TẠO RUN BAN ĐẦU CHO SẮP THỨ TỰ NGOẠI.

TÓM TẮT

Mục đích chính của bài báo là giới thiệu giải thuật tận dụng bộ nhớ chính để tạo ra các run dài hơn các run tự nhiên. Vì vậy, các giải thuật sắp thứ tự ngoại dựa trên việc phân phối và trộn run sẽ được thực hiện nhanh hơn.

NỘI DUNG

1. Tổng quan

Trong tin học, sắp thứ tự là sắp xếp lại dữ liệu theo một thứ tự nào đó để dễ dàng xử lý, tìm kiếm dữ liệu về sau. Các phương pháp sắp thứ tự chia làm 2 nhóm:

-Sắp thứ tự nội (hay sắp thứ tự mảng): toàn bộ dữ liệu cần sắp thứ tự phải được đưa vào bộ nhớ chính. Do đó, dữ liệu cần sắp xếp thứ tự không nhiều lắm vì bị giới hạn bởi dung lượng bộ nhớ chính trên từng máy tính cụ thể.

-Sắp thứ tự ngoại (hay sắp thứ tự tập tin): dữ liệu cần sắp xếp được lưu trữ ở trên bộ nhớ ngoài (băng từ, đĩa từ,…). Dó đó, dữ liệu cần sắp xếp thứ tự tương đối lớn, thời gian truy xuất tương đối chậm.

Phương pháp thường dùng để sắp thứ tự ngoại là phân phối các *run* ở tập tin cần sắp xếp thành các tập tin trung gian nhỏ hơn, sau đó trộn các run này lại và đưa vào tập tin ban đầu. Run (đường chạy) là một dãy các mẩu tin mà khóa của chúng có thứ tự không giảm. Từ các run ban đầu của tập tin cần sắp xếp, quá trình phân phối và trộn run sao cho tập tin chỉ còn lại một run thì tập tin cuối cùng có thứ tự. Nếu phân phối n run vào 2 tập tin trung gian và sau đó trộn run lại từ 2 tập tin này thì số lượng run sẽ giảm đi một nữa. Sau $k=[\log_2 n]$ lần phân phối và trộn, quá trình sắp xếp kết thúc. Thời gian sắp xếp không chỉ phụ thuộc vào kích thước dữ liệu mà còn phụ thuộc số lượng run ban đầu. Các run tự nhiên có sẵn trong tập tin cần sắp xếp thường có kích thước nhỏ nên số lượng run ban đầu rất lớn. Nếu dựa vào run tự nhiên để phân phối và trộn thì thời gian sắp xếp chậm. Bài báo này trình bày một giải thuật sử dụng một phần bộ nhớ chính làm bộ đệm trung gian để tạo ra các run ban đầu cho sắp thứ tự ngoại. Giải thuật này sẽ tạo ra các run có kích thước lớn hơn rất nhiều so với kích thước bộ đệm được cấp phát trong bộ nhớ chính.

2. Nội dung giải thuật

Khi được cấp phát một bộ đệm trên bộ nhớ chính, nếu chỉ đọc các mẩu tin từ tập tin cần sắp xếp vào bộ đệm rồi dùng các giải thuật sắp thứ tự nội để sắp xếp trên bộ đệm này thì run cũng được tạo ra nhưng kích thước của run được tạo ra khi đó là cố định và bằng kích thước bộ đệm. Để tạo ra run có kích thước lớn hơn, bộ đệm được tổ chức lại thành 2 heap: heap đầu tiên ở đầu bộ đệm chứa các mẩu tin thuộc run hiện tại, heap còn lại ở cuối bộ đệm chứa các mẩu tin thuộc run kế tiếp. Vì vậy, quá trình đọc mẩu tin và tạo run được thực hiện liên tục.

Heap là một định nghĩa cơ bản được sử dụng trong giải thuật heapsort, một giải thuật sắp thự tự nội. Mảng a[i], a[i+1],… a[j] là một heap nếu a[k] ≤ a[2k+1] và a[k] ≤ a[2k+2] với mọi k = i, i+1,…,[j/2]. Có hai nhận xét về heap như sau:

-Đối với mảng a[0], a[1],… a[n-1] thì mảng con a[j], a[j+1],… a[n-1] với j>[n/2]+1 cũng là một heap. Do đó, chỉ cần thực hiện phép toán shift cho các phần tử a[i] với 0≤i≤[n/2]+1 để tạo heap cho mảng a.

-Mảng a[0], a[1],… a[n-1] là một heap thì a[0] là nhỏ nhất trong mảng. Nếu b được đọc từ tập tin và b≥a[0] thì b thuộc run hiện tại, ngược lại b thuộc run kế tiếp.

Xét run hiện tại đang nằm trên heap đầu của bộ đệm thì mẩu tin đầu tiên là mẩu tin nhỏ nhất sẽ được ghi xuống tập tin. Vùng nhớ trống sau khi ghi được tận dụng để đọc mẩu tin kế tiếp để làm tăng chiều dài của run. Nếu mẩu tin mới lớn hơn hoặc bằng mẩu tin vừa ghi thì mẩu tin mới thuộc run hiện tại ở heap đầu, phải thực hiện phép toán shift tạo lại heap đầu để tìm mẩu tin nhỏ kế tiếp thuộc run hiện tại. Ngược lại, mẩu tin mới sẽ thuộc run kế tiếp, phải thực hiện phép toán shift tạo heap sau.

Các bước của giải thuật tạo run như sau:

Bước 1: Đọc m mẩu tin từ tập tin vào bộ đệm, tạo heap đầu tiên. (m là kích thước bộ đệm buffer).

Bước 2: Cho q=m-1 và lặp bước 3 cho đến khi hết tập tin.

Bước 3: Ghi mẩu tin đầu tiên buffer[0] xuống tập tin trung gian.

Đọc mẩu tin kế tiếp b;

Nếu b≥ buffer[0] thì buffer[0] = b và gọi shift tạo lại heap đầu.

Ngược lại thì buffer[0] = buffer[q], buffer[q]=b, q=q-1; gọi shift tạo lại heap đầu, gọi shift tạo heap sau.

Nếu q<0 thì q= m-1; (Hết run hiện tại ở heap đầu, xét run kế tiếp trong heap sau).

Bước 4: Ghi run còn lại trong bộ đệm xuống tập tin trung gian.

3. Hiện thực giải thuật

Chương trình viết bằng ngôn ngữ C sau đây sẽ hiện thực giải thuật tạo run đối với tập tin chứa các số nguyên. Giả sử bộ nhớ chính có thể cấp phát bộ đệm buffer có kích thước là BUFSIZE.

Trong giải thuật tạo run, hàm shift sẽ thực hiện tạo ra heap mới từ heap nhỏ hơn. Giả sử a[q+1], a[q+2],…, a[r] là một heap thì hàm shift(a, q, r) sẽ tạo ra heap mới là a[q], a[q+1],…,a[r]. Nếu thực hiện [n/2] lần hàm shift cho [n/2] phần tử đầu của mảng a thì a[0],a[1],.., a[n-1] là một heap. Khi đó, a[0] là phần tử nhỏ nhất của mảng a. Hàm shift được viết như sau:

```
void shift(int a[],int q,int r)
{     int i=q,j=2*i+1,x=a[i];
      while (j<=r)
      {     if ((j<r)&&(a[j]>a[j+1]))
                  j++;
            if (x<a[j])
                  break;
```

```
            else
            {     a[i]=a[j];
                  i=j;
                  j=i*2+1;
            }
      }
      a[i]=x;
}
```

Hàm MakeRun sẽ thực hiện việc tạo run cho tập tin fi và phân phối các run vào các tập tin trung gian fo. Buffer được tổ chức thành 2 heap: heap đầu là buffer[0], buffer[1], … buffer[q] và heap sau là buffer[q+1], buffer[q+2],…, buffer[m-1]. Ở bước 1 của giải thuật, heap đầu chiếm cả bộ đệm buffer, heap sau là rỗng. Hàm MakeRun vừa tạo run và vừa phân phối run được viết như sau:

```
`void MakeRun(FILE * fi,FILE * fo[],int numfile)
{     int n=0,k=0,i,q,r,m,mh;
      int* buffer=(int*)malloc(BUFSIZE*sizeof(int));
      int b;
      while (n<BUFSIZE) /* Step 1  */
            if (ReadElement(fi,&buffer[n]))
                  n++;
            else
                  break;
      m=min(BUFSIZE,n);
      mh=m/2;
      i=mh-1;
      while (i>=0)
      {     shift(buffer,i,m-1);
            i--;
      }
      q = m-1;          /*  Step 2 */
      while (ReadElement(fi,&b))
      {     n++;        /*  Step 3 */
            WriteElement(fo[k],buffer[0]);
            if (buffer[0]<b)  /* Current run */
            {     buffer[0]=b;
                  shift(buffer,0,q);
            }
            else              /* Next run */
            {     buffer[0]=buffer[q];
                  shift(buffer,0,q-1);
                  buffer[q]=b;
                  if (q<mh)
                        shift(buffer,q,m-1);
                  q=q-1;
                  if (q<0)    /* End current run, swap heap. */
                  {     q=m-1;
                        k++;
                        if (k>=numfile)   k-=numfile;
                  }
            }
      }
      r=m-1;  /*  Step 4 */
      while (q>=0)
      {     WriteElement(fo[k],buffer[0]);
            buffer[0]=buffer[q];
            shift(buffer,0,q-1);
            buffer[q]=buffer[r];
            r=r-1;
            if (q<mh)
```

```
                    shift(buffer,q,r);
                q=q-1;
            }
            k++;
            if (k>=numfile)   k-=numfile;
            while (r>=0)
            {     WriteElement(fo[k],buffer[0]);
                  buffer[0]=buffer[r];
                  r=r-1;
                  shift(buffer,0,r);
            }
      }
```

Giả sử tập tin fi có 110 phần tử với 54 run tự nhiên như sau:

```
22 43 11 80 10 94 13 74 2 63 32 29 25 81 30 0 50 54 84 9 44 59 68 38 50 93 47 41 47 17 61 53
14 81 99 87 59 33 82 87 97 75 82 19 96 58 48 0 66 19 23 16 54 14 20 25 63 31 70 80 41 92 8 2
41 34 52 39 11 49 92 4 38 77 0 19 46 97 95 62 29 69 27 98 78 51 26 55 47 69 48 18 36 52 92 45
51 22 79 64 19 77 76 21 82 26 97 30 52 19
```

Với BUFSIZE = 10, sau khi gọi MakeRun(fi, fo, 2) thì các run tạo ra như sau:

- tập tin fo[0] có 3 run:

```
2 10 11 13 22 25 29 30 32 43 50 54 59 63 68 74 80 81 84 93 94    0 14 16 19 19 20 23 25 31 33
41 48 54 58 63 66 70 80 92 92   0 18 26 27 29 36 45 47 48 51 51 52 55 64 69 76 77 79 82 92 97
```

- tập tin fo[1] có 3 run:

```
0 9 14 17 38 41 44 47 47 50 53 59 61 75 81 82 82 87 87 96 97 99   2 4 8 11 19 34 38 39 41 46
49 52 62 69 77 78 95 97 98    19 19 21 22 26 30 52
```

4. Kết luận

Việc sử dụng heap trong giải thuật tạo và phân phối run sẽ đạt được hiệu quả cao nhất trong phương pháp trộn đa pha. Đối với tập tin có các mẩu tin phân bố ngẫu nhiên thì chiều dài trung bình của run tự nhiên là 2. Nếu sử dụng bộ đệm có kích thước m thì chiều dài trung bình của run tạo ra là 2m. Với phương pháp trộn đa pha có số tập tin trung gian n=6, kích thước bộ đệm m=100 và tập tin có 1656801 mẩu tin thì sau 20 pha trộn, số run còn lại khoảng 100 run. Giải thuật tạo run còn được sử dụng để xây dựng chiến lược sao lưu dữ liệu, nhập liệu và sắp thứ tự đồng thời,….

TÀI LIỆU THAM KHẢO

[1]. Donald E. Knuth, *The Art of Computer Programming*, Volume 3, Addison Wesley, 1998.

[2]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, MIT Press, 2001.

[3]. Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft, *Data Structures and Algorithms*, Addison Wesley, 1983.