АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ФЕСТИВАЛ НА ЛИМОНАДАТА

Първо нека разгледаме частния случай на една игра от *X* монети са със стойност *1*. Нека *G* е граф, чийто върхове съответстват на монетите и два върха са свързани с ребро т.с.т.к. са в една купчина. Всеки ход се изразява в това някоя пълно свързана компонента да се раздели на две нови. Наградата за ход е произведението на размерите на новополучените купчини, което от своя страна е равно на броя ребрата между двете купчини. Тъй като разделяме купчините, премахваме ребрата между тях, т.е. намаляваме броя на ребрата в графа с точно толкова, колкото печелим като награда за дадения ход. В края на играта всяка монета е в самостоятелна купчина, т.е. в графа няма повече ребра. По този начин намираме че общата награда, независимо от стратегиите на двамата играчи, е равна на броя на ребрата в първоначалния граф, който е $(X^2-X)/2$ тъй като графът е пълен.

За да моделираме общия случай, в който имаме монети с различни стойности, можем да представяме всяка монета с толкова върха, колкото е стойността й. Тъй като не можем да разделим монета на две, играта ще приключи, когато в графа останат точно компоненти с размери равни на стойностите на монетите. По този начин намираме, че и в общия случай, общата награда не зависи от стратегията на играчите и е равна на

$$\frac{X^2 - X}{2} - \sum \frac{a^2 - a}{2}$$

Нека стойността на i-тата монета на масата е a_i. Общата награда от игра, чиято купчина е съставена от монетите a_i, a_{i+1}, …, a_j е равна на:

$$\frac{1}{2}\left(\left(\sum_{k=i}^{j} a_k\right)^2 - \sum_{k=i}^{j} a_k\right) - \frac{1}{2}\sum_{k=i}^{j}(a_k{}^2 - a_k) = \frac{1}{2}\left(\left(\sum_{k=i}^{j} a_k\right)^2 - \sum_{k=i}^{j} a_k{}^2\right) = \frac{S_{ij}{}^2 - T_{ij}}{2}$$

Където S_{ij} е сумата на стойностите, а T_{ij} е сумата от квадратите на стойностите на монетите от *i* до *j*. В такъв случай, крайният отговор има форма:

$$\frac{\sum_{(i,j)\epsilon H} S_{ij} + T_{all}}{2}$$

Тъй като T_{all} участва винаги в отговора и не зависи от *H*, за да намерим минималния краен отговор, трябва да минимизираме следната сума:

$$\sum_{(i,j)\epsilon H} S_{ij}$$

Нека означим с *f(i, j)* минимума, който търсим, ако първите *i* монети се разпределят оптимално в *j* игри. За да пресметнем *f(i, j)* можед да опитаме всички възможни стойности за дължина на последната група и да изберем отпималната:

$$f(i,j) = \min_{0<k\leq i}\left(f(k-1,j-1) + S_{ij}\right)$$

Директна имплементация с динамично оптимиране води до сложност $O(N^3)$.

По-добро решение

Нека с $b(i,j)$ отбележим оптималния избор за $f(i,j)$, т.е.

$$f(i,j) \;=\; f(b(i,j)-1, j-1) \;+\; S_{b(i,j),j}$$

Първо, редицата $\{\, b(i,j) \,\}_{i=1}^{n}$ е растяща.

Ако разгледаме две позиции x и $y, x \;<\; y$ съществува[1] минимална позиция *z*, такава че $f(x, j) + S_{x+1,z} > f(y, j) + S_{y+1,z}$, с други думи като оптимизираме $f(z', j)$ е по-добре да изберем *y* пред *x* при $z' >= z$ и *x* е по-добър избор от *y* за $z' < z$.

Функцията *beat* пресмята *z* по зададени *x* и *y*. Понеже за дадена позиция *p* е лесно да се каже дали *x* или *y* е по-добър кандидат (функция *first_better*), използваме двоично търсене, за да намерим минималната позиция *z*, за която *y* е по-добър избор от *x*.

Ако за 3 позиции *x, y, z, x < y < z*, е вярно, че *beat(x, y) < beat(y, z)*, казваме че *y* е полезна, защото има поне един момент (именно *beat(x, y)*) такъв, че *y* е по-добър избор от *x*, но *z* още не е по-добър избор от *y*. Или с други думи *y* е най-добрият избор от *x*, *y* и *z*.

Функцията *mid_useful* пресмята дали *y* е полезна, при дадени *x*, *y*, *z*. Поради съображения за скорост имплементацията използва функцията *beat* само веднъж, и просто проверява, дали *b* е все още по-добър избор от *c* в момента, в който *b* е по-добър избор от *a*.

Решението се състои в това, за всяко *j* да се поддържа списък на полезни позиции, за пресмятане на *f(i, j)* за всяко *i*. Във всеки момент в списъка има полезни позиции между *a* и *i - 1*, където *a* е оптималната позиция за *f(i, j)*. Това означава, че всяка позиция е полезна, в контекста на съседните ѝ две, т.е. всяка една от тях ще бъде оптимална за поне едно *i*.

При изчисляване на *f(i, j)* се проверява, дали първата полезна позиция е по-добър кандидат от втората. Ако първата е по-добра се използва тя, ако не -- първата полезна позиция се премахва и се използва втората. След пресмятането на *f(i, j)*, *i* се добавя към списъка с полезни позиции, и евентуално премахва последните няколко позиции, ако се окаже че вече не са полезни, заради новодобавената (използвайки функцията *mid_useful*.

За да пресметнем сложността е достатъчно да отбележим, че за всяка позиция в опашката се вика *mid_useful* максимум два пъти -- веднъж когато я добавяме, и спрем да махаме позиции пред нея, и евентуално веднъж когато я махаме. Това е най-вътрешния цикъл, следователно сложността е $O(NM\ logN)$ защото сложността на *mid_useful* е $O(log\ N)$ заради двоичното търсене.

Сложността на решението е $O(NMlogN)$,

[1] съществува винаги, ако редицата е безкрайна. При крайна редица е възможно *x* да е по-добър избор от y за всяко $x < z \leq N$.

Автори: Емил Ибришимов, Искрен Чернев