	На пълното решение	На частичните решения
Тагове	Минимално покриващо дърво DSU Small to large Паралелно двоично търсене	Дейкстра

Анализ

Подзадача №1

Както винаги оставих подзадача с тестовите примери за обратна връзка от системата.

Подзадача №2

За всяка заявка намираме минималната цена на кола c_u за да достигнем връх u чрез лека модификация на алгоритъма на Дейкстра. Цената на път в този граф е равна на максималното тегло на ребро в пътя, като така определена цена на пътя спазва всички изисквания, нужни, за да е възможно извършването на алгоритъм на Дейкстра. Ако $a_1, a_2, \ldots, a_N$ е сортирания ред на $c_1, c_2, \ldots, c_N$, то отговорът на заявката ще е a_{k_i+1}, тъй като $a_1 = 0$.

Постигната сложност: $O(QMlog_2M)$

Имплементация: `tourism_12p.cpp`

Подзадача №3

Забелязваме, че ако намерим минималното покриващо дърво на графа, то всеки път от гореспоменатите ще лежи на него. Вече ребрата в графа са $N-1$ на брой, което драстично подобрява сложността ни.

Постигната сложност: $O(QNlog_2N)$

Имплементация: `tourism_30p.cpp`

Подзадача №4

Всякъв вид премахване на log_2N от решението ни би свършило работа. Погрижих се и най-дървеният начин – да се компресират теглата на ребрата и да се прилага линейна Дейкстра да минава ограниченията, макар да трябва да се ползва имплементация на вектори, подобна на списък на ребра. По-добър начин е да се приложи DFS и count sort, тъй като графа е дърво.

Постигната сложност: $O(QN)$

Имплементация: `tourism_42p.cpp`

Подзадача №5

Отговорът на заявка за връх u е минималното тегло на ребро, индицдентно на u.

Постигната сложност: $O(N + M + Q)$

Имплементация: `tourism_troll_3p.cpp`

Подзадача №6

Подзадачата е за почти пълни решения. Нека строим минималното покриващо дърво на графа чрез алгоритъм на Крускал и целим да отговорим на заявките офлайн. Нека $K = k_1$. Ние целим да намерим за всеки връх, попаднал в заявка, първия момент, в който компонентата му в Крускала е станала с големина $\geq K-1$. Нека за всяка компонента си пазим в опашка всички заявки за връх от нея и използваме *Small to large* за пренасяне на заявки при *merge* на две компоненти. Така, когато компонента стане с размер $\geq K - 1$, ние *чистим* всички заявки за нея. След изпълнението на алгоритъма ще сме отговорили на всички заявки.

Постигната сложност: $O(M + (N + Q)log_2 N)$

Имплементация: `tourism_AlmostFull_23p.cpp`

Подзадача №7. Решение със Small to Large

Нека разширим идеята на шестата подзадача. Вместо в опашки, ние може да си поддържаме заявките в *Heap*-ове спрямо k_i. Така, пак прилагаме *Small to large*, като постепенно махаме най-горният елемент на *Heap*-овете, когато компонентата надвиши желаната големина за заявката.

Постигната сложност: $O(M + (N + Q)log_2^2 N)$

Имплементация: `tourism_SmallToLarge_100p.cpp`

Подзадача №7. Решение с паралелно двоично търсене

Ако приложим техниката *Паралелно двоично търсене*, използвана в и добре описана в анализа на $B3.\ coloring$ от Есенен турнир 2023, то на ние ще трябва да решим проблема в шеста подзадача log_2 на брой пъти. Ние постепенно ще добавяме ребра в минималното покриващо дърво, след което ще търсим дали големината на определена компонента е поне някаква стойност x. Това може да бъде реализирано бързо.

Постигната сложност: $O(M + (N + Q)\ log_2 N\ \alpha(N))$

Имплементация: `tourism-db.cpp - Добрин Башев`

Автор: Борис Михов