

TopK问题

思路：

1. 排序O(N\*logN)
2. 堆

把N个数建成小堆，选一个删一个，不断选出前K个最小。

此时的时间复杂度为O(N+O(logN\*k))

代码：

/\*\*

 \* Note: The returned array must be malloced, assume caller calls free().

 \*/

 typedef int HPDataType;

typedef struct Heap

{

    HPDataType\* a;

    int size;

    int capacity;

}Heap, HP;

void HeapInit(HP\* php, HPDataType\* a, int n);

void HeapDestroy(HP\* php);

void HeapPush(HP\* php, HPDataType x);

void HeapPop(HP\* php);

HPDataType HeapTop(HP\* php);

int HeapSize(HP\* php);

bool HeaoEmpty(HP\* php);

void Swap(int\* p1, int\* p2);

void AdjustDown(int\* a, int n, int parent);

void HeapPrint(HP\* php);

void AdjustUp(HPDataType\* a, int child);

void Swap(int\* p1, int\* p2)

{

    int tmp = \*p1;

    \*p1 = \*p2;

    \*p2 = tmp;

}

void AdjustDown(int\* a, int n, int parent)

{

    int child = parent \* 2 + 1;

    while (child < n)

    {

        if (a[child] + 1 < n && a[child] + 1 < a[child])  //(a[child] + 1 < n && a[child] + 1 < a[child])

        {

            ++child;

        }

        if (a[child] < parent)  //(a[child] < parent)

        {

            Swap(&a[parent], &a[child]);

            parent = child;

            child = parent \* 2 + 1;

        }

        else

        {

            break;

        }

    }

}

void AdjustUp(HPDataType\* a,int child)

{

    int parent = (child - 1) / 2;

    while (child > 0)

    {

        if (a[child] >= a[parent])

        {

            Swap(&a[child], &a[parent]);

            child = parent;

            parent = (child - 1) / 2;

        }

        else

        {

            break;

        }

    }

}

void HeapInit(HP\* php, HPDataType\* a, int n)

{

    assert(php);

    php->a = (HPDataType)malloc(sizeof(HPDataType) \* n);

    if (php->a == NULL)

    {

        printf("malloc fail\n");

        exit(-1);

    }

    memcpy(php->a, a, sizeof(HPDataType) \* n);

    php->size = n;

    php->capacity = n;

    //建堆,建设一个小堆

    for (int i = (php->size - 1 - 1) / 2; i >= 0; --i)

    {

        AdjustDown(php->a, php->size, i);

    }

}

void HeapDestroy(HP\* php)

{

    assert(php);

    free(php->a);

    php->a = NULL;

    php->size = php->capacity = 0;

}

void HeapPush(HP\* php, HPDataType x)

{

    //满了需要增容

    if (php->size == php->capacity)

    {

        HPDataType\* tmp = (HPDataType\*)realloc(php->a, php->capacity \* 2 \* sizeof(HPDataType));

        if (tmp == NULL)

        {

            printf("realloc fail\n");

            exit(-1);

        }

        php->a = tmp;

        php->capacity \*= 2;

    }

    php->a[php->size] = x;

    php->size++;

    AdjustUp(php->a, php->size);

}

void HeapPop(HP\* php)

{

    assert(php);

    assert(php->size > 0);

    Swao(&php->a[php->size - 1], &php->a[0]);

    //删掉换到最后的原堆数据

    php->size--;

}

HPDataType HeapTop(HP\* php)

{

    assert(php);

    assert(php->size > 0);

    return php->a[0];

}

int HeapSize(HP\* php)

{

    assert(php);

    return php->size;

}

bool HeaoEmpty(HP\* php)

{

    assert(php);

    return php->size == 0;

}

void HeapPrint(HP\* php)

{

    for (int i = 0; i < php->size; ++i)

    {

        printf("%d ", php->a[i]);

    }

    printf("\n");

}

int\* getLeastNumbers(int\* arr, int arrSize, int k, int\* returnSize){

    HP hp;

    HeapInit(&hp,arr,arrsize);

    int\* retArr = (int\*)malloc(sizeof(int)\*k);

    for(int i = 0;i<k;++i)

    {

        retArr[i] = HeapTop(&hp);

        HeapPop(&hp);

    }

    HeapDestory(&hp);

    \*returnSize = k;

    return retArr;

}

这时也会存在一个问题，虽然时间复杂度比较低，但是如果数据非常大的话，占用的空间复杂度会非常的大，如果不用文件读取的形式，会非常的浪费空间，所以这里还需要想一个方法。

1. 先把数组中前K个数据建成一个大堆。
2. 然后将剩下的N-K个数据跟堆顶的数据进行比较，如果比堆顶的数据小，则替换堆顶的数，调堆。
3. 最后得到的就是前K个最小的数。

代码：

int\* getLeastNumbers(int\* arr, int arrSize, int k, int\* returnSize){

   if(k == 0)

    {

        \*returnSize = 0;

        return NULL;

    }

    int\* arrRet = (int\*)malloc(sizeof(int)\*k);

    //前K个数建大堆

    for(int i = 0; i < k; ++i)

    {

        arrRet[i] = arr[i];

    }

    for(int j = (k-1-1)/2;j>=0;--j)

    {

        AdjustDown(arrRet,k,j);

    }

    //剩下的N-K个数,比堆顶的小,就替换堆顶数据,进堆

    for(int i =k;i<arrSizel++i)

    {

        if(arr[i]<arrRet[0])

        {

            arrRet[o] = arrp[i];

            AdjustDown(arrRet,k,0);

        }

    }

    \*returnSize = k;

    return arrRet;

}