Data Structures and Algorithms
CSC 201

SK Hafizul Islam

CSE, IIIT Kalyani

April 25, 2020

Agenda

1. Circularly Doubly Linked List
   - Insert a Node at the Beginning
   - Insert a Node at the End
   - Delete the First Node
   - Delete the Last Node
2. Header Linked Lists
   - Traverse a Circular Header Linked List
   - Search a Node in a Circular Header Linked List
   - Application of Linked Lists
3. Sparse Matrix
   - Representation of Structured Sparse Matrices
   - Representation of Unstructured Sparse Matrices

Circularly Doubly Linked List

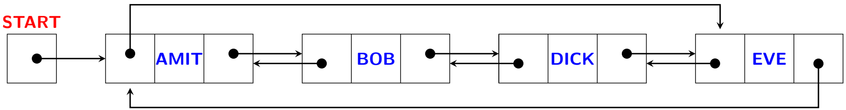


Figure 1: Circularly Doubly Linked List

- A ***circular doubly linked list*** or a ***circular two-way linked list*** does not contain **NULL** in the **PREV** field of the first node and the **NEXT** field of the last node.
- The **NEXT** field of the last node stores the address of the first node of the list, i.e., **START**.
- The **PREV** field of the first field stores the address of the last node.

Insert a Node at the Beginning of a Circular Doubly Linked List

Algorithm 1: INSCDLLFIRST(INFO,PREV,NEXT,START,AVAIL,ITEM)

This algorithm inserts ITEM as first node

1. IF AVAIL = NULL

   WRITE: OVERFLOW and Exit.
2. NEW:=AVAIL and AVAIL:=NEXT[AVAIL]//GETNODE()
3. INFO[NEW]:=ITEM.
4. PTR:=PREV[START].
5. NEXT[PTR]:=NEW.
6. PREV[NEW]:=PTR.
7. NEXT[NEW]:=START.
8. PREV[START]:=NEW.
9. START:=NEW.
10. Exit

- Time complexity for doubly linked list of size n: $O(1)$.

Insert a Node at the End of a Circular Doubly Linked List

Algorithm 2: INSCDLLEND(INFO,PREV,NEXT,START,AVAIL,ITEM)

This algorithm inserts ITEM as last node

1. IF AVAIL = NULL

   WRITE: OVERFLOW and Exit.
2. NEW:=AVAIL and AVAIL:=NEXT[AVAIL]//GETNODE()
3. INFO[NEW]:=ITEM.
4. PTR:=PREV[START].
5. NEXT[PTR]:=NEW.
6. PREV[NEW]:=PTR.
7. NEXT[NEW]:=START.
8. PREV[START]:=NEW.
9. Exit

- Time complexity for doubly linked list of size n: $O(1)$.

Delete the First Node from a Circular Doubly Linked List

Algorithm 3: DELCDLLFIRST(INFO,PREV,NEXT,START,AVAIL)

This algorithm deletes the first node from a circular doubly linked list

1. IF START = NULL

   WRITE: UNDERFLOW and Exit.
2. PTR:=PREV[START] and SAVE:=START.
3. NEXT[PTR]:=NEXT[START].
4. PREV[NEXT[START]]:=PTR.
5. PREV[START]:=NULL.
6. START:=NEXT[START].
7. NEXT[SAVE]:=AVAIL and AVAIL:=SAVE.//FREENODE(SAVE)
8. Exit

- Time complexity for doubly linked list of size n: $O(1)$.

Delete the Last Node from a Circular Doubly Linked List

Algorithm 4: DELCDLLEND(INFO,PREV,NEXT,START,AVAIL)

This algorithm deletes the last node from a circular doubly linked list

1. IF START = NULL

   WRITE: UNDERFLOW and Exit.
2. PTR:=PREV[START].
3. NEXT[PREV[PTR]]:=START.
4. PREV[START]:=PREV[PTR].
5. PREV[PTR]:=NULL.
6. NEXT[PTR]:=AVAIL and AVAIL:=PTR.//FREENODE(PTR)
7. Exit

- Time complexity for doubly linked list of size n: $O(1)$.

Header Linked List

- It is a special kind of linked list that includes a special node, called **header node**, at the front of the list.
- Here, **START** will point the **header node** not the first node of the list.
    - **Grounded header linked list:** It stores **NULL** in the **LINK/NEXT** field of the last node.
        - LINK[START]= NULL indicates the grounded header linked list is empty.
    - **Circular header linked list:** It stores the address of the header node in the **LINK/NEXT** field of the last node. In this case, the header node will denote the end of the list.
        - LINK[START] = START indicates the circular header linked list is empty.

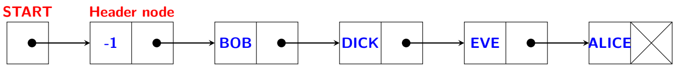


Figure 2: Grounded Header Linked List

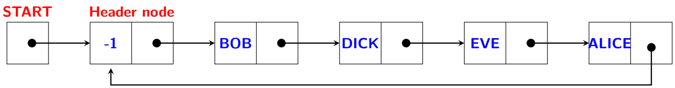


Figure 3: Circular Header Linked List

Traverse a Circular Header Linked List

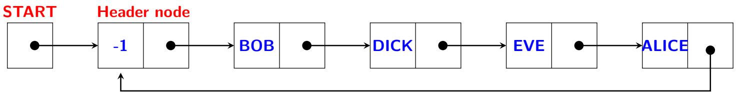
Algorithm 5: TRAVERSECHLL(LIST,START,INFO,LINK)

LIST is a circular header linked list. This algorithm traverses LIST.

1. PTR:=LINK[START].
2. Repeat Step 3 and Step 4 **while PTR$\neq$START**
3. PRINT: INFO[PTR].
4. PTR:=LINK[PTR].

   [End of Loop at Step 2]
5. Exit

- Time complexity for circular header linked list of size n: $O(n)$.



Search a Node in a Circular Header Linked List

Algorithm 6: SRCHCHLL(LIST,START,INFO,LINK,ITEM,LOC)

LIST is a circular header linked list. This algorithm find the location LOC of a node ITEM first appear in the LIST.

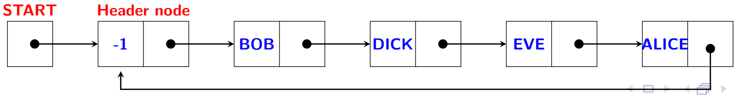
1. PTR:=LINK[START].
2. Repeat Step 3 **while INFO[PTR]≠ITEM** and **PTR≠START**
3. PTR:=LINK[PTR].

   [End of Loop at Step 2]
4. If INFO[PTR] = ITEM, then

   LOC:=PTR.

   Else

   LOC:=NULL.
5. Return LOC.
6. Exit



Application of Linked Lists: Polynomial Representation

- Header Linked lists can be used to represent polynomials and the different operations that can be performed on them.
- Consider a polynomial $p(x) = 6x^3 + 9x^2 + 7x + 1$.
- Every individual term in $p(x)$ consists of two parts, a coefficient and a exponent. Here, 6, 9, 7, and 1 are the coefficients of the terms that have 3, 2, 1, and 0 as their exponents, respectively.

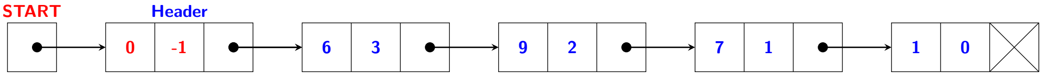


Figure 4: Header linked representation of a polynomial $p(x) = 6x^3 + 9x^2 + 7x + 1$

Application of Linked Lists: Polynomial Representation

```
struct node
{
    int coeff;
    int expo;
    struct node *link;
};
```

Application of Linked Lists: Polynomial Addition

```
struct node *addnode(struct node *start, int c, int n)
{
struct node *ptr, *newp;
 if(start == NULL)
  {
   newp = (struct node *)malloc(sizeof(struct node));
   newp -> coeff = c;
   newp -> expo = n;
   newp -> link = NULL;
   start = newp;
  }
 else
  {
   ptr = start;
   while(ptr -> link != NULL)
       ptr = ptr -> link;
   newp = (struct node *)malloc(sizeof(struct node));
   newp -> coeff = c;
   newp -> expo = n;
   newp -> link = NULL;
   ptr -> link = newp;
  } start; }
```

Application of Linked Lists: Polynomial Addition

```
struct node *addheadpol(struct node *start1, struct node *start2, struct node *start3)
{
 struct node *ptr1, *ptr2;
 int sumcoff;
 start3 = addnode(start3, 0, -1);//Header Node
 ptr1 = start1->link,
 ptr2 = start2->link;
 while(ptr1!= NULL && ptr2!= NULL)
  {
   if(ptr1->expo==ptr2->expo)
   {
     sumcoff = ptr1->coeff + ptr2->coeff;
     start3 = addnode(start3, sumcoff, ptr1->expo);
     ptr1 = ptr1->link;
     ptr2 = ptr2->link;
   }
```

Application of Linked Lists: Polynomial Addition

```
        else if(ptr1->expo > ptr2->expo)
         {
          start3 = addnode(start3, ptr1->coeff, ptr1->expo);
          ptr1 = ptr1 -> link;
         }
        else if(ptr1->expo<ptr2->expo)
         {
          start3 = addnode(start3, ptr2->coeff, ptr2->expo);
          ptr2 = ptr2 -> link;
         }
      }
    if(ptr1 == NULL)
      {
      while(ptr2 != NULL)
       {
        start3 = addnode(start3, ptr2->coeff, ptr2->expo);
        ptr2 = ptr2 -> link;
       }
      }
```

Application of Linked Lists: Polynomial Addition

```
if(ptr2 == NULL)
  {
  while(ptr1 != NULL)
   {
     start3 = addnode(start3, ptr1->coeff, ptr1->expo);
     ptr1 = ptr1 -> link;
   }
  }
return start3;
}
```

- Time Complexity: $O(m + n)$, where m and n are number of nodes in first and second lists, respectively.

Sparse Matrix

- **Sparse matrix** is a square matrix with a relatively high proportion of zero values[1].
- Rather than store such a matrix as a two-dimensional array with lots of zeroes, a common strategy is to save space by explicitly storing only the non-zero elements.
- If the operations using standard matrix structures and algorithms are applied to sparse matrices, then the execution will slow down and the matrix will consume large amount of memory.
- Sparse data can be easily compressed, which in turn can significantly reduce memory usage.
- **Structured sparse matrix**
  - Tridiagonal matrix
  - Lower triangular matrix
  - Upper triangular matrix
- **Unstructured sparse matrix**

[1] Section 4.14: Seymour Lipschutz, Data Structures. Mc-Graw Hill Education

Structured Sparse Matrix

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 2 & 2 & 8 & 0 & 0 \\ 0 & 1 & 3 & 6 & 0 \\ 0 & 0 & 2 & 9 & 8 \\ 0 & 0 & 0 & 3 & 5 \end{bmatrix}$$

(a) Tridiagonal matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 \\ 1 & 3 & 7 & 0 & 0 \\ 3 & 5 & 13 & 9 & 0 \\ 3 & 5 & 13 & 9 & 7 \end{bmatrix}$$

(b) Lower Triangular matrix

$$\begin{bmatrix} 1 & 1 & 3 & 1 & 2 \\ 0 & 2 & 8 & 4 & 5 \\ 0 & 0 & 3 & 6 & 7 \\ 0 & 0 & 0 & 9 & 8 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

(c) Upper triangular matrix

Figure 5: Different types of structured sparse matrix

Lower Triangular Sparse Matrix

$$A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

- Assume that we want to store the lower-triangular sparse matrix A. Therefore, it would be wastage to store all the zeros (above the main diagonal) of A. We only store non-zero values.
- The number of non-zero values in A is

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Lower Triangular Sparse Matrix

- Assume that $B[]$ is linear array, which will store all the non-zero elements of the matrix A.
- Therefore, $B[1] = a_{11}$, $B[2] = a_{21}$, $B[3] = a_{22}$, $B[4] = a_{31}$, $\cdots$
- Assume that $B[i] = a_{jk}$. Therefore, i represents the number of elements in the list up to and including a_{jk}. Now these are

$$1 + 2 + \cdots + (j-1) = \frac{j(j-1)}{2}$$

elements in the row above a_{jk} and there are k elements in the row j up to and including a_{jk}. Accordingly,

$$i = \frac{j(j-1)}{2} + k$$

Representation of Unstructured Sparse Matrices[2]

- **Single linear list in row-major order:** Scan the nonzero elements of the sparse matrix in row-major order each nonzero element is represented by a triple (row, column, value) the list of triples may be an array list or a linked list (chain)

$$\begin{bmatrix} 6 & 0 & 0 & 2 & 0 & 5 \\ 4 & 4 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

(a) Unstructured sparse matrix

Row	0	0	0	1	1	1	2	2	3	3
Col	0	3	5	0	1	5	1	4	3	4
Val	6	2	5	4	4	1	1	2	1	1

(b) linear list

Figure 6: Linear list representation of unstructured sparse matrix

[2]Lecture 11:https://www.cise.ufl.edu/∼sahni/cop3530/presentations.htm

Linked List Representation of Sparse Matrices[3]

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

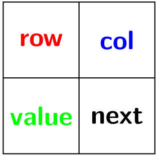


Figure 7: Structure of a Node

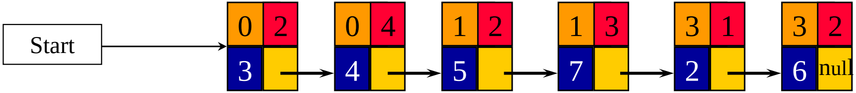
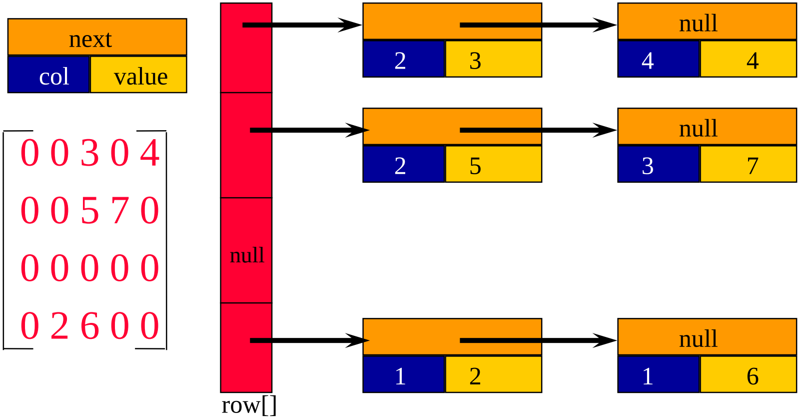


Figure 8: Linked List Representation

[3] Lecture 11: https://www.cise.ufl.edu/∼sahni/cop3530/presentations.htm

Array of Row Chains Representation of Sparse Matrices[4]



[4]Lecture 11: https://www.cise.ufl.edu/~sahni/cop3530/presentations.htm

Orthogonal List Representation of Sparse Matrices[5]

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

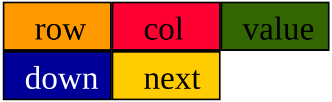


Figure 9: Node Structure

[5] Lecture 11: https://www.cise.ufl.edu/~sahni/cop3530/presentations.htm

Orthogonal List Representation of Sparse Matrices[6]

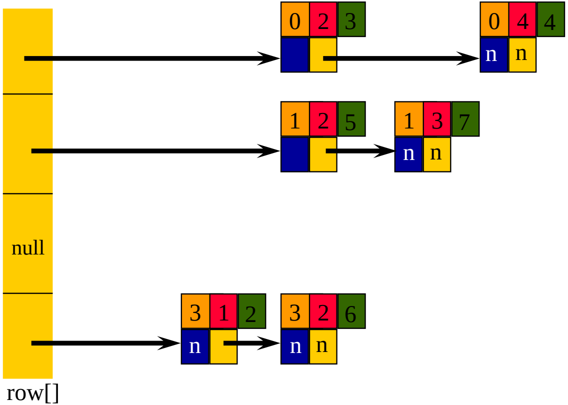


Figure 10: Row list

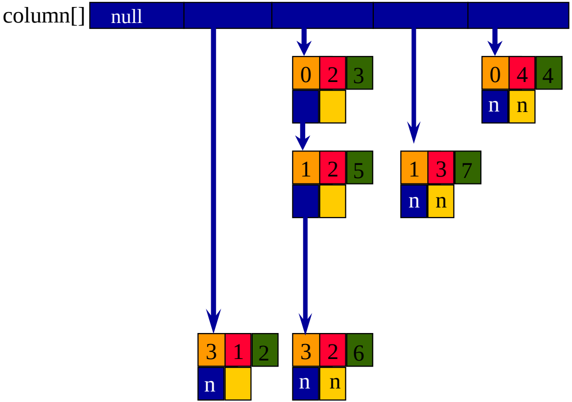


Figure 11: Column list

[6] Lecture 11: https://www.cise.ufl.edu/~sahni/cop3530/presentations.htm

Orthogonal List Representation of Sparse Matrices[7]

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

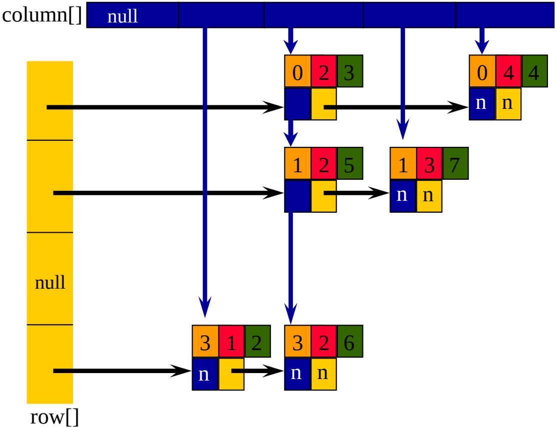


Figure 12: Orthogonal List

[7]Lecture 11: https://www.cise.ufl.edu/~sahni/cop3530/presentations.htm

Thank You