327 Count of Range Sum

LeetCode

Author 张越

看到题目直观的想法是将问题转化为前缀和，那么问题等价于
lower <= sum[j] - sum[i] <= upper => lower + sum[i] <= sum[j] <= upper + sum[i]
只需要统计符合条件的sum(i), sum[j]的数对个数
我们可以发现问题的两个基本的性质
1. 必须符合j>=i
2. sum[j]与sum[i]之间必须满足大小关系

naive algorithm:
O(n^2)暴力枚举sum(i,j)数对，并进行大小的判断

我们发现此问题和求逆序对非常的类似(j>i的同时要求sum[i] 下面看几种不同的归并排序的实现对于算法复杂度的影响。

解法一

```
class Solution {
public:
    int ans = 0;

    int countRangeSum(vector<int>& nums, int lower, int upper) {
        long long int sum = 0;
        vector<long long int> prefix;
        for(int i = 0; i < nums.size(); ++i) {
            sum += nums[i];
            prefix.push_back(sum);
        }
        mergeSort(prefix, 0, prefix.size() - 1, lower, upper);
        return ans;
    }

    void mergeSort(vector<long long int> & sum, int l, int h, int lower, in
t upper) {
        if (l == h ) {if(sum[l] >= lower && sum[l] <= upper) ans++;return;}
        if (l > h) return;
        int m = l + (h - l) / 2;
        mergeSort(sum, l, m, lower, upper);
        mergeSort(sum, m+1, h, lower, upper);
        int low = m + 1;
        int high = h;
        for(int i = l; i <= m; ++i) {
            low = m + 1;high = h;
            while(high >= low && sum[high] - sum[i] > upper) high--;
            while(high >= low && sum[low] - sum[i] < lower) low++;
            if (high >= low) ans += high - low + 1;
        }
        inplace_merge(sum.begin() + l, sum.begin() + m + 1, sum.begin() + h
+ 1);
    }
};
```

递归公式为T(n) = 2T(n/2) + (n/2)^2最后求解得其为O(N^2logN) 明显差于navie的O(N^2)求解，这是由于在combine的过程中没有最优进行combine过程导致的。TLE on 61th test.

进一步优化，考虑利用右半部分有序的性质，进行二分查找。（AC in 32ms）

```
class Solution {
public:
    int ans = 0;
    int countRangeSum(vector<int>& nums, int lower, int upper) {
        long long int sum = 0;
        vector<long long int> prefix;
        for(int i = 0; i < nums.size(); ++i) {
            sum += nums[i];
            prefix.push_back(sum);
        }
        mergeSort(prefix, 0, prefix.size() - 1, lower, upper);
        return ans;
    }

    void mergeSort(vector<long long int> & sum, int l, int h, int lower, in
t upper) {
        if (l == h ) {if(sum[l] >= lower && sum[l] <= upper) ans++;return;}
        if (l > h) return;
        int m = l + (h - l) / 2;
        mergeSort(sum, l, m, lower, upper);
        mergeSort(sum, m+1, h, lower, upper);
        for(int i = l; i <= m; ++i) {
            // low = m + 1;high = h;
            // while(high >= low && sum[high] - sum[i] > upper) high--;
            // while(high >= low && sum[low] - sum[i] < lower) low++;
            vector<long long int>::iterator high =
lower_bound(sum.begin()+m+1, sum.begin()+h+1, upper+sum[i]+1);
            vector<long long int>::iterator low =
lower_bound(sum.begin()+m+1, sum.begin()+h+1, lower+sum[i]);
            if (low != sum.begin()+h+1) {
                if (high != sum.begin()+h+1) ans += high-low;
                else if (sum[h] - sum[i] <= upper) ans += high -low;
            }
        }
        inplace_merge(sum.begin() + l, sum.begin() + m + 1, sum.begin() + h
+ 1);
    }
};
```

T(n) = 2T(n/2) + nlog(n/2) 求解最终的算法复杂度为o(n(logn)^2)由于O(n^2)求解

此时我们可以发现我们并没有有效的利用到左侧序列有序的性质，进一步优化。AC in 19ms

```
class Solution {
public:
    int mergeSort(vector<long>& sum, int lower, int upper, int low, int
high)
    {
        if(high-low <= 1) return 0;
        int mid = (low+high)/2, m = mid, n = mid, count =0;
        count =mergeSort(sum,lower,upper,low,mid)
+mergeSort(sum,lower,upper,mid,high);
        for(int i =low; i< mid; i++)
        {
            while(m < high && sum[m] - sum[i] < lower) m++;
            while(n < high && sum[n] - sum[i] <= upper) n++;
            count += n - m;
        }
        inplace_merge(sum.begin()+low, sum.begin()+mid, sum.begin()+high);
        return count;
    }

    int countRangeSum(vector<int>& nums, int lower, int upper) {
        int len = nums.size();
        vector<long> sum(len + 1, 0);
        for(int i =0; i< len; i++) sum[i+1] = sum[i]+nums[i];
        return mergeSort(sum, lower, upper, 0, len+1);
    }
};
```

此时需要注意的是右侧序列指针移动的方向，一左一右的移动方式肯定是不对的。

观察combine过程，此时对于左侧序列和右侧序列，每个元素最多只会被访问一次。O(N)
因此终于降到了归并排序本来的时间复杂度O(NlogN)

此题还可以进一步优化常数项，比如将combine过程和merge过程结合，不再赘述。

扩展题目
1. 和为k的倍数的最长连续子串和
(1) DP方法

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    int n, k;
    cin >> n;
    vector<int> p(n);
    for (int i = 0; i < n; i++)
        cin >> p[i];
    cin >> k;
    //dp[i][j]记录以p[i]结尾的子串中，子串和对k求余为j的最远index
    vector<vector<int>> dp(n, vector<int>(k, -1));
    dp[0][p[0] % k] = 0; //边界条件

    int ans = 0;
    if (p[0] % k == 0)
        ans = 1;
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < k; j++) {
            p[i] %= k;
            int temp = (k + j - p[i]) % k;
            if (dp[i - 1][temp] != -1)
                dp[i][j] = dp[i - 1][temp];
            if (j == 0 && dp[i][j] != -1) {
                if (i - dp[i][j] + 1 > ans)
                    ans = i - dp[i][j] + 1;
            }
        }
        if (dp[i][p[i]] == -1)//边界条件
            dp[i][p[i]] = i;
        if (p[i] == 0)
            ans = ans == 0 ? 1 : ans;
    }
    cout << ans;
    return 0;
}
```

(2) 利用前缀数组
sum[i] = a[0]+...+a[i] 因此sum(i) - sum(j)正好是sequence i-j的子串和
然后枚举两个起点和终点即可得到最终答案，有的技巧是先从长度长的子串开始遍历，节约时间。
2. 逆序对