**V Venkateswarlu Gupta**

**1602-19-733-119**

**CSE-B**

**Data Structures Lab**

Week – 7

1.DIFFERENTIATE BEWTEEN CIRCULAR QUEUE AND DEQUEUE?

a. CIRCULAR QUEUE:

In a Queue when elements are arranged in sequential manner but logically we assume it in circular format, then such queue is called as “Circular Queue”.

1)It follows “First In First Out” method for insertion and deletion of elements. Insertion and deletion are performed at specific ends.

DEQUEUE:

A Queue in which inserting and deleting of elements is done from both the ends, such queue is called as Double Ended Queue(DeQueue).

1) There are two types of DeQueue. A. Input restricted dequeues B. Output restricted dequeues.

2)Input restricted dequeues allows insertion only at one end but allows deletion of element from both the ends.

3)Output restricted dequeues allwos deletions of elements only at one end but allows insertion of element from both the ends.

2.Represent circular queue with an example

A. 1) QUEUE IS FULL(REACHED ITS MAX LIMIT)

2 5 8 7 9 0 3 5 1

2) PERFORMED TWO DEQUEUE OPERATIONS

8 7 9 0 3 5 1

Front rear

Now it shows as queue is full because rear has reached the Max.

But we have two empty spaces. Circular queue makes use of it.

3) Perform enqueue operation 19

19 8 7 9 0 3 5 1

rear front

3.Write a condition to check for deleting an element from a circular queue.

A. int deletion(){

if(front==-1){

printf("Queue is empty\n");

return;

}

else if(front==rear){

x= queue[0];

front=-1;

rear=-1;

return x;

}

else{x=queue[front];

front=(front+1)%N;

return x;

}

}

4. Write the condition to be checked while inserting an element into a circular queue.

A. void Insertion(int x){

if((rear+1)%N==front){

printf("Queue is full\n");

return;

}

else if(front==-1&&rear==-1){

front=0;

}

rear=(rear+1)%N;

queue[rear]=x;

}

5.What is a dequeue? Write the condition to check for a inserting element into a dequeue?

A. A Queue in which inserting and deleting of elements is done from both the ends, such queue is called as Double Ended Queue(DeQueue).

void Insert\_beginning(){

int x;

printf("Enter the element to u want to insert at the beg\n");

scanf("%d",&x);

if(Isfull()==1){

printf("Queue is full\n");

}

else if(Isempty()==1){

front=0;

rear=0;

dequeue[rear]=x;

printf("Inserted successfully!\n");

}

else{

front=(front-1+N)%N;

dequeue[front]=x;

printf("Inserted successfully!\n");

}

}

void Insert\_end(){

int x;

printf("Enter the element to u want to insert at the end\n");

scanf("%d",&x);

if(Isempty()==1){

front=0;

rear=0;

dequeue[rear]=x;

printf("Inserted successfully!\n");

}

else if(Isfull()==1){

printf("Queue is full\n");

}

else{

rear=(rear+1)%N;

dequeue[rear]=x;

printf("Inserted successfully!\n");

}

}

PreLab Programs:

1. Circular Q
2. #include <stdio.h>
3. #define N 5
4. *int* queue[N];
5. *int* front = -1;
6. *int* rear = -1;
7. *void* Insertion(*int* *x*)
8. {
9. if ((rear + 1) % N == front)
10. {
11. printf("Queue is full\n");
12. return;
13. }
14. else if (front == -1 && rear == -1)
15. {
16. front = 0;
17. }
18. rear = (rear + 1) % N;
19. queue[rear] = *x*;
20. }
21. *void* deletion()
22. {
23. if (front == -1)
24. {
25. printf("Queue is empty\n");
26. return;
27. }
28. else if (front == rear)
29. {
30. front = -1;
31. rear = -1;
32. }
33. else
34. {
35. front = (front + 1) % N;
36. }
37. }
38. *void* display()
39. {
40. if (front == -1 && rear == -1)
41. {
42. printf("Queue is empty\n");
43. }
44. else
45. {
46. *int* i;
47. printf("Elements in the queue are:\n");
48. for (i = front; i != rear; i = (i + 1) % N)
49. {
50. printf("%d ", queue[i]);
51. }
52. printf("%d", queue[i]);
53. }
54. }
55. *int* main()
56. {
57. *int* c, x;
58. do
59. {
60. printf("Choose any option:\n");
61. printf("1.Insertion\n2.Deletion\n3.Display\n");
62. scanf("%d", &c);
63. switch (c)
64. {
65. case 1:
66. printf("Enter the element you want to insert:\n");
67. scanf("%d", &x);
68. Insertion(x);
69. display();
70. break;
71. case 2:
72. deletion();
73. display();
74. break;
75. case 3:
76. display();
77. break;
78. }
79. printf("\nWant to continue?(1/0)\n");
80. scanf("%d", &c);
81. } while (c != 0);
82. }

2. Dequeue.

#include <stdio.h>

#define N 5

*int* dequeue[N];

*int* front = -1;

*int* rear = -1;

*int* Isempty()

{

    if (front == -1 && rear == -1)

    {

        return 1;

    }

    else

        return 0;

}

*int* Isfull()

{

    if (rear + 1 == front || rear == N - 1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

*void* Insert\_beginning()

{

*int* x;

    printf("Enter the element to u want to insert at the beg\n");

    scanf("%d", &x);

    if (Isfull() == 1)

    {

        printf("Queue is full\n");

    }

    else if (Isempty() == 1)

    {

        front = 0;

        rear = 0;

        dequeue[rear] = x;

        printf("Inserted successfully!\n");

    }

    else

    {

        front = (front - 1 + N) % N;

        dequeue[front] = x;

        printf("Inserted successfully!\n");

    }

}

*void* Insert\_end()

{

*int* x;

    printf("Enter the element to u want to insert at the end\n");

    scanf("%d", &x);

    if (Isempty() == 1)

    {

        front = 0;

        rear = 0;

        dequeue[rear] = x;

        printf("Inserted successfully!\n");

    }

    else if (Isfull() == 1)

    {

        printf("Queue is full\n");

    }

    else

    {

        rear = (rear + 1) % N;

        dequeue[rear] = x;

        printf("Inserted successfully!\n");

    }

}

*void* delete\_front()

{

*int* x;

    if (Isempty() == 1)

    {

        printf("Cannot delete as queue is empty\n");

    }

    else

    {

        x = dequeue[front];

        if (front == rear)

        {

            front = -1;

            rear = -1;

        }

        else

        {

            front = (front + 1) % N;

        }

        printf("%d got deleted successfully!\n", x);

    }

}

*void* delete\_end()

{

*int* x;

    if (Isempty() == 1)

    {

        printf("Cannot delete as queue is empty\n");

    }

    else

    {

        x = dequeue[rear];

        if (front == rear)

        {

            front = -1;

            rear = -1;

        }

        else

        {

            rear = (rear - 1 + N) % N;

        }

        printf("%d got deleted successfully!\n", x);

    }

}

*int* display()

{

*int* i;

    printf("The elements in the queue are:\n");

    if (Isempty() == 1)

    {

        printf("Queueis empty\n");

    }

    else

    {

        for (i = front; i != rear; i = (i + 1) % N)

        {

            printf("%d ", dequeue[i]);

        }

        printf("%d", dequeue[rear]);

    }

}

*int* main()

{

*int* c;

    printf("\_\_\_\_\_\_\_\_\_\_DEQUEUE\_\_\_\_\_\_\_\_\_\n");

    do

    {

        printf("\nChoose one of the options:\n1.Insertion at beg\n2.Insertion at the end\n3.Deletion at beg\n4.Deltion at the end\n5.Display\n6.Exit\n");

        scanf("%d", &c);

        switch (c)

        {

        case 1:

            Insert\_beginning();

            display();

            break;

        case 2:

            Insert\_end();

            display();

            break;

        case 3:

            delete\_front();

            display();

            break;

        case 4:

            delete\_end();

            display();

            break;

        case 5:

            display();

            break;

        case 6:

            printf("-----------Exiting----------");

            break;

        }

    } while (c != 6);

}

3.Sparse Matrix using ll

#include<stdio.h>

#include<stdlib.h>

typedef *struct* node{

*int* val;

*int* row;

*int* col;

*struct* node \*link;

}Sparse;

Sparse \*h1,\*h2;

*int* cnt=0,m,n;

Sparse \*storeFromFile(Sparse \**h*,*int* *fileno*){

    FILE \*fp;

    Sparse \*q;

    q=*h*;

    if(*fileno*==1){

        fp=fopen("input1.txt","r");

    }else if(*fileno*==2){

        fp=fopen("input2.txt","r");

    }

*int* i,j,v;

    fscanf(fp,"%d %d",&m,&n);

    for(i=0;i<m;i++){

        for(j=0;j<n;j++){

            fscanf(fp,"%d",&v);

            if(v!=0){

                cnt++;

                Sparse \*t=(Sparse \*)malloc(sizeof(Sparse));

                t->val=v;

                t->row=i;

                t->col=j;

                t->link=NULL;

                if(q==NULL){

*h*=t;

                    q=t;

                }else{

                    q->link=t;

                    q=t;

                }

            }

        }

    }

    return *h*;

}

Sparse \*add(Sparse \**h1*,Sparse \**h2*){

    Sparse \*r,\*q1,\*q2,\*q;

    q1=*h1*;

    q2=*h2*;

    r=NULL;

    q=r;

    while(1){

        Sparse \*t=(Sparse \*)malloc(sizeof(Sparse));

        if(q1!=NULL&&q2!=NULL){

            if(q1->row==q2->row&&q1->col==q2->col){

                t->row=q1->row;

                t->col=q1->col;

                t->val=q1->val+q2->val;

                t->link=NULL;

                if(q==NULL){

                    r=t;

                    q=t;

                }else{

                    q->link=t;

                    q=t;

                }

                q1=q1->link;

                q2=q2->link;

            }else{

                if(q1->row==q2->row){

                    if(q1->col<q2->col){

                        t->row=q1->row;

                        t->col=q1->col;

                        t->val=q1->val;

                        t->link=NULL;

                        if(q==NULL){

                            r=t;

                            q=t;

                        }else{

                            q->link=t;

                            q=t;

                        }

                        q1=q1->link;

                    }else{

                        t->row=q2->row;

                        t->col=q2->col;

                        t->val=q2->val;

                        t->link=NULL;

                        if(q==NULL){

                            r=t;

                            q=t;

                        }else{

                            q->link=t;

                            q=t;

                        }

                        q2=q2->link;

                    }

                }else if(q1->row<q2->row){

                    t->row=q1->row;

                    t->col=q1->col;

                    t->val=q1->val;

                    t->link=NULL;

                    if(q==NULL){

                        r=t;

                        q=t;

                    }else{

                        q->link=t;

                        q=t;

                    }

                    q1=q1->link;

                }else{

                    t->row=q2->row;

                    t->col=q2->col;

                    t->val=q2->val;

                    t->link=NULL;

                    if(q==NULL){

                        r=t;

                        q=t;

                    }else{

                        q->link=t;

                        q=t;

                    }

                    q2=q2->link;

                }

            }

        }else if(q1==NULL){

            t->row=q2->row;

            t->col=q2->col;

            t->val=q2->val;

            t->link=NULL;

            if(q==NULL){

                r=t;

                q=t;

            }else{

                q->link=t;

                q=t;

            }

            q2=q2->link;

        }else{

            t->row=q1->row;

            t->col=q1->col;

            t->val=q1->val;

            t->link=NULL;

            if(q==NULL){

                r=t;

                q=t;

            }else{

                q->link=t;

                q=t;

            }

            q1=q1->link;

        }

        if(q1==NULL&&q2==NULL) break;

    }

    return r;

}

*void* display(Sparse \**h*){

    Sparse \*q;

    q=*h*;

*int* i=0;

    if(q==NULL){

        printf("All elements are ZERO's\n");

    }else if(q->link==NULL){

        printf("ROW \t COLUMN \t VALUE\n");

        printf(" %d  \t    %d  \t  \t %d\n",q->row,q->col,q->val);

    }else{

        printf("ROW \t COLUMN \t VALUE\n");

        while(q!=NULL){

            printf(" %d  \t    %d  \t  \t %d\n",q->row,q->col,q->val);

            q=q->link;

            i++;

        }

    }

}

Sparse \*copy(Sparse \**h*,*int* *rowcol*){

    Sparse \*t,\*q,\*qc,\*hc;

    q=*h*;

    hc=NULL;

    qc=hc;

    while(q!=NULL){

        t=(Sparse \*)calloc(1,sizeof(Sparse));

        t->link=q->link;

        if(*rowcol*==0){

            t->row=q->row;

            t->col=q->col;

        }else if(*rowcol*==1){

            t->row=q->col;

            t->col=q->row;

        }

        t->val=q->val;

        if(hc==NULL){

            hc=t;

            qc=t;

        }else{

            qc->link=t;

            qc=t;

        }

        q=q->link;

    }

    return hc;

}

*void* bubbleSort(Sparse \**h*){

    Sparse \*i,\*j;

    i=*h*;

    while(i!=NULL)

    {

        j=i->link;

        while(j!=NULL)

        {

            if(i->row>j->row)

            {

                i->row=i->row+j->row;

                j->row=i->row-j->row;

                i->row=i->row-j->row;

                i->col=i->col+j->col;

                j->col=i->col-j->col;

                i->col=i->col-j->col;

                i->val=i->val+j->val;

                j->val=i->val-j->val;

                i->val=i->val-j->val;

            }

            j=j->link;

        }

        i=i->link;

    }

}

*void* storeToFile(Sparse \**r*,*int* *m0*,*int* *n0*,*char* \**filename*){

    FILE \*fp;

    Sparse \*q;

    q=*r*;

    fp=fopen(*filename*,"w");

    fprintf(fp,"%d %d\n",*m0*,*n0*);

    for(*int* j,i=0;i<*m0*;i++){

        for(j=0;j<*n0*;j++){

            if(q->row==i&&q->col==j){

                fprintf(fp,"%d ",q->val);

                q=q->link;

            }else{

                fprintf(fp,"0 ");

            }

        }

        fprintf(fp,"\n");

    }

}

Sparse \*transpose(Sparse \**h*){

    Sparse \*q,\*ht=copy(*h*,1);

    q=ht;

    printf("Transpose of the matrix\n");

    bubbleSort(ht);

    display(ht);

    storeToFile(ht,n,m,"transpose.txt");

}

*void* main(){

    h1=NULL;

    h1=storeFromFile(h1,1);

    h2=NULL;

    h2=storeFromFile(h2,2);

    printf("First Sparse Matrix\n");

    display(h1);

    Sparse \*ht=transpose(h1);

    printf("Second Sparse Matrix\n");

    display(h2);

    Sparse \*sum;

    sum=add(h1,h2);

    display(sum);

    storeToFile(sum,m,n,"sum.txt");

}

Lab Programs:

1.Processes

#include<stdio.h>

*int* noOfRoundsRequired=0,noOfProcesses=0,circularQFront=-1,circularQRear=-1;

*int* waitingQFront=-1,waitingQRear=-1;

*int*  waitingQ[100][2],circularQ[4][2];

*void* saving(*int* *activeProcesses*){

    if((circularQRear+1)%*activeProcesses*==circularQFront){

        if(waitingQFront==-1)   waitingQFront++;

        scanf(" %d %d",&waitingQ[++waitingQRear][0],&waitingQ[waitingQRear][1]);

    }

    else{

        if(circularQFront==-1)

            ++circularQFront;

        circularQRear=(circularQRear+1)%*activeProcesses*;

        scanf(" %d %d",&circularQ[circularQRear][0],&circularQ[circularQRear][1]);

    }

}

*void* processing(){

*int* timeSlice;

    printf("Enter time slice : ");

    scanf("%d",&timeSlice);

    while(noOfProcesses>0){

        for(*int* i=circularQFront;i<=circularQRear;i++){

            if(circularQ[i][0]==-1&&circularQ[i][1]==-1){

                continue;

            }

            circularQ[i][0]+=timeSlice;

            if(circularQ[i][0]>=circularQ[i][1]){

                if(waitingQFront!=-1){// waiting q not empty

                    circularQ[i][0]=waitingQ[waitingQFront][0];

                    circularQ[i][1]=waitingQ[waitingQFront][1];

                    waitingQFront++;

                }else{

                    circularQ[i][0]=-1;

                    circularQ[i][1]=-1;

                }

                noOfProcesses--;

            }

        }

        noOfRoundsRequired++;

    }

    printf("No of rounds required to complete all processes : %d",noOfRoundsRequired);

}

*void* main(){

    printf("Enter no of processes : ");

    scanf("%d",&noOfProcesses);

    for(*int* i=0;i<noOfProcesses;i++){

        printf("Enter %dth process : ",i);

        saving(4);

    }

    processing();

}

2.Addition and subtraction of polynomials.

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef *struct* node{

*int* coef;

*int* exp;

*struct* node \*link;

}Polynomial;

Polynomial \*createPolynomial(Polynomial \**h*){

    Polynomial \*q,\*t;

    q=*h*;

*int* n;

*char* ch;

    printf("No of terms : ");

    scanf("%d",&n);

    for(*int* i=0;i<n;i++){

        t=(Polynomial \*)malloc(sizeof(Polynomial));

        scanf("%d%c%c%d",&t->coef,&ch,&ch,&t->exp);

        t->link=NULL;

        if(q==NULL){

            q=t;

*h*=t;

        }else{

            q->link=t;

            q=t;

        }

    }

    return *h*;

}

Polynomial \*addPolynomials(Polynomial \**h1*, Polynomial \**h2*){

    Polynomial \*r,\*q1,\*q2,\*q;

    q1=*h1*;

    q2=*h2*;

    r=NULL;

    q=r;

    while(1){

        Polynomial \*t=(Polynomial \*)malloc(sizeof(Polynomial));

        if(q1!=NULL&&q2!=NULL){

            if(q1->exp==q2->exp){

                t->exp=q1->exp;

                t->coef=q1->coef+q2->coef;

                t->link=NULL;

                if(q==NULL){

                    r=t;

                    q=t;

                }else{

                    q->link=t;

                    q=t;

                }

                q1=q1->link;

                q2=q2->link;

            }else if(q1->exp>q2->exp){

                t->exp=q1->exp;

                t->coef=q1->coef;

                t->link=NULL;

                if(q==NULL){

                    r=t;

                    q=t;

                }else{

                    q->link=t;

                    q=t;

                }

                q1=q1->link;

            }else{

                t->exp=q2->exp;

                t->coef=q2->coef;

                t->link=NULL;

                if(q==NULL){

                    r=t;

                    q=t;

                }else{

                    q->link=t;

                    q=t;

                }

                q2=q2->link;

            }

        }else if(q1==NULL){

            t->exp=q2->exp;

            t->coef=q2->coef;

            t->link=NULL;

            if(q==NULL){

                r=t;

                q=t;

            }else{

                q->link=t;

                q=t;

            }

            q2=q2->link;

        }else{

            t->exp=q1->exp;

            t->coef=q1->coef;

            t->link=NULL;

            if(q==NULL){

                r=t;

                q=t;

            }else{

                q->link=t;

                q=t;

            }

            q1=q1->link;

        }

        if(q1==NULL&&q2==NULL) break;

    }

    return r;

}

*void* display(Polynomial \**h*){

    Polynomial \*q;

    q=*h*;

    while(q->link!=NULL){

        printf("%dx^%d",q->coef,q->exp);

        if(q->link->coef>0){

            printf("+");

        }

        q=q->link;

    }

    printf("%dx^%d",q->coef,q->exp);

}

*void* main(){

    //  12x^6+3x^3+14x^2+9x^1

    //  6x^6+5x^5-6x^3

    Polynomial \*h1, \*h2, \*r;

    h1=NULL;

    h2=NULL;

    h1=createPolynomial(h1);

    h2=createPolynomial(h2);

    r=addPolynomials(h1,h2);

    display(r);

}