TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA TOÁN - TIN HỌC

TẠ THỊ THU PHƯỢNG

THỰC HÀNH CẤU TRÚC VÀ GIẢI THUẬT 1

(Bài Giảng Tóm Tắt)

-- Lưu hành nội bộ --

Đà Lạt 2008

LỜI MỞ ĐẦU

Cùng với học phần "Cấu trúc dữ liệu và giải thuật 1", học phần "Thực hành Cấu trúc dữ liệu và giải thuật 1" nhằm cung cấp cho sinh viên các kiến thức căn bản và kỹ năng thực hành trên các cấu trúc dữ liệu cơ sở có cấu trúc tĩnh và động (thông qua danh sách liên kết và cây, chủ yếu là cây nhị phân) cũng như các thuật toán cơ bản liên quan đến chúng như sắp xếp, tìm kiếm ở bộ nhớ trong. Để có thể nắm bắt các kiến thức được trình bày trong giáo trình này, sinh viên cần có các kiến thức về tin học đại cương và nhập môn lập trình. Các kiến thức trong học phần này sẽ tạo điều kiện cho học viên tiếp tục dễ dàng nắm bắt các kiến thức các học phần tin học về sau như: cấu trúc dữ liệu và giải thuật 2, toán rời rạc, đồ hoạ, hệ điều hành, trí tuệ nhân tạo, ...

Nội dung giáo trình được trình bày thông qua 10 bài thực tập. Mỗi bài thực tập đều có phần hệ thống lại nội dung lý thuyết và phần bài tập thực hành cũng như bài tập nâng cao.

Chắn chắn rằng trong giáo trình sẽ còn nhiều khiếm khuyết, tác giả mong muốn nhận được và rất biết ơn các ý kiến đóng góp quí báu của đồng nghiệp cũng như bạn đọc để giáo trình được hoàn thiện hơn nữa về mặt nội dung cũng như hình thức trong lần tái bản sau.

Đà lạt, 5/2008
Tác giả

MỤC LỤC

Chương 1: GIỚI THIỆU CẤU TRÚC DỮ LIỆU – PHÂN TÍCH THUẬT TOÁN

BÀI THỰC HÀNH SỐ 1

(4 tiết)

Mục tiêu

Thống nhất một số chuẩn và quy ước trong lập trình.
Nắm kiểu dữ liệu có cấu trúc và các thao tác trên chúng.

Nội dung

- Ôn lại kiểu dữ liệu có cấu trúc (kiểu định nghĩa bằng từ khóa struct)
- Các quy ước lập trình.

Yêu cầu

Nắm vững phương pháp lập trình cấu trúc trong C/C^{++} và biết cách sử dụng môi trường lập trình Visual C++ 6.0.

1. CÁC QUY ƯỚC LẬP TRÌNH

Quy ước đặt tên hằng

Trong Visual C^{++} 6.0, hằng số được khai báo bằng từ khóa "#define". Một số quy ước trong việc đặt tên hằng như sau:

i) Tên hằng phải thể hiện được ý nghĩa của nó.

ii) Tên hằng được viết hoa toàn bộ và các từ trong tên cách nhau bằng ký tự "_".

Quy ước đặt tên biến

i) Tên biến phải thể hiện được ý nghĩa của nó.

```
int  t, m;    // Không rõ nghĩa.
int  iTuSo, iMauSo; // Rõ nghĩa.
```

ii) Tên biến được viết hoa các ký tự đầu mỗi từ trong tên, các ký tự còn lại viết thường.

```
int  ituso, imauso;  // Không nên
```

```
int  iTuso, iMauso; // Không nên
int  iTuSo, iMauSo; // nên
```

iii) Tên biến có phần tiếp đầu ngữ (prefix) thể hiện kiểu dữ liệu của biến (phong cách Hungarian):

Kiểu dữ liệu số	
char – c short – s int – i long – l float – f double – d	char cKyTu; short sSoNguyenNgan; int iSoNguyen; long lSoNguyenDai; float fSoThuc; double dSoThucDai; int nSo;

Kiểu dữ liệu luận lý	
bool - b	bool bLuanLy;

Kiểu dữ liệu mảng	
[] – arr	int arrSoNguyen[50]; HocSinh arrDanhSach[50];

Kiểu dữ liệu chuỗi	
char *, char [] – str	char *strChuoi; char strChuoi[50];

Kiểu dữ liệu con trỏ	
* - p	int *pConTro; HocSinh *pDanhSach;

Quy ước đặt tên kiểu dữ liệu tự định nghĩa

i) Tên kiểu dữ liệu tự định nghĩa (struct) thường là danh từ và phải thể hiện được ý nghĩa của kiểu dữ liệu đó.

```
struct TinhPhanSo    // không nên
struct PhanSo    // nên
```

ii) Tên kiểu dữ liệu tự định nghĩa được viết hoa các ký tự đầu mỗi từ trong tên, các

ký tự còn lại viết thường. Ví dụ

```
struct PhanSo
```

Quy ước đặt tên hàm

i) Tên hàm thường là động từ và phải thể hiện hành động cần thực hiện.

```
int DataFile(char *strFileName)  // không nên
int LoadDataFile(char *strFileName) // nên.

int BadValue(long lValue)  // không nên
int CheckForBadValue(long lValue) // nên
```

ii) Tên hàm được viết hoa các ký tự đầu mỗi từ trong tên, các ký tự còn lại viết thường.

```
int checkforbadvalue(long lValue) // không nên
int CheckforBadvalue(long lValue) // không nên
int CheckForBadValue(long lValue) // Nên.
```

Quy ước viết câu lệnh

i) Viết mỗi câu lệnh riêng trên một dòng.

```
// Không nên.
x = 3; y = 5;

// nên viết
x = 3;
y = 5;
```

ii) Viết các dấu "{" "}" riêng trên một dòng.

```
void Swap(int &a, int &b)
{
        int c = a;
        a = b;
        b = c;
}
```

iii) Viết các câu lệnh if, while, for riêng trên một đoạn.

```
if (a > b)
        printf( "a lon hon b");
```

```
for (int i = 0; i < n; i++)
        x = x + 5;

k = k * x;
```

iv) Viết các câu lệnh cùng thực hiện một công việc riêng trên một đoạn.

```
int c = a;
a = b;
b = c;

k = k * a;
x = b + c;
```

Quy ước cách khoảng

i) Viết cách vào một khoảng tab đối với các câu lệnh nằm giữa dấu "{" "}".

```
// không nên
void Swap(int &a, int &b)
{
int c = a;
a = b;
b = c;
}
```

```
// nên viết
void Swap(int &a, int &b)
{
        int c = a;
        a = b;
        b = c;
}
```

ii) Viết cách vào một khoảng tab đối với câu lệnh ngay sau if, else, while, for.

```
// Không nên.
if (a > b)
printf( "a lon hon b");
else
printf("a nho hon  hoac bang b");

for (int i = 0; i < n; i++)
x = x + 5;
```

```
// nên viết
if (a > b)
        printf( "a lon hon b");
else
        printf("a nho hon  hoac bang b");


for (int i = 0; i < n; i++)
        x = x + 5;
```

iii) Viết cách một khoảng trắng xung quanh các toán tử 2 ngôi.

```
x=x+5*a-c; // Không nên
x = x + 5 * a - c; // nên
```

```
if (a>=b) // Không nên
if (a >= b) // nên
```

iv) Viết cách một khoảng trắng sau các dấu “,” “;”.

```
void CalculateValues(int a,int b,int c); // không nên
void CalculateValues(int a, int b, int c); // nên

for (int i = 0;i < n;i++) // không nên
for (int i = 0; i < n; i++) // nên
```

Quy ước viết chú thích

Trong C++, chúng ta dùng dấu “//” hoặc “/*” “*/” để viết chú thích cho chương trình. Một số quy ước khi viết chú thích như sau:

i) Chú thích phải rõ ràng, dễ hiểu và diễn giải được ý nghĩa của đoạn lệnh, hàm...
ii) Dùng dấu “//” thay cho “/*” “*/” khi viết chú thích.

```
/* void Swap(int &a, int &b)          | // nên dùng
{                                     | //void Swap(int &a, int &b)
        int c = a;                    | //{
        a = b;                        | // int c = a;
        b = c;                        | // a = b;
} */                                  | // b = c;
                                      | //}
```

2. BÀI TẬP

Giả sử quy tắc tổ chức quản lý nhân viên của một công ty như sau:

- Thông tin về một nhân viên bao gồm lý lịch và bảng chấm công:

**** Lý lịch nhân viên:***

- Mã nhân viên	: chuỗi 10 ký tự
- Tên nhân viên	: chuỗi 30 ký tự
- Tình trạng gia đình	: 1 ký tự (M = Married, S = Single)
- Số con	: số nguyên ≤ 20
- Trình độ văn hoá	: chuỗi 2 ký tự (C1 = cấp 1, C2 = cấp 2, C3=cấp 3; DH = đại học, CH = cao học, TS = tiến sĩ)
- Lương căn bản	: số ≤ 1 000 000

**** Chấm công nhân viên:***

- Số ngày nghỉ có phép trong tháng : số ≤ 28
- Số ngày nghỉ không phép trong tháng : số ≤ 28
- Số ngày làm thêm trong tháng : số ≤ 28
- Kết quả công việc : chuỗi 2 ký tự (T = tốt, TB = trung bình, K = Kém)
- Lương thực lĩnh trong tháng : số ≤ 2 000 000

- Quy tắc tính lương:
  Lương thực lĩnh = Lương căn bản + Phụ trội
  Trong đó nếu:
  - số con > 2 : Phụ trội = +5% Lương căn bản
  - trình độ văn hoá = CH : Phụ trội = +10% Lương căn bản
  - làm thêm : Phụ trội = +4% Lương căn bản / 1 ngày
  - nghỉ không phép : Phụ trội = -5% Lương căn bản / 1 ngày

- Các chức năng yêu cầu:
  - Cập nhật lý lịch, bảng chấm công cho nhân viên (thêm, xóa, sửa một hay mọi mẫu tin thoả mãn một tính chất nào đó)
  - Xem bảng lương hàng tháng
  - Khai thác (chẳng hạn tìm) thông tin của nhân viên

Hãy chọn cấu trúc dữ liệu thích hợp (và giải thích tại sao?) để biểu diễn các thông tin trên và cài đặt chương trình theo các chức năng đã mô tả. Biết rằng số nhân viên tối đa là 50 người, chú ý các thông tin tĩnh và "động" hay thay đổi và là hệ quả của những thông tin khác.

Chương 2: TÌM KIẾM VÀ SẮP XẾP

BÀI THỰC HÀNH SỐ 2

Các phương pháp tìm kiếm
(4 tiết)

Mục tiêu

Cài đặt các phương pháp tìm kiếm, so sánh các phương pháp.

Nội dung lý thuyết

0. PHÁT BIỂU BÀI TOÁN

Cho dãy *a* gồm *N* phần tử, cần tìm *x* trong dãy *a*.

1. CÁC PHƯƠNG PHÁP TÌM KIẾM

a) Phương pháp tìm kiếm tuyến tính

Ý tưởng

So sánh *x* lần lượt với phần tử thứ 1, thứ 2,…của dãy *a* cho đến khi gặp phần tử có khóa cần tìm, hoặc đã tìm hết dãy mà không thấy *x*.

Giải thuật

Bước 1: $i = 1$; // bắt đầu từ phần tử đầu tiên của dãy

Bước 2: So sánh a[i] với x.

+ Nếu a[i] = x: Tìm thấy. Dừng.
+ Nếu a[i] ≠ x: Sang Bước 3.

Bước 3: $i = i + 1$; // xét tiếp phần tử kế trong dãy

Nếu $i \le N$: lặp lại Bước 2.

Ngược lại: Hết dãy. Không tìm thấy. Dừng

b) Phương pháp tìm kiếm tuyến tính có lính canh

Ý tưởng

- Đặt một phần tử có giá trị *x* vào cuối dãy, gọi đây là phần tử "lính canh". Như vậy, ta bảo đảm luôn tìm thấy *x* trong dãy, và dựa vào vị trí tìm thấy để đưa ra kết luận.
- *Phương pháp cải tiến này giúp giảm bớt một phép so sánh trong vòng lặp.*

Giải thuật

Bước 1: $i = 1$;

a[N+1] = x; // phần tử "lính canh"

Bước 2: So sánh a[i] với x.

+ Nếu a[i] = x: Sang Bước 3
+ Nếu a[i] ≠ x: $i = i + 1$; Lặp lại bước 2.

Bước 3: Nếu $i \le N$: tìm thấy x tại vị trí i.

Ngược lại: không tìm thấy x trong dãy.

c) Phương pháp tìm kiếm nhị phân

Ý tưởng

Đối với những *dãy số đã có thứ tự* (tăng dần), các phần tử đã có quan hệ $a_{i-1} \le a_i \le a_{i+1}$. Nếu $x > a_i$ thì x chỉ có thể xuất hiện trong đoạn $[a_{i+1}, a_N]$, ngược lại nếu $x < a_i$ thì x chỉ có thể xuất hiện trong đoạn $[a_1, a_{i-1}]$.

Giải thuật áp dụng nhận xét trên để giới hạn phạm vi tìm kiếm sau mỗi lần so sánh x với một phần tử trong dãy. Tại mỗi bước, so sánh x với phần tử nằm giữa dãy tìm kiếm hiện hành, dựa vào kết quả so sánh để quyết định giới hạn của dãy tìm kiếm ở bước kế tiếp là nửa trên hay nửa dưới của dãy hiện hành.

Giải thuật

Bước 1: *left* = 1; *right* = N; // tìm kiếm trên tất cả các phần tử

Bước 2: *mid* = (*left* + *right*)/2; // lấy mốc so sánh

So sánh a[*mid*] với x

+ a[*mid*] = x: Tìm thấy. Dừng.
+ a[*mid*] > x: Tìm tiếp x trong dãy con $a_{left}..a_{mid-1}$

  right = *mid* – 1;
+ a[*mid*] < x: Tìm tiếp x trong dãy con $a_{mid+1}..a_{right}$

  left = *mid* + 1;

Bước 3: Nếu *left* ≤ *right* Lặp lại bước 2. // còn phần tử chưa xét

Ngược lại: Không tìm thấy x trong dãy. Dừng

2. BÀI TẬP

1. Cài đặt các thuật toán tìm kiếm
2. Xây dựng và cài đặt thuật toán tìm:
   a) Phần tử lớn nhất (hay nhỏ nhất).
   b) Tất cả các số nguyên tố.
   c) Tìm phần tử đầu tiên trên dãy mà thỏa một tính chất TC nào đó.
   d) Dãy con (là một dãy các phần tử liên tiếp của dãy) tăng dài nhất trong một dãy các phần tử cho trước được cài đặt bằng mảng.

Yêu cầu

- Nắm vững nội dung lý thuyết và các bài thực tập đã tiến hành.
- Kỹ thuật lập trình module.

BÀI THỰC HÀNH SỐ 3
Các phương pháp sắp xếp
(4 tiết)

Mục tiêu

Cài đặt và sử dụng các phương pháp sắp xếp: sắp xếp chọn, sắp xếp chèn, sắp xếp đổi chỗ.

Nội dung lý thuyết

0. PHÁT BIỂU BÀI TOÁN SẮP XẾP

Xét một dãy *a* gồm *N* phần tử. Cần sắp xếp các phần tử của dãy để thu được một dãy có thứ tự (ví dụ tăng dần/giảm dần theo một khóa được chỉ định). Các thuật toán sắp xếp được trình bày sau đây giả sử dãy *a* là một dãy số nguyên.

1. PHƯƠNG PHÁP SẮP XẾP CHỌN

Ý tưởng:

- Chọn phần tử nhỏ nhất *x* trong *N* phần tử ban đầu, đảo vị trí của *x* với phần tử đầu dãy để đưa *x* về đầu dãy.
- Ta không quan tâm đến phần tử đầu dãy nữa, xem dãy bây giờ chỉ còn *N*-1 phần tử, bắt đầu từ vị trí thứ 2.
- Lặp lại xử lí trên cho đến khi dãy hiện hành chỉ còn một phần tử.

Giải thuật:

Bước 1: $i = 1$

Bước 2: Tìm phần tử $a[min]$ nhỏ nhất trong dãy $a[i..N]$.

Bước 3: Hoán vị $a[min]$ và $a[i]$

Bước 4: Nếu $i \le N - 1$ thì $i = i + 1$. Lặp lại Bước 2

Ngược lại: Dừng. // *N*-1 phần tử đã nằm đúng vị trí.

2. PHƯƠNG PHÁP SẮP XẾP CHÈN

a) Sắp xếp chèn đơn giản (Insertion Sort)

Ý tưởng

Xét dãy $a_1, a_2 \ldots, a_n$, trong đó i-1 phần tử đầu tiên $a_1, a_2, \ldots, a_n$ đã có thứ tự. Tìm vị

trí thích hợp để chèn phần tử a_i vào vị trí thích hợp trong i-1 phần tử đã sắp để có dãy mới $a_1, a_2,\ldots,a_i$ trở nên có thứ tự. Vị trí này nằm giữa a_{k-1} và a_k thỏa $a_{k-1} \le a_i < a_k$ $(1 \le k < i)$.

Giải thuật

Bước 1: $i = 2$ // Giả sử có đoạn $a[1]$ đã được sắp

Bước 2: $x = a[i]$; Tìm vị trí *pos* thích hợp trong đoạn $a[1]$ đến $a[i-1]$ để chèn $a[i]$ vào.

Bước 3: Dời chỗ các phần tử từ a[*pos*] đến $a[i-1]$ sang phải 1 vị trí để dành chỗ cho $a[i]$.

Bước 4: $a[pos] = x$; // có đoạn $a[1]..a[i]$ đã được sắp

Bước 5: $i = i + 1$;

Nếu $i \le n$: lặp lại bước 2.

Ngược lại: Dừng

***b) Sắp xếp chèn cải tiến* (Shell Sort)**

Ý tưởng

- Shell Sort là phương pháp cải tiến của Insertion Sort.
- Phân chia dãy ban đầu thành những dãy con gồm các phần tử cách nhau h vị trí.

  Dãy con thứ nhất: $a_1, a_{h+1}, a_{2h+1}\ldots$

  Dãy con thứ hai: $a_2, a_{h+2}, a_{2h+2}\ldots$

  ……..

  Dãy con thứ h: $a_h, a_{2h}, a_{3h}\ldots$
- Sắp xếp các phần tử trong cùng dãy con sẽ làm cho các phần tử được đưa về vị trí đúng tương đối (so với toàn bộ các phần tử trong dãy ban đầu có thể chưa đúng). Giảm khoảng cách **h** để tạo các dãy con mới và lại tiếp tục sắp xếp.
- Thuật toán dừng khi **h** = 1, lúc này bảo đảm tất cả các phần tử trong dãy ban đầu đã được so sánh với nhau để xác định trật tự đúng cuối cùng.

Giải thuật

Bước 1: Chọn ***k*** khoảng cách $h[1], h[2],\ldots, h[k]$; $i = 1$;

Bước 2: Phân chia dãy ban đầu thành các dãy con cách nhau $h[i]$ khoảng cách. Sắp xếp từng dãy con bằng Insertion Sort.

Bước 3: $i = i + 1$;

Nếu $i \le k$: Lặp lại bước 2

Nếu $i > k$: Dừng

3. PHƯƠNG PHÁP SẮP XẾP ĐỔI CHỖ

a) Sắp xếp đổi chỗ đơn giản (Bubble Sort)

Ý tưởng

- Xét dãy gồm N phần tử
- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đầu dãy hiện hành.
- Ta không quan tâm đến phần tử đầu dãy nữa, xem dãy bây giờ chỉ còn N-1 phần tử, bắt đầu từ vị trí thứ 2.
- Lặp lại xử lí trên cho đến khi không còn cặp phần tử nào để xét.

Giải thuật

Bước 1: $i = 1$;

Bước 2: $j = N$; // Duyệt từ cuối dãy ngược về vị trí i

Trong khi $j > i$ thực hiện

Nếu $a[j] < a[j\text{-}1]$: hoán vị $a[j]$ và $a[j\text{-}1]$; // xét cặp phần tử kế cận

$j = j - 1$;

Bước 3: $i = i + 1$

Nếu $i \le N - 1$: Lặp lại bước 2.

Ngược lại: Hết dãy. Dừng.

b) Sắp xếp đổi chỗ cải tiến (Shaker Sort)

Ý tưởng

- Shaker Sort cũng dựa trên nguyên tắc đổi chỗ trực tiếp nhưng tìm các khắc phục nhược điểm của Bubble Sort. Trong mỗi lần sắp xếp, duyệt mảng theo 2 lượt từ 2 phía khác nhau.
  - Lượt đi: đẩy phần tử nhỏ về đầu mảng
  - Lượt về: đẩy phần tử lớn về cuối mảng
- Ghi nhận lại những đoạn đã sắp xếp nhằm tiết kiệm các phép so sánh thừa.

Giải thuật

Bước 1: $l = 1$; $r = n$; // từ l đến r là đoạn cần sắp xếp

$k = n$; // ghi nhận vị trí k xảy ra hoán vị sau cùng

// để làm cơ sở thu hẹp đoạn l đến r.

Bước 2:

Bước 2a: $j = r$; // đẩy phần tử nhỏ về đầu mảng

Trong khi ($j > l$) thực hiện

Nếu $a[j] < a[j\text{-}1]$: hoán vị $a[j]$ và $a[j\text{-}1]$;

$k = j$; // lưu lại nơi xảy ra hoán vị.

$j = j - 1$;

$l = k$; // loại các phần tử đã có thứ tự ở đầu dãy

Bước 2b: $j = l$; // đẩy phần tử lớn về cuối mảng

Trong khi ($j < r$) thực hiện

Nếu $a[j] > a[j\text{-}1]$: hoán vị $a[j]$ và $a[j+1]$;

$k = j$; // lưu lại nơi xảy ra hoán vị.

$j = j + 1$;

$r = k$; // loại các phần tử đã có thứ tự ở cuối dãy

Bước 3: Nếu $l <$ r : Lặp lại bước 2.

Ngược lại: Dừng.

BÀI TẬP

1. Cài đặt các phương pháp sắp xếp trên.
2. Tổ chức hàm main theo dạng menu cho phép người dùng lựa chọn phương pháp sắp xếp cần áp dụng.

BÀI THỰC HÀNH SỐ 4

Các phương pháp sắp xếp (tt)

(4 tiết)

Mục tiêu

Cài đặt và sử dụng các phương pháp sắp xếp: phân hoạch, trên cây có thứ tự, trộn, dựa trên cơ số.

Nội dung lý thuyết

1. QUICK SORT

Ý tưởng

Chọn x là giá trị của một phần tử tùy ý trong dãy ban đầu. Vị trí phần tử thường được chọn là $k = (l + r)/2$.

Thực hiện phân hoạch với x:

$a_k < x$	$a_k = x$	$a_k > x$
$k = 1..i$	$k = i..j$	$k = j..N$

Dãy ban đầu được chia làm 3 phần, trong đó dãy con thứ 2 đã có thứ tự. Dãy ban đầu chỉ có thứ tự nếu dãy con 1 và 3 cũng có thứ tự. Nếu dãy con 1 và 3 chỉ có một phần tử thì chúng đã có thứ tự, ngược lại, ta lần lượt tiến hành phân hoạch từng dãy con theo phương pháp phân hoạch như trên.

Giải thuật phân hoạch

Bước 1: Chọn tùy ý một phần tử $a[k]$ trong dãy là giá trị mốc, $l \le k \le r$

Bước 2: Phát hiện và hiệu chỉnh cặp phần tử $a[i]$, $a[j]$ nằm sai chỗ:

Bước 2a: Trong khi ($a[i] < x$) i++;

Bước 2b: Trong khi ($a[j] > x$) j--;

Bước 2c: Nếu $i < j$ // $a[i] \ge x \ge a[j]$ mà $a[j]$ đứng sau $a[i]$

Hoán vị $a[i]$ và $a[j]$; i ++; j--;

Bước 3: Nếu $i \le j$: Lặp lại bước 2. // chưa xét hết mảng

Nếu $i > j$: Dừng

Giải thuật Quick Sort

Có thể phát biểu giải thuật sắp xếp Quick Sort một cách đệ qui như sau:

Bước 1: Phân hoạch dãy $a_1 \ldots a_r$ thành các dãy con

Dãy con 1: $a_1..a_j \le x$

Dãy con 1: $a_{j+1}..a_{i-1} = x$

Dãy con 1: $a_i..a_r \ge x$

Bước 2: Nếu ($l < j$) // dãy con 1 có nhiều hơn một phần tử

Phân hoạch dãy $a_1..a_j$

Nếu ($i < r$) // dãy con 3 có nhiều hơn một phần tử

Phân hoạch dãy $a_i..a_r$

2. HEAP SORT

Ý tưởng

- Xét dãy gồm N phần tử
- Hiệu chỉnh dãy thành heap. Đảo vị trí của phần tử đầu dãy với phần tử cuối dãy để đưa phần tử lớn nhất về cuối dãy.
- Ta không quan tâm đến phần tử cuối dãy nữa, xem như dãy hiện hành chỉ gồm N-1 phần tử, tính từ 1.
- Lặp lại xử lí trên cho đến khi dãy hiện hành chỉ còn một phần tử.

Giải thuật

Giai đoạn 1: Hiệu chỉnh dãy số ban đầu thành Heap

Giai đoạn 2: Sắp xếp dãy số dựa trên Heap

Bước 1: Đưa phần tử lớn nhất về vị trí đúng ở cuối dãy:

$r = n$; Hoán vị (a_1, a_r);

Bước 2: Loại bỏ phần tử lớn nhất ra khỏi heap: $r = r - 1$;

Hiệu chỉnh phần còn lại của dãy a_1, a_2,…, a_r thành một Heap.

Bước 3: Nếu $r > 1$ (heap còn phần tử): Lặp lại bước 2

Ngược lại: Dừng.

Dựa trên tính chất 3 của Heap, ta có thể thực hiện giai đoạn 1 bằng cách bắt đầu từ heap mặc nhiên $a_{n/2+1}$, $a_{n/2+2}$, …., a_n, lần lượt thêm vào các phần tử $a_{n/2}$, $a_{n/2-1}$, …., a_1 ta sẽ nhận được heap theo mong muốn.

3. MERGE SORT

Ý tưởng

- Gọi k là chiều dài của một dãy con
- Phân chia dãy ban đầu thành các dãy con có độ dài k, các dãy con này được phân phối luân phiên vào 2 dãy phụ.
- Trộn từng cặp dãy con của 2 dãy phụ thành một dãy con của dãy ban đầu, cuối cùng ta sẽ thu được dãy ban đầu có số lượng dãy con giảm phân nửa.
- Tăng độ dài k, lặp lại qui trình trên sau một số bước ta sẽ thu được dãy chỉ gồm 1 dãy con, tức là dãy ban đầu đã được sắp xếp.

Giải thuật

Bước 1: $k = 1$; // k là chiều dài của dãy con trong bước hiện hành

Bước 2:

Tách dãy $a_1, a_2, \ldots, a_n$ thành 2 dãy b, c theo nguyên tắc luân phiên từng nhóm k phần tử:

$b = a_1, \ldots, a_k, a_{2k+1}, \ldots, a_{3k}, \ldots$

$c = a_{k+1}, \ldots, a_{2k+1}, a_{3k+1}, \ldots, a_{4k}, \ldots$

Bước 3: Trộn từng cặp dãy con gồm k phần tử của 2 dãy b, c vào a.

Bước 4: $k = k*2$;

Nếu $k < n$: Lặp lại bước 2.

Ngược lại: Dừng.

4. RADIX SORT

Ý tưởng

Radix Sort dựa trên nguyên tắc phân loại thư của bưu điện. Nó không quan tâm đến việc so sánh giá trị của phần tử và bản thân việc phân loại và trình tự phân loại sẽ tạo ra thứ tự cho các phần tử.

Giải thuật thực hiện như sau:

- Trước tiên, ta giả sử mỗi phần tử a_i trong dãy $a_1, a_2, \ldots, a_n$ là một số nguyên có tối đa m chữ số.

– Phân loại các phần tử lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm…

Giải thuật

Bước 1: // k cho biết chữ số dùng để phân loại hiện hành.

$k = 0$; // $k = 0$: hàng đơn vị; $k = 1$: hàng chục;…

Bước 2: // Tạo các lô chứa các loại phần tử khác nhau.

Khởi tạo 10 lô B_0, B_1,…, B_9 rỗng;

Bước 3: For $i = 1..n$ do

Đặt a_i vào lô B_t với t = chữ số thứ k của a_i ;

Bước 3: Nối B_0, B_1,…, B_9 lại theo đúng trình tự thành a.

Bước 4: $k = k + 1$.

Nếu $k < m$: lặp lại bước 2.

Ngược lại: Dừng

BÀI TẬP

1. Cài đặt các phương pháp sắp xếp trên
2. Tổ chức hàm main theo dạng menu cho phép người dùng lựa chọn phương pháp sắp xếp cần áp dụng.

BÀI THỰC HÀNH SỐ 5

Áp dụng Các phương pháp Sắp xếp và Tìm Kiếm

(4 tiết)

1) Mỗi sinh viên được quản lý ở một hệ thống trường X được bao gồm: họ và tên, năm sinh và địa chỉ email. Để dễ dàng trong việc quản lý, người ta cần một chương trình thực hiện được 2 thao tác cơ bản là sắp xếp và tìm kiếm được mô tả như sau:

   Tìm kiếm

   - Thực hiện tìm kiếm tuyến tính theo tên, năm sinh và địa chỉ email.
   - Thực hiện tìm kiếm nhị phân trên danh sách sau khi đã sắp xếp.

   Sắp xếp

   - Shaker Sort: Sắp xếp tăng dần theo họ và tên.
   - Quick Sort (đệ qui): Sắp xếp tăng dần theo địa chỉ email.
   - Binary Insertion Sort: Sắp xếp tăng dần theo năm sinh.

   Hãy:

   a) Khai báo cấu trúc dữ liệu phù hợp.
   b) Xác định các hàm/thủ tục cần thiết của chương trình.
   c) Viết chương trình thực hiện các hoạt động quản lý sinh viên thỏa mãn các yêu cầu của trường X (như mô tả bên dưới).

2) Viết chương trình thực hiện sắp xếp một dãy gồm N ($N \leq 100$) **số nguyên** với các thuật toán sắp xếp đã học: *Selection Sort*, *Quick sort*, *Merge sort, Shell sort* và *Heap sort*. Trong đó, yêu cầu cụ thể như sau:

   a) Selection Sort: Số âm tăng dần về cuối dãy, số không âm giảm dần về đầu dãy.
   b) Quick Sort: Sắp tăng dần theo trị tuyệt đối.
   c) Merge Sort: Trộn theo cả 2 cách (tự nhiên và trực tiếp). Sắp tăng dần.
   d) Shell Sort: Sắp tăng dần với bước chạy tùy chọn (nên lấy theo sách).
   e) Heap Sort: Sắp giảm dần.

Chương 3: CẤU TRÚC DANH SÁCH LIÊN KẾT

BÀI THỰC HÀNH SỐ 6

Danh sách liên kết đơn

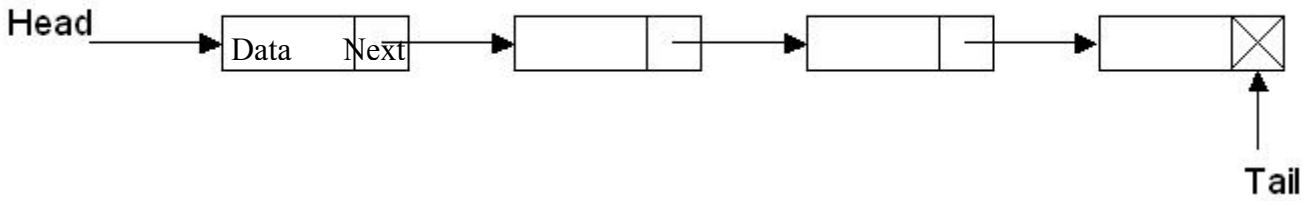
(8 tiết)

Mục tiêu

- Làm việc với kiểu dữ liệu con trỏ.
- Cài đặt danh sách liên kết đơn không có nút câm.

Nội dung lý thuyết

1. CẤU TRÚC DỮ LIỆU CỦA DANH SÁCH LIÊN KẾT



Khai báo danh sách liên kết số nguyên

```
typedef    int   ElementType;
typedef struct node
        {
                ElementType  Data; //Trường dữ liệu
                struct node *Next; // Trường lưu địa chỉ nút kế tiếp
        }NodeType;
typedef  NodeType *NodePointer; //Định nghĩa kiểu con trỏ nút
//Định nghĩa kiểu danh sách liên kết
typedef  struct
        {
                NodePointer Head, Tail;
        } LL;
//Khai báo danh sách liên kết
LL  List
```

2. CÁC THAO TÁC CƠ BẢN TRÊN DANH SÁCH LIÊN KẾT

a) Tạo một nút của danh sách.
b) Thêm một phần tử vào danh sách (chèn đầu, chèn cuối, chèn vào sau một nút).
c) Duyệt danh sách.

d) Tìm một phần tử trên danh sách.
e) Xóa phần tử khỏi danh sách.
f) Sắp xếp trên danh sách.

3. BÀI TẬP

Viết chương trình thực hiện:

1. Tạo danh sách liên kết với dữ liệu được nhập vào từ bàn phím (các phần tử được chèn vào cuối danh sách).
2. Tạo bản sao của một DSLK cho trước.
3. Nối hai DSLK cho trước.
4. Tính số lượng các nút dữ liệu.
5. Tìm nút dữ liệu đầu tiên trong DSLK thỏa một tính chất nào đó, chẳng hạn:
   - nút thứ k,
   - hoặc có trường dữ liệu trùng với một giá trị cùng kiểu K cho trước.

   Nếu có thì trả về con trỏ chỉ đến nút đứng trước nút tìm thấy.
6. Xóa một (hay mọi) nút dữ liệu trong DSLK thỏa một tính chất nào đó, ví dụ:
   - nút thứ k,
   - hoặc có trường dữ liệu trùng với một giá trị cùng kiểu K cho trước.
7. Thêm một nút L vào sau một (hay mọi) nút dữ liệu trong DSLK thỏa một tính chất nào đó, chẳng hạn:
   - nút thứ k,
   - hoặc có trường dữ liệu trùng với một giá trị cùng kiểu K cho trước.
8. Đảo ngược DSLK nói trên theo hai cách:
   - tạo DSLK mới
   - sửa lại chiều con trỏ trong DSLK ban đầu.
9. Gọi M là con trỏ chỉ tới một nút đã có trong DSLK trên và P là con trỏ chỉ tới một DSLK khác cùng loại. Hãy chèn DSLK P này vào sau nút trỏ bởi M.
10. Tách thành 2 DSLK mà DS sau được trỏ bởi M (giả thiết như câu h).
11. So sánh 2 DSLK (có trường dữ liệu của các nút liên tiếp tương ứng bằng nhau hay không ?).
12. Sắp xếp danh sách.
13. Hãy viết thuật toán và chương trình để trộn hai DSLK tăng A, B cho trước thành một DSLK C cũng tăng theo hai cách:
    a. C là DSLK mới (cấp phát bộ nhớ mới cho mọi nút của C) và bảo toàn hai DSLK cũ A, B;
    b. C là DSLK mới do A, B hợp thành (do đổi chỗ vị trí các con trỏ sẵn có trên A, B). Khi đó cấu trúc hai DSLK A, B có thể bị thay đổi.

BÀI THỰC HÀNH SỐ 7
Áp dụng Danh sách liên kết
(8 tiết)

Mục tiêu

- Áp dụng cấu trúc danh sách vào một bài toán cụ thể.

BÀI TẬP

1) Một cửa hàng tạp hoá cần lưu lại thông tin các mặt hàng gồm: *Tên mặt hàng* và *số lượng tồn* (số lượng tồn ≥ 0). Các thao tác nghiệp vụ về việc xuất nhập hàng bao gồm các thao tác đơn giản sau:

- Nhập hàng: thông tin nhập hàng bao gồm: tên mặt hàng, số lượng nhập kho tương ứng. Nếu mặt hàng đã có trong danh mục mặt hàng thì cộng dồn số lượng nhập vào số lượng tồn của mặt hàng đó, ngược lại thì thêm thông tin mặt hàng mới vào danh mục.
- Xuất hàng. Thông tin xuất hàng bao gồm: tên mặt hàng, số lượng xuất kho tương ứng. Khi xuất mặt hàng nào thì số lượng tồn của mặt hàng đó sẽ được giảm tương ứng với số lượng yêu cầu xuất.
- Loại bỏ hoàn toàn một mặt hàng. Thông tin cần cung cấp bao gồm: tên mặt hàng cần loại bỏ. (Mặt hàng X sau khi loại bỏ sẽ không còn được lưu trữ.)
- Sắp xếp danh mục mặt hàng theo tên mặt hàng tăng dần.
- Xem danh mục các mặt hàng.

Giả sử danh mục hàng hóa trên được lưu bằng danh sách liên kết, hãy:

a) Khai báo cấu trúc dữ liệu phù hợp cho danh sách liên kết.
b) Xây dựng các hàm cần thiết của chương trình.
c) Viết chương trình thực hiện các nghiệp vụ quản lý hàng hóa trên.

2) Cho n số nguyên dương (n>1). Hãy xây dựng một danh sách liên kết chứa n số nguyên dương trên và thực hiện các thao tác sau:

a) Xuất ra các số cực đại địa phương và cực tiểu địa phương cùng với vị trí của nó trong danh sách

   Ví dụ : Nhập chuỗi số nguyên từ tập tin : 1, 2, 8, 9, 75, 65, 81, 36, 47, 61, 55, 63
   Các cực đại địa phương : 75, 81 , 61, 63
   Các cực tiểu địa phương : 1, 65, 36, 55

b) Xóa đi tất cả các phần tử trùng nhau (giữ lại một phần tử trong số đó, phần tử nào cũng được).

c) Hãy tìm tất cả các số nguyên tố có trong xâu và thay bằng số nguyên tố liền sau nó (ví dụ: 3 thành 5, 5 thành 7, 13 thành 17)
d) Sắp xếp lại xâu trên tăng dần theo các thuật toán sau:
   - Selection Sort
   - Insertion Sort
e) Thực hiện lần lượt thêm vào danh sách m phần tử sao cho phần tử được thêm vào vị trí sao cho nhỏ hơn hoặc bằng phần tử trước nó và lớn hơn hoặc bằng phần tử sau nó. Nếu nó không tìm được vị trí như vậy sẽ được thêm vào cuối danh sách. Nếu có nhiều hơn 1 vị trí thích hợp thì chọn vị trí đầu tiên.

3) Xây dựng cấu trúc dữ liệu biểu diễn đa thức. Viết các thao tác trên đa thức: tính giá trị, tính đạo hàm, cộng trừ đa thức.

$$P(x) = c_k x^k + c_{k-1} x^{k-1} + \ldots + c_1 x + c_0$$

4) Tính toán số lớn: Viết cấu trúc số nguyên lớn (> 50 chữ số) dùng danh sách liên kết. Thực hiện các thao tác cơ bản tính toán số nguyên: cộng, trừ, nhân

BÀI THỰC HÀNH SỐ 8

Các Thao Tác Trên Stack – Queue

(8 tiết)

Mục tiêu

- Làm việc với Stack và Queue
- Cài đặt Stack và Queue

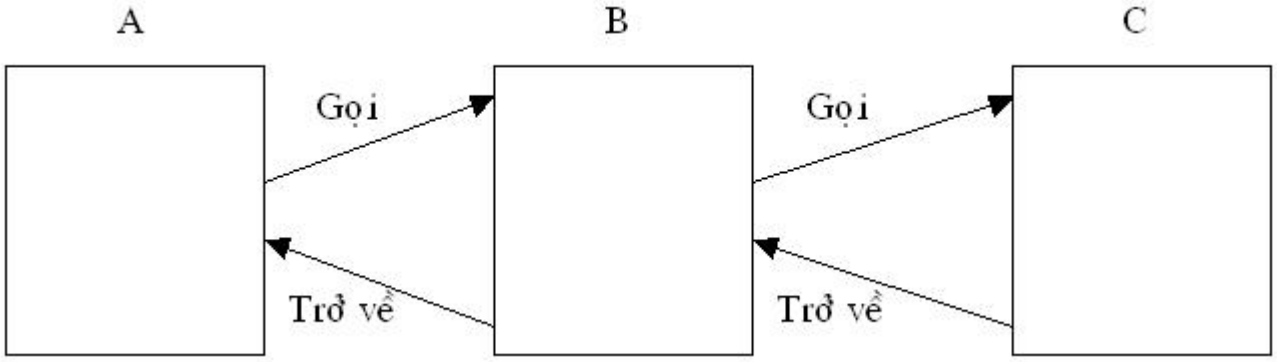
Nội dung lý thuyết

1. NGĂN XẾP (STACK)

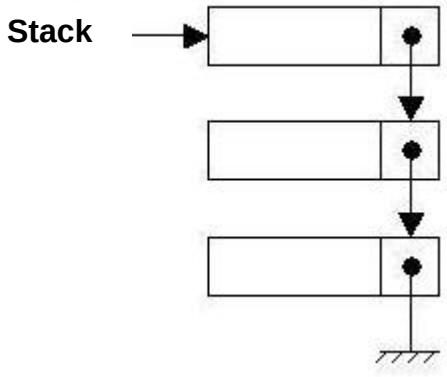
Ngăn xếp là một danh sách mà 2 phép thêm và loại bỏ đều được thực hiện trên một đầu.

Như vậy, phần tử nào thêm vào sau sẽ bị loại bỏ trước, nên ngăn xếp còn được gọi là danh sách LIFO (Last In First Out list) hoặc Pushdown list.

Ngăn xếp có nhiều ứng dụng. Ví dụ: chương trình A gọi chương trình B, chương trình B gọi chương trình C. Khi chương trình C được thực hiện xong thì sự điều khiển chương trình sẽ trở về thực hiện chương trình B, rồi khi chương trình B được thực hiện xong thì sự điều khiển chương trình sẽ trở về thực hiện chương trình A. Như vậy chương trình B được gọi sau sẽ được trở về thực hiện trước chương trình A. Đó là nhờ điểm nhập (entry point) trở về của các chương trình được chứa trong ngăn xếp.



Ngăn xếp có thể được tổ chức theo danh sách liên kết. Vì phép thêm vào và phép loại bỏ chỉ được thực hiện ở cùng một đầu nên ta chỉ cần một con trỏ giữ đỉnh của stack.



Cấu trúc dữ liệu của Stack

```
typedef    ....   ElementType;                    // Kiểu dữ  liệu của nút
typedef struct node
                {
                        ElementType  Data;
                        struct node *Next;
                } NodeType;
typedef  NodeType  *NodePointer;
NodePointer  Stack;
```

Các phép thao tác cơ bản

a) Thao tác **Push** đẩy một mục dữ liệu x vào đỉnh ngăn xếp
b) Thao tác **Pop** lấy ra một phần tử ở đỉnh ngăn xếp
c) Thao tác **Top** xem một phần tử ở đỉnh ngăn xếp

2. HÀNG ĐỢI (QUEUE)

Hàng đợi là một danh sách mà phép thêm vào được thực hiện ở đầu này và loại bỏ được thực hiện ở đầu kia.

Như vậy, phần tử nào vào trước sẽ được loại bỏ trước, phần tử nào vào sau sẽ được loại bỏ sau, nên hàng đợi còn được gọi là danh sách FIFO (First In First Out list).

Cấu trúc dữ liệu

```
typedef    int   ElementType;
typedef struct node
        {
                ElementType  Data;
                struct node *Next;
        }NodeType;
typedef  NodeType *NodePointer; //Định nghĩa kiểu con trỏ nút
typedef  struct
        {
                NodePointer Head, Tail;
        } LL;
```

LL Queue;

Các thao tác cơ bản

a) **EnQueue(O)**: thêm một đối tượng O vào đuôi hàng đợi;
b) **DeQueue()**: lấy ra một đối tượng ở đầu hàng đợi và trả về trị của nó, nếu hàng đợi rỗng sẽ gặp lỗi;
c) **EmptyQueue()**: kiểm tra xem hàng đợi có rỗng hay không;
d) **Front()**: Trả về trị của phần tử ở đầu hàng đợi mà không loại nó khỏi hàng đợi, nếu hàng đợi rỗng sẽ gặp lỗi.

BÀI TẬP

1) Cho 1 chuỗi mô tả các thao tác trên stack/queue gồm các ký tự chữ cái và dấu * với quy tắc như sau:

- Chữ cái tượng trưng cho thao tác thêm chữ cái tương ứng vào stack/queue.
- Dấu * tượng trưng cho thao tác lấy nội dung một phần tử ra khỏi stack/queue.

Thực hiện chuỗi thao tác này trên stack/queue theo thứ tự từ trái sang phải.Sau khi hoàn tất, tiến hành lấy lần lượt tất cả các phần tử còn lại trong stack/queue, hãy cho biết nội dung lấy được.

Ví dụ

AAA**BCC*D

Kết quả khi thao tác trên Stack là DCBA

Kết quả khi thao tác trên Queue là BCCD

2) Viết chương trình dùng ngăn xếp thực hiện chuyển đổi và biểu diễn số nguyên dương N ở dạng cơ số b (b = 2, 8,16).

3) Viết chương trình tính giá trị biểu thức bằng dãy Balan ngược Thực hiện qua 2 bước

***Bước 1*:** Chuyển biểu thức từ dạng trung tố (dạng gốc do người dùng nhập vào) sang dạng hậu tố.

***Bước 2*:** Tính giá trị của biểu thức dạng hậu tố, ta sẽ được kết quả.

Thuật toán chuyển từ dạng trung tố sang hậu tố

- Khởi động stack rỗng (Stack chứa toán tử)
- While (không có lỗi và chưa hết biểu thức)
    - Đọc Token (Token = hằng/biến/toán tử số học /ngoặc trái/ngoặc phải).
    - Nếu Token là
        - Ngoặc trái (: Push vào stack.

- Ngoặc phải): Pop và hiển thị các phần tử của stack đến khi gặp ngoặc trái (pop ngoặc trái nhưng không hiển thị ngoặc trái).
- Toán tử: Nếu stack rỗng hay Token được ưu tiên hơn phần tử ở đỉnh stack thì Push vào Stack .Ngược lại (ưu tiên bằng hoặc ít ưu tiên hơn) pop và hiển thị 1 phần tử ở đỉnh stack .Lặp lại việc so sánh Token với 1 phần tử ở đỉnh stack.
- Toán hạng : hiển thị nó.

❖ Khi hết biểu thức trung tố Pop và hiển thị toàn bộ stack còn lại.

Thuật toán tính giá trị biểu thức hậu tố:

❖ Khởi động stack rỗng.

❖ Lặp lại các bước sau đến khi hết biểu thức:

- Đọc Token (Hằng, biến, toán tử)
- Nếu Token là :
  - Toán hạng: Push vào stack
  - Toán tử:
    - Pop 2 giá trị
    - Áp dụng toán tử cho 2 giá trị lấy ra.
    - Push kết quả vào stack.

Lặp đến hết biểu thức ,giá trị ở đỉnh stack là giá trị của biểu thức.

Độ ưu tiên của các toán tử

‘(’ < {‘+’,’-‘} < {‘*’,’/’}

VD:

(2+3)*5 → 2 3 + 5 * : Sau khi chuyển từ trung tố sang hậu tố.

4*12/3+1 → 4 12 * 3 / 1: Sau khi chuyển từ trung tố sang hậu tố.

5*(2+3) →5 2 3 + * : Sau khi chuyển từ trung tố sang hậu tố.

Chương 4: CẤU TRÚC CÂY

BÀI THỰC HÀNH SỐ 9

Cây Nhị Phân, Cây Nhị Phân Tìm kiếm

(8 tiết)

Mục tiêu

Cài đặt cây nhị phân, cây tìm kiếm nhị phân và các thao tác.

Nội dung lý thuyết

1. CẤU TRÚC DỮ LIỆU CỦA CÂY NHỊ PHÂN

```
typedef    .....  ElementType;  /* Kiểu mục dữ liệu của nút */
typedef struct TN
        {
                ElementType Data; //Để đơn giản, ta xem Data là trường khóa của dữ liệu
                  struct TN * LChild, *RChild;
        } TreeNode;
typedef   TreeNode  *TreePointer;
```

2. CÁC THAO TÁC CƠ BẢN

a) Tạo một nút của cây
b) Chèn một phần tử vào cây
c) Duyệt cây: NLR, LNR, LRN
d) Tìm một phần tử có trên cây
e) Xóa phần tử của cây.

BÀI TẬP

1) *Viết một chương trình có các tác dụng sau:*

a. Nhập từ bàn phím các số nguyên vào một cây nhị phân tìmkiếm (BST) mà nút gốc được trỏ bởi con trỏ Root.
b. Xuất các phần tử trên cây BST trên theo thứ tự: đầu, giữa, cuối.
c. Tìm và xóa (nếu có thể) phần tử trên cây Root có dữ liệu trùng với một mục dữ liệu Item cho trước được nhập từ bàn phím.
d. Sắp xếp n mục dữ liệu (được cài đặt bằng mảng hay DSLK) bằng phương pháp cây nhị phân tìm kiếm BSTSort.

Yêu cầu: viết các thao tác trên bằng 2 phương pháp: đệ quy và lặp (*).

2) Cho cây nhị phân T. Viết chương trình chứa các hàm có tác dụng xác định:

a. *Tổng số nút* của cây.
b. Số nút của cây ở mức k
c. *Chiều cao* của cây.
d. *Tổng giá trị trường dữ liệu (số!)* trên các nút của cây.

e. *Kiểm tra xem nó có phải là một cây nhị phân chặt* (là cây nhị phân mà mỗi nút khác nút lá đều có đúng 2 con) hay không?
f. *Kiểm tra xem T có phải là cây cân bằng hoàn toàn* hay không ?
g. Số nút có đúng 2 con khác rỗng.
h. Số nút có đúng 1 con khác rỗng.
i. Số nút có khóa nhỏ hơn x trên cây nhị phân hoặc cây BST.
j. Số nút có khóa lớn hơn x trên cây nhị phân hoặc cây BST.
k. Số nút có khóa nhỏ hơn x và lớn hơn y ($y \leq x$) trên cây nhị phân hoặc cây BST
l. Duyệt cây theo chiều rộng.
m. Duyệt cây theo chiều sâu.
n. Độ lệch lớn nhất của các nút trên cây (độ lệch của một nút là trị tuyệt đối của hiệu số giữa chiều cao của cây con phải và cây con trái của nó).
o. Đảo nhánh trái và phải của mọi nút trên một cây nhị phân.

BÀI THỰC HÀNH SỐ 10

Các thao tác trên cây nhị phân tìm kiếm cân bằng

(4 tiết)

Mục tiêu

Cài đặt cây nhị phân, cây nhị phân tìm kiếm cân bằng.

BÀI TẬP

Cài đặt các thao tác trên cây AVL. Cho phép người dùng lựa chọn thực hiện các thao tác sau:

1. Kiểm tra một node có thuộc cây hay không?
2. Insert một node vào cây.
3. Delete một node trong cây. Nếu node có hai cây con thì cho phép chọn phần tử thế chỗ thuộc:
    a. Cây con trái.
    b. Cây con phải.
4. Duyệt cây để in ra các node, theo thứ tự:
    a. NLR.
    b. LNR.
    c. LRN.

Để tiện dụng, yêu cầu 2, 3 cần thực hiện vòng while cho đến khi người dùng không muốn insert hay delete nữa.

CÁC BÀI KIỂM TRA

(4 tiết)

Bài kiểm tra số 1:

- Thời gian: 2 tiết (90 phút)
- Hình thức: Thực hành trên máy
- Nội dung:
  - Kiểu dữ liệu có cấu trúc
  - Áp dụng các phương pháp tìm kiếm và sắp xếp

Bài kiểm tra số 2:

- Thời gian: 2 tiết (90 phút)
- Hình thức: Thực hành trên máy
- Nội dung:
  - Các kiểu danh sách và ứng dụng

TÀI LIỆU THAM KHẢO

[1] Trương Chí Tín, Cấu trúc dữ liệu và Giải thuật 1 (Giáo trình tóm tắt), Đại học Đà Lạt, 2008

[2] A.V. AHO , J.E. HOPCROFT , J.D. ULMANN: Data structures and algorithms. Addition Wesley - 1983.

[3] DONALD KNUTH: The Art of Programming. (vol.1: Fundamental Algorithms, vol. 3: Sorting and Searching). Addition Wesley Puplishing Company - 1973.

[4] ĐINH MẠNH TƯỜNG: Cấu trúc dữ liệu và thuật toán. NXB KHKT - 2001.

[5] ĐỖ XUÂN LÔI: Cấu trúc dữ liệu và thuật toán. NXB KHKT - 1995.

[6] LARRY N. HOFF, SANFORD LEESTMA: Lập trình nâng cao bằng Pascal với các cấu trúc dữ liệu. Bản dịch của Lê Minh Trung. Công ty Scitec - 1991.

[7] NGUYỄN TRUNG TRỰC: Cấu trúc dữ liệu. Trung tâm điện toán, trường ĐH Bách khoa TP. HCM – 1992.

[8] NIKLAUS WIRTH: Cấu trúc dữ liệu + Giải thuật = Chươngtrình (Nguyễn Quốc Cường dịch). NXB ĐH và THCN – 1991

[9] TRẦN HẠNH NHI & DƯƠNG ANH ĐỨC: Nhập môn cấu trúc dữ liệu và thuật toán. Khoa Công nghệ thông tin, ĐH KHTN TP HCM – 2000.