**8.12. CÀI ĐẶT**

Ta sẽ cài đặt tất cả các thuật toán sắp xếp nêu trên, với dữ liệu vào được đặt trong file văn bản SORT.INP chứa không nhiều hơn 15000 khoá và giá trị mỗi khoá là số tự nhiên không quá 15000. Kết quả được ghi ra file văn bản SORT.OUT chứa dãy khoá được sắp, mỗi khoá trên một dòng.

|  |  |
| --- | --- |
| SORT.INP | SORT.OUT |
| 1 4 3 2 5 | 1 |
| 7 9 8 | 2 |
| 10 6 | 3 |
|  | 4 |
|  | 5 |
|  | 6 |
|  | 7 |
|  | 8 |
|  | 9 |
|  | 10 |

Chương trình có giao diện dưới dạng menu, mỗi chức năng tương ứng với một thuật toán sắp xếp. Tại mỗi thuật toán sắp xếp, ta thêm một vài lệnh đo thời gian thực tế của nó (chỉ đo thời gian thực hiện giải thuật, không tính thời gian nhập liệu và in kết quả).

Ở thuật toán sắp xếp bằng cơ số theo cách hoán vị phần tử, ta chọn hệ nhị phân. Ở thuật toán sắp xếp bằng cơ số trực tiếp, ta sử dụng hệ cơ số 256, khi đó một giá trị số tự nhiên x  15000 sẽ được biểu diễn bằng hai chữ số trong hệ 256:

Chữ số hàng đơn vị là x mod 256 = x mod 28 = x and 255 = x and $FF; Chữ số còn lại (= chữ số ở hàng cao nhất) là x div 256 = x div 28 = x shr 8;

P\_2\_08\_1.PAS \* Các thuật toán săp xếp

{$M 65520 0 655360}

program SortingAlgorithmsDemo;

uses crt;

const

InputFile = 'SORT.INP';

OutputFile = 'SORT.OUT';

max = 15000;

maxV = 15000;

Interval = 1193180 / 65536; {Tần số đồng hồ  18.2 lần / giây}

nMenu = 12;

SMenu: array[0..nMenu] of String = (

' 0. Display Input',

' 1. SelectionSort',

' 2. BubbleSort',

' 3. InsertionSort',

' 4. InsertionSort with binary searching',

' 5. ShellSort',

' 6. QuickSort',

' 7. HeapSort',

' 8. Distribution Counting',

' 9. Exchange RadixSort',

' 10. Straight RadixSort',

' 11. MergeSort',

' 12. Exit'

);

type

TArr = array[1..max] of Integer;

TCount = array[0..maxV] of Integer;

var

k: TArr;

n: Integer;

selected: Integer;

StTime: LongInt;

Time: LongInt absolute 0:$46C; {Biến đếm nhịp đồng hồ}

procedure Enter; {Trước mỗi thuật toán sắp xếp, gọi thủ tục này để nhập liệu}

var

f: Text;

begin

Assign(f, InputFile);

Reset(f);

n := 0;

while not SeekEof(f) do

begin

Inc(n);

Read(f, k[n]);

end;

Close(f);

StTime := Time; {Nhập xong bắt đầu tính thời gian ngay}

end;

procedure PrintInput; {In dữ liệu}

var

i: Integer;

begin

Enter;

for i := 1 to n do Write(k[i]:8);

Write('Press any key to return to menu…');

ReadKey;

end;

procedure PrintResult; {In kết quả của mỗi thuật toán sắp xếp}

var

f: Text;

i: Integer;

ch: Char;

begin

{Trước hết in ra thời gian thực thi}

WriteLn('Running Time = ', (Time - StTime) / Interval:1:10, ' (s)');

Assign(f, OutputFile);

Rewrite(f);

for i := 1 to n do WriteLn(f, k[i]);

Close(f);

Write('Press <P> to print Output, another key to return to menu…');

ch := ReadKey;

WriteLn(ch);

if Upcase(ch) = 'P' then

begin

for i := 1 to n do Write(k[i]:8);

WriteLn;

Write('Press any key to return to menu…');

ReadKey;

end;

end;

procedure Swap(var x, y: Integer); {Thủ tục đảo giá trị hai tham biến x, y}

var

t: Integer;

begin

t := x;

x := y;

y := t;

end;

(\*\* SELECTIONSORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) procedure SelectionSort;

var

i, j, jmin: Integer;

begin

Enter;

for i := 1 to n - 1 do

begin

jmin := i;

for j := i + 1 to n do

if k[j] < k[jmin] then jmin := j; if jmin <> i then Swap(k[i], k[jmin]);

end;

PrintResult;

end;

(\*\* BUBBLESORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) procedure BubbleSort;

var

i, j: Integer;

begin

Enter;

for i := 2 to n do

for j := n downto i do

if k[j - 1] > k[j] then Swap(k[j - 1], k[j]);

PrintResult;

end;

(\*\* INSERTIONSORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) procedure InsertionSort;

var

i, j, tmp: Integer;

begin

Enter;

for i := 2 to n do

begin

tmp := k[i];

j := i - 1;

while (j > 0) and (tmp < k[j]) do

begin

k[j + 1] := k[j];

Dec(j);

end;

k[j + 1] := tmp;

end;

PrintResult;

end;

(\*\* INSERTIONSORT WITH BINARY SEARCHING \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

procedure AdvancedInsertionSort;

var

i, inf, sup, median, tmp: Integer;

begin

Enter;

for i := 2 to n do

begin

tmp := k[i];

inf := 1;

sup := i - 1;

repeat

median := (inf + sup) shr 1;

if tmp < k[median] then sup := median - 1

else inf := median + 1;

until inf > sup;

Move(k[inf], k[inf + 1], (i - inf) \* SizeOf(k[1]));

k[inf] := tmp;

end;

PrintResult;

end;

(\*\* SHELLSORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) procedure ShellSort;

var

tmp: Integer;

i, j, h: Integer;

begin

Enter;

h := n shr 1;

while h <> 0 do

begin

for i := h + 1 to n do

begin

tmp := k[i];

j := i - h;

while (j > 0) and (k[j] > tmp) do

begin

k[j + h] := k[j];

j := j - h;

end;

k[j + h] := tmp;

end;

h := h shr 1;

end;

PrintResult;

end;

(\*\* QUICKSORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) procedure QuickSort;

procedure Partition(L, H: Integer);

var

i, j: Integer;

Pivot: Integer;

begin

if L >= H then Exit;

Pivot := k[L + Random(H - L + 1)];

i := L;

j := H;

repeat

while k[i] < Pivot do Inc(i);

while k[j] > Pivot do Dec(j);

if i <= j then

begin

if i < j then Swap(k[i], k[j]);

Inc(i);

Dec(j);

end;

until i > j;

Partition(L, j);

Partition(i, H);

end;

begin

Enter;

Partition(1, n);

PrintResult;

end;

(\*\* HEAPSORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) procedure HeapSort;

var

r, i: Integer;

procedure Adjust(root, endnode: Integer);

var

key, c: Integer;

begin

key := k[root];

while root shl 1 <= endnode do

begin

c := root shl 1;

if (c < endnode) and (k[c] < k[c + 1]) then Inc(c);

if k[c] <= key then Break;

k[root] := k[c];

root := c;

end;

k[root] := key;

end;

begin

Enter;

for r := n shr 1 downto 1 do Adjust(r, n);

for i := n downto 2 do

begin

Swap(k[1], k[i]);

Adjust(1, i - 1);

end;

PrintResult;

end;

(\*\* DISTRIBUTION COUNTING \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

procedure DistributionCounting;

var

x: TArr; c: TCount;

i, V: Integer;

begin

Enter;

FillChar(c, SizeOf(c), 0);

for i := 1 to n do Inc(c[k[i]]);

for V := 1 to MaxV do c[V] := c[V - 1] + c[V];

for i := n downto 1 do

begin

V := k[i];

x[c[V]] := k[i];

Dec(c[V]);

end;

k := x;

PrintResult;

end;

(\*\* EXCHANGE RADIXSORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

procedure RadixSort;

const

MaxBit = 13;

var

MaskBit: array[0..MaxBit] of Integer;

MaxValue, i: Integer;

procedure Partition(L, H, BIndex: Integer);

var

i, j, Mask: Integer;

begin

if L >= H then Exit;

i := L; j := H; Mask := MaskBit[BIndex];

repeat

while (i < j) and (k[i] and Mask = 0) do Inc(i);

while (i < j) and (k[j] and Mask <> 0) do Dec(j);

Swap(k[i], k[j]);

until i = j;

if k[j] and Mask = 0 then Inc(j);

if BIndex > 0 then

begin

Partition(L, j - 1, BIndex - 1);

Partition(j, H, BIndex - 1);

end;

end;

begin

Enter;

for i := 0 to MaxBit do MaskBit[i] := 1 shl i;

maxValue := k[1];

for i := 2 to n do

if k[i] > MaxValue then maxValue := k[i];

i := 0;

while (i < MaxBit) and (MaskBit[i + 1] <= MaxValue) do Inc(i);

Partition(1, n, i);

PrintResult;

end;

(\*\* STRAIGHT RADIXSORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

procedure StraightRadixSort;

const

Radix = 256;

nDigit = 2;

var

t: TArr;

p: Integer;

Flag: Boolean;

function GetDigit(key, p: Integer): Integer;

begin

if p = 0 then GetDigit := key and $FF else GetDigit := key shr 8;

end;

procedure DCount(var x, y: TArr; p: Integer);

var

c: array[0..Radix - 1] of Integer; i, d: Integer;

begin

FillChar(c, SizeOf(c), 0);

for i := 1 to n do

begin

d := GetDigit(x[i], p); Inc(c[d]);

end;

for d := 1 to Radix - 1 do c[d] := c[d - 1] + c[d];

for i := n downto 1 do

begin

d := GetDigit(x[i], p);

y[c[d]] := x[i];

Dec(c[d]);

end;

end;

begin

Enter;

Flag := True;

for p := 0 to nDigit - 1 do

begin

if Flag then DCount(k, t, p)

else DCount(t, k, p);

Flag := not Flag;

end;

if not Flag then k := t;

PrintResult;

end;

(\*\* MERGESORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*) procedure MergeSort;

var

t: TArr;

Flag: Boolean;

len: Integer;

procedure Merge(var Source, Dest: TArr; a, b, c: Integer);

var

i, j, p: Integer;

begin

p := a; i := a; j := b + 1;

while (i <= b) and (j <= c) do

begin

if Source[i] <= Source[j] then

begin

Dest[p] := Source[i];

Inc(i);

end;

else

begin

Dest[p] := Source[j];

Inc(j);

end;

Inc(p);

end;

if i <= b then

Move(Source[i], Dest[p], (b - i + 1) \* SizeOf(Source[1]))

else

Move(Source[j], Dest[p], (c - j + 1) \* SizeOf(Source[1]));

end;

procedure MergeByLength(var Source, Dest: TArr; len: Integer);

var

a, b, c: Integer;

begin

a := 1;

b := len;

c := len shl 1;

while c <= n do

begin

Merge(Source, Dest, a, b, c);

a := a + len shl 1;

b := b + len shl 1;

c := c + len shl 1;

end;

if b < n then Merge(Source, Dest, a, b, n)

else

Move(Source[a], Dest[a], (n - a + 1) \* SizeOf(Source[1]));

end;

begin

Enter;

len := 1;

Flag := True;

FillChar(t, SizeOf(t), 0);

while len < n do

begin

if Flag then MergeByLength(k, t, len)

else MergeByLength(t, k, len);

len := len shl 1; Flag := not Flag;

end;

if not Flag then k := t;

PrintResult;

end; (\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

function MenuSelect: Integer;

var

ch: Integer;

begin

Clrscr;

WriteLn('Sorting Algorithms Demos; Input: SORT.INP; Output: SORT.OUT'); for ch := 0 to nMenu do WriteLn(SMenu[ch]);

Write('Enter your choice: '); ReadLn(ch); MenuSelect := ch;

end;

begin

repeat

selected := MenuSelect;

WriteLn(SMenu[selected]);

case selected of

0: PrintInput;

1: SelectionSort;

2: BubbleSort;

3: InsertionSort;

4: AdvancedInsertionSort;

5: ShellSort;

6: QuickSort;

7: HeapSort;

8: DistributionCounting;

9: RadixSort;

10: StraightRadixSort;

11: MergeSort;

12: Halt;

end;

until False;

end.