Linked List

Equivalence Relations, Sparse Matrices & Doubly Linked Lists

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline

1. Equivalence Relations
2. Sparse Matrices Revisted
3. Doubly Linked Lists

Outline

1. Equivalence Relations
2. Sparse Matrices Revisted
3. Doubly Linked Lists

Equivalence Relation

A relation over a set S is said to be an equivalence relation over S iff it is symmertric, reflexive, and transitive over S.

- reflexive: $x \equiv x$ for each $x \in S$.
- symmetric: for $x, y \in S$, if $x \equiv y$, then $y \equiv x$.
- transitive: for x, y, z, if $x \equiv y$ and $y \equiv z$, then $x \equiv z$.

Equivalence Relation

A relation over a set S is said to be an equivalence relation over S iff it is symmertric, reflexive, and transitive over S.

- reflexive: $x \equiv x$ for each $x \in S$.
- symmetric: for $x, y \in S$, if $x \equiv y$, then $y \equiv x$.
- transitive: for x, y, z, if $x \equiv y$ and $y \equiv z$, then $x \equiv z$.

Example

Given $0 \equiv 4$, $3 \equiv 1$, $6 \equiv 10$, $8 \equiv 9$, $7 \equiv 4$, $6 \equiv 8$, $3 \equiv 5$, $2 \equiv 11$, $11 \equiv 1$.

Equivalence Relation

A relation over a set S is said to be an equivalence relation over S iff it is symmertric, reflexive, and transitive over S.

- reflexive: $x \equiv x$ for each $x \in S$.
- symmetric: for $x, y \in S$, if $x \equiv y$, then $y \equiv x$.
- transitive: for x, y, z, if $x \equiv y$ and $y \equiv z$, then $x \equiv z$.

Example

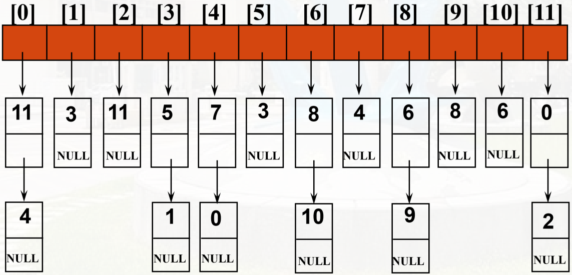
Given $0 \equiv 4$, $3 \equiv 1$, $6 \equiv 10$, $8 \equiv 9$, $7 \equiv 4$, $6 \equiv 8$, $3 \equiv 5$, $2 \equiv 11$, $11 \equiv 1$.
We have three equivalent classes:

$$\{0, 2, 4, 7, 11\}, \{1, 3, 5\}, \{6, 8, 9, 10\}.$$

Lists after Giving Pairs as the Input

$0 \equiv 4, 3 \equiv 1, 6 \equiv 10, 8 \equiv 9, 7 \equiv 4,$
$6 \equiv 8, 3 \equiv 5, 2 \equiv 11, 11 \equiv 0.$

```c
typedef struct node *nodePointer;
typedef struct node {
    int data;
    nodePointer link;
};
```



Outline

Issues for Previous Representation

- When we performed matrix operations such as $+$, $-$, or $*$, the number of **nonzero terms** varied.
- The sequential representation of sparse matrices suffered from the same inadequacies as the similar representation of polynomials.

Solution:

- Linked list representation for sparse matrices.

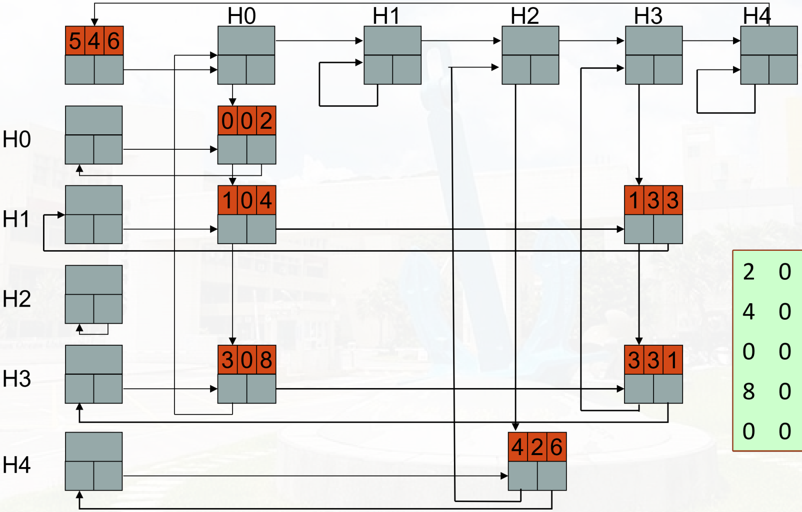
Issues for Previous Representation

- When we performed matrix operations such as $+$, $-$, or $*$, the number of **nonzero terms** varied.
- The sequential representation of sparse matrices suffered from the same inadequacies as the similar representation of polynomials.

Solution:

- Linked list representation for sparse matrices.
- Two types of nodes in the representation: header nodes and element nodes.

header node　　element node



2	0	0	0
4	0	0	3
0	0	0	0
8	0	0	1
0	0	6	0

Sparse Matrix Representation

- We represent each column (row) of a sparse matrix as a circularly linked list with a header node.
- The header node for row *i* is also the header node for column *i*. The number of header nodes is $\max\{\text{numRows}, \text{numCols}\}$.
- Each element node is simultaneously linked into two lists: a row list, and a column list.
- Each head node is belonged to three lists: a row list, a column list, and a header node list.

Outline

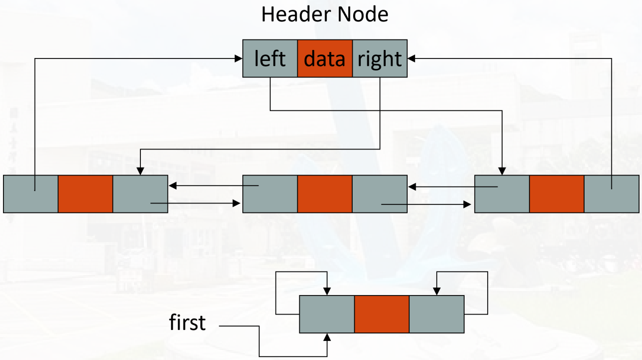
Issues for Singly Linked Lists

- The only way to find the node that precedes some node p is to start at the beginning of the list.
- Sometimes it is necessary to move in either direction.

Doubly linked lists:

```
typedef struct node *nodePointer;
typedef struct node {
    nodePointer llink;
    element data;
    nodePointer rlink;
};
```

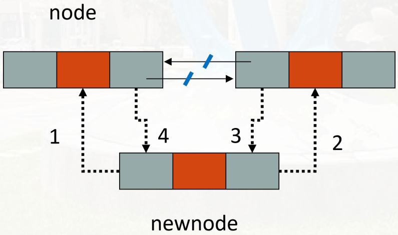
```
ptr = ptr->llink->rlink = ptr->rlink->llink
```



Empty doubly linked circular list with header node

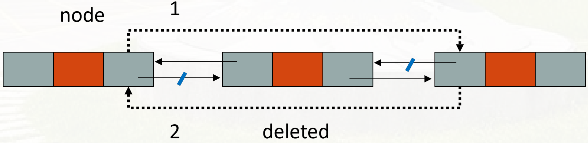
Insertion into a doubly linked circular List

```
void d_LCL_insert(nodePointer node, nodePointer newnode) {
/* insert newnode to the right of node */
    newnode->llink = node;            // 1
    newnode->rlink = node->rlink;     // 2
    node->rlink->llink = newnode;     // 3
    node->rlink = newnode;            // 4
}
```



Insertion into a doubly linked circular List

```
void d_LCL_delete(nodePointer node, nodePointer deleted) {
/* delete from the doubly linked list */
    if (node == deleted)
        printf("Deletion of header node not permitted.\n");
    else {
        deleted->llink->rlink = deleted->rlink;^^I^^I// 1
        deleted->rlink->llink = deleted->llink;^^I^^I// 2
        free(deleted);
    }
}
```



Discussions