MCS-375: Algorithms: Analysis and Design Handout #GF2
San Skulrattanakulchai
Gustavus Adolphus College Nov 4, 2014

Data Structures for Graphs & Digraphs

CLRS: Ch 22.1

Adjacency matrices

A (di)graph G can be represented in a computer by an $n \times n$ *adjacency* matrix $A[1..n, 1..n]$. We name the vertices $1, 2, \ldots, n$. Entry $A[i, j] = 1$ if vertex i is adjacent to vertex j; it equals 0 otherwise.

[Example picture here]

Adjacency matrices have size $\Theta(n^2)$. We can store each entry using 1 bit, but space still grows as n^2.

Problem Calculate the outdegree of each vertex in a digraph. The digraph is represented by an adjacency matrix $A[1..n, 1..n]$.

Solution

```
/* program sets $d[v]$ to the outdegree of $v$, for all vertex $v$ */
for $v \leftarrow 1$ to $n$ do {
    $d[v] \leftarrow 0$
    for $w \leftarrow 1$ to $n$ do
        $d[v] \leftarrow d[v] + A[v, w]$
}
```

Adjacency lists

A (di)graph G can also be represented using adjacency lists.

An *adjacency list* of a vertex v is a list of all vertices w with $\{v, w\}$ ((v, w) if digraph) as an edge.

An *adjacency list representation* for a (di)graph consists of an adjacency list for every vertex.

[Example picture here]

The adjacency list representation uses space $\Theta(m+n)$. Each edge of a digraph appears once on the adjacency lists. Each edge of an undirected graph appears twice on the adjacency lists. An edge may be represented as a `Node` object with a `vertex` field and a `nextNode` field.

An efficient way to implement adjacency lists representation is to use 2 parallel arrays `LINK` & `VERTEX`. They give pointer and vertex information, respectively.

For $1 \le i \le n$, `LINK`$[i]$ is the head of i's adjacency list.

For $n+1 \le i$, `LINK`$[i]$ and `VERTEX`$[i]$. form a node on an adjacency list.

  `LINK`$[i]$ points to next node, `VERTEX`$[i]$ gives the vertex.

  `LINK`$[i] = 0$ if the node at i is the last one on its adjacency list.

For digraphs $i \le m+n$; for undirected graphs $i \le 2m+n$.

Problem Calculate the outdegree of each vertex in a digraph. Assume the digraph is given by an adjacency list representation.

Solution

```
/* program sets $d[v]$ to the outdegree of $v$, for all vertex $v$ */
for $v \leftarrow 1$ to $n$ do {
    $d[v] \leftarrow 0$
    $i \leftarrow$ LINK$[v]$
    while $i \neq 0$ do {
        $d[v] \leftarrow d[v] + 1$
        $i \leftarrow$ LINK$[i]$
    }
}
```

This code calculates all outdegrees in time $\Theta(m+n)$ because the for loop iterates n times and the body of the while loop is executed once for each edge, and each iteration takes time $O(1)$.

A graph algorithm *runs in linear time* if the time bound is $O(m+n)$. Such an algorithm takes advantage of sparsity since if m is $O(n)$ then the time is $O(n)$.

High-level code for Adjacency list representation

We usally omit the details of walking down the adjacency list. So the high-level code for the above problem is.

```
/* program sets $d[v]$ to the outdegree of $v$, for all vertex $v$ */
for each $v \in V$ do {
    $d[v] \leftarrow 0$
    for each edge $(v, w)$ do
        $d[v] \leftarrow d[v] + 1$
}
```

Exercise. What's wrong with this pseudocode to calculate the indegree of each vertex?

```
for each $v \in V$ do {
    $d[v] \leftarrow 0$
    for each edge $(w, v)$ do
        $d[v] \leftarrow d[v] + 1$
}
```

Notes

1. For some application, each vertex (edge) may have some associated information. We can keep such information with the node representing it.

   This technique of storing extra information at the node can be useful in general even when the application itself does not require it. For example, storing the indegrees and/or outdegrees of every vertex may help in processing the graph.

2. Implementation of adjacency lists by singly-linked lists is appropriate only for *static graphs*, ones that do not change during the run of the algorithm. For *dynamic graphs* whose vertices and/or edges may be added and/or deleted, the lists need to be doubly-linked for efficiency. Moreover, for undirected graphs, each node w appearing on the adjacency list of v must have a field `MATE`$[w]$ that points to the node v appearing on the adjacacy list of w, and vice versa. This is required for implementing edge deletion in $O(1)$ time.

3. Multigraphs and multidigraphs may be represented using adjacency lists by either making each edge correspond to each node on the lists, or lumping all parallel edges together as one node. Which method is appropriate depends on the application.