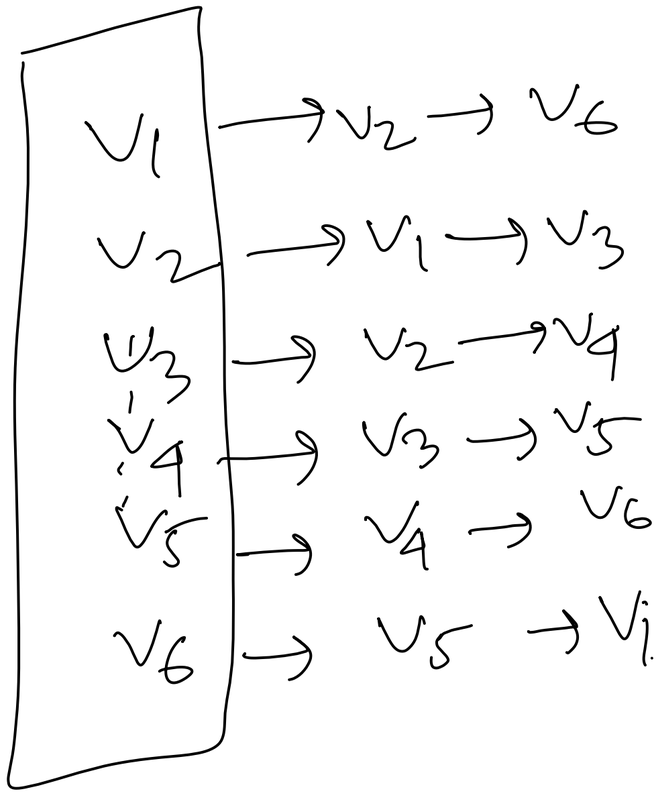
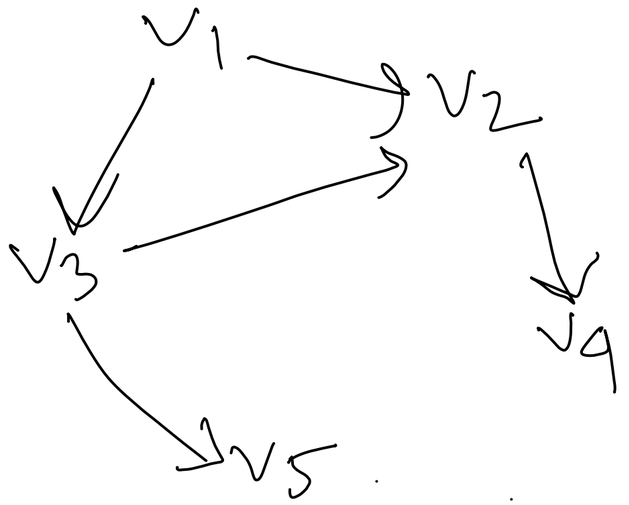




$v_1 \rightarrow v_2 \rightarrow v_3$

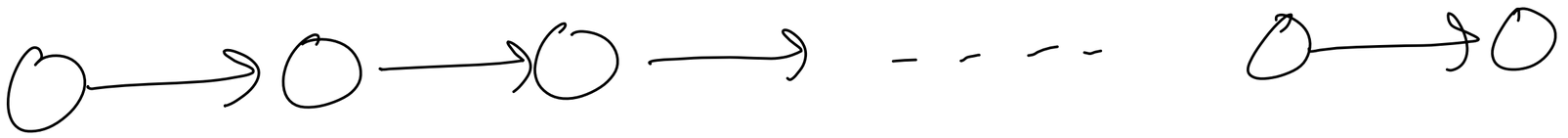
$v_2 \rightarrow v_4$

$v_3 \rightarrow v_5$

v_4 ~~$\rightarrow v_2$~~ (already)

v_5

Linked list



Each list item

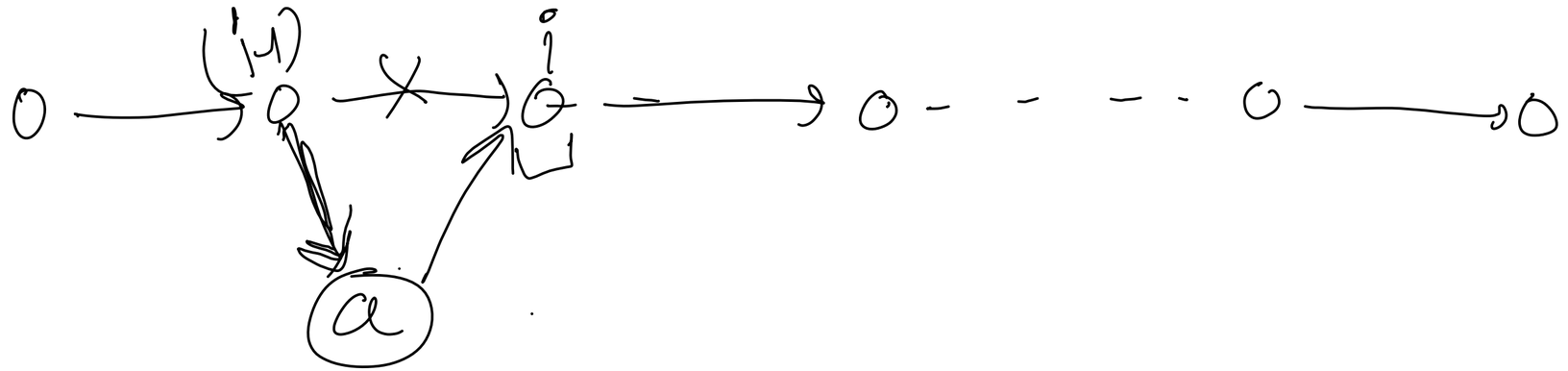
- key
- next

Operations

- Create(a) : creates a singleton list with key a
- Insert(L, i, a) : insert a at i-th pos^n
- Delete(L, i) : remove i-th item from L
- Search(L, a) : yes/no, depending on $a \in L$ or not

```
Procedure Create(a)
  x ← empty linked list item
  x.key ← a
  x.next ← NULL
  return x
```

(:=3



Procedure Insert (L, i, a)

If (i=1)

X ← Create (a)

X.next ← L

return X

temp ← L

while (i > 2) if (temp = NIL) return error.

temp ← temp.next

i - -

endwhile

prev ← temp.next

X ← Create (a)

temp.next ← X

x.next ← prev.

return L.

Delete (L, i)

free ( )

skipped, very similar to merge

```
Procedure Search (L, a)
  temp ← L
  While (temp != NULL)
        if (temp.key = a)  return yes endif
        temp ← temp.next
  end while
  return no.
```

Adj. list vs adj. matrix

Facebook graph:

Adj list storage is much more efficient.

$\Theta(n^2)$, $\Theta(n+m)$ $\quad n =$ billions

sparse graph $\quad$ # edges.

hypothetical example

edge queries lots of $\quad (i, j) \longrightarrow$ yes

$\longrightarrow$ no.

$\Theta(1)$, dense graph. $\quad \Theta(\ell)$, $\ell = |L|$ ← length of vertex linked list

nbrs

Q: How many neighbors v_i has got.

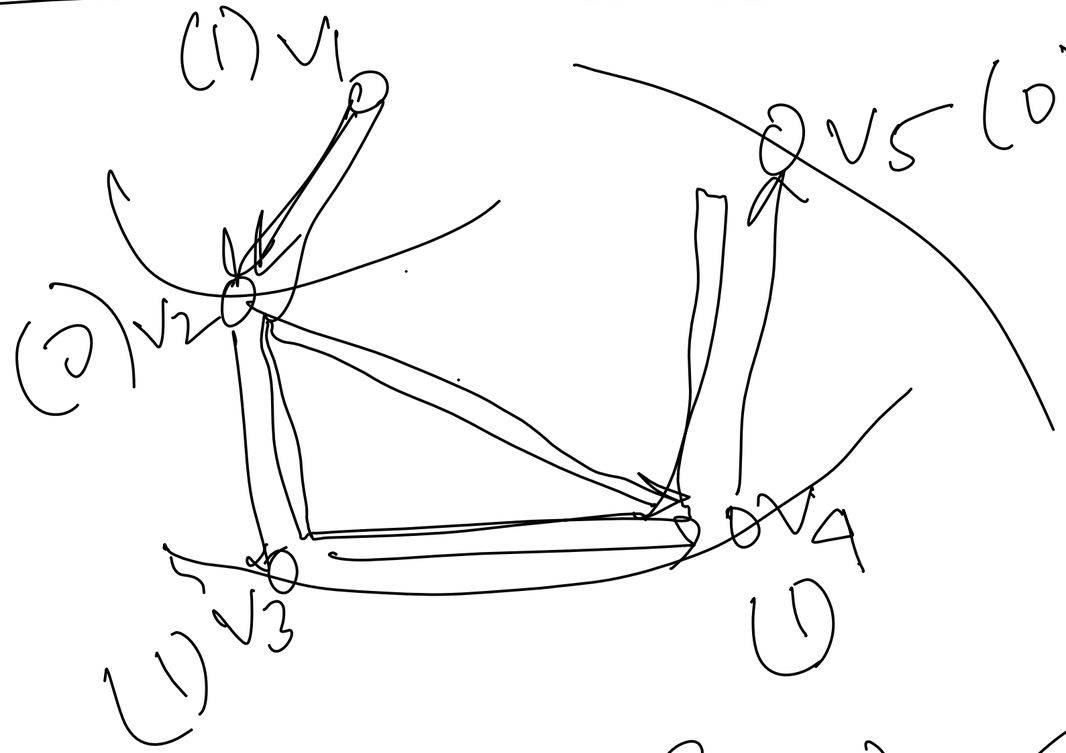
Adj matrix : $\Theta(n)$

Adj list : $\Theta(\ell)$, ℓ = # neighbors of v_i

(Sparse graph)

(i, ?)

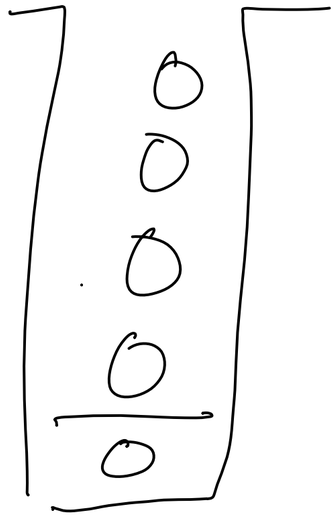
: Graph Search / Exploration :



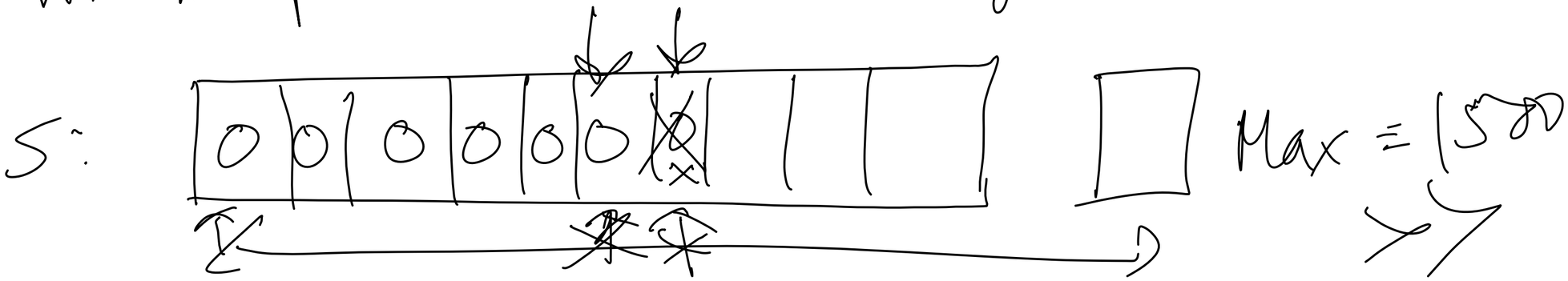
- depth-first-search (DFS) ✓ → Stack
- breadth-first-search (BFS) ✓ → Queue.

Stack Data Structure

- S.top ← index of the top item.
- Push (S, x) ← pushes item x on top of stack S.
- Pop (S) ← removes the top item of S.



We'll implement stack using an array



Initially for an empty stack S.top ← 0

```
Procedure CreateStack ( )
    S ← empty array of size MAX
    S.top ← 0
    return S
```

```
Procedure. Push (S, x)
    S[S.top + 1] ← x
    S.top ++
    if(S.top ++ == MAX) return error.
```

```
Procedure Pop (S)
    if(S.top = 0)
        return error.
    S.top --
    return S[S.top + 1]
```

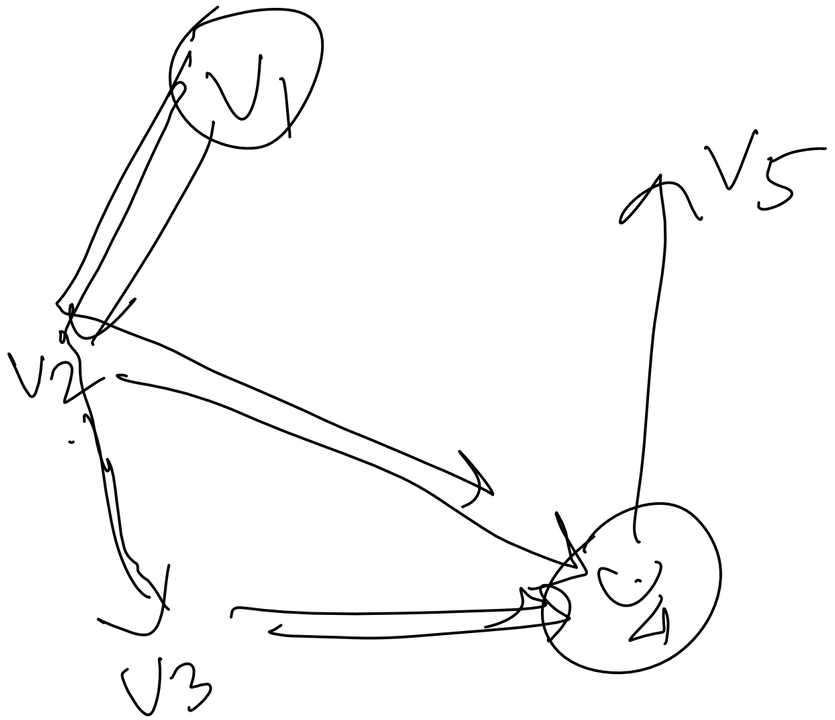
Depth first Search (DFS)

Given are adjacency list, explore the graph.

Q: ~~Uniform~~ Count f(nodes).

Q: reachability: is node v_j reachable from node v_i?

Yes/no.

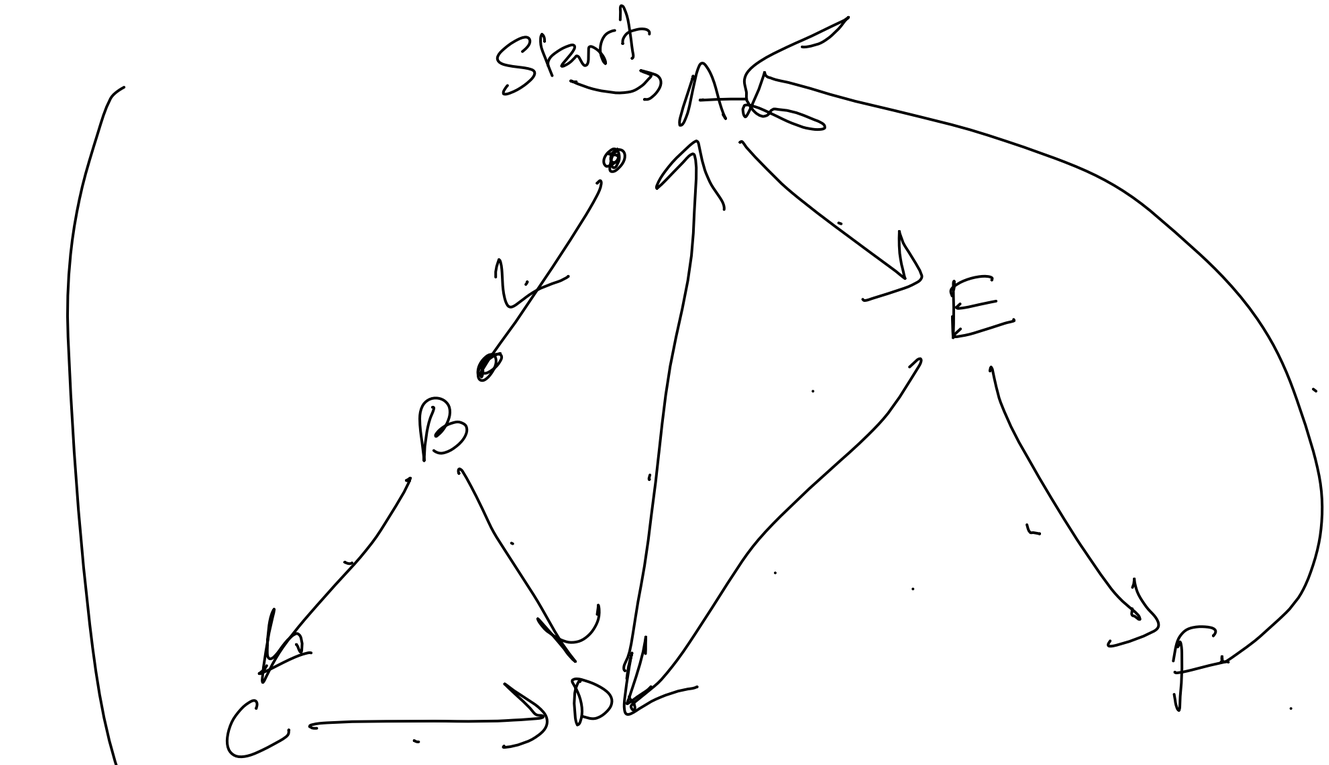


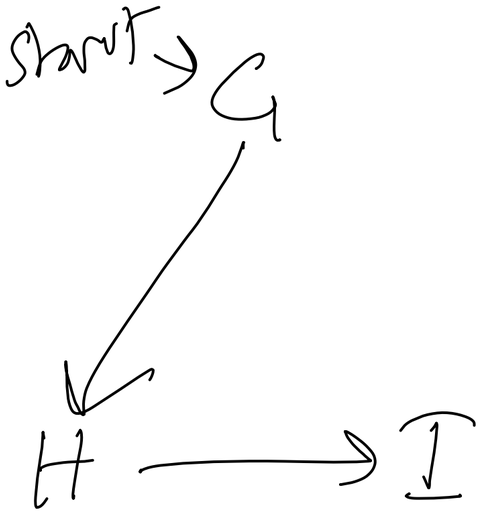
Is V_4 readable from V_1? yes.

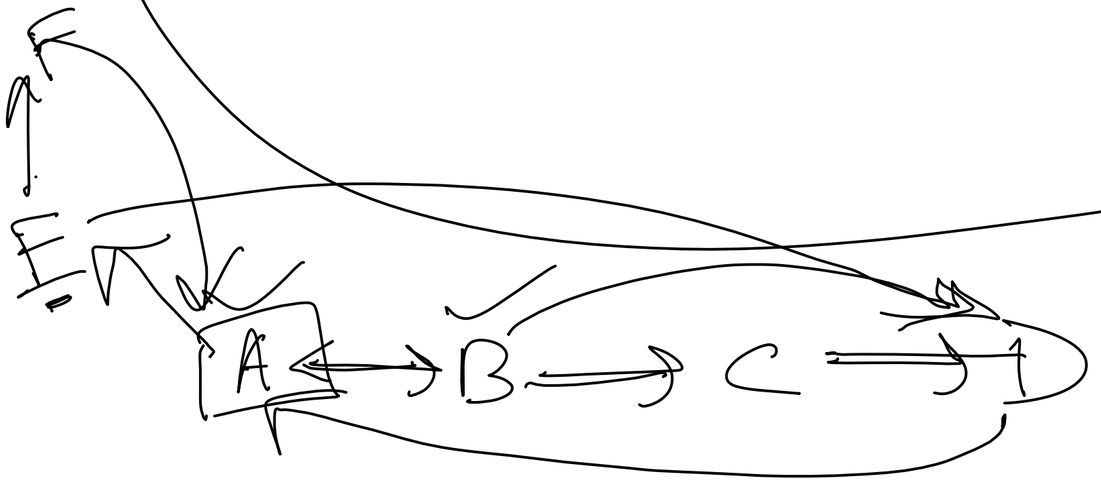
- yes, two paths.

Is V_1 readable from V_5? no.

- ∄ any path.







$n = 9$

$m = 11$

A: | A | B | C | D | E | H | I |

Procedure DFS(G, v), A

(V, E) ← G

V: vertices, E: edges.

v: starting vertex

A is of size n.

A[i] = 1 iff vertex i has been explored.

Stack ← CreateStack( ).

~~Visit for v~~

for each v ∈ V.

A[v] ← 0.

16.08.2024