HW2 보고서 2020203090 한용옥

실행환경

언어 : `c++20`

컴파일러 및 IDE : Microsoft Visual Studio Community 2022 버전 17.13.6

입력, 데이터 처리

`method1, 2` 모두 **점수** 를 기준으로 분할 하지만, 파일 출력은 **학번** 순으로 해야하므로 점수와 학번을 모두 연산 전 보존하여야한다
본 프로그램에서는 `using student = pair<int, int>;`를 사용하여 학생 데이터를 점수와 학번 쌍으로 간주하고 정렬, 계산에 이용한다
이하 서술에서 점수로 정렬, 점수로 분할, 학번 순 정렬 등등 은 모두 `student`형 을 가지고 연산하는 것이다

```
//입력 처리
int n, k, temp; cin >> n >> k; vector<student> s_v;
for (int i = 0; i < n; i += 1)
    { cin >> temp; s_v.push_back(pair{ i + 1, temp }); }
//함수 실행
method1(s_v, k);
method2(s_v, k);
```

입력된 데이터는 `(학번, 점수)` 형태로 배열에 담기게 되고 각 메서드에 넘겨 실행한다
메서드는 복사본을 받기 때문에 같은 배열에 대해 독립적으로 실행 가능하다

Method 1

점수가 높은 학생이 앞쪽 그룹에 있어야 하므로 최적 분할은 연속구간 분할이다
증명 보기

따라서 $\min g_i - \max g_{i+1}$ 는 $s_k - sk + 1$ 형태 일 수 밖에 없다

내림차순 정렬된 데이터 $s_1, s_2, \ldots, s_n$ 에 대해
연속된 데이터의 차이 즉 $s_i - s_{i+1}$ 을 모두 구한 뒤, 차이가 큰 상위 `k-1` 개 뽑아 그 인덱스로 분할 하면 된다

구현

전체 절차

`method1` 는 입력으로 학생데이터 `s_v`와 그룹 수 `k`를 받아 아래를 실행한다

0. 들어온 입력 정렬
1. 입력 배열을 순회하며 연속된 두 값의 차이와 인덱스를 `diff`에 저장
2. `diff`에서 값이 큰 상위 `k-1`개를 뽑음
3. 뽑힌 인덱스로 원본 배열을 분할, 학번 순 정렬
4. 출력

아래는 각 절차에 대한 상세 설명이다

`diff` 배열 생성

```
vector<student> diff;
for (int i = 0; i < s_v.size() - 1; i += 1) {
    int s_v_diff = s_v[i].second - s_v[i + 1].second;
    if (s_v_diff == 0) continue;
    diff.push_back(student{ i + 1, s_v_diff }); }
```

차이 값 배열을 만드는 과정이다
다음 값이 현재 값과 다르다면 차이를 `diff`에 저장한다
이 때 분할 편의를 위해 같은 값이 끝나는 지점 + 1 인덱스를 저장한다

상위 `k-1` 개 뽑기

`diff` 는 차이값 배열이다 즉 `diff` 에서 상위 `k-1`개를 고르면

분할 및 구간 학번순 정렬

실제로 배열을 분할해 따로 저장하지 않고 인덱스를 이용해 그 구간만 학번 순으로 정렬하여 분할 된 것 같은 효과를 낸다

```
for (int i = 0; i < end_idx.size() - 1; i += 1)
    ranges::sort(s_v.begin() + end_idx[i], s_v.begin() + end_idx[i + 1],
                 less(), &student::first);
```

위 명령어를 이용해 앞 단계에서 구한 구간별로 학번이 정렬된다

출력

콘솔에 구한 최대값을 표시하고
앞에서 구한 구간 인덱스를 이용해 `학번(점수)` 형식으로 파일 저장한다

시간복잡도 분석

정렬

입력인 n 개의 데이터를 정렬 `std::ranges::sort`는 $O(n \log n)$ 이므로
$O(n \log n)$

인접 차이 계산

배열을 순회하며 각 원소 차이값, 인덱스만 추가 하므로 $O(n)$

상위 k개 추출

`std::partial_sort`의 시간복잡도는 $O(n \log k)$ 이다
힙을 사용해 정렬 대상 상위 k 개 원소를 유지하며 전체 n 개를 처리하는 방식이다

$O(n \log k)$

분할 및 구간 학번 순 정렬

k 개로 나눴으므로 그룹 별 데이터 개수는
$n_1, n_2, \dots, n_k$ 개(모두 더하면 n) 이다
그룹 별로 정렬하고 $n_i < n$ 이므로 전체 시간복잡도는

$$\sum_{i=1}^{k} O(n_i \log n_i) < \sum_{i=1}^{k} O(n_i \log n) = O(n \log n)$$

따라서 $O(n \log n)$

출력

원본 배열을 순회하며 한 원소를 출력하기만 하므로
시간복잡도는 $O(n)$

전체 과정을 종합하면 `method1` 시간복잡도 $O(n \log n)$

Method 2

보조정리 1

내림차순 정렬 배열 S 에 대해 그룹의 분산의 합이 최소가 되게 한 분할을 P 라 하면
P 의 모든 원소는 연속 S 원소의 열이다

보조정리 1 증명 : 부록 보기

보조정리 1 덕분에 최적 분할은 연속 구간의 모음이므로
마지막 구간이 어디서 시작되는지를 바탕으로 한 점화식을 세울 수 있다

용어	정의
$s_1, s_2, \dots, s_n$	내림차순 된 데이터 n 개
$Var(a, b)$	$s_a, s_{a+1}, \dots, s_b$ 들의 분산
$V(k, i)$	데이터 $s_1, s_2, \dots, s_i$ 를 k개 최적 분할 할 때의 분산값

최적 분할은 연속구간 형태이므로 어떤 데이터의 k 개 최적 분할을 보면
마지막 그룹을 제외한 나머지 데이터는 그들의 $k-1$ 개 최적 분할이다
(그렇지 않으면 그들의 $k-1$ 개 분할로 대체 한 것이 전체 최적이므로)

각 구간에는 한 개 이상의 데이터가 있어야 하므로
맨 끝 그룹이 가를 수 있는 지점의 범위는 $[k-1, i)$ 이다
따라서 점화식은 아래와 같다

$$V(k, i) = \min_{t \in [k-1, i)} (V(k-1, t) + Var(t+1, i))$$

이는 최적 분산값이므로 진짜로 분할하기 위해 각 단계 분할 최적 인덱스인 t 를 저장해야한다
따라서 최적 인덱스를 아래와 같이 정의한다

$$T(k,i) = \underset{t \in [k-1,i)}{\arg\min} \left(V(k-1,t) + Var(t+1,i)\right)$$

정의에 의해 $T(k-1, T(k,i))$ 는 끝에서 두 번째 그룹의 시작 인덱스를 결정하므로
재귀 형태로 분할 인덱스를 추적 가능하다

`method2` 는 n 개의 데이터를 k 개 최적 분할 하는 것이 목표이므로
$V(k,n)$ 을 구하고 $T(k,i)$ 를 추적해가며 인덱스를 구한 후 분할하고
각 분할을 학번 순 정렬하면 된다

구현

위 점화식을 바탕으로한 동적계획법을 메모이제이션으로 구현하였다

전체 절차

`method2` 는 입력으로 학생데이터 `s_v`와 그룹 수 `k`를 받아 아래를 실행한다

0. 들어온 입력 정렬
1. 누적합, 누적제곱합 배열 초기화
2. 값 배열 V, 인덱스 추적 배열 T 생성, $V(k,i)$ 경계값 처리
3. 메모이제이션으로 V, T 계산
4. 분할 인덱스 추적
5. 분할 및 구간 학번순 정렬
6. 출력

아래는 각 절차에 대한 상세 설명이다

누적합, 누적제곱합 배열 초기화 -> 분산을 $O(1)$ 에 구하기

$$sum(n) = \sum_{i=1}^{n} s(i) \quad ssum(n) = \sum_{i=1}^{n} s(i)^2$$

위 누적합, 누적 제곱합을 미리 준비해두면

$$Var(a,b) = \text{제곱의 평균} - \text{평균의 제곱}$$

$$= \frac{ssum(b) - ssum(a-1)}{b-a+1} - \left(\frac{sum(b) - sum(a-1)}{b-a+1}\right)^2$$

이므로 $Var(a,b)$ 를 $O(1)$ 에 구할 수 있다

본 프로그램에선 `double var(int start, int end, ...sum, ...sum2)` 가 누적 합, 제곱합 배열을 이용하여 $Var(start, end)$ 를 $O(1)$ 에 계산한다

값 배열 V, 인덱스 추적 배열 T 생성, $V(k,i)$ 경계값 처리

V, T 는 이차원 함수이므로 이 값들을 기록하기 위해 이차원 배열
`vector<vector<double>> V, vector<vector<int>> T` 를 생성한다
`V, T` 의 크기는 `k * n`이다

배열 인덱스는 `0`부터 시작하므로 `V[k-1][i-1]` = $V(k,i)$ 이다

$V(1,i)$ 는 i 개를 그룹 1개로 분할한다는 뜻으로 분할하지 않을 때의 최적 분산값과 같다
$V(1,i) = Var(1,i)$ 이므로 반복문으로 `V[0][i]` 들을 `var`을 이용해 채운다

메모이제이션으로 V, T 계산

점화식을 재귀함수 `method2_recur(V(값 배열), k(그룹 개수), i(인덱스), T(인덱스 배열), ...)`로 구현하여 모든 가능한 V, 그에 대한 인덱스 T 를 계산한다

```
if (V[k][i] >= 0) return;
```

종료조건은 이미 $V(k,i)$ 가 계산되어 있을 때이다
앞 단계에서 경계값 `V[0][i]`들을 채우고 점화식 형태에 의해 `k`가 감소하므로
종료조건은 위 조건으로 충분하다

```
for (int t = k - 1; t < i; t += 1) {
    method2_recur(V, k - 1, t, T, sum, sum2);
    double temp = V[k - 1][t] + var(t + 1, i, sum, sum2);
    if (temp <= result) { result = temp; opt = t;} }
T[k][i] = opt; V[k][i] = result;
```

위 코드 조각이 V 의 점화식을 구현한 코드이다
`t` 에 대한 반복문으로 $\min_{t\in[k-1,i)}$ 을 구현한다
먼저 함수를 재귀호출해 사용할 V 값을 계산하고 $(V(k-1,t)+Var(t+1,i))$ 값 중
가장 작은 값을 저장하여 `V, T`배열에 갱신한다

종료조건에 의해 재귀함수로 `V` 배열의 값은 한 번 만 계산된다

분할 인덱스 추적

정의에 의해 $T(k-1,T(k,i))$ 식으로 다음 인덱스들을 알아 낼 수 있다

```
void track_opt_idx(const vector<vector<int>>& T, int k, int i, vector<int>& result)
{
    if (k <= 0) return;
    int opt = T[k][i];
    result.push_back(opt + 1);
    track_opt_idx(T, k - 1, opt, result); }
```

위 재귀 함수로 외부 배열에 분할 기준 인덱스를 넣는다
T 의 정의에 의해 분할의 마지막 그룹의 시작 인덱스는 T 의 값에 1을 더한 것이므로 그 값을 저장한다
프로그램에서는 `opt_idx`에 저장된다

분할 및 구간 학번순 정렬

실제로 배열을 분할해 따로 저장하지 않고 인덱스를 이용해 그 구간만 학번 순으로 정렬하여 분할 된 것 같은 효과를 낸다

```
for (int i = 0; i < opt_idx.size() - 1; i += 1)
    ranges::sort(s_v.begin() + opt_idx[i], s_v.begin() + opt_idx[i + 1],
                 less(), &student::first);
```

위 명령어를 이용해 앞 단계에서 구한 구간별로 학번이 정렬된다

출력

콘솔에 구한 최대값을 표시하고
앞에서 구한 구간 인덱스를 이용해 `학번(점수)` 형식으로 파일 저장한다

Method 2 시간복잡도 분석

누적합, 누적제곱합 배열 초기화

길이가 n 인 배열 각각의 원소를 누적해가며 더하는 것으로
$O(n)$ 이다

값 배열 V, 인덱스 추적 배열 T 생성, $V(k,i)$ 경계값 처리

길이가 n 인 배열 각각의 원소에 Var 값을 부여하는 작업이다
`var` 의 시간복잡도는 $O(1)$ 이므로 이 작업의 시간복잡도는 $O(n)$ 이다

메모이제이션으로 V,T 계산

```
void method2_recur(...) {
    //종료 조건
    if (V[k][i] >= 0) return;
    //이하 계산
    int opt = 0; double result = numeric_limits<double>::max();
    for (int t = k - 1; t < i; t += 1) {
        method2_recur(V, k - 1, t, T, sum, sum2);
        double temp = V[k - 1][t] + var(t + 1, i, sum, sum2);
        if (temp <= result) { result = temp; opt = t; } }
    T[k][i] = opt; V[k][i] = result; }
//실제 호출
method2_recur(V, k - 1, size - 1, T, sum, sum2);
```

`method2_recur` 는 `V` 가 계산되어있으면 바로 종료되고
계산되어있지 않을 때(= 처음 접근할 때) 만 아래 내용을 수행한다

실제 호출시 데이터 개수 n , 그룹 수 k 를 넣어 호출한다
함수 내 반복문과 재귀호출로 인해 본 재귀함수는 모든 가능한 $V(a,b)$ 를 계산한다 ($a \in [0,k), b \in [0,n)$)

간단한 증명

> $V(k,n)$ 가 본 함수에 의해 계산되었을 때
> 호출되지 않은 상태 $V(a,b)$ 가 있다고 가정 ($a \in [0,k), b \in [0,n)$)
> 점화식 구현 정의($k-1$ 재귀 호출)에 따라 $V(a+1,b),...,V(k-1,b)$ 는

> $V(a,b)$ 가 필요하므로 계산될 수 없다
> $V(k-1,b)$ 가 계산되지 않는다면 함수의 `for` 문에서 $V(k-1,t)$ 를 이용해 $V(k,n)$ 을 계산할 수 없으므로
> $V(k,n)$ 역시 계산되지 않는다
> 이는 최초 가정과 모순이므로 모든 $V(a,b)$ 는 반드시 계산된다

비록 구현상으로는 반복문 내부에서 재귀 호출이 중첩되어 있으나
종료 조건을 통해 각 값 $V(k,i)$는 단 한 번만 계산되므로

$V(k,n)$을 계산할 때의 시간복잡도는
모든 가능한 $V(a,b)$ 에 대한 계산시 `for` 반복 횟수 합이다
각 상태에서 반복은 $i-k+1$ 번 실행되므로 수식은 아래와 같다

$$\sum_{b=1}^{k}\sum_{a=b}^{n-1}(a-b+1) \in O(kn^2)$$

수식 전개

부록 보기

분할 인덱스 추적

최적 분할의 인덱스는 `k-1` 개 이므로 재귀함수는 `k-1` 번 반복한다
시간복잡도 : $O(k)$

분할 및 구간 학번 순 정렬

k 개로 나눴으므로 그룹 별 데이터 개수는
$n_1, n_2, \ldots, n_k$ 개(모두 더하면 n) 이다
그룹 별로 정렬하고 $n_i < n$ 이므로 전체 시간복잡도는

$$\sum_{i=1}^{k} O(n_i \log n_i) < \sum_{i=1}^{k} O(n_i \log n) = O(n \log n)$$

따라서 $O(n \log n)$

출력

원본 배열을 순회하며 한 원소를 출력하기만 하므로
시간복잡도는 $O(n)$

전체 과정을 종합하면 `method2` 시간복잡도 $O(kn^2)$

논의 : `method2` 에 대한 제약조건 풀이

제약 1

> 같은 점수인 학생은 반드시 같은 그룹에 속해야 함

접근

같은 점수는 같은 그룹이어야하므로 `method1` 의 풀이와 비슷하게
같은 점수끼리 묶고 그 배열에 대해 `method2` 를 적용하면 된다
그러면 자연히 경계는 같은 값이 아니게 되는 지점으로 정해지고 같은 점수가 쪼개지지 않는다

구체적으로 다음 절차를 따르면 된다

0. 입력 데이터 배열을 정렬한다
1. 배열 데이터를 순회하며 다른 값이 있을 경우 `core` 에 현재 값, 값 시작 인덱스, 끝 인덱스를 저장한다
2. 1의 배열에 대해 `method2` 의 나머지 절차를 따른다

구현시 달라져야 할 부분

시간복잡도 변화

1번 절차만 추가되었고 나머지 부분은 변한 것이 없다
1번 절차는 배열을 순회하며 각 원소마다 조건 검사, `diff` 에 값 추가만 하므로
1번 절차의 시간복잡도 : $O(n)$ 이다

따라서 제약 1을 적용한 `method2` 의 시간복잡도는
$O(kn^2)$ 로 변하지 않는다

제약 2

접근

구현시 달라져야 할 부분

시간복잡도 변화

제약 3

접근

구현시 달라져야 할 부분

시간복잡도 변화

제약 4

> 각 학생은 우선순위 값 p_i 를 가진다
> 우선순위의 총합이 높은 그룹이 더 낮은 그룹 번호(즉, 더 좋은 등급)를 가져야 한다

접근

`method2` 를 일단 적용하고
각 그룹을 우선순위 합을 기준으로 정렬하면 된다

구현시 달라져야 할 부분

시간복잡도 변화

프로그램 실행 및 출력

아래는 실행 예시이다 과제 ppt의 예시를 잘 계산하여 출력한다

Microsoft Visual Studio 디버그

```
15 3
50 50 10 20 50 10 50 50 20 20 50 50 50 50 10
40
0.000
```

Partition1.txt

파일 편집 보기

```
1(50) 2(50) 5(50) 7(50) 8(50) 11(50) 12(50) 13(50) 14(50)
4(20) 9(20) 10(20)
3(10) 6(10) 15(10)
```

Partition2.txt

파일 편집 보기

```
1(50) 2(50) 5(50) 7(50) 8(50) 11(50) 12(50) 13(50) 14(50)
4(20) 9(20) 10(20)
3(10) 6(10) 15(10)
```

Microsoft Visual Studio 디버그

```
15 3
50 85 10 35 45 15 75 80 25 30 55 60 65 70 5
20
197.917
```

Partition1.txt

파일 편집 보기

```
1(50) 2(85) 5(45) 7(75) 8(80) 11(55) 12(60) 13(65) 14(70)
4(35) 9(25) 10(30)
3(10) 6(15) 15(5)
```

Partition2.txt

파일 편집 보기

```
2(85) 7(75) 8(80) 13(65) 14(70)
1(50) 5(45) 11(55) 12(60)
3(10) 4(35) 6(15) 9(25) 10(30) 15(5)
```

부록 A

증명 목표

내림차순 정렬된 점수열 $S = (s_1, s_2, \ldots, s_n)$ 에 대해

$$\sum_{i=1}^{k-1} \left(\min g_i - \max g_{i+1}\right)$$

을 최대화하도록 k개의 그룹 $(g_1, g_2, \ldots, g_k)$ 로 나눌 때
각 그룹은 입력 내에서 연속한 데이터형태라고 가정 가능하다

증명

최적 분할 $g_1, g_2, \ldots, g_k$ 중 어떤 그룹 g_j 가 비연속적이라 하자
g_j 는 $[s_a, s_b] \cup [s_c, s_d]$ 처럼 중간에 빈 구간이 있다 ($b + 1 < c$)

S가 내림차순 정렬이므로, $s_b \geq s_{b+1} \geq s_c$
s_{b+1}은 g_j에 포함되지 않았지만 그보다 작은 값이 g_j에 포함돼 있다

s_{b+1} 을 g_j 에 포함시키고, 그보다 작은 값 s_d (현재 g_j 에 포함) 를 제외시키면 그룹의 $\min$ 은 같거나 커지고, $\max$ 는 같거나 작아진다.
$\min g_j - \max g_{j+1}$ 값이 증가하거나 유지된다.

이와 같은 교환을 반복하면, 각 그룹을 연속 구간으로 바꾸더라도
목적 함수 $\sum_{i=1}^{k-1} \left(\min g_i - \max g_{i+1}\right)$ 는 줄어들지 않는다

결론

내림차순 정렬된 점수열에 대해, 점수차이 합을 최대로 하는 모든 그룹 분할은 연속구간 분할로 바꿀 수 있다

따라서 최적 분할문제를 풀 때 최적해가 연속구간 분할이라고 가정해도 된다

복귀

부록 B

복귀

부록 C

시간복잡도는

$$\sum_{b=1}^{k-1} \sum_{a=b}^{n-1} (a - b + 1)$$

먼저 안쪽 합부터 계산한다
$a - b + 1$ 은 a에 대한 1차식이므로 다음과 같이 정리할 수 있다

안쪽 합

$$\sum_{a=b}^{n-1} (a - b + 1) \quad (\text{치환 } \mathrm{i} = \mathrm{a} \text{ - } \mathrm{b})$$

$$= \sum_{i=0}^{n-1-b}(i+1) = \sum_{i=1}^{n-b} i = \frac{(n-b)(n-b+1)}{2}$$

따라서 전체 합은

$$\frac{1}{2}\sum_{b=1}^{k-1}(n-b)(n-b+1)$$

이 식을 전개하자 $x = n - b$ 로 치환하면
$b = 1 \to x = n-1, \quad b = k-1 \to x = n-(k-1)$

시그마 치환변수 x는 $x = n-1$ 부터 $x = n-k+1$ 까지 감소 따라서 전체 합은

$$= \frac{1}{2}\sum_{x=n-k+1}^{n-1} x(x+1)$$

이를 전개하면 $x(x+1) = x^2 + x$ 이므로

$$= \frac{1}{2}\left(\sum_{x=n-k+1}^{n-1} x^2 + \sum_{x=n-k+1}^{n-1} x\right)$$

합 공식

$$\sum_{x=a}^{b} x = \frac{(b-a+1)(a+b)}{2}$$

제곱합 공식

$$\sum_{x=a}^{b} x^2 = \frac{b(b+1)(2b+1)}{6} - \frac{(a-1)a(2a-1)}{6}$$

여기서 $a = n-k+1$, $b = n-1$ 계속 전개시

$$\sum_{b=1}^{k-1}\sum_{a=b}^{n-1}(a-b+1) = \frac{1}{2}\left[\left(\sum_{x=n-k+1}^{n-1} x^2\right) + \left(\sum_{x=n-k+1}^{n-1} x\right)\right]$$

$$= \frac{1}{2}\left[\left(\frac{(n-1)(n)(2n-1)}{6} - \frac{(n-k)(n-k+1)(2n-2k+1)}{6}\right) + \left(\frac{(k-1)(2n-k)}{2}\right)\right]$$

$$= \frac{1}{2}n^2k - \frac{1}{2}n^2 - \frac{1}{2}nk^2 + nk - \frac{1}{2}n + \frac{1}{6}k^3 - \frac{1}{2}k^2 + \frac{1}{3}k$$

$k \leq 12$ 이므로 시간복잡도는 $O(kn^2)$ 이다

복귀