BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

🖎🕮✍

****

**BÁO CÁO**

**Môn: Thực tập cơ sở công nghệ thông tin**

**Đề tài: Sự tối ưu giữa các thuật toán sắp xếp**

***GVHD:*** Nguyễn Mạnh Cương

***Người thực hiện:* Nguyễn Bùi Nhật Hưng**

***Học phần lớp:*** 63.CNTT-5

**Khánh hòa,2024**

**Mục lục**

[CHƯƠNG I. TIÊU ĐỀ VÀ MỤC TIÊU NGHIÊN CỨU 4](#_Toc155683352)

[1.1 Tiêu đề 4](#_Toc155683353)

[1.2 Mục tiêu nghiên cứu 4](#_Toc155683354)

[1.2.1 Mục tiêu 4](#_Toc155683355)

[1.2.2 Cách thức 4](#_Toc155683356)

[CHƯƠNG II: TỔNG QUAN 4](#_Toc155683357)

[2.1 Giải thuật sắp xếp 4](#_Toc155683358)

[2.1.1 Sắp xếp 4](#_Toc155683359)

[2.1.2 Sắp xếp chọn (Selection sort) 5](#_Toc155683360)

[2.1.2 Sắp xếp Chèn (Insertion Sort) 5](#_Toc155683361)

[2.1.3 Sắp xếp nổi bọt (Bubble sort) 6](#_Toc155683362)

[2.1.4 Sắp xếp Nhanh (Quick Sort) 7](#_Toc155683363)

[2.1.5 Sắp xếp vun đống (Heap sort) 8](#_Toc155683364)

[2.1.6 Sắp xếp trộn (Merge sort) 10](#_Toc155683365)

[2.2 Vai trò của tối ưu giải thuật sắp xếp 13](#_Toc155683366)

[CHƯƠNG III: PHẠM VI NGHIÊN CỨU 14](#_Toc155683367)

[3.1 Phạm vi nghiên cứu 14](#_Toc155683368)

[3.2 Câu hỏi cụ thể muốn trả lời trong quá trình nghiên cứu 14](#_Toc155683369)

[CHƯƠNG IV: PHƯƠNG PHÁP NGHIÊN CỨU 14](#_Toc155683370)

[4.1 Thu thập dữ liệu 14](#_Toc155683371)

[4.2 Tham số đánh giá 16](#_Toc155683372)

[4.3 Tiến hành thử nghiệm 16](#_Toc155683373)

[4.4 Phân tích kết quả 16](#_Toc155683374)

[4.2 Đánh giá kết quả 16](#_Toc155683375)

[CHƯƠNG V: KẾT QUẢ VÀ PHÂN TÍCH 17](#_Toc155683376)

[5.1 Kết quả 17](#_Toc155683377)

[5.1.1 Bộ số liệu 10.000 17](#_Toc155683378)

[5.1.2 Bộ số liệu 100.000 17](#_Toc155683379)

[5.2 Đánh giá và kết luận 18](#_Toc155683380)

[C. TÀI LIỆU THAM KHẢO: 19](#_Toc155683381)

# CHƯƠNG I. TIÊU ĐỀ VÀ MỤC TIÊU NGHIÊN CỨU

### Tiêu đề

"Sự tối ưu giữa các giải thuật sắp xếp: Là một bài nghiên cứu, so sánh hiệu suất sắp xếp giữa giải thuật Selection sort, Insertion sort, Bubble sort, Quick sort, Heap sort và Merge sort".

### 1.2 Mục tiêu nghiên cứu

#### 1.2.1 Mục tiêu

Mục tiêu nghiên cứu của báo cáo này là so sánh và đánh giá hiệu suất giữa giải thuật sắp xếp phổ biến là Selection sort, Insertion sort, Bubble sort, Quick sort, Heap sort và Merge sort. Nhằm để xác định được giải thuật nào có hiệu suất tốt nhất trong việc sắp xếp dữ liệu trong nhiều trường hợp khác nhau. Hơn hết là tập trung vào các yếu tố như thời gian hoàn thành của giải thuật đối với các tệp dữ liệu lớn, số lần hoán đổi và số lần so sánh trong quá trình sắp xếp.

#### 1.2.2 Cách thức

Thông qua việc tạo ngẫu nhiên các tệp dữ liệu và đánh giá hiệu suất của các giải thuật sắp xếp trong việc xử lý các tệp dữ liệu này. Kết quả của bài nghiên cứu sẽ giúp người đọc hiểu rõ hơn về ưu và nhược điểm của từng giải thuật. Cung cấp thông tin hữu ích để lựa chọn được giải thuật phù hợp trong các trường hợp ứng dụng vào thực tế.

## CHƯƠNG II: TỔNG QUAN

### 2.1 Giải thuật sắp xếp

#### 2.1.1 Sắp xếp

Sắp xếp là một phần quan trọng trong lĩnh vực khoa học máy tính và giải thuật. Sắp xếp dữ liệu đảm bảo rằng các phần tử trong tập dữ liệu được sắp xếp một cách có trình tự, qua đó giúp tối ưu hóa việc truy xuất và xử lý dữ liệu.

Các giải thuật sắp xếp đóng vai trò cực kỳ quan trọng trong quá trình này, việc tối ưu hóa các giải thuật sắp xếp trở thành một nhiệm vụ quan trọng trong lĩnh vực khoa học máy tính. Mục tiêu của tối ưu hóa giải thuật sắp xếp là cải thiện hiệu suất của các giải thuật, bằng cách tối ưu tốc độ thực thi, nhưng đồng thời cũng phải giảm thiểu sử dụng bộ nhớ, cũng như tối ưu hoá số lần so sánh và hoán đổi.

Có nhiều giải thuật sắp xếp đã được nghiên cứu để đáp ứng các yêu cầu khác nhau. Một số giải thuật sắp xếp phổ biến bao gồm Sắp xếp chọn (Selection sort), Sắp xếp Chèn (Insertion Sort), Sắp xếp nổi bọt (Bubble sort), Sắp xếp Nhanh (Quick Sort), Sắp xếp vun đống (Heap sort), Sắp xếp trộn (Merge sort).

#### 2.1.2 Sắp xếp chọn (Selection sort)

Lựa chọn phần tử có giá trị nhỏ nhất, đổi chỗ cho phần tử đầu tiên. Tiếp theo, lựa chọn phần tử có giá trị nhỏ thứ nhì, đổi chỗ cho phần tử thứ 2. Quá trình tiếp tục cho tới khi toàn bộ dãy được sắp.

Độ phức tạp trung bình của thuật toán là : O(N^2).

Hàm mã nguồn tương ứng để thực hiện sắp xếp:

void selection\_sort()

{

int i, j, k;

for (i = 0; i< N; i++)

{

k = i;

for (j = i+1; j < N; j++)

{

if (a[j] < a[k]) k = j;

}

a[i] = a[i] + a[k];

a[k] = a[i] - a[k];

a[i] = a[i] - a[k];

}

}

#### 2.1.2 Sắp xếp Chèn (Insertion Sort)

Phần đầu là các phần tử đã được sắp. Từ phần tử tiếp theo, chèn nó vào vị trí thích hợp tại nửa đã sắp sao cho nó vẫn được sắp. Để chèn phần tử vào nửa đã sắp, chỉ cần dịch chuyển các phần tử lớn hơn nó sang phải 1 vị trí và đưa phần tử này vào vị trí trống trong dãy.

Độ phức tạp trung bình của thuật toán là : O(N^2).

Hàm mã nguồn tương ứng thực hiện sắp xếp:

void insertion\_sort()

{

int i, j, k, temp;

for (i = 1; i< N; i++)

{

temp = a[i];

j=i-1;

while ((a[j] > temp)&&(j>=0))

{

a[j+1]=a[j];

j--;

}

a[j+1]=temp;

}

}

#### 2.1.3 Sắp xếp nổi bọt (Bubble sort)

Duyệt nhiều lần từ cuối lên đầu dãy, tiến hành đổi chỗ 2 phần tử liên tiếp nếu chúng ngược thứ tự. Đến một bước nào đó, khi không có phép đổi chỗ nào xảy ra thì toàn bộ dãy đã được sắp.

Độ phức tạp trung bình của thuật toán là : O(N^2).

Hàm mã nguồn tương ứng thực hiện sắp xếp:

void bubble\_sort()

{

int i, j, temp, no\_exchange;

i = 1;

do

{

no\_exchange = 0;

for (j=n-1; j >= i; j--)

{

if (a[j-1] > a[j])

{

temp=a[j-1];

a[j-1]=a[j];

a[j]=temp;

no\_exchange = 1;

}

}

i++;

}

while (no\_exchange || (i == n-1));

}

#### 2.1.4 Sắp xếp Nhanh (Quick Sort)

Đầu tiên chọn ngẫu nhiên 1 phần tử nào đó của dãy làm khóa. Trong bước tiếp theo, các phần tử nhỏ hơn khoá phải được xếp vào phía trước khoá và các phần tử lớn hơn đƣợc xếp vào phía sau khoá. Để có được sự phân loại này, các phần tử sẽ được so sánh với khoá và hoán đổi vị trí cho nhau hoặc cho khoá nếu nó lớn hơn khóa mà lại nằm trước hoặc nhỏ hơn khoá mà lại nằm sau. Khi lượt hoán đổi đầu tiên thực hiện xong thì dãy được chia thành 2 đoạn: 1 đoạn bao gồm các phần tử nhỏ hơn khoá, đoạn còn lại bao gồm các phần tử lớn hơn khoá.

Độ phức tạp thuật toán: Trong trường hợp xấu nhất là O(N^2) , trong trường hợp tốt nhất và trung bình là O(NlogN).

Hàm mã nguồn tương ứng thực hiện sắp xếp:

void quick(int left, int right)

{

int i,j,x,y;

i=left; j=right; x= a[left];

do

{

while(a[i]x && j>left) j--;

if(i<=j)

{

y=a[i];

a[i]=a[j]; a[j]=y;

i++; j--;

}

}while (i<=j);

if (left<j) quick(left,j);

if (i<right) quick(i,right);

}

void quick\_sort()

{

quick(0, n-1);

}

#### 2.1.5 Sắp xếp vun đống (Heap sort)

Việc thực hiện giải thuật này đƣợc chia làm 2 giai đoạn. Giai đoạn 1: Tạo heap từ dãy ban đầu. Theo định nghĩa của heap thì nút cha bao giờ cũng lớn hơn các nút con => nút gốc của heap bao giờ cũng là phần tử lớn nhất. Giai đoạn 2 : Sắp dãy dựa trên heap tạo được. Do nút gốc là nút lớn nhất nên nó sẽ được chuyển về vị trí cuối cùng của dãy và phần tử cuối cùng sẽ được thay vào gốc của heap. Khi đó ta có 1 cây mới, không phải heap, với số nút đƣợc bớt đi 1. Lại chuyển cây này về heap và lặp lại quá trình cho tới khi heap chỉ còn 1 nút. Đó chính là phần tử bé nhất của dãy và được đặt lên đầu. Với heap ban đầu chỉ có 1 phần tử là phần tử đầu tiên của dãy, ta lần lượt lấy các phần tử tiếp theo của dãy chèn vào heap sẽ tạo đƣợc 1 heap gồm toàn bộ n phần tử.

Độ phức tạp của giải thuật: O(NlogN) ( Bao gồm trường hợp xấu nhất )

Hàm mã nguồn tương ứng thực hiện sắp xếp:

void upheap(int m)

{

int x; ν x=a[m];

while ((a[(m-1)/2]<=x) && (m>0))

{

a[m]=a[(m-1)/2];

m=(m-1)/2;

}

a[m]=x;

}

void insert\_heap(int x)

{

a[m]=x;

upheap(m);

m++;

}

void downheap(int k)

{

int j, x; x=a[k];

while (k<=(m-2)/2)

{

j=2\*k+1;

if (j=a[j]) break;

a[k]=a[j];

k=j;

}

a[k]=x;

}

int remove\_node()

{

int temp;

temp=a[0];

a[0]=a[m];

m--;

downheap(0);

return temp;

}

void heap\_sort()

{

int i; ν m=0;

for (i=0; i<=n-1; i++) insert\_heap(a[i]);

m=n-1;

for (i=n-1; i>=0; i--) a[i]=remove\_node();

}

#### 2.1.6 Sắp xếp trộn (Merge sort)

Mảng ban đầu được chia đôi thành hai nửa mảng gần bằng nhau. Quá trình chia tiếp tục cho đến khi mỗi mảng chỉ chứa một phần tử. Đối với mỗi nửa mảng, tiếp tục sắp xếp đệ qui bằng cách tiếp tục chia đôi và sắp xếp từng nửa mảng con. Quá trình đệ qui này tiếp tục cho đến khi mảng chỉ chứa một phần tử.Sau khi các nửa mảng con được sắp xếp, ta thực hiện phép trộn để tạo ra mảng đã được sắp xếp. Trong quá trình trộn, ta so sánh các phần tử đầu tiên của hai nửa mảng con và chọn phần tử nhỏ nhất (hoặc lớn nhất) để đặt vào mảng kết quả. Sau đó, chỉ mục của phần tử đã chọn được tăng lên và quá trình này tiếp tục cho đến khi toàn bộ mảng đã được trộn. Quá trình trên được lặp lại cho đến khi mảng ban đầu được sắp xếp hoàn chỉnh.

Độ phức tạp của giải thuật là : O(NlogN)

Hàm mã nguồn tương ứng thực hiện sắp xếp:

void merge(int al, int am, int ar)

{

int i, j, k;

int n1 = am - al + 1;

int n2 = ar - am;

//tao mang tam thoi luu tru cac phan tu

int L[n1], R[n2];

//sao chep du lieu vao cac mang tam thoi

for (i = 0; i < n1; i++)

{

L[i] = td[al + i];

}

for (j = 0; j < n2; j++)

{

R[j] = td[am + 1 + j];

}

//tron cac mang tam thoi vao mang goc td[]

i = 0;

j = 0;

k = al;

while (i < n1 && j < n2)

{

amount\_compare++;

if (L[i] <= R[j])

{

td[k] = L[i];

i++;

}

else

{

td[k] = R[j];

j++;

}

k++;

}

//sao chep cac phan tu con lai cua mang L[] vao mang goc td[]

while (i < n1)

{

td[k] = L[i];

i++;

k++;

}

//sao chep cac phan tu con lai cua mang R[] vao mang td[]

while (j < n2)

{

td[k] = R[j];

j++;

k++;

}

}

void merge\_sort(int left, int right)

{

if (left < right)

{

int middle = left + (right - left) / 2;

//goi de quy de sap xep nua dau va nua sau

merge\_sort(left, middle);

merge\_sort(middle + 1, right);

//hop nhat 2 mang da sap xep

merge(left, middle, right);

}

}

### 2.2 Vai trò của tối ưu giải thuật sắp xếp

Tối ưu hóa các giải thuật sắp xếp đóng vai trò quan trọng và cần thiết trong lĩnh vực khoa học máy tính và các ứng dụng thực tế. Việc tối ưu hóa này mang lại nhiều lợi ích đáng kể.

Trước tiên, việc tối ưu hóa các giải thuật sắp xếp giúp tăng hiệu suất của các ứng dụng. Khi giảm thiểu thời gian thực thi, số lần so sánh và số lần hoán đổi, chúng ta có thể cải thiện tốc độ và hiệu suất của các ứng dụng sử dụng giải thuật sắp xếp. Điều này đặc biệt quan trọng trong các ứng dụng mà yêu cầu xử lý dữ liệu lớn hoặc thời gian thực.

Thứ hai, tối ưu hóa giải thuật sắp xếp cũng đóng góp vào việc tiết kiệm tài nguyên hệ thống. Các giải thuật sắp xếp hiệu quả có thể giảm bớt việc sử dụng bộ nhớ và CPU. Điều này giúp tối ưu hóa việc sử dụng tài nguyên hệ thống và tăng khả năng mở rộng của các ứng dụng.

Thêm vào đó, việc nghiên cứu và tối ưu hóa các giải thuật sắp xếp cũng mang lại những đóng góp quan trọng cho lĩnh vực khoa học máy tính. Nó giúp nâng cao kiến thức về các thuật toán sắp xếp và cung cấp cơ sở cho việc phát triển các thuật toán mới và cải tiến. Việc tối ưu hóa giải thuật sắp xếp cũng đóng vai trò quan trọng trong việc nghiên cứu và phân tích hiệu suất của các thuật toán sắp xếp trong các trường hợp cụ thể.

## CHƯƠNG III: PHẠM VI NGHIÊN CỨU

### 3.1 Phạm vi nghiên cứu

Mô tả cách thức hoạt động của từng giải thuật sắp xếp: Selection Sort, Insertion Sort, Bubble Sort, Quick Sort và Heap Sort. Các bước cụ thể và thuật toán cho mỗi giải thuật.

So sánh hiệu suất của các giải thuật: Đánh giá độ phức tạp, thời gian hoàn thành, số lượng phép so sánh và hoán đổi giữa các giải thuật đối với việc xử lý các tệp dữ liệu lớn.

Đánh giá khả năng ứng dụng: Xem xét các tình huống và loại dữ liệu mà mỗi giải thuật tối ưu tốt nhất. Xác định ưu điểm và nhược điểm của từng giải thuật trong các tình huống khác nhau.

### 3.2 Câu hỏi cụ thể muốn trả lời trong quá trình nghiên cứu

Cách thức hoạt động của mỗi giải thuật sắp xếp là gì?

Giải thuật nào có hiệu suất tốt nhất trong các tình huống khác nhau?

Giải thuật nào phù hợp với các tập dữ liệu lớn?

Có thể cải thiện hiệu suất của các giải thuật thông qua tối ưu hóa?

### CHƯƠNG IV: PHƯƠNG PHÁP NGHIÊN CỨU

### 4.1 Thu thập dữ liệu

Các tệp dữ liệu cần được sắp xếp được dùng để thử nghiệm để đánh giá hiệu suất giữa các giải thuật sắp xếp được tạo ngẫu nhiên. Bao gồm nhiều trường hợp như số liệu nhỏ, số liệu lớn, trường hợp tốt nhất và trường hợp xấu nhất. Bộ dữ liệu trong bài báo cáo này được tạo ngẫu nhiên qua mã nguồn dưới ngôn ngữ C.

int create\_file(int amount)

{

FILE \*file;

char filename[] = "file.inp";

int number;

int i;

file = fopen(filename, "w");

if (file == NULL)

{

printf("Khong the mo file.\n");

return 1;

}

fprintf(file, “%d\n", amount);

srand(time(0));

for (i = 0; i < amount; i++)

{

if (i != 0)

{

fprintf(file, " ");

}

number = rand() % amount + 1;

fprintf(file, "%d", number);

}

fclose(file);

return 0;

}

Đối với việc thay đổi tham số “ amount “ được truyền vào trong mã nguồn sẽ tạo ra “amount “ các số ngẫu nhiên từ 1 đến “amount”.

### 4.2 Tham số đánh giá

Xác định được các tham số để đánh giá hiệu suất hoạt động của các giải thuật. Bao gồm : Thời gian hoàn thành, số lượng các phép hoán đổi, số lượng các phép so sánh.

### 4.3 Tiến hành thử nghiệm

Tạo môi trường để tiến hành thử nghiệm các giải thuật sắp xếp. Quá trình thử nghiệm được dựa trên mã nguồn viết bằng ngôn ngữ C và đo lường các tham số cần được đánh giá. Bên cạnh đó, là mã nguồn C với thư viện đồ họa “graphics.h” được viết trên phần mềm DevC++(5.11 version with TDM-GCC 4.9.2 32-bit release) dùng để mô phỏng sự hoán đổi giữa các con số đối với từng giải thuật sắp xếp.

### 4.4 Phân tích kết quả

Sau khi thực hiện các thử nghiệm, cần phân tích kết quả để so sánh và đánh giá hiệu suất của các giải thuật sắp xếp. Có thể so sánh số liệu thu thập được, như thời gian hoàn thành, số lần so sánh và số lần hoán đổi giữa các giải thuật khác nhau trên cùng một tệp dữ liệu thử nghiệm hoặc giữa các tệp dữ liệu thử nghiệm khác nhau.

### 4.2 Đánh giá kết quả

Dựa trên phân tích kết quả, đánh giá hiệu suất của các giải thuật sắp xếp và rút ra các kết luận về sự ưu nhược điểm của từng giải thuật. Các yếu tố như thời gian thực thi, số lần so sánh và số lần hoán đổi có thể được sử dụng để đánh giá và so sánh hiệu suất của các giải thuật.

## CHƯƠNG V: KẾT QUẢ VÀ PHÂN TÍCH

### 5.1 Kết quả

Để có thể đánh giá được sự tối ưu của từng giải thuật cần dựa trên các tệp dữ liệu từ vừa đến lớn. Từ hàm thu thập dữ liệu đã được đề cập ở mục 4.1 để tạo ra 3 tệp dữ liệu lần lượt là 1.000 và 10.000 số.

#### 5.1.1 Bộ số liệu 10.000

|  |  |  |  |
| --- | --- | --- | --- |
| Giải thuật | Thời gian hoàn thành (s) | Số lần so sánh (lần) | Số lần hoán đổi (lần) |
| Selection sort | 0,08113 | 49.995.000 | 10.000 |
| Insertion sort | 0,06038 | 24.890.274 | 24.880.275 |
| Bubble sort | 0,1468 | 74.867.894 | 24.880.275 |
| Quick sort | 0,0291 | 165.530 | 34.361 |
| Heap sort | 0,02853 | 139.955 | 139.955 |
| Merge sort | 0,0267 | 120.408 | 0 |

#### 5.1.2 Bộ số liệu 100.000

|  |  |  |  |
| --- | --- | --- | --- |
| Giải thuật | Thời gian hoàn thành (s) | Số lần so sánh (lần) | Số lần hoán đổi (lần) |
| Selection sort | 3,599 | 4.999.950.000 | 100.000 |
| Insertion sort | 2,478 | 2.498.799.293 | 2.498.699.294 |
| Bubble sort | 17,04 | 7.498.219.166 | 2.498.699.294 |
| Quick sort | 0,05706 | 1.909.408 | 437.531 |
| Heap sort | 0,05489 | 1.728.529 | 1.728.529 |
| Merge sort | 0,05757 | 1.536.090 | 0 |

### 5.2 Đánh giá và kết luận

Đầu tiên, khi xét đến thời gian hoàn thành, giải thuật Merge sort cho thấy hiệu suất tốt nhất trong cả hai bộ số liệu. Với thời gian hoàn thành chỉ khoảng 0,0267 giây cho bộ số liệu 10.000 và 0,05757 giây cho bộ số liệu 100.000, Merge sort nhanh chóng và hiệu quả trong việc sắp xếp dữ liệu. Quick sort và Heap sort cũng đạt được hiệu suất tốt, với thời gian hoàn thành rất gần Merge sort.

Thứ hai, khi xét đến số lần so sánh và số lần hoán đổi, các giải thuật có sự khác biệt đáng kể. Trong bộ số liệu 10.000, Selection sort yêu cầu số lần so sánh và số lần hoán đổi nhiều nhất, lần lượt là 49.995.000 và 10.000. Điều này gây ra độ phức tạp với dữ liệu lớn, như trong bộ số liệu 100.000 khi số lần so sánh và số lần hoán đổi của Selection sort lần lượt là 4.999.950.000 và 100.000.

Insertion sort và Bubble sort cũng có số lần so sánh và số lần hoán đổi lớn, tăng theo cấp số nhân khi tăng kích thước của bộ số liệu. Quick sort và Heap sort đạt được hiệu suất tốt hơn, với số lần so sánh và số lần hoán đổi ở mức thấp, đặc biệt là trong bộ số liệu 100.000.

Từ những kết quả này, có thể thấy rằng Merge sort, Quick sort và Heap sort là những giải thuật sắp xếp hiệu quả với hiệu suất tốt trong cả thời gian hoàn thành và số lần so sánh/số lần hoán đổi. Trong khi đó, Selection sort, Insertion sort và Bubble sort có hiệu suất kém hơn và không thích hợp cho việc sắp xếp dữ liệu lớn.

# C. TÀI LIỆU THAM KHẢO:

[1] Nguyễn Đình Hoàng Sơn. *Bài giảng Cấu Trúc Dữ Liệu và Giải Thuật*. ĐH Nha Trang

[2] TRẦN THIÊN THÀNH. *Giáo trình CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT.* TRƯỜNG ĐẠI HỌC SƯ PHẠM QUY NHƠN

[3] Trân, N. N. B. 2001. Giáo trình Cấu trúc Dữ liệu và Giải thuật. KhoaCNTT, ĐH Bách KhoaTp.HCM.

[4] Kruse, R. L., and Ryba, A. J. 1999. Data Structures and Program Design in C++. Prentice-Hall Inc.