

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐỒ ÁN I
LẬP TRÌNH

Tóm tắt nội dung

Báo cáo gồm ba phần trình bày các bài toán sắp xếp, giải thuật quay lui, đánh chỉ mục từ khóa trong văn bản. Phần thứ nhất trình bày về ý tưởng, cài đặt và độ phức tạp của các thuật toán sắp xếp cơ bản bao gồm **bubble sort**, **selection sort**, **insertion sort**, **shell sort**, **quick sort**, **heap sort**. Phần thứ hai trình bày một số bài toán về giải thuật quay lui trong bài toán *tám hậu*, *mã di truyền* và kĩ thuật nhánh cận trong bài toán *người du lịch*. Phần thứ ba trình bày về các cách cài đặt khác nhau của bài toán đánh chỉ mục từ khóa trong văn bản trên các cấu trúc dữ liệu khác nhau bao gồm *danh sách liên kết*, *bảng băm*, *cây nhị phân tìm kiếm*.

Các chương trình trong báo cáo được trình bày bằng ngôn ngữ lập trình C++

1 Các thuật toán sắp xếp

1.1 Bài toán sắp xếp

Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng nào đó theo một thứ tự nhất định. Chẳng hạn như thứ tự tăng dần (hay giảm dần) đối với một dãy số, thứ tự từ điển đối với các từ v.v... Yêu cầu về sắp xếp thường xuyên xuất hiện trong các ứng dụng Tin học với các mục đích khác nhau: sắp xếp dữ liệu trong máy tính để tìm kiếm cho thuận lợi, sắp xếp các kết quả xử lý để in ra trên bảng biểu v.v...

Nói chung dữ liệu có thể xuất hiện dưới nhiều dạng khác nhau. Một tập các đối tượng cần được sắp xếp là tập các bản ghi (records), mỗi bản ghi bao gồm một số trường (fields) khác nhau. Nhưng không phải toàn bộ mà chỉ là một trường nào đó (hay một vài trường nào đó được chú ý tới thôi. Trường như vậy chúng ta gọi là **khóa** (textsfkey). Sắp xếp sẽ được tiến hành dựa vào giá trị của khóa này.

Khi sắp xếp, các bản ghi trong bảng sẽ được đặt vào các vị trí sao cho các giá trị khóa tương ứng của chúng có đúng thứ tự đã ấn định. thì kích thước của toàn bản ghi có thể rất lớn, nên nếu việc sắp xếp thực hiện trực tiếp trên các bản ghi sẽ đòi hỏi sự chuyển đổi vị trí của các bản ghi, kéo theo việc thường xuyên phải di chuyển, copy những vùng nhớ lớn, gây ra những tổn phí thời gian khá nhiều. Thường người ta khắc phục tình trạng này bằng cách xây dựng một bảng khoá: Mỗi bản ghi trong bảng ban đầu sẽ tương ứng với một bản ghi trong **bảng khoá**. Bảng khoá cũng gồm các bản ghi nhưng mỗi bản ghi chỉ gồm có hai trường:

- Trường thứ nhất chứa khóa.
- Trường thứ hai chứa liên kết tới một bản ghi trong bảng ban đầu, tức là một thông tin đủ để biết bản ghi tương ứng với nó trong bảng ban đầu là bản ghi nào.

Có thể coi *khóa như là đại diện cho các bản ghi* và để đơn giản, ta chỉ nói tới giá trị khóa mà thôi. Các thao tác trong kĩ thuật sắp xếp lẽ ra là tác động lên toàn bản ghi giờ đây chỉ làm việc trên khóa.

1.2 Một số quy ước

Mục này trình bày các cài đặt một số giải thuật sắp xếp phổ biến. Để thuận tiện hơn trong việc theo dõi chương trình sau này, ta đưa vào một số quy ước như sau:

- Dãy khóa cần được sắp xếp được lưu trong mảng **arr** gồm các phần tử từ $0 \dots len - 1$, trong đó, **len** là số phần tử của mảng.
- hàm **swap(a, b)** có tác dụng đổi chỗ hai phần tử **a** và **b**
- Kí hiệu $arr[i \dots j]$ được hiểu là các phần tử từ $arr[i]$ đến $arr[j]$ trong mảng **arr**.

1.3 Thuật toán sắp xếp nổi bọt

Trong thuật toán sắp xếp nổi bọt, các dãy khóa sẽ được duyệt từ cuối dãy lên đầu dãy (từ $arr[len-1]$ về $arr[0]$), nếu gặp hai khóa kề nhau bị ngược thứ tự thì đổi chỗ của chúng cho nhau, sau lần duyệt như vậy, khóa nhỏ nhất sẽ trở về vị trí đầu tiên, quá trình lại tiếp tục với các khóa từ dãy $arr[1]$ tới $arr[len-1]$:

Cài đặt Cài đặt thuật toán trong C++ như sau:

```
1 void bubblesort()
2 {
3     int i,j;
4     for (i = 1; i < len; i++)
```

```

5   for (j = len-1; j >= i; j--)
6       //Neu arr[j] va arr[j-1] nguoc thu tu
7       if (arr[j] < arr[j-1])
8       {
9           //Doi cho ve dung vi tri
10          swap(arr[j], arr[j-1]);
11          s++;
12      }
13 }

```

Đánh giá độ phức tạp thuật toán Chọn phép toán so sánh $arr[i] < arr[j]$ làm phép toán tích cực để đánh giá hiệu suất thực hiện của thuật toán về mặt thời gian. Ở lượt chọn thứ i thì cần $n - i$ phép so sánh (n là kích thước dữ liệu đầu vào, trong trường hợp này $n = len$). Vậy số lần thực hiện phép so sánh là:

$$(n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}$$

. Vậy thời gian thực hiện của thuật toán là $O(n^2)$

1.4 Thuật toán sắp xếp chọn

Một trong những thuật toán sắp xếp đơn giản nhất là phương pháp sắp xếp chọn. Ý tưởng của thuật toán:

- Trong lần duyệt đầu tiên, duyệt dãy từ $[0...len-1]$ tìm ra phần tử có giá trị nhỏ nhất đổi vị trí về đầu dãy.
- Trong lần duyệt thứ k duyệt dãy từ $[k-1...len-1]$ tìm ra phần tử nhỏ nhất trong dãy và đổi về vị trí $k-1$.
- ...
- Trong lần duyệt thứ $len-1$ chọn trong hai khóa $arr[len-2]$ và $arr[len-1]$ đổi với vị trí $len-2$

Kết thúc quá trình duyệt thì dãy còn lại đã được sắp xếp.

Cài đặt Sau đây là cài đặt của thuật toán trên C++:

```

1   void selectionsort()
2   {
3       int i, j, min;
4       for (i = 0; i < len-1; i++)
5       {
6           //Dat arr[i] lam phan tu be nhat
7           min = i;
8           //Tim phan tu be nhat trong day arr[i]-->arr[len-1]
9           for (j = i+1; j < len; j++)
10              if (arr[j] < arr[min])
11                  min = j;
12              if (i != min)
13                  //Doi cho phan tu be nhat ve vi tri
14                  //Day arr[0]-->arr[i] la day tang dan
15                  swap(arr[i], arr[min]);
16      }
17 }

```

Đánh giá độ phức tạp thuật toán Tương tự như giải thuật *bubble sort*, chọn phép toán $arr[i] < arr[min]$ làm phép toán tích cực để đánh giá độ phức tạp của thuật toán về mặt thời gian. Số lần thực hiện phép toán tích cực tương tự như đối với giải thuật *bubble sort* vẫn là $\frac{n(n-1)}{2} = O(n^2)$.

1.5 Thuật toán sắp xếp kiểu chèn

Xét dãy khóa $arr[0...len-1]$, ta thấy chỉ gồm một khóa $arr[0]$ chỉ gồm 1 khóa và có thể coi là đã sắp xếp rồi. Xét thêm $arr[1]$, ta so sánh nó với $arr[0]$, nếu thấy $arr[1] < arr[0]$ thì ta chèn nó vào trước $arr[1]$,... Một cách tổng quát, ta sẽ sắp xếp dãy $arr[0...i-1]$ trong điều kiện dãy khóa đã được sắp xếp rồi chèn $arr[i]$ vào dãy đó tại đúng vị trí để được dãy $arr[0...i]$ đã được sắp xếp.

Cài đặt

```
1 void insertionsort()
2 {
3     int i,j,tmp;
4
5     for (i = 1; i < len; i++)
6     {
7         tmp = arr[i];
8         //Chen phan tu arr[i] vao day da co thu tu
9         j = i - 1;
10        //Tim tu ben phai sang phan tu be hon arr[i]
11        while ((j >= 0) && (arr[j] > tmp))
12        {
13            //Don cac phan tu ve ben phai tao 1 cho chua cho arr[i]
14            arr[j+1] = arr[j];
15            j = j - 1;
16        }
17        //Chen phan tu vao dung vi tri,
18        //Day arr[0]-->arr[i] la day tang dan
19        arr[j+1] = tmp;
20    }
21 }
```

Đánh giá độ phức tạp thuật toán Đối với thuật toán sắp xếp kiểu chèn, thì chi phí thời gian thực hiện thuật toán phụ thuộc vào tình trạng dãy khoá ban đầu. Nếu coi phép toán tích cực ở đây là phép so sánh $arr[i] > tmp$ thì:

- Trường hợp tốt nhất, ứng với dãy khoá đã sắp xếp rồi, mỗi lượt chỉ cần thực hiện một phép so sánh, như vậy, tổng số phép so sánh cần thực hiện là $n - 1$. Phân tích trong trường hợp tốt nhất, độ phức tạp của thuật toán *Insertion Sort* là $\Theta(n)$.
- Trường hợp tồi nhất ứng với dãy khoá đã có thứ tự ngược lại, ở lượt sắp xếp thứ i cần thực hiện $i - 1$ phép so sánh. Số phép so sánh cần thực hiện là:

$$(n - 1) + (n - 2) + \dots + 1 = \frac{n(n - 1)}{2}$$

Vậy thời gian thực hiện của thuật toán trong trường hợp tồi nhất là $O(n^2)$.

- Trong trường hợp các khoá xuất hiện một cách ngẫu nhiên, có thể coi lượt thứ i cần thực hiện $i/2$ phép so sánh. Tổng số phép so sánh trong trường hợp này là:

$$\frac{1}{2} + \frac{2}{2} + \dots + \frac{n}{2} = \frac{n(n - 1)}{4}$$

Vậy phân tích trong trường hợp trung bình, độ phức tạp tính toán của *Insertion Sort* là $\Theta(n^2)$.

1.6 Shell sort

Nhược điểm của thuật toán sắp xếp chèn thể hiện khi mà ta luôn phải chèn một khoá vào vị trí gần đầu dãy. Trong trường hợp đó, người ta sử dụng phương pháp *Shell Sort*. Xét dãy khoá $arr[0...len-1]$. Với một số nguyên dương $0 \leq h \leq len - 1$, ta có thể chia dãy đó thành dãy con:

- Dãy con 1: $arr[0], arr[h], arr[2h], \dots$
- Dãy con 2: $arr[1], arr[h+1], arr[2h+1], \dots$
- Dãy con 3: $arr[2], arr[h+2], arr[2h+2], \dots$
- ...
- Dãy con $h-1$: $arr[h-1], arr[2h-1], arr[3h-1], \dots$

Những dãy con như vậy được gọi là dãy con sắp xếp theo độ dài h . Tư tưởng của thuật toán ShellSort là: Với một bước h , áp dụng thuật toán sắp xếp kiểu chèn từng dãy con độc lập để làm mịn dần dãy khoá chính. Rồi lại làm tương tự đối với bước $h\%2 \dots$ cho tới khi $h = 1$ thì ta được dãy khoá đã sắp xếp. Đây chính là nguyên nhân ShellSort hiệu quả hơn thuật toán sắp xếp chèn: khoá nhỏ được nhanh chóng đưa về gần vị trí đúng của nó.

Cài đặt

```
1 void shellSort()
2 {
3     int i,j,h=len/2,tmp;
4
5     while (h > 0)
6     {
7         for (i = h; i < len; i++) //Sap xep chen voi cac day con
8         {
9             tmp = arr[i];
10            j = i - h;
11            while (j >= 0 && arr[j] > tmp)
12            {
13                arr[j+h] = arr[j];
14                j = j - h;
15            }
16            arr[j+h] = tmp;
17        }
18        h = h/2; //Tang so day con
19    }
20 }
```

Đánh giá độ phức tạp thuật toán Việc đánh giá độ phức tạp của *Shell Sort* là tương đối khó, ta thừa nhận các kết quả sau:

- Nếu các bước h được chọn theo thứ tự ngược từ dãy $1, 3, 7, 15, \dots, 2^i - 1, \dots$ thì độ phức tạp là $O(n^{\frac{3}{2}})$.
- Nếu các bước h được chọn theo thứ tự ngược từ dãy $1, 8, 23, 77, \dots, 4^i + 1 + 3 \cdot 2^i + 1, \dots$ thì độ phức tạp là $O(n^{\frac{4}{3}})$.
- Nếu các bước h được chọn theo thứ tự ngược từ dãy $1, 2, 3, 4, 6, 8, 12, 16, \dots, 2^i 3^i, \dots$ thì độ phức tạp là $O(n \log^2 n)$.

1.7 Thuật toán sắp xếp Quick Sort

Thuật toán Quick Sort được đề xuất bởi C.A.R.Hoare là một phương pháp sắp xếp tốt nhất, nghĩa là dù dãy khóa thuộc kiểu dữ liệu *có thứ tự nào*, Quick Sort cũng có thể sắp xếp được và chaww có một thuật toán sắp xếp tổng quát nào nhanh hơn Quick Sort về mặt tốc độ trung bình.

Ý tưởng chủ đạo của phương pháp có thể tóm tắt như sau:

Sắp xếp dãy khóa $arr[0 \dots len-1]$ thì có thể coi là sắp xếp từ chỉ số 0 tới chỉ số $len-1$ trong dãy khóa đó. Để sắp xếp một đoạn trong dãy khóa, nếu đoạn đó có ít hơn 1 khóa thì không cần phải làm gì cả, còn nếu đoạn đó có ít nhất 2 khóa thì ta chọn ngẫu nhiên khóa nào đó của đoạn làm "chốt" (*pivot*). Một khóa nhỏ hơn chốt được xếp vào vị trí đứng sau chốt. Sau phép hoán chuyển như vậy thì đoạn đang xét được chia làm hai đoạn khá rộng mà mọi khóa trong đoạn đầu đều \leq chốt và một khóa trong đoạn sau đều \geq chốt. Và vấn đề trở thành sắp xếp hai đoạn mới tạo ra (có độ dài ngắn hơn đoạn ban đầu) bằng phương pháp tương tự.

Để cài đặt thuật toán này trong C++, ta sử dụng hai hàm:

- Hàm `void partition(int L, int H)` chọn một chốt pivot bất kì trong dãy từ $arr[L \dots H]$, chuyển các phần tử trong dãy có giá trị bé hơn pivot về trước pivot và các phần tử có giá trị lớn pivot về sau pivot. Sau đó gọi đệ quy tiếp tục với hai dãy con nhỏ hơn.
- Hàm `void quicksort()` gọi hàm `partition(0, len-1)` thực hiện sắp xếp toàn bộ mảng. Thực ra ta có thể không cần tới hàm này.

Cài đặt quicksort:

```
1 void partition(int L, int H)
2 {
3     int i,j,pivot;
4     //Neu chi co duoi 1 phan tu thi khong phai lam di
5     if (L>=H)
6         return;
7     i=L;j=H;
```

39

thuộc vào trạng thái của dây khóa dầu vào.

$$T(n) = 2T(\frac{n}{2}) + Cn$$

trong đó C là hằng số.

$$\begin{aligned} T(n) &= 2(2T(\frac{n}{4}) + C\frac{n}{2}) + Cn \\ \Rightarrow T(n) &= 2^2T(\frac{n}{4}) + 2Cn \\ \Rightarrow T(n) &= \\ \Rightarrow T(n) &= 2^kT(\frac{n}{2^k}) + kCn \end{aligned}$$

$T(\frac{n}{2^k}) = T(1)$ khi $n = 2^k \Rightarrow k = \log_2 n$. Thay $k = \log_2 n$ vào công thức trên, ta được:

$$T(n) = 2^{\log_2 n} T(1) + \log_2 n Cn$$

$$\Rightarrow T(n) = nT(1) + Cn \log_2 n = \Theta(n \log n)$$

$$\begin{aligned} T(n) &= n - 1 + T(n - 1) \\ &= (n - 1) + (n - 2) + T(n - 2) \\ &= \\ &= (n - 1) + (n - 2) + \dots + 1 \\ &= \frac{n(n-1)}{2} \end{aligned}$$

¹giá trị mà số khóa bé hơn giá trị đó bằng số khóa lớn hơn giá trị đó

- Trong trường hợp trung bình, cách tính độ phức tạp thuật toán tương tự như trường hợp tốt nhất. Giả sử ở lần gọi đệ quy thứ i được chia thành 2 dãy $\frac{a_i}{c}n_i$ phần tử và $\frac{b_i}{c}n_i$ phần tử, trong đó $a_i + b_i = n_i$ và $a_i < b_i$, ta có:

$$\begin{aligned} T(n) &= T\left(\frac{a_{n-1}}{c}\right) + T\left(\frac{b_{n-1}}{c}\right) + Cn \\ T(n) &= \left(T\left(\frac{a_{n-1}}{c}\right) + \frac{a_{n-1}}{c}Cn\right) + \left(T\left(\frac{b_{n-1}}{c}\right) + \frac{b_{n-1}}{c}Cn\right) \\ T(n) &= \max\left(T\left(\frac{a_{n-1}}{c}\right) + \frac{a_{n-1}}{c}Cn, T\left(\frac{b_{n-1}}{c}\right) + \frac{b_{n-1}}{c}Cn\right) \\ T(n) &= T\left(\frac{b_{n-1}}{c}\right) + \frac{b_{n-1}}{c}Cn \end{aligned}$$

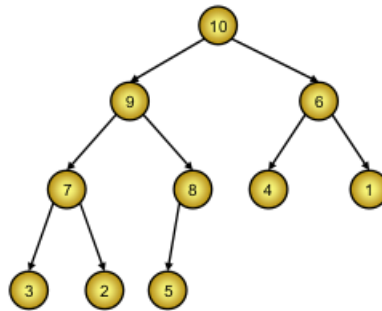
Giải tương tự như trường hợp 1, ta tính được độ phức tạp của thuật toán *Quick Sort* trong trường hợp này là $\Theta(n \log n)$.

1.8 Thuật toán sắp xếp vung đồng - Heap Sort

Heap Sort được đề xuất bởi J.W.J. Williams năm 1981, thuật toán không những đóng góp một phương pháp sắp xếp quan trọng để biểu diễn một cấu trúc dữ liệu quan trọng

1.8.1 Đồng - Heap

Đồng là một dạng *cây nhị phân hoàn chỉnh đặc biệt* mà giá trị tại mọi nút có độ ưu tiên cao hơn hay bằng giá trị lưu trong hai nút con của nó. Trong thuật toán sắp xếp kiểu vung đồng, ta coi quan hệ "ưu tiên hơn hay bằng" là quan hệ "lớn hơn hay bằng": \geq



Hình 1: Đồng

1.8.2 Vun đồng

Để biểu diễn dãy khóa $arr[0..len-1]$ thành một cây nhị phân hoàn chỉnh. Giá trị $arr[i]$ lưu trong nút thứ i (i bắt đầu từ 0). Nút con trái của nút thứ i là $arr[2i+1]$ và con phải của nút thứ i là $arr[2i+2]$. Nút cha của nút thứ j là $(j-1)\%2$.

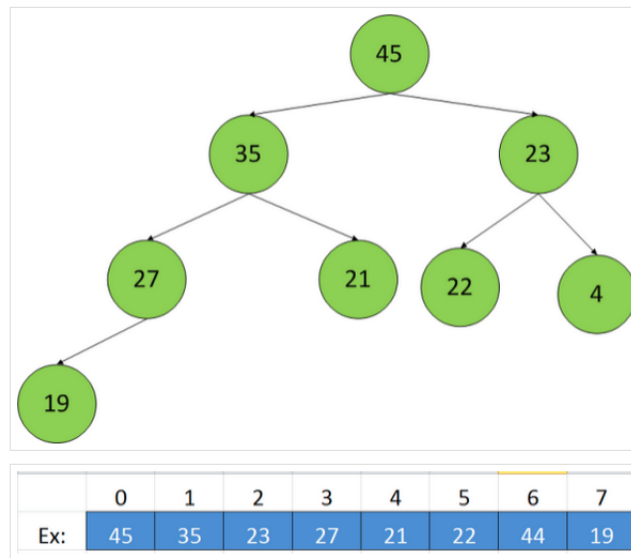
Vì cây nhị phân gồm một nút thì hiển nhiên là đồng. nên **để vun một nhánh cây gốc r thành đồng, ta có thể coi hai nhánh con của nó là đồng rồi và thực hiện thuật toán vun đồng từ dưới lên đối với cây**. Gọi h là chiều cao của cây, nút ở mức h (nút lá) đã là một đồng, ta vun lên những nút ở mức $h-1$ cũng là gốc của đồng,... cứ như vậy cho tới mức 1 cũng là gốc của đồng.

Thuật toán vun đồng đối với gốc r khi hai nhánh con đã là đồng Giả sử nút r chứa giá trị V . Từ r , ta cứ đi tới nút con chứa giá trị lớn nhất trong hai nút con, cho tới khi gặp phải một nút c mà mọi nút con của c đều có giá trị $\leq V$ (nút lá cũng là trường hợp riêng của điều kiện này). Dọc đường đi từ r tới c , ta đẩy giá trị chứa ở nút con lên nút cha và đặt giá trị V vào nút c .

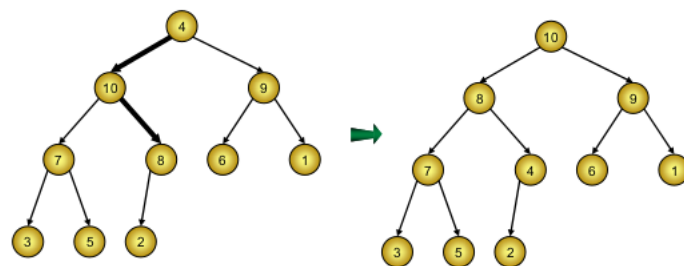
1.8.3 Tư tưởng của Heap Sort

Đầu tiên với dãy khóa $arr[0..len-1]$ được vun từ dưới lên để nó biểu diễn một đồng, khi đó khóa $arr[0]$ tương ứng với nút gốc của đồng là khóa lớn nhất, ta đảo giá trị khóa đó cho $arr[len-1]$ và không tính tới $arr[len-1]$ nữa. Còn lại dãy khóa $arr[0..len-2]$ tuy không là đồng nữa nhưng là biểu diễn của cây nhị phân hoàn chỉnh mà hai nhánh cây đã là đồng rồi. Vậy chỉ cần vun một lần, ta lại được một đồng, đảo giá trị $arr[0]$ với $arr[len-2]$ rồi tiếp tục cho tới khi đồng chỉ còn lại một nút.

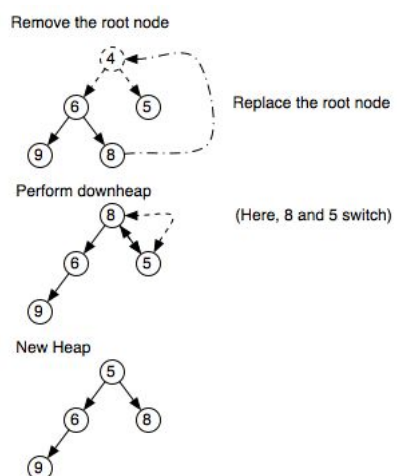
Thuật toán Heap Sort có hai hàm chính:



Hình 2: Biểu diễn đồng bằng mảng



Hình 3: Vun đống



Hình 4: Đảo giá trị của $\text{arr}[0]$ và $\text{arr}[\text{len}-1]$ rồi xét lại

- Hàm `maxHeapify` làm nhiệm vụ vun cây gốc root thành đồng trong điều kiện hai gốc con $2 * root + 1$ và $2 * root + 2$ đã là đồng rồi. Các nút từ `endnode+1` tới `len-1` đã nằm đúng vị trí, không được tính tới nữa.
- Hàm `heapSort` mô tả lại quá trình vun đồng theo ý tưởng trên

Cài đặt

```

1 void maxHeapify(int root, int endnode)
2 {
3     int c1, c2, bg;
4
5     while (root <= endnode/2-1) //Khi nao root chưa phải là la
6     {
7         c1 = 2*root + 1; //c1 = con trai
8         c2 = 2*root + 2; //c2 = con phải
9         bg = c1; //con lớn hơn là c1
10        //Nếu con phải lớn hơn con trai thì gán bigger cho con phải
11        if ((c2 < len) && (arr[c2] > arr[bg]))
12            bg = c2;
13        //Nếu 1 trong 2 con lớn hơn root thì đổi cho
14        if (arr[bg] > arr[root])
15            swap(arr[root], arr[bg]); //Đổi cho cho con lớn hơn
16        //Tiếp tục với vị trí đã chuyển
17        root = bg;
18    }
19 }
20
21 void heapSort()
22 {
23     int r, i;
24     for (r = (len/2-1); r >= 0; r--) // Xây dựng đồng trên toàn bộ cây
25         maxHeapify(r, len);
26
27     for (i = len-1; i > 0; i--)
28     {
29         swap(arr[0], arr[i]); //Đổi cho nút lớn nhất về cuối dãy
30         maxHeapify(0, i-1); //Xây lại đồng, bỏ bớt nút vừa hoán đổi
31     }
32 }

```

Độ phức tạp thuật toán Thuật toán sắp xếp *Heap Sort* có độ phức tạp $n \log n$ trong mọi trường hợp.

1.9 Đánh giá thời gian thực tế của thuật toán

Sau đây là một số kết quả chạy thử với các bộ dữ liệu đầu vào là các dãy số được tạo ngẫu nhiên. Thời gian được tính theo đơn vị *ms*.

```
thongbkvn@EliteBook:~/Project/DA/src/bail$ ./bail
LENGTH: 20
Buble Sort: 0.005

15 15 25 31 32 34 44 45 48 52
57 61 63 65 68 70 70 75 87 91

Selection Sort: 0.004

15 15 25 31 32 34 44 45 48 52
57 61 63 65 68 70 70 75 87 91

Insertion Sort: 0.002

15 15 25 31 32 34 44 45 48 52
57 61 63 65 68 70 70 75 87 91

Shell Sort: 0.003

15 15 25 31 32 34 44 45 48 52
57 61 63 65 68 70 70 75 87 91

heapSort: 0.005

15 15 25 31 32 34 44 45 48 52
57 61 63 65 68 70 70 75 87 91

Quick Sort: 0.101

15 15 25 31 32 34 44 45 48 52
57 61 63 65 68 70 70 75 87 91

thongbkvn@EliteBook:~/Project/DA/src/bail$
```

Hình 5: Dãy đầu vào có 20 phần tử

Trường hợp này, *Quick Sort* chạy lâu nhất bởi có nhiều lời gọi đệ quy nhất. Tốc độ thực hiện kém xa các thuật toán khác.

```
thongbkvn@EliteBook:~/Project/DA/src/bail$ ./bail
LENGTH: 100
Buble Sort: 0.153
Selection Sort: 0.057
Insertion Sort: 0.017
Shell Sort: 0.015
heapSort: 0.042
Quick Sort: 0.504
```

Hình 6: Dãy đầu vào có 100 phần tử.

Quick Sort vẫn tỏ ra kém hơn so với các thuật toán khác. *Buble Sort* bắt đầu tỏ ra chậm hơn so với *Selection Sort* và *Insertion Sort* do thực hiện nhiều phép trao đổi.

```
thongbkvn@EliteBook:~/Project/DA/src/bail$ ./bail
LENGTH: 10000
Buble Sort: 504.957
Selection Sort: 132.699
Insertion Sort: 92.985
Shell Sort: 2.357
heapSort: 2.807
Quick Sort: 14.915
```

Hình 7: Dãy đầu vào có 10 000 phần tử.

Các thuật toán sắp xếp *Shell Sort*, *Heap Sort*, *Quick Sort* vượt trội hơn hẳn. *Buble Sort* chậm nhất với 0.5s.

```
thongbkvn@EliteBook:~/Project/DA/src/bai1$ ./bai1
LENGTH: 100000
Buble Sort: 57515
Selection Sort: 13578.3
Insertion Sort: 11379.4
Shell Sort: 34.281
heapSort: 36.019
Quick Sort: 158.146
```

Hình 8: Dãy đầu vào có 100 000 phần tử.

Các thuật toán sắp xếp như *Buble Sort*, *Selection Sort*, *Insertion Sort* yếu kém hơn hẳn, ta sẽ không xét đến chúng trong ví dụ tới nữa. *Shell Sort* và *Heap Sort* vẫn rất ấn tượng với thời gian sắp xếp là 0.34s.

```
thongbkvn@EliteBook:~/Project/DA/src/bai1$ ./bai1
LENGTH: 10000000
Shell Sort: 7101.02
heapSort: 7283
Quick Sort: 18321.9

thongbkvn@EliteBook:~/Project/DA/src/bai1$ ./bai1
LENGTH: 10000000
Shell Sort: 7287.67
heapSort: 6936.55
Quick Sort: 19720.8

thongbkvn@EliteBook:~/Project/DA/src/bai1$ ./bai1
LENGTH: 10000000
Shell Sort: 7232.96
heapSort: 6936.97
Quick Sort: 19647.1
```

Hình 9: Dãy đầu vào có 10 000 000 phần tử.

Heap Sort và *Shell Sort* vẫn tỏ ra rất vượt trội.

2 Các bài toán về thuật toán quay lui và kĩ thuật nhánh cận

2.1 Thuật toán quay lui

Thuật toán quay lui dùng để giải bài toán *liệt kê* các cấu hình. Mỗi cấu hình được xây dựng bằng cách xây dựng từng phần tử, mỗi phần tử được chọn bằng cách thử tất cả các khả năng. Giả sử cấu hình liệt kê có dạng $X[1..n]$, khi đó thuật toán quay lui thực hiện qua các bước:

1. Xét tất cả các giá trị $X[1]$ có thể nhận, thử cho $X[1]$ nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho $X[1]$ ta sẽ:
2. Xét tất cả các giá trị $X[2]$ có thể nhận, thử cho $X[2]$ nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho $X[2]$ lại xét tiếp khả năng chọn $X[3]...$ cứ tiếp tục như vậy đến bước:
3. Xét tất cả các giá trị $X[n]$ có thể nhận, thử cho $X[n]$ nhận lần lượt các giá trị đó, thông báo cấu hình tìm được $(X[1], X[2], \dots, X[n])$.

Trên phương diện quy nạp, có thể nói rằng thuật toán quay lui liệt kê các cấu hình n phần tử dạng $X[1..n]$ bằng cách thử cho $X[1]$ nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho $X[1]$ bài toán trở thành liệt kê cấu hình $n-1$ phần tử sau $X[2..n]$.

Mô hình của thuật toán quay lui có thể mô tả như sau:

```
void Try(int i)
{
    for (mọi giá trị V có thể gán cho X[i])
    {
        (Thử cho X[i] = V);
        if (X[i] là phần tử cuối cùng trong cấu hình)
            (thông báo cấu hình tìm được);
        else
```

```

    {
        (Ghi nhận việc X[i] đã được gán giá trị V);
        Try(i+1);
        (Nếu cần, bỏ việc ghi nhận X[i] = V để thử giá trị khác);
    }
}
}

```

Thuật toán quay lui sẽ bắt đầu bằng lời gọi Try(1) ²

2.2 Bài toán mã đi tuần

2.2.1 Bài toán

Xét bàn cờ kích thước kích thước $N * N$. Một quân mã xuất phát tại vị trí (firstRow, firstCol). Quân mã phải đi theo đúng quy tắc bàn cờ vua và đi hết các ô trên bàn cờ sao cho mỗi ô đều được đi qua 1 lần.

2.2.2 Cài đặt thuật toán

Với bài toán này, ta sẽ giải quyết theo thuật toán quay lui. Bài toán sử dụng các biến và hàm sau:

Hằng N Kích thước của bàn cờ.

Biến firstRow, firstCol Vị trí ban đầu của quân mã.

Mảng A[N][N] Lưu thứ tự các nước đi của quân mã, đồng thời cũng là mảng đóng vai trò đánh dấu khi ô tương ứng đã được đi qua. Vì vậy cần bỏ việc ghi nhận sau lời gọi đệ quy Try(i+1) để thử giá trị khác theo sơ đồ trên.

Hàm printBoard() In ra bàn cờ và thứ tự vị trí quân mã đã đi qua.

Hàm Patrol Hàm thực hiện quay lui để thử tất cả các cách đi quân mã trên bàn cờ có thể được và xuất kết quả ra màn hình nếu đi hết các ô.

Cài đặt thuật toán trên C++

```

1  #include <iostream>
2  #include <iomanip>
3  #include <math.h>
4  using namespace std;
5  #define N 4 // Kích thước bàn cờ
6
7  /*
8   CLASS BAN CO
9   Thuộc tính:
10  +) A[N][N]: Trạng thái bàn cờ về những vị trí quân mã đã tới
11  +) firstRow, firstCol: Vị trí xuất phát của quân mã
12  Phương thức:
13  +) chessBoard(m,n): Khởi tạo bàn cờ, m,n là vị trí xuất phát của quân mã
14  +) printBoard(): Hiện thị bàn cờ
15  +) Patrol(m,n): Di tuần từ vị trí m*n
16  */
17  class chessboard
18  {
19  private:
20      int A[N][N];
21      int firstRow, firstCol;
22  public:
23      chessboard(int m=0, int n=0)
24      {
25          int i, j;
26          for (i = 0; i < N; i++)
27              for (j = 0; j < N; j++)
28                  A[i][j] = 0;
29          firstRow = m;

```

²Theo mô tả thuật toán quay lui ở đây thì sẽ bắt đầu với Try(1). Không phải các bài toán sử dụng thuật toán quay lui phải luôn bắt đầu bằng lời gọi Try(1), tùy theo việc cài đặt thuật toán.

```

30         firstCol = n;
31         A[m][n] = 1;
32     }
33
34     void printBoard() //Hien thi ban co
35     {
36         int i, j;
37         static int n = 1; // Bien phu, khong can de y
38         cout<<"\nCach " <<n<<":\n"<<endl;
39         for (i = 0; i < N; i++)
40         {
41             for (j = 0; j < N; j++)
42                 cout<<setw(3)<<A[i][j];
43             cout<<endl;
44         }
45         n++;
46         cout<<endl;
47     }
48     // Tim nuoc di, nuoc di dau tien da duoc di tai firstRow, firstCol
49     void Patrol(int m, int n, int k=2)
50     {
51         int i, j;
52         for (i = 0; i < N; i++)
53             for (j = 0; j < N; j++)
54                 // Neu co the di duoc toi o i,j va o do chua duoc di toi
55                 if (((fabs(i-m) + fabs(j-n)) == 3) && (i!=m && j!=n && A[i][j]==0))
56                 {
57                     // Thu di toi A[i][j], luu ket qua va dau hieu vao mang A
58                     A[i][j] = k;
59                     // Neu da la nuoc di cuoi
60                     if (k == N*N)
61                     {
62                         printBoard(); // Hien thi ban co
63                         // Buoc dau tien thi khong bo danh dau
64                         //Neu khong thi bo danh dau
65                         if ((i != firstRow) && (j != firstCol))
66                             A[i][j]=0;
67                     }
68                     else
69                     {
70                         Patrol(i, j, k+1); // Thu di nuoc tiep theo
71                         A[i][j] = 0; // Bo danh dau
72                     }
73                 }
74     }
75
76
77 };
78
79
80
81 int main()
82 {
83     int m,n;
84     cout<<"BAI TOAN MA DI TUAN\nKich thuoc: " <<N<<'x'<<N<<endl;
85     cout<<"Nhap vi tri xuat phat: ";
86     cin>>m>>n;
87     chessboard a(m,n);
88     a.Patrol(m,n);
89     cout<<endl;
90     return 0;
91 }
92

```

```

BAI TOAN MA DI TUAN
Kich thuc: 8x8
Nhap vi tri xuat phat: 0 0

Cach 1:

 1 12  9  6  3 14 17 20
10  7  2 13 18 21  4 15
31 28 11  8  5 16 19 22
64 25 32 29 36 23 48 45
33 30 27 24 49 46 37 58
26 63 52 35 40 57 44 47
53 34 61 50 55 42 59 38
62 51 54 41 60 39 56 43

```

Hình 10: Kết quả chạy thử trên bàn cờ 8x8

Kết quả chạy thử

2.3 Bài toán tám quân hậu

2.3.1 Bài toán

Bài toán tám quân hậu là bài toán đặt tám *quân hậu* trên bàn cờ vua kích thước 8x8 sao cho không có quân hậu nào có thể "ăn" được quân hậu khác. Như vậy, lời giải của bài toán là một cách xếp tám quân hậu trên bàn cờ sao cho không có hai quân nào đứng trên cùng hàng, hoặc cùng cột hoặc cùng đường chéo. Bài toán tám quân hậu có thể được tổng quát hóa thành bài toán N quân hậu.

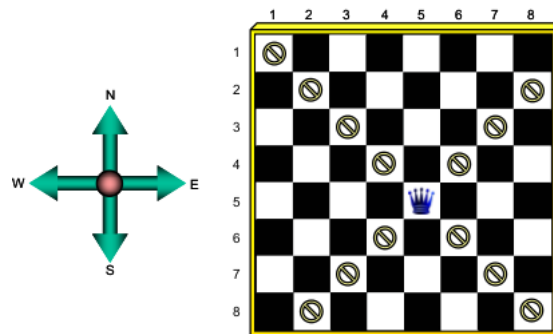
2.3.2 Cài đặt thuật toán

Bài toán này cũng sẽ được giải quyết theo thuật toán quay lui.

Phân tích Rõ ràng n quân hậu, mỗi con sẽ được đặt ở một hàng vì quân hậu được ăn ngang, ta gọi quân hậu sẽ đặt ở hàng 0 là quân hậu 0, quân hậu được đặt ở hàng 1 là quân hậu 1, quân hậu được đặt ở hàng n là quân hậu n. Như vậy, một nghiệm của bài toán sẽ được biết khi ta tìm ra được **vị trí cột của những quân hậu**.

Nếu ta định hướng Đông - Phải, Tây - Trái, Nam - Dưới, Bắc - Trên thì ta sẽ nhận thấy rằng:

- Một đường chéo theo hướng ĐB-TN bất kì sẽ đi qua một số ô, các ô đó có tính chất Hàng + Cột = C (hằng số). Với mỗi đường chéo ĐB-TN ta có một hằng số C và với một hằng số C: $0 \leq C \leq 2n - 2$ xác định duy nhất một đường chéo ĐB-TN từ 0 đến $2n - 2$.
- Một đường chéo hướng DN-TB bất kì đi qua một số ô, các ô có tính chất: Hàng - Cột = C. Với mỗi đường chéo DN-TB ta có một hằng số C và với một hằng số C: $-N + 1 \leq C \leq N - 1$ xác định duy nhất một đường chéo DN-TB.



Hình 11: Đường chéo ĐB-TN mang chỉ số 10 và đường chéo DN-TB mang chỉ số 0

Cài đặt

- Sử dụng 3 mảng để đánh dấu:
 - $C[N]$ $C[i] = 0$ nếu cột i còn tự do, $C[i] \neq 0$ nếu cột i đã bị một quân hậu khống chế.

$D1[2*N]$ $D1[i] = 0$ nếu đường chéo DB-TN thứ i $[0 \leq i = row + col \leq 2N]$ còn tự do, $D1[i] \neq 0$ nếu đường chéo đó đã bị khống chế.
 $D2[2*N]$ $D2[i] = 0$ nếu đường chéo DN-TB thứ i $[0 \leq i = row - col + N - 1 \leq 2N]$ còn tự do, $D2[i] \neq 0$ nếu đã bị khống chế.

- Sử dụng mảng $A[N][N]$ để lưu kết quả.
- Hàm `printBoard` dùng để xuất kết quả ra màn hình.
- Hàm `Set` dùng để để đánh dấu và hủy đánh dấu các mảng đánh dấu khi thử đặt quân hậu vào vị trí hàng m , cột n . Tham số `unset` mặc định là 0 - bật đánh dấu. Nếu đặt bằng 1 sẽ là hủy đánh dấu.
- Hàm `checkBoard` dùng để kiểm tra xem nước đi vào vị trí hàng i , cột j có đi được không bằng cách kiểm tra các mảng đánh dấu.
- Hàm `setBoard` dùng giải thuật quay lui để thử tất cả các cách đặt quân hậu có thể được. Đây là chính trong bài toán này.

Cài đặt bài toán tám hậu trên C++

```

1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  #define N 8
5
6  /*
7   CLASS BAN CO
8   -Thuoc tinh:
9   +)C[N]: Mảng ngang cột đã bị kiểm soát
10  +)D1[2*N]: Đường chéo i+j=const đã bị kiểm soát
11  +)D2[2*N]: Đường chéo i-j=const đã bị kiểm soát
12  +)A[N][N]: Mảng biểu diễn cách đặt các quân co
13  -Phương thức
14  +)printBoard: Hiện thị bàn co
15  +)Set(m,n,unset): Đặt quân co vào vị trí A[m][n] và cài
16  đặt các thuộc tính, nếu unset!=1 thì khôi phục lại
17  các thuộc tính
18  +)checkBoard(m,n): Kiểm tra xem có thể đặt được quân hậu
19  tại vị trí A[m][n] không
20  +setBoard(k): Đặt co theo phương pháp quay lui
21  */
22  class chessboard
23  {
24  private:
25      int C[N]; //Cột đã bị kiểm soát
26      int D1[2*N]; // Đường chéo / (i+j=const, 0 <= i+j <= 2*N-2) đã bị kiểm soát
27      int D2[2*N]; // Đường chéo \ (i-j=const, -N+1 <= i+j <= N-1) đã bị kiểm soát
28      int A[N][N]; //Bàn co, nếu Board[i][j]!=0 là đã được đặt
29
30  public:
31      chessboard() //Hàm tạo cho bàn co
32      {
33          int i, j;
34          for (i = 0; i < N; i++)
35          {
36              C[i] = 0;
37              D1[i] = 0;
38              D1[N+i] = 0;
39              D2[i] = 0;
40              D2[N+i] = 0;
41          }
42          for (i = 0; i < N; i++)
43              for (j = 0; j < N; j++)
44                  A[i][j] = 0;
45      }
46  }

```

```

47 void printBoard() //In ban co tu ma tran A[N][N]
48 {
49     int i, j;
50     static int n = 1;
51     cout<<"\n\nCach "<<n<<":\n"<<endl;
52     for (i = 0; i < N; i++)
53     {
54         for (j = 0; j < N; j++)
55             cout<<setw(3)<<A[i][j];
56         cout<<endl;
57     }
58     cout<<endl;
59     n++;
60 }
61
62 void Set(int m, int n,int unset=1) //Thu di quan co vao A[m][n]
63 {
64     if (unset == 1) //Danh dau la da di
65     {
66         C[n] = 1;
67         D1[m+n] = 1;
68         D2[m-n+N-1] = 1;
69     }
70     else //Quay tro lai, khong di buoc do nua
71     {
72         C[n] = 0;
73         D1[m+n] = 0;
74         D2[m-n+N-1] = 0;
75         A[m][n] = 0; //Van de do bieu dien du lieu dau ra
76     }
77 }
78
79 int checkBoard(int i, int j) //Kiem tra xem nuoc di co duoc khong
80 {
81     if ((C[j]==0) && (D1[i+j]==0) && (D2[i-j+N-1]==0))
82         return 1;
83     return 0;
84 }
85
86 void setBoard(int k=0)
87 {
88     int i;
89
90     for (i = 0; i < N; i++) //Voi moi cot
91         if (checkBoard(k, i)) //Kiem tra xem co the dat quan hau o hang
92             //thu k vao cot thu i hay ko
93         {
94             A[k][i] = k + 1; //Luu thong tin vao ban co
95             if (k == N-1) //Neu dat cau hinh cuoi
96             {
97                 printBoard(); //Hien thi ban co
98                 A[k][i] = 0; //Van de ve cach bieu dien du lieu dau ra
99             }
100             else
101             {
102                 Set(k, i, 1); //Danh dau la da di
103                 setBoard(k + 1); //Di quan co tiep theo
104                 Set(k, i, 0); //Quay tro lai vi tri truoc, bo danh dau
105                 // coi nhu chua di quan nay
106             }
107         }
108     }
109 };

```



```

110
111
112 int main()
113 {
114     chessboard a;
115     a.setBoard();
116     cout<<endl;
117     return 0;
118 }

```

× - □ Phạm Văn Thông	× - □ Phạm Văn Thông	× - □ Phạm Văn Thông
Cach 54:	Cach 91:	Cach 68:
0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 1 0 0
0 0 2 0 0 0 0 0	0 0 2 0 0 0 0 0	0 0 2 0 0 0 0 0
3 0 0 0 0 0 0 0	3 0 0 0 0 0 0 0	3 0 0 0 0 0 0 0
0 0 0 0 0 4 0 0	0 0 0 0 0 4 0 0	0 0 0 0 0 0 4 0
0 0 0 0 0 0 0 5	0 5 0 0 0 0 0 0	0 0 0 0 5 0 0 0
0 6 0 0 0 0 0 0	0 0 0 0 6 0 0 0	0 0 0 0 0 0 0 6
0 0 0 7 0 0 0 0	0 0 0 0 0 0 7 0	0 7 0 0 0 0 0 0
0 0 0 0 0 0 8 0	0 0 0 8 0 0 0 0	0 0 0 8 0 0 0 0
Cach 55:	Cach 92:	Cach 69:
0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 1 0 0
0 0 2 0 0 0 0 0	0 0 0 2 0 0 0 0	0 0 2 0 0 0 0 0
3 0 0 0 0 0 0 0	3 0 0 0 0 0 0 0	3 0 0 0 0 0 0 0
0 0 0 0 0 0 4 0	0 0 4 0 0 0 0 0	0 0 0 0 0 0 0 4
0 5 0 0 0 0 0 0	0 0 0 0 0 5 0 0	0 0 0 5 0 0 0 0
0 0 0 0 0 0 0 6	0 6 0 0 0 0 0 0	0 6 0 0 0 0 0 0
0 0 0 0 0 7 0 0	0 0 0 0 0 0 7 0	0 0 0 0 0 0 7 0
0 0 0 8 0 0 0 0	0 0 0 0 8 0 0 0	0 0 0 0 8 0 0 0

Hình 12: Kết quả chạy thử

Kết quả chạy thử

2.4 Bài toán người du lịch

2.4.1 Bài toán

Có một người giao hàng cần đi giao hàng tại n thành phố. Anh ta xuất phát từ một thành phố nào đó, đi qua các thành phố khác để giao hàng và trở về thành phố ban đầu. Mỗi thành phố chỉ đến một lần, và khoảng cách từ một thành phố đến các thành phố khác đã được biết trước. Hãy tìm một chu trình (một đường đi khép kín thỏa mãn điều kiện trên) sao cho tổng độ dài các cạnh là nhỏ nhất.

2.4.2 Cài đặt thuật toán

Ý tưởng Hành trình cần tìm có dạng $X[0...N]$, trong đó, $X[0] = X[N] = 0$, và ngoại trừ thành phố 0 là thành phố xuất phát thì không có thành phố nào được lặp lại 2 lần. Có nghĩ dãy $X[1...N-1]$ lập thành một hoán vị của $(1, 2, ..., N-1)$.

Duyệt quay lui: $X[1]$ có thể chọn một trong các thành phố có thể đi từ $X[0]$, với mỗi cách thử chọn $X[1]$ như vậy thì chọn $X[2]$ mà không trùng với $X[0]$, ... Tổng quát: $X[i]$ có thể chọn được các thành phố mà không trùng với các thành phố từ $X[0]$ đến $X[i-1]$.

Nhánh cận: Khởi tạo cấu hình **BestConfig** có chi phí $+\infty$. Với mỗi bước thử chọn $X[i]$ xem chi phí đường đi cho tới lúc đó có $<$ chi phí của cấu hình **BestConfig** không. Nếu không nhỏ hơn thì thử giá trị khác ngay bởi có đi tiếp cũng chỉ tốn thêm. Khi thử được một giá trị $X[N-1]$ thì đánh giá chi phí từ thành phố 1 đến thành phố $X[N-1]$ cộng với chi phí từ $X[N-1]$ về thành phố 0, nếu nhỏ hơn chi phí của đường đi ở **BestConfig** thì cập nhật lại **BestConfig** bằng cách đi mới.

Cài đặt

Mảng X , **BestWay** X ghi nhận đường đi trong quá trình thử, **BestWay** ghi nhận nghiệm.

Mảng T , **Free** Mảng T dùng để lưu tổng chi phí, $T[i]$ là chi phí từ thành phố 0 tới thành phố i . **Free** dùng để đánh dấu thành phố đã được đi qua hay chưa.

MinCost Số tiền tối thiểu của chi phí đường đi trong BestConfig, khởi tạo với giá trị 100000 biểu thị giá trị vô cùng lớn.

findWay Hàm quay lui thực hiện thử các khả năng. Tham số i biểu thị cho bước đi tới thành phố thứ i.

Cài đặt bài toán người du lịch trên C++

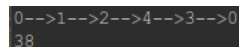
Input File văn bản "bai2.in".

- Dòng 1 chứa số N chỉ số lượng thành phố sẽ đi qua.
- N dòng tiếp theo, mỗi dòng là một hàng gồm N số nguyên dương biểu thị ma trận chi phí A. Trong đó $A[i][j]$ là chi phí đi từ thành phố i đến thành phố j. Ma trận A trong bài này là đối xứng.

```
5
0 6 12 6 20
6 0 6 16 14
12 6 0 14 16
6 16 14 0 4
20 14 16 4 0
```

Hình 13: Đầu vào từ file "bai2.in"

Output Xuất ra màn hình lộ trình tìm được và chi phí của lộ trình.



```
0-->1-->2-->4-->3-->0
38
```

Hình 14: Đầu ra chứa dãy biểu thị lộ trình và chi phí đường đi.

```
1  #include <iostream>
2  #include <fstream>
3  #include <time.h>
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <iomanip>
7  #define CITY 5
8  #define RANGE 20
9  using namespace std;
10
11
12  int N, **C;
13  int X[100], BestWay[100]; //X thu các khả năng, BestWay ghi nhận nghiệm
14  int T[100], Free[100], MinCost=100000; //T lưu số tiền, Free đánh dấu những điểm đã qua
15
16  //Hàm cấp phát bộ nhớ
17  void allocate(int n);
18  //Hàm nhập dữ liệu từ file
19  void importFile(const char* filename="bai2.in");
20  //Hàm xuất dữ liệu từ file ra màn hình, dùng để debug
21  void outScreen();
22  //Hàm tạo dữ liệu vào ngẫu nhiên cho chương trình
23  void makeData(int n, const char* filename="bai2.in");
24  //Khởi tạo điểm bắt đầu là thành phố 0
25  void init();
26
27  void findWay(int i)
28  {
29      int j;
30      for (j = 1; j < N; j++) //Các giá trị có thể nhận
31      {
32          if (Free[j] == 0) //Nếu chưa đi qua
33          {
34              X[i] = j; //Thu đi qua j
35              T[i] = T[i-1] + C[X[i-1]][j];
```

```

36         if (T[i] < MinCost) //Neu co kha nang di qua vs chi phi thap hon
37         {
38             if (i < N-1) //Neu di chua het
39             {
40                 Free[j] = 1; //Danh dau da di qua
41                 findWay(i + 1); //Di tiep
42                 Free[j] = 0; //Quay lai thanh pho, bo danh dau
43             }
44             else //Neu da di het
45                 if ((T[N-1] + C[X[N-1]][0]) < MinCost) //Neu ton it chi phi hon
46                 {
47                     int k;
48                     for (k = 1; k < N; k++) //Cap nhat cau hinh tot nhat
49                         BestWay[k] = X[k];
50                     MinCost = T[N-1] + C[X[N-1]][0];
51                 }
52             } //endif (T[i] < MinCost)
53         } //endif (Free[i] == 0)
54     } //endfor
55 } //end function
56
57
58 int main()
59 {
60     int i;
61     init();
62     makeData(CITY); //Tao du lieu dau vao
63     importFile();
64
65     findWay(1);
66
67     for (i=0;i<N;i++)
68         cout<<BestWay[i]<<"-->";
69     cout<<0<<endl;
70     cout<<MinCost;
71
72     cout<<endl;
73 }
74
75 void init()
76 {
77     Free[0]=1; //Da di qua thanh pho 0
78     X[0]=0; //Xuat phat tu thanh pho 0
79     T[0]=0;
80 }
81
82
83 void allocate(int n)
84 {
85     N=n;
86     C=new int*[n];
87     int i;
88     for (i=0;i<n;i++)
89         C[i]=new int[n];
90 }
91
92 void importFile(const char* filename)
93 {
94     ifstream inp(filename);
95     int n,i,j;
96     inp>>n;
97     allocate(n);
98     for (i=0;i<n;i++)

```

```

99         for (j=0;j<n;j++)
100             inp>>C[i][j];
101     }
102
103
104
105 void outScreen()
106 {
107     int i,j;
108     cout<<"\nXuat:"<<endl;
109     for (i=0;i<N;i++)
110     {
111         for (j=0;j<N;j++)
112         {
113             cout<<setw(4)<<C[i][j];
114         }
115         cout<<endl;
116     }
117     cout<<endl;
118 }
119
120 void makeData(int n,const char* filename)
121 {
122     int i,j,a,tmp[100][100];
123     ofstream outfile(filename);
124     outfile<<n<<endl;
125     srand(time(NULL));
126     for (i=0;i<n;i++)
127     {
128         for (j=i;j<n;j++)
129         {
130             if (i!=j)
131                 tmp[i][j]=tmp[j][i]=1+rand()%RANGE;
132             else
133                 tmp[i][j]=0;
134         }
135     }
136     for (i=0;i<n;i++)
137     {
138         for (j=0;j<n;j++)
139             outfile<<setw(4)<<tmp[i][j];
140         outfile<<endl;
141     }
142     outfile<<endl;
143     outfile.close();
144 }

```

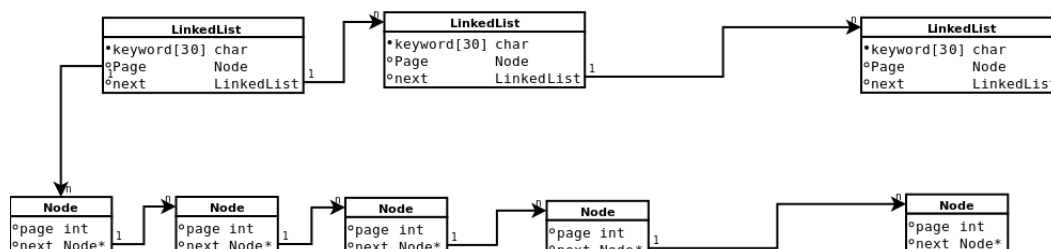
3 Bài toán đánh chỉ mục cho danh sách từ khóa

3.1 Bài toán

Có một file văn bản gồm danh sách các từ khóa và một file văn bản khác là tài liệu chứa các từ khóa đó. Viết chương trình lưu danh sách từ khóa và dòng xuất hiện của từ khóa đó trong tài liệu, cài đặt với các cấu trúc dữ liệu khác nhau: danh sách liên kết, bảng băm, cây nhị phân tìm kiếm.

3.2 Ý tưởng

Lưu các từ khóa trong các cấu trúc dữ liệu khác nhau - danh sách liên kết, bảng băm và cây nhị phân tìm kiếm. Với mỗi nút, ngoài mảng ký tự chứa từ khóa, ta còn lưu thêm một biến kiểu Node chính



Hình 15: Danh sách liên kết chứa từ khóa.

là *Header* của danh sách liên kết, mà mỗi phần tử của danh sách liên kết là một số nguyên, biểu thị cho vị trí xuất hiện của từ khóa.

Đọc lần lượt từng dòng của *tài liệu* cần đánh chỉ mục, với mỗi dòng, đọc được, ta lần lượt tìm kiếm các từ xuất hiện trong dòng đó xem có trùng với từ khóa được lưu trữ trong cấu trúc dữ liệu trước đó không, nếu trùng, thêm vị trí xuất hiện của từ khóa (số thứ tự dòng) vào danh sách liên kết có header là phần tử kiểu Node. Xem hình minh họa để hiểu rõ hơn chi tiết cài đặt.

3.3 Cài đặt với danh sách liên kết

```
1  #include <iostream>
2  #include <fstream>
3  #include <string.h>
4  #define MAX_LENGTH 1000
5  #define MAX_CHAR 50
6  using namespace std;
7
8
9  class Node
10 {
11     int page;
12     Node *next;
13
14     Node()
15     {
16         page = 0;
17         next = NULL;
18     }
19     //Them phan tu vao cuoi danh sach
20     void addNode(int page)
21     {
22         Node *tmp, *p = this;
23         tmp = new Node;
24         tmp->page = page;
25         while (p->next != NULL)
26             p = p->next;
27         p->next = tmp;
28     }
29     //Xuat danh sach lien ket tro boi this->next
30     int listNode()
```

```

31 {
32     Node *p;
33     int s = 0;
34     for (p = next; p != NULL; p = p->next)
35     {
36         cout<<p->page<<' ';
37         s++;
38     }
39     cout<<endl;
40     return s;
41 }
42
43 friend class List;
44 };
45
46
47 /*
48  Cac phuong thuc:
49  -Insert(): Them 1 tu khoa, neu da ton tai thi khong them vao nua.
50  -IndexKey(): Them chi muc cho tu khoa.
51  -ListIndex(): Hien thi danh sach chi muc cua tu khoa
52 */
53 class List
54 {
55     char key[MAX_CHAR];
56     Node P;
57     List *next;
58 public:
59     List()
60     {
61         bzero(key, MAX_CHAR);
62         next = NULL;
63     }
64     //Them tu khoa theo thu tu tu dien
65     // neu tu khoa da ton tai thi khong them vao nua
66     bool Insert(const char* keyword)
67     {
68         List *p = this;
69
70         //Vong lap tim vi tri chen theo thu tu tang dan
71         while (p->next != NULL)
72         {
73             if (strcasemp(p->next->key, keyword) >= 0)
74                 break;
75             p = p->next;
76         }
77         //Chen them tu khoa luu y truong hop p->next == NULL neu co gang
78         //truy xuat p->next->key se bi loi
79         if ((p->next == NULL) || (strcasemp(p->next->key, keyword) != 0))
80         {
81             List *tmp = new List;
82             strcpy(tmp->key, keyword);
83             tmp->next = p->next;
84             p->next = tmp;
85             return true;
86         }
87         return false;
88     }
89
90     //Danh chi muc cho tu khoa
91     bool IndexKey(const char* keyword, int page)
92     {
93         List *p;

```

```

94     //Tim trong danh sach tu khoa
95     for (p = this; p != NULL; p = p->next)
96         //Neu trung tu khoa trong danh sach
97         if (strcasecmp(p->key, keyword) == 0)
98         {
99             //Them page vao dslk tro boi p->next
100            p->P.addNode(page);
101            return true;
102        }
103    return false;
104 }
105
106 //Hien danh sach chi muc cua tu khoa
107 bool ListIndex(const char* keyword)
108 {
109     List *p;
110     //Tim vi tri xuat hien cua tu khoa
111     for (p = this; (p != NULL) && (strcasecmp(p->key, keyword) != 0); p = p->next);
112     //Neu den cuoi danh sach thi bao ko xuat hien tu khoa
113     if (p == NULL)
114         return false;
115     //Neu co tu khoa trong danh sach nhung ko co tu khoa trong tai lieu thi bao loi
116     if (p->P.next == NULL)
117         return false;
118     //Neu co danh sach chi muc thi in ra man hinh
119     p->P.listNode();
120     return true;
121 }
122
123 //Hien danh sach cac tu khoa duoc danh chi muc, dung de debug
124 int ListKeyWord()
125 {
126     List *p;
127     int n = 0;
128     cout<<"List of keyword: "<<endl;
129     //Hien thi cac tu khoa trong danh sach tro boi this->next
130     for (p = this->next; p != NULL; p = p->next)
131     {
132         cout<<p->key<<' ';
133         if (n%10 == 9)
134             cout<<endl;
135         n++;
136     }
137     cout<<endl;
138     return n;
139 }
140 };
141
142
143 //Chia sentence thanh cac word, thay doi gia tri con tro cua sentence
144 void split(char* &sentence, char* &word)
145 {
146     while (*sentence==' ')
147         sentence++;
148     word=sentence;
149     while ((*sentence!=' ')&&(*sentence!='\0'))
150         sentence++;
151     if (*sentence==' ')
152     {
153         *sentence='\0';
154         sentence++;
155     }
156 }

```

```

157 int main()
158 {
159     List a;
160     ifstream ListWord("bai3.in"); //Danh sach tu khoa
161     ifstream Document("document.in"); //File can danh chi muc
162     char *line = new char[MAX_LENGTH], *word, *tmp, KeyWord[MAX_CHAR];
163     int n = 1; //Bien dem so trang
164     cout<<"LINK LIST VERSION\n-----"<<endl;
165
166     //Them tu khoa vao danh sach lien ket
167     cout<<"Import keyword from \"bai3.in\"..."<<endl;
168     while (!ListWord.eof())
169     {
170         //Luu tu khoa vao bien tam
171         ListWord>>KeyWord;
172         //Chen tu khoa vao danh sach
173         a.Insert(KeyWord);
174     }
175     ListWord.close();
176     //Hien thi danh sach tu khoa ra man hinh
177     a.ListKeyWord();
178
179     //Danh chi muc cho tu khoa
180     cout<<"\nIndexing..."<<endl;
181     while (!Document.eof())
182     {
183         //Ghi dong dang doc vao line
184         Document.getline(line, MAX_LENGTH);
185         //Bien tam thay the line, neu dung line thang vao ham
186         //split() thi se bi loi ve sau do split nhan tham chieu
187         tmp = line;
188         while (strlen(tmp)>0)
189         {
190             //Chi cau thanh cac tu nho
191             split(tmp, word);
192             //Danh chi muc cho tu khoa
193             a.IndexKey(word, n);
194         }
195         n++;
196     }
197     delete [] line;
198     Document.close();
199
200     word = new char[MAX_CHAR];
201     do
202     {
203         cout<<"Enter keyword: ";
204         cin>>word;
205         //Neu nhap vao QUIT thi thoat chuong trinh
206         if (strcmp(word, "QUIT") == 0)
207             break;
208         //Neu tu khoa khong ton tai hoac khong co chi muc thi thong bao cho nguoi dung
209         if (!a.ListIndex(word))
210             cout<<"Keyword isn't in the index list or not found in document"<<endl;
211     } while (1);
212     delete [] word;
213
214     cout<<"\n-----\nExit program"<<endl;
215     return 0;
216 }
217
218
219

```


3.4 Cài đặt với bảng băm

```
1  #include <iostream>
2  #include <fstream>
3  #include <string.h>
4  #include <iomanip>
5  #define MAX_LENGTH 1000
6  #define MAX_CHAR 50
7
8  using namespace std;
9
10 //Luu danh sach chi muc
11 class Node
12 {
13     int page;
14     Node *next;
15
16     Node()
17     {
18         next = NULL;
19         page = 0;
20     }
21
22     //Them phan tu vao cuoi danh sach
23     void AddNode(int page)
24     {
25         Node *tmp, *p = this;
26         tmp = new Node;
27         tmp->page = page;
28         while (p->next != NULL)
29             p = p->next;
30         p->next = tmp;
31     }
32     //Xuat danh sach lien ket tro boi this->next
33     int ListNode()
34     {
35         Node *p;
36         int s = 0; //Bien luu so trang trong danh sach
37         for (p = next; p != NULL; p = p->next)
38         {
39             cout<<p->page<<' ' ;
40             s++;
41         }
42         cout<<endl;
43         return s;
44     }
45
46     friend class Cell;
47 };
48
49 //Luu cac tu khoa va con tro toi danh sach chi muc
50 class Cell
51 {
52     char key[MAX_CHAR]; //Mang luu tu khoa
53     Node P; //Header cua dslk tuu danh sach cac trang
54     Cell *next;
55
56     //Ham tao
57     Cell(const char* word=NULL)
58     {
59         bzero(key, MAX_CHAR);
60         if (word != NULL)
61             strcpy(key, word);
62         next = NULL;
```

```

63     }
64
65     //Them chi muc vao keyword trong Cell hien tai
66     void AddIndex(int page)
67     {
68         P.AddNode(page);
69     }
70
71     //In danh sach chi muc duoc tro boi P
72     int ListNode()
73     {
74         return P.ListNode();
75     }
76
77     friend class HashTable;
78 };
79
80 class HashTable
81 {
82     Cell **T; //Con tro toi dau moi danh sach keyword
83     int m; //Kich thuoc cua bang bam
84
85 public:
86     //Ham tao
87     HashTable(int size=100)
88     {
89         m = size;
90         T = new Cell*[m];
91         for (int i =0 ; i < m; i++)
92             T[i] = NULL;
93     }
94     //Ham bam
95     int HashFunc(const char* keyword)
96     {
97         int S=0;
98         for (int i = 0; i < strlen(keyword); i++)
99             S += tolower(keyword[i]);
100         return S % 100;
101     }
102
103     /*Chen phan tu vao bang bam
104     Cac keyword khong duoc phep trung
105     vi khong co theo tackiem tra tu khoa
106     truoc khi nhap vao bang bam*/
107     bool Insert(const char* keyword)
108     {
109         int k = HashFunc(keyword);
110         //Neu phan tu T[k] con rong thi tao moi va chen vao
111         if (T[k] == NULL)
112         {
113             T[k] = new Cell;
114             strcpy(T[k]->key, keyword);
115             return true;
116         }
117         //Neu khong thi chen vao dau dslik tro boi T[k]->next
118         else
119         {
120             Cell *tmp = new Cell;
121             strcpy(tmp->key, keyword);
122             tmp->next = T[k]->next;
123             T[k]->next = tmp;
124             return true;
125         }

```

```

126     }
127
128     //Tim kiem phan tu trong bang bam
129     bool IndexKey(const char* keyword, int page)
130     {
131         int k = HashFunc(keyword);
132         Cell *p;
133         //Tim tu khoa trong bang bam
134         for (p = T[k]; p != NULL; p = p->next)
135             //Neu co them trang vao danh sach chi muc
136             if (strcasecmp(p->key, keyword) == 0)
137             {
138                 p->AddIndex(page);
139                 return true;
140             }
141         return false;
142     }
143
144     //Hien danh sach chi muc cua tu khoa keyword
145     bool ListIndex(const char* keyword)
146     {
147         int k = HashFunc(keyword);
148         Cell *p;
149         //Tim tu khoa trong bang bam
150         for (p = T[k]; p != NULL; p = p->next)
151             //Neu co thi in ra danh sach chi muc
152             if (strcasecmp(p->key, keyword) == 0)
153             {
154                 if (p->ListNode())
155                     return true;
156                 return false;
157             }
158         return false;
159     }
160
161     /*Hien cac phan tu co cung ma bam
162     Dung de kiem tra, debug*/
163     int List(const char* keyword)
164     {
165         int k = HashFunc(keyword), n=0;
166         Cell *p;
167         for (p = T[k]; p != NULL; p = p->next)
168         {
169             cout<<p->key<<' ';
170             n++;
171         }
172         cout<<endl;
173         return n;
174     }
175 };
176
177     //Chia cau thanh cac tu bang nhau
178     void split(char* &sentence, char* &word)
179     {
180         while (*sentence==' ')
181             sentence++;
182         word=sentence;
183         while ((*sentence!=' ')&&(*sentence!='\0'))
184             sentence++;
185         if (*sentence==' ')
186         {
187             *sentence='\0';
188             sentence++;

```

```

189     }
190 }
191
192 int main()
193 {
194     HashTable Index;
195     ifstream input("bai3.in"); //Danh sach tu khoa
196     ifstream document("document.in"); //Tai lieu can danh chi muc
197     char *line = new char[MAX_LENGTH], *word, *tmp;
198     char KeyWord[MAX_CHAR];
199     int n = 1;
200     cout<<"HASH TABLE VERSION\n-----\n"<<endl;
201
202     //Nhap danh sach tu khoa vao bang bam
203     cout<<"Import keyword from \"bai3.in\"..."<<endl;
204     while (!input.eof())
205     {
206         input>>KeyWord;
207         Index.Insert(KeyWord);
208     }
209     input.close();
210
211     cout<<"Indexing..."<<endl;
212     //Danh chi muc cho tu khoa
213     while (!document.eof())
214     {
215         //Doc tung cau
216         document.getline(line,1000);
217         tmp = line;
218         while (strlen(tmp) > 0)
219         {
220             //Tach cau thanh cac tu
221             split(tmp, word);
222             Index.IndexKey(word,n);
223         }
224         n++;
225     }
226     document.close();
227
228     word = new char[MAX_CHAR];
229     do
230     {
231         cout<<"Enter keyword: ";
232         cin>>word;
233         if (strcmp(word, "QUIT") == 0)
234             break;
235         if (!Index.ListIndex(word))
236             cout<<"Keyword isn't in the index list or not found in document"<<endl;
237     }
238     while (1);
239     delete [] line;
240     delete[] word;
241     cout<<endl;
242     return 0;
243 }

```

3.5 Cài đặt với cây nhị phân tìm kiếm

```
1  #include <iostream>
2  #include <fstream>
3  #include <string.h>
4  #define MAX_CHAR 50
5  #define MAX_LENGTH 1000
6  using namespace std;
7
8  class Node
9  {
10     int page;
11     Node *next;
12
13     Node()
14     {
15         page = 0;
16         next = NULL;
17     }
18
19     //Them phan tu vao cuoi danh sach
20     void AddNode(int page)
21     {
22         Node *tmp, *p = this;
23         tmp = new Node;
24         tmp->page = page;
25         while (p->next != NULL)
26             p = p->next;
27         p->next = tmp;
28     }
29     //Xuat danh sach lien ket tro boi this->next
30     int ListNode()
31     {
32         Node *p;
33         int s = 0; //Bien luu so trang trong danh sach
34         for (p = next; p != NULL; p = p->next)
35         {
36             cout<<p->page<<' ';
37             s++;
38         }
39         cout<<endl;
40         return s;
41     }
42     friend class Tree;
43 };
44
45 class Tree
46 {
47     char key[MAX_CHAR];
48     Node P;
49     Tree *left;
50     Tree *right;
51 public:
52     //Ham tao
53     Tree()
54     {
55         bzero(key, MAX_CHAR);
56         left = NULL;
57         right = NULL;
58     }
59
60     //Chen tu khoa vao cay
61     bool Insert(Tree *&root, const char* keyword)
62     {
```

```

63      //Loai bo trung tu khoa
64      if ((root != NULL) && (strcasecmp(root->key, keyword) ==0))
65          return false;
66      //Neu cay hien thoi rong thi tao nut moi va gan vao cay
67      if (root == NULL)
68      {
69          Tree *tmp = new Tree;
70          strcpy(tmp->key, keyword);
71          root = tmp;
72      }
73      //Neu tu khoa be hon thi chen vao cay con ben trai
74      else if (strcasecmp(keyword, root->key) < 0)
75          Insert(root->left, keyword);
76      //Nguoc lai chen vao cay con ben phai
77      else
78          Insert(root->right, keyword);
79      return true;
80  }
81  //Danh chi muc
82  bool IndexKey(Tree *root, const char* keyword, int page)
83  {
84      int a = 0;
85      if (root == NULL)
86          return false;
87      //neu nut chua tu khoa, co gia tri bang keyword thi them page
88      //vao danh sach tro boi root->P (root->P)
89      if (strcasecmp(keyword, root->key) == 0)
90      {
91          root->P.AddNode(page);
92      }
93      //Neu tu khoa be hon xau trong nut hien tai thi tim trong cay con trai
94      else if (strcasecmp(keyword, root->key) < 0)
95          a += IndexKey(root->left, keyword, page);
96      //Nguoc lai tim trong cay con phai
97      else
98          a +=IndexKey(root->right, keyword, page);
99      return a;
100  }
101
102  //Hien thi danh sach tu khoa
103  bool ListIndex(Tree *root,const char* keyword)
104  {
105      if (root == NULL)
106          return false;
107      //Neu nut chua tu khoa
108      if (strcasecmp(keyword, root->key) == 0)
109      {
110          //Neu tu khoa do da duoc danh chi muc
111          if (root->P.ListNode() > 0)
112              return true;
113          return false;
114      }
115      //Tim trong cay con trai va cay con phai
116      if (ListIndex(root->left, keyword) + ListIndex(root->right, keyword) == 0)
117          return false;
118      return true;
119  }
120
121  //Duyet cay theo thu tu giua
122  void Inorder(Tree *root)
123  {
124      if (root == NULL)
125          return;

```

```

126         Inorder(root->left);
127         cout<<root->key<<' ';
128         Inorder(root->right);
129     }
130 };
131
132
133 void split(char* &sentence, char* &word)
134 {
135     while (*sentence==' ')
136         sentence++;
137     word=sentence;
138     while ((*sentence!=' ')&&(*sentence!='\0'))
139         sentence++;
140     if (*sentence==' ')
141     {
142         *sentence='\0';
143         sentence++;
144     }
145 }
146
147 int main()
148 {
149     Tree *a = NULL;
150     ifstream inp("bai3.in"); //File luu danh sach cac tu.
151     ifstream doc("document.in"); //File can danh chi muc.
152     char *line = new char[MAX_LENGTH], *word, *tmp, KeyWord[MAX_CHAR];
153     int n = 1;
154     //Nhap danh sach tu khoa
155     cout<<"BINARY SEARCH TREE VERSION\n-----"<<endl;
156     cout<<"Import keyword from \"bai3.in\"..."<<endl;
157     while (!inp.eof())
158     {
159         inp>>KeyWord;
160         a->Insert(a, KeyWord);
161     }
162     cout<<"List of keyword: "<<endl;
163     //Hien thi danh sach tu khoa
164     a->Inorder(a);
165     cout<<endl;
166     inp.close();
167
168     //Danh chi muc cho tu khoa
169     cout<<"\nIndexing..."<<endl;
170     while (!doc.eof())
171     {
172         //Doc file theo dong
173         doc.getline(line, MAX_LENGTH);
174         tmp = line;
175         while (strlen(tmp) > 0)
176         {
177             //Chia cau thanh cac tu
178             split(tmp, word);
179             a->IndexKey(a, word, n);
180         }
181         n++;
182     }
183     delete [] line;
184     doc.close();
185
186     word = new char[MAX_CHAR];
187
188     do

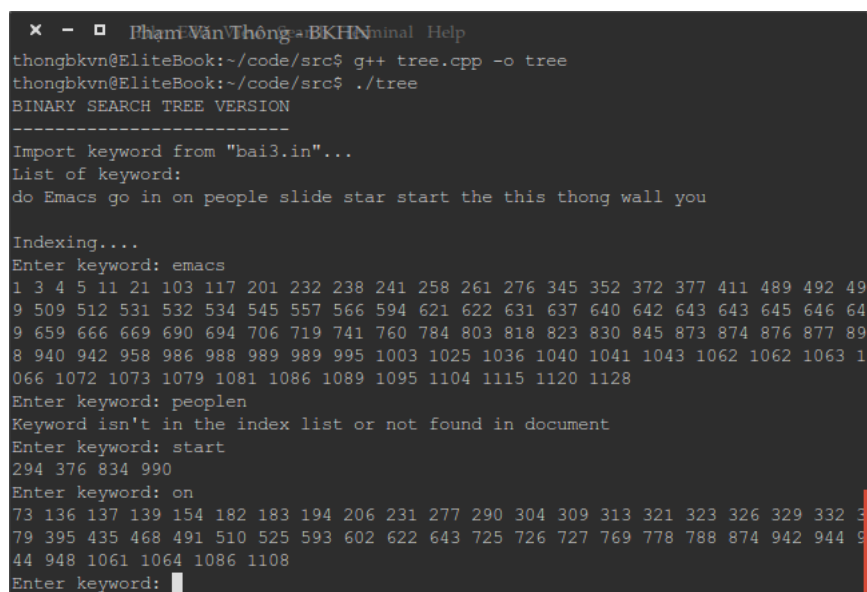
```

```

189     {
190         cout<<"Enter keyword: ";
191         cin>>word;
192         if (strcmp(word, "QUIT") == 0)
193             break;
194         if (!a->ListIndex(a, word))
195             cout<<"Keyword isn't in the index list or not found in document"<<endl;
196     }
197     while (1);
198
199
200     cout<<endl;
201     return 0;
202 }

```

3.6 Kết quả chạy thử



```

x - □ Phạm Văn Thông - BKHN: ninal Help
thongbkvn@EliteBook:~/code/src$ g++ tree.cpp -o tree
thongbkvn@EliteBook:~/code/src$ ./tree
BINARY SEARCH TREE VERSION
-----
Import keyword from "bai3.in"...
List of keyword:
do Emacs go in on people slide star start the this thong wall you

Indexing....
Enter keyword: emacs
1 3 4 5 11 21 103 117 201 232 238 241 258 261 276 345 352 372 377 411 489 492 49
9 509 512 531 532 534 545 557 566 594 621 622 631 637 640 642 643 643 645 646 64
9 659 666 669 690 694 706 719 741 760 784 803 818 823 830 845 873 874 876 877 89
8 940 942 958 986 988 989 989 995 1003 1025 1036 1040 1041 1043 1062 1062 1063 1
066 1072 1073 1079 1081 1086 1089 1095 1104 1115 1120 1128
Enter keyword: peoplen
Keyword isn't in the index list or not found in document
Enter keyword: start
294 376 834 990
Enter keyword: on
73 136 137 139 154 182 183 194 206 231 277 290 304 309 313 321 323 326 329 332 3
79 395 435 468 491 510 525 593 602 622 643 725 726 727 769 778 788 874 942 944 9
44 948 1061 1064 1086 1108
Enter keyword: 

```

Hình 16: Kết quả chạy thử