**Đề cương Cấu trúc dữ liệu và thuật toán**

**Mục lục**

[**Từ viết tắt:** 1](#_Toc26879736)

[**Khái niệm MHDL, phân loại? VD?** 2](#_Toc26879737)

[**Khái niệm KDLTT? VD?** 2](#_Toc26879738)

[**Mối quan hệ giữa CTDL&TT** 2](#_Toc26879739)

[**Vai trò của CTDL? Các tiêu chí lựa chọn CTDL?** 2](#_Toc26879740)

[**Khái niệm đệ quy, giải thuật đệ quy? VD?** 2](#_Toc26879741)

[**Các đặc điểm của giải thuật đệ quy? Viết giải thuật đệ quy tính n!** 2](#_Toc26879742)

[**Công thức dãy Fibonacci? Cho Fn-1 và Fn-2 , tính Fn** 3](#_Toc26879743)

[**Khái niệm DS? VD?** 3](#_Toc26879744)

[**So sánh DS kế tiếp và DS liên kết?** 3](#_Toc26879745)

[**Khái niệm hàng đợi, ngăn xếp? VD?** 4](#_Toc26879746)

[**So sánh ngăn xếp và hàng đợi?** 4](#_Toc26879747)

[**Định nghĩa cây đa phân? Vẽ cây VD? Duyệt cây?** 4](#_Toc26879748)

[**Định nghĩa cây NP? Các dạng cây NP đặc biệt? VD? Vẽ cây NP? Duyệt cây?** 5](#_Toc26879749)

[**Định nghĩa cây NPTK? Vẽ cây VD? Duyệt cây?** 5](#_Toc26879750)

[**Định nghĩa đồ thị? Vẽ VD? Duyệt đồ thị theo chiều rộng, chiều sâu.** 5](#_Toc26879751)

[**Khái niệm đồ thị có hướng, đồ thị vô hướng, đồ thị liên thông, đồ thị đơn, đồ thị đa? VD?** 6](#_Toc26879752)

[**Định nghĩa tập hợp? VD?** 7](#_Toc26879753)

[**Các phương pháp mô tả tập hợp? VD?** 7](#_Toc26879754)

[**Nêu tư tưởng của bảng băm? Hàm băm có tác dụng gì? Các tiêu chí lựa chọn hàm băm?** 7](#_Toc26879755)

[**Code Tổng quát** 8](#_Toc26879756)

[**Dạng cài đặt** 8](#_Toc26879757)

[**Thêm** 8](#_Toc26879758)

[**Xoá** 10](#_Toc26879759)

**Từ viết tắt:**

VD: Ví dụ

CTDL&TT: Cấu trúc dữ liệu và thuật toán

MHDL: Mô hình dữ liệu

KDLTT: Kiểu dữ liệu trừu tượng

DS: Danh sách

NX: Ngăn xếp

HĐ: Hàng đợi

**Khái niệm MHDL, phân loại? VD?**

MHDL là mô hình toán học, biểu diễn các cấu trúc logic của dữ liệu, cùng với các phép toán có thể thực hiện trên các đối tượng của mô hình.

Phân loại: MHDL tuyến tính, phân cấp, đồ thị, tập hợp.

VD: Mô hình danh sách, cây, đồ thị, tập hợp,…

**Khái niệm KDLTT? VD?**

KDLTT là một MHDL được xét cùng với một số xác định các phép toán.

VD: MHDL danh sách, chỉ xét đến các phép toán thêm vào và lấy ra → KDLTT hàng đợi hoặc ngăn xếp. MHDL tập hợp, chỉ xét đến các phép toán thêm vào, loại bỏ và tìm kiếm → KDLTT từ điển.

**Mối quan hệ giữa CTDL&TT**

+ Trong một chương trình máy tính, CTDL&TT có mối quan hệ chặt chẽ với nhau: CTDL+TT=Chương trình;

+ Với một CTDL đã chọn, sẽ có những TT tương ứng hoặc phù hợp;

+ Khi CTDL thay đổi thì TT cũng thường phải thay đổi theo để tránh xử lý gượng ép, thiếu tự nhiên;

+ Một CTDL tốt sẽ giúp TT xử lý trên đó có thể phát huy tác dụng tốt hơn (đáp ứng nhanh, tiết kiệm bộ nhớ, TT dễ hiểu và đơn giản hơn).

**Vai trò của CTDL? Các tiêu chí lựa chọn CTDL?**

CTDL tương ứng với Các dữ liệu thành phần, trong đó, chúng luôn có quan hệ nào đó với nhau, mỗi dữ liệu thành phần thuộc một kiểu dữ liệu xác định.

CTDL cần phải thoả mãn các tiêu chí sau: Phản ánh đúng thực tế, Phù hợp với các thao tác trên đó và Tiết kiệm tài nguyên hệ thống.

**Khái niệm đệ quy, giải thuật đệ quy? VD?**

Đệ quy là khi đối tượng bao gồm chính nó như một bộ phận hoặc được định nghĩa dưới dạng của chính nó.

Nếu lời giải của bài toán T được thực hiện bởi lời giải của bài toán T’ (có dạng như T) thì đó là một lời giải đệ quy. Giải thuật chứa lời giải đệ quy thì gọi là giải thuật đệ quy (T’<T)

VD: Tính giai thừa: Giai thừa của n được định nghĩa qua giai thừa của n-1, n-1 được định nghĩa qua giai thừa của n-2, cứ như vậy tạo nên công thức đệ quy

**Các đặc điểm của giải thuật đệ quy? Viết giải thuật đệ quy tính n!**

+ Giải thuật đệ quy bao giờ cũng có lời gọi đến chính tên giải thuật;

+ Mỗi lần có lời gọi thì kích thước của bài toán sẽ thu nhỏ hơn trước;

+ Có một trường hợp đặc biệt là suy biến: Bài toán sẽ được giải quyết theo một cách khác hẳn và gọi đệ quy cũng kết thúc.

Tính P(n)=n! : Điều kiện dừng: P(0)=0!=1 (Lặp đến khi n=0 thì dừng).

Công thức tính: P(n)=P(n-1)\*n

Chương trình:

long Giaithua (int n) {

if(n==0) return 1;

else return (GiaiThua(n-1)\*n);

}

**Công thức dãy Fibonacci? Cho Fn-1 và Fn-2 , tính Fn**

Fibonacci có công thức tính như sau: Fn= Fn-1+ Fn-2 (Số thứ 3 bằng tổng hai số đằng trước nó)

Chương trình:

int Fibonacci (int n) {

if(n==1||n==2) return 1;

else return (Fibonacci(n-1)+Fibonacci(n-2));

}

**Khái niệm DS? VD?**

DS là một MHDL chứa một tập hợp hữu hạn, biến động các phần tử thuộc cùng một lớp đối tượng nào đó (có cùng một kiểu dữ liệu).

VD: Tập hợp 6 số nguyên: 10, 20, 30, 40, 50, 60. Tập hợp tên các trường đại học thuộc Đại học Thái Nguyên. (Danh sách này có x phần tử, mỗi phần tử có một vị trí trong danh sách theo thứ tự xuất hiện của nó).

**So sánh DS kế tiếp và DS liên kết?**

|  |  |
| --- | --- |
| **DS kế tiếp** | **DS liên kết** |
| + Thuộc loại cấu trúc dữ liệu tĩnh. | + Thuộc loại cấu trúc dữ liệu động. |
| + Vùng nhớ của các phần tử trong mảng được sắp xếp liên tục nhau. | + Vùng nhớ của các phần tử trong DS nằm rải rác trong bộ nhớ. Các phần tử lưu một con trỏ để trỏ tới phần tử tiếp theo. |
| + Truy cập tới phần tử là truy cập trực tiếp dựa vào chỉ số. | + Cần phải duyệt tuần tự khi muốn truy cấp tới các phần tử trong DS liên kết. |
| + Kích thước của mảng là hằng số, không thay đổi khi chạy chương trình. | + Kích thước của DS liên kết có thể thay đổi khi chạy chương trình. |
| + Sử dụng mảng không tối ưu được bộ nhớ. Có thể thừa hoặc thiếu bộ nhớ khi xoá hoặc chèn phần tử vào mảng. | + Sử dụng DS liên kết sẽ tối ưu được bộ nhớ. Vùng nhớ được cấp phát thêm khi cần chèm thêm phần tử mới, vùng nhớ được free khi xoá phần tử. |

**Khái niệm hàng đợi, ngăn xếp? VD?**

NX (Stack) là một DS đặc biệt, trong đó việc thêm vào hay lấy một phần tử ra khỏi ngăn xếp chỉ thực hiện tại một đầu của NX, đầu này gọi là đỉnh của NX.

VD: Chồng sách đặt trên bàn: Muốn thêm vào chồng sách đó ta để sách mới lên đỉnh, muốn lấy các sách ra khỏi chồng ta cũng lấy các sách từ trên đỉnh trước. Như vậy ngăn xếp cũng là một cấu trúc có tính chất vào sau ra trước hay vào trước ra sau.

HĐ (Queue) là một DS đặc biệt, trong đó việc thêm vào chỉ thực hiện tại một đầu của hàng (cuối hàng), còn việc lấy một phần tử ra thì thực hiện tại đầu còn lại (đầu hàng)

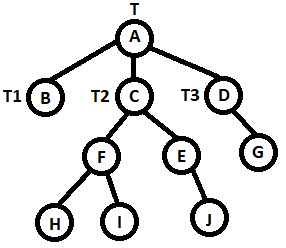
VD: Xếp hàng đi mua IPhone 12 PRO MAX: Người mới đến sẽ đi vào cuối hàng còn người ở đầu hàng mua vé và ra khỏi hàng, vì vậy hàng đợi còn được gọi là cấu trúc hoạt động theo nguyên tắc vào trước ra trước hay vào sau ra sau.

**So sánh ngăn xếp và hàng đợi?**

|  |  |
| --- | --- |
| **Ngăn xếp** | **Hàng đợi** |
| + LIFO: Last In First Out (vào sau ra trước). | + FIFO: First In First Out (vào trước ra trước). |
| + Thêm vào và lấy ra chỉ thực hiện tại một đầu (đỉnh - Top). | + Thêm vào và lấy ra được thực hiện ở hai đầu khác nhau: Thêm là vào cuối hàng còn lấy ra là ở đầu hàng. |
| + Chỉ sử dụng một con trỏ để chỉ vị trí của phần tử. | + Sử dụng hai con trỏ (một con trỏ chỉ về đầu hàng và một con trỏ chỉ về cuối hàng đối với trường hợp hàng đợi có cấu trúc đơn giản). |
| + Thực hiện thao tác Push (thêm vào đỉnh) và Pop (lấy ra khỏi đỉnh). | + Thực hiện thao tác EnQueue (Thêm vào cuối hàng) và DeQueue (Xoá phần tử ở đầu hàng). |
| + Không gây ra sự lãng phí không gian bộ nhớ | + Có gây ra sự lãng phí không gian bộ nhớ |

**Định nghĩa cây đa phân? Vẽ cây VD? Duyệt cây?**

Cây đa phân gồm một tập hữu hạn các đỉnh, trong đó có một đỉnh đặc biệt gọi là gốc. Giữa các đỉnh có một quan hệ phân cấp gọi là quan hệ cha-con. Đỉnh ở mức trên là đỉnh cha và đỉnh ở mức dưới là đỉnh con.

VD: Xét cây

|  |  |
| --- | --- |
| Kết quả duyệt: | Theo thứ tự trước: A B CFHIEJ DG |
| Theo thứ tự giữa: B A HFICJE GD |
| Theo thứ tự sau: B HIFJEC GD A |

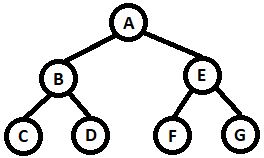
**Định nghĩa cây NP? Các dạng cây NP đặc biệt? VD? Vẽ cây NP? Duyệt cây?**

Cây nhị phân là cây rỗng hoặc là cây mà mỗi nút có tối đa là hai nút con. Trong đó các nút con của cây được phân biệt thứ tự rõ ràng, một nút con gọi là nút con trái và một nút con gọi là nút con phải.

Các dạng cây nhị phân đặc biệt: Cây nhị phân suy biến: Cây lệch trái, cây lệch phải, cây dích dắc; Cây nhị phân hoàn chỉnh; Cây nhị phân đầy đủ.

|  |  |
| --- | --- |
| Kết quả duyệt: | Theo thứ tự trước: A BCD EFG |
| Theo thứ tự giữa: CBD A FEG |
| Theo thứ tự sau: CDB FGEA |

VD: Xét cây nhị phân



**Định nghĩa cây NPTK? Vẽ cây VD? Duyệt cây?**

Cây NPTK là một cây nhị phân thoả mãn các điều kiện sau:

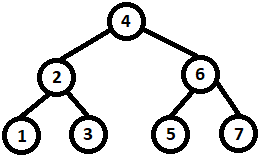
Điều kiện 1: Tất cả các khoá tại đỉnh của cây con bên trái đều có giá trị đi trước nhỏ hơn các khoá tại đỉnh gốc.

Điều kiện 2: Khoá tại gốc đi trước phải nhỏ hơn tất cả các khoá ở các đỉnh của cây con bên phải.

Điều kiện 3: Cây con bên trái và cây con bên phải cũng là cây tìm kiếm nhị phân.

Ví dụ: Xét cây nhị phân tìm kiếm

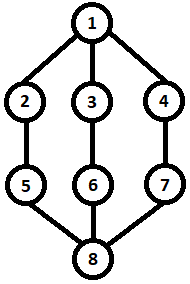
|  |  |
| --- | --- |
| Kết quả duyệt: | Theo thứ tự trước: 4 213 657 |
| Theo thứ tự giữa: 123 4 567 |
| Theo thứ tự sau: 132 4 576 |



**Định nghĩa đồ thị? Vẽ VD? Duyệt đồ thị theo chiều rộng, chiều sâu.**

Đồ thị là một cấu trúc rời rạc, dùng để mô tả một tập hợp các đối tượng rời rạc có mối quan hệ n-m với nhau (n,m >=0). Đồ thị được ký hiệu là G=<V,E>, trong đó V là tập các đỉnh; E là tập các cạnh, cung: Tập các cặp (u,v) mà u, v là hai đỉnh thuộc V.

Ví dụ: Xét đồ thị:

Duyệt cây (theo chiều rộng): 1 2 3 4 5 6 7 8

Duyệt cây (theo chiều sâu): 1 2 5 8 6 3 7 4

**Khái niệm đồ thị có hướng, đồ thị vô hướng, đồ thị liên thông, đồ thị đơn, đồ thị đa? VD?**

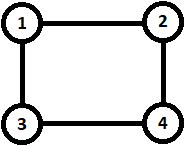
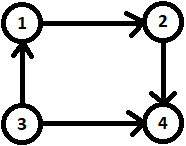
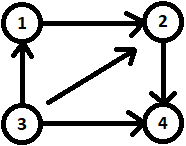
Đồ thị có hướng: Nếu E là tập các cặp (u,v)<>(v,u), gọi là các cung

Đồ thị vô hướng: Nếu E là tập các cặp (u,v)=(v,u), gọi là các cạnh

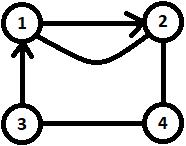
Đồ thị liên thông: Là đồ thị có đường đi ở giữa hai đỉnh bất kỳ của nó.

Đồ thị đơn: Nếu giữa hai đỉnh u,v bất kỳ có nhiều nhất một cạnh/cung

Đồ thị đa: Nếu giữa hai đỉnh u,v bất kỳ có thể có nhiều hơn một cạnh/cung

VD:

*Đồ thị vô hướng Đồ thị có hướng Đồ thị liên thông*





*Đồ thị đa Đồ thị đơn*

**Định nghĩa tập hợp? VD?**

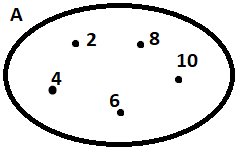
Tập hợp là một cấu trúc rời rạc được nói là chứa các phần tử của nó. Dùng để nhóm các phần tử có các tính chất chung lại với nhau, nhưng nó cũng có thể chứa các phần tử chẳng có mối quan hệ gì với nhau.

VD: Tập hợp A={16, B, ĐTTT, ICTU}

**Các phương pháp mô tả tập hợp? VD?**

Để mô tả một tập hợp, trong toán học thường có các cách sau để xác định một tập hợp A

1. Dùng biểu đồ Ven là một đường cong khép kín, các điểm trong đường cong đó chỉ các phần tử của tập hợp.

VD: Tập hợp A gồm có các phần tử được thể hiện qua biểu đồ Ven:

2. Liệt kê tất cả các phần tử của tập hợp A. Ví dụ: A = {1, 2, 3}

3. Nêu lên các đặc trưng cho biết chính xác một đối tượng bất kỳ có là một phần tử của tập A hay không. Ví dụ: A = {x | x là số nguyên chẵn}

**Nêu tư tưởng của bảng băm? Hàm băm có tác dụng gì? Các tiêu chí lựa chọn hàm băm?**

Tư tưởng của bảng Băm: Phân chia một tập hợp đã cho thành một số cố định các lớp N, được đánh số từ 0,1,…N-1. Sau đó sử dụng mảng T với chỉ số chạy từ 0 đến N-1. Mỗi thành phần T[i] chứa một con trỏ, trỏ tới phần tử đầu tiên của danh sách các phần tử thuộc lớp i. Các phần tử của tập hợp thuộc mỗi lớp được tổ chức dưới dạng một DS liên kết, mỗi danh sách được gọi là 1 “rổ”. T được gọi là bảng Băm.

Hàm Băm có tác dụng chia các phần tử của tập hợp vào các lớp của bảng Băm.

Các tiêu chí lựa chọn hàm Băm:

+ Hàm băm phải cho phép tính được dễ dàng và nhanh chóng giá trị băm của mỗi khoá.

+ Các khoá được phân bố đều vào các rỗ trong bảng.

+ Ít xảy ra đụng độ.

+ Xử lí được các loại khoá có kiểu dữ liệu khác nhau.

**Code Danh sách, Ngăn xếp, Hàng đợi**

**Dạng cài đặt**

|  |  |
| --- | --- |
| **Danh sách kế tiếp (mảng)** | **Danh sách liên kết (con trỏ)** |
| ***+) Dạng cài đặt:***  typedef int item;  typedef struct {  item Elems[100];  int count;  } List;  List L; | ***+) Dạng cài đặt:***  typedef int item;  typedef struct Node {  item Data;  Node \*next;  } Node;  typedef Node \*List |
| **Ngăn xếp kế tiếp (mảng)** | **Ngăn xếp liên kết (con trỏ)** |
| ***+) Dạng cài đặt:***  typedef int item;  typedef struct {  item Elems[100];  int top;  } Stack;  Stack S; | ***+) Dạng cài đặt:***  typedef int item;  typedef struct Node {  item Data;  Node \*next;  } Node;  typedef Node \*Stack |
| **Hàng đợi kế tiếp (mảng)** | **Hàng đợi liên kết (con trỏ)** |
| ***+) Dạng cài đặt:***  typedef int item;  typedef struct {  item Elems[100];  int Front;  int Rear;  int count;  } Queue;  Queue Q; | ***+) Dạng cài đặt:***  typedef int item;  typedef struct Node {  item Data;  Node \*next;  } Node;  typedef struct Queue {  Node \*Front, \*Rear;  } |

**Thêm**

***Danh sách kế tiếp***

int Insert\_k (List \*L, item x, int k)//chen x vao vi tri k

{

if (Isfull(\*L)) //kiem tra danh sach day

{

printf("Danh sach day !");

return 0;

}

if (k<1 || k>(\*L).size+1) //kiem tra dieu kien vi tri chen

{

printf("Vi tri chen khong hop le !\n");

return 0;

}

printf ("Nhap thong tin: ");

x = init\_x(); //gan x = ham khoi tao x

int i;

//di chuyen cac phan tu ve cuoi danh sach

for (i = (\*L).size; i >= k; i--)

(\*L).Elems[i] = (\*L).Elems[i-1];

(\*L).Elems[k-1]=x;//chen x vao vi tri k

(\*L).size++;//tang size len 1 don vi.

return 1;

}

***Danh sách kế tiếp***

void Insert\_k (List &L, item x, int k) //chen x vao vi tri k

{

    Node \*P, \*Q = L;

    int i=1;

    if (k<1 || k> len(L)+1) printf("Vi tri chen khong hop le !"); //kiem tra dieu kien

    else

    {

        P = Make\_Node(P,x); //tao 1 Node P

        if (k == 1)

{

P->next = L; //Cho P tro den L

L = P; //L tro ve P

}

        else //chen vao k != 1

        {

            while (Q != NULL && i != k-1) //duyet den vi tri k-1

            {

                i++;

                Q = Q->next;

            }

            P->next = Q->next;

            Q->next = P;

        }

    }

}

***Ngăn xếp kế tiếp***

void Push(Stack &S, item x) //them phan tu vao Stack

{

    if (!Isfull(S))

    {

        S.Data[S.Top] = x; //Gan du lieu

        S.Top ++; //Tang Top len 1

    }

}

***Ngăn xếp liên kết***

void Push(Stack &S, item x) //them phan tu vao Stack

{

    Node \*P = MakeNode(x);

    P->Next = S.Top;

    S.Top = P;

}

***Hàng đợi kế tiếp***

void Push(Queue &Q, item x) //them phan tu vao cuoi Queue

{

    if (Isfull(Q)) printf("Hang doi day !");

    else

    {

        Q.Data[++Q.Rear] = x; //tang Rear len va gan phan tu vao

        Q.count++; //tang so phan tu len

    }

}

***Hàng đợi liên kết***

void Push(Queue &Q, item x) //them phan tu vao cuoi Queue

{

    Node \*P = MakeNode(x); //Neu Q rong

    if (Isempty(Q))

    {

        Q.Front = Q.Rear = P; //dau va cuoi deu tro den P

    }

    else //Khong rong

    {

        Q.Rear->Next = P;

        Q.Rear = P;

    }

    Q.count++; //tang so phan tu len

}

**Xoá**

***Danh sách kế tiếp***

int Del\_k (List \*L, item \*x, int k)

{

if (Isempty(\*L))

{

printf("Danh sach rong !");

return 0;

}

if (k<1 || k>(\*L).size)

{

printf("Vi tri xoa khong hop le !");

return 0;

}

\*x=(\*L).Elems[k-1]; //luu lai gia tri cua phan tu can xoa

int i;

for (i=k; i<(\*L).size-1; i++) //don cac phan tu ve truoc

(\*L).Elems[i]=(\*L).Elems[i+1];

(\*L).size--; //giam size

return 1;

}

***Danh sách liên kết***

void Del\_k (List &L, item &x, int k) //Xoa Node k trong danh sach

{

    Node \*P=L;

    int i=1;

    if (k<1 || k>len(L)) printf("Vi tri xoa khong hop le !");

    else

    {

        if (k==1) Del\_first(L,x); //xoa vi tri dau tien

        else //xoa vi tri k != 1

        {

            while (P != NULL && i != k-1) //duyet den vi tri k-1

            {

                P=P->next;

                i++;

            }

            P->next = P->next->next; //P tro sang Node ke tiep vi tri k

        }

    }

}

***Ngăn xếp kế tiếp***

int Pop(Stack &S) //Loai bo phan tu khoi Stack

{

    if (!Isempty(S))

    {

        S.Top --; //Giam Top

        return S.Data[S.Top]; //Lay du lieu tai Top

    }

}

***Ngăn xếp liên kết***

int Pop(Stack &S) //Loai bo phan tu khoi Stack

{

    if (!Isempty(S))

    {

        item x = S.Top->Data; //luu lai gia tri

        S.Top = S.Top->Next; //Xoa phan tu Top

        return x;

    }

}

***Hàng đợi kế tiếp***

int Pop(Queue &Q) //Loai bo phan tu khoi dau hang doi

{

    if (Isempty(Q)) printf("Hang doi rong !");

    else

    {

        item x = Q.Data[Q.Front];

        for (int i=Q.Front; i<Q.Rear; i++) //di chuyen cac phan tu ve dau hang

            Q.Data[i] = Q.Data[i+1];

        Q.Rear--; //giam vi tri phan tu cuoi xuong

        Q.count--; //giam so phan tu xuong

        return x; //tra ve phan tu lay ra

    }

}

***Hàng đợi liên kết***

int Pop(Queue &Q) //Loai bo phan tu khoi dau hang doi

{

    if (Isempty(Q))

    {

        printf("Hang doi rong !");

        return 0;

    }

    else

    {

        item x = Q.Front->Data;

        if (Q.count == 1) //neu co 1 phan tu

            Init(Q);

        else

            Q.Front = Q.Front->Next;

        Q.count--;

        return x; //tra ve phan tu lay ra

    }

}

**Kiểm tra tính rỗng đầy**

***Hàng đợi kế tiếp***

void Init (List \*L) {

    (\*L).count = 0;

}

int Isempty (List L) {

    Return(L.count == 0);

}

int Isfull (List L) {

    return(L.count == N);

}

***Hàng đợi liên kết***

void Init(List &L) {

    L = NULL;

}

int isempty(List L) {

    Return(L == NULL);

}

***Ngăn xếp kế tiếp***

void Init (Stack &S) {

    S.Top = 0;

}

int Isempty(Stack S) {

    return (S.Top == 0);

}

int Isfull(Stack S) {

    return (S.Top == Max);

}

***Ngăn xếp liên kết***

void Init (Stack &S) {

    S.Top = NULL;

}

int Isempty(Stack S) {

    return (S.Top == NULL);

}

***Hàng đợi kế tiếp***

int Isempty (Queue Q) {

    if (Q.count == 0) return 1;

    return 0;

}

int Isfull (Queue Q) {

    if (Q.count == Max) return 1;

    return 0;

}

***Hàng đợi liên kết***

void Init (Queue &Q) {

    Q.Front = Q.Rear = NULL;

}

int Isempty (Queue Q) {

    if (Q.count == 0) return 1;

    return 0;

}

**Tính độ dài & Tạo một Node**

***Danh sách liên kết***

int Len (List L)

{

    Node \*P=L;

    int i=0;

    while (P!=NULL)

    {

        i++;

        P=P->next;

    }

    return i;

}

Node \*Tao\_Node (Node \*P, item x) {

P=(Node\*) malloc (sizeof(Node));

P->next=NULL;

P->Data=x;

return P;

}

***Ngăn xếp liên kết***

int Len (Stack S)

{

    Node \*P=S.top;

    int i=0;

    while(P!=NULL)

    {

        i++;

        P=P->next;

    }

    return i;

}

Node \*Tao\_Node (item x) {

Node \*P=(Node\*) malloc (sizeof(Node));

P->next=NULL;

P->Data=x;

return P;

}

***Hàng đợi liên kết***

int Len (Queue Q) {

    Node \*P=Q.Front;

    int i=0;

    while(P!=NULL)

    {

        i++;

        P=P->next;

    }

    return i;

}

Node \*Tao\_Node (item x) {

Node \*P=(Node\*) malloc (sizeof(Node));

P->next=NULL;

P->Data=x;

return P;

}

**Nhập/Xuất**

***Danh sách kế tiếp***

void Input (List \*L) {

    int n;

    printf("Nhap so phan tu cua danh sach: ");

    scanf("%d",&(\*L).size);

    int i;

    for (i=0; i<(\*L).size; i++)

    {

        printf("Nhap phan tu thu %d : ",i+1);

        (\*L).Elems[i] = init\_x();

    }

}

void Output (List L) {

    printf("Danh sach: \n");

    int i;

    for (i=0; i<L.size; i++)

        printf("%5d",L.Elems[i]);

    printf("\n");

}

***Danh sách liên kết đơn***

void Input (List &L) {

    int i=0;

    item x;

    do

    {

        i++;

        printf ("Nhap phan tu thu %d : ",i);

        scanf("%d",&x);

        if (x != 0) Insert\_k(L,x,len(L)+1);

    } while(x != 0); //nhap 0 de ket thuc

}

void Output (List L) {

    Node \*P=L;

    while (P != NULL)

    {

        printf("%5d",P->Data);

        P = P->next;

    }

    printf("\n");

}

***Ngăn xếp kế tiếp***

void Input (Stack &S) {

    int n;

    item x;

    do

    {

        printf("Nhap so phan tu cua Stack (<%d): ",Max);

        scanf("%d",&n);

    } while (n>Max || n<1);

    for (int i=0; i<n; i++)

    {

        printf("Nhap phan tu thu %d : ", i+1);

        scanf("%d",&x);

        Push(S,x);

    }

}

void Output (Stack S) {

    for (int i=S.Top-1; i>=0; i--)

        printf("%d   ",S.Data[i]);

    printf("\n");

}

***Ngăn xếp liên kết***

void Input (Stack &S) {

    int i=0;

    item x;

    do

    {

        i++;

        printf ("Nhap phan tu thu %d : ",i);

        scanf("%d",&x);

        if (x != 0) Push(S,x);

    } while(x != 0); //nhap 0 de ket thuc

}

void Output (Stack S) {

    Node \*P = S.Top;

    while (P != NULL)

    {

        printf("%d   ",P->Data);

        P = P->Next;

    }

    printf("\n");

}

***Hàng đợi kế tiếp***

void Input (Queue &Q) //nhap hang doi

{

    int n;

    item x;

    do

    {

        printf("Nhap so phan tu cua Queue (<%d): ",Max);

        scanf("%d",&n);

    } while (n>Max || n<1);

    for (int i=0; i<n; i++)

    {

        printf("Nhap phan tu thu %d: ", i+1);

        scanf("%d",&x);

        Push(Q,x);

    }

}

void Output(Queue Q)

{

    if (Isempty(Q)) printf("Hang doi rong !");

    else

    {

        for (int i=Q.Front; i<=Q.Rear; i++)

            printf("%d   ",Q.Data[i]);

        printf("\n");

    }

}

***Hàng đợi liên kết***

void Input (Queue &Q) //nhap hang doi

{

    int i=0;

    item x;

    do

    {

        i++;

        printf ("Nhap phan tu thu %d: ",i);

        scanf("%d",&x);

        if (x != 0) Push(Q,x);

    } while(x != 0); //nhap 0 de ket thuc

}

void Output(Queue Q)

{

    Node \*P = Q.Front;

    while (P != NULL)

    {

        printf("%d   ",P->Data);

        P = P->Next;

    }

    printf("\n");

}

**Cây**

**Dạng cài đặt**

|  |  |
| --- | --- |
| **Cây đa phân (tổng quát)** | **Cây đa phân bằng con trưởng và em liền kề** |
| ***+ Dạng cài đặt***  typedef struct Node  {  int Key;  Node \*left;  Node \*right;  } Node;  typedef Node \*Tree; | ***+ Dạng cài đặt bằng mảng***  typedef struct Node {  int ID;  Node \*Next;  } Node;  typedef struct Member {  int Infor;  struct Node\* Child;  } Member;  typedef struct Tree {  Member elems[N];  int size;  } Tree; |
| **Cây nhị phân** | **Cây đa phân bằng con trưởng và em liền kề** |
| ***+ Dạng cài đặt***  typedef struct Node {  item data;  Node \*left,\*right;  } Node;  Node \*bTree2; | ***+ Dạng cài đặt bằng con trỏ***  typedef struct Node {  item Info;  int EldestChild; //Con truong  int NextSibling; //Em lien ke  } Node;  typedef struct Tree {  struct Node Elems[N];  int size;  } Tree; |
| **Cây nhị phân tìm kiếm** |  |
| typedef struct Node {  int key;  int data;  Node \*left, \*right;  } Node;  typedef Node \*SearchTree; |  |

**Tìm kiếm trong cây**

Node\* search (SearchTree T, int x) //Tim kiem 1 khoa x tren cay NPTK T

{

Node\* V=T;

int found=0;

while ((V!=NULL) && (found==0))

if (x<V->key) V->left;

else if (x>V->key) V=V->right;

else found=1;

return V;

}

**Thêm một nút vào cây**

void insertTree (SearchTree &T, int x) {

Node \*V,\*P,\*N;

int found=0;

N=(Node\*) calloc (1,sizeof(Node)); // cap phat 1 nut

if (N!=NULL)

{

N->left=NULL;

N->right=NULL;

N->key=x;

if (T==NULL) T=N; // neu cay rong, nut moi them chinh la goc cua cay

else

{

V=T;

while ((V!=NULL)&&(!found))

{

if (x<V->key)

{

P=V;

V=V->left;

}

else if (x>V->key)

{

P=V;

V=V->right;

}

else found=1;

}

if (found==0)

{

if (x<P->key) P->left =N; //them N vao ben trai cua P

if (x>P->key) P->right=N;

}

}

}

else printf("\n Khong cap phat duoc bo nho");

}

**Duyệt cây NPTK**

***Thứ tự trước (PreOrder):***

void PreOrder (SearchTree T)

{

if (T!=NULL)

{

printf(‘‘%5d",T->key);

PreOrder (T->left);

PreOrder (T->right);

}

}

***Thứ tự giữa (InOrder):***

void InOrder(SearchTree T)

{

if (T!=NULL)

{

InOrder (T->left);

printf(‘‘%5d",T->key);

InOrder (T->right);

}

}

***Thứ tự sau (PostOrder)***

void PostOrder(SearchTree T)

{

if (T!=NULL)

{

PostOrder (T->left);

PostOrder (T->right);

printf(‘‘%5d",T->key);

}

}

**Duyệt cây đa phân con trưởng và em liền kề bằng con trỏ**

***Duyệt cây theo thứ tự trước***

void PreOrder (Tree T) {

NodeType C;

if (T==rong) printf("cay rong");

else

{

Visit (T);// tham goc hay in goc

C = eldestChild (T); // con truong

while (C!=rong)

{

PreOrder (C);

C = nextSibling (C);// em lien ke

}

}

}

***Duyệt cây theo thứ tự giữa***

void InOrder (Tree T)

{

NodeType C;

if (T==rong) printf ("Cay rong");

else

{

C = eldestChild (T);// tim con truong

if (C!= rong)

{

InOrder(C);

C = nextSibling(C);//tim em lien ke

}

Visit(T); //tham goc

while (C!=rong)

{

InOrder (C);

C = nextSibling(C);//em lien ke

}

}

}

***Duyệt cây theo thứ tự sau***

void PostOrder (Tree T)

{

NodeType C;

if (T==rong) printf("cay rong");

else

{

C = eldestChild(T);// con truong

while (C!= rong)

{

PostOrder(C);

C = nextSibling(C);// em lien ke

}

Visit(T); //Tham cha hay in cha

}

}

***Duyệt cây nhị phân cài đặt bằng con trỏ***

***Hàm duyệt theo thứ tự trước:***

void PreOrder(bTree2 \* r)

{

if (r != NULL)

{

printf("%c", r->data);

PreOrder (r->lefltChild);

PreOrder (r->rightChild);

}

}

***Hàm duyệt theo thứ tự giữa:***

void InOrder(bTree2 \* r)

{

if (r != NULL)

{

InOrder (r->leftChild);

printf("%c", r->data);

InOrder (r->rightChild);

}

}

***Hàm duyệt theo thứ tự sau:***

void PostOrder(bTree2 \* r)

{

if (r != NULL)

{

PostOrder (r->leftChild);

PostOrder (r->rightChild);

printf("%c", r->data);

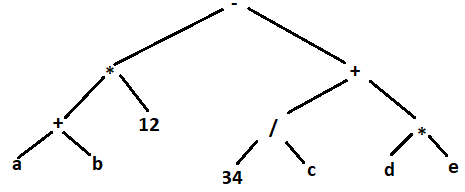
}

}

**(a+b)\*12-34/c+d\*e**

**<=> ((a+b)\*12))-(34/c)+(d\*e)**

**Cây nhị phân**

**Duyệt trước:** - \* + a b 12 + / 34 c \* d e

**Duyệt giữa:** a + b \* 12 – 34 / c + d \* e

**Duyệt sau:** a b + 12 \* 34 c / d e \* + -