TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Đề tài: MÔ PHỎNG CÁC THUẬT TOÁN SẮP XẾP BẰNG GRAPHICS.H

Sinh viên thực hiện: Nhóm 09-13

21110940: Lê Thành Vinh

21110699: Lê Trần Hữu Trí

GVHD: TS.HUỲNH XUÂN PHỤNG

TIỂU LUẬN KẾT THÚC HỌC PHẦN

Học kỳ 1 /2022-2023

Mục lục:

Phần 1: Mở đầu:

I. Lý do chọn đề tài:

Ngày nay, với sự phát triển vượt bậc của khoa học kỹ thuật. Công nghệ thông tin là một lĩnh vực nhiều ứng dụng thiết thực nhất trong mọi lĩnh vực của cuộc sống và xã hội chúng ta, đặc biệt nó là công cụ hỗ trợ đắc lực không thể thiếu trong công tác tổ chức, quản lý. Dễ dàng thấy rằng cơ sở dữ liệu là một trong ứng dụng quan trọng của công tác tin học hoá với những ứng dụng hữu ích cho mọi lĩnh vực của cuộc sống. Nhờ vào công tác tin học hoá mà công tác quản lý, tổ chức sắp xếp dữ liệu tỏ ra có hiệu quả, nhanh chóng, chính xác, lưu trữ gọn, bảo mật cao và dễ dàng. Chính vì lẽ đó mà cơ sở dữ liệu như là một giải pháp hữu hiệu nhất cho mọi cơ quan, tổ chức có thể làm việc 1 cách khoa học và đạt năng suất cao.

Trong 2 thập niên qua, với sự phát triển không ngừng của ngành công nghệ thông tin, các ứng dụng của công nghệ thông tin ngày càng đi sâu vào các lĩnh vực của cuộc sống. nó giữ một vai trò cực kỳ quan trọng, góp phần giúp nhà quản lý thực hiện chức năng quản lý một cách khoa học, chính xác và hiệu quả. Sự phát triển của các công nghệ phận mềm, trên các lĩnh vực như đồ họa, ứng dụng văn phòng, tổ chức dữ liệu, thiết kế web .v.v.. Trong đó cấu trúc dữ liệu và giải thuật là một bộ môn giúp cho nhà lập trình xây dựng ý tưởng và thiết kế nên những phần mềm ứng dụng quan trọng, mà các giải thuật sắp xếp là một trong những ứng dụng quan trọng của bộ môn cấu trúc dữ liệu và giải thuật, nó giúp cho chúng ta có thể tổ chức sắp xếp dữ liệu theo nhiều phương án khác nhau, thuận tiện cho việc truy xuất, kiểm tra, đối chiếu dữ liệu, thế nên việc nghiên cứu các giải thuật sắp xếp để đưa vào ứng dụng trong cuộc sống là một nhu cầu cấp thiết mà người lập trình phải tìm hiểu và nghiên cứu. Vì vậy mà nhóm em lựa chọn đề tài số 5 để làm đồ án cuối kì này.

II. Mục đích của đề tài:

Đồ án này trong giới hạn kiến thức và thời gian, em chỉ tập trung nghiên cứu các nội dung cơ bản sau

- Nghiên cứu tổng quan về giải thuật.
- Nghiên cứu các thuật toán sắp xếp
- Thiết kế các giải thuật sắp xếp.
- Áp dụng kết quả nghiên cứu làm một demo cài đặt các giải thật sắp xếp

III. Phương pháp nghiên cứu:

- Phân chia công việc và làm việc theo nhóm
- Viết code bằng ngôn ngữ C/C++ để chạy thuật toán

Phần 2: Nội dung:

I. Các thuật toán:

1. Comparison Counting Sort:

- Bước 1: Tạo 1 mảng có kiểu dữ liệu là Record có **n** phần tử được đặt như sau (**Record rc[n])**, mỗi phần tử Record chứa 2 kiểu dữ liệu (char **Data; int Key)**
- Bước 2: Tạo 1 mảng có kiểu dữ liệu int đặt tên là Count có n phần tử được đặt như sau (**int Count[n]**), ban đầu ta xét Count[0] = Count[1] = … = Count[n-1] = 0.
- Bước 3: Tạo 1 vòng lặp đối với biến **int i**. Vòng lặp được bắt đầu áp dụng từ bước 4 cho tới bước 5, biến **i** sẽ được lặp từ **n-1** cho tới **i=1**.
- Bước 4: Tạo 1 vòng lặp cho biến **int j** bên trong vòng lặp cho biến **i**. **j** đi từ **i-1** cho tới khi **i=0**.
- Bước 5: Chúng ta so sánh **rc[i].Key và rc[j].Key**, nếu **rc[i].Key < rc[j].Key** thì **Count[j]+=1**. Ngược lại, nếu **rc[i].Key≥≥rc[j].Key** thì **Count[i]+=1**.
- Bước 6: Sau khi vòng lặp biến **i** ở trên kết thúc. Chúng ta đã thu được một mảng **Count[n]** có các giá trị đã được thay đổi. Cuối cùng, tạo 1 Record **result[n]** có **n** phần tử. Tạo 1 vòng lặp cho **k** từ **0 -> n**. Ta đặt **result[Count[k]] = rc[k]**.
- Bước 7: Kết thúc thuật toán, ta thu được 1 mảng **result[n]** đã được sắp xếp từ mảng **rc[n]** ban đầu

-> Như vậy thuật toán này sắp xếp dựa trên một mảng phụ và mảng phụ này có chức năng sẽ lưu lại vị trí xuất hiện của các phần tử trong mảng gốc dựa theo giá trị của chúng so với nhau.

2. Shell Sort:

- Bước 1: Tạo 1 mảng có liểu dữ liệu là Record có **n** phần tử được đặt như sau (**Record rc[n]**), mỗi phần tử Record chứa 2 kiểu dữ liệu (**char Data; int Key;**).

- Bước 2: Tạo biến **int x= n/2**. Sau đó tạo 1 mảng **int h[]** có **n/2** phần tử sao cho **h[0] =1, h[1]=2,…h[x-1]=n/2**. Mảng **h** này sẽ lưu các khoảng cách của các phần tử sẽ được so sánh với nhau trong mảng **rc[]**.

- Bước 3: Tạo vòng lặp với biến s và s đi từ (n/2 -1)=x cho tới 0

- Bước 4: Đặt biến **temp = h[s]**. Và tiếp theo ta sẽ tạo 1 vòng lặp với biến **j,** trong đó **j** đi từ temp cho tới **n-1**

- Bước 5: Trong vòng lặp cho biến **j**, ta đặt biến **int i=j-temp**, **int K = rc[j].Key, Record R = rc[j]**.

- Bước 6: Nếu **rc[i].Key** ≤≤ **K**, ta thực hiện trực tiếp bước 8.

- Bước 7: Đặt **rc[i+temp] = rc[i]**. Nếu **i>0** thì quay lại bước 6.

- Bước 8: Đặt **rc[i+temp] = R**. Sau đó quay lại bước 5.

-> Như vậy, Thuật toán này hoạt động bằng việc hoán đổi các phần tử cách nhau 1 khoảng cách nào đó. Sau khi hoàn thành 1 lượt sắp xếp, khoảng cách sẽ giảm đi 1 và nếu phù hợp điều kiện sẽ lại hoán đổi các phần tử theo khoảng cách mới. Khi sắp xếp xong thì khoảng cách sẽ trở về 0.

3. Quick Sort:

- Bước 1: Tạo 1 mảng có liểu dữ liệu là **Record** có **n** phần tử được đặt như sau (**Record rc[n]**), mỗi phần tử Record chứa 2 kiểu dữ liệu (**char Data; int Key;**), tạo 1 **Stack** có tên là **s**. Stack này sẽ có chức năng lưu trữ các node, mỗi **node** đều có chứa 2 giá trị là (**int l**, **int r**)

- Bước 2: Đặt biến l=0, r=n-1, s rỗng và M=1

- Bước 3: Đặt i=l, j = r+1 và K = rc[l].Key

- Bước 4: Chúng ta tăng i lên 1, sau đó kiểm tra nếu rc[i].Key < K thì lặp lại bước này

- Bước 5: Chúng ta giảm j đi 1, sau đó kiểm tra nếu rc[j].Key > K thì lặp lại bước này

- Bước 6: Kiểm tra xem nếu j ≤≤i thì tiến hành hoán đổi rc[l] và rc[j] và sau đó tới bước 8

- Bước 8: Nếu r-j ≥≥ j-l > M thì tiến hành push(node(j+1,r)) vào stack s và quay lại bước 3. Nếu j-l > r-j > M, tiến hành push(node(l,j-1)) vào stack s và quay lại bước 3. Nếu r-j > M ≥≥ j-l, ta gán l=j+1 rồi quay lại bước 3. Nếu j-l > M ≥ ≥ r-j thì gán r = j- 1 rồi quay lại bước 3

- Bước 9. Kiểm tra xem stack có rỗng hay không, nếu có thì thoát khỏi vòng lặp và kết thúc thuật toán.

- Bước 10. Nếu stack không rỗng thì tiến hành việc đặt Node* p = s.pop() . Đặt l = p->l, r= p->r. sau đó lại quay lại bước 3.

-> Thuật toán trên sắp xếp dựa trên việc hoán đổi các giá trị dựa theo vị trí từ các khoảng từ l tới r khác nhau. Trong đó khoảng l và r được cập nhật liên tục. Nếu khoảng nào phù hợp với điều kiện mà chưa được xét thì sẽ được đẩy vào đỉnh stack nhằm phục vụ cho việc hoán đổi trong các khoảng sau này. Khi thuật toán kết thúc thì đồng nghĩa với việc stack sẽ rỗng.

4. Selection Sort:

- Bước 1: Tạo 1 mảng có liểu dữ liệu là Record có **n** phần tử được đặt như sau (**Record rc[n]**), mỗi phần tử Record chứa 2 kiểu dữ liệu (**char Data; int Key;**).

- Bước 2: Tạo 1 vòng lặp với biến **j**, trong đó j đi từ **n-1** tới **1**

- Bước 3: Tiếp theo, ta tìm số lớn nhất trong khoảng từ **rc[j]** cho tới **0**. Bằng cách đặt 1 biến có tên là **max_idx =j**.

- Bước 4: Tạo 1 vòng lặp con có biến **i** trong vòng lặp của biến **j**, (**i = j-1** cho tới **0**). Nếu ta nhận thấy có **rc[i].Key>rc[max_idx]** thì đi tới bước 5

- Bước 5: Hoán đổi **rc[max_idx]** và **rc[i]**, sau đó đặt **max_idx = i**.

-> Thuật toán này hoạt động dựa trên việc kiểm tra từ vị trí n-1 tới vị trí 0. Trong quá trình kiểm tra đó, nó sẽ liên tục cập nhật vị trí có giá trị cao nhất và sau đó là hoán đổi vị trí đang xét và vị trí có giá trị cao nhất đó. Cứ liên tục làm như thế cho tới khi vị trí được xét trở thành 1

5. Two way Merge Sort:

- Bước 1: Tạo 2 mảng **Record** được đặt tên là **X** và **Y**, lần lượt có **m,n** phần tử. Lưu ý 2 mảng này đã được sắp xếp có trật tự. Đồng thời ta cũng sẽ tạo thêm 1 mảng **Record** được đặt tên là **Z** có **m+n** phần tử, mảng **Z** này sẽ là kết quả sắp xếp được khi trộn 2 mảng **X** và **Y** trên.

- Bước 2: Đặt các biến **i = j = k = 0**.

- Bước 3: So sánh nếu **X[i].Key** $\leq\leq$ **Y[i].Key** thì đi tới bước 4. Nếu không thì đi tới bước 6.

- Bước 4: Đặt **Z[k] = X[i], k = k+1, i=i+1**, nếu **i**$\leq\leq$ **m** thì quay lại bước 3

- Bước 5: Đặt **Z[k], … ,Z[m+n] = Y[j], … , Y[n]** và kết thúc thuật toán.
- Bước 6: Đặt **Z[k] = Y[j], k = k+1, j = j+1**, nếu **j≤ ≤ n** thì lại quay lại bước 3
- Bước 7: Đặt **Z[k], … ,Z[m+n] = X[i], … , X[m]** và kết thúc thuật toán.

-> Thuật toán này được thực hiện thông qua 2 mảng đã được sắp xếp sẵn, sau đó trộn 2 mảng trên theo trật tự để tạo nên được 1 mảng mới sẽ được sắp xếp

II. Mô tả các thuật toán bằng C++:

1. Comparison Counting Sort:

```
void ComparisonCountingSort(Record rc[], int n,int Count[])
{
  for (int i = n - 1; i >= 1; i--)
  {
    for (int j = i - 1; j >= 0; j--)
    {
      if (rc[i].key > rc[j].key)
      {
        Count[i]++;
      }
      else
      {
        Count[j]++;
      }
    }
  }
  // Output the result
  Record rs[MAX];
  for (int k = 0; k < n; k++)
  {
    rs[Count[k]] = rc[k];
  }
  // Copy the sorted array to the original array
  for (int i = 0; i < n; i++)
  {
    rc[i] = rs[i];
  }
}
```

2. Shell Sort:

```
void ShellSort(Record rc[], int n)
{
  int* h = new int[n];
  int x, temp, Key, i;
  Record record;
  x = n / 2;
  for (int i = x - 1; i >= 0; i--)
  {
    h[i] = x;
    x--;
  }
  int s;
  int gap = n / 2;
```

```
  for (s = (n / 2) - 1; s >= 0; s--)
  {
    temp = h[s];
    for (int j = temp; j < n; j++)
    {
      i = j - temp;
      Key = rc[j].key;
      record = rc[j];
      if ((i >= 0) && Key < rc[i].key)
      {
        while (i >= 0 & Key < rc[i].key)
        {
          mySwap(rc[i], rc[i + temp]);
          i = i - temp;
        }
      }
      else
      {
        rc[i + temp] = record;
      }
    }
    gap--;
  }
}
```

3. Quick Sort:

```
void QuickSort(Record rc[], int n, Stack s)
{
  s.init();
  Node* p = CreateNode(0, n - 1);
  s.push(p);
  while (!s.isEmpty())
  {
    p = s.pop();
    int l = p->l;
    int r = p->r;
    if (l < r)
    {
      int i = l;
      int j = r;
      Record x = rc[l];
      while (i < j)
      {
        while (i < j && rc[j].key >= x.key) j--;
        if (i < j) rc[i++] = rc[j];
        while (i < j && rc[i].key <= x.key) i++;
        if (i < j) rc[j--] = rc[i];
      }
      rc[i] = x;
      s.push(CreateNode(l, i - 1));
      s.push(CreateNode(i + 1, r));
    }
  }
}
```

4. Selection Sort:

```
void SelectionSort(Record rc[], int n)
{
  int i, j, max_idx;
  for (i = n - 1; i >= 1; i--)
  {
    max_idx = i;
    for (j = i - 1; j >= 0; j--)
    {
      if (rc[max_idx].key < rc[j].key)
      {
        max_idx = j;
      }
    }
    if (max_idx != i)
    {
      Record temp = rc[i];
      rc[i] = rc[max_idx];
      rc[max_idx] = temp;
    }
  }
}
```

5. Two Way Merge Sort:

```
void TwoWayMergeSort(Record X[], Record Y[], int nx, int ny, Record Z[])
{
  int i = 0, j = 0, k = 0;
  while (i < nx && j < ny)
  {
    if (X[i].key < Y[j].key)
    {
      Z[k] = X[i];
      i++;
    }
    else
    {
      Z[k] = Y[j];
      j++;
    }
    k++;
  }
  while (i < nx)
  {
    Z[k] = X[i];
    i++;
    k++;
  }
  while (j < ny)
  {
    Z[k] = Y[j];
    j++;
    k++;
  }
}
```

Phần 3: Kết luận:

Qua việc làm đồ án lần này, Nhóm chúng em cũng học được cách làm việc theo nhóm sao cho hiệu quả, thêm vào đó là học được các thuật toán mới và cũng học được các hướng giải quyết khác cho các thuật toán đó, giúp mở mang thêm kiến thức. Ngoài ra, nhóm còn học được cách tự học, và chọn lọc kiến thức thông qua việc tìm kiếm các thông tin mà mình cần được thực hiện.