K路归并

笔记本:	0_leetcode		
创建时间:	2020/8/28 9:49	**更新时间:**	2020/8/28 23:21
作者:	415669592@qq.com		
URL:	https://www.educative.io/courses/grokking-the-coding-interview/myAqDMyRXn3		

K路归并

0. 介绍

关键词：给定K个有序链表。使用堆进行合并。

1. 入门

> 合并K个有序链表

思路

直接使用堆来进行合并。

1点点分析：
假设总元素数目是N，共有K条链表。如果我们直接将所有元素放到一个数组中，并进行排序，时间复杂度是O(N * logN)

如果采用堆的话：
我们建立大小为K的堆。遍历N个元素，其时间复杂度是O(N * logK)。原因是我们利用了每条链表都有序的这个条件。

K路归并当然不止这个好处，其空间复杂度是有可以操作的，这个在后续谈到。

```
using namespace std;

#include <iostream>
#include <queue>
#include <vector>

class ListNode {
  public:
    int value = 0;
    ListNode *next;

    ListNode( int value ) {
        this->value = value;
        this->next = nullptr;
    }
};

class Compare {
  public:
    bool operator()( const std::pair<ListNode *, int> &x1, const
std::pair<ListNode *, int> &x2 ) {
        return x1.first->value > x2.first->value;
    }
};
class MergeKSortedLists {
  public:
    static ListNode *merge( vector<ListNode *> &lists ) {
        ListNode dummy = ListNode{ 0 };
```

```
        ListNode *scan = &dummy;
        priority_queue<std::pair<ListNode *, int>, vector<std::pair<ListNode
*, int>>, Compare> min_heap;
        for ( int i = 0; i < lists.size(); i++ ) {
            min_heap.push( make_pair( lists[0], i ) );
        }
        while ( !min_heap.empty() ) {
            auto node = min_heap.top();
            min_heap.pop();
            scan->next = node.first;
            scan = scan->next;
            if ( lists[node.second] ) {
                auto record = lists[node.second];
                lists[node.second] = lists[node.second]->next;
                min_heap.push( make_pair( record, node.second ) );
            }
        }
        scan->next = nullptr;
        // TODO: Write your code here
        return dummy.next;
    }
};

int main( int argc, char *argv[] ) {
    ListNode *l1 = new ListNode( 2 );
    l1->next = new ListNode( 6 );
    l1->next->next = new ListNode( 8 );

    ListNode *l2 = new ListNode( 3 );
    l2->next = new ListNode( 6 );
    l2->next->next = new ListNode( 7 );

    ListNode *l3 = new ListNode( 1 );
    l3->next = new ListNode( 3 );
    l3->next->next = new ListNode( 4 );

    ListNode *result = MergeKSortedLists::merge( vector<ListNode *> {l1, l2,
l3} );
    cout << "Here are the elements form the merged list: ";
    while ( result != nullptr ) {
        cout << result->value << " ";
        result = result->next;
    }
    system( "pause" );
```

```
}
```

单机内存不够时的K路归并。

2. 实战

2.1 K路有序链表中第M个最小元素

> 给定K个有序链表，获得第M小的元素。

e.g.1

```
Input: L1=[2, 6, 8], L2=[3, 6, 7], L3=[1, 3, 4], K=5
Output: 4
Explanation: The 5th smallest number among all the arrays is 4, this can be
verified from the merged list of all the arrays: [1, 2, 3, 3, 4, 6, 6, 7, 8]
```

思路

额，这个思路和入门一模一样。直接能给出时间复杂度O(M * logK)

和上题唯一的区别是链表换成了数组而已。

```
using namespace std;

#include <iostream>
#include <queue>
#include <vector>

struct Item {
    int value;
    int which_list;
    int index;
};

class Compare {
```

```
  public:
    bool operator()( const Item &x1, const Item &x2 ) {
        return x1.value > x2.value;
    }
};
class KthSmallestInMSortedArrays {
  public:
    static int findKthSmallest( const vector<vector<int>> &lists, int k ) {
        int result = -1;
        priority_queue<Item, vector<Item>, Compare> min_heap;
        // TODO: Write your code here
        for ( int i = 0; i < lists.size(); i++ ) {
            min_heap.push( Item{lists[i][0], i, 0} );
        }
        while ( k-- ) {
            auto min_elem = min_heap.top();
            min_heap.pop();
            result = min_elem.value;
            if ( min_elem.index + 1 < lists[min_elem.which_list].size() ) {
                min_heap.push( Item{ lists[min_elem.which_list]
[min_elem.index+1], min_elem.which_list, min_elem.index + 1 } );

            }
        }
        return result;
    }
};

int main( int argc, char *argv[] ) {
    vector<vector<int>> lists = {{2, 6, 8}, {3, 6, 7}, {1, 3, 4}};
    int result = KthSmallestInMSortedArrays::findKthSmallest( lists, 5 );
    cout << "Kth smallest number is: " << result;
    system( "pause" );
}
```

相似题目

1. 给定M个有序数组。找所有元素的中位数。
2. 合并K个有序数组。

2-2 矩阵中第K大值

> 给定N * N的有序矩阵。行和列均为升序。寻找第K大的元素。

e.g.1

```
Input: Matrix=[    [2, 6, 8],     [3, 7, 10],    [5, 8, 11]  ],   K=5
Output: 7
Explanation: The 5th smallest number in the matrix is 7.
```

思路

```
2   6   8
3   7  10
5   8  11
```

比2大的数字是其右边和下边。将这俩个数字加入最小堆中。
选取3。将3的右边和下边再次加入堆中。
然后取5， 然后取6。发现存在重复取7的问题。因此需要标记。

大概思路有了。

俩件事情

1. 记录row，col坐标，方便知道下次选取值。
2. 记录哪些值被选取。之后不再选取。

```cpp
using namespace std;

#include <iostream>
#include <queue>
#include <vector>

struct Point {
    int value;
    int row;
    int col;
};

class Compare {
  public:
```

```
    bool operator()( const Point &x1, const Point &x2 ) {
        return x1.value > x2.value;
    }
};
class KthSmallestInSortedMatrix {
  public:
    static int findKthSmallest( vector<vector<int>> &matrix, int k ) {
        int result = -1;
        int m = matrix.size();
        int n = matrix[0].size();
        priority_queue<Point, vector<Point>, Compare> min_heap;
        min_heap.push( Point{ matrix[0][0], 0, 0 } );
        while ( !min_heap.empty() && k-- ) {
            auto min_elem = min_heap.top();
            result = min_elem.value;
            min_heap.pop();
            if ( min_elem.row + 1 < m && matrix[min_elem.row+1][min_elem.col]
!= -1 ) {
                int row = min_elem.row + 1;
                int col = min_elem.col;
                min_heap.push( Point{ matrix[row][col], row, col } );
                matrix[row][col] = -1;
            }
            if ( min_elem.col + 1 < n && matrix[min_elem.row][min_elem.col +
1] != -1 ) {
                int row = min_elem.row ;
                int col = min_elem.col +1;
                min_heap.push( Point{ matrix[row][col], row, col } );
                matrix[row][col] = -1;
            }
        }
        // TODO: Write your code here
        return result;
    }
};

int main( int argc, char *argv[] ) {
    vector<vector<int>> matrix2 = {vector<int>{2, 6, 8}, vector<int>{3, 7,
10},
                                   vector<int>{5, 8, 11}
                                  };
    int result = KthSmallestInSortedMatrix::findKthSmallest( matrix2, 5 );
    cout << "Kth smallest number is: " << result << endl;
    system( "pause" );
```

```
}
```

当然这道题最简单的思路实际上是，把其当做K路归并来做。即每个数组中只取前K个值。然后按上题的做法来做。

2-3 寻找包含区间

> 给定M个有序区间，寻找一个最小的range。这个range每个数组必须至少包含一个数字。

e.g.1

```
Input: L1=[1, 5, 8], L2=[4, 12], L3=[7, 8, 10]
Output: [4, 7]
Explanation: The range [4, 7] includes 5 from L1, 4 from L2 and 7 from L3.
```

e.g.2

```
Input: L1=[1, 9], L2=[4, 12], L3=[7, 10, 16]
Output: [9, 12]
Explanation: The range [9, 12] includes 9 from L1, 12 from L2 and 10 from L3.
```

思路

1 5 8
4 12
7 8 10

感觉是用一个最小堆来确定最小值。然后用一个最大堆来确定一个最大值。

简单草稿

[1 7]
走1。5插入最小堆，最大堆。

[4, 7]， 走4。
12 插入最小最大堆。
5 12 走5
8 插入最小最大堆。
7 12 走 7
8 12 走8发现没有元素了。结束。

```cpp
using namespace std;

#include <iostream>
#include <limits>
#include <queue>
#include <vector>


struct Item {
    int value;
    int which_list;
    int next_index;
};
struct CompareGreater {
    bool operator()( const Item &x1, const Item &x2 ) {
        return x1.value > x2.value;
    }
};
class SmallestRange {
  public:
    static pair<int, int> findSmallestRange( const vector<vector<int>> &lists
) {
        priority_queue<Item, vector<Item>, CompareGreater> min_heap;
        priority_queue<int, vector<int>, less<int>> max_heap;
        for ( int i = 0; i < lists.size(); i++ ) {
            min_heap.push( Item{ lists[i][0], i, 1 } );
            max_heap.push(  lists[i][0]   );
        }
        pair<int, int>result = { 0, INT_MAX };
        while ( 1 ) {
            auto min_elem = min_heap.top();
            min_heap.pop();
            auto max_elem = max_heap.top();
            if ( max_elem- min_elem.value < result.second - result.first ) {
                result = make_pair( min_elem.value, max_elem );
            }
            if ( min_elem.next_index < lists[min_elem.which_list].size() ) {
                min_heap.push( Item{ lists[min_elem.which_list]
[min_elem.next_index], min_elem.which_list, min_elem.next_index + 1 } );
                max_heap.push( lists[min_elem.which_list]
[min_elem.next_index] );
            } else {
                break;
            }
```

```
        }
        return result;
    }
};

int main( int argc, char *argv[] ) {
    vector<vector<int>> lists = {{1, 5, 8}, {4, 12}, {7, 8, 10}};
    auto result = SmallestRange::findSmallestRange( lists );
    cout << "Smallest range is: [" << result.first << ", " << result.second
<< "]";
    cout << endl;

    vector<vector<int>> lists1 = {{1, 9}, {4, 12}, {7, 10, 16}};
    result = SmallestRange::findSmallestRange( lists1 );
    cout << "Smallest range is: [" << result.first << ", " << result.second
<< "]";
    cout << endl;
    system( "pause" );
}
```

不难发现，实际上我们根本不需要使用最大堆来维护最大值。直接使用一个int就够了。

2-4 前K个最大的pair

> 给定俩个降序数组。寻找前K个sum最大的pair。这个pair中的元素必须来自俩个数组。

e.g.1

```
Input: L1=[9, 8, 2], L2=[6, 3, 1], K=3
Output: [9, 3], [9, 6], [8, 6]
Explanation: These 3 pairs have the largest sum. No other pair has a sum
larger than any of these.
```

e.g.2

```
Input: L1=[5, 2, 1], L2=[2, -1], K=3
Output: [5, 2], [5, -1], [2, 2]
```

思路

[9 6 2]

[6 4 1]

第一个一定选9,6坑定没问题。选择下一个时，需要比较[9, 4]和[6 6] 选择[9, 4]。
然后比较[9 1] 和[6 6]选择[6 6]
然后比较[2 6] 和[9 1]

大概思路如下
i, j 分别指向俩个数组。

[9 6] 然后将[9 4] [6 6]分别加入最大堆。
走[9 4]然后将[6 4][9 1]加入最大堆
即每走一个[i][j]就将[i+1][j] [i][j+1]加入堆中。

```
using namespace std;

#include <iostream>
#include <vector>
#include <queue>

struct Item {
    int x;
    int y;
    int index_x;
    int index_y;
};
struct Compare {
    bool operator()( const Item &x1, const Item &x2 ) {
        return x1.x + x1.y < x2.x + x2.y;
    }
};
class LargestPairs {
  public:

    static vector<pair<int, int>> findKLargestPairs( const vector<int>
&nums1, const vector<int> &nums2, int k ) {
        vector<pair<int, int>> result;
        priority_queue<Item, vector<Item>, Compare> max_heap;
        // TODO: Write your code here
        max_heap.push( Item{nums1[0], nums2[0], 0, 0} );
        while ( k-- ) {
            auto max_sum_elem = max_heap.top();
```

```
            max_heap.pop();
            result.push_back( make_pair( max_sum_elem.x, max_sum_elem.y ) );
            if ( max_sum_elem.index_y + 1 < nums2.size() &&
max_sum_elem.index_x < nums1.size() ) {
                max_heap.push( {
                    nums1[max_sum_elem.index_x],
                    nums2[max_sum_elem.index_y + 1],
                    max_sum_elem.index_x,
                    max_sum_elem.index_y + 1
                } );
                max_heap.push( {
                    nums1[max_sum_elem.index_x + 1],
                    nums2[max_sum_elem.index_y ],
                    max_sum_elem.index_x + 1,
                    max_sum_elem.index_y
                } );
            }
        }
        return result;
    }
};

int main( int argc, char *argv[] ) {
    auto result = LargestPairs::findKLargestPairs( {9, 8, 2}, {6, 3, 1}, 3 );
    cout << "Pairs with largest sum are: ";
    for ( auto pair : result ) {
        cout << "[" << pair.first << ", " << pair.second << "] ";
    }
    system( "pause" );
}
```

第二种思路，类似选取前K个最大值的思路。即建立一个K大小的小根堆。然后利用for循环暴力遍历得到所有pair。

俩点优化：

1. 因为我们只选取前K个。因此没有必要暴力所有。只需要遍历俩个数组中的前K个即可。
2. 提前结束条件。即当前pair的sum小于小根堆的top元素的sum。那么就可以提前结束这次循环。因为descend降序排列，所以后续的sum一定会小于当前sum。

```
using namespace std;

#include <iostream>
```

```cpp
#include <queue>
#include <vector>

class LargestPairs {
 public:
  struct sumCompare {
    bool operator()(const pair<int, int> &x, const pair<int, int> &y) {
      return x.first + x.second > y.first + y.second;
    }
  };

  static vector<pair<int, int>> findKLargestPairs(const vector<int> &nums1,
                                                  const vector<int> &nums2,
int k) {
    vector<pair<int, int>> minHeap;
    for (int i = 0; i < nums1.size() && i < k; i++) {
      for (int j = 0; j < nums2.size() && j < k; j++) {
        if (minHeap.size() < k) {
          minHeap.push_back(make_pair(nums1[i], nums2[j]));
          push_heap(minHeap.begin(), minHeap.end(), sumCompare());
        } else {
          // if the sum of the two numbers from the two arrays is smaller
than the smallest (top)
          // element of the heap, we can 'break' here. Since the arrays are
sorted in the descending
          // order, we'll not be able to find a pair with a higher sum moving
forward.
          if (nums1[i] + nums2[j] < minHeap.front().first +
minHeap.front().second) {
            break;
          } else {  // else: we have a pair with a larger sum, remove top and
insert this pair in
                    // the heap
            pop_heap(minHeap.begin(), minHeap.end(), sumCompare());
            minHeap.pop_back();
            minHeap.push_back(make_pair(nums1[i], nums2[j]));
            push_heap(minHeap.begin(), minHeap.end(), sumCompare());
          }
        }
      }
    }
    return minHeap;
  }
};
```

```
int main(int argc, char *argv[]) {
  auto result = LargestPairs::findKLargestPairs({9, 8, 2}, {6, 3, 1}, 3);
  cout << "Pairs with largest sum are: ";
  for (auto pair : result) {
    cout << "[" << pair.first << ", " << pair.second << "] ";
  }
}
```

3. K路归并的内存优化问题

3.1 单机内存不够的K路归并

> 给定数组。数组很大。10G。但是机器内存只有1G。如何排序。

数据存放在磁盘上。分为100组。每组100M。每组先进行排序处理。
将每组数据的10M读入到内存中。刚好占用1G。然后进行100路K路归并。
因为100大小的堆所占内存远远小于1M。所以当前忽略其内存。

3.2 多机K路归并

思路是一样的。就是数据单台机器存放不下了。磁盘也存不小。那么就将数据分散存储在多态机器上。利用上面方法将每台机器上的数据排好序之后。然后通过网络进行K路归并排序。

3.3 优先级队列

注意这里的优先级队列一般指的是消息队列。假如我们有K个接收线程。每个线程有一个阻塞队列。里面存放的消息具有优先级。当前只有一个处理线程。

就可以使用一个堆。每次选取优先级最高的任务进行处理。

这个可以做到简历中去。

4. 总结

K路归并的关键在于每路有序。所以我们才能构建大小为K的堆来选取其中的最值元素。