[백준] 합병정렬과 퀵정렬

imhyun · 방금 전

알고리즘 정렬 알고리즘 퀵정렬 합병정렬

https://www.acmicpc.net/problem/11650

문제

> 이 문제는 X좌표가 증가함에 따라 정렬하되, X좌표가 같다면 Y좌표가 증가하는 순으로 좌표를 정렬하는 문제이다. **X좌표를 기준으로 정렬하되, 같을 때 Y좌표를 기준으로 정렬**하는 알고리즘을 짜야 한다.
> 합병정렬을 통해 구현해보자.

개념

분할 통치법

> 어떤 명령 주어지면, 그 일을 쪼개 하위 부서들한테 토스하는 방법. 일을 **분할하고** 같은 일들을 반복하여**(재귀)** 그 일을 **통치**하는 방법이다.
> 분할 통치법을 사용하는 정렬 알고리즘에는 **합병 정렬(merge-sort)**, **퀵 정렬(quick-sort)**가 있다.

📏 순서

분할

- 입력된 데이터 L을 둘 이상의 분리된 **부분 집합 L1,L2로 나눈다.**

재귀

- 부분 집합 L1,L2에 대한 **부문제를 재귀적**으로 해결
- **베이스 케이스가 상수 크기의 부문제**로 귀속되도록!!

통치

- 부문제들의 해결을 **합쳐 L을 해결**

합병 정렬

> 언뜻 보기에, 분할하는 과정이 중요한 거 같지만 실제로는 **합병**하는 과정이 더 중요하다!!

기본

- **분할 통치법**에 기초한 정렬 알고리즘
- 힙정렬
  - **비교**에 기초한 정렬
  - **O(nlogn)** 시간에 수행
- 힙정렬과는 달리
  - 외부의 **우선순위 큐**를 사용하지 **않음.**
  - 데이터를 **순차적** 방식으로 접근(고용량 데이터에 접근하기 용이함.)
- 이진트리로 도식화하며 이해할 수 있다.

알고리즘

```
Alg mergeSort(L)
    input list L with n elements
    output sorted list L

1. if(L.size()>1)
            //**분할 (상수시간 소요)**
            L1,L2<-partition(L,n/2)

            //**재귀**적 정렬
            mergeSort(L1)
            mergeSort(L2)

            //순서리스트로 **합병**
            L<-merge(L1,L2)

2.return
```

두 개의 정렬 리스트 합병하기(merge함수)

- 이미 정렬된 순서리스트를 합병한 순서리스트를 만들기 위한 과정.
- **O(n)**시간의 소요

```
Alg merge(L1,L2):
    input sorted list L1 and L2 with n/2 elements each
    output sorted list of L1 U L2

1. L<-empty list
2. while(!L1.isEmpty()&!L2.isEmpty())
            if(L1.get(1)<=L2.get(1))
                L.addLast(L1.removeFirst())
            else
                L.addLast(L2.removeFirst())

3.while(1.L1.isEmpty())
        L.addLast(L1.removeFirst())

4.while(1.L2.isEmpty())
        L.addLast(L2.removeFirst())

5.return L
```

합병 정렬 분석

- 합병 정렬 트리의 높이 : **O**(log**n**)
- 각 깊이에서 이뤄지는 작업량 : **O**(**n**)
- 최종적으로 이뤄지는 시간은 **O**(**n**log**n**)

배열로 합병 정렬 구현하기

- 배열을 임시 공간으로 활용한다는 아이디어를 갖고 알고리즘을 짰다.
  아래는 크기가 8인 배열 임시로 설정해서 짠 코드이다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma warning (disable:4996)

int partition(int, int);
void merge(int*,int,int,int);
void merge_sort(int*,int,int);

int main(void)
{
    int A[8] = { 6,1,8,2,3,5,7,4 };

    merge_sort(A,0,7);

    for (int i = 0; i <= 7; i++)
        printf("%d ", A[i]);

}

int partition(int l, int r) {
    return (l + r) / 2;
}

void merge(int* A, int l, int r, int m) {
    int B[8];
```

```
    int i=l;
    int j=m+1;
    int k = l;


    while ((i <= m) && (j <= r)) {
        if (A[i] < A[j])
            B[k++] = A[i++];
        else
            B[k++] = A[j++];
    }

    while (i <= m ) {
        B[k++] = A[i++];
    }

    while (j <= r) {
        B[k++] = A[j++];
    }

    for (i = l; i <= r; i++)
        A[i] = B[i];
}

void merge_sort(int* A,int l,int r) {

    if (l < r) {
        //분할
        int m = partition(l, r);

        //재귀
        merge_sort(A, l, m);
        merge_sort(A, m + 1, r);

        //합병
        merge(A, l, r, m);
    }

    else
        return;
}
```

첫시도

아이디어

X,Y가 한 번에 교환되어야 하니 이중배열을 쓸까 하다 이중배열과 함수 사이에 연결하는 부분이 헷갈릴 거 같아 **Coordinate이라는 구조체를 만들어 X,Y 좌표를 한 번에 관리**했다.

X와 Y좌표 모두 서로 연결이 되어야 하기 때문에(교환되면 같이 교환) X,Y를 나누어 합병 정렬을 하도록 하였다. 이때 merge는 기본 합병 정렬에서 X,Y와 함께 교환하도록 하는 식과 X가 같을 때 Y의 크기에 따라 교환되는 알고리즘을 추가 작성해주었다.

이 과정에서 교환할 때 저장할 A,B 배열까지 사용하니 일반 합병 정렬에 비해 2배 정도의 메모리를 사용했다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma warning (disable:4996)

//좌표
typedef struct coordinate {
    int num;
    int* X;
    int* Y;
}coordinate;

//좌표 구조체
void init(coordinate*,int);

//합병정렬
int partition(int, int);
void merge(coordinate*, int, int, int);
void merge_sort(coordinate*, int, int);

int main(void)
{
    int N;
    coordinate C;

    scanf("%d", &N);

    init(&C,N);

    for (int i = 0; i < N; i++)
        scanf("%d %d",&C.X[i], &C.Y[i]);

    merge_sort(&C, 0, N-1);

    for (int i = 0; i < N; i++)
        printf("%d %d\n",C.X[i],C.Y[i]);

    free(C.X);
    free(C.Y);

}

void init(coordinate* C,int N) {
    C->num =N;
    C->X = (int*)malloc(sizeof(int) * N);
    C->Y = (int*)malloc(sizeof(int) * N);

}

int partition(int l, int r) {
    return (l + r) / 2;
}

void merge(coordinate* C, int l, int r, int m) {
    /**/
    int n = C->num;
```

```
        int* A = (int*)malloc(sizeof(int) * n);
        int* B=(int*)malloc(sizeof(int)*n);

        int i = l;
        int j = m + 1;
        int k = l;


        while ((i <= m) && (j <= r)) {
            if (C->X[i] < C->X[j]) {
                A[k] = C->X[i];
                B[k++] = C->Y[i++];

            }

            else if(C->X[i] > C->X[j]){
                A[k] = C->X[j];
                B[k++] = C->Y[j++];
            }

            else {
                if (C->Y[i] < C->Y[j]) {
                    A[k] = C->X[i];
                    B[k++] = C->Y[i++];
                }
                else {
                    A[k] = C->X[j];
                    B[k++] = C->Y[j++];
                }

            }
        }

        while (i <= m) {
            A[k] = C->X[i];
            B[k++] = C->Y[i++];
        }

        while (j <= r) {
            A[k] = C->X[j];
            B[k++] = C->Y[j++];
        }

        for (i = l; i <= r; i++) {
            C->X[i]= A[i] ;
            C->Y[i] = B[i];
        }

        free(A);
        free(B);
    }

    void merge_sort(coordinate* C, int l, int r) {

        if (l < r) {
            //분할
            int m = partition(l, r);

            //재귀
            merge_sort(C, l, m);
            merge_sort(C, m + 1, r);
```

```
            //합병
            merge(C, l, r, m);
        }

        else
            return;
    }
```

오래 걸린 부분

1. merge에서 i,j,k 인덱스 두번 ++함
2. 함수는 인덱스를 넣어주는데 N을 넣어줌. (N-1이 올바름)
3. 결과는 메모리 초과...=>free하니까 해결
4. 근데 시간초과...??

코드 문제점과 해결방안

합병 정렬을 X,Y 두 번 진행하는 것과 비슷하고 또한 이 과정에서 X,Y 배열 2개 임시 저장 공간 A,B 2개를 쓰니 메모리에서 많이 나갈 뿐만 아니라 할당하고 해제하는 시간 때문인지 **시간 초과**가 떴다.

퀵정렬을 아직 학습하지 않은 상태라 최대한 합병정렬 내에서 해결하기 위해서 퀵정렬과 합병정렬의 차이점을 먼저 알아보았고 그 결과 **합병정렬은 배열보다는 연결리스트에서 유리**하다는 것을 알았다. 따라서 연결리스트로 합병정렬 기본 코드를 완성했다. 하지만 배열을 통해서는 같은 인덱스를 옮겨준다는 아이디어로 X,Y를 관리했지만 연**결리스트는 같은 인덱스로 통제할 수 없이 분리**되어 있기 때문에 기본 코드만 작성하고 해결하지는 못 하였다...

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#pragma warning (disable:4996)

typedef struct ListNode {
    int elem;
    struct Node* next;
}ListNode;

typedef struct LinkedList {
    ListNode* H;
    int size;
}LinkedList;

//연결 리스트 형성
void init(LinkedList*);
ListNode* makeNode(LinkedList*, int);
void insertLast(LinkedList*, int);
void print(LinkedList*);

//merge sort 연결 리스트 버전
void partion(LinkedList*,LinkedList *, LinkedList*);
LinkedList merge_sort(LinkedList*);
LinkedList merge(LinkedList*, LinkedList*);
```

```
int main(void)
{
    LinkedList L;

    init(&L);

    int n;
    int e;
    scanf("%d", &n);

    L.size = n;

    for (int i = 0; i < n; i++) {
        scanf("%d", &e);
        insertLast(&L, e);
    }

    L=merge_sort(&L);

    print(&L);

}

//연결 리스트 기본 함수

void init(LinkedList* L) {
    L->H = NULL;
    L->size = 0;
}

void insertLast(LinkedList* L, int e) {
    ListNode* p = makeNode(L, e);
    ListNode* q = L->H;

    if (L-> H== NULL) {
        L->H = p;
    }
    else {
        while (q->next != NULL)
            q = q->next;

        q->next = p;
    }
}

ListNode* makeNode(LinkedList* L, int e) {
    ListNode* node = (ListNode*)malloc(sizeof(ListNode));
    node->elem = e;
    node->next = NULL;

    return node;
}

void print(LinkedList* L) {
    ListNode* p = L->H;

    for (int i = 0;i<L->size; i++) {
        printf("%d -> ", p->elem);
        p = p->next;
    }
    printf("\b\b\b   \n");
```

```
}

//merge_sort 연결 리스트 ver.
void partion(LinkedList* L, LinkedList* L1, LinkedList* L2) {
    int m = (L->size) / 2;

    //L1 형성
    L1->H = L->H;
    L1->size = m;

    //L2 형성
    ListNode* p = L->H;

    for (int i = 0; i < m; i++)
        p = p->next;

    L2->H = p;
    L2->size = L->size - m;
}

LinkedList merge_sort(LinkedList * L) {

    if (L->size > 1) {
        LinkedList L1, L2;

        init(&L1);
        init(&L2);

        partion(L, &L1, &L2);

        L1 =merge_sort(&L1);
        L2 = merge_sort(&L2);

        LinkedList fin = merge(&L1,&L2);

        return fin;
    }

}

LinkedList merge(LinkedList* L1, LinkedList* L2) {
    LinkedList L;

    init(&L);

    //비교
    ListNode* p = L1->H;
    ListNode* q = L2->H;

    int i = 1;
    int j = 1;

    L.size = L1->size + L2->size;

    //Head 설정
    if (p->elem < q->elem) {
        L.H = p;
        i++;
        p = p->next;
    }

    else {
```

```
            L.H = q;
            j++;
            q = q->next;
        }

        ListNode* r = L.H;

        while ((i <= L1->size) && (j <= L2->size)) {
            if (p->elem < q->elem) {
                r->next = p;
                i++;
                p = p->next;
            }

            else {
                r->next = q;
                j++;
                q = q->next;
            }
            r = r->next;

        }

        while (i <= L1->size) {
            r->next = p;
            i++;
            r = r->next;
            p = p->next;

        }

        while (j <= L2->size) {
            r->next = q;
            j++;
            r = r->next;
            q = q->next;
        }

        return L;
    }```
```

박시현

이전 포스트
[백준] 힙과 힙정렬

0개의 댓글

댓글을 작성하세요

댓글 작성