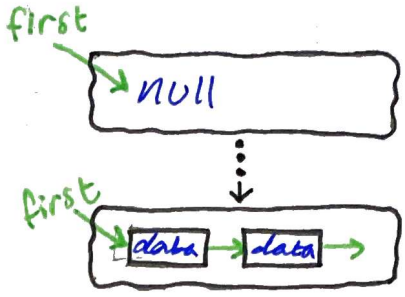
GRAPH

Bag

```
Bag {
    first = null;
    n = 0;
}
```

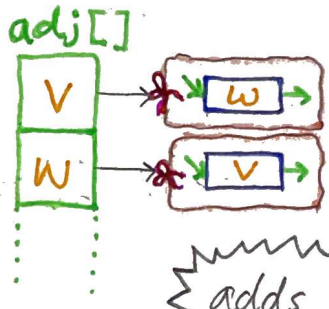


add(int, int)
to front

Graph

```
Graph (int V) {
    this.V = V;
    this.E = 0;
    adj = (Bag<Integer>[]) new Bag [V];
    for (int v=0; v< V; v++)
        adj [v]= new Bag<>();
}
```


```
addEdge(int v, int w) {
    E++;
    adj [v]. add(w);
    adj [w]. add(v);
```



adds to front

GRAPHS

SymbolGraph

delimiter

```
SymbolGraph (File, String) {
   st = new ST<>();
   Scanner scan = new Scanner (filename);
   while (scan.hasNextLine()) {
      String [] a = scan.nextLine(). split (delimiter);
      for (int i=0; i<a.length; i++) {
         if (!st.contains(a[i]))
            st.put(a[i], st.size());
   }
}
```

input:

line: AL FL

line: AL GA

line: AR LA

line: AZ CA

line: AZ NV

line: AZ UT

delimiter

st

AL	FL	key
0	1	value = st.size()

AL	FL	GA
0	1	2

AL	FL	GA	AR	LA
0	1	2	3	4

AL	FL	GA	AR	LA	AZ	CA
0	1	2	3	4	5	6

AL	FL	GA	AR	LA	AZ	CA	NV
0	1	2	3	4	5	6	7

AL	FL	GA	AR	LA	AZ	CA	NV	UT
0	1	2	3	4	5	6	7	8

GRAPHS

... SymbolGraph

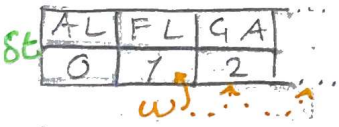
... SymbolGraph (...) {

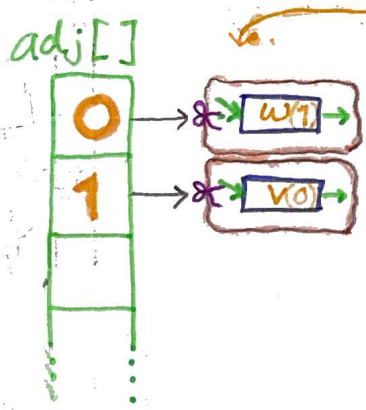
```
Keys = new String [st.size()];
for (String name : st.keys())   ← queue AL,FL,GA...
   Keys [st.get(name)] = name;
```

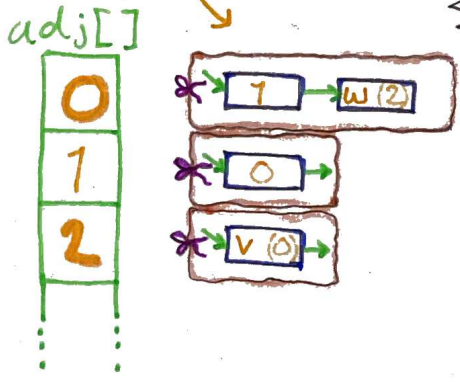
Keys []

a[] consists of 1 line at the time only

```
graph = new Graph(st.size());
scan = new Scanner (filename);
While (scan.hasNextLine()) {
   String[] a = scan.nextLine().split (delimiter);
   int v = st.get( a[0] );
   for (int i = 1; i < a.length; i++)
      int w = st.get (a[i]);
      graph.addEdge (v,w);
```







adds to front

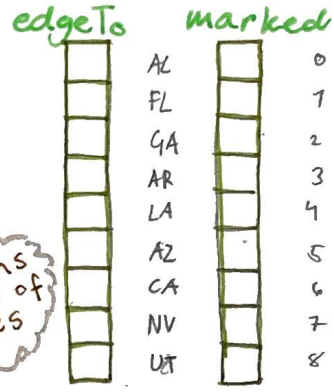
GRAPHS

DepthFirstPaths

```
DepthFirstPaths (Graph , int) {
    this.s = s;
    edgeTo = new int [G.V()];
    marked = new boolean [G.V()];
    validateVertex(s);
    dfs(G,s);
}
```

checks if vertex s is in Graph

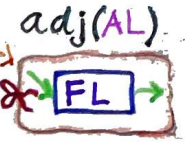
returns number of vertices

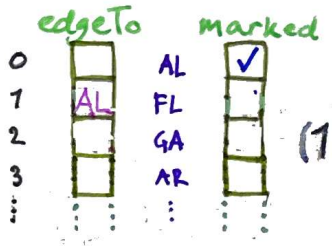



```
dfs (Graph, int) {
    marked [v] = true;
    for (int w : G.adj(v)) {
        if(!marked [w]) {
            edgeTo [w] = v;
            dfs(G,w);
        }
    }
}
```

returns Bag<> of adjacent vertices

until every vertex in some G.adj(v) is true



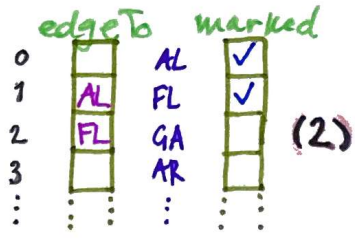



```
pathTo (int) {          S=AL, T=GA
    for(int x = T; x != s; x= edgeTo[x]){
        push(x), x = T
                                    2
        push(x), x = edgeTo[x]
                                    2
    }
    return AL FL GA     push(S)
```

creates and pushes to Stack<>()

GA

FL GA



GRAPHS

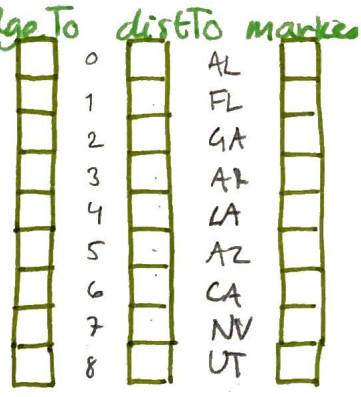
Graph
s AL ↔ FL
FL ↔ GA
AR ↔ LA
AZ ↔ CA
AZ ↔ NV
AZ ↔ UT

BreadthFirstPaths

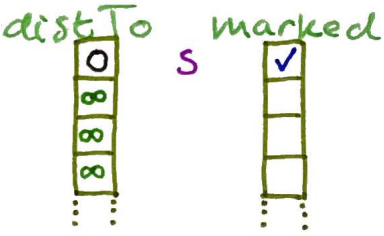
```
BreadthFirstPaths (Graph , int) {
    marked = new boolean [G.V()];
    distTo = new int [G.V()];
    edgeTo = new int [G.V()];
    validateVertex (s);
    bfs(G,s);
}
```

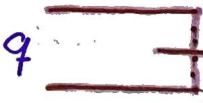
checks if vertex s is in Graph

returns number of vertices

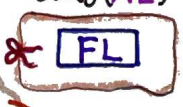



```
bfs (Graph, int) {
    Queue<Integer> q = new Queue<>();
    for (int v=0; v< G.V() ; v++)
        distTo [v] = INFINITY;
    marked [s] = true;
    distTo [s] = 0;
    q.enqueue(s);
    while (!q.isEmpty()) {
        int v = q.dequeue();
        for (int w : G.adj(v)) {
            if (!marked[w])
                edgeTo[w] = v;
                distTo[w] = distTo[v] +1;
                marked[w] = true ;
                q.enqueue(w);
    }
}
```



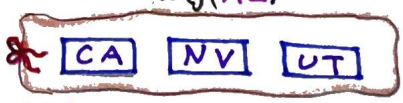


ex.1 adj(AL)

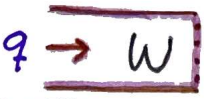


returns Bag<>() of adjacent vertices

ex.2 adj(AZ)

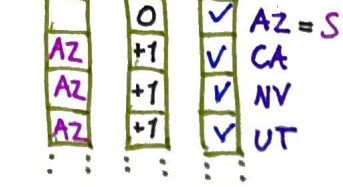


ex.1



queue will consist of every adjacent vertex

ex.2



GRAPHS

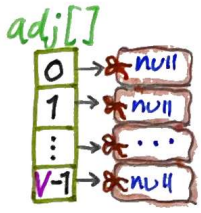
Digraph

```
Digraph ( int) {
    this.V = V;
    this.E = 0;

    indegree = new int [V]

    adj = (Bag<Integer> [ ]) new Bag [V];

    for (int v=0; v< V; v++)
        adj [v] = new Bag<>();
```



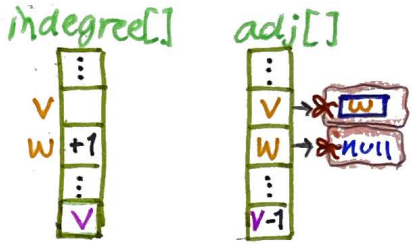

```
addEdge (int, int) {
```

vertex v is validated
vertex w is validated

```
    adj [v]. add (w);

    indegree[w] ++;

    E ++;
}
```



add() adds to front

GRAPHS

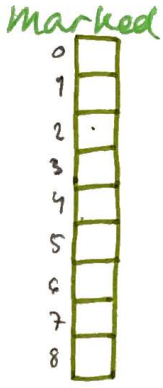
Digraph
s AL → FL
FL → GA
AR → LA
AZ → CA
AZ → NV
AZ → UT

Directed DFS

```
DirectedDFS (Digraph, int) {
    marked = new boolean [G.V()];
    validateVertex (s);
    dfs (G,s);
}
```

checks if vertex s is in Digraph

returns number of vertices

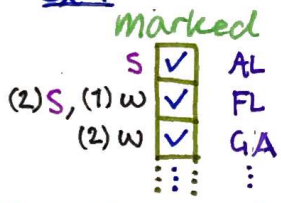



```
dfs (Digraph, int) {
    count++;
    marked[v] = true;
    for (int w : G.adj(v))
        if (!marked[w])
            dfs (G,w);
}
```

returns Bag<>() of adjacent vertices

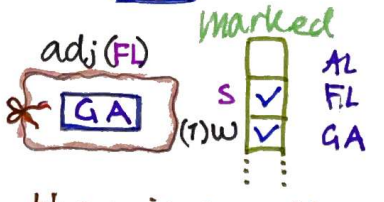
ex
adj(AL)
FL

ex 1



there is a path between : AL → FL
FL → GA
AL → GA

ex 2



there is a path between : FL → GA