Tópicos Especiais em Algoritmos
Estruturas de Dados Lineares
Editorial

Daniel Saad Nogueira Nunes

CSES: Nearest smaller values

Este problema pode ser resolvido com o auxílio de uma pilha, que inicialmente começa vazia e, durante a execução do algoritmo, possuirá o índice de determinados elementos do vetor $X = (x_1, \ldots, x_n)$ de entrada. Percorre-se o vetor X da esquerda para a direita e, para cada elemento x_i devemos verificar os seguintes casos:

- Caso a pilha esteja vazia, não temos elemento $j < i$, $x_j < x_i$.
- Seja t o índice do topo da pilha. Caso $x_i > x_t$, imprime-se t e insere-se i na pilha.
- Caso contrário, remove-se o elemento da pilha e analisa-se a situação novamente de acordo com o estado da pilha.

Complexidade

Como cada elemento só pode ser inserido e retirado da pilha uma única vez, esta solução leva tempo $O(n)$.

Seletiva UnB 2020: Cotas e rateio linear

Observando as dimensões do problema, temos que $n \leq 10^5$ e $q \leq 10^{18}$, logo, qualquer solução que tente simular o procedimento de distribuição de cotas irá levar tempo proporcional ao número de cotas, o que é inviável. A estratégia é ir na linha de um algoritmo com complexidade baseada em N, o número de pessoas.

Inicialmente, temos N pessoas participando do processo de divisão de cotas, em que cada pessoa solicita um número v_i de cotas, com $V = (v_0, \ldots, v_{n-1})$. A solução, baseada em iteração, deve tentar eliminar pelo menos uma pessoa a cada iteração do algoritmo.

Para cada iteração do algoritmo, sejam:

- (min, i_{min}): um par com o valor mínimo dentre V e i_{min}: o índice deste elemento em V. Temos duas situações.
- n': o número de pessoas que ainda estão participando do rateio ($n' \leq n$).
- $total$: o número de cotas restantes.
- cur: o número de cotas que foi distribuída para os participantes que ainda estão no rateio.

Assim, temos:

- Se $min - cur \cdot (n') < total$, então podemos seguramente distribuir $min - cur$ cotas para cada participante ativo. Após o final desta iteração, $total \leftarrow total - ((min - cur) \cdot n')$, $n' \leftarrow n - 1$, e $cur \leftarrow min$. Certamente, i_{min} pode ser eliminado do rateio, já que recebeu todas as suas cotas.
- Caso contrário, podemos seguramente distribuir $\lfloor total/n' \rfloor$ cotas para cada participante, sobrando $total \mod n'$ cotas, os quais devem ser distribuídas aos $total \mod n'$ primeiros participantes ativos e encerrar o processo.

Complexidade

Como eliminamos, no mínimo, um participante da jogada a cada iteração, o tempo total, no pior caso, é $O(n \cdot t_{min})$, em que t_{min} é o tempo gasto para escolher o par (min, i_{min}). Utilizando uma fila de prioridades, t_{min} corresponde à $O(\lg n)$, assim, o tempo total é $O(n \lg n)$.

Detalhes de Implementação

Uma fila de prioridades de mínimo do `C++` pode ser declarada da seguinte forma:

`priority_queue<T,vector<T>,greater<T>>`, em que `T` é o tipo em questão.

UVA 12207: That is your queue

Este problema pode ser resolvido através de uma simples simulação através de um deque. Para consultas do tipo '`N`', basta retirar o paciente do deque e recolocá-lo no final. Caso contrário, podemos varrer o deque em busca do participante, apagá-lo do deque e inserí-lo no início do deque.

O deque só precisa ser inicializado com $\min(P, C)$ pessoas, haja vista que serão processadas apenas C consultas.

Complexidade

A complexidade da solução é $O(\min(P, C)^2)$, haja vista que há $O(\min(P, C))$ pessoas na fila e, para procurarmos um elemento em específico do deque e apagá-lo, gasta-se tempo $O(\min(P, C))$.

Detalhes de Implementação

Para apagar um elemento do deque, pode-se recorrer ao método `erase`.

Codeforces 1279C: Stack of Presents

Simular o procedimento leva tempo quadrático no número de presentes, o que não é desejável, dado que $n \leq 10^5$. Podemos calcular o número total de movimentos sem precisar manipular a pilha de fato.

Sejam $A = (a_1, \ldots, a_n)$ a ordem dos presentes na pilha, sendo a_1 o elemento do topo da pilha, e $B = (b_1, \ldots, b_m)$ a ordem na qual os presentes devem ser entregues.

Podemos calcular uma permutação $pos[a_i] = i$ em tempo linear, desta forma, $pos[j]$ nos dará a posição do elemento a_j na pilha. Com posse de pos_j, analisamos cada elemento b_i, então temos. Inicialmente, tome que $last := -1$.

- Se $last < b_i$, precisamos realizar $1 + 2 \cdot (pos[b_i] - (i - 1))$ movimentos, os quais já deixam os elementos que estavam no topo de b_i reorganizados para retirada do elemento b_{i+1}, caso ele esteja neste conjunto. Atualizamos $last := pos[b_i]$.
- Caso contrário, o elemento b_i está no topo devido à uma reestruturação prévia, e levamos apenas 1 movimento para retirá-lo.

Complexidade

O algoritmo leva tempo $O(n + m)$.