

MỤC LỤC

LỜI MỞ ĐẦU.....	1
CHƯƠNG I: BÀI TOÁN SẮP XẾP.....	2
CHƯƠNG II: BÀI TOÁN QUAY LUI.....	25
CHƯƠNG III: BÀI TOÁN ĐÁNH CHỈ MỤC TỪ KHÓA.....	42
KẾT LUẬN.....	45
PHỤ LỤC.....	46

LỜI MỞ ĐẦU

Bài báo cáo được chia làm 3 phần trình bày các thuật toán sắp xếp, thuật toán quay lui và đánh chỉ mục từ khóa đoạn văn bản.

Trong đó phần thứ nhất trình bày về ý tưởng, cài đặt bài toán và đánh giá độ phức tạp của các thuật toán sắp xếp cơ bản như: QuickSort, HeapSort, BubbleSort, InsertionSort, MergeSort. Phần thứ 2 trình bày 3 bài toán sử dụng giải thuật quay lui như: Bài toán xếp hậu, Bài toán mã đi tuần và Bài toán Người du lịch sử dụng kỹ thuật nhánh cận. Phần thứ 3 trình bày về các cách cài đặt khác nhau của bài toán đánh chỉ mục từ khóa đoạn văn bản dựa trên các cấu trúc dữ liệu khác nhau như: Bảng băm, Cây nhị phân tìm kiếm và Danh sách liên kết.

Ngôn ngữ được sử dụng trong các cài đặt là ngôn ngữ lập trình C/C++.

CHƯƠNG I: CÁC BÀI TOÁN SẮP XẾP

I. Ý tưởng và cài đặt các thuật toán

Bài toán sắp xếp là quá trình bố trí lại vị trí các phần tử của một tập đối tượng nào đó theo một thứ tự nhất định (VD: Tăng dần hoặc giảm dần với một dãy số, Sắp xếp các chữ cái theo thứ tự từ điển,...). Yêu cầu về sắp xếp thường xuyên xuất hiện trong cuộc sống cũng như trong học tập với nhiều mục đích khác nhau. Tuy nhiên ở đây ta nghiên cứu bài toán sắp xếp các phần tử trong một dãy số theo thứ tự tăng dần.

1. Thuật toán sắp xếp nổi bọt (BubbleSort)

Trong thuật toán sắp xếp nổi bọt các dãy khóa sẽ duyệt từ cuối lên đầu dãy (từ $a[n-1]$ đến $a[0]$), nếu gặp 2 khóa kề nhau bị ngược thứ tự thì đổi chỗ 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 $a[1]$ tới $a[n-1]$

Cài đặt:

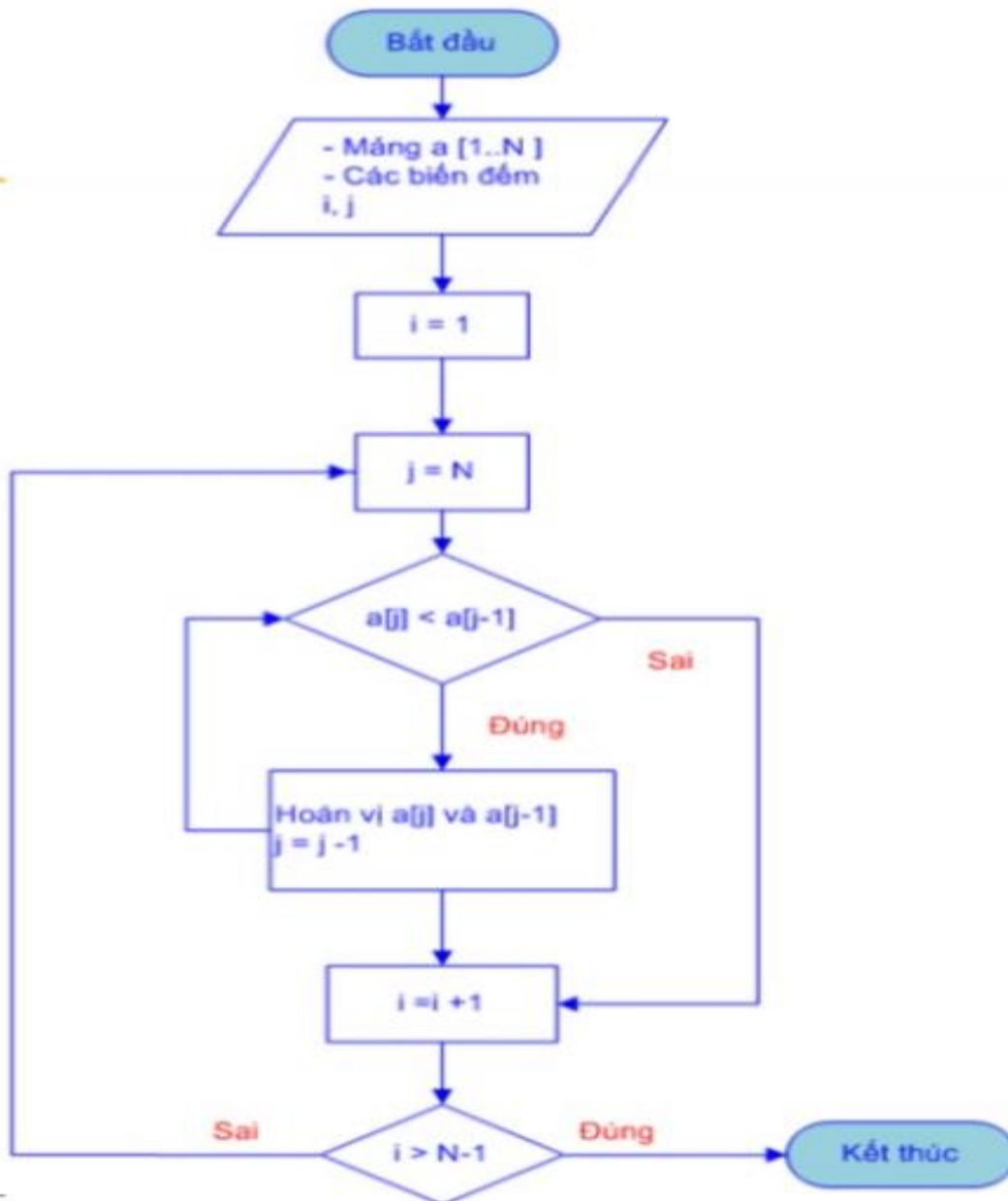
```
void Bubblesort (int A[], int n){  
    for (int i=1;i<n;i++)  
        for (int j=n-1;j>=i;j--)  
            if (A[j]<A[j-1])  
                Swap (A[j],A[j-1]);  
}
```

Đánh giá độ phức tạp thuật toán: Chọn phép toán so sánh $a[i] < a[j]$ 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). Vậy số lần thực hiện phép so sánh là :

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

=> Thời gian tính của thuật toán BubbleSort là $O(n^2)$

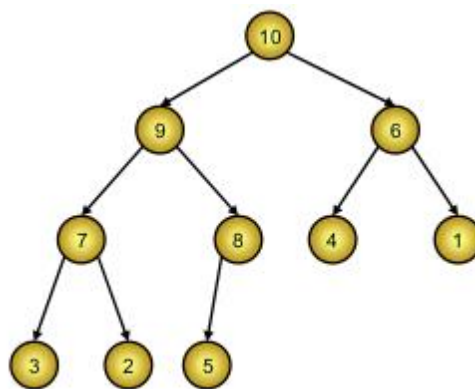
Lưu đồ thuật toán:



2. Thuật toán sắp xếp vun đống (HeapSort)

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

Đống - Heap: Đống 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 vun đống ta coi quan hệ “ưu tiên hay bằng” là quan hệ “lớn hơn hay bằng”: \geq



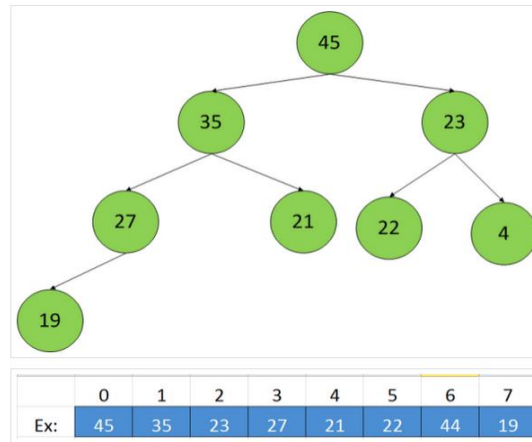
Hình 1.2.1: Đống

Vun đống:

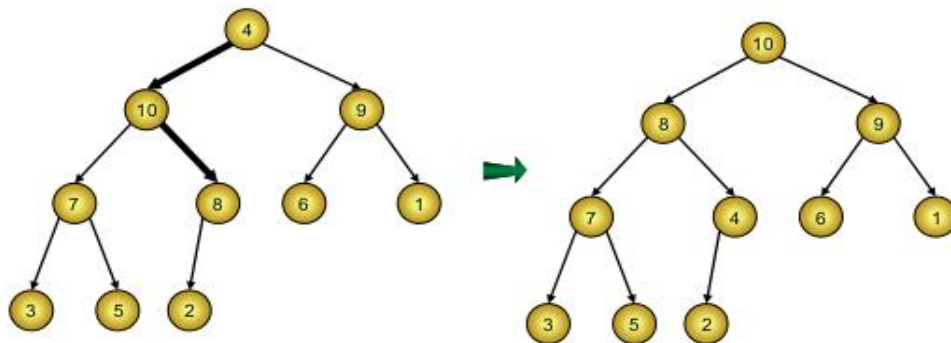
Để biểu diễn dãy khóa $a[0 \dots n-1]$ thành một cây nhị phân hoàn chỉnh. Giá trị $a[i]$ lưu trong nút thứ i (i bắt đầu từ 0). Nút con trái của nút thứ i là $a[2i+1]$ và con phải của nút thứ i là $a[2i+2]$. Nút cha của nút thứ j là $(j-1)/2$.

Vì cây nhị phân gồm một nút thì hiển nhiên là đống, nên để vun một nhánh cây gốc r thành đống, ta có thể coi 2 nhánh con của nó là đống rồi và thực hiện thuật toán vun đống từ dưới lên đối với cây. Gọi h là chiều cao của cây, nút ở mức h (nút lá) đã là một đống, ta vun lên những nút ở mức $h-1$ cũng là gốc của đống... cứ như vậy cho tới mức 1 cũng là gốc của đống.

Thuật toán vun đống đối với gốc r khi hai nhánh con đã là đống: 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 .



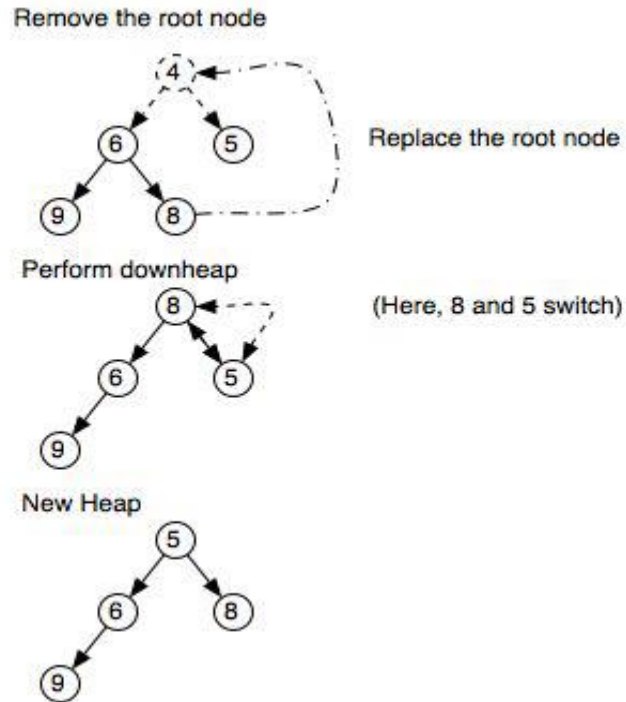
Hình 1.2.2: Biểu diễn đồng bằng mảng



Hình 1.2.3: Vun đồng

Tư tưởng của HeapSort:

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



Hình 1.2.4: Đảo giá trị của $a[0]$ và $a[n-1]$ rồi xét lại

Đánh giá độ phức tạp thuật toán: Thuật toán sắp xếp HeapSort có độ phức tạp $n \log(n)$ trong mọi trường hợp.

Cài đặt:

```
void Heap_Sort(int A[], int n) {
    Build_Max_Heap(A, n);
    for(int i=n; i>=2; i--) {
        Swap(A[i],A[1]);
        Max_Heapify(A, 1, i-1);
    }
    return;
}
```

```

void Max_Heapify(int A[], int i, int n) {
    int largest;
    int left=2*i;
    int right=2*i+1;
    if (left<=n&&A[left]>A[i]) {
        largest=left;
    }
    else {
        largest=i;
    }
    if (right<=n&&A[right]>A[largest]) {
        largest=right;
    }
    if(largest!=i) {
        Swap(A[largest],A[i]);
        Max_Heapify(A,largest,n);
    }
    return;
}

void Build_Max_Heap(int A[], int n) {
    for (int i=n/2; i>=1; i--) {
        Max_Heapify(A, i, n);
    }
    return;
}

```


3. Thuật toán sắp xếp nhanh (QuickSort)

Thuật toán QuickSort 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, QuickSort cũng có thể sắp xếp được và chưa có một thuật toán tổng quát nào nhanh hơn QuickSort về mặt tốc độ trung bình.

Quicksort là một thuật toán theo tư tưởng chia để trị, nó dựa trên thủ tục phân chia như sau: để chia một dãy ta chọn một phần tử được gọi là "chốt" (pivot), chuyển tất cả các phần tử nhỏ hơn chốt về trước chốt, chuyển tất cả các phần tử lớn hơn chốt về sau nó (nếu sắp xếp theo dãy theo thứ tự tăng dần), nếu sắp xếp dãy theo thứ tự giảm dần ta chuyển tất cả các phần tử nhỏ hơn chốt về bên phải chốt và lớn hơn chốt về bên trái chốt. Thủ tục này có thể thực hiện trong thời gian tuyến tính. Tiếp tục phân chia các dãy con đó như trên cho đến khi các dãy con chỉ còn một phần tử.

Cài đặt:

```
int Partition(int A[], int left, int right) {
    int i=left, j=right, pivot=A[left];
    while(i<j) {
        while(i<=right&&A[i]<=pivot) {
            i++;
        }
        while(j>=left&&A[j]>pivot) {
            j--;
        }
        if (i<j) {
            Swap(A[i],A[j]);
        }
    }
    Swap(A[left],A[j]);
    return j;
}
```

```

void Quick_Sort(int A[], int left, int right) {
    if (left < right) {
        int p = Partition(A, left, right);
        Quick_Sort(A, left, p-1);
        Quick_Sort(A, p+1, right);
    }
    return;
}

```

Đánh giá độ phức tạp thuật toán: QuickSort có chi phí thời gian thực hiện phụ thuộc vào trạng thái của dãy khóa đầu vào.

- Trong trường hợp tốt nhất, ở mỗi bước luôn chọn đúng trung vị của dãy khóa, gọi thời gian thực hiện giải thuật là $T(n)$.

$$\begin{aligned}
 T(n) &= 2T\left(\frac{n}{2}\right) + Cn \quad (C \text{ là hằng số}) \\
 \Rightarrow T(n) &= 2\left(2T\left(\frac{n}{4}\right) + C\frac{n}{2}\right) + Cn \\
 \Rightarrow T(n) &= 2^2T\left(\frac{n}{4}\right) + 2Cn \\
 \Rightarrow T(n) &= \dots\dots\dots \\
 \Rightarrow T(n) &= 2^kT\left(\frac{n}{2^k}\right) + kCn
 \end{aligned}$$

Khi $n=2^k \Rightarrow k = \log_2 n$. Thay $k = \log_2 n$ vào công thức trên ta được:

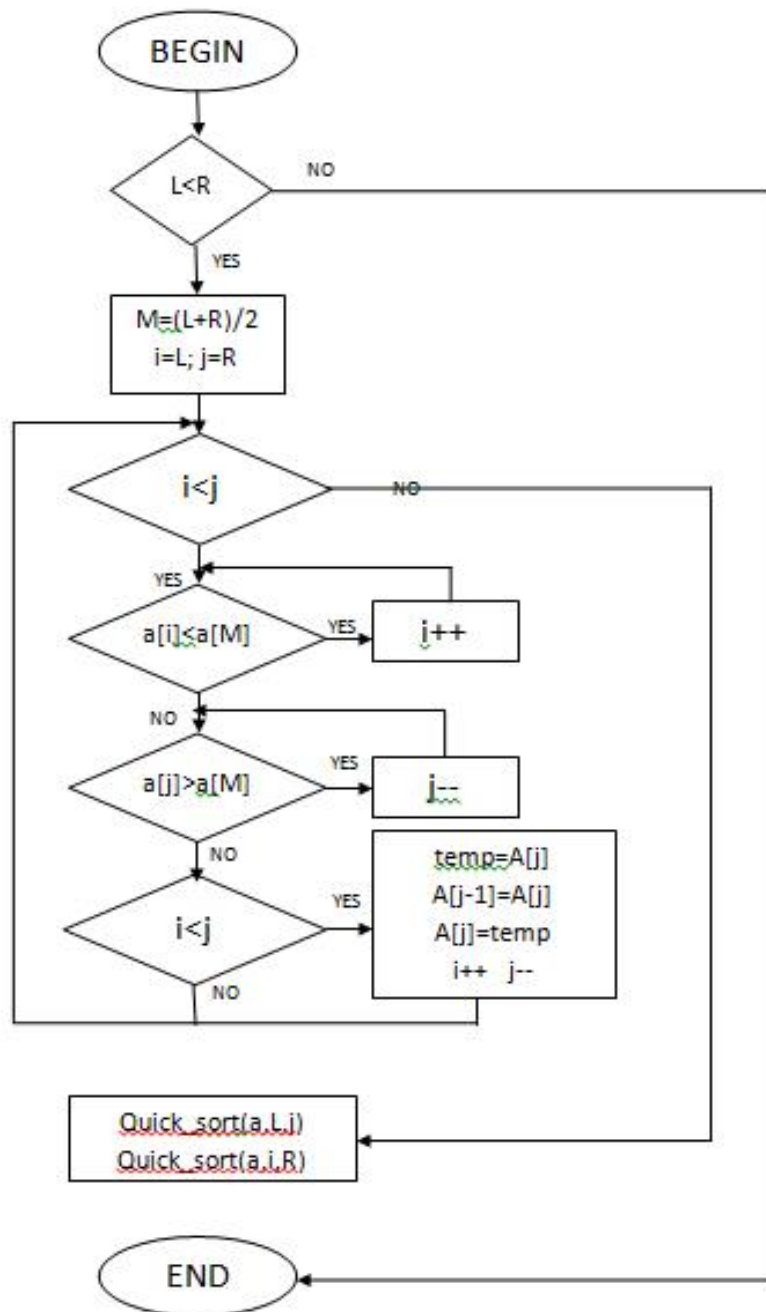
$$\begin{aligned}
 T(n) &= 2^{\log_2 n} T(1) + \log_2 n \cdot Cn \\
 \Rightarrow T(n) &= nT(1) + Cn \log_2 n = \theta(n \log(n))
 \end{aligned}$$

- Trong trường hợp xấu nhất, ở mỗi bước ta luôn chọn đúng phần tử đầu dãy hoặc cuối dãy, số lần thực hiện vòng lặp $i \leq j$ ở lần đầu tiên là $n-1$, tiếp theo dãy được chia thành 2 dãy gồm dãy có 1 phần tử, mất $T(1)$ và dãy còn lại $n-1$ phần tử. Tương tự ta có:

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

- Trong trường hợp trung bình, tương tự như trường hợp tốt nhất ta tính được độ phức tạp là $\theta(n\log(n))$.

Lưu đồ thuật toán:



4. Thuật toán sắp xếp chèn (InsertionSort)

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

Cài đặt:

```
void Intertion_Sort(int A[], int n) {
    for(int i=1; i<n; i++) {
        int last=A[i];
        int j=i;
        while(j>0&&last<A[j-1]) {
            A[j]=A[j-1];
            j--;
        }
        A[j]=last;
    }
}
```

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

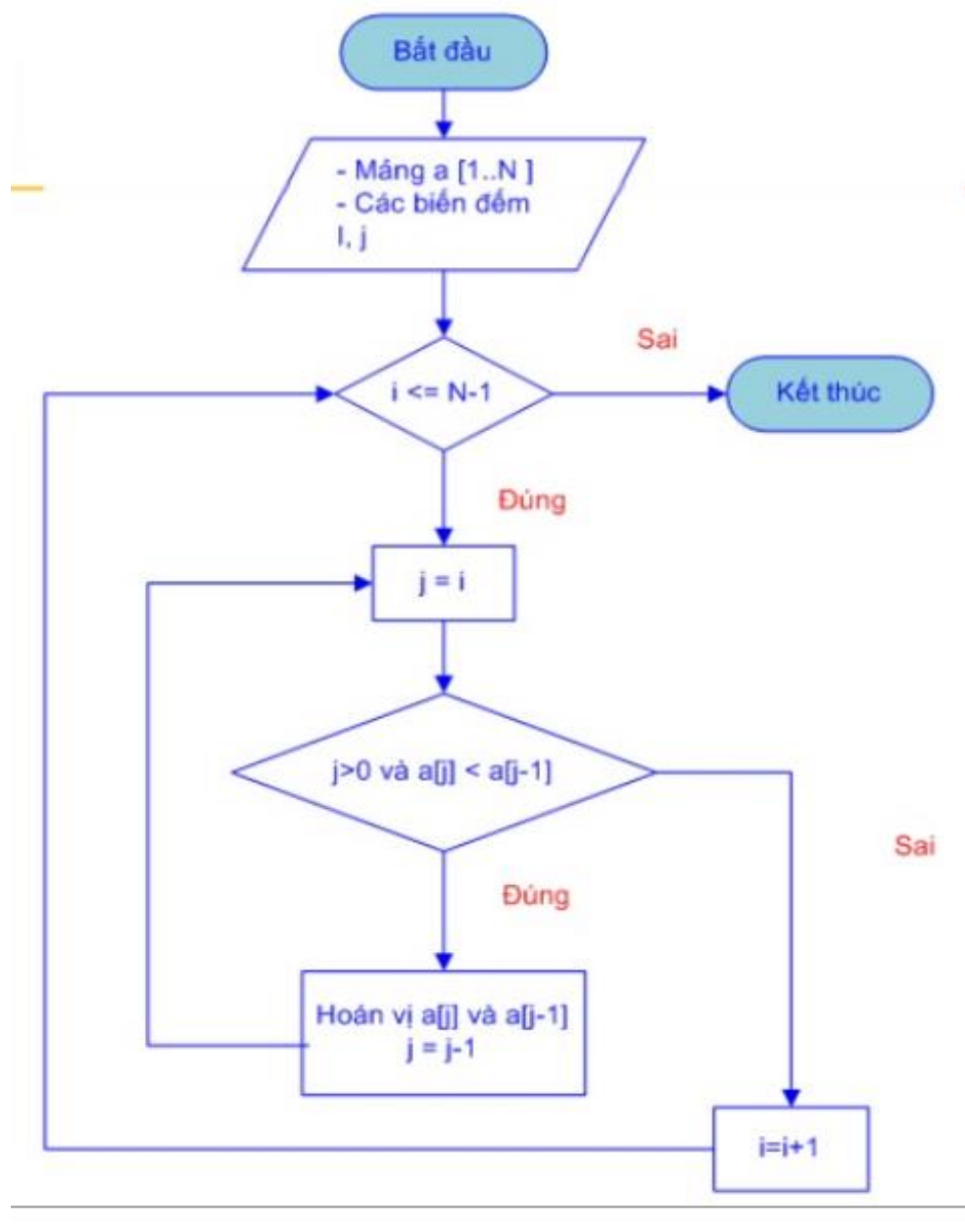
- Trường hợp tốt nhất, ứng với dãy khóa đã sắp xếp rồi, mỗi lượt chỉ cần thực hiện một phép so sánh \Rightarrow tổng số phép so sánh là $n-1$
 \Rightarrow Thời gian thực hiện thuật toán InsertionSort trong TH tốt nhất là $\theta(n)$.
- Trường hợp tồi nhất ứng với dãy khóa đã có thứ tự ngược lại, ở lượt sắp xếp thứ i cần thực hiện $i-1$ phép so sánh. Số phép so sánh cần thực hiện là:
$$(n-1)+(n-2)+\dots+1 = \frac{n(n-1)}{2}$$

 \Rightarrow Thời gian thực hiện thuật toán InsertionSort trong TH tồi nhất là $\theta(n^2)$.

- Trường hợp các khóa xuất hiện ngẫu nhiên, ở lượt sắp xếp thứ i cần thực hiện $\frac{i}{2}$ phép so sánh. Số phép so sánh cần thực hiện là:

$$\frac{1}{2} + \frac{2}{2} + \dots + \frac{n}{2} = \frac{n(n-1)}{4}$$
 \Rightarrow Thời gian thực hiện thuật toán InsertionSort trong TH trung bình là $\theta(n^2)$.

Lưu đồ thuật toán:



5. Thuật toán sắp xếp trộn (MergeSort)

Ý tưởng: Sắp xếp trộn (merge sort) cùng với sắp xếp nhanh là hai thuật toán sắp xếp dựa vào tư tưởng "chia để trị" (divide and conquer). Giả sử dãy $a[1], \dots, a[i], \dots, a[n]$ có hai đoạn đã được sắp xếp không giảm là $a[1], \dots, a[i]$ và $a[i+1], \dots, a[n]$ ta tiến hành tạo ra dãy mới bằng cách lần lượt lấy hai phần tử đầu tiên của mỗi đoạn và so sánh với nhau. Phần tử nào nhỏ hơn thì lấy ra dãy mới và bỏ nó ra khỏi đoạn đang xét. Cứ tiến hành như vậy cho tới khi một dãy bị vét hết (không còn phần tử nào). Lấy toàn bộ phần tử còn lại của đoạn không xét hết nối vào sau dãy đích. Như vậy dãy mới là một dãy đã được sắp xếp. Trong mọi trường hợp, có thể coi dãy gồm duy nhất 1 phần tử đã được sắp xếp. Lợi dụng việc này, ta phân chia một dãy ra thành nhiều dãy con chỉ gồm 1 phần tử rồi lần lượt hòa nhập từng đôi một với nhau.

Cài đặt:

```
void Tron(int A[],int first, int mid, int last) {
    int i1=first, i2=mid;
    int j1=mid+1, j2=last;
    int k=i1, tA[100000];
    for(;(i1<=i2)&&(j1<=j2);) {
        if(A[i1]<A[j1]) {
            tA[k]=A[i1];
            i1++;
        }
        else {
            tA[k]=A[j1];
            j1++;
        }
        k++;
    }
    for(;i1<=i2;) {
        tA[k]=A[i1];
```

```

        i1++;
        k++;
    }
    for(;j1<=j2;) {
        tA[k]=A[j1];
        j1++;
        k++;
    }
    for (int i=first;i<=last;i++) {
        A[i]=tA[i];
    }
    return;
}

void Sapxep(int A[],int first, int last) {
    int mid;
    if (first<last) {
        mid=(first+last)/2;
        Sapxep(A, first, mid);
        Sapxep(A, mid+1, last);
        Tron(A, first, mid, last);
    }
    return;
}

```

Đánh giá độ phức tạp thuật toán:

Sắp xếp mảng a với n phần tử, ta có:

$$T(n)=T(k) + T(n-k) + Cn \quad (C \text{ là hằng số})$$

Ở đây ta xét trường hợp tốt nhất

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

$$\Rightarrow T(n) = 2\left(2T\left(\frac{n}{4}\right) + C\frac{n}{2}\right) + Cn$$

$$\Rightarrow T(n) = 2^2T\left(\frac{n}{4}\right) + 2Cn$$

$$\Rightarrow T(n) = \dots\dots\dots$$

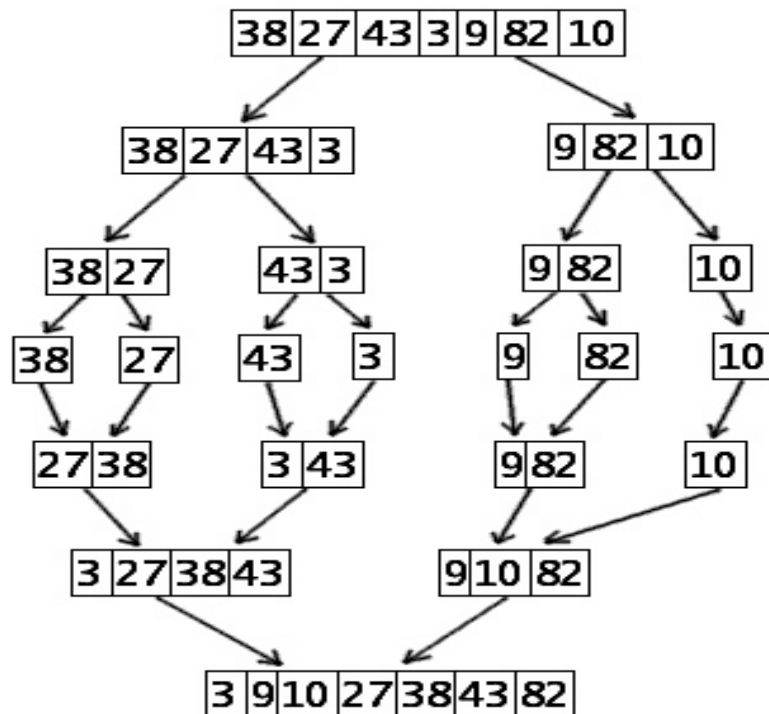
$$\Rightarrow T(n) = 2^kT\left(\frac{n}{2^k}\right) + kCn$$

Khi $n=2^k \Rightarrow k = \log_2 n$. Thay $k = \log_2 n$ vào công thức trên ta được:

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

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

Vậy thời gian thực hiện thuật toán MergeSort trong TH tốt nhất là $\theta(n \log(n))$.



Hình 1.5.1: Minh họa thuật toán sắp xếp trộn

II. Kiểm thử và so sánh thời gian thực tế các thuật toán

1. Kiểm thử

```
E:\Code\DA1.1\Sapxepnoibot.exe
-----BUBBLE SORT VERSION-----
Nhap n:10

Day so ban dau la:
41 467 334 500 169 724 478 358 962 464

Ket qua sau khi sap xep tang dan la:
41 169 334 358 464 467 478 500 724 962

Thoi gian thuc hien thuat toan sap xep noi bot la:0.00000
Nhap n:100

Day so ban dau la:
705 145 281 827 961 491 995 942 827 436 391 604 902 153 292 382 421 716 718 895 447 726 771 538
869 912 667 299 35 894 703 811 322 333 673 664 141 711 253 868 547 644 662 757 37 859 723 741
529 778 316 35 190 842 288 106 40 942 264 648 446 805 890 729 370 350 6 101 393 548 629 623
84 954 756 840 966 376 931 308 944 439 626 323 537 538 118 82 929 541 833 115 639 658 704 930
977 306 673 386

Ket qua sau khi sap xep tang dan la:
6 35 35 37 40 82 84 101 106 115 118 141 145 153 190 253 264 281 288 292 299 306 308 316
322 323 333 350 370 376 382 386 391 393 421 436 439 446 447 491 529 537 538 538 541 547 548 604
623 626 629 639 644 648 658 662 664 667 673 673 703 704 705 711 716 718 723 726 729 741 756 757
771 778 805 811 827 827 833 840 842 859 868 869 890 894 895 902 912 929 930 931 942 942 944 954
961 966 977 995

Thoi gian thuc hien thuat toan sap xep noi bot la:0.00000
Nhap n:
```

Hình 2.1.1: Kết quả chạy thử BubbleSort với mảng n=10 và n=100 phần tử

```
E:\Code\DA1.1\Full_Heap_Sort.exe
-----HEAP SORT VERSION-----
Nhap n:10

Day so ban dau la:
33 35 166 900 737 356 198 678 322 896

Ket qua sau khi sap xep tang dan la:
33 35 166 198 322 356 678 737 896 900

Thoi gian thuc hien thuat toan heap sort la:0.00000
Nhap n:100

Day so ban dau la:
577 481 737 419 737 483 931 678 707 292 647 260 806 129 292 70 5 260 262 423 327 198 419 258
837 448 67 675 643 678 7 259 578 613 257 544 805 519 581 708 963 996 902 997 581 547 515 517
897 770 4 963 678 802 288 386 832 742 832 96 806 229 514 577 802 998 646 357 833 452 165 327
516 482 324 576 870 192 611 708 544 679 34 35 417 34 742 34 385 133 737 387 519 962 160 706
641 258 929 866

Ket qua sau khi sap xep tang dan la:
4 5 7 34 34 34 35 67 70 96 129 133 160 165 192 198 229 257 258 258 259 260 260 262
288 292 292 324 327 327 357 385 386 387 417 419 419 423 448 452 481 482 483 514 515 516 517 519
519 544 544 547 576 577 577 578 581 581 611 613 641 643 646 647 675 678 678 678 679 706 707 708
708 737 737 737 742 742 770 802 802 805 806 806 832 832 833 837 866 870 897 902 929 931 962 963
963 996 997 998

Thoi gian thuc hien thuat toan heap sort la:0.00000
Nhap n:
```

Hình 2.1.2: Kết quả chạy thử HeapSort với mảng n=10 và n=100 phần tử

```

E:\Code\DA1.1\Sapxepchen.exe
-----INSERTION SORT VERSION-----
Nhap n:10

Day so ban dau la:
 33  35 166 900 737 356 198 678 322 896

Ket qua sau khi sap xep tang dan la:
 33  35 166 198 322 356 678 737 896 900

Thoi gian thuc hien thuat toan sap xep chen la:0.00000
Nhap n:100

Day so ban dau la:
577 481 737 419 737 483 931 678 707 292 647 260 806 129 292 70 5 260 262 423 327 198 419 258
837 448 67 675 643 678 7 259 578 613 257 544 805 519 581 708 963 996 902 997 581 547 515 517
897 770 4 963 678 802 288 386 832 742 832 96 806 229 514 577 802 998 646 357 833 452 165 327
516 482 324 576 870 192 611 708 544 679 34 35 417 34 742 34 385 133 737 387 519 962 160 706
641 258 929 866

Ket qua sau khi sap xep tang dan la:
 4 5 7 34 34 34 35 67 70 96 129 133 160 165 192 198 229 257 258 258 259 260 260 262
288 292 292 324 327 327 357 385 386 387 417 419 419 423 448 452 481 482 483 514 515 516 517 519
519 544 544 547 576 577 577 578 581 581 611 613 641 643 646 647 675 678 678 678 679 706 707 708
708 737 737 737 742 742 770 802 802 805 806 806 832 832 833 837 866 870 897 902 929 931 962 963
963 996 997 998

Thoi gian thuc hien thuat toan sap xep chen la:0.00000
Nhap n:

```

Hình 2.1.3: Kết quả chạy thử InsertionSort với mảng n=10 và n=100 phần tử

```

E:\Code\DA1.1\fullSapxeptron.exe
-----MERGE SORT VERSION-----
Nhap n:10

Day so ban dau la:
 41 485 340 526 188 739 489 387 988 488

Ket qua sau khi sap xep tang dan la:
 41 188 340 387 485 488 489 526 739 988

Thoi gian thuc hien thuat toan sap xep tron la:0.00000
Nhap n:100

Day so ban dau la:
710 173 304 843 970 491 997 953 831 441 423 618 905 153 292 394 438 734 737 914 452 747 785 549
870 931 692 325 52 903 731 834 353 363 690 668 156 718 281 874 572 671 694 789 57 871 731 750
556 778 328 38 212 843 288 136 49 950 283 670 473 828 905 735 394 365 21 132 417 551 648 635
108 973 774 851 970 383 944 334 960 471 650 334 542 559 134 84 951 557 837 146 643 687 726 939
990 308 704 408

Ket qua sau khi sap xep tang dan la:
 21 38 49 52 57 84 108 132 134 136 146 153 156 173 212 281 283 288 292 304 308 325 328 334
334 353 363 365 383 394 394 408 417 423 438 441 452 471 473 491 542 549 551 556 557 559 572 618
635 643 648 650 668 670 671 687 690 692 694 704 710 718 726 731 731 734 735 737 747 750 774 778
785 789 828 831 834 837 843 843 851 870 871 874 903 905 905 914 931 939 944 950 951 953 960 970
970 973 990 997

Thoi gian thuc hien thuat toan sap xep tron la:0.00000
Nhap n:

```

Hình 2.1.4: Kết quả chạy thử MergeSort với mảng n=10 và n=100 phần tử

```

E:\Code\DA1.1\FullQuick_Sort.exe
-----QUICK SORT VERSION-----
Nhap n:10

Day so ban dau la:
  41  485  340  526  188  739  489  387  988  488

Ket qua sau khi sap xep tang dan la:
  41  188  340  387  485  488  489  526  739  988

Thoi gian thuc hien thuat toan quick sort la:0.00000
Nhap n:100

Day so ban dau la:
  710  173  304  843  970  491  997  953  831  441  423  618  905  153  292  394  438  734  737  914  452  747  785  549
  870  931  692  325  52  903  731  834  353  363  690  668  156  718  281  874  572  671  694  789  57  871  731  750
  556  778  328  38  212  843  288  136  49  950  283  670  473  828  905  735  394  365  21  132  417  551  648  635
  108  973  774  851  970  383  944  334  960  471  650  334  542  559  134  84  951  557  837  146  643  687  726  939
  990  308  704  408

Ket qua sau khi sap xep tang dan la:
  21  38  49  52  57  84  108  132  134  136  146  153  156  173  212  281  283  288  292  304  308  325  328  334
  334  353  363  365  383  394  394  408  417  423  438  441  452  471  473  491  542  549  551  556  557  559  572  618
  635  643  648  650  668  670  671  687  690  692  694  704  710  718  726  731  731  734  735  737  747  750  774  778
  785  789  828  831  834  837  843  843  851  870  871  874  903  905  905  914  931  939  944  950  951  953  960  970

Thoi gian thuc hien thuat toan quick sort la:0.00000
Nhap n:

```

Hình 2.1.5: Kết quả chạy thử QuickSort với mảng n=10 và n=100 phần tử

2.So sánh thời gian tính thực tế các thuật toán

```

E:\Code\DA1.1\Sapxepnoiobot.exe
-----BUBBLE SORT VERSION-----
Nhap n:100

Thoi gian thuc hien thuat toan sap xep noi bot la:0.00000
Nhap n:1000

Thoi gian thuc hien thuat toan sap xep noi bot la:0.01600
Nhap n:2000

Thoi gian thuc hien thuat toan sap xep noi bot la:0.00000
Nhap n:4000

Thoi gian thuc hien thuat toan sap xep noi bot la:0.04700
Nhap n:8000

Thoi gian thuc hien thuat toan sap xep noi bot la:0.22400
Nhap n:16000

Thoi gian thuc hien thuat toan sap xep noi bot la:0.98500
Nhap n:32000

Thoi gian thuc hien thuat toan sap xep noi bot la:4.23800
Nhap n:64000

Thoi gian thuc hien thuat toan sap xep noi bot la:17.70400
Nhap n:100000

Thoi gian thuc hien thuat toan sap xep noi bot la:44.68800
Nhap n:

```

Hình 2.2.1: Thời gian chạy thực tế thuật toán BubbleSort với các giá trị đầu vào khác nhau

```
E:\Code\DA1.1\Full_Heap_Sort.exe
-----HEAP SORT VERSION-----
Nhap n:100000
Thoi gian thuc hien thuat toan heap sort la:0.03100
Nhap n:200000
Thoi gian thuc hien thuat toan heap sort la:0.08400
Nhap n:300000
Thoi gian thuc hien thuat toan heap sort la:0.09400
Nhap n:400000
Thoi gian thuc hien thuat toan heap sort la:0.14100
Nhap n:500000
Thoi gian thuc hien thuat toan heap sort la:0.18800
Nhap n:
```

Hình 2.2.2: Thời gian chạy thực tế thuật toán HeapSort với các giá trị đầu vào khác nhau

```
E:\Code\DA1.1\Sapxepchen.exe
-----INSERTION SORT VERSION-----
Nhap n:100
Thoi gian thuc hien thuat toan sap xep chen la:0.00000
Nhap n:1000
Thoi gian thuc hien thuat toan sap xep chen la:0.00000
Nhap n:2000
Thoi gian thuc hien thuat toan sap xep chen la:0.00000
Nhap n:4000
Thoi gian thuc hien thuat toan sap xep chen la:0.01600
Nhap n:8000
Thoi gian thuc hien thuat toan sap xep chen la:0.06200
Nhap n:16000
Thoi gian thuc hien thuat toan sap xep chen la:0.20900
Nhap n:32000
Thoi gian thuc hien thuat toan sap xep chen la:0.85000
Nhap n:64000
Thoi gian thuc hien thuat toan sap xep chen la:3.44000
Nhap n:100000
Thoi gian thuc hien thuat toan sap xep chen la:8.79100
Nhap n: _
```

Hình 2.2.3: Thời gian chạy thực tế thuật toán InsertionSort với các giá trị đầu vào khác nhau

```
E:\Code\DA1.1\fullSapxeptron.exe
-----MERGE SORT VERSION-----
Nhap n:100

Thoi gian thuc hien thuat toan sap xep tron la:0.00000
Nhap n:1000

Thoi gian thuc hien thuat toan sap xep tron la:0.00000
Nhap n:2000

Thoi gian thuc hien thuat toan sap xep tron la:0.01600
Nhap n:4000

Thoi gian thuc hien thuat toan sap xep tron la:0.00000
Nhap n:8000

Thoi gian thuc hien thuat toan sap xep tron la:0.00000
Nhap n:16000

Thoi gian thuc hien thuat toan sap xep tron la:0.03100
Nhap n:32000

Thoi gian thuc hien thuat toan sap xep tron la:0.03100
Nhap n:64000

Thoi gian thuc hien thuat toan sap xep tron la:0.07800
Nhap n:100000

Thoi gian thuc hien thuat toan sap xep tron la:0.12500
Nhap n:
```

Hình 2.2.4: Thời gian chạy thực tế thuật toán MergeSort với các giá trị đầu vào khác nhau

```
E:\Code\DA1.1\FullQuick_Sort.exe
-----QUICK SORT VERSION-----
Nhap n:100000

Thoi gian thuc hien thuat toan quick sort la:0.03100
Nhap n:200000

Thoi gian thuc hien thuat toan quick sort la:0.09300
Nhap n:300000

Thoi gian thuc hien thuat toan quick sort la:0.18700
Nhap n:400000

Thoi gian thuc hien thuat toan quick sort la:0.32800
Nhap n:500000

Thoi gian thuc hien thuat toan quick sort la:0.46800
Nhap n:
```

Hình 2.2.5: Thời gian chạy thực tế thuật toán QuickSort với các giá trị đầu vào khác nhau

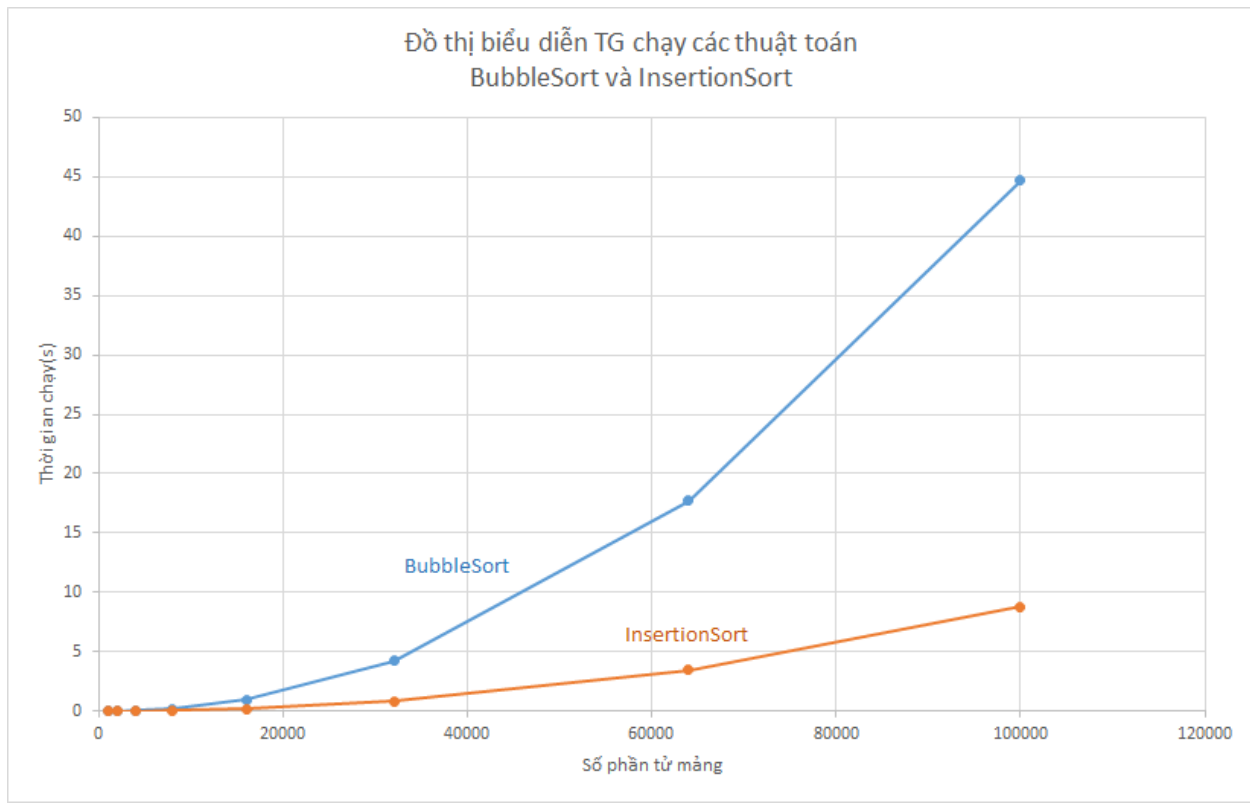
Nhận xét: Nhìn vào thời gian chạy thực tế ta có thể thấy tốc độ sắp xếp các thuật toán theo thứ tự:

- Nhanh nhất là HeapSort
- Xếp thứ 2 là QuickSort
- Xếp thứ 3 là MergeSort
- Xếp thứ 4 là InsertionSort
- Chậm nhất là BubbleSort

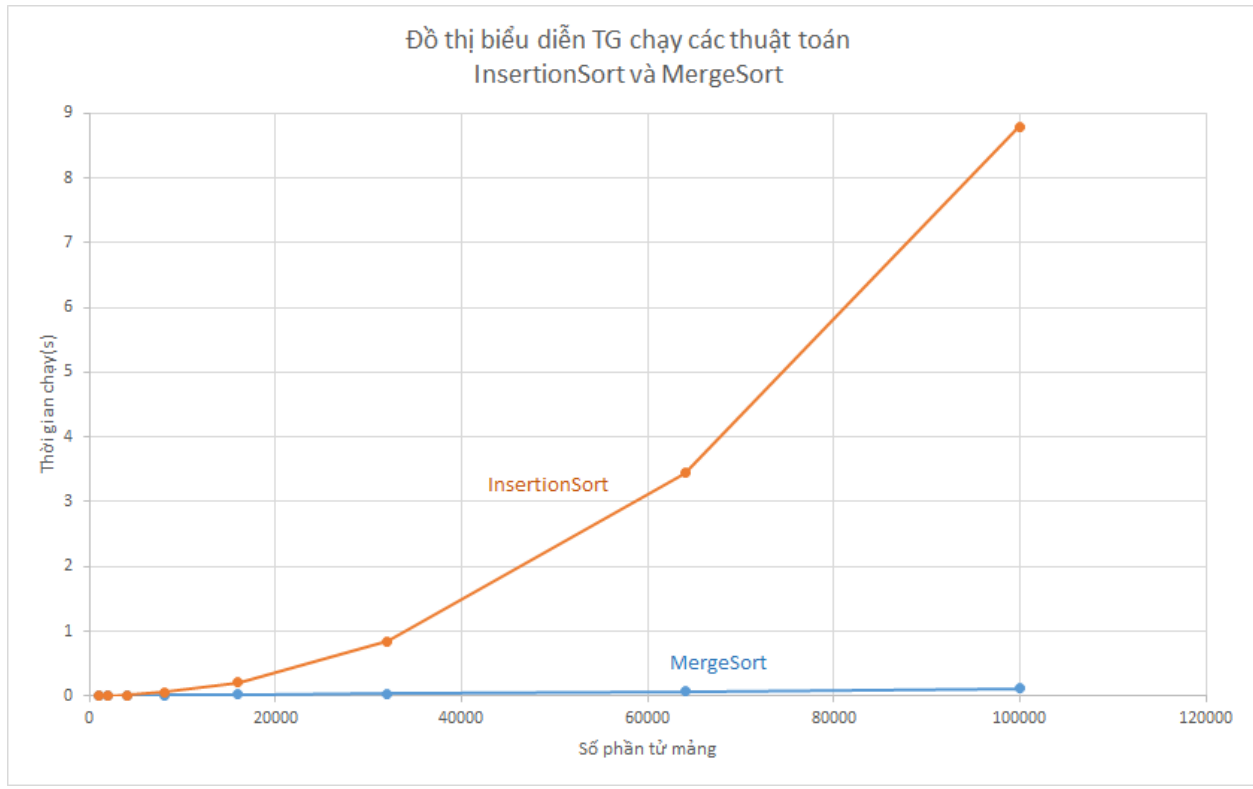
Nổi bật về tốc độ ở đây là 2 thuật toán HeapSort và QuickSort

⇒ Từ việc chạy thực tế các thuật toán sắp xếp ta có thể thấy thời gian chạy các thuật toán không chính xác trong các trường hợp đầu vào cố định mà nó còn phụ thuộc vào nhiều yếu tố ví dụ như: Bộ nhớ, thiết bị lưu trữ của máy tính mà thuật toán làm việc, tùy theo độ phức tạp của dữ liệu (trong bài ở đây dữ liệu vào là sinh ngẫu nhiên), cấu trúc dữ liệu,...v.v...

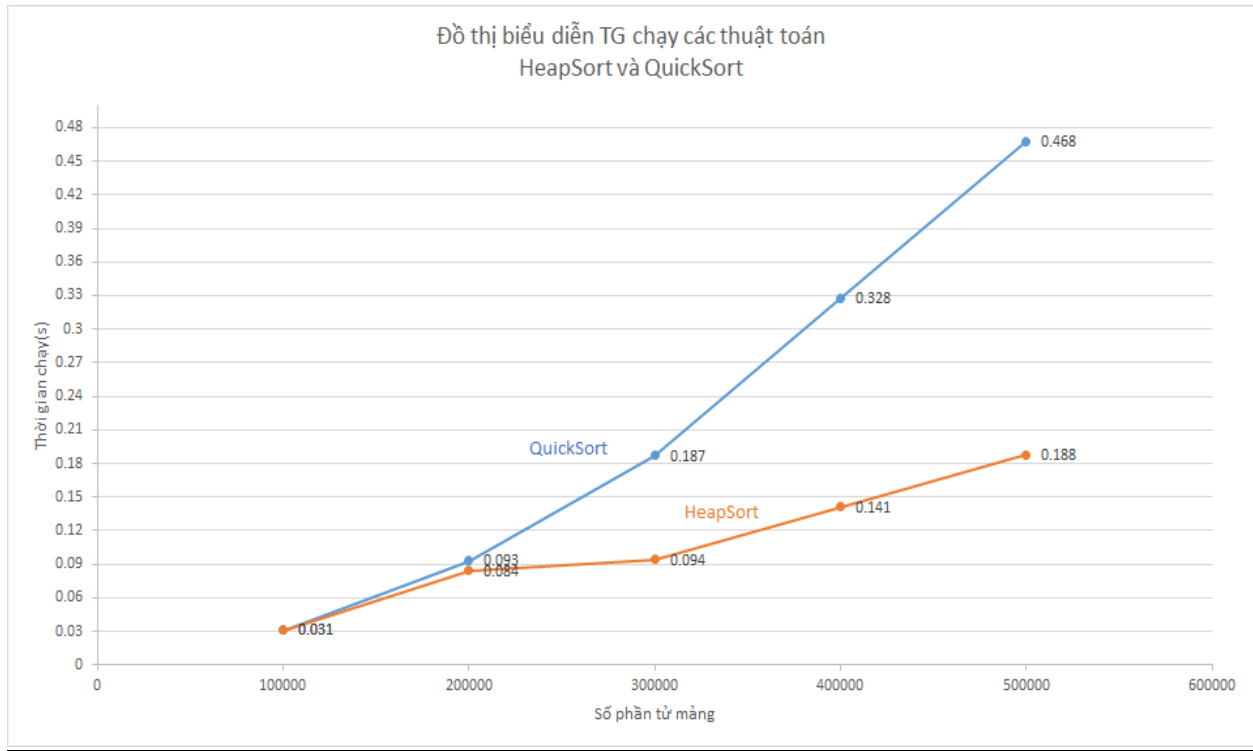
3. Đồ thị biểu diễn thời gian chạy các thuật toán



Hình 2.3.1: Đồ thị so sánh TG chạy thực tế của BubbleSort và InsertionSort



Hình 2.3.2: Đồ thị so sánh TG chạy thực tế của MergeSort và InsertionSort



Hình 2.3.3: Đồ thị so sánh TG chạy thực tế của HeapSort và QuickSort

CHƯƠNG II: CÁC BÀI TOÁN QUAY LUI

I. Bài toán quay lui

Tổng quát: 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 $X[1 \dots n]$, 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 gán cho $X[2]$ lại tiếp tục xét khả năng chọn $X[3]$...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 $(X[1], X[2], \dots, X[n])$

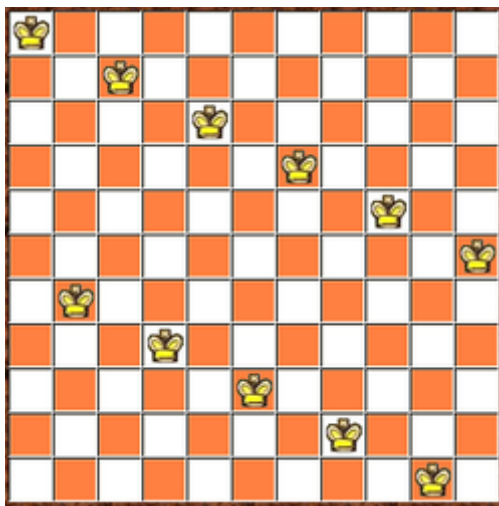
Trên phương diện quy nạp có thể nói rằng thuật toán quay lui liệt kê các cấu hình n phần tử dạng $X[1..n]$ bằng cách thử cho $X[1]$ nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho $X[1]$ bài toán trở thành liệt kê cấu hình $n-1$ phần tử sau $X[2..n]$.

Mô hình của thuật toán quay lui:

```
void Try(int i) {
    for (mọi giá trị 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);
        }
    }
}
```

II. Bài toán xếp hậu

Bài toán: Bài toán xếp hậu là bài toán đặt n quân hậu ($n \geq 4$) trên bàn cờ vua kích thước $n \times n$ sao cho không quân hậu nào có thể “ăn” được quân hậu khác.



Hình 2.1: Một lời giải VD bài toán xếp hậu bàn cờ 8×8

Phân tích và cài đặt bài toán:

Thiết kế thuật toán: Khi đã đặt quân hậu thứ i ta cần xem xét đặt quân hậu tiếp theo như thế nào? Theo luật cờ vua, 1 quân hậu có thể ăn quân hậu khác nếu cùng nằm trên 1 đường (hàng, cột, đường chéo) \Rightarrow Mỗi hàng chỉ có thể chứa 1 quân hậu.

Sử dụng giải thuật đệ quy quay lui cho bài toán mã đi tuần, ta có:

- Try(j): Chọn dòng để đặt quân hậu thứ j (ở cột j).
- Các phương án chọn: Như trên đã nêu, đối với quân hậu thứ j , vị trí của nó chỉ chọn trong cột thứ j . Vậy tham biến j trở thành chỉ số cột và việc chọn lựa “các phương án chọn” được tiến hành trên 8 giá trị số hàng i (chọn dòng i : 1, 2, 3, ..., 8).
- Chọn được: Để lựa chọn i được chấp nhận, thì hàng i và hai đường chéo qua ô (i, j) phải không có quân hậu nào ở trên đó. Ta thấy trong đường chéo theo chiều ngược (vuông góc với chéo chính) có các ô (i, j) mà tổng $i + j$ không đổi, còn đường chéo theo chiều xuôi (song song với chéo chính) có các ô (i, j) mà $i - j$ không đổi.

Quy ước:

$x[i]$: chỉ số quân hậu thứ i , nằm ở hàng thứ i

Giải quyết đường chéo đi qua hàng i cột j:

- Có 2 đường chéo đi qua hàng i cột j trong bàn cờ, gọi là đường chéo ngược và đường chéo xuôi
- Đường chéo ngược: Tất cả các ô đều có tổng 2 tọa độ i và j là hằng
- Đường chéo xuôi: Tất cả các ô đều có hiệu 2 tọa độ i và j là hằng

Lệnh đặt quân hậu:

```
x[i]=j; //đặt quân hậu thứ i trên cột j
a[j]=1; //khi quân hậu đặt ở cột j thì cột j không còn trống nữa
b[i+j]=1; //các đường chéo ngược tương ứng không còn trống nữa
c[i-j]=1; //các đường chéo xuôi tương ứng không còn trống nữa
```

Lệnh dời quân hậu:

```
a[j]=0; //không có quân hậu nào đặt ở cột j
b[i+j]=0; //không có quân hậu nào trên đường chéo ngược
c[i-j]=0; //không có quân hậu nào trên đường chéo xuôi
```

Điều kiện an toàn để đặt quân hậu:

```
a[j]==0 && b[i+j]==0 && c[i-j]==0
```

Mã giả bài toán

```
procedure TRY( j: byte );
begin
    for i :=1 to 8 do
        if (a[i]) and (b[i+j]) and (c[i-j]) then
            begin
                x[j] := i; a[i] := false;
                b[i+j] := false;
                c[i-j]:=false;
                if j=8 then
                    begin
                        for k:=1 to 8 do write(x[k], ' ')
                        writeln;
```

```

end;
else TRY(j+1);
    a[i]:=true;
    b[i+j]:=true;
    c[i-j]:=true;
end;
end;
end;

```

Kết quả chạy thử:

```

E:\Code\DA1.2\XH.exe
-----QUEEN PUZZLE-----
Kích thước bàn cờ: 8x8

Cach 1:
1 0 0 0 0 0 0 0
0 0 0 0 2 0 0 0
0 0 0 0 0 0 0 3
0 0 0 0 0 4 0 0
0 0 5 0 0 0 0 0
0 0 0 0 0 0 6 0
0 7 0 0 0 0 0 0
0 0 0 8 0 0 0 0

Cach 2:
1 0 0 0 0 0 0 0
0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 3
0 0 4 0 0 0 0 0
0 0 0 0 0 0 5 0
0 0 0 6 0 0 0 0
0 7 0 0 0 0 0 0
0 0 0 0 8 0 0 0

Cach 3:

```

Hình 2.2: Kết quả chạy bài toán xếp hậu với kích thước bàn cờ 8x8

```
E:\Code\DA1.2\XH.exe

Cach 3:
1 0 0 0 0 0 0 0
0 0 0 0 0 0 2 0
0 0 0 3 0 0 0 0
0 0 0 0 0 4 0 0
0 0 0 0 0 0 0 5
0 6 0 0 0 0 0 0
0 0 0 0 7 0 0 0
0 0 8 0 0 0 0 0

Cach 4:
1 0 0 0 0 0 0 0
0 0 0 0 0 0 2 0
0 0 0 0 3 0 0 0
0 0 0 0 0 0 0 4
0 5 0 0 0 0 0 0
0 0 0 6 0 0 0 0
0 0 0 0 0 7 0 0
0 0 8 0 0 0 0 0

Cach 5:
0 1 0 0 0 0 0 0
```

Hình 2.3: Kết quả chạy bài toán xếp hậu với kích thước bàn cờ 8x8

```
Select E:\Code\DA1.2\XH.exe

Cach 91:
0 0 0 0 0 0 0 1
0 0 2 0 0 0 0 0
3 0 0 0 0 0 0 0
0 0 0 0 0 4 0 0
0 5 0 0 0 0 0 0
0 0 0 0 6 0 0 0
0 0 0 0 0 0 7 0
0 0 0 8 0 0 0 0

Cach 92:
0 0 0 0 0 0 0 1
0 0 0 2 0 0 0 0
3 0 0 0 0 0 0 0
0 0 4 0 0 0 0 0
0 0 0 0 0 5 0 0
0 6 0 0 0 0 0 0
0 0 0 0 0 0 7 0
0 0 0 0 8 0 0 0
```

Hình 2.4: Kết quả chạy bài toán xếp hậu với kích thước bàn cờ 8x8

```
E:\Code\DA1.2\XH.exe
-----QUEEN PUZZLE-----
Kich thuc ban co: 5x5

Cach 1:
1 0 0 0 0
0 0 2 0 0
0 0 0 0 3
0 4 0 0 0
0 0 0 5 0

Cach 2:
1 0 0 0 0
0 0 0 2 0
0 3 0 0 0
0 0 0 0 4
0 0 5 0 0

Cach 3:
0 1 0 0 0
0 0 0 2 0
3 0 0 0 0
0 0 4 0 0
0 0 0 0 5
```

Hình 2.5: Kết quả chạy bài toán xếp hậu với kích thước bàn cờ 5x5

```
E:\Code\DA1.2\XH.exe

Cach 8:
0 0 0 1 0
0 2 0 0 0
0 0 0 0 3
0 0 4 0 0
5 0 0 0 0

Cach 9:
0 0 0 0 1
0 2 0 0 0
0 0 0 3 0
4 0 0 0 0
0 0 5 0 0

Cach 10:
0 0 0 0 1
0 0 2 0 0
3 0 0 0 0
0 0 0 4 0
0 5 0 0 0
```

Hình 2.6: Kết quả chạy bài toán xếp hậu với kích thước bàn cờ 5x5

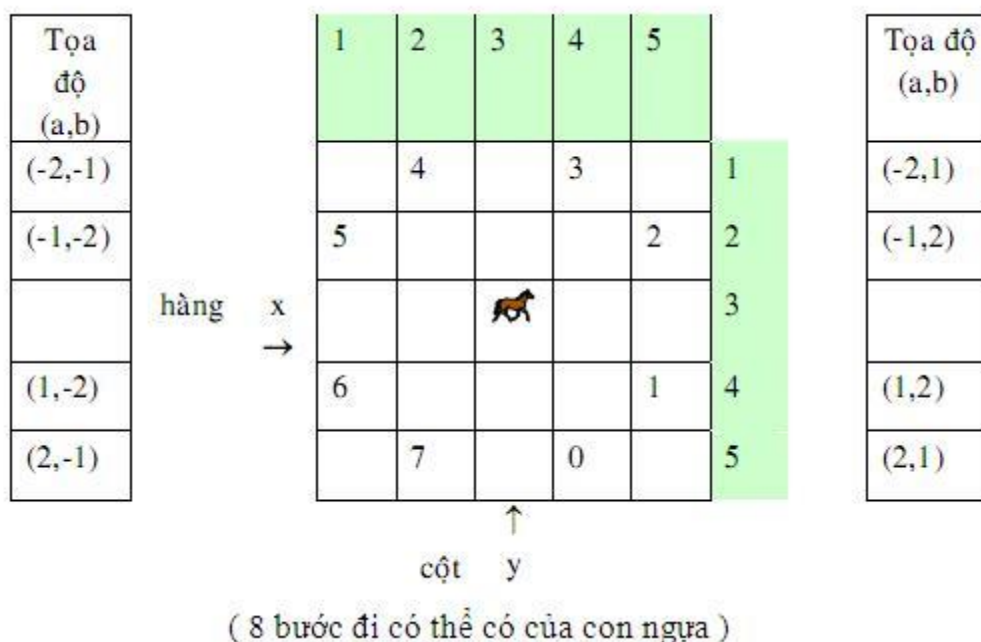
III. Bài toán mã đi tuần

Bài toán: Cho bàn cờ $n \times n$ ô. Một con mã được phép đi theo luật cờ vua, đầu tiên nó được đặt ở ô có tọa độ x_0, y_0 . Câu hỏi là: Hãy tìm cách sao cho con mã đi được tất cả các ô của bàn cờ, mỗi ô đi qua đúng 1 lần.

Phân tích bài toán: Đối với bài toán mã đi tuần ta nhận thấy rằng, ban đầu tọa độ của quân mã là (i, j) bất kỳ trên bàn cờ. Tại vị trí này theo quy luật di chuyển của quân mã trên bàn cờ thì quân mã có tối đa 8 hướng đi tiếp theo. Rõ ràng để đi đến các lời giải ta phải thử tất cả các trường hợp quân mã có thể đi đến ở bước 1. Với mỗi vị trí thử như vậy ta lại thử tất cả các trường hợp của quân mã có thể đi đến ở bước 2, ... Cứ tiếp tục như vậy cho đến khi quân mã đi qua tất cả các ô trên bàn cờ vua thì dừng lại. Như vậy, tính chất đệ quy của bài toán đó thể hiện trong các phép thử hướng đi của quân mã ở trên.

Xây dựng thuật giải:

Luật đi của con mã trên bàn cờ: Tại một ô có tọa độ cột x_0 , hàng y_0 (x_0, y_0) trên bàn cờ con mã có 1 trong 8 nước đi như sau:



Hình 3.1: Minh họa nước đi có thể của con mã

Với tọa độ bắt đầu (x_0, y_0) , có tất cả 8 ô (u, v) mà con mã có thể đi đến được. Chúng được đánh số từ 0 đến 7 trong hình 3.1

Phương pháp đơn giản để có được u, v từ x, y là cộng các chênh lệch cột, dòng về tọa độ được lưu trong 2 mảng a và b . Các giá trị trong 2 mảng a, b đã được khởi động thích ứng như sau:

Ta xem như có 1 hệ trục tọa độ (Oxy) ngay tại vị trí (x_0, y_0) của con mã, thì :

- Vị trí 1 mà con mã có thể đi được là : $u = x_0 + 2, v = y_0 + 1$
- Vị trí 2 mà con mã có thể đi được là : $u = x_0 + 1, v = y_0 + 2$
- Vị trí 3 mà con mã có thể đi được là : $u = x_0 + (-1), v = y_0 + 2$
-

Như vậy, mảng a và b có giá trị sau:

$\text{int } p[8] = \{2, 1, -1, -2, -2, -1, 1, 2\};$

$\text{int } q[8] = \{1, 2, 2, 1, -1, -2, -2, -1\};$

Ta thể hiện mỗi ô cờ bằng 1 số nguyên để đánh dấu ô đó đã được đi qua chưa, vì ta muốn lần dò theo quá trình di chuyển của con mã. Và ta qui ước như sau:

$\text{BanCo } [x][y] = 0$; ô (x, y) chưa đi qua

$\text{BanCo } [x][y] = i$; ô (x, y) đã được đi qua ở nước thứ i ($1 \leq i \leq n^2$)

Mã giả bài toán:

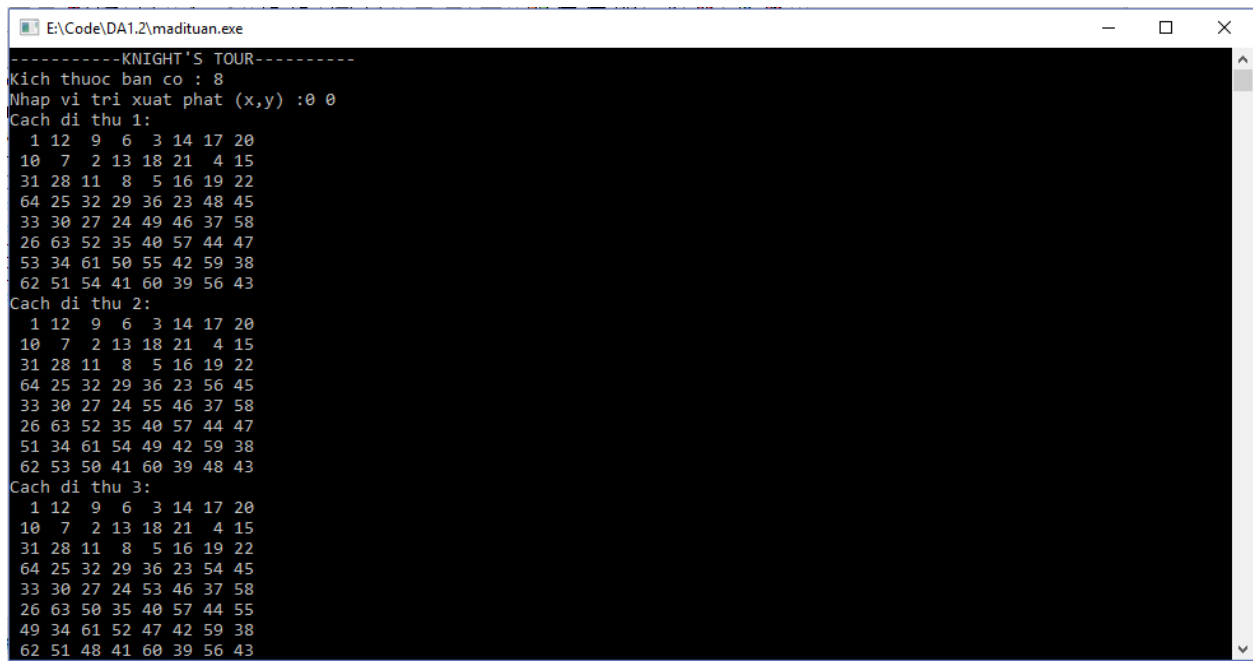
```
Try(i)
  for (j=1 to k)
    if (x[i] chấp nhận khả năng k)
    {
      //Xác định x[i] theo khả năng k;
      //Ghi nhận trạng thái mới
      if (i < n*n)
        Try(i+1);
      else
        //Ghi nhận nghiệm
        //Trả lại trạng thái cũ của bài toán
    }
```

Nhận xét:

- Để xác định tọa độ (u, v) của nước đi kế ($1 \leq i \leq 7$), ta thực hiện như sau:
 $x_1 = x + p[i]; y_1 = y + q[i]$
- Điều kiện để nước đi kế chấp nhận được là (x_1, y_1) phải thuộc bàn cờ và con mã chưa đi qua ô đó, nghĩa là ta phải thỏa các điều kiện sau:
($0 \leq x_1 < \text{KICHTHUOC} \ \&\& \ 0 \leq y_1 < \text{KICHTHUOC} \ \&\& \ \text{BanCo}[x_1][y_1] == 0$)

- Ghi nhận nước đi thứ n, nghĩa là BanCo [x1][y1] = n; còn bỏ việc ghi nhận nước đi là BanCo [x1][y1] = 0
- Bàn cờ đầy khi ta đã đi qua tất cả các ô trong BanCo, lúc đó: $k = KỊCHTHUOC^2$.

Kết quả chạy thử:



```

E:\Code\DA1.2\madituan.exe
-----KNIGHT'S TOUR-----
Kích thước bàn cờ : 8
Nhập vị trí xuất phát (x,y) :0 0
CÁCH ĐI THỨ 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
CÁCH ĐI THỨ 2:
 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 56 45
33 30 27 24 55 46 37 58
26 63 52 35 40 57 44 47
51 34 61 54 49 42 59 38
62 53 50 41 60 39 48 43
CÁCH ĐI THỨ 3:
 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 54 45
33 30 27 24 53 46 37 58
26 63 50 35 40 57 44 55
49 34 61 52 47 42 59 38
62 51 48 41 60 39 56 43

```

Hình 2.6: Kết quả chạy bài toán mã đi tuần với kích thước bàn cờ 8x8 và vị trí xuất phát (0;0)

```
E:\Code\DA1.2\madituan.exe
-----KNIGHT'S TOUR-----
Kich thước bàn cờ : 5
Nhập vị trí xuất phát (x,y) :0 0
Cach đi thu 1:
 1 18  5 10  3
 6 11  2 19 14
17 22 13  4  9
12  7 24 15 20
23 16 21  8 25
Cach đi thu 2:
 1 16  5 10  3
 6 11  2 15 20
17 22 19  4  9
12  7 24 21 14
23 18 13  8 25
Cach đi thu 3:
 1 20  5 10  3
 6 11  2 21 16
19 22 15  4  9
12  7 24 17 14
23 18 13  8 25
Cach đi thu 4:
 1 18  5 12  3
 8 13  2 17  6
19 22  7  4 11
14  9 24 21 16
23 20 15 10 25
Cach đi thu 5:
 1 10  5 18  3
14 19  2 11  6
```

Hình 2.6: Kết quả chạy bài toán mã đi tuần với kích thước bàn cờ 5x5 và vị trí xuất phát (0;0)

```
E:\Code\DA1.2\madituan.exe
-----KNIGHT'S TOUR-----
Kich thước bàn cờ : 5
Nhập vị trí xuất phát (x,y) :2 2
Cach đi thu 1:
19  2 13  8 25
12  7 18  3 14
17 20  1 24  9
 6 11 22 15  4
21 16  5 10 23
Cach đi thu 2:
25  2 13  8 19
12  7 18  3 14
17 24  1 20  9
 6 11 22 15  4
23 16  5 10 21
Cach đi thu 3:
21  2 13  8 23
12  7 22  3 14
17 20  1 24  9
 6 11 18 15  4
19 16  5 10 25
Cach đi thu 4:
23  2 13  8 21
12  7 22  3 14
17 24  1 20  9
 6 11 18 15  4
25 16  5 10 19
Cach đi thu 5:
19  2  7 12 21
 8 13 20 17  6
```

Hình 2.6: Kết quả chạy bài toán mã đi tuần với kích thước bàn cờ 5x5 và vị trí xuất phát (2;2)

III. Bài toán người du lịch

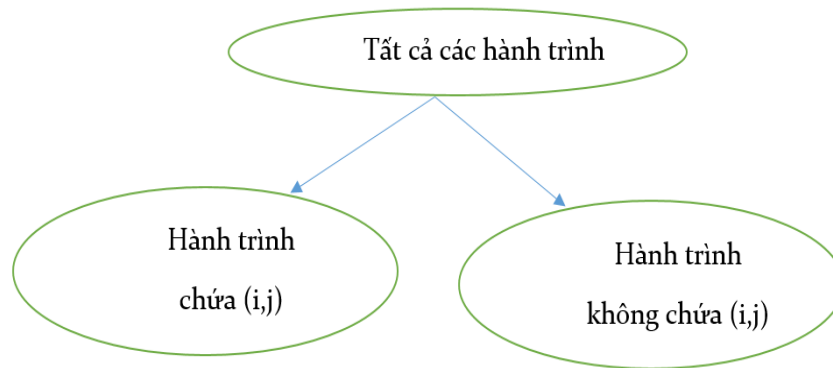
1. Bài toán:

Một người du lịch muốn tham quan n thành phố T_1, \dots, T_n . Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua đúng 1 lần rồi quay trở lại thành phố xuất phát. Từ thành phố T_i đến thành phố T_j mất chi phí là C_{ij} , chi phí có thể là thời gian, quãng đường hoặc giá tiền. Hãy tìm một hành trình thỏa yêu cầu bài toán sao cho chi phí là nhỏ nhất.

2. Thuật toán nhánh cận giải bài toán người du lịch:

Thuật toán nhánh cận là một trong những phương pháp giải chủ yếu của tối ưu tổ hợp. Như trong mục trước đã thấy, tư tưởng cơ bản của nó là trong quá trình tìm kiếm lời giải ta sẽ phân hoạch tập các phương án của bài toán ra thành hai hay nhiều tập con được biểu diễn như là các nút của cây tìm kiếm và cố gắng bằng phép đánh giá cận cho các nút, tìm cách loại bỏ những nhánh của cây tìm kiếm (những tập con các phương án của bài toán) mà ta biết chắc chắn là không chứa phương án tối ưu. Mặc dù trong tình huống tồi nhất thuật toán sẽ trở thành duyệt toàn bộ, nhưng trong nhiều trường hợp cụ thể, kỹ thuật đó cho phép rút ngắn được một cách đáng kể quá trình tìm kiếm.

Trong bài toán người du lịch khi tiến hành tìm kiếm lời giải chúng ta sẽ phân tập các hành trình ra thành hai tập con: một tập gồm những hành trình chứa một cạnh $\{i, j\}$ nào đó còn tập kia gồm những hành trình không chứa cạnh này. Ta gọi việc làm đó là phân nhánh và mỗi tập con nói trên sẽ được gọi là một nhánh hay một nút của cây tìm kiếm. Việc phân nhánh được minh họa bởi cây tìm kiếm:



Hình 3.1: Sơ đồ phân nhánh

Việc phân nhánh sẽ được thực hiện dựa trên một quy tắc heuristic nào đó cho phép ta rút ngắn quá trình tìm kiếm phương án tối ưu. Sau khi phân nhánh ta sẽ tính cận dưới của giá trị hàm mục tiêu trên mỗi một trong hai tập con nói trên. Việc tìm kiếm sẽ được tiếp tục trên tập con có giá trị cận dưới nhỏ hơn. Thủ tục này sẽ được tiếp tục cho đến khi thu được một hành trình đầy đủ tức là một phương án của bài toán người du lịch. Khi đó ta cần xét những tập con các phương án nào có cận dưới nhỏ hơn giá trị hàm mục tiêu tại phương án tìm được. Quá trình phân nhánh và tính cận dưới trên tập các phương án của bài toán thông thường cho phép rút ngắn một cách đáng kể quá trình tìm kiếm do ta loại được khá nhiều tập con chắc chắn không chứa phương án tối ưu.

a, Thủ tục rút gọn:

Rõ ràng tổng chi phí của một hành trình của người đi du lịch sẽ chứa đúng một phần tử của mỗi dòng và đúng một phần tử của mỗi cột trong ma trận chi phí C . Do đó nếu ta trừ với mỗi phần tử của dòng (hay cột) của ma trận C đi cùng một số α thì chi phí của tất cả các hành trình cũng sẽ giảm đi α , vì thế hành trình tối ưu cũng sẽ không thay đổi. Vì vậy nếu ta tiến hành trừ bớt các phần tử của mỗi dòng và mỗi cột của nó đi phần tử “khác 0 nhỏ nhất” thì ma trận thu được ma trận gồm các phần tử không âm mà trong mỗi dòng và mỗi cột của nó đều có ít nhất một số 0 thì tổng các hằng số trừ đó sẽ cho ta cận dưới của mọi hành trình. Thủ tục trừ bớt này sẽ được gọi là thủ tục rút gọn, các hằng số trừ ở mỗi dòng(cột) sẽ được gọi là hằng số rút gọn theo dòng(cột) còn ma trận thu được gọi là ma trận rút gọn.

Mã giả thủ tục rút gọn:

```
function Reduce(A, k): real;
begin
    sum := 0;
    for i := 1 to k do      //k là kích thước của A
    begin
        r[i] :=;           //phần tử nhỏ nhất trong dòng i
        if r[i] > 0 then
        begin
            sum := sum + r[i];    //bớt mọi phần tử của dòng i đi r[i]
        end;
    end;
    for j := 1 to k do
    begin
        s[j] :=;           //phần tử nhỏ nhất trong cột j
        if s[j] > 0 then
        begin
            sum := sum + s[j];    //bớt mọi phần tử của cột j đi s[j]
        end;
    end;
    Reduce := sum;
end;
```

b, Thủ tục chọn cạnh phân nhánh (r,c)

Đầu vào: Ma trận rút gọn A kích thước $k \times k$.

Đầu ra: Cạnh phân nhánh (r,c) và tổng hằng số rút gọn theo dòng r cột c là beta.

Mã giả thủ tục phân nhánh:

```
procedure BestEdge(A,k,r,c,beta);
begin
    beta := -∞;
    for i := 1 to k do
        for j := 1 to k do
            if a[i,j] = 0 then
                begin
                    minr := <phan tu nho nhat tren dong i khac a[i,j];
                    minc := <phan tu nho nhat tren cot j khac a[i,j];
                    total := minr + minc;
                    if total > beta then
                        begin
                            beta := total;
                            r := i;      //chi so dong cua canh tot nhat
                            c := j;      //chi so cot cua canh tot nhat
                        end;
                    end;
                end;
        end;
    end;
end;
```

Ngăn cấm tạo chu trình con kín: Theo nội dung của bài toán, một hành trình hợp lệ là phải đi qua tất cả các thành phố trong danh sách, nói cách khác hành trình sẽ tạo thành một vòng khép kín. Vấn đề đặt ra là nếu trong danh sách chưa đủ mặt các thành phố mà hành trình đã tạo thành một vòng con khép kín, hiện tượng này được gọi là tạo thành chu trình con kín, do đó trong quá trình thực hiện thủ tục chọn cung và phân nhánh cần loại bỏ khả năng tạo thành chu trình con kín.

Tổng quát hơn khi phân nhánh dựa vào cung(p,q) ta phải thêm cạnh này vào hành trình, ở mỗi bước lựa chọn cung để kết nạp vào ta cần phải kiểm tra và xem xét nó

có tạo thành chu trình con kín khi kết hợp với các cung đã có hay không ? nếu có ta cần loại bỏ cung này và chọn cung có độ ưu tiên (độ ưu tiên được xác định bởi phần tử 0 có chỉ số bằng hoặc nhỏ hơn) tiếp theo.

c, Thuật toán nhánh cận giải bài toán người du lịch

Bây giờ ta có thể mô tả các bước chính của thuật toán nhánh cận giải bài toán người du lịch trong thủ tục đệ quy TSP sau đây. Thủ tục TSP xét hành trình bộ phận với Edges cạnh đã được chọn và tiến hành tìm kiếm tiếp theo. Ta sử dụng các biến;

Edges - số cạnh trong hành trình bộ phận;

A - ma trận chi phí tương ứng với kích thước $(n\text{-edges}) \times (n\text{-edges})$;

cost - chi phí của hành trình bộ phận;

MinCost - chi phí của hành trình tốt nhất đã tìm được.

Mã giả thủ tục phân nhánh:

```
procedure TSP(edges, cost, A);
begin
  cost := cost + Reduce(A, n-edges);
  if cost < MinCost then
    if edges = n-2 then
      begin
        <bo sung 2 canh con lai>;
        MinCost := cost;
        <ghi nhan hanh trinh tot nhat>
      end;
    else
      begin
        BestEdge(A, n-edges, r, c, beta);
        LowerBound := cost + beta;
        <ngan cam tao chu trinh con>;
        NewA := <A loai bo dong r cot c>;
```

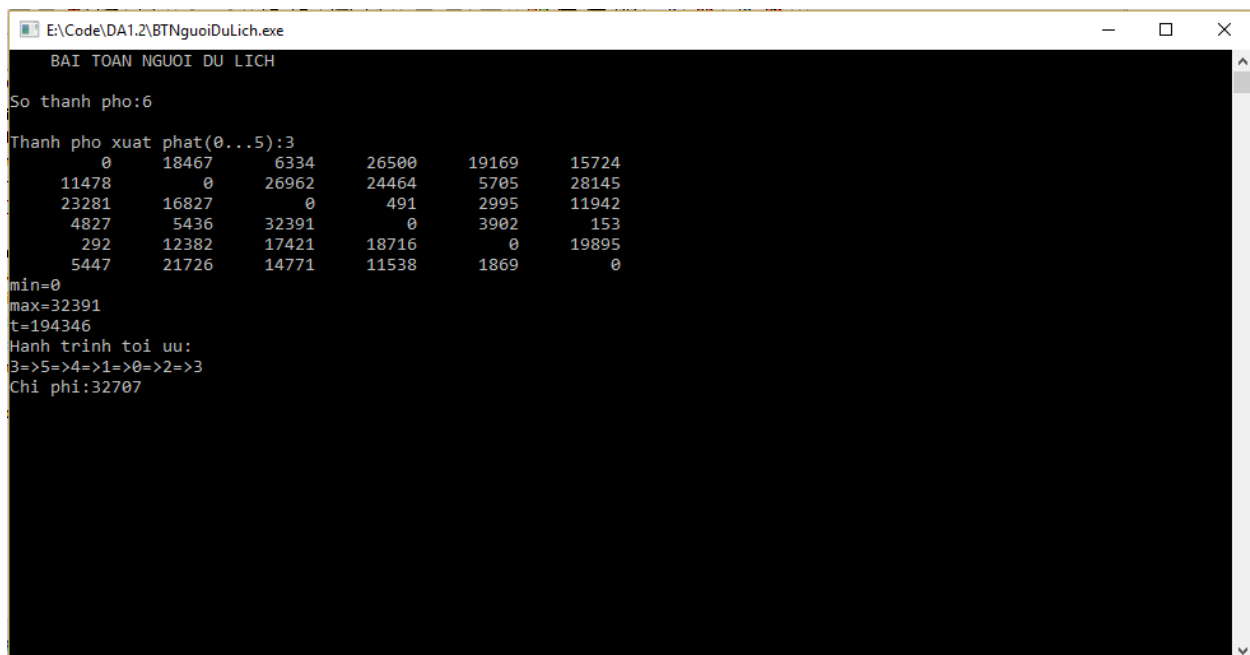


```

TSP(edges+1,cost,New A );    //di theo nhanh trai
<khoi phuc A bang cach bo sung lai dong r cot k>;
if LowerBound < MinCost then
begin                          //di theo nhanh phai
    A[r,c]:=∞;
    TSP(edges, cost, A);
    A [r,c ]:=0;
end;
end;
<khoi phuc ma tran A>;
end;

```

3. Kết quả chạy thử:



```

E:\Code\DA1.2\BTNguoiDuLich.exe
BAI TOAN NGUOI DU LICH
So thanh pho:6
Thanh pho xuat phat(0...5):3
    0      18467    6334    26500    19169    15724
11478      0      26962    24464    5705    28145
23281    16827      0      491    2995    11942
4827     5436    32391      0    3902     153
292     12382    17421    18716      0    19895
5447     21726    14771    11538    1869      0
min=0
max=32391
t=194346
Hanh trinh toi uu:
3->5->4->1->0->2->3
Chi phi:32707

```

Hình 3.3.1: Kết quả chạy thử

```
E:\Code\DA1.2\BTNguoiDuLich.exe
BAI TOAN NGUOI DU LICH
So thanh pho:10
Thanh pho xuat phat(0...9):4
0 18467 6334 26500 19169 15724 11478 29358 26962 24464
5705 0 23281 16827 9961 491 2995 11942 4827 5436
32391 14604 0 153 292 12382 17421 18716 19718 19895
5447 21726 14771 0 1869 19912 25667 26299 17035 9894
28703 23811 31322 30333 0 4664 15141 7711 28253 6868
25547 27644 32662 32757 20037 0 8723 9741 27529 778
12316 3035 22190 1842 288 30106 0 8942 19264 22648
27446 23805 15890 6729 24370 15350 15006 0 24393 3548
19629 12623 24084 19954 18756 11840 4966 7376 0 26308
16944 32439 24626 11323 5537 21538 16118 2082 22929 0
min=0
max=32757
t=327570
Hanh trinh toi uu:
4=>5=>9=>7=>3=>0=>2=>1=>8=>6=>4
Chi phi:50719
```

Hình 3.3.2: Kết quả chạy thử

CHƯƠNG III: BÀI TOÁN ĐÁNH CHỈ MỤC TỪ KHÓA

I. Bài toán

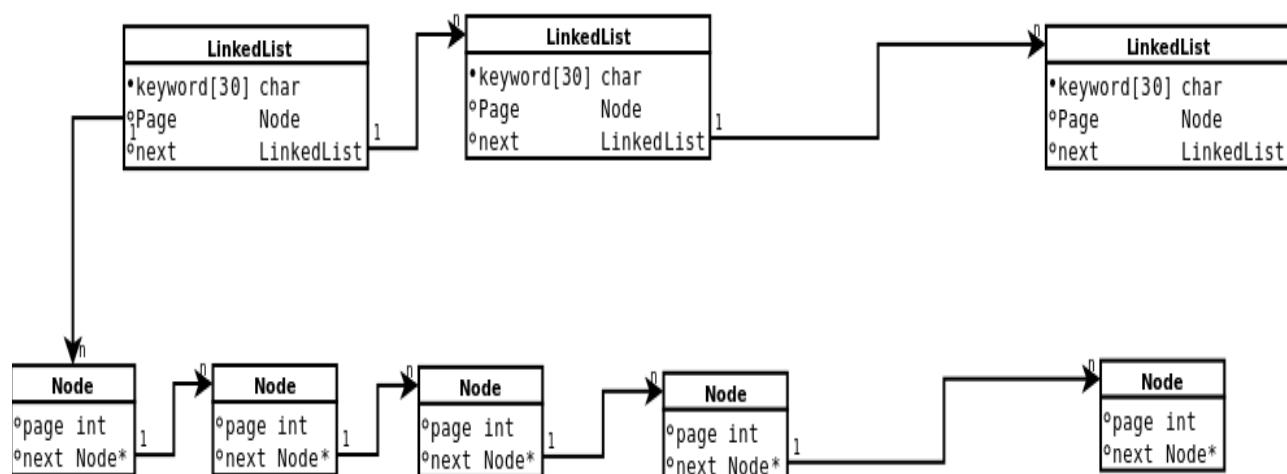
Có 1 file văn bản gồm danh sách các từ khóa và một file văn bản khác là tài liệu chứa các từ khóa đó. Viết chương trình lưu danh sách từ khóa và dòng xuất hiện của từ khóa đó trong tài liệu, cài đặt với các cấu trúc dữ liệu khác nhau: Danh sách liên kết, Bảng băm và Cây nhị phân tìm kiếm.

II. Ý tưởng

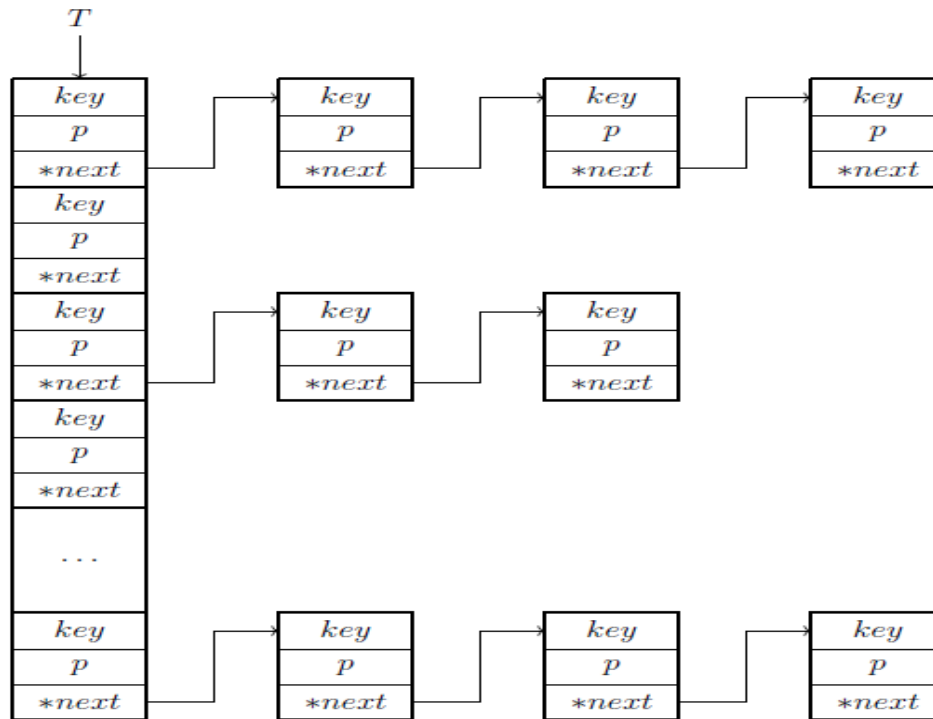
Lưu các từ khóa trong các cấu trúc dữ liệu khác nhau. Với mỗi nút, ngoài mảng kí tự chứa từ khóa, ta còn lưu thêm một biến kiểu Node là Header của danh sách liên kết, mà mỗi phần tử của danh sách liên kết là một số nguyên, biểu thị cho vị trí xuất hiện của từ khóa.

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:



Hình 2.1: Danh sách liên kết chứa từ khóa



Hình 2.2: Lưu các từ khóa vào bảng băm

VI. Kết quả chạy thử

```

E:\Code\DA1.3\New folder\hashtable.exe
Dong 7: mind. The second purpose ideally requires a language that is close to the problem to be solved
Dong 8: so that the concepts of a solution can be expressed directly and concisely. The facilities added to C
Dong 9: to create C, such as function argument checking, const, classes, constructors and destructors,
Dong 10: exceptions, and templates, were primarily designed with this in mind. Thus, C is based on the
Dong 11: idea of providing both.

+-----+-----+-----+-----+
| Tu khoa | GT Bam | So lan XH | XH o dong |
+-----+-----+-----+-----+
| useful  | 538    | 0         | 1 2 3 4 5 7 8 |
| go      | 170    | 0         | 1 2 3 4 6 7 8 9 |
| a       | 0      | 9         | 1 3 5 8 11 |
| to      | 508    | 10        | 3 5 8 9 10 |
| of      | 369    | 5         | 10 |
| and     | 13     | 1         | 1 2 4 6 7 |
| on      | 377    | 5         |
| language| 286    |
+-----+-----+-----+-----+

Process exited after 0.07629 seconds with return value 0
Press any key to continue . . .

```

Hình 6.1: Kết quả chạy thử với bảng băm

```
Select E:\Code\DA1.3\New folder\linklist.exe
Dong 7: mind. The second purpose ideally requires a language that is close to the problem to be solved
Dong 8: so that the concepts of a solution can be expressed directly and concisely. The facilities added to C
Dong 9: to create C, such as function argument checking, const, classes, constructors and destructors,
Dong 10: exceptions, and templates, were primarily designed with this in mind. Thus, C is based on the
Dong 11: idea of providing both.

+-----+-----+-----+
| Tu khoa | So lan XH | XH o dong |
+-----+-----+-----+
| useful  | 0         |            |
| go      | 0         |            |
| a       | 9         | 1 2 3 4 5 7 8 |
| to      | 10        | 1 2 3 4 6 7 8 9 |
| of      | 5         | 1 3 5 8 11 |
| and     | 5         | 3 5 8 9 10 |
| on      | 1         | 10 |
| language| 5         | 1 2 4 6 7 |
+-----+-----+-----+

Process exited after 0.09933 seconds with return value 0
Press any key to continue . . .
```

Hình 6.2: Kết quả chạy thử với danh sách liên kết

```
E:\Code\DA1.3\New folder\tree.exe
Dong 7: mind. The second purpose ideally requires a language that is close to the problem to be solved
Dong 8: so that the concepts of a solution can be expressed directly and concisely. The facilities added to C
Dong 9: to create C, such as function argument checking, const, classes, constructors and destructors,
Dong 10: exceptions, and templates, were primarily designed with this in mind. Thus, C is based on the
Dong 11: idea of providing both.

+-----+-----+-----+
| Tu khoa | So lan XH | XH o dong |
+-----+-----+-----+
| useful  | 0         |            |
| go      | 0         |            |
| a       | 9         | 1 2 3 4 5 7 8 |
| to      | 10        | 1 2 3 4 6 7 8 9 |
| of      | 5         | 1 3 5 8 11 |
| and     | 5         | 3 5 8 9 10 |
| on      | 1         | 10 |
| language| 5         | 1 2 4 6 7 |
+-----+-----+-----+

Process exited after 0.1366 seconds with return value 0
Press any key to continue . . .
```

Hình 6.1: Kết quả chạy thử với cây

KẾT LUẬN

Qua bài tập ở 3 chương em đã tìm hiểu và cài đặt được các bài toán áp dụng các thuật toán khác nhau: Thuật toán sắp xếp, thuật toán quay lui nhánh cận, thuật toán tìm kiếm.

Ưu điểm:

- Hiểu và mô tả được quá trình thực hiện các thuật toán, tìm hiểu được ưu nhược điểm của mỗi phương pháp khác nhau.
- Demo bằng các dữ liệu ngẫu nhiên để test và chương trình chạy cho kết quả đúng.
- Giao diện chương trình dễ nhìn, dễ sử dụng.

Khuyết điểm, hạn chế:

- Chương trình còn đơn giản, chưa biểu diễn hết chi tiết được các quá trình sắp xếp, quay lui và tìm kiếm.
- Chưa cài đặt được các yêu cầu nâng cao.
- Thời gian hoàn thành bài còn chậm.
- Chưa hiểu được sâu về các thuật toán quay lui và tìm kiếm.

Bài học rút ra:

- Cần tìm hiểu kĩ hơn về các thuật toán, ngôn ngữ lập trình để chương trình có giao diện đẹp hơn, lập trình tốt hơn.
- Do đây là lần đầu thực hiện bài tập theo chủ đề, kiến thức lập trình còn hạn chế nên chương trình còn nhiều chỗ chưa được hoàn thiện. Em rất mong nhận được sự đóng góp ý kiến của thầy và các bạn để em có thêm kinh nghiệm làm những bài tập chủ đề lần sau tốt hơn.

Để hoàn thành tốt được bài tập lần này là do em nhận được sự hướng dẫn tận tình của thầy Phạm Đăng Hải. Em rất mong nhận được thêm sự giúp đỡ và chỉ bảo của thầy để có thể cải thiện được khả năng của bản thân.

Em xin chân thành cảm ơn thầy!

PHỤ LỤC

Vì độ dài các chương trình nguồn khi đưa vào bài không tiện nên e đã tải lên các SourceCode để có thể tiện theo dõi:

1. Link GoogleDrive:

<https://drive.google.com/file/d/0B1VW5aRVFgyUbVRRME1DbHBfVnM/view?usp=sharing>

2. Link Codepad.org:

Bài 1:

- HeapSort: <http://codepad.org/SKYSunRT>
- QuickSort: <http://codepad.org/bSkCMn3c>
- MergeSort: <http://codepad.org/N1GXjErm>
- InsertionSort: <http://codepad.org/rQdMX0DT>
- BubbleSort: <http://codepad.org/uCitHqRu>

Bài 2:

- Xếp hậu: <http://codepad.org/u16lbcqe>
- Mã đi tuần: <http://codepad.org/M3H0Ix1R>
- Người du lịch: <http://codepad.org/CB5JRjmv>

Bài 3:

- Danh sách liên kết: <http://codepad.org/RiZsmIMS>
- Cây: <http://codepad.org/koOdr8ip>
- Bảng băm: <http://codepad.org/K6Qka5tE>