DANH SÁCH LIÊN KẾT

1. Giới thiệu về danh sách liên kết:

a) Định nghĩa:

Danh sách liên kết là một cấu trúc dữ liệu được sử dụng để lưu trữ các phần tử tương tự như mảng nhưng có nhiều điểm khác biệt.

1. Giới thiệu về danh sách liên kết:

b) Tính chất:

- DSLK có thể mở rộng và thu hẹp một cách linh hoạt.
- Phần tử cuối cùng trong DSLK trỏ vào NULL.
- Không lãng phí bộ nhớ nhưng cần thêm bộ nhớ để lưu phần con trỏ.
- Các phần tử trong DSLK được gọi là Node, được cấp phát động.
- Đây là CTDL cấp phát động nên khi còn bộ nhớ thì sẽ còn thêm được phần tử vào DSLK.

(Hình ảnh danh sách liên kết có 5 node)

2. So sánh giữa mảng và DSLK:

a) Mảng:

Bộ nhớ được cấp phát cho mảng là một khối bộ nhớ liên tiếp nhau, các phần tử trong mảng có thể truy cập thông qua chỉ số với độ phức tạp O(1).

Ưu điểm

- Đơn giản và dễ sử dụng.
- Truy cập mảng với độ phức tạp là hằng số.

Nhược điểm

- Có thể gây lãng phí bộ nhớ nếu không sử dụng hết bộ nhớ xin cấp phát cho mảng.
- Kích thước mảng là cố định.
- Bộ nhớ cấp phát theo khối.
- Việc chèn và xóa phần tử khó khăn.

2. So sánh giữa mảng và DSLK:

b) DSLK:

Bộ nhớ cấp phát cho các node trong DSLK có thể nằm rải rác nhau trong bộ nhớ.

Ưu điểm

- Có thể mở rộng với độ phức tạp là hằng số.
- Dễ dàng mở rộng và thu hẹp kích thước.
- Có thể cấp phát số lượng lớn các node tùy vào bộ nhớ.

Nhược điểm

- Khó khăn trong việc truy cập một phần tử ở vị trí bất kỳ (O(n)).
- Khó khăn trong việc cài đặt.
- Tốn thêm bộ nhớ cho phần tham chiếu bổ sung.

3. Độ phức tạp của các thao tác với mảng và DSLK:

Thao tác	DSLK	Mảng
Truy xuất phần tử	O(n)	O(1)
Chèn/Xóa ở đầu	O(1)	O(n) nếu mảng chưa full
Chèn ở cuối	O(n)	O(1) nếu mảng chưa full
Xóa ở cuối	O(n)	O(1)
Chèn giữa	O(n)	O(n) nếu mảng chưa full
Xóa giữa	O(n)	O(n)

4. DSLK đơn (Single Linked List):

DSLK bao gồm một số các node trong đó mỗi node có con trỏ next tới node tiếp theo nó trong DSLK. Liên kết của node cuối cùng trong DSLK là cont trỏ NULL.

Head

4. DSLK đơn (Single Linked List):

Cấu trúc một node của DSLK

```
struct node{
    int data;
    struct node *next; // link
};
```

Giải thích ý nghĩa của cấu trúc node:

Node ở đây có phần dữ liệu là một số nguyên lưu ở data, ngoài ra nó còn có 1 phần con trỏ tới chính struct node. Phần này chính là địa chỉ của node tiếp theo của nó trong DSLK.

Như vậy mỗi node sẽ có dữ liệu của nó và có địa chỉ của node tiếp sau nó. Đối với con trỏ cuối cùng trong DSLK thì phần địa chỉ này sẽ là con trỏ NULL.

Các bạn cần nhớ rằng, mỗi node trong DSLK đều được cấp phát động.

5. Các thao tác trên DSLK đơn:

a) Tạo một node mới:

```
struct node{
    int data;
    struct node* next;
};

node* makeNode(int x){
    node *newNode = (node*)malloc(sizeof(node));
    newNode->data = x;
    newNode->next = NULL;
    return newNode;
}
```

5. Các thao tác trên DSLK đơn:

b) Duyệt DSLK và đếm số node:

Duyệt DSLK

```
void duyet(node *head){
    while(head != NULL){
        cout << head->data << ' ';
        head = head->next;
    }
}
```

Đếm số node có trong DSLK

```
int size(node *head){
    int cnt = 0;
    while(head != NULL){
        ++cnt;
        head = head->next;
    }
    return cnt;
}
```

5. Các thao tác trên DSLK đơn:

c) Thêm node vào DSLK:

Thêm vào đầu DSLK

Để thêm 1 node mới vào đầu DSLK ta thực hiện 2 bước:

Bước 1: Cập nhật phần next của node mới lưu nội dung mà con trỏ node head của DSLK đang lưu.

Bước 2: Cập nhật nội dung node head.

```
void pushFront(node **head, int x){
    node* newNode = makeNode(x);
    newNode->next = (*head);
    (*head) = newNode;
}
```

5. Các thao tác trên DSLK đơn:

c) Thêm node vào DSLK:

Thêm vào cuối DSLK

Để thêm vào 1 node vào cuối DSLK, nếu DSLK đang rỗng thì làm tương tự như thêm đầu, ngược lại ta thực hiện theo 2 bước:

Bước 1: Duyệt từ đầu danh sách tới node cuối cùng thì dừng lại.

Bước 2: Cho phần next của node cuối trỏ vào node mới được thêm vào.

```
void pushBack(node **head, int x){
    node* newNode = makeNode(x);
    if(*head == NULL){
        *head = newNode; return;
    }
    node* tmp = *head;
    while(tmp->next != NULL){
        tmp = tmp->next;
    }
    tmp->next = newNode;
}
```

5. Các thao tác trên DSLK đơn:

c) Thêm node vào DSLK:

Thêm vào vị trí thứ K DSLK

Nếu K < 1 hoặc K lớn hơn cỡ của DSLK thì vị trí chèn không hợp lệ. Nếu K = 1 thì ta thêm vào đầu, ngược lại ta thực hiện theo 3 bước

Bước 1: Duyệt tới node trước vị trí cần chèn gọi là node K - 1.

Bước 2: Cho phần next của node mới lưu node next của node K - 1.

Bước 3: Cho phần next của node K - 1 lưu node mới.

```
void insert(node **head, int k, int x){
    int n = size(*head);
    if(k < 1 || k > n){
        cout << "Vi tri chen khong hop le !\n";
        return;
    }
    if(k == 1){
        pushFront(head, x); return;
    }
    node *temp = *head;
    for(int i = 1; i < k - 1; i++){
        temp = temp->next;
    }
    node *newNode = makeNode(x);
    newNode->next = temp->next;
    temp->next = newNode;
}
```

5. Các thao tác trên DSLK đơn:

d) Xóa node khỏi DSLK:

Xóa node khỏi đầu DSLK

Để xóa node khỏi đầu DSLK ta thực hiện 2 bước:

Bước 1: Cho node head lưu node thứ 2 hiện tại trong DSLK.

Bước 2: Giải phóng node đầu tiên trong DSLK.

```
void popFront(node **head){
    if(*head == NULL) return;
    node* tmp = *head;
    *head = tmp->next;
    delete tmp;
}
```

5. Các thao tác trên DSLK đơn:

d) Xóa node khỏi DSLK:

Xóa node khỏi cuối DSLK

Nếu DSLK chỉ có 1 node thì bạn cần xử lý riêng: Cho con trỏ head trỏ vào NULL

Bước 1: Tìm tới node thứ 2 từ cuối về gọi là tmp.

Bước 2: Cho phần next của node tmp trỏ vào NULL.

Bước 3: Giải phóng node cuối cùng trong DSLK.

```
void popBack(node **head){
    if(*head == NULL) return;
    node *tmp = *head;
    if(tmp->next == NULL){
        *head = NULL; delete tmp;
        return;
    }
    while(tmp->next->next != NULL){
        tmp = tmp->next;
    }
    node *last = tmp->next;
    tmp->next = NULL;
    delete tmp;
}
```

5. Các thao tác trên DSLK đơn:

d) Xóa node khỏi DSLK:

Xóa node ở vị trí K trong DSLK

Chú ý : Nếu vị trí không hợp lệ thì ta không thực hiện, nếu vị trí K = 1 thì ta gọi hàm popFront để xóa đầu.

```
void erase(node **head, int k){
    int n = size(*head);
    if(k < 1 || k > n) return;
    if(k == 1) popFront(head);
    else{
        node *truoc = *head;
        node *sau = *head;
        for(int i = 1; i <= k - 1; i++){
            sau = truoc;
            truoc = truoc->next;
        }
        sau->next = truoc->next;
        delete truoc;
    }
}
```

5. Các thao tác trên DSLK đơn:

e) Các thao tác tìm kiếm:

Tìm node thứ K trong DSLK

```
void kthNode(node *head, int pos){
    int n = size(head);
    if(pos < 1 || pos > n){
        cout << "Vi tri khong hop le !\n";
        return;
    }
    node *temp = head;
    for(int i = 1; i <= pos - 1; i++){
        temp = temp->next;
    }
    cout << temp->data << ' ';
}
```

Tìm node có giá trị lớn nhất trong DSLK

```
void maxNode(node *head){
    int res = -1e9;
    while(head != NULL){
        res = fmax(res, head->data);
        head = head->next;
    }
    cout << res << ' ';
}
```

5. Các thao tác trên DSLK đơn:

f) Sắp xếp DSLK:

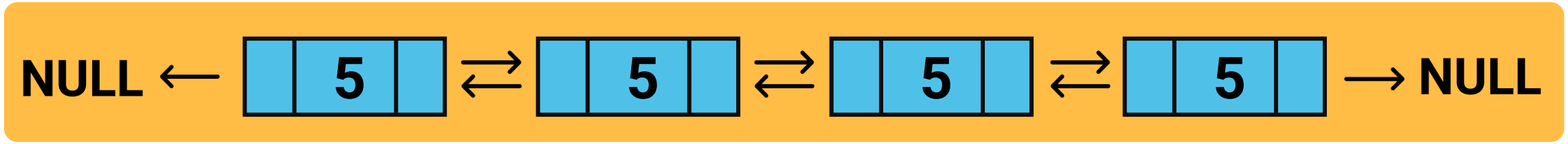
```
void sort(node **head){
    for(node *i = *head; i != NULL; i = i->next){
        node *min = i;
        for(node *j = i->next; j != NULL; j = j->next){
            if(j->data < min->data){
                min = j;
            }
        }
        int tmp = min->data;
        min->data = i->data;
        i->data = tmp;
    }
}
```

- **Thuật toán sử dụng:** Selection Sort

Chú ý là ta chỉ cần hoán vị phần data của 2 node còn không cần phải hoán vị next.

6. DSLK đôi (Double Linked List):

Ưu điểm của DSLK đôi đó là có thể di chuyển DSLK theo cả 2 chiều, tuy nhiên nó cùng cần thêm bộ nhớ để lưu con trỏ tới node liền trước cũng như như các thao tác trên DSLK đôi sẽ nhiều hơn so với DSLK đơn.



Cấu trúc một node của DSLK đôi

```
struct node{
    int data;
    struct node *next;
    struct node *prev;
}
```

7. Các thao tác trên DSLK đôi:

a) Tạo một node mới:

```
node* makeNode(int x){
    node *newNode = (node*)malloc(sizeof(node));
    newNode->data = x;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
```

7. Các thao tác trên DSLK đôi:

b) Duyệt DSLK và đếm số node:

Đếm số node có trong DSLK

```
int size(node *head){
    int cnt = 0;
    while(head != NULL){
        ++cnt;
        head = head->next;
    }
    return cnt;
}
```

7. Các thao tác trên DSLK đôi:

b) Duyệt DSLK và đếm số node:

Duyệt DSLK theo chiều thuận

```
void duyetThuan(node *head){
   while(head != NULL){
      cout << head->data << ' ';
      head = head->next;
   }
}
```

Duyệt DSLK theo chiều ngược

```
void duyetNguoc(node *head){
   while(head->next != NULL){
      head = head->next;
   }
   while(head != NULL){
      cout << head->data << ' ';
      head = head->prev;
   }
}
```

7. Các thao tác trên DSLK đôi:

c) Thêm một node mới vào DSLK:

Thêm vào đầu DSLK

```
void pushFront(node **head, int x){
    node* newNode = makeNode(x);
    newNode->next = (*head);
    if(*head != NULL)
        (*head)->prev = newNode;
        (*head) = newNode;
}
```

Thêm vào cuối DSLK

```
void pushBack(node **head, int x){
    node* newNode = makeNode(x);
    if(*head == NULL){
        *head = newNode; return;
    }
    node* tmp = *head;
    while(tmp->next != NULL){
        tmp = tmp->next;
    }
    tmp->next = newNode;
    newNode->prev = tmp;
}
```

7. Các thao tác trên DSLK đôi:

Thêm vào node thứ K trong DSLK

```
void insert(node **head, int k, int x){
   int n = size(*head);
   if(k < 1 || k > n){
      cout << "Vi tri chen khong hop le !\n"); return;
   }
   if(k == 1){
      pushFront(head, x); return;
   }
   node *temp = *head;
   for(int i = 1; i <= k - 1; i++){
      temp = temp->next;
   }
   node *newNode = makeNode(x);
   newNode->next = temp;
   temp->prev->next = newNode;
   newNode->prev = temp->prev;
   temp->prev = newNode;
}
```

7. Các thao tác trên DSLK đôi:

d) Xóa node khỏi DSLK:

Xóa node đầu khỏi DSLK

```
void popFront(node **head){
    if(*head == NULL) return;
    node* tmp = *head;
    *head = tmp->next;
    if(*head != NULL)
        (*head)->prev = NULL;
    delete tmp;
}
```

Xóa node cuối khỏi DSLK

```
void popBack(node **head){
    if(*head == NULL) return;
    node *tmp = *head;
    if(tmp->next == NULL){
        *head = NULL; free(tmp);
        return;
    }
    while(tmp->next != NULL){
        tmp = tmp->next;
    }
    tmp->prev->next = NULL;
    delete tmp;
}
```

7. Các thao tác trên DSLK đôi:

d) Xóa node khỏi DSLK:

Xóa node ở vị trí thứ K trong DSLK

```
void erase(node **head, int k){
   int n = size(*head);
   if(k < 1 || k > n) return;
   if(k == 1) popFront(head);
   else{
      node *temp = *head;
      for(int i = 1; i <= k - 1; i++){
         temp = temp->next;
      }
      temp->prev->next = temp->next;
      temp->next->prev = temp->prev;
      delete temp;
   }
}
```