Problem A. PE class

Subtasks 1 and 2

Let's iterate through all possible strings of length 2^{2n} and check each one. For $n = 2$, all cases can be examined manually.

Subtask 3

There are only two cases: $a_i = 0$ or $a_i = -1$. When $a_i = 0$, the answer will be $LRLRLR\ldots$, and when $a_i = -1$, the answer will be $RLRLRL\ldots$.

Subtask 4

If $a_1 = 0$, then $s_1 = L$, otherwise $s_1 = R$. Then we can keep track of how many L's and R's have been placed in the prefix, and from this we can uniquely calculate the next character.

Subtask 6

Claim. The multiset of numbers obtained for L is the same as the multiset obtained for R.

Proof. It can be proven using mathematical induction. In any case, there will be adjacent L and R. It is easy to notice that the obtained numbers for them are the same. And since they do not affect all the others, we can remove them. Then we can proceed to the problem with $n - 2$.

It turns out that each number in the array a occurs an even number of times. If we leave half of the occurrences for each number, these will be the obtained numbers for all L. Let's denote this array as La. Now, using the minimum number of R symbols, we will obtain all the numbers from the array La. To do this, we will sort the values in the array in ascending order and insert the necessary elements in this order.

Problem B. Batyr I and Tima the Great

Subtask 1 ($m = 0$) — 5 points

In this subtasks, it is simple to see that you either move in circle clockwise or anticlockwise so answer is minimum between $|x_i - y_i|$ or $L - |x_i - y_i|$.

Subtask 2 $L, m, q \leq 10^2$ — 8 points

Let's build graph for each query, then Batyr's roads has weight 1 and Tima's roads has weight 0. And run shortest path algorithms to find answer.

Subtask 3 $L, m, q \leq 10^3$ — 11 points

Let's build graph before processing queries like in second subtask, and then run BFS or Dijkstra for each pair in query to find answer.

Subtask 4 $m, q \leq 10^3$ — 10 points

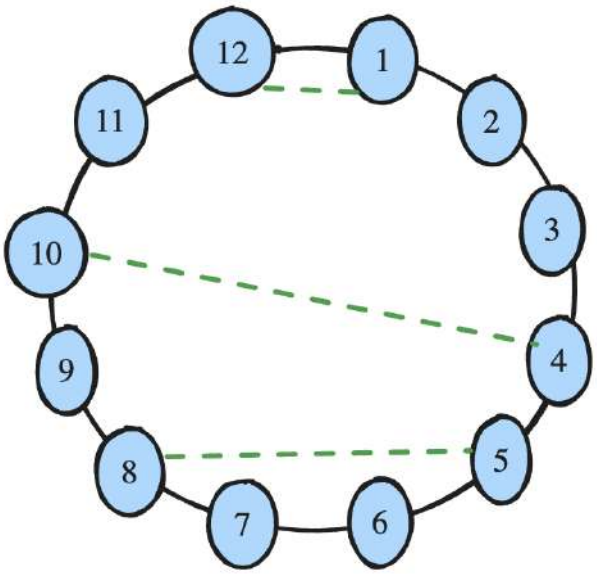
Notice that in previous subtask, you can build graph with nodes which appear only in one of queries or in Tima's edges. so we have $2(m+q)$ nodes on which we can build graph, and solve similar to third subtask.

Subtask 5 $b_i < a_{i+1}$ — **12 points**

Consider Tima's roads as segments. Observe that no two segments intersect, and none of them completely cover each other. When moving from x to y, the distance covered is equal to $y - x -$ sum, where sum is the total length $\sum(b_i - a_i)$ of segments that lie entirely within the range $[x, y]$.

Subtask 6 $a_i < a_{i+1}, b_{i+1} < b_i$ — **14 points**

Consider Tima's roads as segments. Note that while no segment intersects, each of them covers the other, implying a hierarchy among them. Add segment $[0, L]$ and then build a tree graph on this set of segments. For each segment, identify the shortest segment covering it, which becomes its parent in the tree. Additionally, in this tree, each point is part of the shortest segment covering it. Achieve this using a scanline: sort segments by their left side and maintain a stack of segments to easily find the parent. The weight of each edge is the distance between the closest points of the parent and son segments. To better understand how to handle queries, consider the given graph below, where $L = 12$, and Tima's roads are represented by the green lines.



As example to understand, let's use 3 and 9. so 3 will be linked to segment $[1, 12]$. while 9 to $[4, 10]$ in our tree. So in this case it is optimal to move from 3 to segment $[4, 10]$ and then from 10 to 9, so answer is 2. And let's consider another query 4, 9 in this case it is trivial we move from 4 to 10 and then 9, so answer is 1. in first example we moved to segment $[4, 10]$ which is son lca ($[1, 12]$) of segments $[4, 10]$ and $[1, 12]$, and in second it is going to lca ($[4, 10]$). It can be shown, that it's optimal to either go from both points to lca, or going to son of lca and then between them to each other. Time complexity is $O(nlogn)$.

Subtask 7 $|x_i - y_i| = 1$ $(1 \le i \le q)$ — **18 points**

In this version of problem, if x_i and y_i are connected by Tima's roads, the answer is 0; otherwise, it is 1. To determine if they are connected in that way, let's merge two of Tima's roads if they intersect each other but do not entirely cover one another. First, connect roads by their endpoints. When considering roads a and b, we need to identify all the roads that intersect them. For instance, if $a < b$, we deem some endpoints of road u as valid if u lies between a and b, and the rightmost endpoint from u is greater than or equal to b. Conversely, if $a > b$, for endpoints u in between, we need to check if the leftmost endpoint from u is less than or equal to b. We can find such endpoints using a segment tree and use a disjoint-set union (dsu) structure to connect them as components. This process can be accomplished in $O(n \log n)$ time.

Subtask 8 No additional constraints — 22 points

After building components same to subtask 7, we are left we almost same problem as subtask 6 but you also need to consider that compare to subtask 6 components are set of points rather than segment.

Problem C. Lazy, but honest

- $D = 1$, 5 points.

  We have to complete each task we receive on the same day. First, check that every $b_i \leq a_i$. Then print the number of days with $b_i \neq 0$. $O(n)$

- $N \leq 18$, 7 points

  If we decide to work some day it's always optimal to do as many tasks as possible and we start from the earliest tasks.

  For each day we have two options: not to work or to do as many tasks as possible. Use recursion. Store for each day how many tasks are left uncompleted in that day. $O(2^n * n)$

- All a_i equal, 18 points.

  Greedy solution: if we decide not to work on that day can we complete all tasks(current and future) in any number of working days? If not we must work or else skip that day. To check that we can use a segment tree to find an interval with maximal value $b_l - a_l + b_{l+1} - a_{l+1} + \ldots + b_r - a_r$. $O(nlog(n))$

- $D = N$, 18 points.

  Let's find a solution for the suffix i with the maximal sum of b_j in working days. To move from i to $i-1$: If we can complete a_{i-1} tasks in current working days use them. Else find the largest b_j in not working days and mark it as a working day until we complete all a_{i-1} tasks. $O(nlog(n))$

- $N \leq 2000$, 16 points.

  Let x_i be the number of completed tasks in days $1, 2, \ldots i$. Then x_i must be not greater than $b_1 + b_2 \ldots + b_i$ and not less than $b_1 + b_2 \ldots + b_{i-D+1}$.

  Use $dp_{i,j} = x$ where i number of days, j number of working days, x maximal number of completed tasks. Transitions are easy and don't forget to check $b_1 + b_2 \ldots + b_{i-D+1} \leq dp_{i,j} \leq b_1 + b_2 \ldots + b_i$. $O(n^2)$

- No additional constraints, 36 points

  Use the same dp from the previous subtask. The key observation is that dp_i is a convex function. Then we can use *set* to store $dp_{i,j+1} - dp_{i,j}$. And transitions are just inserting b_i. To check bounds for value x store min value and max value in dp. $O(nlog(n))$

Problem A. Дене шынықтыру

1 және 2 ішкі есеп

Барлық мүмкін 2^{2n} сөздерді қарастырайық және олардың әрқайсысын тексерейік. $n = 2$ үшін барлық жағдайларды қолмен қарауға болады.

3 ішкі есеп

Тек екі мүмкін жағдай бар $a_i = 0$ немесе $a_i = -1$. $a_i = 0$ болғанда жауап $LRLRLR\ldots$, ал $a_i = -1$ жауап $RLRLRL\ldots$ болады.

4 ішкі есеп

Егер $a_1 = 0$, онда $s_1 = L$, болмаса $s_1 = R$. Ары қарай префиксте неше L,R қойғанымызды санап, келесі әріпті анықтай аламыз.

6 ішкі есеп

Тұжырым. Lдер үшін саналған жиын, Rлер үшін саналған жиынға тең.

Дәлел. Математикалық индукция арқылы дәлелдеуге болады. Кез келген жағдайда көршілес тұрған L және R болады. Олар үшін алынған сандар бірдей екенін байқау қиын емес. Және олар басқаларға әсер етпейтіндіктен, біз оларды алып тастай аламыз. Одан кейін $n - 2$ мәселесіне көшеміз.

a массивіндегі әрбір сан жұп рет кездеседі екен. Егер әрбір сан үшін жарты кездескенің қалдырсақ, онда бұл барлық L үшін нәтиже сандары болады. Бұл массивді La деп белгілейік. Енді R таңбаларының ең аз санын пайдаланып, біз La массивінен барлық сандарды аламыз. Ол үшін массивтегі мәндерді өсу ретімен сұрыптаймыз. Және осы ретпен біз қажетті элементтерді енгіземіз.

Problem B. Батыр I және Ұлы Тима

1 ішкі есеп

Бұл ішкі есепте шеңбер бойымен сағат тілімен немесе сағат тіліне қарсы қозғалатыныңызды көру оңай, сондықтан жауап $|x_i - y_i|$ немесе $L - |x_i - y_i|$ арасындағы ең азы болады.

2 ішкі есеп

Әр сұрау үшін график құрастырайық, сонда Батырдың жолдарының салмағы 1, Тиманың жолдарының салмағы 0. Жауапты табу үшін ең қысқа жол алгоритмдерін қолданамыз.

3 ішкі есеп

Екінші ішкі тапсырмадағы сияқты сұрауларды өңдеуден бұрын графты құрастырайық, содан кейін жауап табу үшін сұраудағы әрбір жұп үшін BFS немесе Dijkstra іске қосыңыз.

4 ішкі есеп

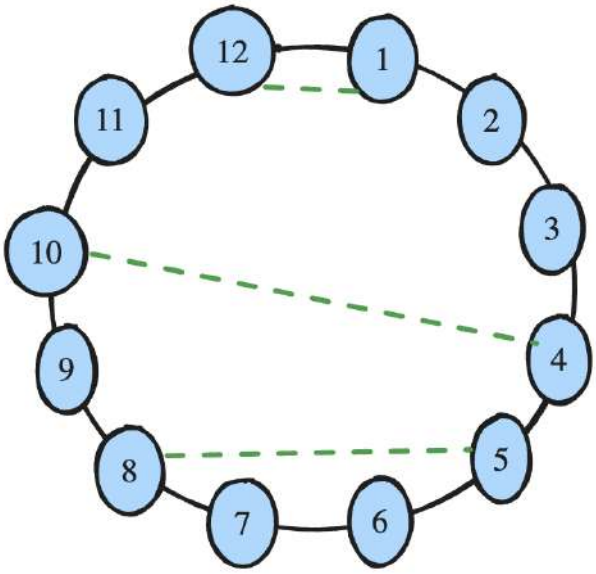
Алдыңғы ішкі тапсырмада сұраулардың бірінде немесе Тиманың жолдарының шеттерінде ғана пайда болатын түйіндермен граф құруға болатынын ескеріңіз. сондықтан бізде $2(m + q)$ түйіндері бар, оларда біз графті құра аламыз және үшінші ішкі тапсырмаға ұқсас шеше аламыз.

5 ішкі есеп

Тиманың жолдарын сегменттер ретінде қарастырыңыз. Ешбір екі сегменттің қиылыспайтынын және олардың ешқайсысы бір-бірін толығымен жабатынын байқаңыз. x-дан y-ға жылжытқанда, өтетін қашықтық $y - x - \text{sum}$ тең болады, мұнда sum жалпы ұзындығы $\sum(b_i - a_i)$ $[x, y]$ ауқымында толығымен жататын сегменттер.

6 ішкі есеп

Тиманың жолдарын сегменттер ретінде қарастырыңыз. Ешбір сегмент қиылыспаса да, олардың әрқайсысы бір-бірін жабады, бұл олардың арасындағы иерархияны білдіреді. $[0, L]$ сегментін қосыңыз, содан кейін осы сегменттер жиынында ағаш диаграммасын жасаңыз. Әрбір сегмент үшін оны жабатын ең қысқа сегментті анықтаңыз, ол ағашта оның ата-анасы болады. Сонымен қатар, бұл ағашта әрбір нүкте оны жабатын ең қысқа сегменттің бөлігі болып табылады. Бұған сканерлеу сызығының көмегімен қол жеткізіңіз: сегменттерді сол жағы бойынша сұрыптаңыз және негізгі элементті оңай табу үшін сегменттер дестесін сақтаңыз. Әрбір жиектің салмағы ата-ана және ұл сегменттерінің ең жақын нүктелерінің арасындағы қашықтық болып табылады. Сұрауларды қалай өңдеу керектігін жақсырақ түсіну үшін төменде берілген графикті қарастырыңыз, мұнда $L = 12$ және Тиманың жолдары жасыл сызықтармен көрсетілген.



Түсіну үшін мысал ретінде 3 және 9 қолданайық. сондықтан 3 $[1, 12]$ сегментімен байланыстырылады. ал біздің ағашта 9 - $[4, 10]$. Сондықтан бұл жағдайда 3-дан $[4, 10]$ сегментіне, содан кейін 10-дан 9-ға өту оңтайлы, сондықтан жауап 2. Тағы 4, 9 сұрауын қарастырайық, бұл жағдайда біз 4-дан 10-ға, содан кейін 9-ға көшеміз, сондықтан жауап 1. бірінші мысалда $[4, 10]$ сегментіне көштік, ол $[4, 10]$ және $[1, 12]$ сегменттерінің son lca ($[1, 12]$), ал екіншісінде ол lca-ға барады ($[4, 10]$). Екі нүктеден lca-ға өту немесе lca ұлына, содан кейін олардың арасында бір-біріне өту оңтайлы екенін көрсетуге болады. Уақыт күрделілігі $O(nlogn)$.

7 ішкі есеп

Мәселенің осы нұсқасында, егер x_i және y_i Тиманың жолдарымен қосылса, жауап 0; әйтпесе, бұл 1. Олардың осылай қосылғанын анықтау үшін Тиманың екі жолын біріктірейік, егер олар бір-бірімен қиылыса, бірақ бір-бірін толығымен жаппаса. Алдымен жолдарды соңғы нүктелері бойынша қосыңыз. a және b жолдарын қарастырғанда, біз оларды қиып өтетін барлық жолдарды анықтауымыз керек. Мысалы, $a < b$ болса, u жолының кейбір соңғы нүктелері u a және b арасында болса және u

нүктесінің ең оң жақ соңғы нүктесі $i, iuii.b$. Керісінше, $a > b$ болса, u арасындағы соңғы нүктелер үшін u-дан сол жақ шеткі нүкте b-дан аз немесе оған тең екенін тексеруіміз керек. Біз мұндай соңғы нүктелерді сегмент ағашының көмегімен таба аламыз және оларды құрамдас бөліктер ретінде қосу үшін бөлінген жиынтық бірлестік (dsu) құрылымын пайдалана аламыз. Бұл процесті $O(n \log n)$ уақытында орындауға болады.

8 ішкі есеп

7 ішкі тапсырмаға бірдей құрамдастарды құрастырғаннан кейін біз 6 қосалқы тапсырмамен бірдей дерлік мәселе қалдырамыз, бірақ сонымен бірге 6 қосалқы тапсырмамен салыстыру компоненттері сегмент емес, нүктелер жиыны екенін ескеру қажет.

Problem C. Жалқау, бірақ адал

- $D = 1$, 5 ұпай.

  Біз алған әрбір тапсырманы сол күні орындауымыз керек. Алдымен, әрбір $b_i \leq a_i$ екенін тексеріңіз. Содан кейін $b_i \neq 0$ болатын күндер санын шығарыңыз. $O(n)$

- $N \leq 18$, 7 ұпай

  Егер біз бір күні жұмыс істеуді шешсек, мүмкіндігінше көп тапсырмаларды орындау әрқашан оңтайлы болады және біз ең ерте тапсырмалардан бастаймыз.

  Әр күн үшін бізде екі нұсқа бар: жұмыс істемеу немесе мүмкіндігінше көп тапсырмаларды орындау. Рекурсияны қолданыңыз. Әр күн үшін сол күні қанша тапсырма орындалмағанын сақтаңыз. $O(2^n * n)$

- Барлық a_i тең, 18 ұпай.

  Ашкөздік шешім: егер біз сол күні жұмыс істемеуді шешсек, кез келген жұмыс күнінде барлық тапсырмаларды (ағымдағы және болашақ) орындай аламыз ба? Әйтпесе, жұмыс істеуіміз керек немесе ол күнді өткізіп жіберуіміз керек. $b_l - a_l + b_{l+1} - a_{l+1} + \ldots + b_r - a_r$ максималды мәні бар аралықты табу үшін сегмент ағашын қолдануға болатынын тексеру үшін. $O(nlog(n))$

- $D = N$, 18 ұпай.

  Жұмыс күніндегі b_j максималды қосындысы бар i жұрнағы шешімін табайық. i-дан $i-1$-ға жылжыту үшін: a_{i-1} тапсырмаларын ағымдағы жұмыс күндерінде орындай алатын болсақ, оларды пайдаланыңыз. Әйтпесе, жұмыс істемейтін күндердегі ең үлкен b_j табыңыз және біз барлық a_{i-1} тапсырмаларын орындамайынша, оны жұмыс күні ретінде белгілеңіз. $O(nlog(n))$

- $N \leq 2000$, 16 ұпай.

  x_i $1, 2, \ldots i$ күндерінде орындалған тапсырмалар саны болсын. Сонда x_i $b_1 + b_2 \ldots + b_i$ артық емес және $b_1 + b_2 \ldots + b_{i-D+1}$ кем болмауы керек.

  $dp_{i,j} = x$ пайдаланыңыз, мұнда i күн саны, j жұмыс күні саны, x орындалған тапсырмалардың максималды саны. Өтпелер оңай және $b_1 + b_2 \ldots + b_{i-D+1} \leq dp_{i,j} \leq b_1 + b_2 \ldots + b_i$ тексеруді ұмытпаңыз. $O(n^2)$

- Қосымша шектеу жоқ, 36 ұпай

  Алдыңғы ішкі тапсырмадағы бірдей dp пайдаланыңыз. Негізгі байқау dp_i дөңес функция болып табылады. Содан кейін $dp_{i,j+1} - dp_{i,j}$ сақтау үшін *set* пайдалана аламыз. Ал ауысулар тек b_i кірістіреді. x мәнінің шектерін тексеру үшін dp ішінде минимум және максимум мәнді сақтаңыз. $O(nlog(n))$

Задача A. Физкультура

Подзадачи 1 и 2

Переберем все возможные строки длины 2^{2n} и проверим каждую из них. Для $n = 2$ все случаи можно посмотреть в ручную.

Подзадача 3

Есть только два случая $a_i = 0$ или $a_i = -1$. При $a_i = 0$, ответ будет $LRLRLR\ldots$, а при $a_i = -1$ ответом будет $RLRLRL\ldots$.

Подзадача 4

Если $a_1 = 0$, то $s_1 = L$, иначе $s_1 = R$. Далее мы можем поддерживать сколько поставили L,R на префиксе, и благодаря этому однозначно вычисляем следующий символ.

Подзадача 6

Утверждение. Мультимножество чисел полученные для L, совпадает с мультимножеством полученных для R.

Доказательство. Можно доказать с помощью математический индукции. В любом случае найдутся рядом стоящие L и R. Легко заметить, что для них полученные числа совпадают. А так как для всех других они не влияют, поэтому можем их удалить. И перейдем к задаче с $n - 2$.

Получается каждое число в массиве a встречается четное количество раз. Если для каждого числа оставим половину вхождении, то это будет полученные числа для всех L. Обозначим этот массив как La. Теперь используя минимальное количество символов R получим все числа из массива La. Для этого отсортируем значения в массиве по возрастанию. И в этом порядке будем вставлять нужные элементы.

Задача B. Батыр I и Тима Великий

Подзадача 1 $(m = 0)$ — 5 баллов

В этой подзадаче просто видеть, что вы либо двигаетесь по кругу по часовой стрелке, либо против часовой стрелки, поэтому ответ - минимум между $|x_i - y_i|$ или $L - |x_i - y_i|$.

Подзадача 2 $L, m, q \leqslant 10^2$ — 8 баллов

Давайте строить граф для каждого запроса, затем дороги Батыра имеют вес 1, а дороги Тимы - вес 0. И запустите алгоритмы поиска кратчайшего пути, чтобы найти ответ.

Подзадача 3 $L, m, q \leqslant 10^3$ — 11 баллов

Давайте построим граф перед обработкой запросов, как во второй подзадаче, а затем запустим BFS или Dijkstra для каждой пары в запросе, чтобы найти ответ.

Подзадача 4 $m, q \leqslant 10^3$ — 10 баллов

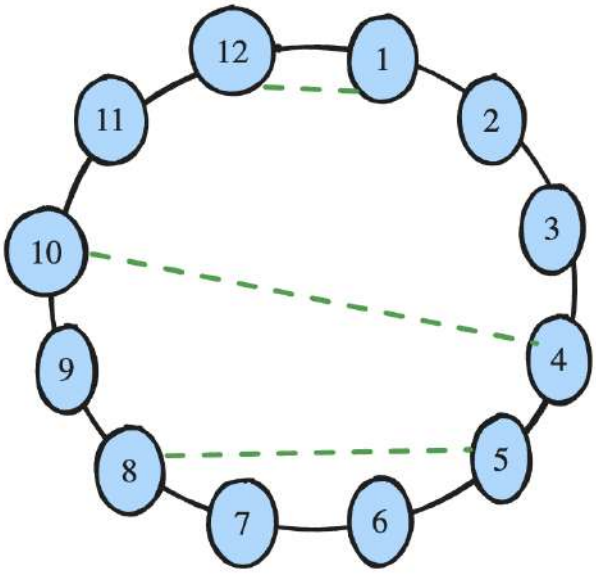
Заметим, что в предыдущей подзадаче можно построить граф с узлами, которые появляются только в одном из запросов или в ребрах Тимы. Таким образом, у нас есть $2(m + q)$ узлов, на которых мы можем построить граф, и решить аналогично третьей подзадаче.

Подзадача 5 $b_i < a_{i+1}$ — 12 баллов

Рассмотрим дороги Тимы как отрезки. Заметим, что ни один из отрезков не пересекается, и ни один из них не полностью покрывает другой. При перемещении от x к y пройденное расстояние равно $y - x - \text{sum}$, где sum - общая длина $\sum(b_i - a_i)$ отрезков, которые полностью находятся в пределах диапазона $[x, y]$.

Подзадача 6 $a_i < a_{i+1}, b_{i+1} < b_i$ — 14 баллов

Рассмотрим дороги Тимы как отрезки. Заметим, что хотя ни один отрезок не пересекается, каждый из них покрывает другой, что подразумевает иерархию среди них. Добавьте отрезок $[0, L]$ и затем постройте дерево графа на этом наборе отрезков. Для каждого отрезка определите кратчайший отрезок, покрывающий его, который становится его родителем в дереве. Кроме того, в этом дереве каждая точка является частью кратчайшего отрезка, покрывающего ее. Для этого используйте сканирующую линию: отсортируйте отрезки по их левому краю и поддерживайте стек отрезков для легкого поиска родителя. Вес каждого ребра - расстояние между ближайшими точками родительского и дочернего отрезков. Чтобы лучше понять, как обрабатывать запросы, рассмотрите данный граф, где $L = 12$, и дороги Тимы представлены зелеными линиями.



В качестве примера для понимания, давайте используем 3 и 9. Таким образом, 3 будет связан с отрезком $[1, 12]$. в то время как 9 с $[4, 10]$ в нашем дереве. Таким образом, в этом случае оптимально переместиться от 3 к отрезку $[4, 10]$, а затем от 10 к 9, поэтому ответ - 2. И рассмотрим другой запрос 4, 9, в этом случае это тривиально, мы перемещаемся от 4 к 10, а затем к 9, поэтому ответ - 1. В первом примере мы переместились к отрезку $[4, 10]$, который является сыном lca ($[1, 12]$) отрезков $[4, 10]$ и $[1, 12]$, а во втором случае мы идем к lca ($[4, 10]$). Можно показать, что оптимально либо идти от обоих точек к lca, либо идти к сыну lca, а затем между ними друг к другу. Временная сложность - $O(nlogn)$.

Подзадача 7 $|x_i - y_i| = 1$ $(1 \leqslant i \leqslant q)$ — 18 баллов

В этой версии задачи, если x_i и y_i соединены дорогами Тимы, ответ - 0; в противном случае - 1. Чтобы определить, соединены ли они таким образом, давайте объединим две дороги Тимы, если они пересекаются, но не полностью покрывают друг друга. Сначала соедините дороги их конечными точками. При рассмотрении дорог a и b нам нужно определить все дороги, которые их пересекают. Например, если $a < b$, мы считаем некоторые конечные точки дороги u допустимыми, если u находится между a и b, и самая правая конечная точка от u больше или равна b. Напротив, если $a > b$, для конечных точек u между ними нам нужно проверить, что самая левая конечная точка

от u меньше или равна b. Мы можем найти такие конечные точки, используя дерево отрезков, и использовать структуру объединения непересекающихся множеств (dsu), чтобы соединить их как компоненты. Этот процесс можно выполнить за время $O(n \log n)$.

Подзадача 8 Без дополнительных ограничений — 22 балла

После построения компонент, аналогично подзадаче 7, у нас остается почти та же задача, что и в подзадаче 6, но нужно также учитывать, что в отличие от подзадачи 6, компоненты являются набором точек, а не отрезков.

Задача C. Ленивый, но честный

- $D = 1$, 5 баллов.

  Мы должны завершить каждую задачу, которую получаем в тот же день. Сначала проверьте, что каждый $b_i \leqslant a_i$. Затем выведите количество дней с $b_i \neq 0$. $O(n)$

- $N \leqslant 18$, 7 баллов

  Если мы решаем работать в какой-то день, всегда оптимально выполнять как можно больше задач, и мы начинаем с самых ранних задач.

  Для каждого дня у нас есть два варианта: не работать или выполнять как можно больше задач. Используйте рекурсию. Для каждого дня сохраните, сколько задач осталось невыполненными в этот день. $O(2^n * n)$

- Все a_i равны, 18 баллов.

  Жадное решение: если мы решаем не работать в этот день, можем ли мы завершить все задачи (текущие и будущие) в любом количестве рабочих дней? Если нет, мы должны работать, иначе пропустить этот день. Для проверки этого мы можем использовать дерево отрезков, чтобы найти интервал с максимальным значением $b_l - a_l + b_{l+1} - a_{l+1} + \ldots + b_r - a_r$. $O(nlog(n))$

- $D = N$, 18 баллов.

  Давайте найдем решение для суффикса i с максимальной суммой b_j в рабочих днях. Чтобы перейти от i к $i-1$: если мы можем завершить a_{i-1} задач в текущих рабочих днях, используйте их. В противном случае найдите наибольшее b_j в нерабочих днях и отметьте его как рабочий день, пока мы не завершим все a_{i-1} задач. $O(nlog(n))$

- $N \leqslant 2000$, 16 баллов.

  Пусть x_i - количество выполненных задач в дни $1, 2, \ldots i$. Тогда x_i должен быть не больше $b_1 + b_2 \ldots + b_i$ и не меньше $b_1 + b_2 \ldots + b_{i-D+1}$.

  Используйте $dp_{i,j} = x$, где i - количество дней, j - количество рабочих дней, x - максимальное количество выполненных задач. Переходы просты, и не забудьте проверить $b_1 + b_2 \ldots + b_{i-D+1} \leqslant dp_{i,j} \leqslant b_1 + b_2 \ldots + b_i$. $O(n^2)$

- Никаких дополнительных ограничений, 36 баллов

  Используйте тот же dp из предыдущего подпункта. Ключевое наблюдение заключается в том, что dp_i является вогнутой функцией. Затем мы можем использовать *set*, чтобы хранить $dp_{i,j+1} - dp_{i,j}$. И переходы - просто вставка b_i. Чтобы проверить границы для значения x, сохраните минимальное значение и максимальное значение в dp. $O(nlog(n))$

Problem D. Watermelons

Let's, for convenience, immediately exclude the case for all subtasks when $n+m<K$, since the answer here is -1, because each friend should receive at least one watermelon.

Subtask 1 ($m=0$, $a_i=a_{i+1}$ $(1\le i<n)$) — 9 points

Since all a_i are equal, let's define the number $x=a[1]$. In this subtask, it is sufficient to evenly distribute all n watermelons among K friends, then the distribution will be fair, because the difference in watermelons between any two friends is no more than x, and the sum of the watermelons of the remaining friends is at least $2\cdot x$, due to $n\ge 3$. This can be easily done, for example, as follows: first, distribute $\lfloor\frac{n}{K}\rfloor$ watermelons to all friends, and then distribute one watermelon to friends with numbers from 1 to $n \bmod K$.

Subtask 2 ($a_i=a_{i+1}$ $(1\le i<n)$, $b_j=b_{j+1}$ $(1\le j<m)$) — 19 points

This subtask can be divided into three cases:

1. If $K\le n$, then the solution is exactly the same as in subtask 1.
2. If $K=n+1$, then distribute one watermelon a_1 to the first n friends, and one watermelon b_1 to the $(n+1)$-th friend. If it turns out that $b_1>n\cdot a_1$, then give one more watermelon b_1 to the 1-st friend (if in this case $m=1$, then the answer is -1). In both cases, the distribution will be fair.
3. If $K\ge n+2$, then distribute one watermelon a_1 to the first n friends, and one watermelon b_1 to the remaining $K-n$ friends. Then the distribution will be fair, because there are at least 2 friends with a_1 and at least 2 friends with b_1.

Subtask 3 (It is not necessary to output the distribution of watermelons) — 24 points

The solution to this subtask makes a significant contribution to solving the full problem. Let's say we bought v watermelons from the seller, and received a new array of watermelons c of $n+v$ watermelons. Now we want to know if it is possible to make a fair distribution with the current set of watermelons.

Claim: If $2\cdot maxC\le\sum_{i=1}^{n+v}c_i$, then the answer exists ($maxC$ – the maximum number in the array c). Let's call this condition the *general* condition.

Proof: If $2\cdot maxC>\sum_{i=1}^{n+v}c_i$, then there is no answer, because $maxC>\sum_{i=1}^{n+v}c_i-maxC$ contradicts a fair distribution. Otherwise, we will show that it is enough to have 3 friends for a fair distribution, and if there are more friends, it cannot spoil anything. Let's say we have 3 friends. Let's define S_i – the sum of the weights of watermelons of the i-th friend. Let's first sort the watermelons in non-decreasing order. Give the watermelon c_{n+v} (i.e., one watermelon with the maximum weight) to the first friend. Now start distributing the watermelons with numbers from 1 to $n+v-1$ one by one to the remaining two friends according to the following algorithm: give the i-th watermelon to the 2-nd friend if at the moment $S_2\le S_3$, otherwise, give it to the 3-rd friend. After such a distribution, it will be true that $|S_2-S_3|\le maxC$, because when we give the i-th watermelon to the 2-nd friend, then before the transfer, $S_2\le S_3$, and after the transfer, it cannot be that $S_2+c_i>S_3+maxC$, because $c_i\le maxC$ (and vice versa, if we give it to the 3-rd friend). In the end, it will turn out that $S_1=maxC$, and $|S_2-S_3|\le maxC$. Now it remains only to prove that $S_1\le S_2+S_3$, $S_2\le S_1+S_3$, and $S_3\le S_1+S_2$. According to the initial condition, we know that $maxC\le\sum_{i=1}^{n+v}c_i-maxC$, therefore $S_1\le S_2+S_3$. Since the cases for S_2 and S_3 are analogous, let's assume that $S_3\ge S_2$, and prove the correctness for S_3. Since $|S_3-S_2|\le maxC$, then $S_3\le S_2+maxC$, which proves a fair distribution. The remaining $K-3$ friends can be given one watermelon from S_2 or S_3 each, so that S_2 and S_3 do not remain empty. Obviously, if S_2 and S_3 satisfy the condition of a fair distribution, then one watermelon from S_2 or one watermelon from S_3 cannot fail to satisfy the condition.

Now we need to find the minimum number of elements from the array b that, when obtained, will give us an array c that satisfies the general condition. The answer is 0 if the array a already satisfies the general

condition, and $n \geq K$. Otherwise, let's sort the array b in non-decreasing order, and iterate through the maximum b_i we want to take. It is obvious that the most optimal next element of the array b that we will take is b_{i-1}, because if the element does not change the maximum, then for the general condition it is better for the sum to be as large as possible. Thus, if we have fixed the maximum b_i, we will go from right to left until the general condition is met, or we have fewer watermelons than K. Thus, we will find the maximum j for which, if we buy watermelons $b_j, b_{j+1}, \ldots, b_i$, we will be able to make a fair distribution. It is not difficult to understand that j can be found by *binary search* or *two-pointer method*, because if j fits, then $j-1$ will also fit. In the end, from all suitable $i-j+1$, we will output the minimum (if none fits, then -1).

Subtask 4 ($m = 0$) — 20 points

From subtask 3, we know the general condition, which determines whether there is an answer for a specific set of watermelons or not. If the answer exists, then it can be restored exactly as written in the Proof of the general condition in subtask 3. That is, first fill in S_1, S_2, S_3 for the first three friends, and then distribute one watermelon from S_2 and S_3 to the remaining friends.

Subtask 5 ($b_i = 1$ $(1 \leq i \leq m)$) — 10 points

Since all elements of the array b are equal, we only need to know the number of watermelons we will buy. It is sufficient to iterate through how many watermelons we will buy, check the general condition from subtask 3, and restore the answer as in subtask 4.

Subtask 6 (No additional constraints) — 18 points

From subtask 3, we know how many watermelons need to be bought and the segment of watermelons that need to be bought in the sorted array b. From subtask 4, we know how to restore the answer if we have a specific set of watermelons. By combining these solutions, we can get a complete solution.

Problem E. Candies

Subtask 1

Since all a_i are equal to 1, the answer for each query is $r_i - l_i + 1$, which is the length of the interval.

Subtask 2

Due to the fact that each friend receives the same set of candies as other friends, each friend's set is equal to the interval's set. Furthermore, to ensure that after giving candies to the first friend other friends could receive the same type of candies, set of candies on the interval's prefix must be equal to the set of elements after this prefix.

Hence, it can be concluded that greedily picking the first prefix of candies that satisfies the requirement above is a valid strategy. Let's call the process of picking the required prefix as "jump". Now, this idea can be implemented for this subtask.

Subtask 3

For each query, there can be two solutions depending on the number of distinct candies. If there is only a single type of candy, then it is the same as in subtask 1.

Otherwise, there can be only two types of candies on the interval. One can pre-calculate the closest position of the next distinct candy for each position, and use it to optimize solution of subtask 2. Now, it will work in O(ans), where ans is the answer for the query. However, it is too slow, so binary lifting could be used to answer queries in $O(log(n))$.

Subtask 4

If the length of query's interval is odd answer is equals to 1, because there would be some element that has unique occurrence. Otherwise, the answer could be 2 if and only if the set of candies on the first half of the interval is equal to the set of the second half. Counting number of distinct elements is a famous task that can be done in $O(n \cdot log(n) + q \cdot log(n)))$ by using segment tree of fenwick tree.

Subtask 5

Observe, that after giving candies to the first friend, we left with the same task but for the new interval. Also, since each friend's set of candies is equal to the interval's set the number of distinct candies is the same for both of them.

Now, define $prefAns_i$ as the answer for the prefix ending at position i. Using the greedy strategy, if we can find the closest left position j where number of unique candies on interval $(j, i]$ is the same as on $[1, i]$, then $prefAns_i = prefAns_j + 1$ if number of unique candies on interval $[1, j]$ is the same as $[1, i]$. Otherwise, it will be equal to 1. Number of unique candies for each prefix could be calculated in $O(n)$, and position j could be found in $O(log(n))$ by keeping the right-most occurrence of each candy in $std :: set$.

Subtask 6

We can use the same solution from subtask 2, but do a little modification for it. Since number of unique candies on the interval cannot be larger than 100, one can answer queries when number of unique candies on query is equal to K independently. It is enough to pre-calculate for each i the next closest position where number of unique candies will be equal to K, which can be done in $O(n)$ using two pointers, and do

dfs and binary search, or binary lifting to answer queries. As result, it will be done in $O(n \cdot D + q \cdot log(n))$, or $(n \cdot D \cdot log(n))$ depending on the realization, where D is number of distinct candies.

Subtask 7

We are going to use SQRT optimization. Suppose the number of unique candies on the query's interval is less that $\sqrt{n}$, then we can use the solution from subtask 6 to answer all those queries in $O(n \cdot \sqrt{n} + q \cdot log(n))$.

As for the other case, the answer can not be greater than $\sqrt{n}$, because each friend gets an interval with at least $\sqrt{n}$ elements.

Let's fix some R, and answer all of the queries that end in the position R. For current R we will maintain an array $next_i$. It will contain the next closest position where number of unique elements is the same as on the interval $[i, R]$ (this array stores information about the intervals of candies that our friends get). Using this array we can "jump"to the next interval and find an answer in no more than $\sqrt{n}$ "jumps".

However, we need to update this array when we increment R. To do so we need to observe that $next_i$ is going to change only for positions after the previous occurrence of a_{R+1}. For such positions we have to assign $next_i = R+1$. If we do this updates and keep track of unique numbers using SQRT decomposition, we can get $next_i$ and number of unique numbers on interval $[i, R+1]$ in $O(1)$. Overall we get a solution that works in $O(q \cdot \sqrt{n} + n \cdot \sqrt{n})$.

Subtask 8

To solve the last subtask we will use different approach. Let's solve it using Divide And Conquer algorithm. We define recursive function $solve(L, R)$, this function finds an answer for all queries i such that $l_i \leq M$ and $M < r_i$, where $M = [\frac{L+R}{2}]$. For the queries that lie in the left half of the $[L, R]$ we will recursively call $solve(L, M)$, for other queries we will use $solve(M + 1, R)$.

Now, we have to observe that part of the query that lies in the left side of $[L, R]$ is independent from the right side. It can be obtained this way: notice that if the set of $[l_i, M]$ is the same as the set of $[l_i, r_i]$ our greedy approach will make "jumps"independently from the right side of the query (similar applies for the right side). Additionally, if the set of $[l_i, M]$ is smaller than the set of $[l_i, r_i]$ we can ignore "jumps"from the left side, and use "jumps"from the right side.

Now we can solve problem independently on the suffixes of $[L, M]$ and the prefixes of $[M + 1, R]$. This problem is similar to the Subtask 5. For the last detail we have to store the last "jump"from the left side and the last "jump"from the right side, as well as the number of jumps, and the number of different elements on the $[l_i, M]$ and $[M + 1, r_i]$. It is easy to store this values by applying small changes to the solution of the Subtask 5, however, we can make it slightly faster by using "two pointers"instead of $std :: set$.

The last detail that is left is the last "jump"from the left side and the last "jump"from the right side. We have to merge this to "jumps"because sometimes they form a valid interval, in this case answer for the query should be incremented.

Problem F. Alikhan and studying

This problem is about calculating the sum of k-th largest distance from a point on an edge to some subset of vertices and summing it up over all integer points on that edge. Let's call this subset of vertices as important vertices. Let's also denote the shortest distance from u to v by $d(u, v)$. It's pretty straightforward to precalculate all pairwise shortest distances $d(u, v)$ by starting n Dijkstra's algorithms from each of the vertex in $O(n(n+m)\log m)$ time, which is fast enough with the problem's constraints.

Group 1 $(S = 1)$ — 8 points

In this test group, exactly one vertex is important (let's call it A), thus $k = 1$ and we need to calculate the sum of distances from all points on the edge (u, v, w) to this important vertex A.

Let's understand how the shortest path from the point x on the edge (u, v, w) to the vertex A might look like. Denote $d(x, u) = \alpha$, then $d(x, v) = w - \alpha$. Obviously,

$$\begin{aligned} d(x, A) &= \min(d(x, u) + d(u, A), d(x, v) + d(v, A)) \\ &= \min(\alpha + d(u, A), w - \alpha + d(v, A)). \end{aligned}$$

If we draw this function on a coordinate plane with x-axis being the value of α (on the interval $[0, w]$) and the y-axis being the value of $d(x, A)$, then we should get a piecewise linear function, consisting of a line segment with slope equal to 1 and a line segment following it with slope equal to -1.

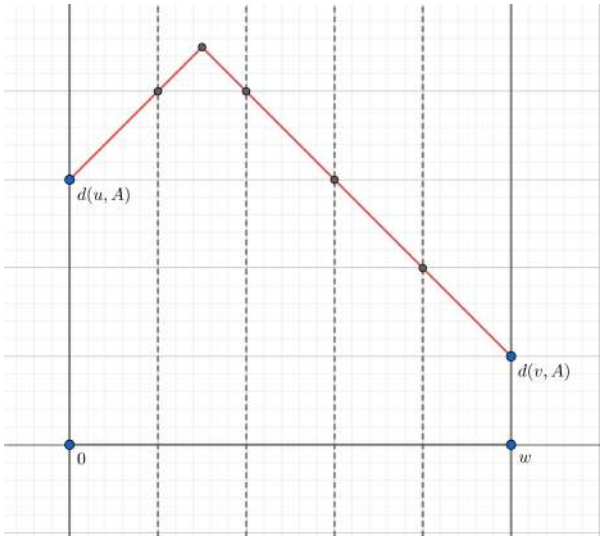


Рис. 1: Plot of the function $f(\alpha) = d(x, A)$, where x is the point on (u, v, w) such that $d(u, x) = \alpha$. The answer is the sum of the values of this piecewise linear function over integer points.

Note that the peak of this function is a point x, such that $d(x, u) + d(u, A) = d(x, v) + d(v, A)$, or, equivalently,

$$\alpha = \frac{d(v, A) + w - d(u, A)}{2}.$$

Hence, we can easily obtain all of the corner points of this piecewise linear function in $O(1)$ time by simply computing the formula stated above and the only thing left to do is to find the required sum of distances. This can be easily done by considering each straight line segment independently and carefully handling double counting. For a single line segment $(L_x, L_y) \to (R_x, R_y)$ we can round the left endpoint up and the right endpoint down. Then, if the line is ascending (which means its slope is 1), the answer will look like:

$$\sum_{x=L_x}^{R_x} (L_y + x - L_x) = L_y \cdot (R_x - L_x + 1) + \sum_{x=0}^{R_x - L_x} x = L_y \cdot (R_x - L_x + 1) + \frac{(R_x - L_x)(R_x - L_x + 1)}{2}.$$

The same formula can be obtained if the line is descending.

To sum up, for each query, we need to find the peak of the mentioned above piecewise linear function, divide it into two straight line segments, calculate answers on each of them, add them up, and subtrack double counting part if needed. This works in $O(n(n+m)\log m + q)$, since all of the computations are done in $O(1)$.

Group 2 $(n, q \le 100, L = 1)$ — 8 points

In this test group, the constraints for n and q are pretty small and all of the edges have length of exactly 1.

For this subtask, we only need to consider two cases: $x = u$ and $x = v$. For each of them, we need to sort all of the important vertices by their distance to the corresponding vertex u or v and obtain the k'th largest distance.

Total asymptotics will be $O(n(n+m)\log m + qn\log n)$.

Group 3 $(n, L \le 100, q \le 10^3)$ — 12 points

In this test group, the constraint for n is still small, but edge lengths can now be up to 100 (but it's still small enough) and the number of queries is small $q \le 10^3$.

Such constraints lead to a slow solution which iterates over all possible x points and finds k'th distance to important points directly as in the previous subtask. Recall that $d(x, A) = min(\alpha + d(u, A), w - \alpha + d(v, A))$, so for each point x we can calculate all distances from that point to important points and get the k'th largest distance by explicitly sorting these values.

Total asymptotics will be $O(n(n+m)\log m + qLn\log n)$.

Group 4 $(m = n-1, u_i = i, v_i = i+1, k_j = 1)$ — 10 points

In this test group, our graph is a bamboo (with no constraints on its size) $1 \to 2 \to \ldots \to n$ and $k_j = 1$, which means that we are interested in finding only the farthest vertex from x among important vertices.

Let's consider an edge $(i, i+1, w)$ and fix a point x on it. To find the farthest important vertex from point x, we, obviously, need to consider only the leftmost important vertex among $1, 2, \ldots i$ and the rightmost important vertex among $i{+}1, i{+}2, \ldots, n$. Since they don't depend on the choice of x, we can simply find them by storing all important vertices in a set and get the minimum and maximum value from that set (or even just by iterating over all vertices from 1 to n, since overall constraints allow that). After finding these leftmost and rightmost important vertices l and r, we again end up with a piecewise linear function:

$$\begin{aligned} f(\alpha) &= \max(d(x,i) + d(i,l), d(x,i+1) + d(i+1,r)) \\ &= \max(\alpha + d(i,l), w - \alpha + d(i+1,r)). \end{aligned}$$

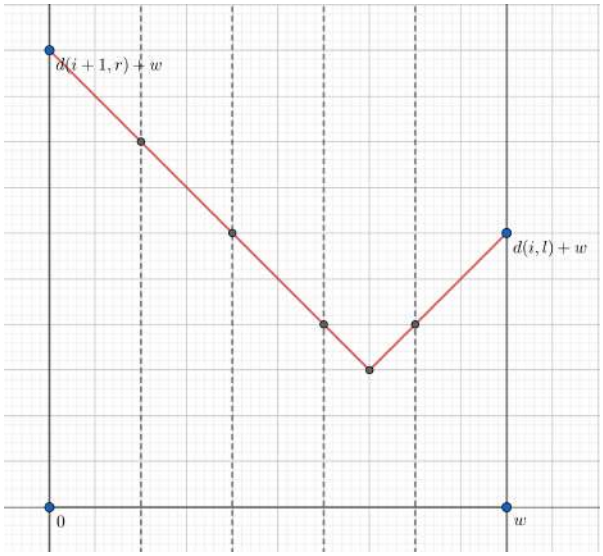


Рис. 2: Plot of the function $f(\alpha) = d(x, A)$, where x is the point on $(i, i{+}1, w)$ such that $d(i, x) = \alpha$, l and r are the leftmost and rightmost important vertices repsectively. The answer is the sum of the values of this piecewise linear function over integer points.

Recall that almost the same construction appeared in the first subtask and can be solved in exactly the same way, except for finding the middle corner point.

Total asymptotics for this test group will be $O(n(n+m)\log m + q\log n)$.

Group 5 $(m = n-1, k_j = 1)$ — 15 points

In this test group, our graph is a tree (with no constraints on its size) and $k_j = 1$, which means that we are interested in finding only the farthest vertex from x among important vertices.

Let's consider an edge (u, v, w) and fix a point x on it. To find the farthest important vertex from x, we, obviously, need to only consider the farthest important vertex from u in the subtree of u and the farthest important vertex from v in the subtree of v, since the farthest path from x passes either through u, or

through v to the correspoing farthest vertex in the subtree. To accomplish this, we can iterate over all important vertices A and check in which of the two subtrees they are (if $d(u, A) + w = d(v, A)$, A lies in the subtree of u and vice versa) and update the farthest vertex in that subtree correspondingly. After finding these two vertices, the solution goes exactly the same as in the previous subtask 4.

Total asymptotics will be $O(n(n+m)\log m + qn)$.

Group 6 ($k_j = 1$) — 24 points

In this test group, for each point x we only need to consider the farthest important vertex.

Let's consider an edge (u, v, w) and a point x on it. Recall from Figure 3 how the distance from point x to an important vertex A looks like when we move x from u to v. It is a piecewise linear function that starts as a line with slope 1 and ends as a line with slope -1. Let's denote by $A_1, \ldots, A_l$ all of the important vertices. If we draw such graph for each A_i, we would need to consider only the points on the upper bound of all of these distances $d(x, A_i)$.

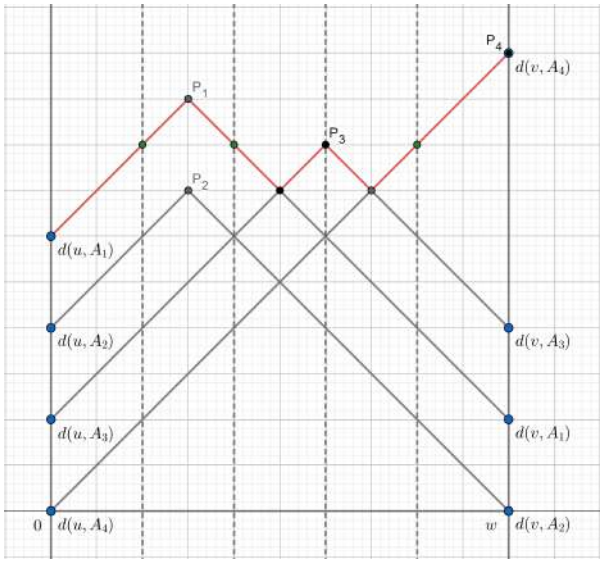


Рис. 3: Plot of the distances $d(x, A_i)$ for all important vertices $A_1, \ldots, A_l$. The upper bound is highlighted in red. The answer is the sum of the values of the upper bound piecewise linear function over integer points.

Note that if there are two vertices A_i and A_j such that $d(u, A_i) \geq d(u, A_j)$ and $d(v, A_i) \geq d(v, A_j)$, then the graph of $d(x, A_i)$ is always superior over the graph of $d(x, A_j)$, thus, we can leave A_j out of the consideration. Let's order important points so that $d(u, A_1) > \ldots > d(u, A_l)$. Then, by the remark above, we get $d(v, A_1) < \ldots < d(v, A_l)$. It's easy to see that each of these remaining vertices will contribute some part in the resulting upper bound. Moreover, their parts will be ordered in exactly the same order $(A_1, \ldots, A_l)$.

To obtain these vertices and their corresponding order, we need to precalculate for each vertex v a list of all vertices, sorted by their distance to v in descending order. This can be done in $O(n^2 \log n)$ time. If we have such a list, then, to obtain all superior points, we can simply iterate over the precalculated list for the vertex u and maintain a stack of vertices as it is done in the algorithm for finding convex hull of a set of points on plane.

To get all of the corner points of the upper bound, we need to calculate the peak graphs for each A_i and the intersections of graphs A_i and A_{i+1}, totally being done in $O(l) = O(n)$ time.

Finally, after computing all of the corner points of the upper bound of the target function, we can do the same thing as in the subtask 1 (calculate the answer independently on each line subsegment and handle double counting).

Total asymptotics will be $O(n(n+m)\log m + n^2 \log n + qn)$.

Group 7 (no additional constraints) — 23 points

In this test group, no additional constraints are given, so we need to solve the full problem.

Let's consider an edge (u, v, w) and a point x on it. Let's say that $(A_1, \ldots, A_l)$ are all important vertices. Let's draw graphs for each of them as we did in the previous subtask, but in this case we cannot delete inferior vertices since they still can contribute to the answer.

Notice that if we move x from u to v, k'th maximum value moves continuously as a piecewise linear function. It can be proven by considering how this value changes at the points of intersection of some of the graphs and at the peaks of these graphs.

At first glance, it might be very difficult to track this piecewise linear function directly from u to v. To simplify it, we partition the interval $[0, w]$ using the x-coordinates of the peaks for each A_i and do a scanline over the resulting subintervals for x.

Let's look at how the graphs look like on a single scanline subinterval. We must have t lines with slope 1 and $n - t$ lines with slope -1 for some t. Obviously, we need at most k highest lines with slope 1 and at most k highest lines with slope -1, since lower lines will never contribute to the k-th maximum value. We can obtain these lines by maintaining a set of lines with slope 1 and a set of lines with slope -1 during our scanline in $O(n \log n)$ time and get k highest lines from each set in $O(k+\log n)$ time for each subinterval (iterate over a set from its end). By doing so, we obtain a vector of at most k lines with slope 1 and a vector of at most k lines with slope -1, both of them sorted in descending order.

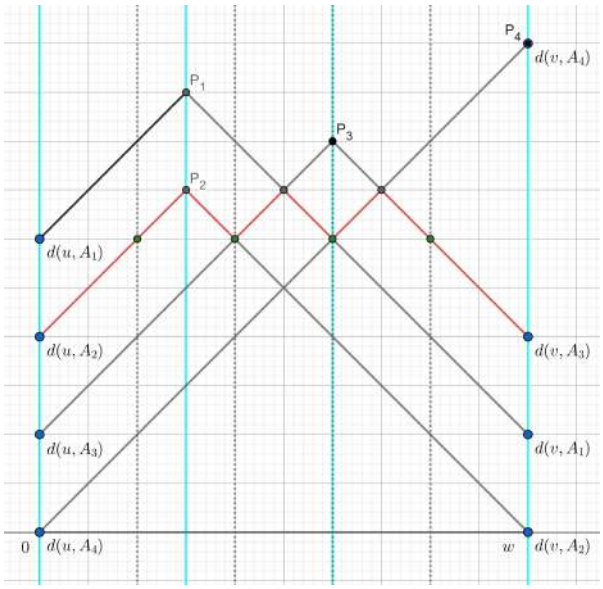


Рис. 4: Plot of the distances $d(x, A_i)$ for all important vertices $A_1, \ldots, A_l$. The k'th distance function for $k = 2$ is highlighted in red, scanline partition events are highlighted in blue. The answer is the sum of the values of the red piecewise linear function over integer points.

Now we'll show that we can completely construct the k-th maximum value function on this subinterval in $O(k)$ time.

First, we need to find its leftmost point. It appears to be the k-th maximum value among the intersections of lines with slope 1 and -1 with the left partition line of the scanline. It can be found in $O(k)$ time using merge sort, since we store these lines in vectors in descending order. Further in the solution, we'll use 0-based indexing of these vectors. Assume that k-th maximum starts on a descending j-th line (see Figure 3 on the second subinterval with $j = 1$). It's easy to see that the first ascending line it is going to intersect will be $i = k - j - 1$ ascending line. It's also easy to see that the order of lines for k-th maximum during the movement of x from left to right will alternate between ascending and descending lines with increasing i and descreasing j. The same can be said if our starting line is ascending. Thus, coupled with the fact that we currently have at most $2k$ lines, all of the corner points of the piecewise linear function of k-th maximum value for this subinterval can be constructed in $O(k)$ time.

After finding exact form of the k-th maximum value function on each subinterval, we can simply merge them and obtain the function over the entire interval $[0, w]$. After that, by doing the same thing as in the previous subtask, we can calculate the final answer. Moreover, since each line can contribute to the k-th maximum function at most once, this function will contain at most $2n$ line segments (but it's irrelevant for our solution).

Total asymptotics will be $O(n(n + m) \log m + qn(k + \log n))$.

Model solution

Author's model solution can be found in the following link:

`https://gist.github.com/dxtvzw/3d6604238d1f267cef28b88ce9f93849`

Problem D. Қарбыздар

Ыңғайлы болу үшін барлық ішкі есептер үшін бірден $n+m<K$ болған жағдайды кесіп алайық, өйткені мұнда жауап -1, өйткені әрбір дос кем дегенде бір қарбыз алуы керек.

1-ішкі есеп ($m=0$, $a_i=a_{i+1}$ $(1\le i<n)$) — 9 ұпай

Барлық a_i тең болғандықтан, $x=a[1]$ санын анықтайық. Бұл ішкі есепте барлық n қарбыздарды K достарының арасында біркелкі бөлу жеткілікті, сонда бөлу әділ болады, өйткені кез келген екі достың арасындағы қарбыздың айырмашылығы x-тен аспайды және қалған достардың қарбыздары $2\cdot x$ кем емес, себеі $n\ge 3$. Мұны оңай жасауға болады, мысалы: біріншіден, біз $\lfloor\frac{n}{K}\rfloor$ қарбыздарын барлық достарға таратамыз, содан кейін достарына 1-ден бастап $n\ mod\ K$-ға дейінгі достарға бір қарбыздан береміз.

2-ішкі есеп ($a_i=a_{i+1}$ $(1\le i<n)$, $b_j=b_{j+1}$ $(1\le j<m)$) — 19 ұпай

Бұл ішкі есепті үш жағдайға бөлуге болады:

1. Егер $K\le n$ болса, онда шешім 1-ішкі есептегідей болады.
2. Егер $K=n+1$ болса, онда бірінші n достарға бір a_1 қарбызынан таратамыз, ал $(n+1)$-ші досқа b_1 қарбызын таратамыз. Егер $b_1>n\cdot a_1$ болып шықса, онда 1-досына тағы бір қарбыз b_1 береміз (егер бұл жағдайда $m=1$ болса, онда жауап -1 болады). Екі жағдайда да бөлу әділ болады.
3. Егер $K\ge n+2$ болса, онда бірінші n достарға бір қарбыз a_1, ал қалған $K-n$ достарына бір қарбыз b_1 таратамыз. Сонда бөлу әділ болады, себебі a_1 бар кем дегенде 2 достар және b_1 бар кем дегенде 2 достар бар.

3-ішкі есеп (Қарбыздың таралуын көрсету міндетті емес) — 24 ұпай

Бұл ішкі есепті шешу толық есепті шешуге үлкен үлес қосады. Біз v сатушысынан қарбыз сатып алдық және $n+v$ қарбызынан c жаңа қарбыз алдық делік. Енді қазіргі қарбыз жиынтығымен әділ бөлуге бола ма, жоқ па, соны білгіміз келеді.

Мәлімдеме: $2\cdot maxC\le\sum_{i=1}^{n+v}c_i$ болса, онда жауап ($maxC$ — c массивіндегі ең үлкен сан). Бұл шартты *маңызды* деп атайық.

Дәлелдеу: $2\cdot maxC>\sum_{i=1}^{n+v}c_i$ болса, онда жауап жоқ, себебі $maxC>\sum_{i=1}^{n+v}c_i-maxC$ әділ бөлуге қайшы келеді. Әйтпесе, әділ бөлу үшін 3 достардың болуы жеткілікті екенін көрсетеміз, ал егер достар көп болса, бұл ештеңені бұза алмайды. Бізге 3 дос болсын. S_i i-ші досының қарбыздарының салмағының қосындысы ретінде анықтайық. Алдымен қарбыздарды кемусіз ретімен сұрыптайық. Бірінші досымызға қарбыз c_{n+v} (яғни салмағы ең жоғары бір қарбыз) берейік. Енді солдан оңға қарай 1-ден $n+v-1$-ға дейінгі сандары бар қарбыздарды келесі алгоритм бойынша қалған екі досқа тарата бастаймыз: біз i-ші қарбызды келесі алгоритм бойынша береміз. 2-ші дос, егер қазір $S_2\le S_3$ болса, әйтпесе досымызға 3 береміз. Мұндай бөлуден кейін $|S_2-S_3|i\le maxC$, өйткені i-ші қарбызды 2-досымызға бергенде, аударым алдында $S_2\le S_3$ болған, ал аударылғаннан кейін $S_2+c_i>i.S_3+maxC$, себебі $c_i\le maxC$ (және керісінше, егер біз досымызға 3 берсек). Соңында $S_1=maxC$ және $|S_2-S_3|\le maxC$. Енді $S_1\le S_2+S_3$, $S_2\le S_1+S_3$ және $S_3\le S_1+S_2$ екенін дәлелдеу ғана қалды. Бастапқы шарт бойынша $maxC\le\sum_{i=1}^{n+v}c_i-maxC$, сондықтан $S_1\le S_2+S_3$ екенін білеміз. S_2 және S_3 жағдайлары ұқсас болғандықтан, $S_3\ge S_2$ деп есептейік және S_3 дұрыстығын дәлелдейміз. $|S_3-S_2|\le maxC$, содан кейін $S_3\le S_2+maxC$, бұл әділ бөлуді дәлелдейді. S_2 және S_3 бос қалмас үшін, қалған $K-3$ достарына S_2 немесе S_3-дан бір қарбыз беруге болады. Әлбетте, егер S_2 және S_3 әділ бөлу шартын қанағаттандырса, онда S_2-дан бір қарбыз немесе S_3-дан бір қарбыз шартты қанағаттандыра алмайды.

b массивінен элементтердің ең аз санын табу қалады, оны алғаннан кейін бізде маңызды шартты қанағаттандыратын c массиві болады. Жауап 0 болады, егер a массиві маңызды шартты қанағаттандырса және $n \geq K$. Әйтпесе, b массивін кемімейтін ретпен сұрыптайық және біз алғымыз келетін ең көп b_i итерациясын жасайық. Әлбетте, b массивінің келесі ең оңтайлы элементі b_{i-1} болып табылады, өйткені егер элемент максимумды өзгертпесе, маңызды шарт үшін қосындының соншалықты үлкен болғаны жақсы. мүмкіндігінше. Осылайша, егер біз максималды b_i белгілеген болсақ, онда маңызды шарт орындалғанша оңнан солға қарай жүреміз немесе бізде K-дан аз қарбыз бар. i үшін біз $b_j, b_{j+1}, \ldots, b_i$ қарбыздарын сатып алсақ, әділ бөлуге болатындай максимум j табамыз. j мәнін *екілік іздеу* немесе *екі көрсеткіш әдісі* арқылы табуға болатынын түсіну оңай, себебі j сәйкес келсе, $j-1$ да сәйкес келеді. Соңында барлық қолайлы $i-j+1$ ішінен минималдысын шығарамыз (егер олардың ешқайсысы сәйкес келмесе, -1).

4-ішкі есеп ($m = 0$) — 20 ұпай

3-ішкі есебінен біз белгілі бір қарбыз жиынтығы үшін жауап бар-жоғын анықтайтын маңызды шартты білеміз. Жауап бар болса, оны 3-ішкі есебінде маңызды шартты дәлелдеуде жазылғандай қалпына келтіруге болады. Яғни, бірінші үш досқа S_1, S_2, S_3 толтырамыз, сосын S_2 және S_3-дан қалған достарға бір қарбыздан таратамыз.

5-ішкі есеп ($b_i = 1$ $(1 \leq i \leq m)$) — 10 ұпай

b массивінің барлық элементтері тең болғандықтан, біз тек сатып алатын қарбыз санына мән береміз. Қанша қарбыз сатып алатынымызды тексеріп, 3 ішкі есебінен маңызды жағдайды тексеріп, 4 ішкі есебіндегідей жауапты қалпына келтірсек жеткілікті.

6-ішкі есеп (Қосымша шектеулер жоқ) — 18 ұпай

3-ішкі есебінен біз қанша қарбыз сатып алу керектігін және b сұрыпталған массивінде сатып алу қажет қарбыз бөлімшесін білеміз. 4-ішкі есебінен біз белгілі бір қарбыз жинағы болса, жауапты қалай қалпына келтіру керектігін білеміз. Осы шешімдерді біріктіру арқылы толық шешімді алуға болады.

Problem E. Кәмпиттер

1-ішкі есеп

Барлық a_i 1 болғандықтан, әрбір сұрауға жауап $r_i - l_i + 1$ болады, бұл интервал ұзындығы.

2-ішкі есеп

Әрбір дос басқа достар сияқты кәмпиттер жиынтығын алатындықтан, әр достың жинағы интервал жиынтығына тең. Сонымен қатар, бірінші досқа кәмпит берілгеннен кейін басқа достар кәмпиттің бір түрін ала алатынын қамтамасыз ету үшін интервал префиксіндегі кәмпиттер жиынтығы осы префикстен кейінгі элементтер жиынына тең болуы керек.

Осылайша, жоғарыда аталған талапты қанағаттандыратын бірінші кәмпит префиксін ашкөздікпен таңдау дұрыс стратегия деп қорытынды жасауға болады. Қажетті префиксті таңдау процесін «секіру» деп атаймыз. Енді бұл идеяны осы ішкі есеп үшін іске асыруға болады.

3-ішкі есеп

Әр сұрау үшін әртүрлі кәмпиттер санына байланысты екі шешім болуы мүмкін. Егер кәмпиттің бір ғана түрі болса, онда ол 1-ішкі есебіндегідей.

Әйтпесе, аралықта кәмпиттердің екі түрі ғана болуы мүмкін. Біз әрбір позиция үшін келесі әртүрлі кәмпиттің ең жақын орнын алдын ала есептей аламыз және оны 2-ішкі есептің шешімін оңтайландыру үшін пайдалана аламыз. Бұл енді O(ans) тілінде орындалады, мұнда ans сұрауға жауап болады. Дегенмен, бұл тым баяу, сондықтан екілік көтеруді сұрауларға жауап беру үшін пайдалануға болады, бұл оларды $O(log(n))$ ішінде шешуге мүмкіндік береді.

4-ішкі есеп

Сұрау аралығының ұзындығы тақ болса, жауап 1 болады, себебі бірегей орын алатын кейбір элемент болады. Әйтпесе, интервалдың бірінші жартысындағы кәмпиттер жиынтығы екінші жартысының жиынтығына тең болған жағдайда ғана жауап 2 болуы мүмкін. Айрықша элементтердің санын санау - $O(n \cdot log(n) + q \cdot log(n))$ ішінде сегменттер ағашын немесе Фенвик ағашын пайдаланып орындауға болатын белгілі мәселе.

5-ішкі есеп

Кәмпитті бірінші досқа бергеннен кейін бізде бірдей мәселе қалғанын ескеріңіз, бірақ жаңа аралықта. Оның үстіне, әр достың кәмпит жинағы интервал жиынтығына тең болғандықтан, әртүрлі кәмпиттердің саны екеуі үшін де бірдей.

Енді i позициясында аяқталатын префикстің жауабы ретінде $prefAns_i$ анықтайық. Ашкөздік стратегиясын қолдана отырып, егер $(j, i]$ интервалындағы бірегей кәмпиттердің саны $[1, i]$-дағымен бірдей болатын ең жақын сол j орнын таба алсақ, $prefAns_i = prefAns_j + 1$, егер $[1, j]$ интервалындағы бірегей кәмпиттердің саны $[1, i]$ бойынша бірдей болса, әйтпесе ол 1-ге тең болады. Әрбір префикс үшін бірегей кәмпиттердің санын $O(n)$ арқылы есептеуге болады, ал j орнын $O(log(n))$ ішінде әрбір кәмпиттің ең оң жақтағы пайда болуын $std :: set$ ішінде сақтау арқылы табуға болады.

6-ішкі есеп

Біз 2-ішкі есебінің бірдей шешімін пайдалана аламыз, бірақ аздап өзгертулер енгіземіз. Аралықтағы бірегей кәмпиттердің саны 100-нан көп болмауы үшін, сұраудағы бірегей кәмпиттердің саны K

болғанда сұрауларға жауап беруге болады. Әрбір i үшін бірегей кәмпиттердің саны K-ға тең болатын келесі ең жақын позицияны алдын ала есептеу жеткілікті, оны екі көрсеткішті пайдаланып $O(n)$ орындауға болады және dfs және екілік орындаңыз. сұрауларға жауап беру үшін іздеу немесе екілік көтеру. Нәтижесінде бұл $O(n \cdot D + q \cdot log(n))$ немесе $(n \cdot D \cdot log(n))$ орындалуына байланысты аяқталады, мұнда D - әртүрлі тәттілердің саны.

7-ішкі есеп

Біз SQRT оңтайландыруын қолданамыз. Сұрау интервалындағы бірегей кәмпиттердің саны $\sqrt{n}$-нан аз деп есептейік, онда $O(n \cdot \sqrt{n} + q \cdot log(n))$ уақытында барлық осы сұрауларға жауап беру үшін 6-ішкі есептің шешімін пайдалана аламыз.

Басқа жағдайға келетін болсақ, жауап $\sqrt{n}$ мәнінен үлкен болмауы керек, себебі әрбір дос кемінде $\sqrt{n}$ элементтері бар интервалды алады.

Біраз R орындап, R позициясында аяқталатын барлық сұрауларға жауап берейік. Ағымдағы R үшін біз $next_i$ массивін сақтаймыз. Ол бірегей элементтердің саны $[i, R]$ интервалындағымен бірдей болатын келесі ең жақын позицияны қамтиды (бұл массив достарымыз алатын кәмпиттердің интервалдары туралы ақпаратты сақтайды). Бұл массивтің көмегімен біз келесі интервалға «секіре» аламыз және жауапты ең көбі $\sqrt{n}$ «секіру» арқылы таба аламыз.

Однако нам нужно обновить этот массив, когда мы увеличиваем R. Для этого нам нужно заметить, что $next_i$ будет изменяться только для позиций после предыдущего вхождения a_{R+1}. Для таких позиций мы должны присвоить $next_i = R+1$. Если мы будем обновлять эти значения и отслеживать уникальные числа с использованием разложения SQRT, мы сможем получить $next_i$ и количество уникальных чисел на интервале $[i, R+1]$ за $O(1)$. В целом мы получаем решение, которое работает за $O(q \cdot \sqrt{n} + n \cdot \sqrt{n})$.

Дегенмен, біз R көбейткен кезде бұл массивді жаңартуымыз керек. Ол үшін $next_i$ a_{R+1} алдыңғы орын алған соң ғана позициялар үшін өзгеретінін ескеруіміз керек. Мұндай позициялар үшін $next_i = R+1$ тағайындауымыз керек. Егер біз осы мәндерді жаңартып, SQRT декомпозициясын пайдаланып бірегей сандарды бақылайтын болсақ, $next_i$ және $O(1)$ ішіндегі $[i, R+1]$ интервалындағы бірегей сандар санын аламыз. Жалпы, $O(q \cdot \sqrt{n} + n \cdot \sqrt{n})$ ішінде жұмыс істейтін шешімді аламыз.

8-ішкі есеп

Соңғы ішкі есепті шешу үшін біз басқа тәсілді қолданамыз. Бөліп ал және жеңу алгоритмі арқылы шешейік. $solve(L, R)$ рекурсивті функциясын анықтаймыз, бұл функция $l_i \leq M$ және $M < r_i$ болатындай i барлық сұрауларының жауабын табады, мұнда $M = [\frac{L+R}{2}]$. $[L, R]$ сол жақ жартысында орналасқан сұраулар үшін $solve(L, M)$ рекурсивті шақырамыз, басқа сұраулар үшін $solve(M+1, R)$ қолданамыз.

Енді $[L, R]$ сол жағындағы сұраудың бөлігі оң жаққа тәуелсіз екенін байқағанымыз жөн. Мұны келесідей алуға болады: $[l_i, M]$ жиыны $[l_i, r_i]$ жиынымен бірдей болса, біздің ашкөз көзқарас сұраудың оң жағына қарамастан «секірулер» жасайтынын ескеріңіз (оң жаққа ұқсас). Сондай-ақ, $[l_i, M]$ жиыны $[l_i, r_i]$ жиынынан кішірек болса, сол жақтағы «секірулерді» елемей, оң жақтағы «секірулерді» қолдануға болады.

Енді $[L, M]$ суффикстері мен $[M+1, R]$ префикстері үшін мәселені дербес шеше аламыз. Бұл есеп 5-ішкі есепке ұқсас. Соңғы бөлім үшін біз соңғы «секіруді» сол жақта және соңғы «секіруді» оң жақта сақтауымыз керек, сонымен қатар «секірулер» саны мен $[l_i, M]$ және $[M+1, r_i]$ бойынша әр түрлі элементтер. Бұл мәндер 5-ішкі есептің шешіміне кішкене өзгертулермен оңай сақталады, бірақ біз оны $std :: set$ орнына "екі көрсеткішті"пайдалану арқылы жылдамырақ жасай аламыз.

Қалған соңғы бөлік - сол жақтағы соңғы «секіру» және оң жақтағы соңғы «секіру». Біз бұл «секірулерді» біріктіруіміз керек, өйткені кейде олар жарамды интервалды құрайды, бұл жағдайда сұрауға жауап ұлғайтылуы керек.

Problem F. Алихан және оқу

Бұл есеп шеттегі нүктеден төбелердің кейбір ішкі жиынына дейінгі k-шы ең үлкен қашықтықтың қосындысын есептеуден және оны сол шеттегі барлық бүтін нүктелер бойынша қосудан тұрады. Осы төбелер жиынын маңызды төбелер деп атаймыз. Сондай-ақ u-дан v-ға дейінгі ең қысқа қашықтықты $d(u, v)$ деп белгілейік. n Дейкстра алгоритмдерін әрбір төбеден $O(n(n + m) \log m)$ уақытында іске қосу арқылы $d(u, v)$ барлық жұптық ең қысқа қашықтықтарды алдын ала есептеу өте оңай, бұл өте оңай. жылдам. мәселенің шектеулерімен.

1-топ ($S = 1$) — 8 ұпай

Бұл сынақ тобында дәл бір төбе маңызды (оны A деп атаймыз), сондықтан $k = 1$ және бізге (u, v, w) қырындағы барлық нүктелерден A маңызды төбесіне дейінгі қашықтықтардың қосындысын есептеу керек.

(u, v, w) қырындағы x нүктесінен A төбесіне дейінгі ең қысқа жол қандай болуы мүмкін екенін қарастырайық. $d(x, u) = \alpha$, содан кейін $d(x, v) = w - \alpha$ деп белгілейік. Әлбетте,

$$\begin{aligned} d(x, A) &= \min(d(x, u) + d(u, A), d(x, v) + d(v, A)) \\ &= \min(\alpha + d(u, A), w - \alpha + d(v, A)). \end{aligned}$$

Егер бұл функцияны координаталық жазықтықта салсақ, онда x осі α мәнін ($[0, w]$ интервалында) және y осі $d(x, A)$ мәні болса, онда 1-ға тең еңісі бар кесіндіден және -1-ға тең келесі кесіндіден тұратын бөліктік сызықтық функцияны алуымыз керек.

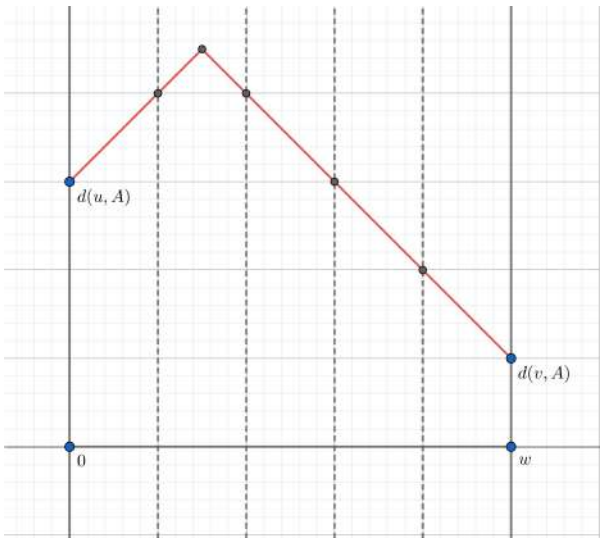


Рис. 1: $f(\alpha) = d(x, A)$ функциясының графигі, мұндағы x - (u, v, w) нүктесінде $d(u, x) = \alpha$ болатын нүкте. Жауап - бұл бөліктік сызықтық функцияның бүтін нүктелердегі мәндерінің қосындысы.

Бұл функцияның шыңы $d(x, u) + d(u, A) = d(x, v) + d(v, A)$ болатындай x нүктесі екенін ескеріңіз, немесе, эквивалентінде ,

$$\alpha = \frac{d(v, A) + w - d(u, A)}{2}.$$

Сондықтан, біз $O(1)$ уақытында осы бөліктік сызықтық функцияның барлық бұрыштық нүктелерін жоғарыдағы формуланы жай ғана есептеу арқылы оңай ала аламыз және тек қажетті қосындыны табу ғана қалады. қашықтықтар. Мұны әрбір сызық сегментін бөлек қарастыру және қосарланған санауды мұқият өңдеу арқылы оңай жасауға болады. Бір сегмент үшін $(L_x, L_y) \to (R_x, R_y)$ дейін біз сол жақ шеткі нүктені жоғары және оң жақ шеткі нүктені дөңгелектей аламыз. Содан кейін, егер сызық өсетін болса (яғни оның еңісі 1), жауап келесідей болады:

$$\sum_{x=L_x}^{R_x} (L_y + x - L_x) = L_y \cdot (R_x - L_x + 1) + \sum_{x=0}^{R_x - L_x} x = L_y \cdot (R_x - L_x + 1) + \frac{(R_x - L_x)(R_x - L_x + 1)}{2}.$$

Дәл осындай формуланы сызық кему кезінде алуға болады.

Қорытындылай келе, әрбір сұрау үшін біз жоғарыда аталған бөліктік сызықтық функцияның шыңын табуымыз керек, оны екі жолдық сегментке бөліп, олардың әрқайсысына жауаптарды санап, оларды қосып, қажет болған жағдайда қосарланған санаудың бір бөлігін алып тастауымыз керек. Бұл $O(n(n+m) \log m+q)$ тілінде жұмыс істейді, өйткені барлық есептеулер $O(1)$ ішінде орындалады.

2-топ $(n, q \leq 100, L = 1)$ — **8 ұпай**

Бұл сынақ тобында n және q шектеулері өте аз және барлық қырлардың ұзындығы дәл 1 құрайды.

Бұл ішкі есеп үшін біз тек екі жағдайды қарастыруымыз керек: $x = u$ және $x = v$. Олардың әрқайсысы үшін біз барлық маңызды төбелерді сәйкес u немесе v төбесіне дейінгі қашықтық бойынша сұрыптап, k-ші ең үлкен қашықтықты алуымыз керек.

Толық асимптотика $O(n(n+m)\log m + qn\log n)$ болады.

3-топ $(n, L \leq 100, q \leq 10^3)$ — **12 ұпай**

Бұл сынақ тобында n шегі әлі де кішкентай, бірақ қырларының ұзындығы енді 100 жетуі мүмкін (бірақ әлі де өте аз) және сұраулар саны аз $q \leq 10^3$.

Мұндай шектеулер барлық мүмкін болатын x нүктелері арқылы қайталанатын және алдыңғы ішкі мәселедегідей тікелей маңызды нүктелерге k-ші қашықтықты табатын баяу шешімге әкеледі. Еске салайық, $d(x, A) = min(\alpha + d(u, A), w - \alpha + d(v, A))$, сондықтан әрбір x нүктесі үшін осыдан барлық қашықтықты есептей аламыз. нүкте. маңызды нүктелерді көрсетіңіз және сол мәндерді анық сұрыптау арқылы kth ең үлкен қашықтықты алыңыз.

Толық асимптотика $O(n(n+m)\log m + qLn\log n)$ болады.

4-топ $(m = n-1, u_i = i, v_i = i+1, k_j = 1)$ — **10 ұпай**

Бұл сынақ тобындағы біздің граф бамбук (өлшеміне ешқандай шектеулер жоқ) $1 \to 2 \to \ldots \to n$ және $k_j = 1$, яғни бізді x-тен маңызды төбелердің ішінен ең алыс төбе ғана қызықтырады.

$(i, i+1, w)$ қырын қарастырып, оған x нүктесін бекітейік. x нүктесінен ең алыс маңызды төбені табу үшін $1, 2, \ldots i$ арасындағы ең сол жақ ең маңызды төбені және $i+1, i+2, \ldots, n$ арасындағы ең оң жақтағы маңызды төбені қарастыруымыз керек. Олар x таңдауына тәуелді болмағандықтан, біз оларды барлық маңызды төбелерді жиында сақтау және сол жиынтықтан минимум және максимум мәндерді алу (немесе тіпті 1-дан $-iii$).n, өйткені жалпы шектеулер рұқсат етілген). Осы ең сол және оң жақ ең маңызды l және r төбелерін тауып, біз қайтадан бөліктік сызықтық функцияны аламыз:

$$f(\alpha) = \max(d(x,i) + d(i,l), d(x,i+1) + d(i+1,r)) \\ = \max(\alpha + d(i,l), w - \alpha + d(i+1,)).$$

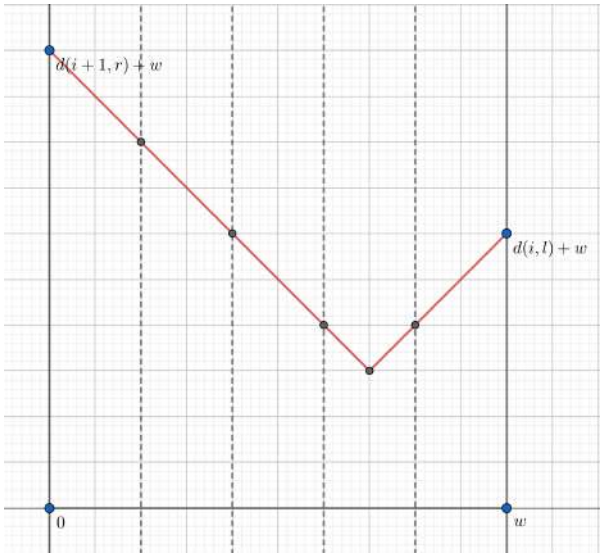


Рис. 2: $f(\alpha) = d(x, A)$ функциясының графигі, мұндағы x — $(i, i+1, w)$ нүктесінде $d(i, x) = \alpha$ болатындай, l және r сәйкесінше ең сол және оң жақ маңызды төбелер. Жауап - бұл бөліктік сызықтық функцияның бүтін нүктелердегі мәндерінің қосындысы.

Еске салайық, дәл осындай құрылыс бірінші қосалқы тапсырмада пайда болды және ортаңғы бұрыш нүктесін табуды қоспағанда, дәл осылай шешіледі.

Бұл сынақ тобы үшін толық асимптотика $O(n(n+m)\log m + q\log n)$ болады.

5-топ $(m = n-1, k_j = 1)$ — **15 ұпай**

Бұл сынақ тобында біздің граф ағаш (оның өлшеміне ешқандай шектеулер жоқ) және $k_j = 1$ болып табылады, бұл маңызды төбелердің ішінде

бізді x-тен ең алыс төбе ғана қызықтыратынын білдіреді.

(u, v, w) қырын қарастырып, оған x нүктесін бекітейік. x нүктесінен ең алыс маңызды төбені табу үшін біз u ішкі ағашындағы u нүктесінен ең алыс маңызды төбені және v ішкі ағашындағы v нүктесінен ең алыс маңызды төбені ғана қарастыруымыз керек, өйткені ең алыс жол x u немесе v арқылы ішкі ағаштың сәйкес ең алыс төбесінде өтеді. Мұны істеу үшін біз A барлық маңызды төбелерін қайталап, олардың екі ішкі төбесінің қайсысында екенін тексеруге болады (егер $d(u, A) + w = d(v, A)$, A u ішкі ағашында жатыр және керісінше) және сол ішкі ағаштың ең алыс төбесін сәйкесінше жаңартыңыз. Осы екі төбені тапқаннан кейін шешім алдыңғы 4 ішкі есептегідей орындалады.

Толық асимптотика $O(n(n + m) \log m + qn)$ болады.

6-топ $(k_j = 1)$ — 24 ұпай

Бұл сынақ тобында әрбір x нүктесі үшін біз ең алыс маңызды төбені ғана қарастыруымыз керек.

(u, v, w) қабырғасын және ондағы x нүктесін қарастырайық. 3 суретінде x мәнін u-дан v-ға жылжытқанда x нүктесінен A маңызды төбесіне дейінгі қашықтық қандай болатынын еске түсірейік. Бұл 1 көлбеу сызықтан басталып, -1 көлбеу сызықпен аяқталатын бөліктік сызықтық функция. Барлық маңызды төбелерді $A_1, \ldots, A_l$ деп белгілейік. Егер біз әрбір A_i үшін осындай график сызатын болсақ, бізге тек $d(x, A_i)$ қашықтықтарының жоғарғы шегіндегі нүктелерді қарастыру керек еді.

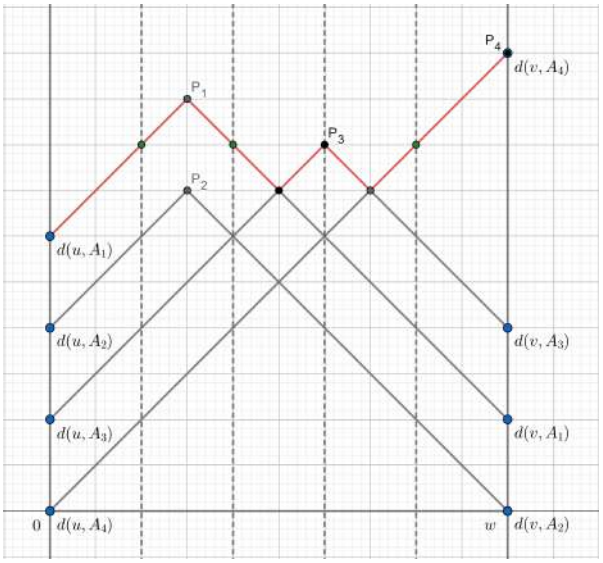


Рис. 3: $A_1, \ldots, A_l$ барлық маңызды төбелері үшін $d(x, A_i)$ қашықтықтарының графигі. Жоғарғы шекара қызыл түспен белгіленген. Жауап - бүтін нүктелердегі бөліктік сызықтық функцияның жоғарғы шекарасының мәндерінің қосындысы.

$d(u, A_i) \geq d(u, A_j)$ және $d(v, A_i) \geq d(v, A_j)$ болатын A_i және A_j екі төбесі болса, ескеріңіз. онда $d(x, A_i)$ кестесі әрқашан $d(x, A_j)$ кестесінен асып түседі, сондықтан A_j елемеуі мүмкін. Маңызды нүктелерді $d(u, A_1) > \ldots > d(u, A_l)$ болатындай реттейік. Содан кейін жоғарыдағы ескерту арқылы $d(v, A_1) < \ldots < d(v, A_l)$ аламыз. Осы қалған төбелердің әрқайсысы нәтиженің жоғарғы шегіне белгілі бір соманы қосатынын көру оңай. Бұл жағдайда олардың бөліктері дәл осындай $(A_1, \ldots, A_l)$ ретімен реттеледі.

Осы төбелерді және олардың сәйкес ретін алу үшін алдымен әрбір v төбесі үшін олардың қашықтығы бойынша v дейін кему ретімен сұрыпталған барлық төбелердің тізімін есептеу керек. Мұны $O(n^2 \log n)$ уақытында жасауға болады. Егер бізде мұндай тізім болса, онда барлық ең жоғары ұпайларды алу үшін біз u төбесі үшін алдын ала есептелген тізім арқылы қайталап, дөңес корпусты іздеу алгоритмінде орындалғандай төбелер дестесін сақтай аламыз. жазықтықтағы нүктелер.

Жоғарғы шекараның барлық бұрыштық нүктелерін алу үшін біз әрбір A_i үшін ең жоғары нүктелерді және A_i және A_{i+1} сызбаларының қиылысуын есептеуіміз керек, ол $O(l) = O(n)$-да толығымен орындалады.

Ақырында, мақсат функциясының жоғарғы шекарасының барлық бұрыштық нүктелерін есептеп, біз 1 ішкі мәселесіндегідей әрекет жасай аламыз (әрбір жол бөлімшесінде жауапты дербес есептеп, қосарланған санауды өңдеңіз).

Жалпы асимптотика $O(n(n + m) \log m + n^2 \log n + qn)$ болады.

7-топ (қосымша шектеусіз) — 23 ұпай

Бұл сынақ тобында ешқандай қосымша шектеулер көрсетілмеген, сондықтан біз барлық мәселені шешуіміз керек.

(u, v, w) қырын және ондағы x нүктесін қарастырайық. $(A_1, \ldots, A_l)$ маңызды төбелер делік. Алдыңғы ішкі есепте жасағандай, олардың әрқайсысы үшін графиктер салайық, бірақ бұл жағдайда төменгі төбелерді алып тастай алмаймыз, өйткені олар әлі де жауапқа үлес қоса алады.

x мәнін u-дан v-ға жылжытатын болсақ, k ең үлкен мәні бөліктік сызықтық функция ретінде үздіксіз қозғалатынын ескеріңіз. Мұны кейбір графиктердің қиылысу нүктелерінде және осы графиктердің төбелерінде бұл мәннің қалай өзгеретінін қарастыру арқылы тексеруге болады.

Бір қарағанда, бұл бөліктік сызықтық функцияны тікелей u-дан v-ға дейін анықтау өте қиын болуы мүмкін. Жеңілдету үшін біз $[0, w]$ аралығын әрбір A_i үшін төбелердің x-координаталары арқылы бөлеміз және x үшін алынған ішкі интервалдарды сыпырып аламыз.

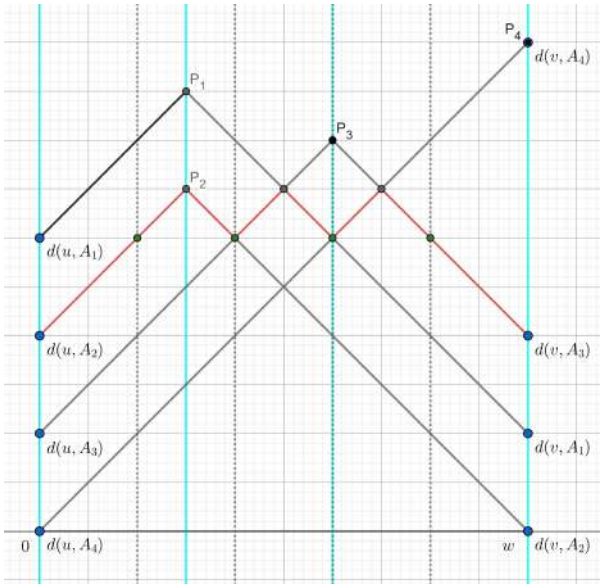


Рис. 4: $A_1, \ldots, A_l$ барлық маңызды төбелері үшін $d(x, A_i)$ қашықтықтарының графигі. $k = 2$ үшін k-ші қашықтық функциясы қызыл түспен, сканерлеу сызығын бөлу оқиғалары көк түсте. Жауап - бүтін нүктелердегі қызыл бөліктік сызықтық функция мәндерінің қосындысы.

Сканерлеу сызығының бір ішкі интервалында сызбалар қалай көрінетінін көрейік. Бізде 1 көлбеу t сызықтары және t үшін -1 көлбеу $n - t$ сызықтары болуы керек. Әлбетте, бізге ең көбі 1 ең жоғары k және ең көбі -1 ең жоғары k ең жоғары сызықтар қажет, өйткені төменгі сызықтар ешқашан kth максимумға ықпал етпейді. мән. $O(n \log n)$ уақытында сызықты сканерлеу кезінде 1 көлбеуі бар сызықтар жинағын және -1 еңісі бар сызықтар жинағын сақтап, k ең жоғары сызықтарды алу арқылы осы сызықтарды аламыз. әрбір жиынтық. әрбір ішкі интервал үшін $O(k + \log n)$ уақытында (жиынды соңынан іздеу). Бұл бізге кему ретімен сұрыпталған ең көбі 1 ең көп k сызықтарының векторын және -1 еңісі бар k сызықтарының векторын береді.

Енді осы ішкі интервалда k-ші максимум функциясын $O(k)$ уақытында толығымен құруға болатынын көрсетеміз.

Алдымен біз оның ең сол жақ нүктесін табуымыз керек. Бұл 1 және -1 көлбеу сызықтарының сол жақ сканерлеу сызығының бөлім сызығымен қиылысулары арасындағы k-th ең үлкен мән болып шығады. Оны $O(k)$ уақытында біріктіру сұрыптауы арқылы табуға болады, өйткені біз бұл жолдарды кему ретімен векторларда сақтаймыз. Кейінірек шешімде біз осы векторлардың 0 негізіндегі индекстеуін қолданамыз. kth максимумы төмендейтін jth жолынан басталады деп есептейік ($j = 1$ болатын екінші ішкі интервалдағы **??** суретін қараңыз). Ол қиып өтетін бірінші өсу сызығы $i = k - j - 1$ өсу сызығы болатынын көру оңай. Сондай-ақ, x солдан оңға қарай жылжыған кезде k-th максимум жолдардың реті i өсетін және j кеметін өсу және кему сызықтары арасында ауысатынын оңай көруге болады. Біздің бастапқы сызығымыз көтерілсе де солай айтуға болады. Осылайша, бізде қазіргі уақытта ең көбі $2k$ сызықтары бар фактімен біріктірілгенде, осы ішкі интервал үшін k-ші максималды мәнінің бөлшек сызықтық функциясының барлық бұрыштық нүктелерін $O(k)$ уақытында салуға болады.

Әрбір ішкі интервалдағы k-ші максимум функциясының нақты формасын тапқаннан кейін, біз оларды жай ғана біріктіріп, функцияны бүкіл $[0, w]$ интервалында аламыз. Осыдан кейін алдыңғы ішкі мәселедегідей әрекетті орындау арқылы біз соңғы жауапты есептей аламыз. Сонымен қатар, әрбір жол kth максималды функцияға бір рет үлес қоса алатындықтан, бұл функцияда ең көбі $2n$

сегменттері болады (бірақ бұл біздің шешіміміз үшін маңызды емес).

Толық асимптотика $O(n(n+m)\log m + qn(k+\log n))$ болады.

Үлгі шешім

Авторлық үлгілік шешімді келесі сілтемеден табуға болады:

https://gist.github.com/dxtvzw/3d6604238d1f267cef28b88ce9f93849

Задача D. Арбузы

Давайте для удобства сразу для всех подзадач отсечем случай, когда $n + m < K$, так как здесь ответ -1, потому что каждый друг должен получить хотя бы по одному арбузу.

Подзадача 1 ($m = 0$, $a_i = a_{i+1}$ $(1 \leqslant i < n)$) — 9 баллов

Так как все a_i равны, давайте определим число $x = a[1]$. В данной подзадаче достаточно равномерно распределить все n арбузов по K друзьям, тогда раздача будет справедливой, потому что разница арбузов между любыми двумя друзьями не более x, а сумма арбузов остальных друзей не менее $2 \cdot x$, из-за $n \geqslant 3$. Это можно легко сделать к примеру так: сначала раздадим всем друзьям по $\lfloor \frac{n}{K} \rfloor$ арбузов, а затем раздать друзьям с номерами от 1 до $n \ mod \ K$ по одному арбузу.

Подзадача 2 ($a_i = a_{i+1}$ $(1 \leqslant i < n)$, $b_j = b_{j+1}$ $(1 \leqslant j < m)$) — 19 баллов

Эту подзадачу можно разбить на три случая:

1. Если $K \leqslant n$, то решение точно такое же, как и в подзадаче 1.
2. Если $K = n + 1$, то раздаем первым n друзьям по одному арбузу a_1, а $(n + 1)$-у другу b_1. Если оказалось, что $b_1 > n \cdot a_1$, тогда передаем 1-у другу еще один арбуз b_1 (если в этом случае $m = 1$, то ответ -1). В обеих случаях раздача будет справедливой.
3. Если $K \geqslant n+2$, то раздаем первым n друзьям по одному арбузу a_1, а остальным $K-n$ друзьям по одному арбузу b_1. Тогда раздача будет справедливой, потому что есть хотя бы 2 друга с a_1 и хотя бы 2 друга с b_1.

Подзадача 3 (Необязательно выводить раздачу арбузов) — 24 баллов

Решение данной подзадачи делает весомый вклад в решение полной задачи. Пусть мы купили у продавца v некоторых арбузов, и получили новый массив арбузов c из $n + v$ арбузов. Теперь хотим узнать, можно ли сделать справедливую раздачу текущим набором арбузов.

Утверждение: Если $2 \cdot maxC \leqslant \sum_{i=1}^{n+v} c_i$, то ответ есть ($maxC$ – максимальное число в массиве c). Назовем данное условие *генеральной*.

Доказательство: Если $2 \cdot maxC > \sum_{i=1}^{n+v} c_i$, то ответа нет, потому что $maxC > \sum_{i=1}^{n+v} c_i - maxC$ противоречит справедливой раздаче. Иначе, мы покажем, что достаточно иметь 3 друзей для справедливой раздачи, а если друзей больше, то это ничего не может испортить. Пусть у нас 3 друга. Определим S_i – сумма весов арбузов i-го друга. Давайте сначала отсортируем арбузы в порядке неубывания. Отдадим арбуз c_{n+v} (то есть один арбуз с максимальным весом) первому другу. Теперь начнем раздавать арбузы с номерами от 1 по $n+v-1$ по одному слева направо двум оставшимся друзьям по такому алгоритму: i-й арбуз отдадим 2-у другу, если в данный момент $S_2 \leqslant S_3$, иначе, отдадим 3-у другу. После такой раздачи будет верно, что $|S_2 - S_3| \leqslant maxC$, потому что когда мы отдаем i-й арбуз 2-у другу, то перед передачой было $S_2 \leqslant S_3$, а после передачи не может быть $S_2 + c_i > S_3 + maxC$, потому что $c_i \leqslant maxC$ (и наоборот, если мы отдаем 3-у другу). В конце выйдет так, что $S_1 = maxC$, а $|S_2 - S_3| \leqslant maxC$. Теперь осталось лишь доказать, что $S_1 \leqslant S_2 + S_3$, $S_2 \leqslant S_1 + S_3$ и $S_3 \leqslant S_1 + S_2$. По изначальному условию мы знаем, что $maxC \leqslant \sum_{i=1}^{n+v} c_i - maxC$, поэтому $S_1 \leqslant S_2 + S_3$. Так как случаи для S_2 и S_3 аналогичны, давайте считать, что $S_3 \geqslant S_2$, и докажем правильность для S_3. Так как $|S_3 - S_2| \leqslant maxC$, то $S_3 \leqslant S_2 + maxC$, что и доказывает справедливую раздачу. Остальным $K - 3$ друзьям можно отдавать по одному арбузу из S_2 или S_3 так, чтобы S_2 и S_3 не остались пустыми. Очевидно, что если S_2 и S_3 удовлетворяют условию справедливой раздачи, то один арбуз из S_2 или один арбуз из S_3 не могут не удовлетворять условию.

Осталось найти минимальное количество элементов из массива b, получив которых, у нас выйдет массив c, который удовлетворяет генеральному условию. Ответ 0, если массив a и так удовлетворяет генеральному условию, и $n \geqslant K$. Иначе, давайте отсортируем массив b по неубыванию, и будем перебирать максимальный b_i, которого хотим взять. Очевидно, что самый оптимальный следующий

элемент массива b, которого мы возьмем – b_{i-1}, потому что если элемент не меняет максимум, то для генерального условия лучше, чтобы сумма стала как можно больше. Таким образом, если мы зафиксировали максимальный b_i, то будем идти справа налево пока генеральное условие не выполняется, или у нас меньше арбузов, чем K. Выходит, мы найдем для i максимальный такой j, что если купим арбузы $b_j, b_{j+1}, \ldots, b_i$, то можно будет сделать справедливую раздачу. Несложно понять, что j можно найти *бинарным поиском* или *методом двух указателей*, потому что если j подходит, то $j-1$ тоже подойдет. В конце из всех подходящих $i-j+1$ выведем минимальный (если ни один не подходит, то -1).

Подзадача 4 ($m = 0$) — 20 баллов

Из подзадачи 3 мы знаем генеральное условие, которое определяет, существует ответ для конкретного набора арбузов или нет. Если ответ существует, то его можно восстановить точно так же, как написано в Доказательстве генерального условия в подзадаче 3. То есть, сначала заполняем S_1, S_2, S_3 для первых трех друзей, а затем из S_2 и S_3 раздаем по одному арбузу оставшимся друзья.

Подзадача 5 ($b_i = 1$ $(1 \leqslant i \leqslant m)$) — 10 баллов

Так как все элементы массива b равны, нам важно только количество арбузов, которых мы купим. Достаточно перебрать сколько арбузов мы купим, проверить генеральное условие из подзадачи 3 и восстановить ответ, как в подзадаче 4.

Подзадача 6 (Нет дополнительных ограничений) — 18 баллов

Из подзадачи 3 мы знаем сколько арбузов нужно купить и подотрезок арбузов, которых нужно купить в отсортированном массиве b. Из подзадачи 4 мы знаем как восстановливать ответ, если у нас есть конкретный набор арбузов. Объединив эти решения, можно получить полное решение.

Задача E. Конфеты

Подзадача 1

Поскольку все a_i равны 1, ответом для каждого запроса будет $r_i - l_i + 1$, что является длиной интервала.

Подзадача 2

В связи с тем, что каждый друг получает тот же набор конфет, что и другие друзья, набор каждого друга равен набору интервала. Кроме того, чтобы обеспечить, чтобы после того, как конфеты были отданы первому другу, другие друзья могли получить тот же тип конфет, набор конфет на префиксе интервала должен быть равен набору элементов после этого префикса.

Таким образом, можно сделать вывод, что жадно выбирать первый префикс конфет, который удовлетворяет вышеуказанному требованию, является допустимой стратегией. Давайте назовем процесс выбора необходимого префикса "прыжком". Теперь эту идею можно реализовать для этой подзадачи.

Подзадача 3

Для каждого запроса может быть два решения в зависимости от количества различных конфет. Если есть только один тип конфет, то это то же самое, что и в подзадаче 1.

В противном случае на интервале может быть только два типа конфет. Можно предварительно вычислить ближайшую позицию следующей различной конфеты для каждой позиции и использовать это для оптимизации решения подзадачи 2. Теперь это будет работать за O(ans), где ans - ответ на запрос. Однако это слишком медленно, поэтому для ответа на запросы можно использовать бинарный подъем, что позволит решать их за $O(log(n))$.

Подзадача 4

Если длина интервала запроса нечетна, ответ равен 1, потому что будет какой-то элемент, который встречается уникально. В противном случае ответ может быть 2 только в том случае, если набор конфет на первой половине интервала равен набору второй половины. Подсчет количества различных элементов - это известная задача, которую можно выполнить за $O(n \cdot log(n) + q \cdot log(n)))$ с использованием дерева отрезков или дерева Фенвика.

Подзадача 5

Заметим, что после того, как конфеты были отданы первому другу, у нас остается та же задача, но для нового интервала. Кроме того, поскольку набор конфет каждого друга равен набору интервала, количество различных конфет одинаково для обоих.

Теперь определим $prefAns_i$ как ответ для префикса, заканчивающегося в позиции i. Используя жадную стратегию, если мы можем найти ближайшую левую позицию j, где количество уникальных конфет на интервале $(j, i]$ такое же, как на $[1, i]$, то $prefAns_i = prefAns_j + 1$, если количество уникальных конфет на интервале $[1, j]$ такое же, как на $[1, i]$. В противном случае он будет равен 1. Количество уникальных конфет для каждого префикса можно вычислить за $O(n)$, а позицию j можно найти за $O(log(n))$, храня самое правое вхождение каждой конфеты в $std :: set$.

Подзадача 6

Мы можем использовать то же решение из подзадачи 2, но сделать небольшую модификацию. Поскольку количество уникальных конфет на интервале не может быть больше 100, можно отвечать на запросы, когда количество уникальных конфет в запросе равно K, независимо. Достаточно предварительно вычислить для каждого i следующую ближайшую позицию, где количество уникальных конфет будет равно K, что можно сделать за $O(n)$ с использованием двух указателей, и выполнить

dfs и бинарный поиск или бинарный подъем для ответа на запросы. В результате это будет выполнено за $O(n \cdot D + q \cdot log(n))$, или $(n \cdot D \cdot log(n))$ в зависимости от реализации, где D - количество различных конфет.

Подзадача 7

Мы собираемся использовать оптимизацию SQRT. Предположим, что количество уникальных конфет на интервале запроса меньше, чем $\sqrt{n}$, тогда мы можем использовать решение из подзадачи 6 для ответа на все эти запросы за $O(n \cdot \sqrt{n} + q \cdot log(n))$.

Что касается другого случая, ответ не может быть больше $\sqrt{n}$, потому что каждый друг получает интервал с по крайней мере $\sqrt{n}$ элементов.

Давайте зафиксируем некоторое R, и ответим на все запросы, которые заканчиваются в позиции R. Для текущего R мы будем поддерживать массив $next_i$. Он будет содержать следующую ближайшую позицию, где количество уникальных элементов такое же, как на интервале $[i, R]$ (этот массив хранит информацию о интервалах конфет, которые получают наши друзья). Используя этот массив, мы можем "прыгнуть"к следующему интервалу и найти ответ не более чем за $\sqrt{n}$ "прыжков".

Однако нам нужно обновить этот массив, когда мы увеличиваем R. Для этого нам нужно заметить, что $next_i$ будет изменяться только для позиций после предыдущего вхождения a_{R+1}. Для таких позиций мы должны присвоить $next_i = R + 1$. Если мы будем обновлять эти значения и отслеживать уникальные числа с использованием разложения SQRT, мы сможем получить $next_i$ и количество уникальных чисел на интервале $[i, R+1]$ за $O(1)$. В целом мы получаем решение, которое работает за $O(q \cdot \sqrt{n} + n \cdot \sqrt{n})$.

Подзадача 8

Для решения последней подзадачи мы будем использовать другой подход. Давайте решим ее с помощью алгоритма "Разделяй и властвуй". Мы определяем рекурсивную функцию $solve(L, R)$, эта функция находит ответ для всех запросов i, таких что $l_i \leqslant M$ и $M < r_i$, где $M = [\frac{L+R}{2}]$. Для запросов, которые находятся в левой половине $[L, R]$, мы будем рекурсивно вызывать $solve(L, M)$, для других запросов мы будем использовать $solve(M + 1, R)$.

Теперь нам нужно заметить, что часть запроса, которая находится в левой части $[L, R]$, независима от правой части. Это можно получить следующим образом: заметим, что если набор $[l_i, M]$ такой же, как набор $[l_i, r_i]$, наш жадный подход сделает "прыжки"независимо от правой части запроса (аналогично для правой части). Кроме того, если набор $[l_i, M]$ меньше, чем набор $[l_i, r_i]$, мы можем игнорировать "прыжки"с левой стороны и использовать "прыжки"с правой стороны.

Теперь мы можем решить задачу независимо для суффиксов $[L, M]$ и префиксов $[M + 1, R]$. Эта задача аналогична подзадаче 5. Для последней детали нам нужно хранить последний "прыжок"с левой стороны и последний "прыжок"с правой стороны, а также количество "прыжков"и количество различных элементов на $[l_i, M]$ и $[M + 1, r_i]$. Эти значения легко хранятся с помощью небольших изменений в решении подзадачи 5, однако мы можем сделать это немного быстрее, используя "два указателя"вместо $std :: set$.

Последняя деталь, которая остается, это последний "прыжок"с левой стороны и последний "прыжок"с правой стороны. Нам нужно объединить эти "прыжки потому что иногда они образуют допустимый интервал, в этом случае ответ на запрос должен быть увеличен.

Задача F. Алихан и учеба

Эта задача заключается в вычислении суммы k-го наибольшего расстояния от точки на ребре до некоторого подмножества вершин и суммировании ее по всем целочисленным точкам на этом ребре. Назовем это подмножество вершин важными вершинами. Обозначим также кратчайшее расстояние от u до v через $d(u, v)$. Предварительно вычислить все попарные кратчайшие расстояния $d(u, v)$ довольно просто, запустив алгоритмы n Дейкстры из каждой вершины за $O(n(n+m)\log m)$ время, что достаточно быстро. с ограничениями проблемы.

Группа 1 ($S = 1$) — 8 баллов

В этой тестовой группе важна ровно одна вершина (назовем ее A), поэтому $k = 1$ и нам нужно вычислить сумму расстояний от всех точек ребра (u, v, w) до этой важной вершины A.

Давайте разберемся, как может выглядеть кратчайший путь от точки x на ребре (u, v, w) до вершины A. Обозначим $d(x, u) = \alpha$, тогда $d(x, v) = w - \alpha$. Очевидно,

$$\begin{aligned} d(x, A) &= \min(d(x, u) + d(u, A), d(x, v) + d(v, A)) \\ &= \min(\alpha + d(u, A), w - \alpha + d(v, A)). \end{aligned}$$

Если мы нарисуем эту функцию на координатной плоскости, где ось x представляет собой значение α (на интервале $[0, w]$), а ось y представляет собой значение $d(x, A)$, то мы должны получить кусочно-линейную функцию, состоящую из отрезка с наклоном, равным 1, и следующего за ним отрезка с наклоном, равным -1.

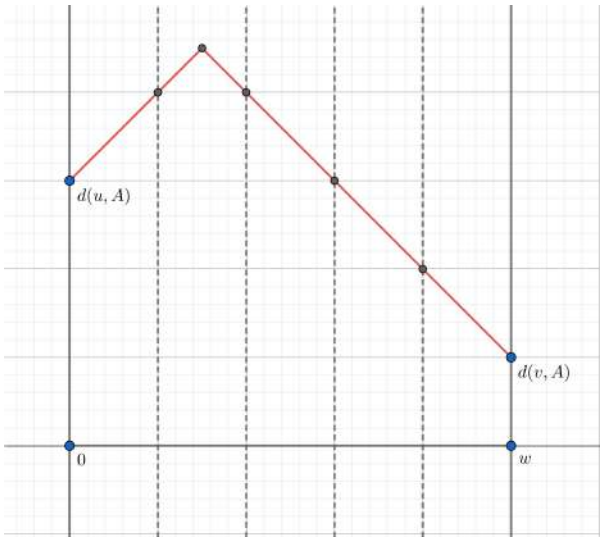


Рис. 1: График функции $f(\alpha) = d(x, A)$, где x — точка на (u, v, w) такая, что $d(u, x) = \alpha$. Ответом является сумма значений этой кусочно-линейной функции по целым точкам.

Заметим, что пиком этой функции является точка x, такая что $d(x, u) + d(u, A) = d(x, v) + d(v, A)$, или, что то же самое,

$$\alpha = \frac{d(v, A) + w - d(u, A)}{2}.$$

Следовательно, мы можем легко получить все угловые точки этой кусочно-линейной функции за время $O(1)$, просто вычислив приведенную выше формулу, и единственное, что осталось сделать, это найти искомую сумму расстояний. Это можно легко сделать, рассматривая каждый сегмент прямой отдельно и тщательно обрабатывая двойной счет. Для одного отрезка $(L_x, L_y) \to (R_x, R_y)$ мы можем округлить левую конечную точку вверх и правую конечную точку вниз. Тогда, если линия восходящая (т.е. ее наклон равен 1), ответ будет выглядеть так:

$$\sum_{x=L_x}^{R_x} (L_y + x - L_x) = L_y \cdot (R_x - L_x + 1) + \sum_{x=0}^{R_x - L_x} x = L_y \cdot (R_x - L_x + 1) + \frac{(R_x - L_x)(R_x - L_x + 1)}{2}.$$

Ту же формулу можно получить, если линия нисходящая.

Подводя итог, для каждого запроса нам нужно найти пик упомянутой выше кусочно-линейной функции, разделить ее на два отрезка прямой, посчитать ответы на каждом из них, сложить их и при необходимости вычесть часть двойного счета. Это работает в $O(n(n+m)\log m + q)$, поскольку все вычисления выполняются в $O(1)$.

Группа 2 ($n, q \leqslant 100, L = 1$) — 8 баллов

В этой тестовой группе ограничения на n и q довольно малы, а длина всех ребер равна ровно 1.

Для этой подзадачи нам нужно рассмотреть всего два случая: $x = u$ и $x = v$. Для каждой из них нам нужно отсортировать все важные вершины по расстоянию до соответствующей вершины u или v и получить k-е наибольшее расстояние.

Полная асимптотика будет $O(n(n+m)\log m + qn\log n)$.

Группа 3 $(n, L \leqslant 100, q \leqslant 10^3)$ — 12 баллов

В этой тестовой группе ограничение на n по-прежнему невелико, но длина ребер теперь может достигать 100 (но все равно достаточно мала), а количество запросов невелико $q \leqslant 10^3$.

Такие ограничения приводят к медленному решению, которое перебирает все возможные x точки и находит k-е расстояние до важных точек напрямую, как и в предыдущей подзадаче. Напомним, что $d(x, A) = min(\alpha + d(u, A), w - \alpha + d(v, A))$, поэтому для каждой точки x мы можем вычислить все расстояния от этой точки. укажите на важные точки и получите k-е наибольшее расстояние, явно отсортировав эти значения.

Полная асимптотика будет $O(n(n+m)\log m + qLn\log n)$.

Группа 4 $(m = n-1, u_i = i, v_i = i+1, k_j = 1)$ — 10 баллов

В этой тестовой группе наш граф представляет собой бамбук (без ограничений на его размер) $1 \to 2 \to \ldots \to n$ и $k_j = 1$, что означает, что нас интересует только самая дальняя вершина от x среди важных вершин.

Рассмотрим ребро $(i, i+1, w)$ и зафиксируем на нем точку x. Чтобы найти самую дальнюю важную вершину от точки x, нам, очевидно, нужно рассматривать только самую левую важную вершину среди $1, 2, \ldots i$ и самую правую важную вершину среди $i+1, i+2, \ldots, n$. Поскольку они не зависят от выбора x, мы можем просто найти их, сохранив все важные вершины в наборе и получить минимальное и максимальное значение из этого набора (или даже просто перебрав все вершины от 1 до n, поскольку общие ограничения это позволяют). Найдя эти самые левую и самую правую важные вершины l и r, мы снова получаем кусочно-линейную функцию:

$$f(\alpha) = \max(d(x,i) + d(i,l), d(x,i+1) + d(i+1,r)) \\ = \max(\alpha + d(i,l), w - \alpha + d(i+1,)).$$

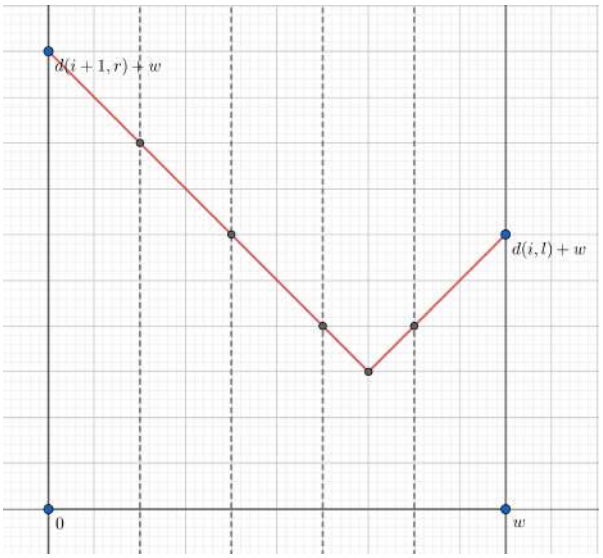


Рис. 2: График функции $f(\alpha) = d(x, A)$, где x — точка на $(i, i+1, w)$ такая, что $d(i, x) = \alpha$, l и r — самая левая и самая правая важные вершины соответственно. Ответом является сумма значений этой кусочно-линейной функции по целым точкам.

Напомним, что почти такая же конструкция появилась в первой подзадаче и решается точно так же, за исключением нахождения средней угловой точки.

Полная асимптотика для этой тестовой группы будет $O(n(n+m)\log m + q\log n)$.

Группа 5 $(m = n-1, k_j = 1)$ — 15 баллов

В этой тестовой группе наш граф представляет собой дерево (без ограничений на его размер) и $k_j = 1$, а это значит, что нас интересует среди важных вершин только самая дальняя от x вершина.

Рассмотрим ребро (u, v, w) и зафиксируем на нем точку x. Чтобы найти самую дальнюю важную вершину от x, нам, очевидно, нужно рассматривать только самую дальнюю важную вершину от u в поддереве u и самую дальнюю важную вершину от v в поддереве v , поскольку самый дальний путь

от x проходит либо через u, либо через v до соответствующей самой дальней вершины поддерева. Для этого мы можем перебрать все важные вершины A и проверить, в каком из двух поддеревьев они находятся (если $d(u, A) + w = d(v, A)$, A лежит в поддерево u и наоборот) и соответствующим образом обновить самую дальнюю вершину в этом поддереве. После нахождения этих двух вершин решение происходит точно так же, как и в предыдущей подзадаче 4.

Полная асимптотика будет $O(n(n+m)\log m + qn)$.

Группа 6 ($k_j = 1$) — 24 балла

В этой тестовой группе для каждой точки x нам нужно рассмотреть только самую дальнюю важную вершину.

Рассмотрим ребро (u, v, w) и точку x на нем. Вспомним на рисунке 3, как выглядит расстояние от точки x до важной вершины A, когда мы перемещаем x из u в v. Это кусочно-линейная функция, которая начинается линией с наклоном 1 и заканчивается линией с наклоном -1. Обозначим через $A_1, \ldots, A_l$ все важные вершины. Если бы мы рисовали такой график для каждого A_i, нам нужно было бы рассматривать только точки на верхней границе всех этих расстояний $d(x, A_i)$.

Заметим, что если существуют две вершины A_i и A_j такие, что $d(u, A_i) \geqslant d(u, A_j)$ и $d(v, A_i) \geqslant d(v, A_j)$, тогда график $d(x, A_i)$ всегда превосходит график $d(x, A_j)$, поэтому A_j можно не учитывать. Расположим важные точки так, чтобы $d(u, A_1) > \ldots > d(u, A_l)$. Тогда по замечанию выше получаем $d(v, A_1) < \ldots < d(v, A_l)$. Легко видеть, что каждая из этих оставшихся вершин будет вносить некоторый вклад в полученную верхнюю границу. При этом их части будут упорядочены точно в одном порядке $(A_1, \ldots, A_l)$.

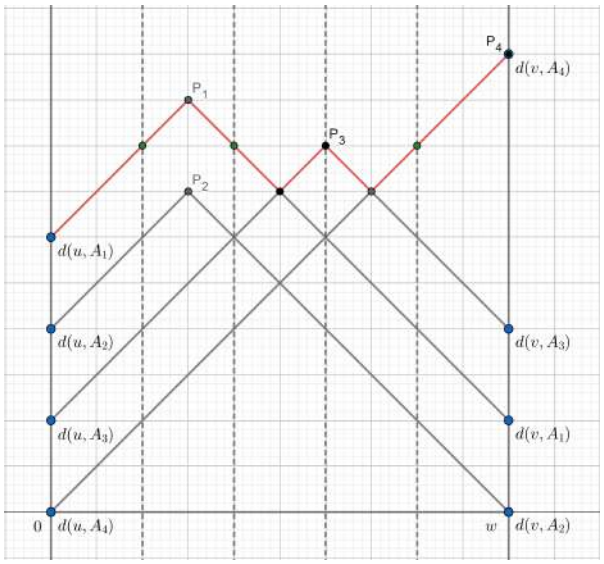


Рис. 3: График расстояний $d(x, A_i)$ для всех важных вершин $A_1, \ldots, A_l$. Верхняя граница выделена красным. Ответом является сумма значений верхней границы кусочно-линейной функции по целым точкам.

Чтобы получить эти вершины и соответствующий им порядок, нам нужно предварительно вычислить для каждой вершины v список всех вершин, отсортированных по их расстоянию до v в порядке убывания. Это можно сделать за $O(n^2 \log n)$ времени. Если у нас есть такой список, то для получения всех старших точек мы можем просто перебирать заранее рассчитанный список для вершины u и поддерживать стек вершин, как это сделано в алгоритме поиска выпуклой оболочки множества вершин. точки на плоскости.

Чтобы получить все угловые точки верхней границы, нам нужно вычислить графики пиков для каждого A_i и пересечения графиков A_i и A_{i+1}, что полностью делается за $O(l) = O(n)$ времени.

Наконец, вычислив все угловые точки верхней границы целевой функции, мы можем сделать то же самое, что и в подзадаче 1 (вычислить ответ независимо на каждом подотрезке линии и обработать двойной счет).

Общая асимптотика будет $O(n(n+m)\log m + n^2 \log n + qn)$.

Группа 7 (без дополнительных ограничений) — 23 балла

В этой тестовой группе не задано никаких дополнительных ограничений, поэтому нам нужно решить задачу целиком.

Рассмотрим ребро (u, v, w) и точку x на нем. Допустим, $(A_1, \ldots, A_l)$ — важные вершины. Нарисуем графы для каждой из них, как мы это делали в предыдущей подзадаче, но в этом случае мы не можем удалять низшие вершины, так как они все равно могут внести свой вклад в ответ.

Обратите внимание: если мы переместим x из u в v, максимальное значение k будет непрерывно перемещаться как кусочно-линейная функция. В этом можно убедиться, рассмотрев, как меняется эта величина в точках пересечения некоторых графиков и в вершинах этих графиков.

На первый взгляд может быть очень сложно проследить эту кусочно-линейную функцию напрямую от u до v. Для упрощения разобьем интервал $[0, w]$, используя x-координаты вершин для каждого A_i, и проведем развертку по полученным подинтервалам для x.

Давайте посмотрим, как выглядят графики на одном подинтервале строки развертки. У нас должны быть линии t с наклоном 1 и линии $n-t$ с наклоном -1 для некоторого t. Очевидно, нам нужно не более k самых высоких линий с наклоном 1 и не более k самых высоких линий с наклоном -1, поскольку нижние линии никогда не будут вносить вклад в k-ое максимальное значение. Мы можем получить эти строки, поддерживая набор линий с наклоном 1 и набор линий с наклоном -1 во время нашей строки сканирования за время $O(n \log n)$ и получая k самых высоких строк из каждого набора. за $O(k+\log n)$ время для каждого подинтервала (перебор множества с его конца). Таким образом, мы получаем вектор из не более k линий с наклоном 1 и вектор из не более чем k линий с наклоном -1, причем оба они отсортированы в порядке убывания.

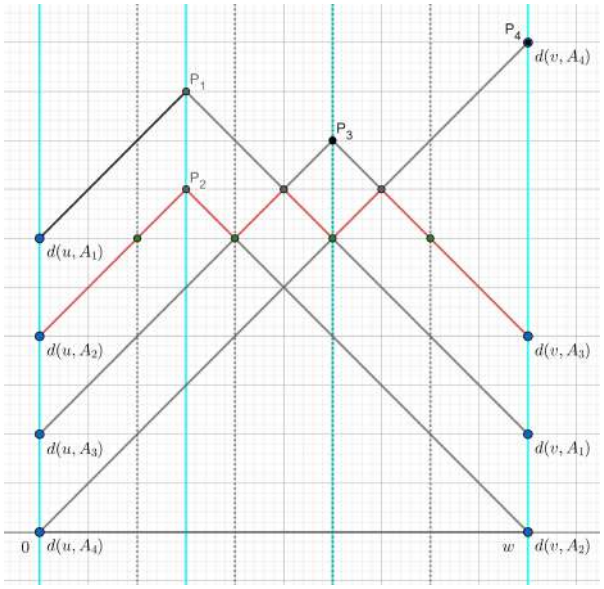


Рис. 4: График расстояний $d(x, A_i)$ для всех важных вершин $A_1, \ldots, A_l$. k-я функция расстояния для $k = 2$ выделена красным, события разбиения строки развертки — синим. Ответом является сумма значений красной кусочно-линейной функции по целым точкам.

Теперь мы покажем, что можно полностью построить k-ю функцию максимума на этом подинтервале за $O(k)$ время.

Сначала нам нужно найти его самую левую точку. Это оказывается k-ое максимальное значение среди пересечений линий с наклоном 1 и -1 с левой линией раздела строки развертки. Его можно найти за время $O(k)$ с помощью сортировки слиянием, поскольку мы храним эти строки в векторах в порядке убывания. Далее в решении мы будем использовать индексацию этих векторов на основе 0. Предположим, что k-й максимум начинается на нисходящей j-й линии (см. рисунок 3 на втором подинтервале с $j = 1$). Легко видеть, что первая восходящая линия, которую он пересечет, будет $i = k - j - 1$ восходящей линией. Также легко заметить, что порядок линий k-го максимума при движении x слева направо будет чередоваться между восходящими и нисходящими линиями с увеличением i и уменьшением j. То же самое можно сказать, если наша стартовая линия идет вверх. Таким образом, в сочетании с тем, что на данный момент у нас имеется не более $2k$ линий, все угловые точки кусочно-линейной функции k-го максимального значения для этого подинтервала могут быть построены за $O(k)$ время.

Найдя точный вид k-й функции максимума на каждом подинтервале, мы можем просто объединить их и получить функцию на всем интервале $[0, w]$. После этого, проделав то же самое, что и в предыдущей подзадаче, мы сможем вычислить окончательный ответ. Более того, поскольку каждая линия может вносить вклад в k-ю функцию максимума не более одного раза, эта функция будет содержать не более $2n$ отрезков (но это не имеет значения для нашего решения).

Полная асимптотика будет $O(n(n+m)\log m + qn(k+\log n))$.

Модельное решение

Авторское модельное решение можно найти по следующей ссылке:
`https://gist.github.com/dxtvzw/3d6604238d1f267cef28b88ce9f93849`