**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**MÔN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**NHÓM 17 LỚP 8**

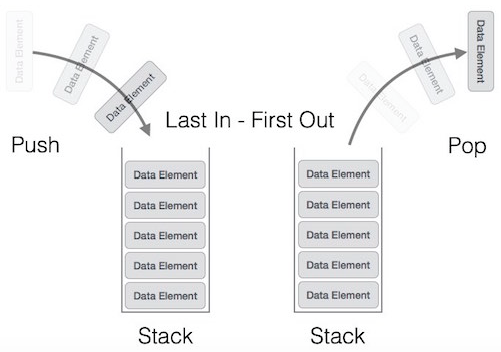
Chi tiết nhiệm vụ

|  |  |
| --- | --- |
| **Họ tên** | **MSSV** |
| Nguyễn Chí Thanh | 21110644 |
| Nguyễn Thành Nhơn | 21110907 |
| Nguyễn Phạm Mạnh Hóa | 21110885 |

**STACK**

**Giới thiệu về STACK**

Đây là cấu trúc dữ liệu hoạt động theo nguyên tắc: vào sau ra trước (Last in first out - LIFO).



**Các hàm sử dụng trong bài:**

**Khai báo cấu trúc của node và stack:**

Text

Description automatically generated

**Hàm khởi tạo Stack:**

Graphical user interface, text

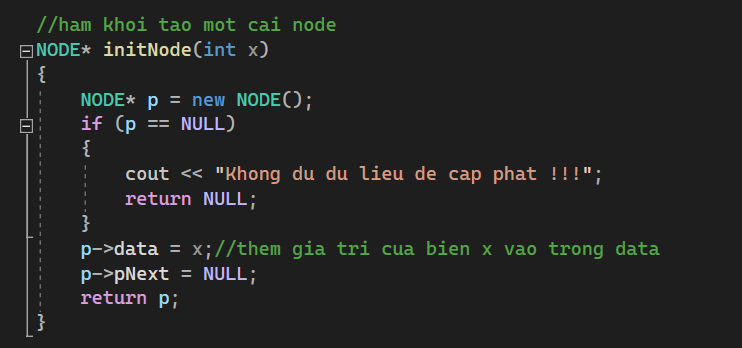
Description automatically generated

**Hàm khởi tạo 1 node:**

Kiểm tra p có bằng NULL hay không?

Nếu p == NULL thì trả về NULL

Nếu p != NULL thì gắn data của p bằng x và phần next p bằng NULL



**Hàm kiểm tra xem stack có rỗng hay không:**

Kiểm tra Top của Stack. Nếu bằng NULL là stack rỗng sẽ trả true. Khác NULL là stack có phần tử sẽ trả về false

Graphical user interface, text

Description automatically generated

**Hàm thêm phần tử vào stack:**

Tạo 1 node p và kiểm tra xem nó có rỗng hay không ? Rỗng thì trả về fasle

Tiếp theo kiểm tra stack có phần tử nào hay không ? Nếu không có thì gắn node p vào top. Nếu có thì gắn vào phần NEXT của node p là phần tử đầu stack và cập nhật lại phần tử đầu stack là node p

Text

Description automatically generated

**Hàm lấy phần tử đầu stack và hủy nó đi:**

Kiểm tra danh sách rỗng không ? Nếu rỗng thì in ra “Danh sach rong”

Nếu không rỗng thì khai báo node p bằng top stack, và gắn data của p vào x; sau đó cập nhật lại top stack bằng phần tử tiếp theo và xóa node p.

Text

Description automatically generated

**Hàm lấy phần tử đầu stack mà không hủy:**

Kiểm tra danh sách rỗng không ? Nếu rỗng thì in ra “Danh sach rong”

Nếu không rỗng thì gắn x bằng phần data của phần tử top stack.

A screenshot of a computer

Description automatically generated with medium confidence

**Hàm xem tất cả giá trị trong stack mà không xóa:**

Kiểm tra danh sách rỗng không ? Nếu rỗng thì in ra “Danh sach rong”

Nếu không thì khai báo 1 node temp bằng phần tử top của stack, sau đó duyệt bằng cách cho node temp=temp-> next và khi node temp = null thì duyệt hết stack và ngừng duyệt

Text

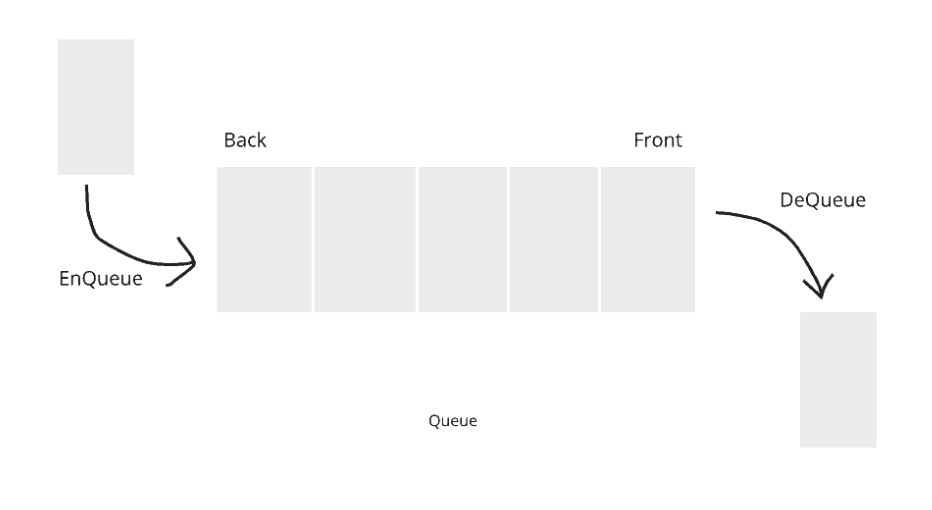
Description automatically generated

Queue

Queue là một cấu trúc dữ liệu dạng “vào trước, ra trước” (First In First Out)

Một Queue sẽ hỗ trợ các thao tác sau:

* Thêm phần tử vào cuối \Loại bỏ phần tử ở đầu queue
* Lấy ra phần tử đầu tiên trong queue
* Lấy ra kích thước của queue



Khai báo cấu trúc của một node trong queue:

struct node

{

int data;//queue dang chua cac so nguyen

struct node\* pNext; //con tro lien ket giua cac node voi nhau

};

typedef struct node NODE;

Khai báo cấu trúc của queue:

struct queue

{

NODE\* pHead;// dung con tro dau va cuoi de quan ly queue

NODE\* pTail;

};

typedef struct queue QUEUE;

Hàm khởi tạo queue:

void khoiTaoQueue(QUEUE& q)

{

q.pHead = NULL;

q.pTail = NULL;

}

Hàm khởi tạo node:

NODE\* khoiTaoNode(int x)

{

NODE\* p = new NODE();

if (p == NULL)

{

cout << "\nKhong du du lieu de cap phat !!!";

return NULL;

}

p->data = x;//them gia tri cua bien x vao trong data

p->pNext = NULL;

return p;

}

Hàm kiểm tra node có rỗng không:

bool IsEmpty(QUEUE q)

{

//neu danh sach rong

if (q.pHead == NULL)

return true;

return false;//danh sach co phan tu

}

Hàm thêm một phần tử vào cuối queue:

bool Enqueue(QUEUE& q, NODE\* p)

{

//neu stack rong

if (IsEmpty(q) == true)

{

q.pHead = q.pTail = p;//node p gio la node dau va node cuoi cua queue

}

else //danh sach da co phan tu

{

q.pTail->pNext = p;//lien ket con tro cuoi cua danh sach voi phan tu p can them vao

q.pTail = p;//cap nhap lai con tro cuoi la node p

}

return true;

}

Hàm lấy phần tử đầu queue và xóa khỏi queue:

bool Dequeue(QUEUE& q, int& x)//x chinh la gia tri can lay trong node

{

//neu danh sach rong

if (IsEmpty(q) == true)

{

return false;

}

else

{

NODE\* p = q.pHead; //node p la node de chung ta xoa

x = p->data; //gan gia tri dau danh sach vao bien x

cout << x << endl;

q.pHead = q.pHead->pNext; //cap nhap lai node dau cua queue

delete p; //xoa node dau queue vua lay

}

return true;//lay phan tu dau thanh cong

}

Hàm xem giá trị phần tử đầu của queue mà không thay đổi giá trị của nó:

bool Top(QUEUE& q, int& x)//x la bien de lay gia tri can xem

{

if (IsEmpty(q) == true)

{

return false;

}

x = q.pHead->data;//lay gia tri phan tu dau queue ra de xem

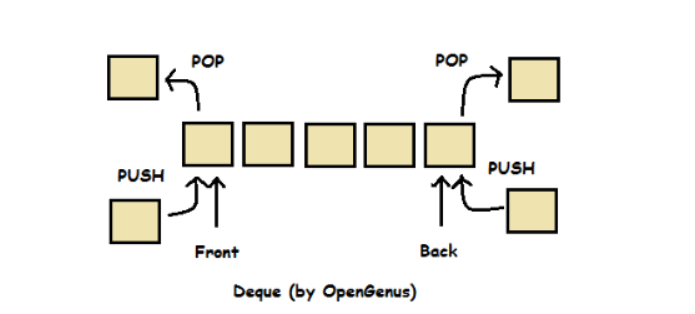
return true;

}

Deque

Deque (double-ended queue) có nghĩa là hàng đợi hai đầu. Một deque sẽ hỗ trợ các phương thức sau:

* Thêm một phần tử vào cuối deque
* Thêm một phần tử vào đầu deque
* Bỏ đi phần tử ở cuối deque
* Bỏ đi phần tử ở đầu deque
* Lấy ra giá trị phần tử đầu deque
* Lấy ra giá trị phần tử cuối deque



Khai báo cấu trúc của một node trong deque:

struct node

{

int data;//deque dang chua cac so nguyen

struct node\* pNext; //con tro lien ket giua cac node voi nhau

};

typedef struct node NODE;

Khai báo cấu trúc của deque:

struct deque

{

NODE\* pHead;// dung con tro dau va cuoi de quan ly deque

NODE\* pTail;

};

typedef struct deque DEQUE;

Hàm khởi tạo deque:

void khoiTaoDeque(DEQUE& q)

{

q.pHead = NULL;

q.pTail = NULL;

}

Hàm khởi tạo một cái node:

NODE\* khoiTaoNode(int x)

{

NODE\* p = new NODE();

if (p == NULL)

{

cout << "\nKhong du du lieu de cap phat !!!";

return NULL;

}

p->data = x;//them gia tri cua bien x vao trong data

p->pNext = NULL;

return p;

}

Hàm kiểm tra deque có rỗng không:

bool IsEmpty(DEQUE q)

{

//neu danh sach rong

if (q.pHead == NULL)

return true;

return false;//danh sach co phan tu

}

Hàm thêm phần tử vào đầu deque:

void Pushfront(DEQUE& q, NODE\* p)

{

if (q.pHead == NULL)// danh sách rỗng

{

q.pHead = q.pTail = p;

}

else

{

p->pNext = q.pHead;//cho node p trỏ đến node đầu tiên

q.pHead = p;//cập nhập lại node đầu bằng p

}

}

Hàm thêm phần tử vào cuối deque:

void Pushback(DEQUE& q, NODE\* p)

{

if (q.pHead == NULL)

{

q.pHead = q.pTail = p;

}

else

{

q.pTail->pNext = p;

q.pTail = p;

}

}

Hàm xóa phần tử đầu deque:

void Popfront(DEQUE& q)

{

//nếu danh sách rỗng

if (q.pHead == NULL)

{

return;

}

NODE\* p = q.pHead;//node p là node sẽ xóa

q.pHead = q.pHead->pNext;//cập nhập lại pHead là phần tử kế tiếp

delete p;

}

Hàm xóa phần tử cuối deque:

void Popback(DEQUE& q)

{

//nếu deque rỗng

if (q.pHead == NULL)

return;

//trường hợp deque có một phần tử

if (q.pHead->pNext == NULL)

{

Popfront(q);

return;

}

//duyệt từ đầu deque đến phần tử kế cuối

for (NODE\* k = q.pHead; k != NULL; k = k->pNext)

{

//phát hiện phần tử kế cuối

if (k->pNext == q.pTail)

{

delete q.pTail;// xóa đi phần tử cuối

k->pNext = NULL;// cho con trỏ của node kế cuối trỏ đến vùng nhớ NULL

q.pTail = k;// cập nhập lại l.pTail.

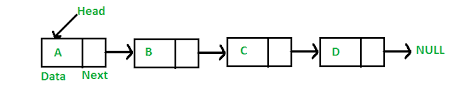
break;

}

}

}

Danh sách liên kết (Linked List): Là một tập hợp tuyến tính các phần tử dữ liệu, với thứ tự không được đưa ra bởi vị trí vật lý của chúng trong bộ nhớ. Thay vào đó, mỗi phần tử chỉ đến phần tử tiếp theo. Nó là một cấu trúc dữ liệu bao gồm một tập hợp các nút. Ở dạng cơ bản nhất, mỗi nút chứa: dữ liệu, và một tham chiếu (hay nói cách khác là liên kết) tới nút kế tiếp trong dãy. Cấu trúc này cho phép chèn hay loại bỏ phần tử khỏi bất kì vị trí nào trong chuỗi một cách hiệu quả trong quá trình lặp. Các biến thể phức tạp hơn như thêm các liên kết bổ sung, cho phép chèn hay loại bỏ các nút hiệu quả hơn tại vị trí bất kì. Một nhược điểm của danh sách liên kết là thời gian truy cập là tuyến tính. Ưu điểm so với mảng đó là có thể mở rộng và thu nhỏ nhờ cơ chế cấp phát động. Để quản lý một danh sách liên kết chỉ cần lưu thông tin của node đầu tiên trong danh sách.



Cấu trúc một node:

struct node

{

int data;

struct node\* pNext;

};

typedef struct node NODE;

Cấu trúc của list:

struct list

{

NODE\* pHead;//con trỏ đầu

NODE\* pTail;//con trỏ cuối

};

typedef struct list LIST;

Hàm khởi tạo danh sách liên kết:

void khoiTaoDSLK(LIST& l)

{

l.pHead = NULL;

l.pTail = NULL;

}

Hàm khởi tạo node:

NODE\* khoiTaoNODE(int x)

{

NODE\* p = new NODE;

p->data = x;

p->pNext = NULL;

return p;

}

Hàm thêm phần tử vào đầu danh sách:

void themVaoDau(LIST& l, NODE\* p)

{

if (l.pHead == NULL)// danh sách rỗng

{

l.pHead = l.pTail = p;

}

else

{

p->pNext = l.pHead;//cho node p trỏ đến node đầu tiên

l.pHead = p;//cập nhập lại node đầu bằng p

}

}

Hàm thêm phần tử vào cuối danh sách:

void themVaoCuoi(LIST& l, NODE\* p)

{

if (l.pHead == NULL)// danh sách rỗng

{

l.pHead = l.pTail = p;

}

else

{

l.pTail->pNext = p;//cho node cuối trỏ đến node p

l.pTail = p;// cập nhập lại node cuối bằng p

}

}

Hàm thêm node p vào vị trí bất ki:

void themNodePBatKi(LIST& l, NODE\* p, int vt, int n)

{

//danh sach rong

if (l.pHead == NULL || vt == 1)

{

//them p vao dau danh sach

themVaoDau(l, p);

}

else if (vt == n + 1)

{

//them node p vao cuoi danh sach

themVaoCuoi(l, p);

}

else //vt nam trong khoang [2,n]

{

int dem = 0; // xac dinh vi tri can them

NODE\* g = new NODE;//node giu lien ket voi node nam truoc vt trong danh sach

for (NODE\* k = l.pHead; k != NULL; k = k->pNext)

{

dem++;

//tim vi tri vt can them

if (dem == vt)

{

// thuc hien them node p vao sau node g <=> them node p vao truoc node q

NODE\* h = khoiTaoNODE(p->data);

h->pNext = k;//tao moi lien ket giua p va q

g->pNext = h;//tao moi lien ket giua node truoc q voi p

break;

}

g = k;//giu lien ket lan luot voi node nam truoc node q trong danh sach

}

}

}

Hàm xóa phần tử đầu danh sách:

void xoaDau(LIST& l)

{

//nếu danh sách rỗng

if (l.pHead == NULL)

{

return;

}

NODE\* p = l.pHead;//node p là node sẽ xóa

l.pHead = l.pHead->pNext;//cập nhập lại pHead là phần tử kế tiếp

delete p;

}

Hàm xóa phần tử cuối danh sách:

void xoaCuoi(LIST& l)

{

//nếu danh sách rỗng

if (l.pHead == NULL)

return;

//trường hợp danh sách có một phần tử

if (l.pHead->pNext == NULL)

{

xoaDau(l);

return;

}

//duyệt từ đầu danh sách đến thằng kế cuối

for (NODE\* k = l.pHead; k != NULL; k = k->pNext)

{

//phát hiện phần tử kế cuối

if (k->pNext == l.pTail)

{

delete l.pTail;// xóa đi phần tử cuối

k->pNext = NULL;// cho con trỏ của node kế cuối trỏ đến vùng nhớ NULL

l.pTail = k;// cập nhập lại l.pTail.

break;

}

}

}

Hàm xóa một node có khóa k bất ki:

void xoaNodeCoKhoaBatKy(LIST& l, int x)

{

// nếu danh sách rỗng

if (l.pHead == NULL)

{

return;

}

//nếu node cần xóa nằm đầu danh sách

if (l.pHead->data == x)

{

xoaDau(l);

return;

}

// nếu node cần xóa nằm cuối danh sách

if (l.pTail->data == x)

{

xoaCuoi(l);

return;

}

// duyệt danh sách tìm node cần xóa

NODE\* g = new NODE;// node đứng trước phần tử cần xóa

for (NODE\* k = l.pHead; k != NULL; k = k->pNext)

{

if (k->data == x)//phát hiện phần tử cần xóa

{

g->pNext = k->pNext;//

delete k;// xóa node nằm sau node g;

return;

}

g = k;//cho node g trỏ đến node k node truoc node chung ta can xoa

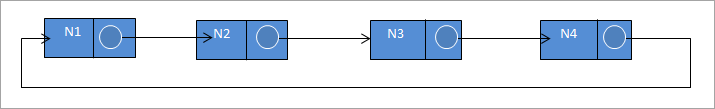
}

cout << "Khong co " << x << " trong danh sach" << endl;

system("pause");

}

Danh sách liên kết vòng (Circle List): Danh sách liên kết vòng là một biến thể của danh sách liên kết. Nó là một danh sách liên kết có các nút được kết nối theo cách nó tạo thành một vòng tròn. Trong danh sách liên kết vòng, con trỏ tiếp theo của nút cuối cùng không được đặt thành null mà nó chứa địa chỉ của nút đầu tiên, do đó tạo thành một vòng tròn.



Cấu trúc một node:

struct Node

{

int data;

struct Node\* next;

};

Thêm một node mới vào list trống:

struct Node\* insertInEmpty(struct Node\* last, int new\_data)

{

// kiểm tra mảng có trống không, không thì dừng

if (last != NULL)

return last;

// cấp phát bộ nhớ cho node

struct Node\* temp = new Node;

// gán dữ liệu

temp->data = new\_data;

last = temp;

// tạo liên kết

last->next = last;

return last;

}

Chèn node mới vào đầu danh sách:

struct Node\* insertAtBegin(struct Node\* last, int new\_data)

{

//nếu danh sách trống thì thêm nút bằng cách gọi insertInEmpty

if (last == NULL)

return insertInEmpty(last, new\_data);

// tạo một nút mới

struct Node\* temp = new Node;

//khởi tạo dữ liệu cho node

temp->data = new\_data;

temp->next = last->next;

last->next = temp;

return last;

}

Chèn node mới vào cuối danh sách:

struct Node\* insertAtEnd(struct Node\* last, int new\_data)

{

//nếu danh sách trống thì thêm node bằng cách gọi insertInEmpty

if (last == NULL)

return insertInEmpty(last, new\_data);

//tạo một node mới

struct Node\* temp = new Node;

//gán dữ liệu cho nút mới

temp->data = new\_data;

temp->next = last->next;

last->next = temp;

last = temp;

return last;

}

Chèn node mới vào giữa các node:

struct Node\* insertAfter(struct Node\* last, int new\_data, int after\_item)

{

//trả về null nếu danh sách trống

if (last == NULL)

return NULL;

struct Node\* temp, \* p;

p = last->next;//node đầu

do

{

if (p->data == after\_item)//kiểm tra dữ liệu của các node trong danh sách có bằng node cần tìm

{

temp = new Node;

temp->data = new\_data;

temp->next = p->next;

p->next = temp;

if (p == last)

last = temp;

return last;

}

p = p->next;

} while (p != last->next);

cout << "Nut co du lieu " << after\_item << " khong co trong danh sach." << endl;

system("pause");

return last;

}

Xóa node khỏi danh sách:

void deleteNode(Node\*\* head, int key)

{

// Nếu danh sách được liên kết trống, return

if (\*head == NULL)

return;

// Nếu danh sách chỉ chứa một nút, hãy xóa nút đó; Danh sách trống

if ((\*head)->data == key && (\*head)->next == \*head) {

free(\*head);

\*head = NULL;

}

Node\* last = \*head, \* d;

// Nếu key là đầu

if ((\*head)->data == key) {

while (last->next != \*head) // Tìm nút cuối cùng của danh sách

last = last->next;

// Trỏ nút cuối cùng tới nút tiếp theo của đầu hoặc nút thứ hai của danh sách

last->next = (\*head)->next;

free(\*head);

\*head = last->next;

}

// Đã đến cuối danh sách hoặc nút bị xóa không có trong danh sách

while (last->next != \*head && last->next->data != key) {

last = last->next;

}

//Đã tìm thấy nút cần xóa, vì vậy hãy giải phóng bộ nhớ và hiển thị danh sách

if (last->next->data == key) {

d = last->next;

last->next = d->next;

cout << "Nut co du lieu " << key << " bi xoa khoi danh sach " << endl;

free(d);

cout << endl;

cout << "Danh sach lien ket vong sau khi xoa " << key << " la nhu sau:" << endl;

xuat(last);

}

else

{

cout << "Nut co du lieu " << key << " khong tim thay trong danh sach" << endl;

}

}

Topological sort

Định nghĩa các cấu trúc

Text

Description automatically generated

Nhập và cài đặt input(Step1+ Step2 +Step3):

Text

Description automatically generated

Text

Description automatically generated

Step4:

Text, letter

Description automatically generated

Text

Description automatically generated

Step 5 +Step 6 +Step 7:

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

Step5:Kiểm tra nếu F=0:

Tới bước Step8 : Kết thúc chương trình;

Nếu F!=0:

xuất F ra. Giảm N 1 đơn vị. Gán P =TOP[F]. Qua Step6;

Step6: Nếu P = nullptr:

Tới bước T7;

Nếu P ≠ nullptr :

COUNT[SUC(P)] giảm 1;

Kiểm tra nếu COUNT[SUC(P)] =0:

Gắn QLINK[R]=SUC[P] và R=SUC(P).;

Gắn P=P->next và thực hiện lại Step6;

Step7: Gắn F=QLINK[F]. Quay lại bước 5;