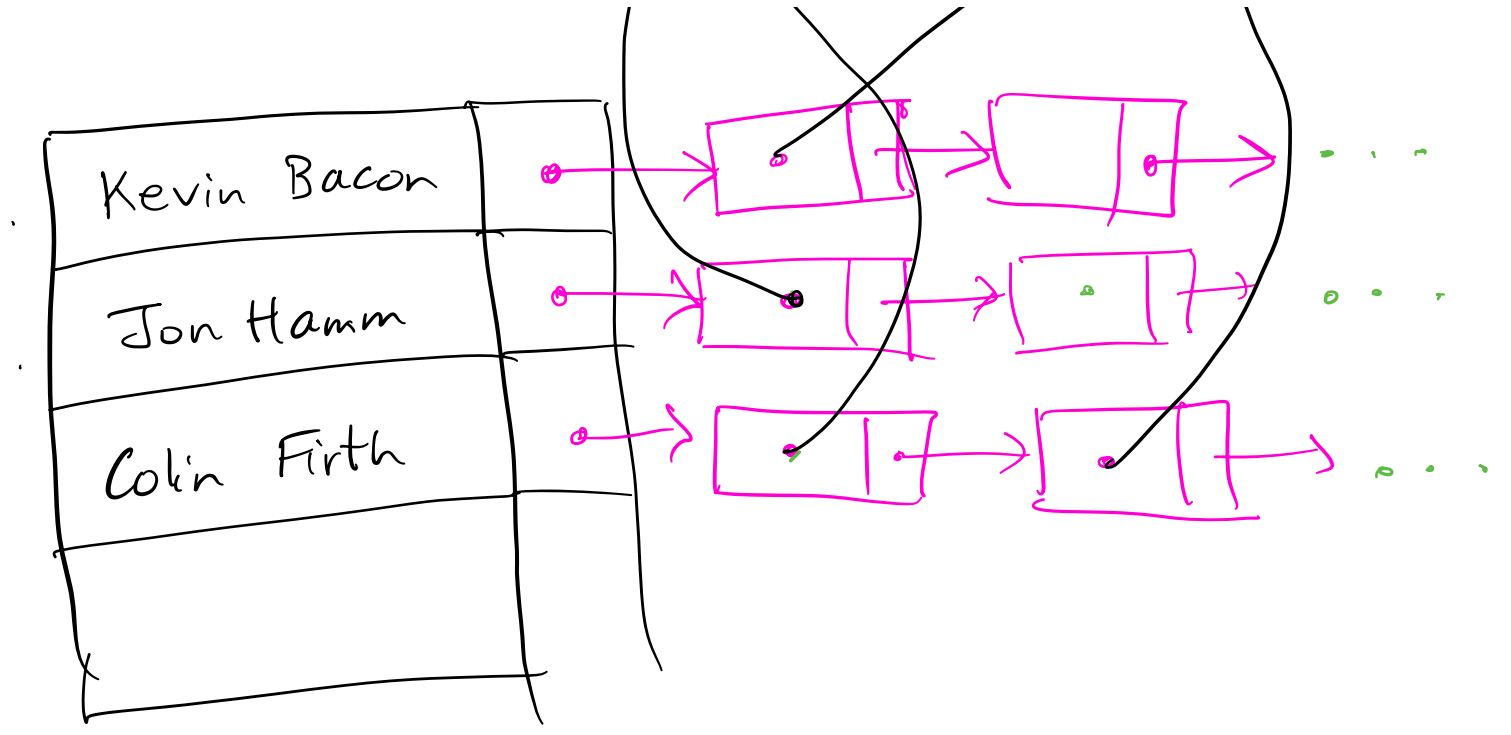
Bacon number: length of shortest path to Kevin Bacon in the "worked with" graph.

path length = # edges

Note that a vertex can have extra info stored "at" the vertex

Also, an edge may have data associated with it.





Blazingly Fast Graph Algs $O(n+m)$

Graph Search (G, s)

Input: An undirected or directed graph $G=(V,E)$ and a start vertex $s \in V$

Goal: Identify the vertices of G reachable from s.

"reachable" means $\exists$ a path from s

Generic Search

Input: graph $G = (V, E)$, vertex $s \in V$

Postcondition: vertex $u \in V$ is reachable from s iff marked[u] = true

marked[s] = true

while $\exists$ an edge (v, w) with // underdetermined

marked[v] == true and

marked[w] == false,

- Choose such an edge (v, w)

- marked[w] = true.

Claim: (Correctness of Generic Search)

At the conclusion of Generic Search, a vertex v has marked[v] == true iff $\exists$ an $s - v$ path in G.

Proof: We will assume G is undirected, but this proof can be adapted to the directed case.

I. ($\Rightarrow$) BWOC. Suppose Generic Search is run, and let u be the first vertex that is not reachable from s to be marked. Then $\exists$ an edge (w, u) where w was already marked by this time.

∴ w is reachable from s, ie $\exists$ path

$s, v_1, v_2, \ldots, w$.

But then $s, v_1, v_2, \ldots, w, u$ is also a path in G.

Then u is reachable from s in G

$\Rightarrow\Leftarrow$

∴ if u is marked, then u is reachable.

II ($\Leftarrow$) By induction on the distance of a vertex from s.

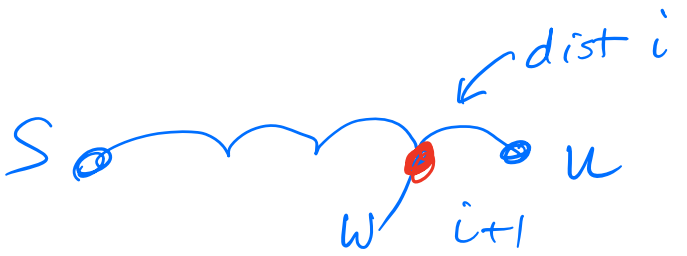
distance = # of edges in a path from s

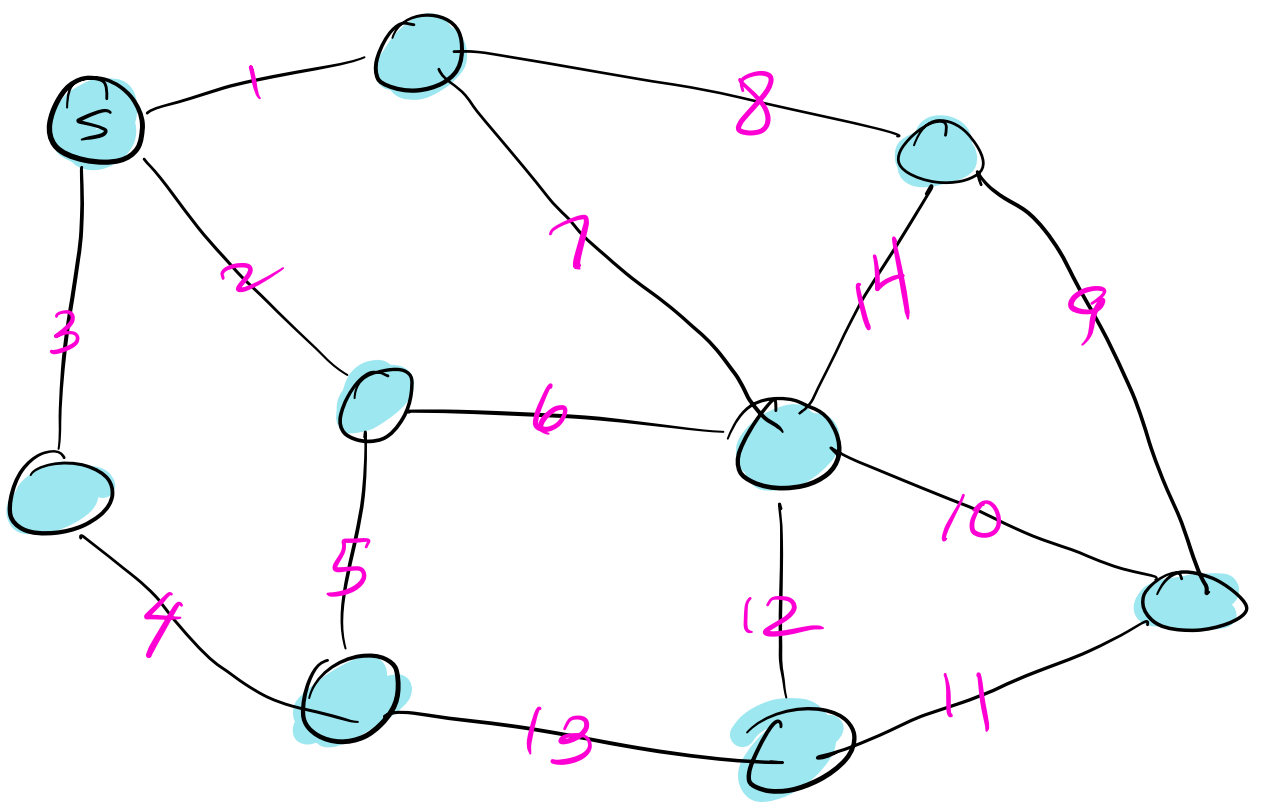
distance $= 0$: s is marked in first step.

$\forall$ true for dist $= i$. > 0

Let u be a vertex at distance $i+1$

Then ∃ edge (w, u) where w is at distance i.
Then w is marked.
Then u will eventually be marked
(algorithm cannot halt while ∃ an edge like w).





Breadth First Search.

Input: graph $G=(V,E)$ as adj-list; vertex S.

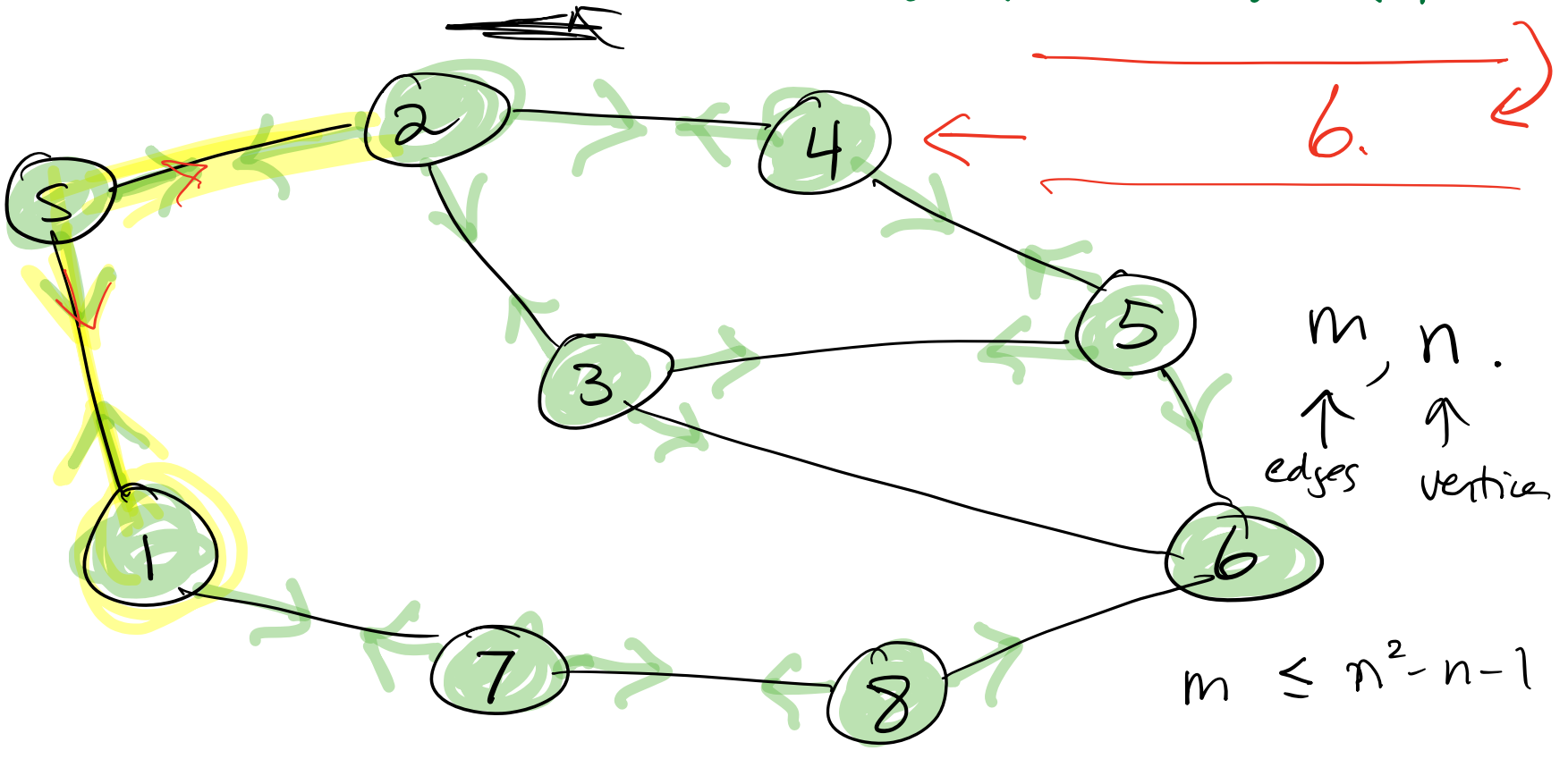
postcondition: marked[v] == true $\Leftrightarrow$ v is reachable from S.

1. marked[s] = true
2. Q = queue data structure, initially containing just S

} $\Theta(1)$.

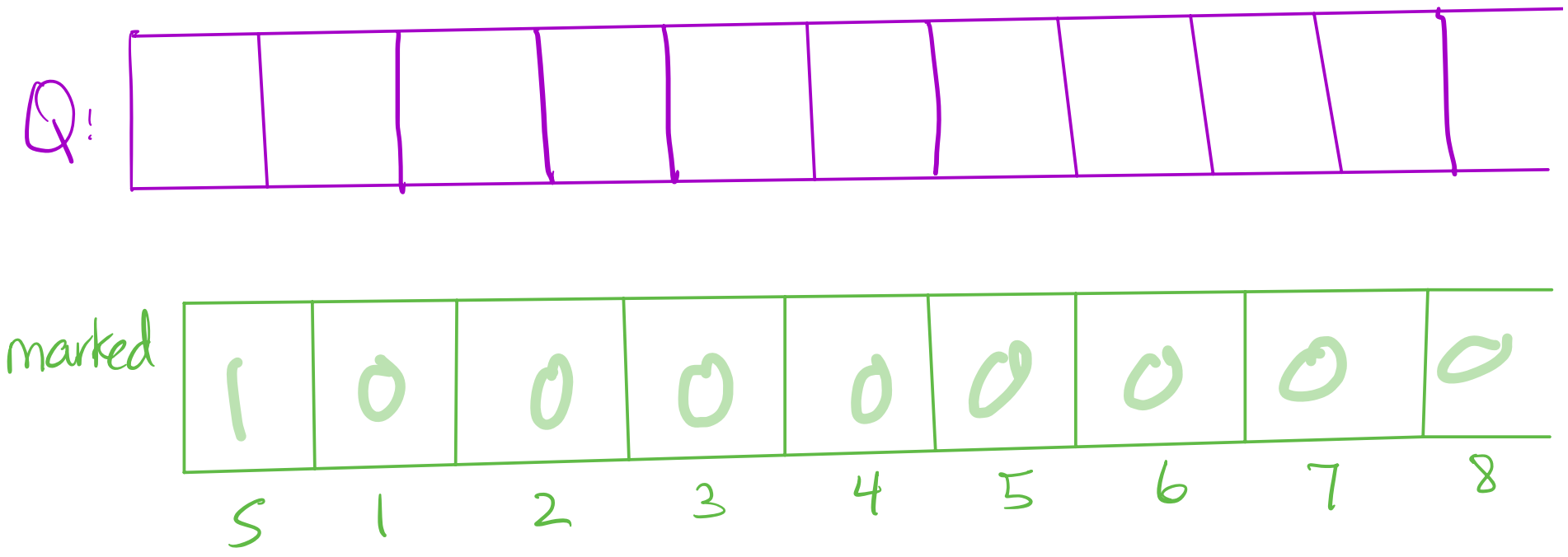
3. while Q is not empty do
4. $\Theta(n')$ remove vertex v from front of Q
5. for each edge (v, w) in v's adj-list do
6. if !marked[w]
7. marked[w] = true
8. add w to end of Q.

Work is $\Theta(2m) = \Theta(m)$



m, n.
↑ edges ↑ vertices

$m \leq n^2 - n - 1$



Running Time of BFS

```
mark s
Q.init(); Q.insert(s)
while !Q.empty()
    v = Q.front();
    Q.pop();
    for (w = front of v's adjlist;
          w ≠ Null; w = next in
          v's adj list)
        if w not marked
            mark w
            Q.insert(w)
```

Correctness of BFS:

- it will halt, as no vertex can be Q-inserted twice
- It is an instantiation of Generic Search, so when it halts, it will be correct.

Running time of BFS(G, s):

$\Theta(m')$ for all the edge-checking.

$+ \Theta(n')$ for all the Q insertions, pop

Where m' = #edges in the connected component containing s

n' = #vertices in C.C. containing s.

Running time is $\Theta(n'+m')$