**АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА**

**ОБИРИ**

Най-напред да решим задачата за случая, когато се разглежда само един вариант (*M*=1). Да преположим, че индексите на банките, които ще бъдат ограбени при оптимална организация на обирите са *ind[1] < ind[2] < ind[3] < . . . < ind[r]*. Подредили сме ги във възходящ ред, но това не значи, че обирите се извършват в този ред. Очевидно е справедливо неравенството *ind[i+1] > ind[i]+1*, тъй като която и от банките с индекси *ind[i]* или *ind[i+1]*  да бъде ограбена първа, другата ще бъде снабдена с охранителна система на следващата сутрин и няма да може да бъде ограбена. Лесно се вижда, че това условие (*ind[i+1] > ind[i]+1*) е не само необходимо, но и достатъчно, за да могат да бъдат ограбени всички банки с индекси *ind[1] < ind[2] < ind[3] < . . . < ind[r]*. Наистина при изпълнението на условието обирите могат да се извършват всяка вечер по един в нарастващ ред ред на индексите и крадците винаги ще изпреварват поставянето на охранителните системи.

Получава се доста стандартна задача – да се намери подредица с максимална сума, като подредицата не включва никои два съседни елемента от първоначалната редица.

Тази задача се решава с динамично оптимиране. Нека *f[k]* е максималната сума, която може да се получи, спазвайки условието да не се сумират два съседни елемента, разглеждайки първите *k* елемента на редицата. Ако положим *f[-1]=f[0]=0*, в сила е равенството

*f[k] = max{f[k − 1]; f[k − 2] + a[k]}, k ⩾ 1.*

За да стигнем до това равенство, трябва да разгледаме два случая:

* *a[k]* участва във формирането на *f[k]*: тогава *а[k-1]* не може да се използва, но могат да се използват елементите до *a[k-2]* – нас ни интересува максималната сума, която може да се получи от теаи елементи при спазване на условието, т.е. *f[k − 2].*
* *a[k]* не участва във формирането на *f[k]* – тогава *f[k]=f[k-1]*.

Използвайки горната зависимост *f[N]* може да бъде изчислено за време *O(N)*.

За съжаление изменението на една стойност на масива *a[]* може да измени *O(N)* стойности на *f[]*, така че, при едно изменение на елемент от масива, процесът на преизчисляване на *f[N]* трябва да бъде повторен, което ще доведе до сложност *O(M\*N)* на решението на цялата задача.

Да се опитаме да разширим горната идея.

Нека имаме два масива *A* и *B*, за всеки от които са известни четири стойности: оптималните суми от нашата задача, но за масивите, които:

* Включват първия и последния елементи на оригиналния масив;
* Включват първия, но не включват последния елементи на оригиналния масив;
* Не включват първия, но включват последния елементи на оригиналния масив;
* Не включват първия и не включват последния елементи на оригиналния масив.

Ако образуваме масив *C=A+B*, където + означава конкатенация на двата масива, т.е. добавяне на елементите на *B* след елементите на *A*, то можем лесно да пресметнем същите четири стойности за масива *C*. За целта трябва да разгледаме случаите, когато в „точката на слепваме“ вземаме последния елемент на *A* или първия на *B* (не можем да вземем и двата, защото ще получим два съседни елемента в сумата), както и различните варианти за начало и край чрез другите краища на двата масива. Така, със сложност *O(1)* можем да пресметнем четирите стойности за масива *C.*

Използвайки тази конструкция задачата може да бъде решена по следния начин: строим интервално дърво, във всеки връх на което се пазят четирите гореописани стойности за съответния интервал (в корена – за целия масив). Тогава, при промяна на количеството тугрици в една банка, е достатъчно да се сменят стойностите в *O(logN)* върха, отговарящи за интервалите, съдържащи тази банка. В корена ще се получат стойностите за целия масив. Това решение има сложност *O(MlogN).*