Chương 8
CẤU TRÚC DỮ LIỆU HÀNG ĐỢI-QUEUE

TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

Bài 5 - 1

1. NGUYÊN LÝ HOẠT ĐỘNG

Cấu trúc dữ liệu hàng đợi hoạt động theo nguyên lý **vào trước, ra trước** (FIFO - First In- First Out)

TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

Click to see Figure 1-2

2. CẤU TRÚC DỮ LIỆU CỦA HÀNG ĐỢI

```
1. struct queue
2. {
3.     int n;
4.     KDL a[100];
5. };
6. typedef struct queue QUEUE;
```

- KDL là kiểu dữ liệu của đối tượng được lưu trong `queue`.

3. KHỞI TẠO HÀNG ĐỢI

- Khái niệm: Khởi tạo hàng đợi là tạo ra hàng đợi rỗng không chứa đối tượng nào hết.
- Định nghĩa hàm

```
1. void Init(QUEUE &que)
2. {
3.     que.n=0;
4. }
```

4. KIỂM TRA HÀNG ĐỢI RỖNG

- Khái niệm: Kiểm tra hàng đợi rỗng là hàm trả về giá trị 1 khi hàng đợi rỗng. Trong tình huống hàng đợi chưa rỗng thì hàm sẽ trả về giá trị 0.
- Định nghĩa hàm

```
1. int IsEmpty(QUEUE que)
2. {
3.     if(que.n==0)
4.         return 1;
5.     return 0;
6. }
```

5. KIỂM TRA HÀNG ĐỢI ĐẦY

- Khái niệm: Kiểm tra hàng đợi đầy là hàm trả về giá trị 1 khi hàng đợi đã đầy và trả về giá trị 0 khi hàng đợi chưa đầy.
- Định nghĩa hàm:

```
1. int IsFull(QUEUE que)
2. {
3.     if(que.n==100)
4.         return 1;
5.     return 0;
6. }
```

6. THÊM MỘT ĐỐI TƯỢNG VÀO TRONG HÀNG ĐỢI

- Khái niệm: Thêm một đối tượng vào hàng đợi xét về mặt kỹ thuật với CTDL đã được khai báo bên trên là việc thêm đối tượng đó vào cuối mảng a đang có n phần tử của hàng đợi mà thôi.
- Định nghĩa hàm

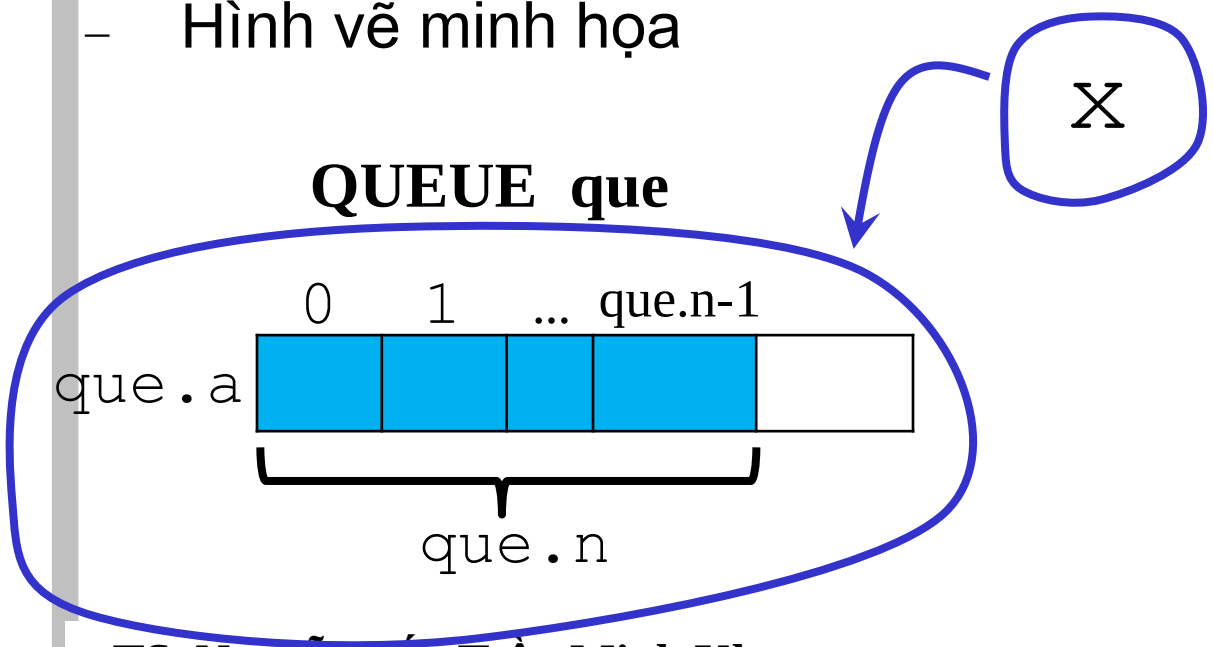
```
1. void EnQueue(QUEUE &que,
                KDL x)
2. {
3.     que.a[que.n] = x;
4.     que.n++;
5. }
```

6. THÊM MỘT ĐỐI TƯỢNG VÀO TRONG HÀNG ĐỢI

- Định nghĩa hàm

```
1. void EnQueue(QUEUE &que,
                KDL x)
2. {
3.     que.a[que.n] = x;
4.     que.n++;
5. }
```

- Hình vẽ minh họa



7. LẤY MỘT ĐỐI TƯỢNG RA KHỎI HÀNG ĐỢI

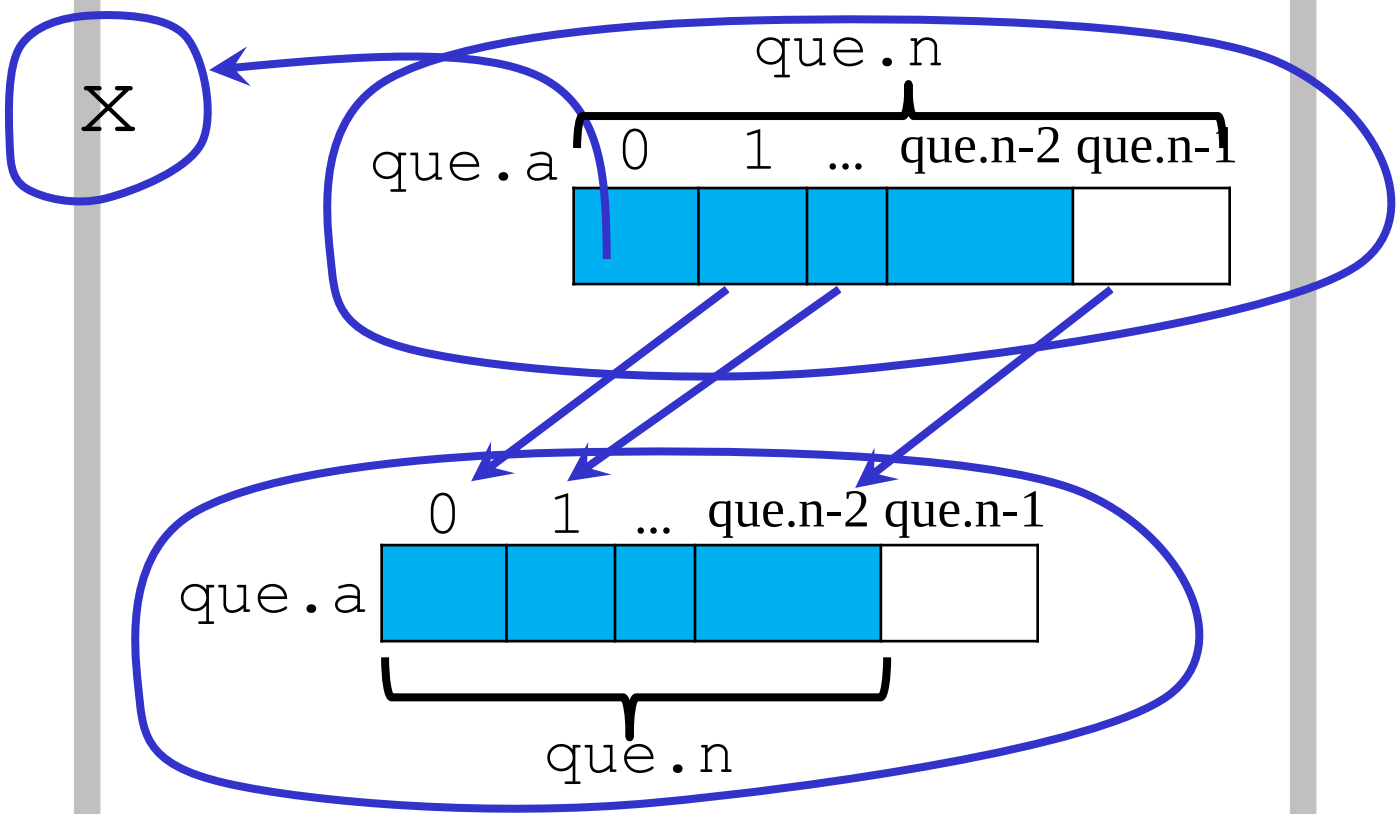
- Khái niệm: Lấy một đối tượng ra khỏi hàng đợi xét về mặt kỹ thuật với CTDL đã được khai báo bên trên là việc lấy đối tượng đầu mảng a của hàng đợi (queue) ra khỏi mảng mà thôi.
- Định nghĩa hàm

```
1. KDL DeQueue(QUEUE &que)
2. {
3.     KDL x = que.a[0];
4.     for(int i=0;i<=que.n-2;i++)
5.         que.a[i] = que.a[i+1];
6.     que.n--;
7.     return x;
8. }
```

- Định nghĩa hàm

```
1. KDL DeQueue(QUEUE &que)
2. {
3.     KDL x = que.a[0];
4.     for(int i=0;i<=que.n-2;i++)
5.         que.a[i] = que.a[i+1];
6.     que.n--;
7.     return x;
8. }
```

- Định nghĩa hàm **QUEUE que**



QUEUE que

ỨNG DỤNG

- Bài toán: Định nghĩa hàm tính tổng các giá trị trong cây nhị phân các số thực bằng hai phương pháp
  - Đệ quy.
  - Khử đệ quy với kỹ thuật queue.

ỨNG DỤNG (tiếp)

- Cách 1: Đệ quy

```
11.struct node
12.{
13.|    float info;
14.|    struct node *pLeft;
15.|    struct node *pRight;
16.};
17.typedef struct node NODE;
18.typedef NODE*TREE;
19.float Tong(TREE t)
20.{
21.|    if (!t)
22.|        return 0;
23.|    float a = Tong(t->pLeft);
24.|    float b = Tong(t->pRight);
25.|    return a+b+t->info;
26.}
```

ỨNG DỤNG (tiếp)

- Cách 2: Khử đệ quy
  - Cấu trúc dữ liệu:

```
11. struct node
12. {
13.     float info;
14.     struct node *pLeft;
15.     struct node *pRight;
16. };
17. typedef struct node NODE;
18. typedef NODE*TREE;
9.  struct queue
10. {
11.     int n;
12.     NODE* a[100];
13. };
14. typedef struct queue QUEUE;
```

ỨNG DỤNG (tiếp)

- Định nghĩa hàm

```
11. void Init(QUEUE &que)
12. {
13.     que.n=0;
14. }
15. int IsEmpty(QUEUE que)
16. {
17.     if (que.n==0)
18.         return 1;
19.     return 0;
20. }
```

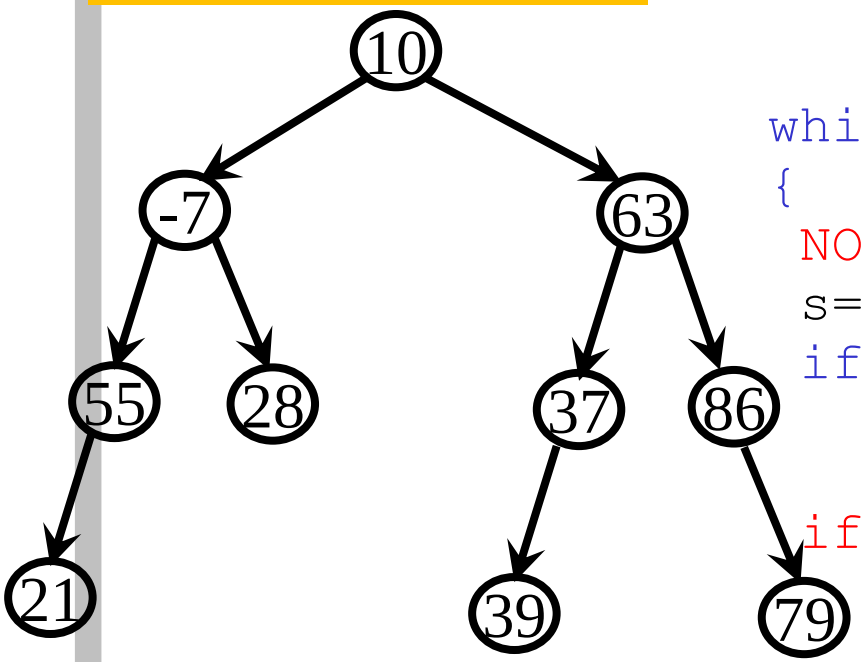
ỨNG DỤNG (tiếp)

```
11. int IsFulll(QUEUE que)
12. {
13.     if(que.n==100)
14.         return 1;
15.     return 0;
16. }
17. void EnQueue(QUEUE&que,NODE* x)
18. {
19.     que.a[que.n]=x;
20.     que.n++;
21. }
22. NODE* DeQueue(QUEUE &que)
23. {
24.    NODE* x = que.a[0];
25.    for(int i=0;i<=que.n-2;i++)
26.        que.a[i]=que.a[i+1];
27.    que.n--;
28.    return x;
29. }
```

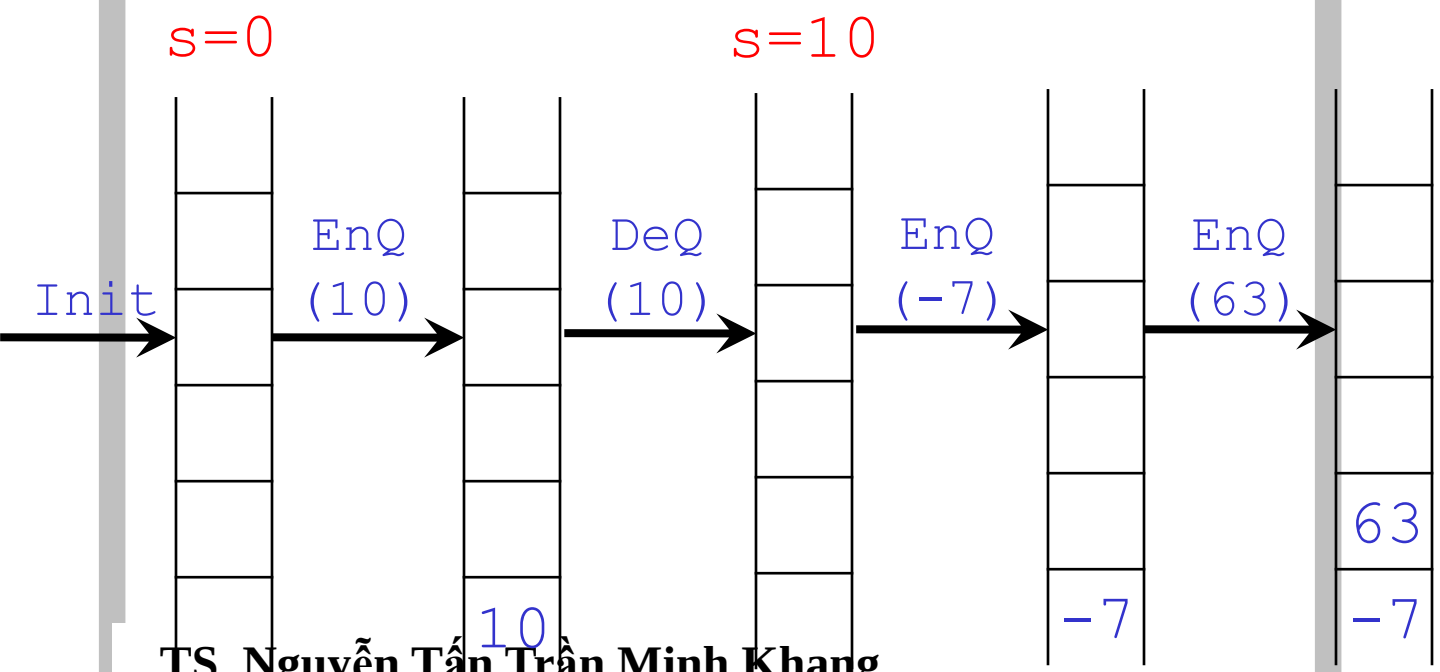
ỨNG DỤNG (tiếp)

```
11. float Tong(TREE t)
12. {
13.     float s=0;
14.     QUEUE q;
15.     Init(q);
16.     if (t!=NULL)
17.         EnQueue(q,t);
18.     while(IsEmty(q)==0)
19.     {
20.         NODE*p = DeQueue(q);
21.         s=s+p->info;
22.         if (p->pLeft)
23.             EnQueue(q,p->pLeft);
24.         if (p->pRight)
25.             EnQueue(q,p->pRight);
26.     }
27.     return s;
28. }
```

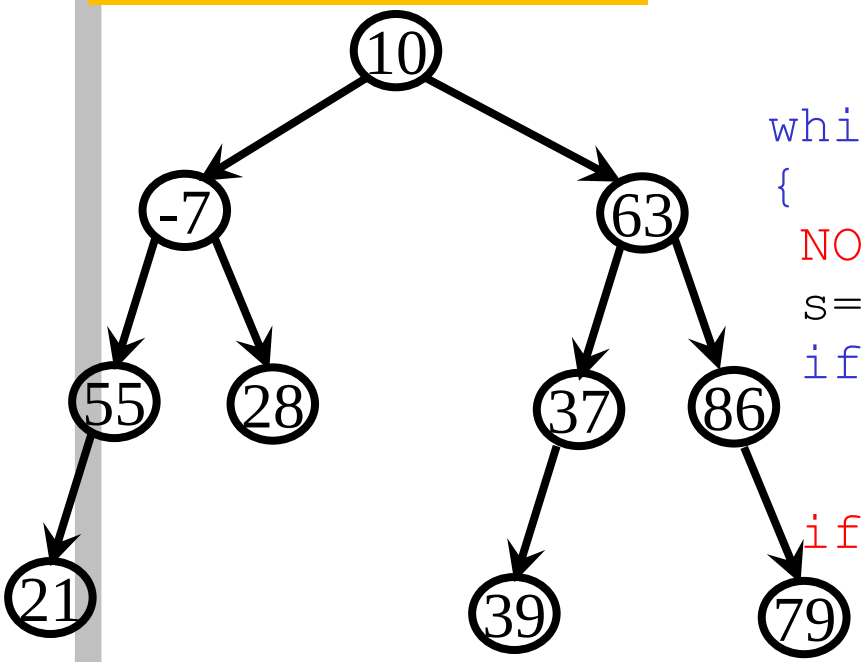
Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:



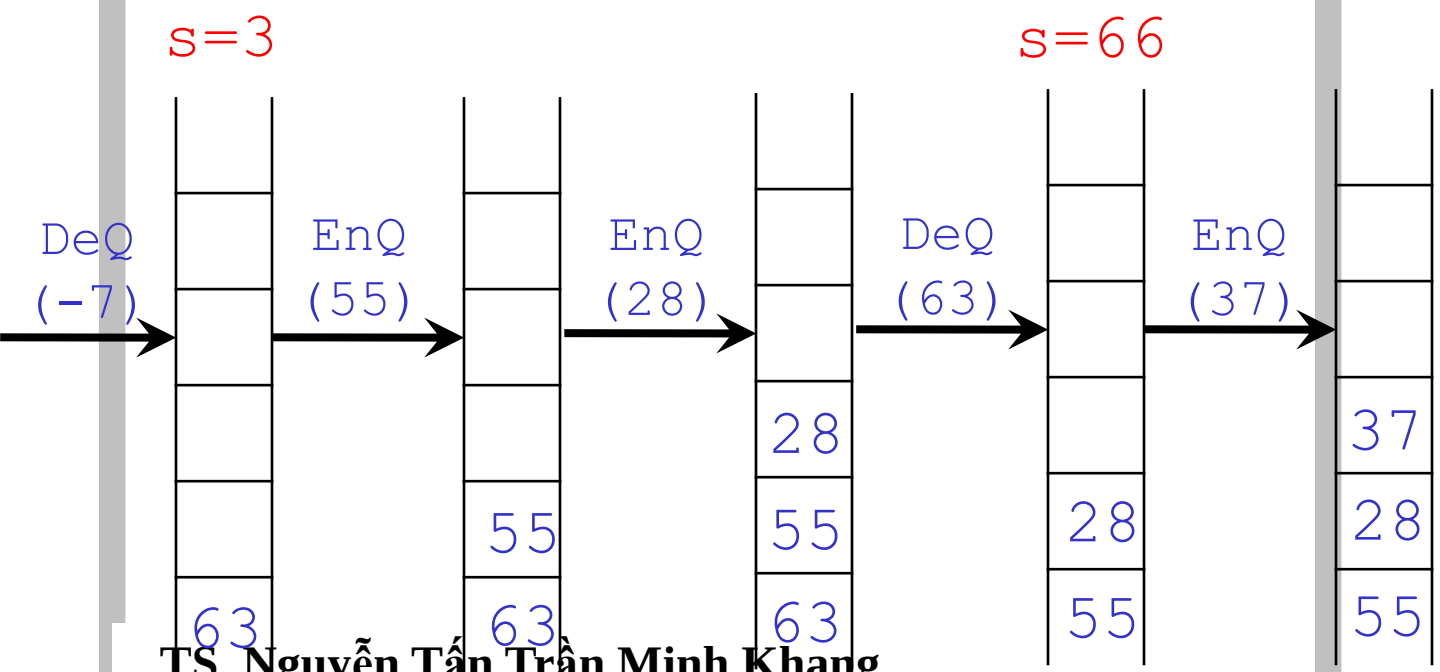

```
float s=0;
QUEUE q;
Init(q);
if (t!=NULL)
    EnQueue(q,t);
while(IsEmty(q)==0)
{
 NODE*p=DeQueue(q);
 s=s+p->info;
 if (p->pLeft)
   EnQueue(q,
           p->pLeft);
 if (p->pRight)
  EnQueue(q,
           p->pRight);
}
```



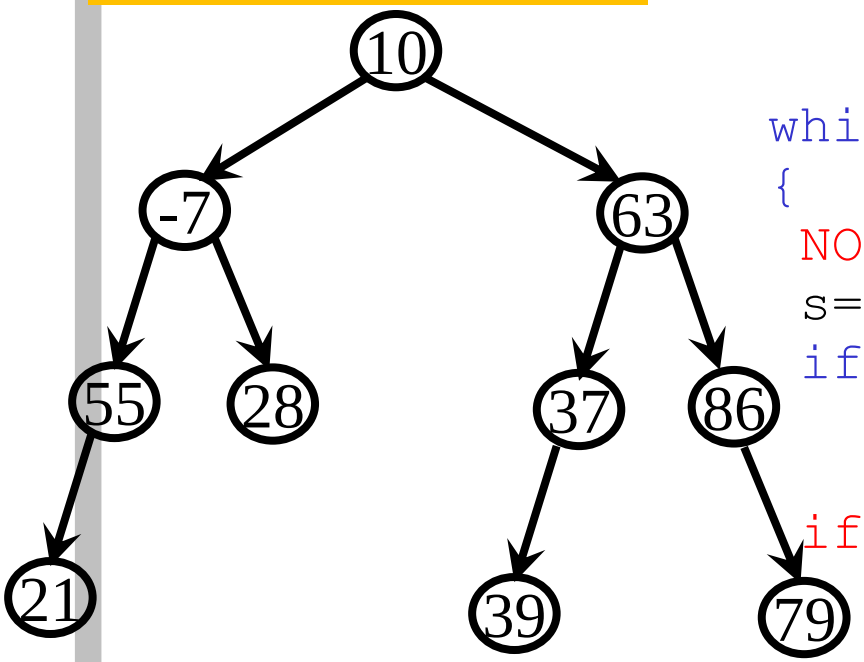
Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:




```
float s=0;
QUEUE q;
Init(q);
if (t!=NULL)
    EnQueue(q,t);
while(IsEmty(q)==0)
{
 NODE*p=DeQueue(q);
 s=s+p->info;
 if (p->pLeft)
   EnQueue(q,
           p->pLeft);
if (p->pRight)
 EnQueue(q,
           p->pRight);
}
```

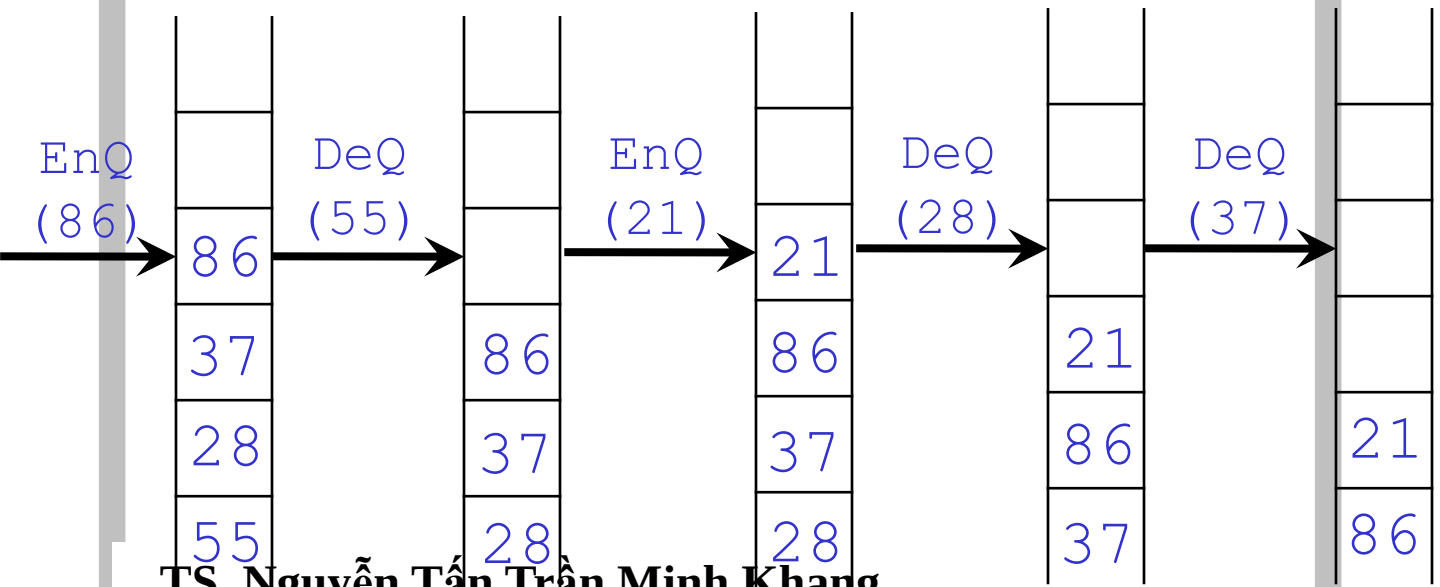


Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:

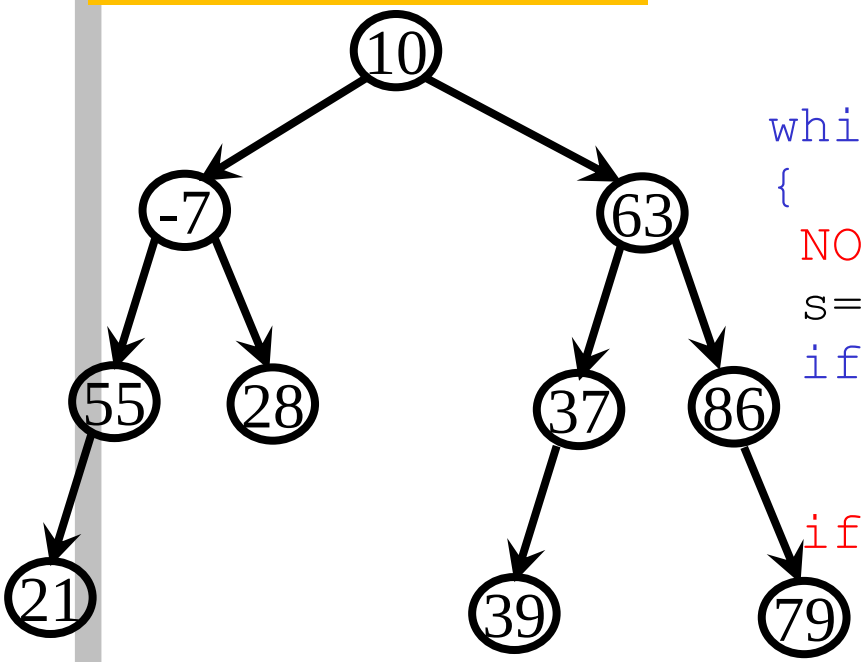



```
float s=0;
QUEUE q;
Init(q);
if (t!=NULL)
    EnQueue(q,t);
while(IsEmty(q)==0)
{
 NODE*p=DeQueue(q);
 s=s+p->info;
 if (p->pLeft)
   EnQueue(q,
           p->pLeft);
 if (p->pRight)
  EnQueue(q,
           p->pRight);
}
```

s=121 s=149 s=186

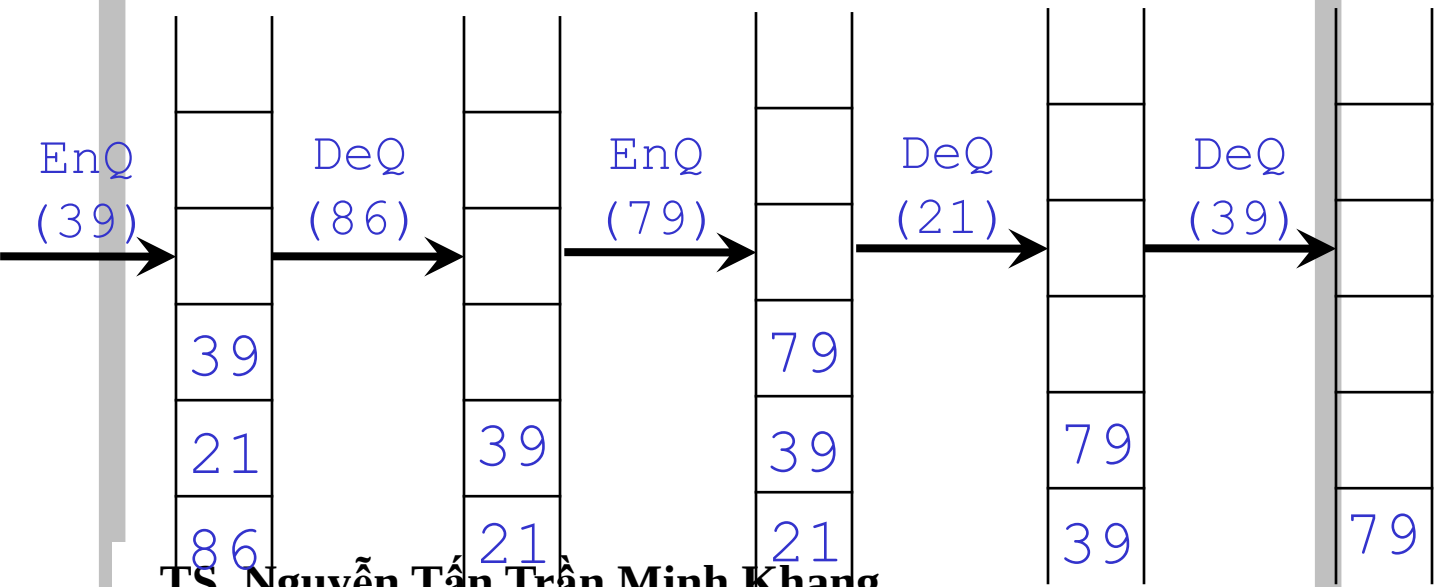


Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:

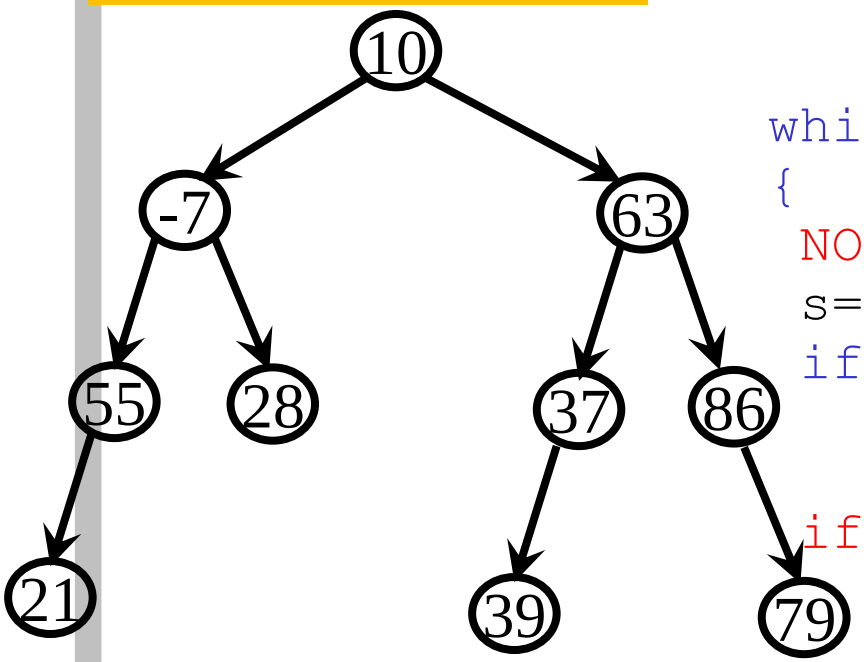



```
float s=0;
QUEUE q;
Init(q);
if (t!=NULL)
    EnQueue(q,t);
while(IsEmty(q)==0)
{
 NODE*p=DeQueue(q);
 s=s+p->info;
 if (p->pLeft)
   EnQueue(q,
           p->pLeft);
 if (p->pRight)
  EnQueue(q,
           p->pRight);
}
```

s=272 s=293 s=332

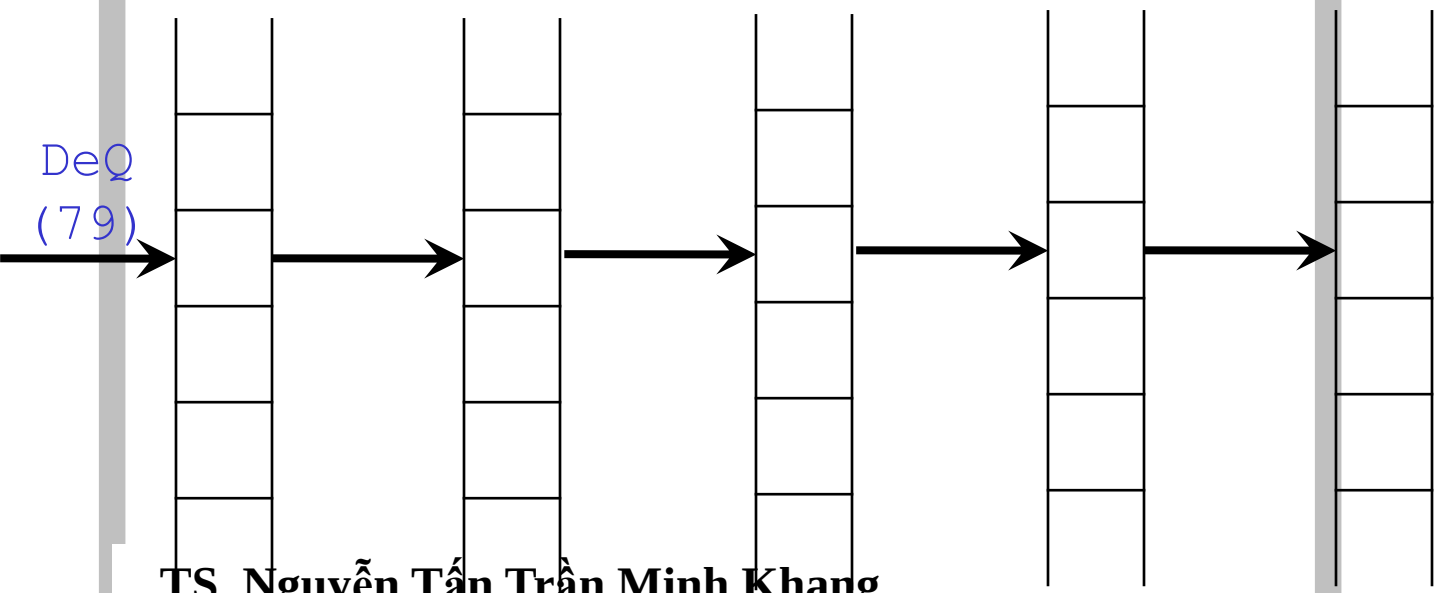


Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:




```
float s=0;
QUEUE q;
Init(q);
if (t!=NULL)
    EnQueue(q,t);
while(IsEmty(q)==0)
{
 NODE*p=DeQueue(q);
 s=s+p->info;
 if (p->pLeft)
   EnQueue(q,
           p->pLeft);
 if (p->pRight)
   EnQueue(q,
           p->pRight);
}
```

s=411



BÀI TẬP

- Làm tất cả các bài tập trong chương cây nhị phân bằng phương pháp khử đệ quy với kỹ thuật hàng đợi.

BÀI TẬP

- Cho một hàng đợi q và một đoạn chương trình như sau:

```
1. struct QUEUE q;
2. int x=5, y=3;
3. Enqueue(q,8);
4. Enqueue(q,9);
5. Enqueue(q,y);
6. Dequeue(q,x);
7. Enqueue(q,18);
8. Dequeue(q,x);
9. Enqueue(q,22);
10.while(IsEmpty(q)==0)
11.{
12.     Dequeue(q,y);
13.     printf("%d",y);
14.}
```

- Hãy cho biết kết quả in ra màn hình của đoạn chương trình trên.

BÀI TẬP

- Cho một hàng đợi q và một đoạn chương trình như sau:

```
1. struct QUEUE q;
```

- Hàng đợi rỗng

```
2. int x=5, y=3;
```

- x=5, y=3

```
3. Enqueue(q,8);
```

- Hàng đợi chứa (8)

```
4. Enqueue(q,9);
```

- Hàng đợi chứa (9, 8)

```
5. Enqueue(q,y);
```

- Hàng đợi chứa (3, 9, 8)

```
6. Dequeue(q,x);
```

- Hàng đợi chứa (3, 9) x=8, y=3

```
7. Enqueue(q,18);
```

- Hàng đợi chứa (18, 3, 9) x=8, y=3

```
8. Dequeue(q,x);
```

- Hàng đợi chứa (18, 3) x=9, y=3

```
9. Enqueue(q,22);
```

- Hàng đợi chứa (22, 18, 3) x=9, y=3

BÀI TẬP

- Hàng đợi chứa (22, 18, 3) x=9, y=3

```
10. while(IsEmpty(q)==0)
11. {
12.       Dequeue(q,y);
13.       printf("%d",y);
14. }
```

- **Lần lặp 1**
  1. **Dequeue(q,y);**
  2. Hàng đợi chứa (22, 18) x=9, y=3
  3. Xuất 3
- **Lần lặp 2**
  1. **Dequeue(q,y);**
  2. Hàng đợi chứa (22) x=9, y=18
  3. Xuất 18
- **Lần lặp 3**
  1. **Dequeue(q,y);**
  2. Hàng đợi chứa () x=9, y=22
  3. Xuất 22
- **Kết luận: Đoạn chương trên xuất 3 18 22.**