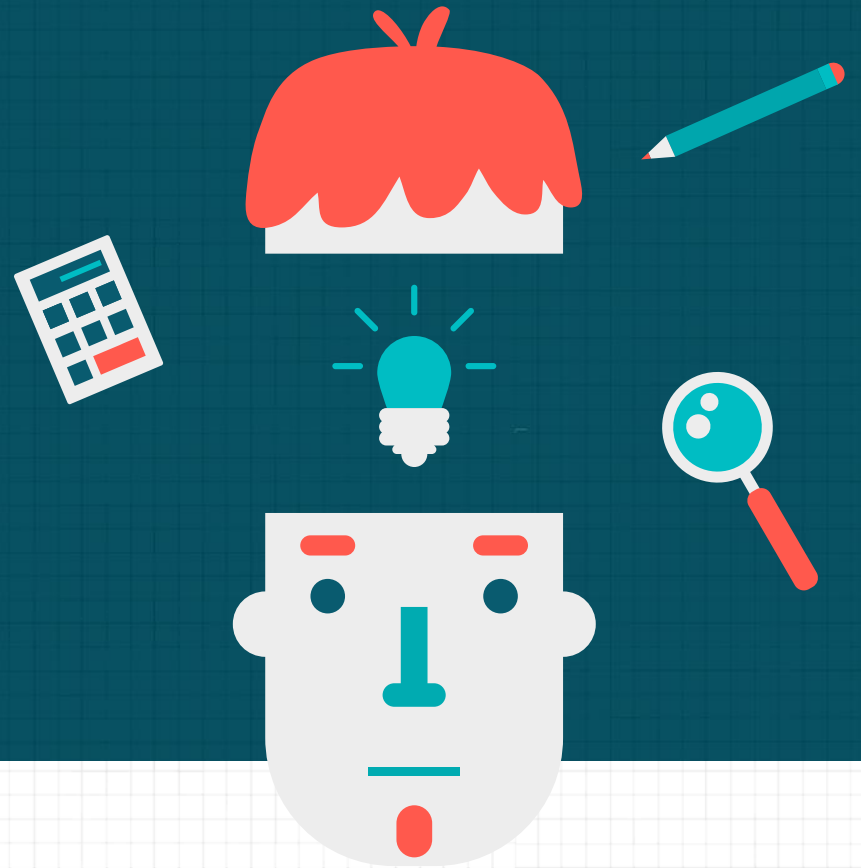
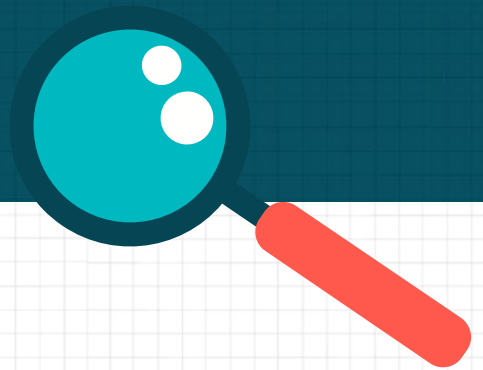
Bài 5
Danh sách liên kết

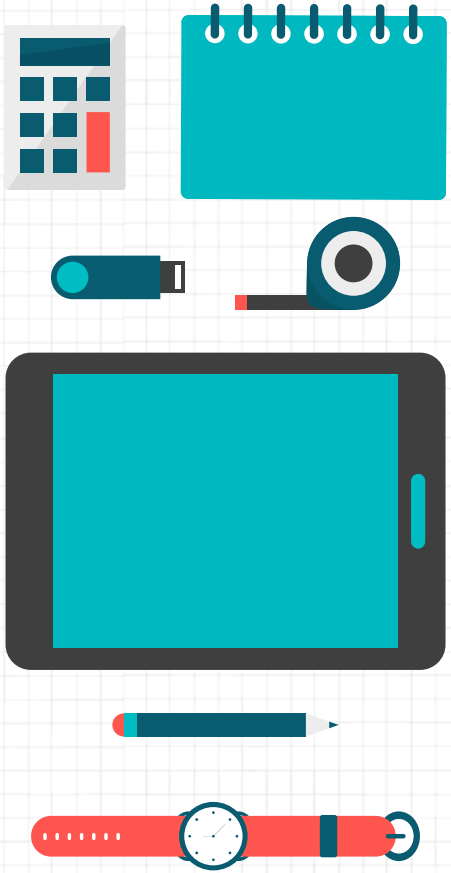


Ths. Phạm Minh Hoàng

Nội dung



- Các khái niệm cơ bản
- Các thao tác trên danh sách liên kết đơn
- Ngăn xếp – hàng đợi
- Ứng dụng



01

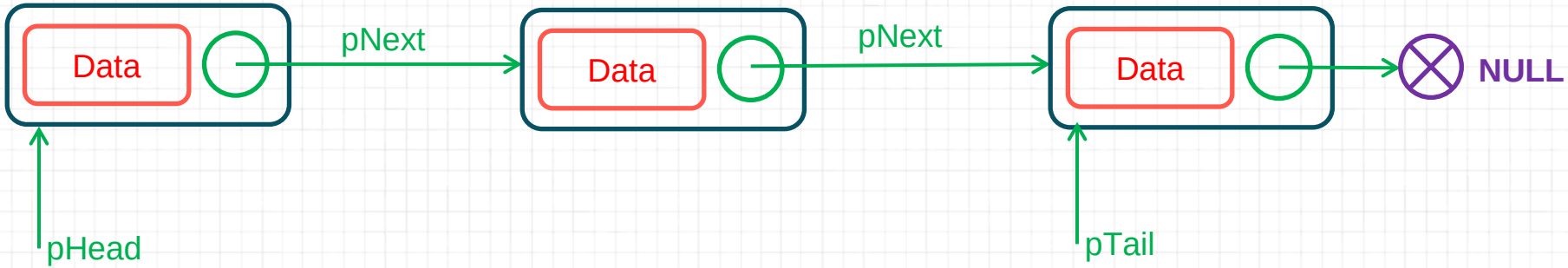
Các khái niệm cơ bản

Vấn đề của mảng

- Tính chất mảng
  - Các phần tử được lưu liên tiếp nhau trên bộ nhớ
  - Kích thước của mảng là cố định
- Ưu điểm
  - Truy xuất ngẫu nhiên → nhanh chóng
  - Cấp phát và giải phóng vùng nhớ nhanh chóng
- Khuyết điểm
  - Thêm xóa phần tử không thuận tiện
  - Khó tìm vùng nhớ liên tiếp nếu số lượng phần tử lớn

Danh sách liên kết là gì

- DSLK đơn là cấu trúc dữ liệu
  - Dãy các phần tử không nằm liên tiếp nhau trên bộ nhớ
  - Mỗi phần tử gồm 2 thành phần: dữ liệu + con trỏ tới phần tử kế tiếp
  - Con trỏ của phần tử cuối danh sách giữ địa chỉ NULL
  - Để quản lý 1 DSLK đơn chỉ cần giữ địa chỉ phần tử đầu



Khai báo DSLK

- Khai báo cấu trúc phần tử (node) trong DSLK đơn

```
struct Node {
    int data;
    Node* pNext;
}
```

```
typedef struct Node* ptrNode;
struct Node {
    int data;
    ptrNode pNext;
}
```

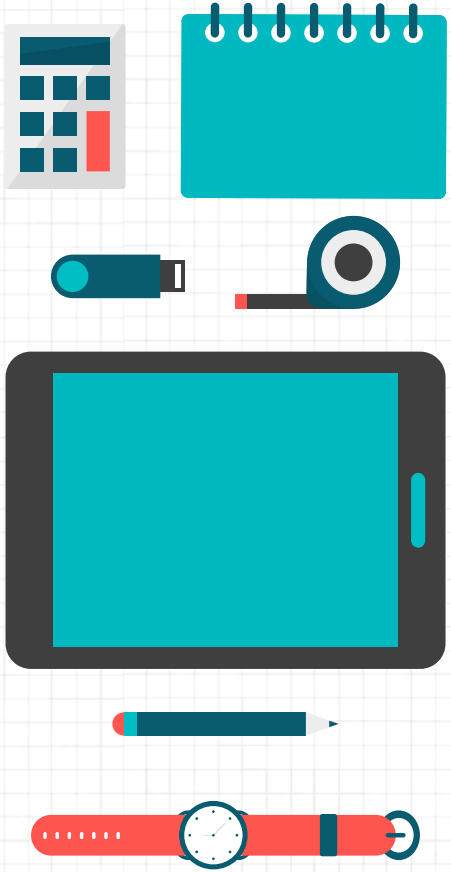
- Khai báo cấu trúc danh sách liên kết

```
struct List {
    ptrNode pHead;
}
```

```
struct List {
    ptrNode pHead;
    ptrNode pTail;
}
```

Phân loại DSLK

- Các loại DSLK
  - DSLK đơn: mỗi node chứa dữ liệu và con trỏ đến node tiếp theo
  - DSLK kép: mỗi node chứa dữ liệu và 2 con trỏ: 1 con trỏ đến node tiếp theo và 1 con trỏ đến node trước đó
  - DSLK vòng: con trỏ của node cuối cùng trỏ đến node đầu tiên



02

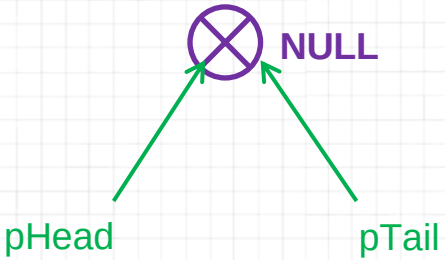
Các thao tác trên DSLK đơn

Các thao tác trên DSLK

- Các thao tác trên DSLK
  - Khởi tạo danh sách rỗng
  - Kiểm tra danh sách có rỗng hay không
  - Tạo 1 node mới
  - Thêm node mới
  - Tạo DSLK bằng cách thêm phần tử
  - Duyệt DSLK
  - Xóa node trong DSLK
  - Xóa DSLK
  - Tìm phần tử trong DSLK

Khởi tạo danh sách rỗng

- DSLK rỗng không có bất kỳ node nào
- Các con trỏ pHead, pTail giữ địa chỉ NULL




```
void initList(List &l){
    l.pHead = NULL;
    l.pTail = NULL;
}
```

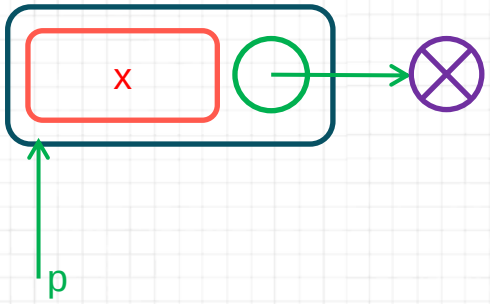
Kiểm tra DSLK rỗng

- Kiểm tra con trỏ pHead, pTail có NULL hay không

```
int isEmpty(List l){
    return (l.pHead == NULL && l.pTail == NULL);
}
```

Tạo node mới

- Tạo node p với dữ liệu x cho trước, pNext giữ địa chỉ NULL




```
ptrNode createNode(int x){
    ptrNode p = new Node;
    p->data = x;
    p->pNext = NULL;
    return p;
}
```

Thêm node mới

- Thêm node mới
  - Đầu danh sách

```
void insertHead(List &l, int x);
```

  - Cuối danh sách

```
void insertTail(List &l, int x);
```

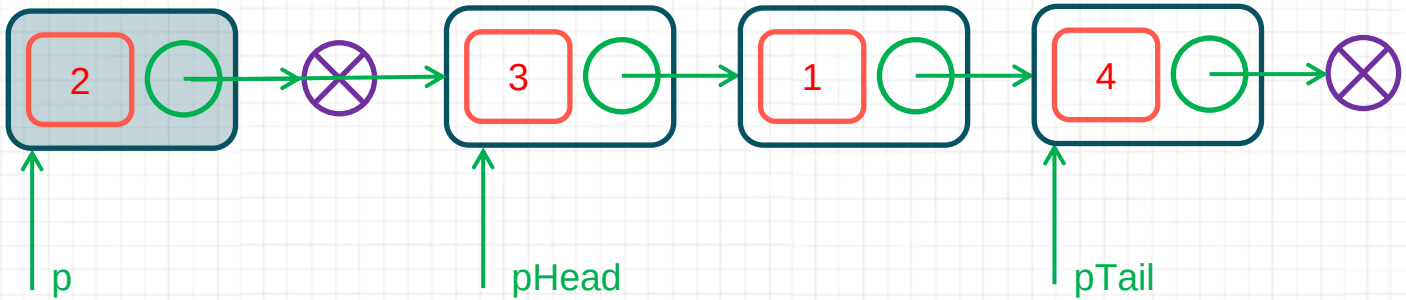
  - Sau node q

```
void insertAfter(List &l, ptrNode q, int x);
```

  - Trước node q

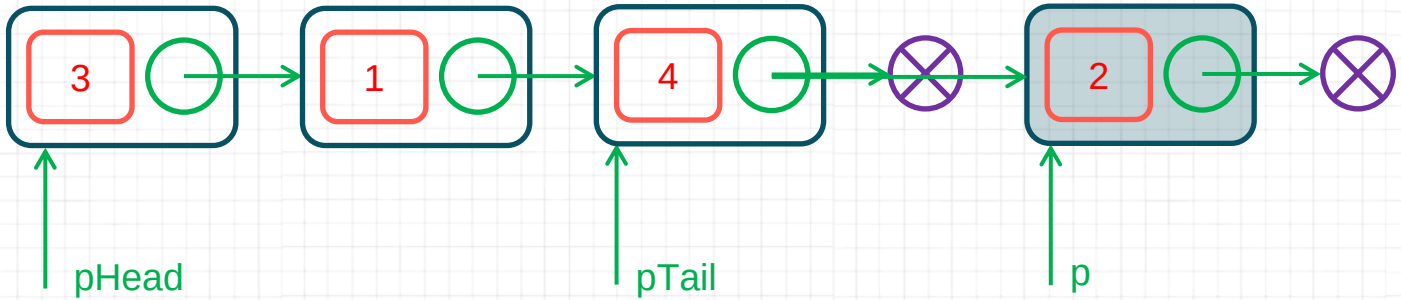
```
void insertBefore(List &l, ptrNode q, int x);
```

Thêm node đầu DSLK



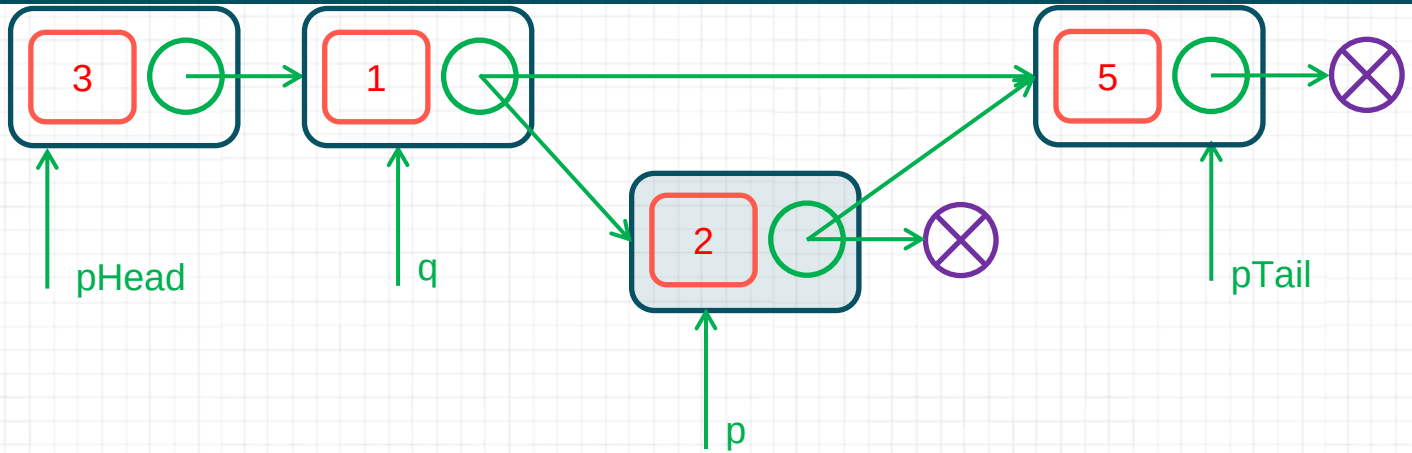

```
void insertHead(List& l, int x){
    ptrNode p = createNode(x);
    if(isEmpty(l))
        l.pHead = l.pTail = p;
    else{
        p->pNext = l.pHead;
        l.pHead = p;
    }
}
```

Thêm node cuối DSLK




```
void insertTail(List& l, int x){
    ptrNode p = createNode(x);
    if(isEmpty(l))
        l.pHead = l.pTail = p;
    else{
        l.pTail->pNext = p;
        l.pTail = p;
    }
}
```

Thêm node sau node q




```
void insertAfter(List& l, ptrNode q, int x){
    ptrNode p = createNode(x);
    p->pNext = q->pNext;
    q->pNext = p;
    if(l.pTail == q)
        l.pTail = p;
}
```

Thêm node trước node q

```
void insertBefore(List& l, ptrNode q, int x){
    ptrNode p = createNode(x);
    if(l.pHead == q)
        insertHead(l, p);
    else{
        *p = *q;
        q->pNext = p;
        q->data = x;
        if(l.pTail == q)
            l.pTail = p;
    }
}
```

Tạo DSLK

- Tạo DSLK từ mảng a có n phần tử bằng cách
  - Khởi tạo DSLK rỗng
  - Thêm lần lượt từng phần tử của mảng a vào đầu DSLK
  - Thêm lần lượt từng phần tử của mảng a vào cuối DSLK

```
List createList(int a[], int n);
```

Duyệt DSLK

```
void print(List l){
    ptrNode p = l.pHead;
    while(p){
        cout << p->data << " ";
        p = p->pNext;
    }
}
```

```
void print(List l){
    for(ptrNode p = l.pHead; p; p=p->pNext)
        cout << p->data << " ";
}
```

Xóa node

- Xóa node trong DSLK
  - Xóa node pHead

```
void deleteHead(List &l);
```

  - Xóa node pTail

```
void deleteTail(List &l);
```

  - Xóa node q

```
void deleteNode(List &l, ptrNode q);
```

Xóa node đầu DSLK

```
void deleteHead(List& l){
    if(isEmpty(l)){
        if(l.pHead == l.pTail){
            delete l.pHead;
            l.pHead = l.pTail = NULL;
        }
        else{
            ptrNode p = l.pHead;
            l.pHead = l.pHead->pNext;
            delete p;
        }
    }
}
```

Xóa node cuối DSLK

```
void deleteTail(List& l){
    if(l.pHead == l.pTail){
        delete l.pTail;
        l.pHead = l.pTail = NULL;
    }
    else{
        ptrNode p = l.pHead;
        while(p->pNext != l.pTail)
            p = p->pNext;
        delete l.pTail;
        p->pNext = NULL;
        l.pTail = p;
    }
```

Xóa node q

```
void deleteNode(List& l, ptrNode q){
    if(q != l.pTail){
        ptrNode r = q->pNext;
        *q = *r;
        delete r;
    }
}
```

Xóa DSLK

- Sau khi xử lý xong phải giải phóng vùng nhớ cho DSLK
- Xóa DSLK bằng cách xóa lần lượt từng node

```
void destroyList(List &l){
    ptrNode q;
    ptrNode p = l.pHead;
    while(p){
        q = p;
        p = p->pNext;
        delete q;
    }
    l.pHead = l.pTail = NULL;
}
```

Tìm kiếm phần tử trong DSLK

- Tìm phần tử
  - Có dữ liệu x

```
ptrNode findItemByData(List l, int x);
```

  - Thứ i trong DSLK

```
ptrNode findItemByIndex(List l, int i);
```

Mảng vs DSLK

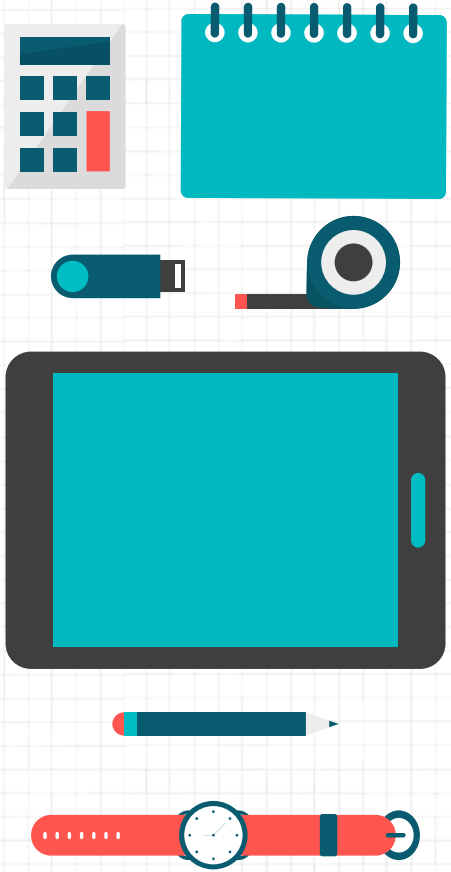
	DSLK	Mảng
Lưu phần tử	Không liên tiếp	Liên tiếp nhau
Thay đổi Kích thước	Tùy ý Độ phức tạp O(1)	Cấp phát lại + sao chép Độ phức tạp O(n)
Truy xuất phần tử	Tuần tự Độ phức tạp O(n)	Ngẫu nhiên Độ phức tạp O(1)
Thêm/xóa phần tử	Không dời phần tử Đa số độ phức tạp O(1)	Phải dời phần tử Độ phức tạp O(n)

Mảng vs DSLK

	DSLK	Mảng
Tìm kiếm	Không quay lui được Độ phức tạp O(n)	Tiến và lùi được Độ phức tạp O(n)
Hao phí bộ nhớ	Mỗi phần tử phải lưu thêm địa chỉ của phần tử kế tiếp Tận dụng vùng nhớ do phần tử nằm rời rạc nhau	Mỗi phần tử chỉ cần lưu dữ liệu Khó tìm đủ vùng nhớ liên tiếp nếu cấp phát nhiều phần tử

Bài tập

- Cho số nguyên dương N. Tạo DSLK đơn L có N phần tử bằng cách cho người dùng nhập các phần tử từ bàn phím và thêm phần tử vào cuối DSLK
  - Tìm phần tử lớn nhất/nhỏ nhất/chính giữa trong DSLK
  - Xóa phần tử đứng trước phần tử nhỏ nhất
  - Xóa phần tử đứng sau phần tử lớn nhất
  - Thêm phần tử mới vào chính giữa DSLK
  - Đảo ngược DSLK
  - Tách DSLK L thành 2 DSLK mới L1 và L2: DSLK L1 gồm toàn số lẻ từ L, DSLK L2 gồm toàn số chẵn từ L. Thứ tự các phần tử không đổi
  - Tại mỗi bước in DSLK kết quả ra màn hình

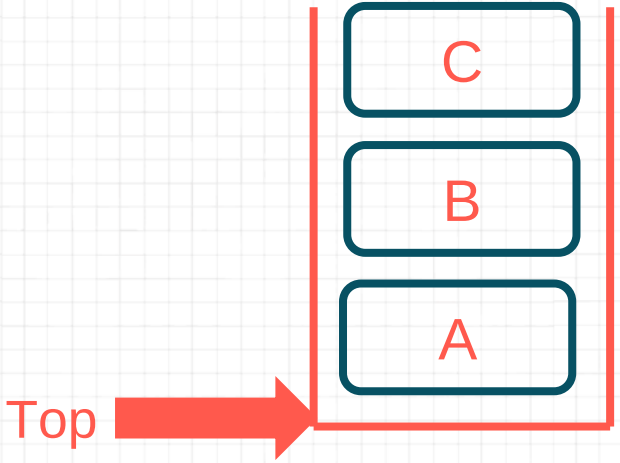


03

Ngăn xếp & hàng đợi

Ngăn xếp

- Ngăn xếp là cấu trúc dùng để lưu các phần tử theo nguyên tắc LIFO (Last In First Out): phần tử đưa vào cuối cùng sẽ được lấy ra đầu tiên
- Cài đặt ngăn xếp
  - Sử dụng mảng
  - Sử dụng DSLK

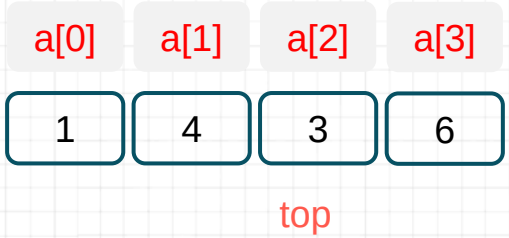


Ngăn xếp

- Các thao tác trên ngăn xếp
  - Khởi tạo ngăn xếp
  - Kiểm tra ngăn xếp rỗng/đầy
  - Đẩy phần tử mới vào đỉnh ngăn xếp (push)
  - Lấy phần tử ở đỉnh ngăn xếp (pop)
  - Đọc nội dung phần tử ở đỉnh ngăn xếp
  - Hủy ngăn xếp

Cài đặt ngăn xếp bằng mảng

- Dùng 1 biến chỉ số top để đánh dấu vị trí đỉnh ngăn xếp
- Push: thêm phần tử và tăng top lên
- Pop: lấy phần tử tại vị trí top, và giảm top xuống



Cài đặt ngăn xếp bằng mảng

- Khai báo ngăn xếp
  - Bằng mảng tĩnh

```
#define MAX_SIZE 100
struct Stack{
    int data[MAX_SIZE];
    int top;
}
```

  - Bằng mảng động

```
struct Stack{
    int* data;
    int size;
    int top;
}
```

Cài đặt ngăn xếp bằng mảng

- Khởi tạo ngăn xếp có kích thước size

```
Stack initStack(int size){
    Stack s;
    s.size = size;
    s.data = new int[size];
    s.top = 0;
    return s;
}
```

Cài đặt ngăn xếp bằng mảng

- Kiểm tra ngăn xếp rỗng/đầy

```
int isEmpty(Stack s){
    return s.top == 0;
}
```

```
int isFull(Stack s){
    return s.top == s.size - 1;
}
```

Cài đặt ngăn xếp bằng mảng

- Lấy 1 phần tử (pop) từ đỉnh stack ra

```
int pop(Stack s){
    item = s.data[s.top];
    s.top--;
    return item;
}
```

- Đưa 1 phần tử (push) vào stack

```
int push(Stack s, int k){
   if(isFull(s))
       return 0;
   s.top++;
   s.data[s.top] = k;
   return 1;
}
```

Cài đặt ngăn xếp bằng mảng

- Đọc nội dung phần tử ở đỉnh ngăn xếp

```
int top(Stack s){
    return s.data[s.top];
}
```

- Hủy ngăn xếp

```
void deleteStack(Stack& s){
    if(s.data != NULL)
        delete[] s.data;
}
```

Cài đặt ngăn xếp bằng DSLK

- Dùng DSLK để lưu các phần tử trong stack
- Sử dụng con trỏ pHead để làm phần tử top trên đỉnh stack
- Push: thêm node mới vào đầu DSLK
- Pop: lấy node ở đầu ra khỏi DSLK

Cài đặt ngăn xếp bằng DSLK

- Khai báo ngăn xếp

```
typedef struct Node* ptrNode;
struct Node{
   int data;
   ptrNode pNext;
}
```

```
struct Stack{
   ptrNode top;
}
```

Cài đặt ngăn xếp bằng DSLK

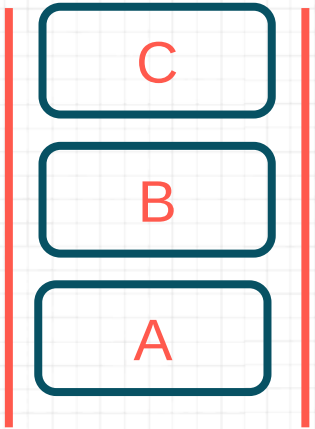
- Dùng DSLK để lưu các phần tử trong stack
- Sử dụng con trỏ pHead để làm phần tử top trên đỉnh stack
- Push: thêm node mới vào đầu DSLK
- Pop: lấy node ở đầu ra khỏi DSLK

Cài đặt ngăn xếp bằng DSLK

- Khởi tạo stack
- Kiểm tra stack rỗng/đầy
- Push: đẩy phần tử mới vào đỉnh stack
- Pop: lấy phần tử từ đỉnh stack
- Đọc nội dung đỉnh stack
- Hủy ngăn xếp

Hàng đợi

- Hàng đợi (queue) là cấu trúc dùng để lưu các phần tử theo nguyên tắc FIFO (First In First Out): phần tử đưa vào đầu tiên sẽ được lấy ra đầu tiên
- Cài đặt hàng đợi
  - Sử dụng mảng
  - Sử dụng DSLK
- Phân loại
  - Hàng đợi truyền thống
  - Hàng đợi 2 đầu
  - Hàng đợi ưu tiên



Hàng đợi truyền thống

- Khai báo hàng đợi bằng mảng

```
struct Queue{
    int* data;
    int size;
    int head;
    int tail;
}
```

Hàng đợi truyền thống

- Khai báo hàng đợi bằng DSLK

```
typedef struct Node* ptrNode;
struct Node{
    int data;
    ptrNode pNext;
}
```

```
struct Queue{
    ptrNode pHead;
    ptrNode pTail;
}
```

Hàng đợi truyền thống

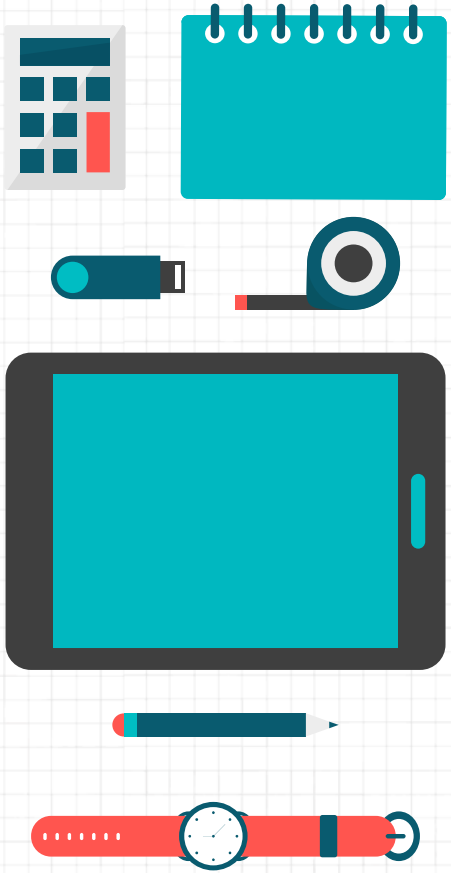
- Các thao tác trên hàng đợi truyền thống
  - Khởi tạo hàng đợi
  - Kiểm tra hàng đợi rỗng/đầy
  - Đẩy phần mới tử vào hàng đợi (push)
  - Lấy phần tử ở đỉnh hàng đợi (pop)
  - Đọc nội dung phần tử ở đỉnh hàng đợi
  - Hủy hàng đợi

Hàng đợi hai đầu

- Hàng đợi hai đầu cho phép thực hiện push và pop phần tử ở cả hai đầu của hàng đợi:
- Các thao tác
  - Khởi tạo hàng đợi
  - Kiểm tra hàng đợi rỗng/đầy
  - Đẩy phần tử (push) vào đầu/cuối hàng đợi
  - Lầy phần tử (pop) ở đầu/cuối hàng đợi
  - Đọc nội dung phần tử ở đầu/cuối hàng đợi
  - Hủy hàng đợi

Hàng đợi ưu tiên

- Mỗi phần tử có một độ ưu tiên
- Phần tử có độ ưu tiên cao nhất sẽ được lấy ra khỏi hàng đợi trước tiên
- Các phần tử có cùng độ ưu tiên được lấy theo nguyên tắc FIFO
- Các thao tác
  - Khởi tạo hàng đợi
  - Kiểm tra hàng đợi rỗng/đầy
  - Đẩy phần tử (push) vào hàng đợi
  - Lầy phần tử (pop) có độ ưu tiên cao nhất
  - Đọc nội dung phần tử có độ ưu tiên cao nhất
  - Hủy hàng đợi

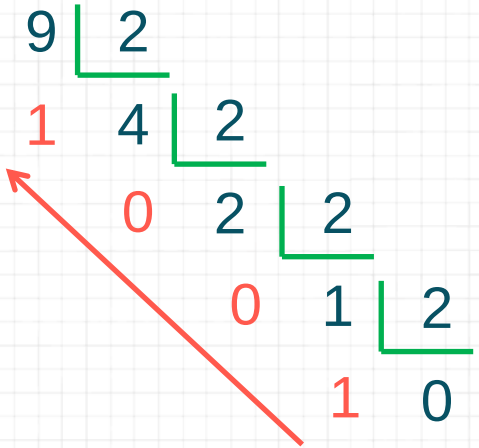


04

Ứng dụng

Khử đệ quy

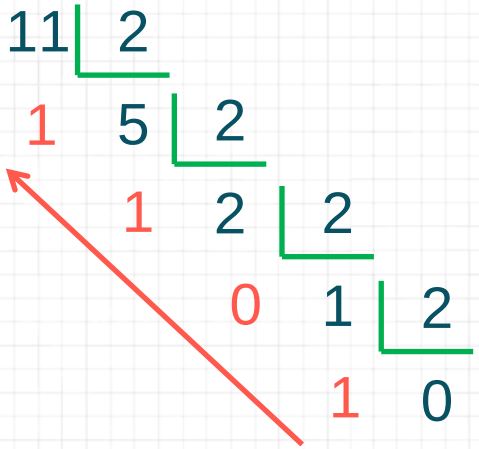
- Dùng ngăn xếp để khử đệ quy
- Ví dụ: chuyển số nguyên sang hệ nhị phân

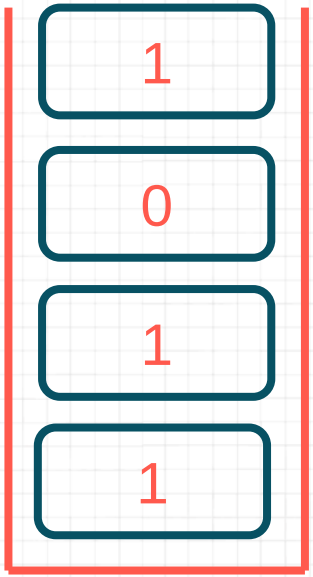



```
int dec2bin(int a)
{
    if (a == 0)
        return 0;
    return dec2bin(a/2)*10 + a%2;
}
```

Khử đệ quy

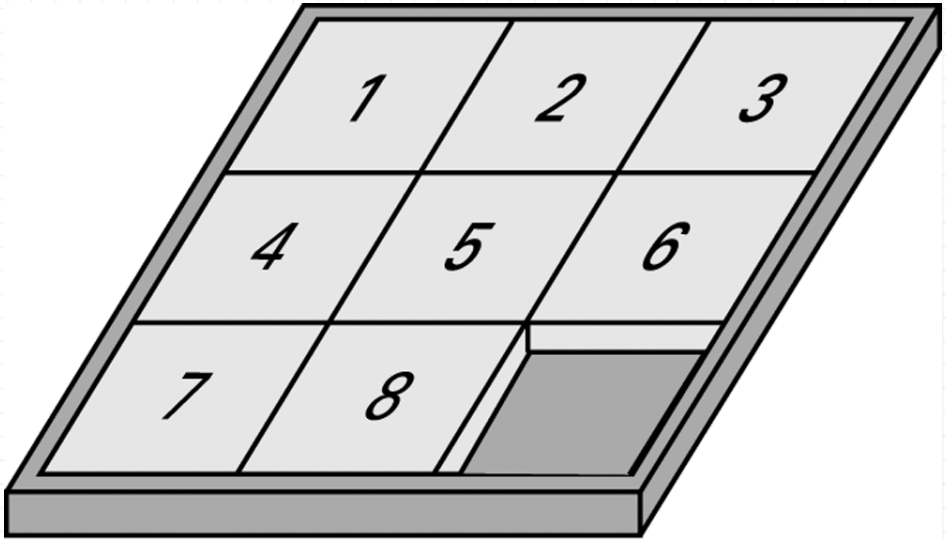
- Dùng ngăn xếp để khử đệ quy
- Ví dụ: chuyển số nguyên sang hệ nhị phân





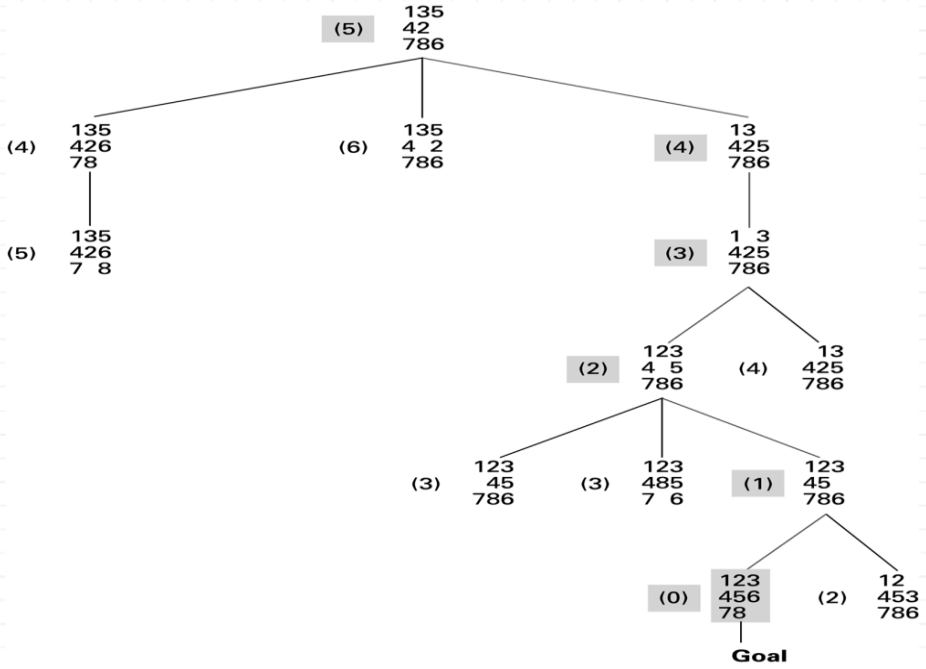
Tìm kiếm trong không gian trạng thái

- Ví dụ: Bài toán 8-puzzle



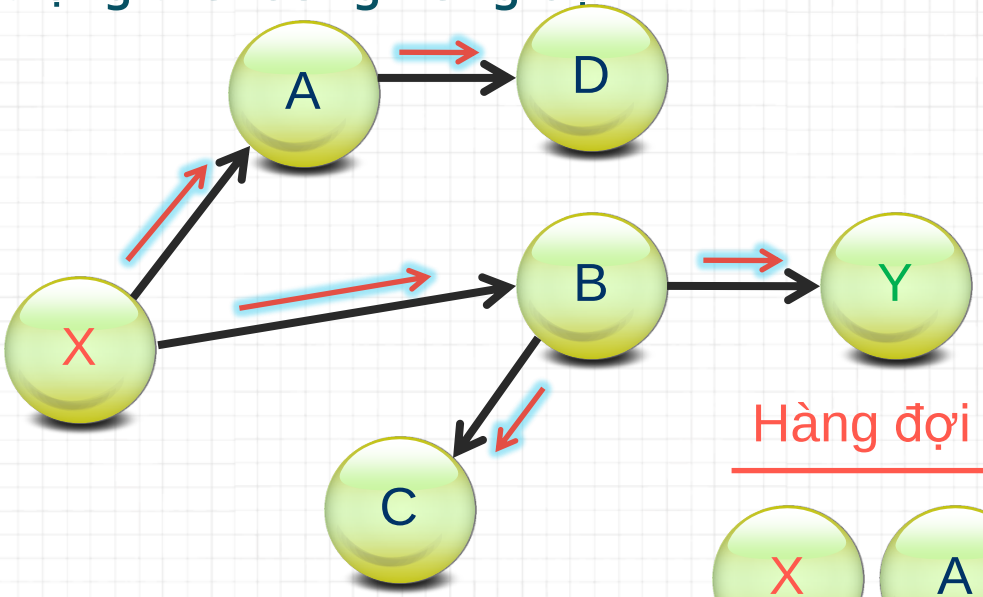
Tìm kiếm trong không gian trạng thái

- Khi di chuyển ô số → đi từ trạng thái này sang trạng thái khác

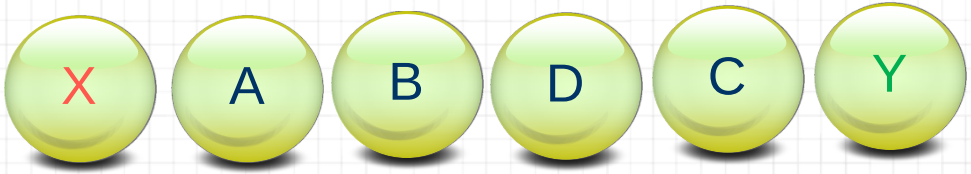


Tìm kiếm trong không gian trạng thái

- Tìm đường đi từ trạng thái X đến trạng thái Y bằng cách lưu các trạng thái bằng hàng đợi

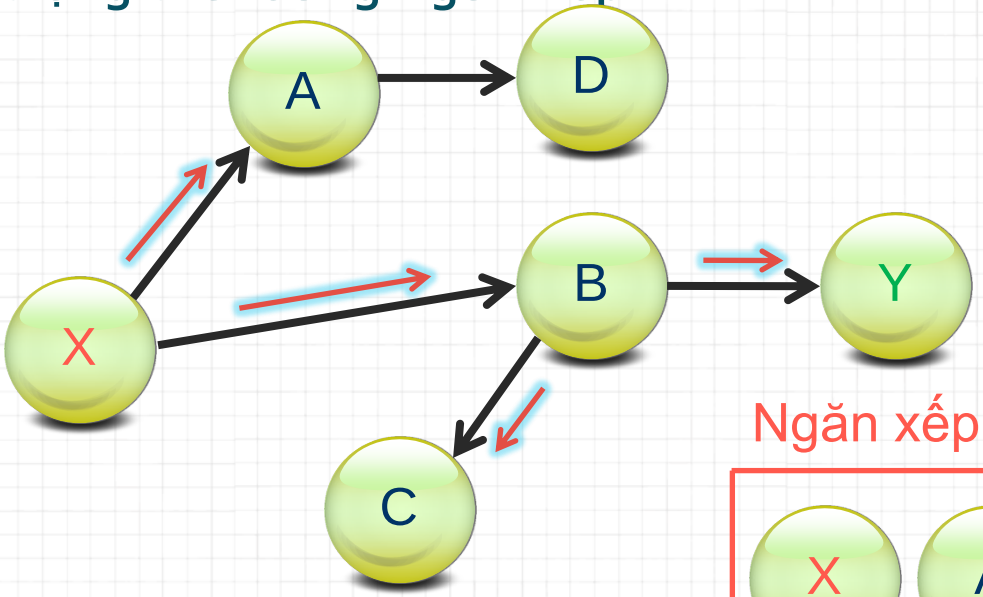


Hàng đợi

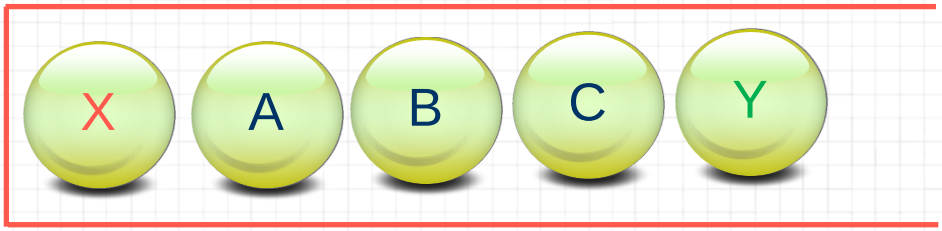


Tìm kiếm trong không gian trạng thái

- Tìm đường đi từ trạng thái X đến trạng thái Y bằng cách lưu các trạng thái bằng ngăn xếp



Ngăn xếp

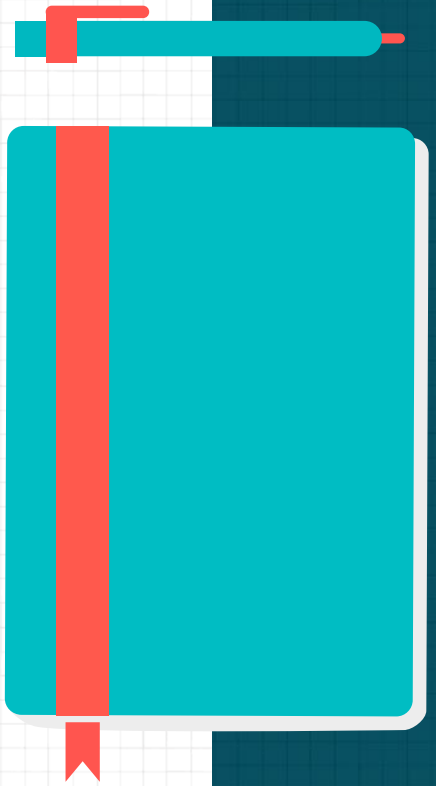


Bài tập 1

- Viết các hàm
  - Chia đôi DSLK thành hai DSLK
  - Chia đôi DSLK tại node q thành hai DSLK
  - Chia DSLK thành hai DSLK mới, một gồm toàn số chẵn, một gồm toàn số lẻ
  - Trộn hai DSLK thành DSLK mới có các phần tử nằm xen kẽ nhau
  - Trộn hai DSLK có thứ tự tăng thành một DSLK cũng có thứ tự tăng

Bài tập 2

- Cài đặt
  - Các thao tác trên ngăn xếp bằng mảng, DSLK
  - Các thao tác trên Hàng đợi truyền thống bằng mảng, DSLK
  - Các thao tác trên Hàng đợi hai đầu bằng mảng, DSLK
  - Các thao tác trên Hàng đợi ưu tiên bằng mảng, DSLK



The End!

Do you have any questions?