Graphs

sequence/linear (1 to 1)

first   ith   last

hierarchical
(1 to many)

graph (many to many)

1

Graph Representations

◆ For graphs to be computationally useful, they have to be conveniently represented in programs

◆ There are two computer representations of graphs:

- Adjacency matrix representation
- Adjacency lists representation

2

Adjacency Matrix Representation

- ◆ Assume V = {1, 2, ..., n}
- ◆ An *adjacency matrix* represents the graph as a n x n matrix A:
  - A[i, j] = 1 if edge (i, j) ∈ E   (or weight of edge)
              = 0 if edge (i, j) ∉ E

A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

3

Adjacency Matrix Representation

- ◆ Assume V = {1, 2, ..., n}
- ◆ An *adjacency matrix* represents the graph as a n x n matrix A:
  - A[i, j] = 1 if edge (i, j) ∈ E   (or weight of edge)
              = 0 if edge (i, j) ∉ E

A	1	2	3	4
1	0	5	7	0
2	0	0	3	0
3	0	0	0	0
4	0	0	2	0

4

2

Adjacency Matrix Representation

Time: O(1)

dense ✓

Undirected Graph

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

n × n
Build: O(n²)
Space: O(n²)

(3,5) ?

very good

Adjacency time = O(1)

sparse   O(n)

Directed Graph

```
          1 2 3 4 5 6
        1 0 1 0 1 0 0
        2 0 0 0 0 1 0
        3 0 0 0 0 1 1
        4 0 1 0 0 0 0
        5 0 0 0 1 0 0
        6 0 0 0 0 0 1
```

Space = O(n²)
Time to build = O(n²)

5

Adjacency Matrix Representation

Undirected Graph

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

Directed Graph

```
      1 2 3 4 5 6
    1 0 1 0 1 0 0
    2 0 0 0 0 1 0
    3 0 0 0 0 1 1
    4 0 1 0 0 0 0
    5 0 0 0 1 0 0
    6 0 0 0 0 0 1
```

The adjacency matrix for an undirected graph is symmetric;
the adjacency matrix for a digraph need not be symmetric

6

3

Adjacency Matrix Representation

A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

◆ Pros:
- Simple to implement
- Easy and fast to tell if a pair (i, j) is an edge: simply check if A[i, j] is 1 or 0   *n = 10,000*
- Can be very efficient for small graphs

good for dense graph, but it is not good for large and space graph.

◆ Cons:
- No matter how few edges the graph has, the matrix takes $O(n^2)$ in memory, *also to build an adjacency matrix, it needs $O(n^2)$ time.*

7

$2+4+2+3+3 = \boxed{14} = 2 \times 7 = 2m$ n = no. of vertices
m = no. of edges

Adjacency Lists Representation

◆ A graph is represented by a one-dimensional array L of linked lists, where
- L[i] is the linked list containing all the nodes adjacent to node i.

L[i] ← Array of pointers (heads)

1 → 2 → 5   d(1) = 2
2 → 1 → 5 → 4 → 3   d(2) = 4
3 → 2 → 4   d(3) = 2
u, 4 → 3 → 2 → 5   d(4) = 3
5 → 1 → 2 → 4   d(5) = 3

Undirected Graph

Space
No. of Heads = n
No. of nodes = d(1) + d(2) + ... + d(n)
= Σ d(v)
= 2m
Total space = O(n+m)

Directed Graph

(1,5)? yes   (1,4)? No
(2,5)? yes   (u,v)?
*   Time = O(d(u))*
(v,u), Time = O(d(v))

degree of a vertex v
d(v)
d(2) = 4

8

4

Slide 1

$2+4+2+3+3 = \boxed{14} = 2 \times 7 = 2m$ | $n =$ no. of vertices
$m =$ no. of edges

Adjacency Lists Representation

♦ A graph is represented by a one-dimensional array L of linked lists, where
- L[i] is the linked list containing all the nodes adjacent to node i.

Adj. Lists

Total space $= O(n+m)$

Dense graph : space $= O(n + n^2) = O(n^2)$
$m = O(n^2)$

Sparse graph : space $= O(n+n) = O(n)$
$m = O(n)$

Undirected Graph

Space
Dense — matrix : $O(n^2)$
        List : $O(n^2)$

Sparse — matrix : $O(n^2)$
         List : $O(n)$
$= n+2m = O(n+m)$

∴ Dense graph $= O(n^2)$
  Sparse graph $= O(n^2)$

Time — Dense — Matrix : $O(n^2)$
       Sparse : $O(n+m)$

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Slide 2

Adjacency Lists Representation

♦ A graph is represented by a one-dimensional array L of linked lists, where
- L[i] is the linked list containing all the nodes adjacent to node i.

Undirected Graph

1	→ 2 → 5 /
2	→ 1 → 5 → 3 → 4 /
3	→ 2 → 4 /
4	→ 2 → 5 → 3 /
5	→ 4 → 1 → 2 /

Directed Graph

1	→ 2 → 4 /
2	→ 5 /
3	→ 6 → 5 /
4	→ 2 /
5	→ 4 /
6	→ 6 /

10

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Adjacency Lists Representation

◆ Pros:
  ▪ Saves on space (memory): the representation takes O(|V|+|E|) memory.
  ▪ Good for large, sparse graphs (e.g., planar maps)

◆ Cons:
  ▪ It can take up to O(n) time to determine if a pair of nodes (i, j) is an edge: one would have to search the linked list L[i], which takes time proportional to the length of L[i].

11

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Asymptotic Performance

Assumptions: (1) n vertices, m edges, (2) simple graph

Operations	Adjacency List	Adjacency Matrix
Space	$O(n + m)$	$O(n^2)$
incidentEdges(v)	$O(\deg(v))$	$O(n)$
areAdjacent(v, w)	$O(\min(\deg(v), \deg(w)))$	$O(1)$
insertVertex(v)	$O(1)$	$O(n^2)$
insertEdge(e)	$O(1)$	$O(1)$
removeVertex(v)	$O(\deg(v))$	$O(n^2)$
removeEdge(e)	$O(1)$	$O(1)$

12

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET