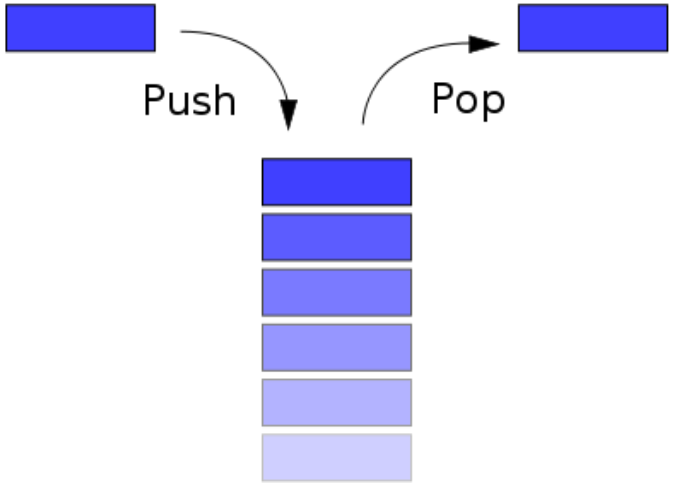
Bài 5. Ngăn xếp và hàng đợi

5.1.Ngăn xếp (Stack)

- Ngăn xếp là một kiểu danh sách được trang bị hai phép toán:
  - bổ sung một phần tử vào **cuối danh sách**
  - loại bỏ một phần tử cũng ở **cuối danh sách**



- Có thể hình dung ngăn xếp như hình ảnh một chồng đĩa, đĩa nào được đặt vào chồng sau cùng sẽ nằm trên tất cả các đĩa khác và sẽ được lấy ra đầu tiên.
- Vì nguyên tắc "vào sau ra trước" đó, Stack còn có tên gọi là danh sách kiểu **LIFO** (Last In First Out)
- Vị trí cuối danh sách được gọi là đỉnh (**top**) của Stack.

5.1.1. Biểu diễn ngăn xếp bằng mảng

- Khi mô tả Stack bằng mảng:
  - Bổ sung một phần tử vào Stack ~ thêm một phần tử vào cuối mảng.
  - Loại bỏ một phần tử khỏi Stack ~ loại bỏ một phần tử ở cuối mảng.
  - Stack bị tràn khi bổ sung vào mảng đã đầy
  - Stack là rỗng khi số phần tử thực sự đang chứa trong mảng = 0.

5.1.1. Biểu diễn ngăn xếp bằng mảng

```
#define max = 10000
int Stack[max];
int Top;

void StackInit() {
    Top = 0;
}

void Push(int V) {
    if (Top >= max-1)
        printf("Stack is full");
    else {
        Top++;
        Stack[Top] = V;
    }
}
```

```
int Pop() {
    if (Top == 0) {
        printf("Stack is empty");
        return -1;
    }
    else {
        int res = Stack[Top];
        Top--;
        return res;
    }
}
```

Độ phức tạp tính toán của các thao tác Init, Push, Pop đều là $\Theta(1)$

5.1.2. Biểu diễn ngăn xếp bằng danh sách nối đơn kiểu LIFO

- Khi cài đặt Stack bằng danh sách nối đơn kiểu LIFO, ta bỏ qua việc kiểm tra Stack tràn.

```
typedef struct node {
    int data;
    TNode *next;
} TNode;
TNode * Top;
void StackInit() {
    Top = NULL;
}
void Push(int V) {
    TNode * P = (TNode *)
malloc(sizeof(TNode));
    P->data = V;
    P->next = Top;
    Top = P;
}
```

```
int Pop() {
if (Top == NULL) {
    print('Stack is empty');
    return -1;
}
else {
    int res = Top->data;
    TNode * P = Top->next;
    free(Top);
    Top = P;
    return res;
}
}
```

5.1.3. Ngăn xếp – ứng dụng

- Đổi số nguyên từ hệ thập phân sang hệ nhị phân. Ví dụ 23_{10} = $\mathbf{10111}_2$
- Thuật toán
  1. Input số n
  2. Nếu n=0 nhảy đến bước 4, nếu không thực hiện bước 3
  3. In số dư của n khi chia cho 2 (d = n%2) và gán n = n/2
  4. Kết thúc
- Tuy nhiên với thuật toán như vậy thứ tự các chữ số trong hệ nhị phân **bị ngược**. Thật vậy:

5.1.3. Ngăn xếp – ứng dụng

- Ví dụ đổi số n = 23 sang hệ nhị phân, theo thuật toán trên:

  n=23 ≠ 0, in ra 23 % 2 = **1**, gán n = 23/2 =11
  n=11 ≠ 0, in ra 11 % 2 = **1**, gán n = 11/2 = 5

  n=5 ≠ 0, in ra 5 % 2 = **1**, gán n = 5/2 = 2
  n=2 ≠ 0, in ra 2 % 2 = **0**, gán n = 2/2 = 1
  n=1 ≠ 0, in ra 1 % 2 = **1**, gán n = 1/2 = 0

  n=0, thuật toán kết thúc, ta thu được dãy chữ số **11101**
- Mà trên thực tế 23_{10} = $\mathbf{10111}_2$
- Để giải quyết tình trạng thứ tự ngược, ta có thể lợi dụng tính chất vào sau, ra trước của ngăn xếp

5.1.3. Ngăn xếp – ứng dụng

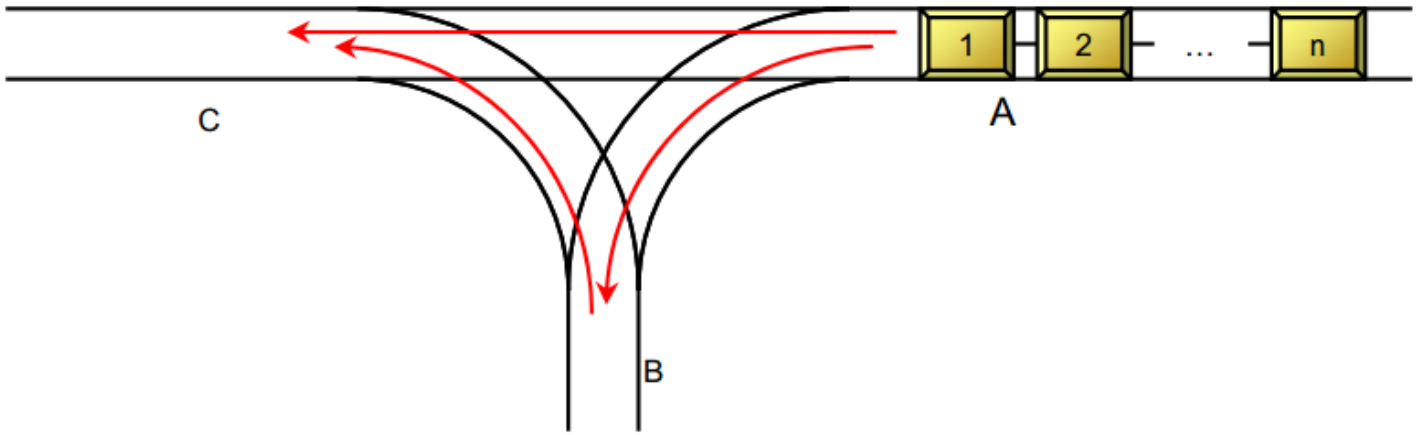
- Hàm in số n ở dạng nhị phân

```
void outputInBinary(int n) {
    while (n > 0) {
        int bit = n % 2;
        push(bit); // đẩy vào stack
        n = n / 2;
}
while <stack không rỗng> {
     printf("%d",pop()); // lấy ra khỏi stack
}
```

5.1.3. Ngăn xếp – ứng dụng

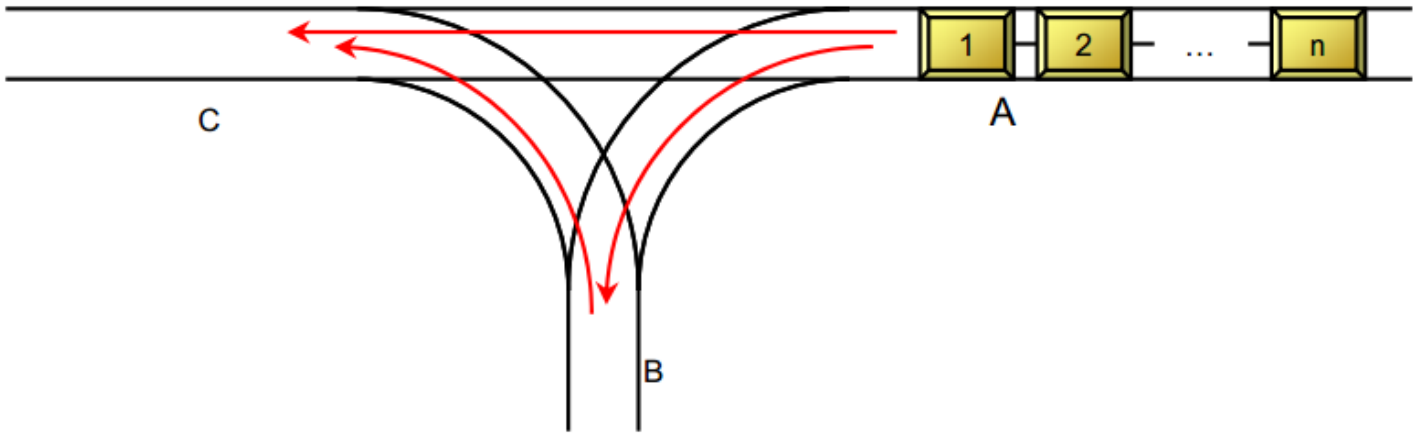
- Có thể dùng cơ chế stack để khử đệ quy cho tất cả các hàm đệ quy
- Bài toán tháp Hà Nội: mỗi cọc trong số 3 cọc đều có mô hình của một stack
- Stack có thể được dùng để phân tích và tính giá trị của biểu thức đại số, vi dụ: (2 * 5 - 1 * 2) / (11 - 9) (kết quả: 4), hoặc đổi ký pháp trung tố thành hậu tố (kết quả: 2 5 * 1 2 * - 11 9 - /)
- Bài toán tìm kiếm theo chiều sâu trên đồ thị
- …

5.1.3. Ngăn xếp – ứng dụng



- Một đoàn tàu chứa n toa tàu đánh số từ 1 đến n đang nằm trên đường ray A. Do nhu cầu vận chuyển, người ta muốn sắp xếp lại các toa tàu theo một thứ tự nào đó và chuyển đoàn tàu với thứ tự toa mới qua đường ray C. Thứ tự mới này sẽ là một hoán vị của (1, 2, ..., n). Quy tắc chuyển như sau: chỉ được đưa các toa tàu chạy theo đường ray theo hướng mũi tên, có thể dùng đoạn đường ray B (đường tránh) để chứa tạm các toa tàu trong quá trình di chuyển.

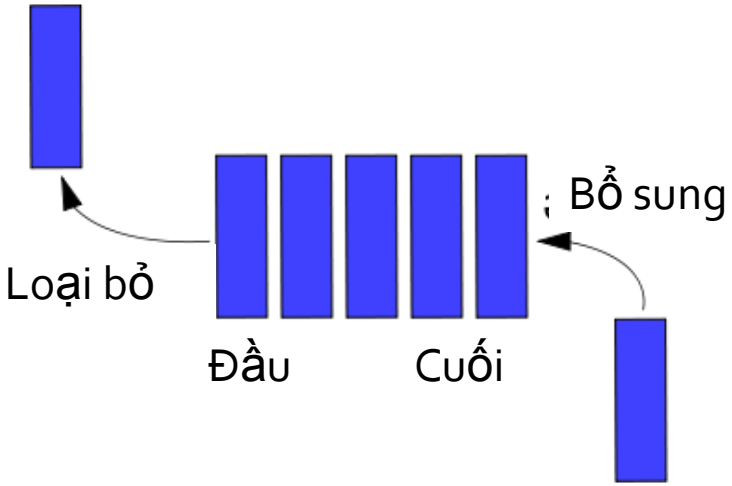
5.1.3. Ngăn xếp – ứng dụng



- Bài toán đặt ra là nhập vào thứ tự mới cần có của các toa tàu, cho biết có phương án chuyển hay không, và nếu có hãy đưa ra cách chuyển.
- Ví dụ: n = 4; Thứ tự cần có (1, 4, 3, 2)
  1)A→C; 2)A→B; 3)A→B; 4)A→C; 5)B→C; 6)B→C
- Hoặc tổng quát hơn:liệt kê tất cả các hoán vị của thứ tự các toa có thể tạo thành trên đoạn đường ray C với luật di chuyển như trên

5.2. Hàng đợi (Queue)

- Hàng đợi là một kiểu danh sách được trang bị hai phép toán:
  - bổ sung một phần tử vào **cuối danh sách** (Rear)
  - loại bỏ một phần tử ở **đầu danh sách** (Front)



- Có thể hình dung hàng đợi như một đoàn người xếp hàng mua vé: Người nào xếp hàng trước sẽ được mua vé trước và ra khỏi hàng để nhường vị trí cho người xếp hàng ngay phía sau.
- Vì nguyên tắc "vào trước ra trước" đó, Queue còn có tên gọi là danh sách kiểu **FIFO** (First In First Out).

5.2.1. Mô tả hàng đợi bằng mảng

- Khi mô tả Queue bằng mảng, ta có hai chỉ số Front và Rear lưu chỉ số phần tử đầu và chỉ số cuối Queue, khởi tạo Queue rỗng: Front = 1 và Rear = 0;
  - Để thêm một phần tử vào Queue, ta tăng Rear lên 1 và đưa giá trị đó vào phần tử thứ Rear.
  - Để loại một phần tử khỏi Queue, ta lấy giá trị ở vị trí Front và tăng Front lên 1.
  - Khi Rear tăng lên hết khoảng chỉ số của mảng thì mảng đã đầy, không thể đẩy thêm phần tử vào nữa.
  - Khi Front > Rear thì tức là Queue đang rỗng
- Như vậy chỉ một phần của mảng từ vị trí Front tới Rear được sử dụng làm Queue.

5.2.1. Mô tả hàng đợi bằng mảng

```
#define max = 10000
int Queue[max];
int Front, Rear;

void QueueInit {
  Front = 1; Rear = 0;
}

void Push(int V) {
   if (Rear >= max-1)
      printf("Queue is full");
   else {
      Rear++;
      Queue[Rear] = V;
   }
}
```

```
int Pop() {
   if (Front > Rear) {
      printf("Queue is empty");
     return -1;
   }
   else {
      int res = Queue[Front];
      Front++;
      return res;
   }
}
```

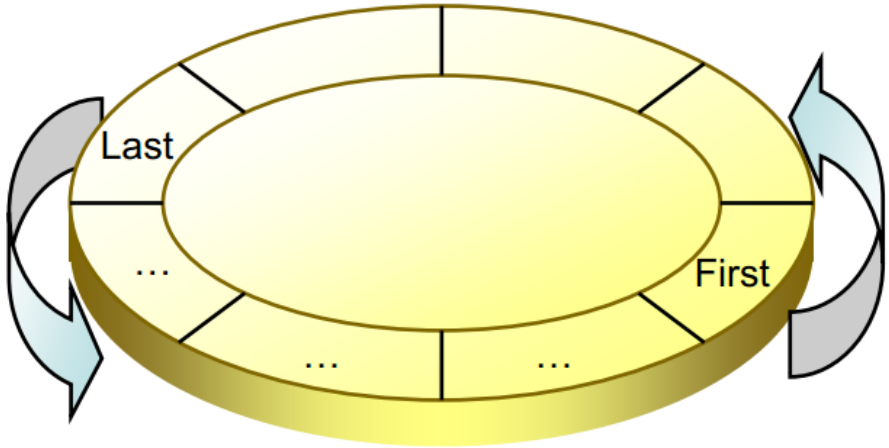
Độ phức tạp tính toán của các thao tác Init, Push, Pop đều là Θ(1)

5.2.1. Mô tả hàng đợi bằng mảng

- Nhược điểm:
  - Trong cách cài đặt bằng mảng ta thấy các chỉ số Rear và Front chỉ tăng lên chứ không giảm đi, kể cả khi lấy các phần tử ra khỏi hàng đợi: Chỉ có các phần tử từ vị trí Front tới Rear là thuộc Queue, các phần tử từ vị trí 1 tới Front - 1 bị lãng phí
  - Nhược điểm này không có khi cài đặt ngăn xếp (vì chỉ số Top trong ngăn xếp tăng rồi lại giảm trong các thao tác thêm vào/lấy ra)
- Để khắc phục nhược điểm này: biểu diễn bằng danh sách vòng

5.2.2. Mô tả hàng đợi danh sách vòng

Danh sách vòng biểu diễn hàng đợi có thể cài đặt bằng mảng hoặc danh sách liên kết. Ở đây ta xét phép cài đặt bằng mảng:



- Coi như các phần tử của mảng được xếp quanh vòng theo một hướng nào đó
- Các phần tử nằm trên phần cung tròn từ vị trí Front tới vị trí Rear là các phần tử của Queue.
- Thêm một phần tử vào Queue: dịch chỉ số Rear theo vòng một vị trí rồi đặt giá trị mới vào đó
- Loại bỏ một phần tử trong Queue: lấy ra phần tử tại vị trí Front rồi dịch chỉ số Front theo vòng

5.2.2. Mô tả hàng đợi danh sách vòng

```
#define max = 10000
int Queue[max];
int Front, Rear, i, n;

void QueueInit {
    n = Front = 0;
    Rear = max - 1;
}

void Push(int V) {
    if (n>=max)
        printf("Queue's full");
    else {
        Rear = (Rear+1) % max;
        Queue[Rear] = V;
        n++;
    }
}
```

```
int Pop() {
    if (n == 0) {
        printf("Queue is Empty");
        return -1;
    }
    else
    {
        int res = Queue[Front];
        Front = (Front+1)% max;
        n--;
        return res;
    }
}
```

Độ phức tạp tính toán của các thao tác Init, Push, Pop đều là Θ(1)

5.2.3. Mô tả hàng đợi bằng danh sách nối đơn kiểu FIFO

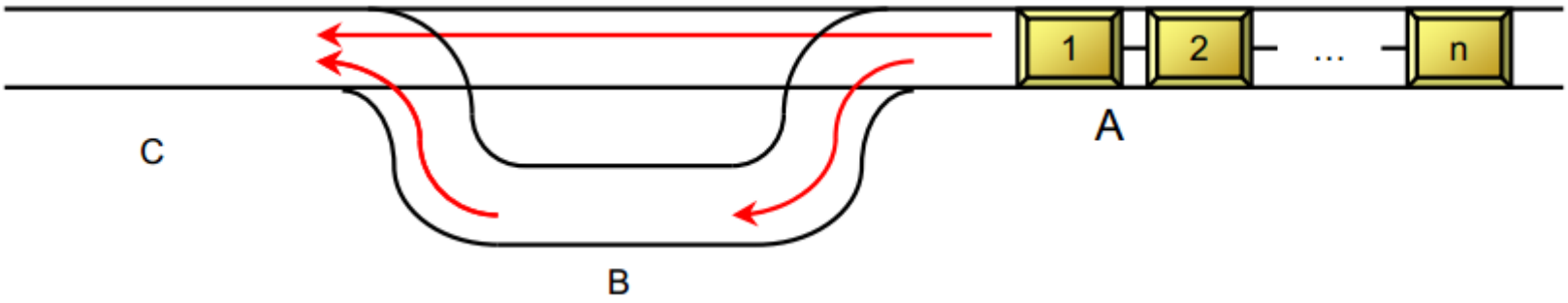
```
typedef struct node {
    int data;
    TNode *next;
} TNode;
TNode * Front, Rear;
void QueueInit(){
    Front = NULL;
}
void Push(int V) {
    TNode * P = (TNode *)
malloc(sizeof(TNode));
    P->data = V;
    if (Front==NULL)
      Front = P;
    else Rear->next = P;
    Rear = P;
}
```

```
int Pop() {
    if (Front == NULL) {
       printf("Queue is empty");
       return -1;
    } else {
       int res = Front->data;
       TNode * P = Front->next;
       free(Front);
       Front := P;
       return res;
    }
}
```

5.2.4. Hàng đợi - ứng dụng

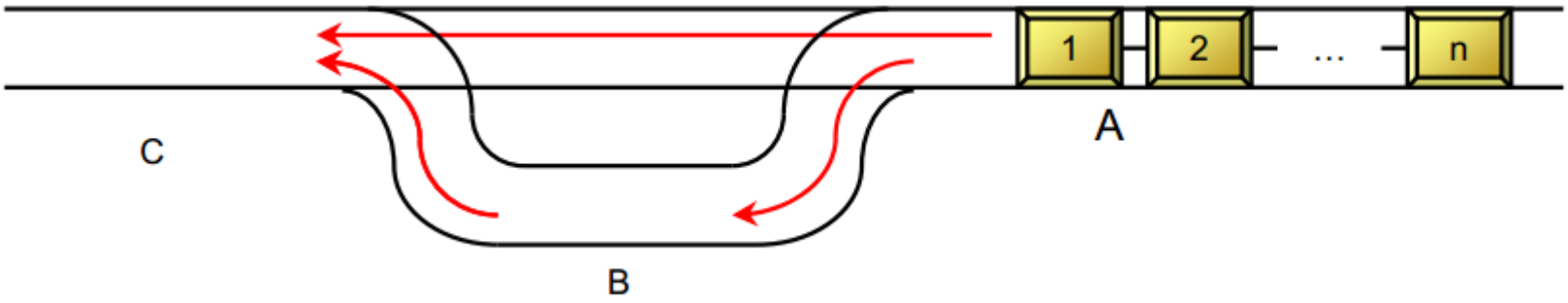
- Bài toán tìm kiếm theo chiều rộng trên đồ thị,
- Thuật toán loang
- Queue được dùng trong hầu hết các hệ điều hành để xếp thứ tự các tiến trình truy cập vào cùng một tài nguyên,
- Dùng trong hệ thống mạng để xếp hàng chờ cho các gói tin trước khi định tuyến chúng đến hướng đích,
- Một biến thể của hàng đợi là hàng đợi có độ ưu tiên (priority queue hay còn gọi là heap) được dùng
  - Trong thuật toán sắp xếp rất hiệu quả là thuật toán sắp xếp kiểu vun đống (heap sort),
  - Dùng trong bài toán quản lý băng thông,
  - Bài toán tìm đường đi ngắn nhất trong đồ thị có trọng số,
  - Trong một số thuật toán mã hóa và giải mã (như thuật toán Huffman).

5.2.4. Hàng đợi - ứng dụng



- Một đoàn tàu chứa n toa tàu đánh số từ 1 đến n đang nằm trên đường ray A. Do nhu cầu vận chuyển, người ta muốn sắp xếp lại các toa tàu theo một thứ tự nào đó và chuyển đoàn tàu với thứ tự toa mới qua đường ray C. Thứ tự mới này sẽ là một hoán vị của (1, 2, ..., n). Quy tắc chuyển như sau: chỉ được đưa các toa tàu chạy theo đường ray theo hướng mũi tên, có thể dùng đoạn đường ray B (đường tránh) để chứa tạm các toa tàu trong quá trình di chuyển.

5.2.4. Hàng đợi - ứng dụng



- Bài toán đặt ra là nhập vào thứ tự mới cần có của các toa tàu, cho biết có phương án chuyển hay không, và nếu có hãy đưa ra cách chuyển.
- Ví dụ: n = 4; Thứ tự cần có (1, 4, 2, 3)
  1)A→C; 2)A→B; 3)A→B; 4)A→C; 5)B→C; 6)B→C
- Hoặc tổng quát hơn:liệt kê tất cả các hoán vị của thứ tự các toa có thể tạo thành trên đoạn đường ray C với luật di chuyển như trên