АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА РЕДИЦА

Ограниченията на подзадача 1 са такива, че и пълно изчерпване би свършило работа – и точно това се изисква от състезателите. При $N \leq 20$ можем да започнем да генерираме всички възможни разбивания на редицата с backtracking и да гледаме дали се получава вярно неравенство. Разбира се, необходимо е да се използват някои съображения – най-общо казано да не „задълбаваме“ в рекурсията, ако вече сме открили, че неравенството се „чупи“ с до тук генерираните подредици, поставяйки знаците на необходимите места.

За подзадача 2 се изисква решение със сложност $O(N^3)$. Такова решение ще предложим по-долу, което е базирано на метода на динамичното оптимиране. Нека:

min[x][y] е най-малката възможна сума на последната подредица, ако разглеждаме само подзадачата за $a_1, a_2, \ldots, a_x$ и $b_1, b_2, \ldots, b_y$ за някои $x < N$ и $y < K$. Т.е. тази подзадача има поне едно решение и *min[x][y]* е сумата на последната подредица в неравенството за такова решение (разбиване на подредици), където тази сума е минимална.

max[x][y] е най-голямата възможна сума, т.е., аналогично на предното, но тук се интересуваме от максималната възможна сума на последна подредица за тази подзадача.

Нека разгледаме някоя подзадача *(x, y)*, т.е. за $a_1, a_2, \ldots, a_x$ и $b_1, b_2, \ldots, b_y$ за някои $x < N$ и $y < K$. Нека се опитаме да намерим решение (за тази подзадача), в което последната подредица се състои от последните t на брой елемента. Сумата на тази последна подредица е $a_{x-t+1} + a_{x-t+2} + \ldots + a_x = s$. Нека последния знак b_y е “>”. Това означава, че търсим решение на подзадачата $(x - t, y - 1)$ с такава сума на последната подредица s', за която е изпълнено $s' > s$. Да, но в $max[x - t][y - 1]$ имаме най-голямата такава сума на последна подредица, ако въобще съществува решение на подзадачата $(x - t, y - 1)$. Разсъжденията, ако b_y е “<” са аналогични, но там използваме $min[x - t][y - 1]$ и обръщаме знаците, т.е. искаме $s' < s$. Направихме това разглеждане, за да видим каква е логиката на образуване на решения, използвайки *min[x][y]* и *max[x][y]*. Как се смятат те? Първо, тривиалните случаи на динамичното оптимиране са за подзадачите $(x, 0)$ за всяко x. Независимо дали става въпрос за *min[x][0]* или *max[x][0]*, това са сумата на елементите от a_1 до a_x. Рекурентните връзки са:

min[x][y] = $a_{x-t+1} + a_{x-t+2} + \ldots + a_x$, където $0 < t \leq x$ и t е възможно най-малкото, за което е изпълнено:

- Ако b_y е “<”: да съществува $min[x - t][y - 1]$ и $min[x - t][y - 1] < a_{x-t+1} + a_{x-t+2} + \ldots + a_x$
- Ако b_y е “>”: да съществува $max[x - t][y - 1]$ и $max[x - t][y - 1] > a_{x-t+1} + a_{x-t+2} + \ldots + a_x$

$max[x][y] = a_{x-t+1} + a_{x-t+2} + \ldots + a_x$, където $0 < t \leq x$ и t е възможно най-голямото, за което е изпълнено:

- Ако b_y е "<": да съществува $min[x - t][y - 1]$ и $min[x - t][y - 1] < a_{x-t+1} + a_{x-t+2} + \ldots + a_x$
- Ако b_y е ">": да съществува $max[x - t][y - 1]$ и $max[x - t][y - 1] > a_{x-t+1} + a_{x-t+2} + \ldots + a_x$

Така получаваме решение със сложност $O(N^2K)$, което е исканото за подзадача 2.

За цялостното решение на задачата се иска решение с още по-ниска сложност. Решението, което ще предложим, е със сложност $O(N^2 \log N)$ и отново стъпва на принципите на разделянето на задачата на подзадачи, както и на същата „стратегия" за дадена подзадача (x, y) да намираме *min* и *max*, които да означават същото както по-горе и да ни служат за същите разсъждения в опит да намиране решения за дадена подзадача. Идеята този път, обаче, е, веднъж решили (и сметнали съответните *min* и *max)* някаква подзадача, да можем да направим някакви заключения за по-големи подзадачи (възможно не една на брой, а повече). Нека, наистина, допуснем, че сме решили някаква подзадача (x, y) и сме пресметнали съответните $min[x][y]$ и $max[x][y]$. Нека b_{y+1} е ">". Следващите разсъждения са аналогични за обратната ситуация, с разглеждане вместо на $max[x][y]$ на $min[x][y]$, смяна на няколко знаци и индекси, но е важна идеята. Щом b_{y+1} е ">", имаме полза да разглеждаме $max[x][y]$. Какво означава „имаме полза"? Означава, че каквато и следваща подредица да образуваме, за нея знаем, че:
1. тя ще започва задължително от a_{x+1} и
2. тя (или по-скоро сумата на елементите ѝ) задължително трябва да е по-малка от тази, която завършва в a_x.

Точно заради това имаме изгода да гледаме $max[x][y]$, защото това ни е максимумът, който евентуално можем да постигнем като сума на редица завършваща в a_x. Тъй като следващата подредица започва задължително от a_{x+1}, то сумата ѝ е от вида $a_{x+1} + a_{x+2} + \ldots + a_{x+t}$. Тогава можем да заключим, че или $a_{x+1} > max[x][y]$ (т.е., въобще не съществува редица, започваща от a_{x+1}, която да може ни свърши работа, предвид намерения от нас $max[x][y]$), или съществува такова t, за което е изпълнено:

1. $max[x][y] > a_{x+1} + a_{x+2} + \ldots a_{x+t}$,
2. $max[x][y] \leq a_{x+1} + a_{x+2} + \ldots a_{x+t} + a_{x+t+1}$ (ако съществува a_{x+t+1}, т.е. $x+t+1 \leq N$)

С други думи, t е възможната най-голяма дължина на непосредствено следващата редица след тази, която свършва в a_x, за която можем да образуваме решение, взимайки предвид $max[x][y]$. Забележете, че щом можем да образуваме решение за подзадачата $(x + t, y + 1)$, то можем да образуваме решение и за всяка подзадача $(x + i, y + 1)$ за $1 \leq i \leq t$. t намираме с двоично търсене по сумите започващи от a_{x+1} и завършващи в $[x + 1, N]$. Разбира се, за целта трябва предварително да сме подготвили масив $sum[i][j]$, който да ни дава сумата на елементите от a_i до a_j включително. Целта в този момент ни

е да можем да кажем на някаква структура „За всяка подзадача *(x + i, y + 1)*, $1 \leq i \leq t$, намерих решение с последен елемент $a_{x+1} + a_{x+2} + \ldots + a_{x+i}$". Всъщност са ни нужни две такива структури, които да осведомим с горното изречение. Една, чиято цел е за всяка подзадача *(x', y')* да поддържа максималното възможно решение (т.е. максимална възможна сума на последна подредица), и втора, която да поддържа минималното възможно решение (т.е. минимална възможна сума на последна подредица). Такива структури са интервални дървета. Те имат ъпдейт метод, който за цял интервал може да обновява даден минимум/максимум, и функция-заявка, която за една точка (т.е. не за цял интервал) може да върне съответния минимум/максимум, взимайки предвид всички поправки, направени за интервали, които включват тази точка. И двете процедури се извършват със сложност *O(logN)*. След като сме разгледали всички подзадачи *(x, y)* за всяко *x* и някакво фиксирано *y*, можем твърдо да пресметнем подзадачите *(x, y + 1)* за всяко *x* с точно по едно запитване върху съответното интервално дърво. Т.е. за *min[x][y]* трябва да видим какво ще ни върне запитването върху дървото, което ни е отговорно за поддържането на минимума, за точката *x*. Това дърво (както и другото, за максимума) сме обновили с множество от интервали, за които сме сигурни, че сме намерили решения, а то е било отговорно за всяка точка да поддържа минимума (максимума).

След горните разсъждения, задачата се решава като се гледат първо всички задачи *(x, 1)* за всяко *x*, после всички подзадачи *(x, 2)* за всяко *x* и т.н.

Разбира се, за възпроизвеждането на редицата *p* е нужно да пазим масиви *back_min[][]* и *back_max[][]*, за да можем да се връщаме назад по подзадачите, т.е. в *back_min[N][K]* ще стои къде сме сложили предпоследния знак – да речем, че сме го сложили на индекс *x*, тогава предната подзадача, която трябва да гледаме, е подзадачата *(x, K-1)*. Много важен момент, когато прескачаме от подзадача към подзадача назад е да взимаме предвид отново знаците. Т.е. ако b_K е "**>**", то за намирането на решението на подзадачата *(N, K)* сме гледали *max[x][K-1]*, а не *min[x][K-1]* и точно заради това трябва да гледаме *back_max[x][K-1]*. В обратния случай трябваше да гледаме *back_min[x][K-1]*.

Автор: Момчил Иванов