**Bài tập chương 2 – Câu hỏi**

**Câu 1:**

* Khái niệm danh sách đặc:
* **Danh sách đặc** là một danh sách mà các phần tử trong danh sách có cùng kiểu dữ liệu, và được cấp phát liên tục trong bộ nhớ.
* Ví dụ:
* Có danh sách gồm n = 7 phần tử được lưu trữ trong một danh sách đặc tại các vị trí từ 0 đến 6( gồm các phần tử a[0], a[1], a[2], …, a[6]) , tìm kiếm giá trị x = 30 trong danh sách:

a[0] = 10 a[1] = 50 a[2]= 20 a[3] = 70

a[4] = 30 a[5] = 60 a[7] = 40

int **search** ( int **a**[], int **n**, int **x** )

{

int I = 0;

while ( ( I < n ) && ( a[i] != x) )

i++;

if ( I == n )

return -1;

return I;

}

**Câu 2:**

* Khái niệm danh sách liên kết **( Linked list )**
* Là một dãy các cấu trúc dữ liệu được kết nối với nhau thông qua các liên kết **(link).** Hiểu một cách đơn giản thì Danh sách liên kết là một cấu trúc dữ liệu bao gồm một nhóm các nút **(node)** tạo thành một chuỗi. Mỗi nút gồm dữ liệu ở nút đó và tham chiếu đến nút kế tiếp trong chuỗi.
* Danh sách liên kết có 2 loại:
* **Danh sách liên kết đơn** là một danh sách mà các phần tử được cấp phát rời rạc nhau và cố định trong bộ nhớ. Mỗi phần tử trong danh sách gồm có 2 thành phần :
* **Phần 1:** vùng thông tin chứa giá trị cần quản lí.
* **Phần 2:** vùng liên kết, chứa địa chỉ bộ nhơ của phần tử kế tiếp.
* **Danh sách liên kết kép** là một danh sách liên kết mà mỗi phần tử trong danh sách bao gồm 3 thành phần:
* **Phần 1:** vùng chứa thông tin ( info )
* **Phần 2:** vùng liên kết ( next ) trỏ đến phần tử đứng liền sau nó
* **Phần 3:** vùng liên kết ( previous ) trỏ đến phần tử đứng liền trước nó
* Ví dụ:
* **Danh sách liên kết đơn**

+ Duyệt một danh sách gồm n phần tử, phần tử đầu tiên được gán bằng biến first và phần tử cuối cùng được gắn bằng biến NULL. Khi duyệt ta sẽ duyệt từ phần tử đầu tiên ( first ) đi qua con trỏ liên kết tới nút liên kết của phần tử sau nó và khi nào con trỏ gặp NULL thì ta đã duyệt xong danh sách.

* **Danh sách liên kết đôi**

+ Một DSLK đôi với 4 mắt xích. DSLK đôi so với DSLK đơn cần nhiều bộ nhớ hơn để lưu trữ thêm một con trỏ. Các thao tác xóa và thêm mới cũng lâu hơn vì ta còn phải cập nhật cả con trỏ prev mỗi khi ta cần xóa hoặc thêm vào DSLK. Về điểm mạnh, DSLK đôi mềm dẻo hơn vì từ một nút, ta có thể đi đến nút trước nó mà không phải duyệt từ đầu DSLK. Tính mềm dẻo này được Knuth [2] khai thác triệt để trong thiết kế cấu trúc Dancing Links, một cấu trúc dữ liệu hỗ trợ các thuật toán quay lui cho bài toán liệt kê tổ hợp.

**Câu 3:**

* **STACK** và **QUEUE** là danh sách hạn chế vì :
* Vì khi thêm và lấy ra đều tuân theo qui tắc.
* Giới hạn về kích thước.
* Stack & Queue là cấu trúc dữ liệu rỗng.
* Cấp phát rời rạc vô hạn.
* Cấu trúc dữ liệu hiệu tính, cấp phát liên tục, có giơi hạn.
* Ví dụ:
* **STACK**

+ Chuyển đổi số thập phân sang nhị phân: Khi chuyển đổi số thập phân sang nhị phân, chúng ta sẽ thưc hiện thao tác chia số thập phân này cho 2 và viết phần dư ngược lại thứ tự mà nó được sinh ra. Sử dụng stack, ta lưu lần lượt phần dư sau mỗi lần chia và sau khi kết thúc thao tác chia, đọc stack vừa rồi sẽ cho ra biểu diễn nhị phân cần tìm.

**Câu 4:**

* Khái niệm LIFO, FIFO:
* **Ngăn xếp (Stack)** là một danh sách các phần tử được quản lý theo thứ tự như sau: Phần tử được *thêm vào* ngăn xếp *sau*, sẽ được lấy ra **(xóa)** khỏi ngăn xếp trước**. Last In First Out – LIFO.**
* **Hàng đợi (Queue)** là danh sách chứa các phần tử được quản lý theo thứ tự sau: Phần tử được thêm vào trước, sẽ được lấy ra **(xóa)** trước. **First In First Out – FIFO.**
* Ví dụ:
* **LIFO**

int **Push** ( int **a**[], int **&sp**, int **x** )

{

if (sp < MAX - 1)

{

a[++sp] = x;

return 1;

}

return 0;

}

* **FIFO**

int **Push**( int **a**[], int **&front**, int **&rear**, int **x** )

{

if (rear – front = MAX - 1) *// hang đợi đầy*

return 0;

else

{

if (front == -1) *// hang đợi rỗng*

front = 0;

if (rear == MAX - 1) *// hang đợi bị tràn*

{

for(int i = front; i<=rear; i++)

// thực hiện dời tịnh tiến các phần tử trong hàng

a[i-front]=a[i];

rear = MAX -1-front;

//*rear nhận giá trị mới, sau khi tinh tiến*

front = 0; *// front nhận giá trị mới, sau khi tịnh tiến*

}

a[++rear] = x;

return 1;

}

}

**Câu 6:**

* Khử đệ qui, tổ chức lưu vết các quá trình tìm kiếm theo chiều sâu và quay lui, vét cạn, ứng dụng trong các bài toán tính toán biểu thức.
* Trong trình biên dịch **(thông dịch)**, khi thực hiện các thủ tục, DSLK được sử dụng để lưu môi trường của các thủ tục.
* Trong một số bài toán của lý thuyết đồ thị (như tìm đường đi), DSLK cũng thường được sử dụng để lưu dữ liệu khi giải các bài toán.
* Bài toán sản xuất và tiêu thụ (ứng dụng trong các hệ điều hành song song).
* Bộ đệm (ví dụ: Nhấn phím -> Bộ đệm -> CPU xử lý).
* Xử lý các lệnh trong máy tính (ứng dụng trong HÐH, trình biên dịch), hàng đượi các tiến trình chờ được xử lý,…
* Danh sách có nhiều mối liên kết thường được xử dụng trong các ứng dụng quản lý một cơ sở dữ liệu lớn với những nhu cầu tìm kiếm dữ liệu theo những khoá khác nhau.

**Câu 7:**

* Cấu trúc dữ liệu động:
* Danh sách liên kết cấp phát động lúc chạy chương trình. Các phần tử nằm rải rác ở nhiều nơi trong bộ nhớ. Kích thước danh sách chỉ bị giới hạn do RAM. Thao tác thêm xoá đơn giản.
* Cần xây dựng cấu trúc dữ liệu đáp ứng được các yêu cầu: Linh động hơn. Có thể thay đổi kích thước, cấu trúc trong suốt thời gian sống.
* Ví dụ:
* Một số đối tượng dữ liệu trong chu kỳ sống của nó có thể thay đổi về cấu trúc, độ lớn, như danh sách các học viên trong một lớp học có thể tăng thêm, giảm đi ... Khi đó nếu cố tình dùng những cấu trúc dữ liệu tĩnh đã biết như mảng để biểu diễn những đối tượng đó lập trình viên phải sử dụng những thao tác phức tạp, kém tự nhiên khiến chương trình trở nên khó đọc, do đó khó bảo trì và nhất là khó có thể sử dụng bộ nhớ một cách có hiệu quả.