Subject : 2013

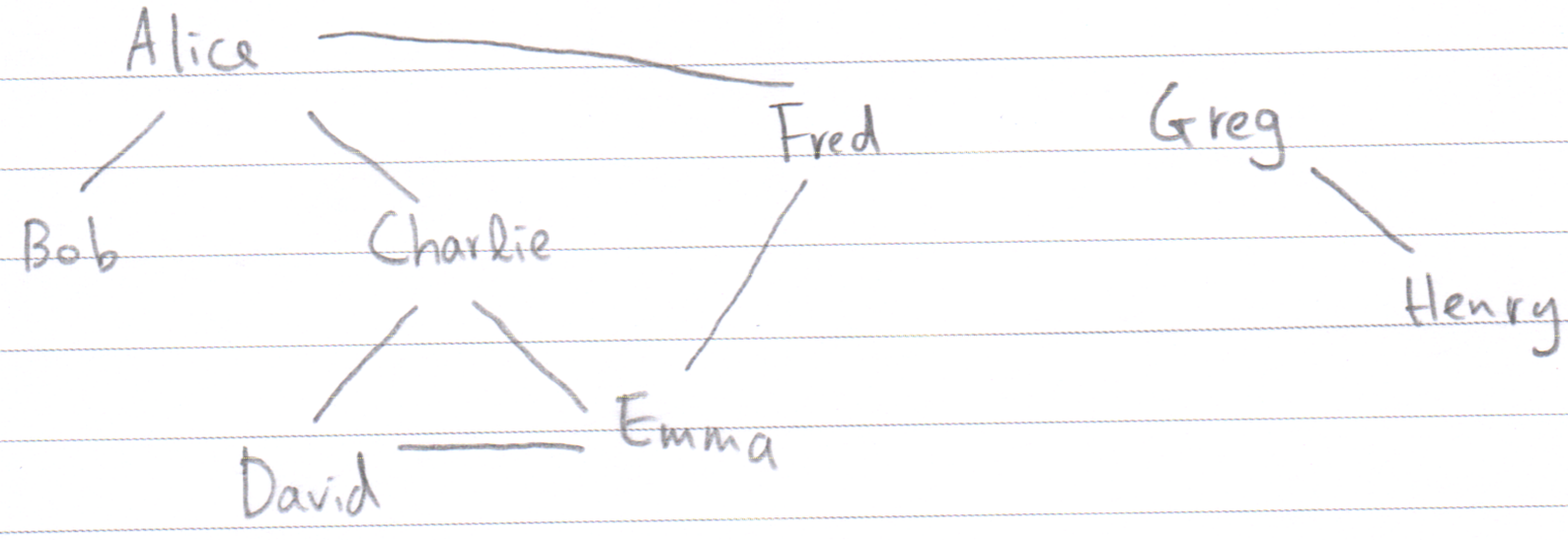
No. : 13-1

Date : ...../...../.....

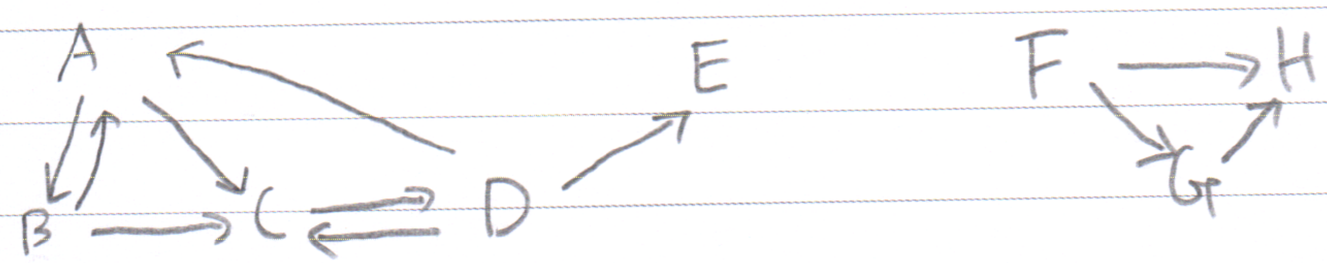
* Graph

"connected access"

① facebook friends : undirected graph



② road in GPS : directed graph



* $G = (V, E)$

V ↓ vertex (node)

E ↓ edge

n (vertex) $\Rightarrow$ $n(n-1)$ directed edges if

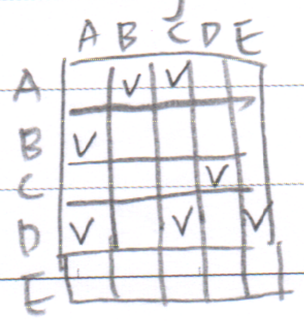
{ no duplicated edges
  no self-loops } called "simple" graph

* connection to other data structures

① tree is a special undirected graph
(or directed hierarchically)

② (simple) graph is essentially a special sparse 2D array

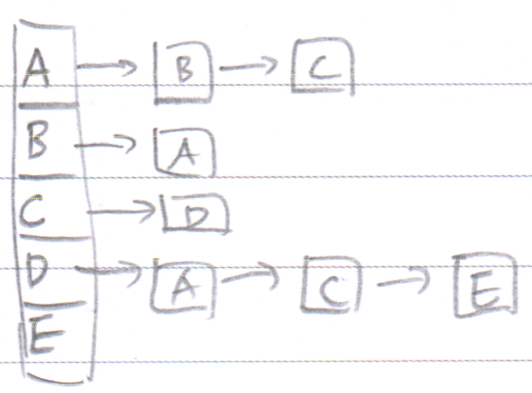
• Adj Matrix representation $\approx$ dense 2D



Double A

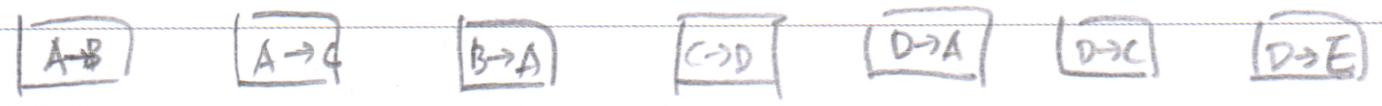
- Adj List representation

  ≈ dense row index + sparse row vectors (by linked list)



- Edge List representation

  ≈ sparse "pair" entries



* Maze Problem

- entry point : one vertex (say A)
- exit point : some other vertex (or maybe no exit and then need to "traverse" whole graph)

* Get-Out-Recursive (G, (A))

function Get-Out-Recursive (G, v)

mark v as visited

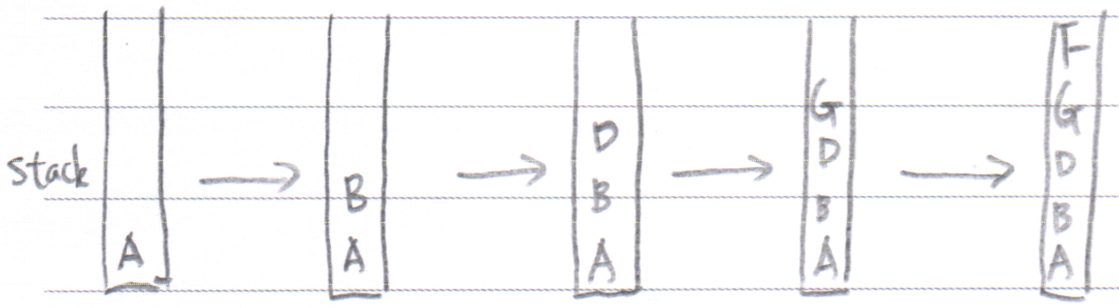
for each unvisited 'u' from 'v' do
(connected)

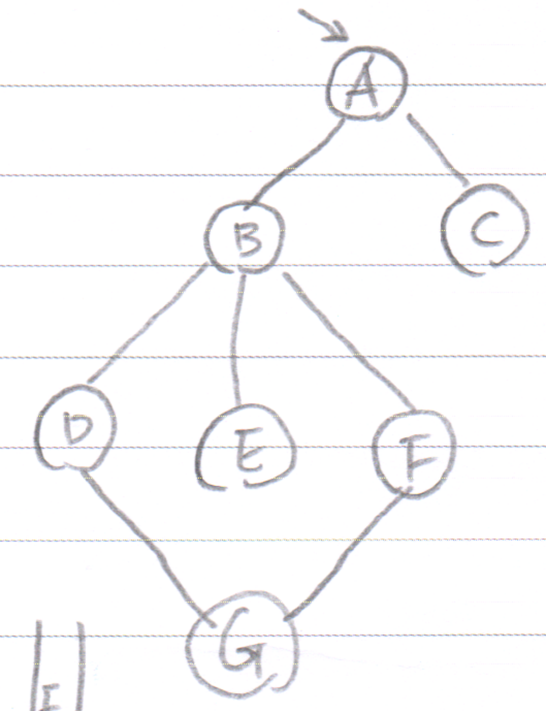
if u is an exit
return TRUE; (backtrack the path)

if Get-Out-Recursive (G, u)
return TRUE;

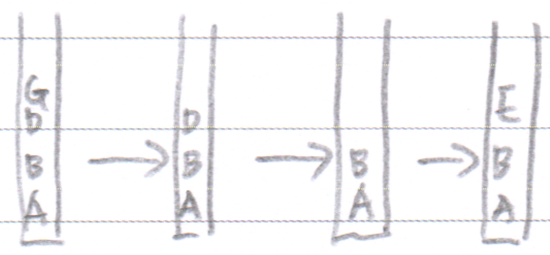
return FALSE;

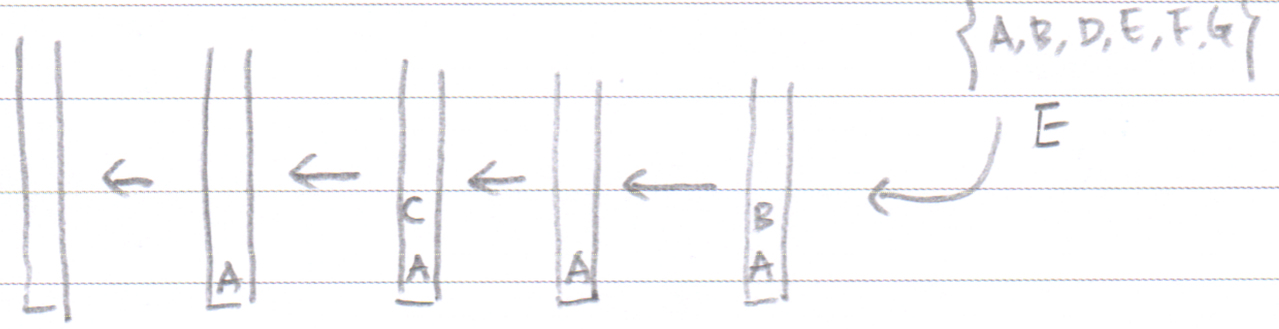
* the stack in the recursive calls
"store the current path being traced"





visited {A} {A,B} {A,B,D} {A,B,D,G} {A,B,D,F,G}
output A B D G F





{A,B,D,E,F,G}
E

{A,B,C,D,E,F,G}
C

depth-first search:
traverse the graph in the "visited" order in Get-Out-Recursive

function DFS(G, Ⓐ)
push Ⓐ to stack
mark Ⓐ as visited (and output)
while stack not empty
v ← stack.top
if there is an unvisited 'u' connected from 'v'
push u to stack
mark u as visited (and output)
else
stack.pop (backtrack)

Subject : 2013

No. : 13-4

Date : ...../...../.....

* an equivalent DFS

use stack to store "to be visited"

function DFS2(G, Ⓐ)

push Ⓐ to stack

while stack not empty

u ← stack.pop

if u is really not visited

mark u as visited (and output)

push all unvisited 'z' connected from 'u' to stack (reversely)

output A B D G F E C

* DFS on tree w/ Ⓐ ==root : ① pre-order traversal

② trace the tree "deeply"

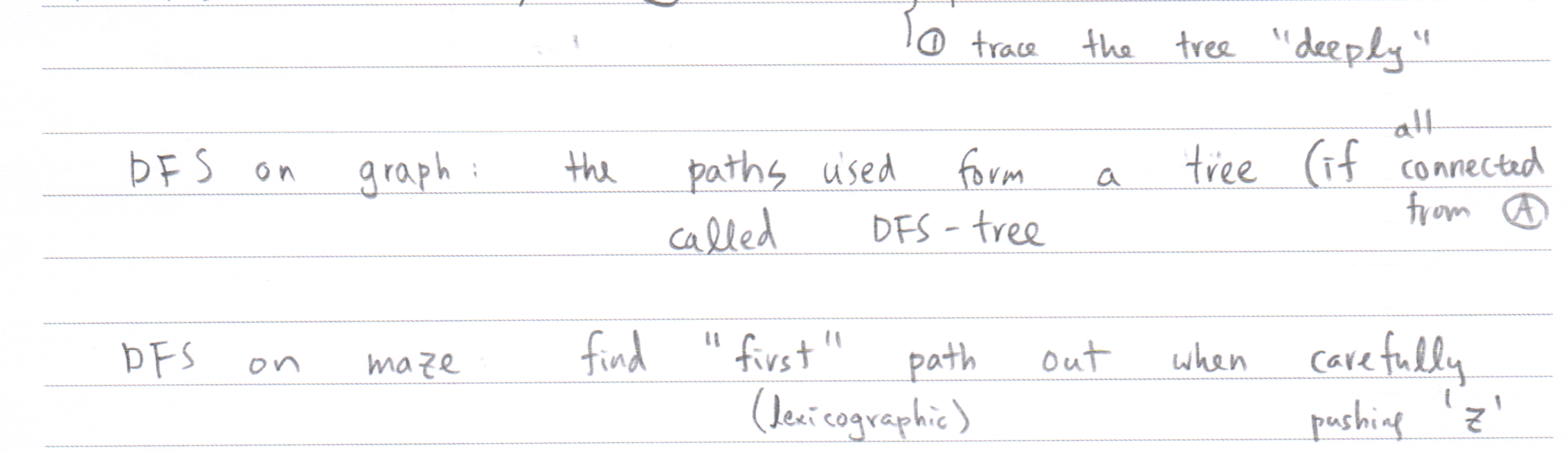
DFS on graph : the paths used form a tree (if all connected from Ⓐ)

called DFS-tree

DFS on maze find "first" path out when carefully pushing 'z' into stack

(lexicographic)

* using queue instead of stack ?



Double A

Subject : 2013

No. : 13-5

Date : ...../...../.....

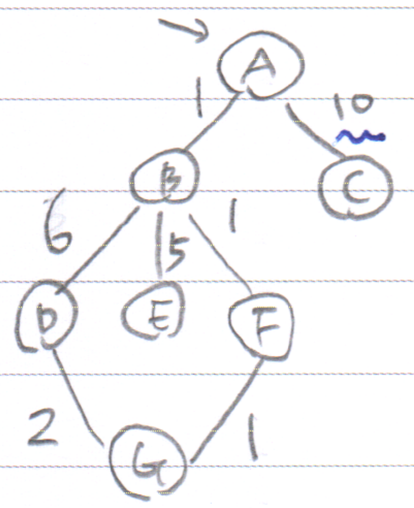
* closest (by # edges) to (A) is outputted first
called Breadth-First Search (BFS)

BFS on tree w/ (A) == root : ⑥ level-wise traversal
① trace the tree "broadly"

BFS on graph : the paths used form a tree (if all connected from (A))
called BFS-tree

BFS on maze : find "closest" path out
(step-wise)

*



← distance
(weight of edge)

what if we want "closest" distance-wise
instead of step-wise?

⇒ shortest path problem (from (A))
source

how? change from queue to priority queue
called Dijkstra's algorithm, works if weight ≥ 0
"pick the closest one in the tovisit pile"