HDOJ 1024 M子段最大和问题

问题：

给定由n个整数(可能为负整数)组成的序列e1,e2,…,en,以及一个正整数m，要求确定序列的m个不相交子段，使这m个子段的总和达到最大。

分析：

设b(i，j)表示数组e的前j项中i个子段和的最大值，且第i个子段含e[j](1£ i £m，i£ j £n)。以下称b(i, j)为"最后一个元素属于第i子段的j元素i子段问题"。则n个元素中求i个子段的最优值显然为：

best(i, n) = Max{ b(i, j) } (i <= j <= n)

计算b(i，j)的最优子结构为：

b(i，j) = Max{ b(i, j-1) + e[j], Max{ b(i-1, t) } + e[j] } (i <= t < j)

这样，可以得到时间复杂度为O(m * n ^ 2)和空间复杂度为O(m * n)的MS相当漂亮而且容易理解的DP算法。当n不大的时候，这个算法足够优秀，然而，当n很大的时候，这个算法显然是不能让人满意的！

优化：

观察上面的最优子结构，我们发现b(i, j)的计算只和b(i, j-1)和b(i-1, t)有关，也就是说只和最后一个元素属于第i子段的j-1元素i子段问题和前j-1个元素的最大i-1子

段问题有关（可以分别理解为将e[j]作为最后一个元素而并入第i子段和将e[j]另起一段作为第i分段）。这样，我们只要分别用curr_best和prev_best两个一维数组保

存当前阶段和前一阶段的状态值b(i, *)和b(i-1, *) 就行了，内存使用也就可以降为O(2 * n)。

再来看看时间。分析发现，原算法低效主要是在求max_sum(i, t) = Max{b(i, t)} (i <= t < j)的时候用了O(n)的时间。其实，在求b(i, j)的过程中，我们完全

可以同时计算出max_sum(i, t)，因为max_sum(i,j) = Max{b(i,j), max_sum(i,j-1)}，这个只花费O(1)的时间。而max_sum(i,t)不就是i+1阶段中要用到的吗？

关键问题已经解决了！那如何保存max_sum呢？再开一个数组？我们可以在prev_best数组中保存！这个数组的任务相当艰巨，它既存放着i-1阶段的max_sum数

值，又存放这供i+1阶段使用的i阶段的max_sum值。MS这有点矛盾？其实这是可行的。注意到我们在计算b(i,j)时只使用了prev_best[j-1]，使用完了再也没有用

了，这样空闲着岂不浪费？其实我们可以将max_sum(i, j-1)存放到prev_best[j-1]里面——这个主意相当不错，它让所有问题迎刃而解。

现在，我们得到了一个时间复杂度为O(m * n)、空间复杂度为(2 * n)的算法。这个算法相当优秀，以至于m为小常数，n = 1000000时，结果也是瞬间就出来

了（此时算法的时间复杂度可以认为是O(n)的）。

```
#include<stdio.h>
#include<stdlib.h>
 #define MIN_SUM 0x80000000

 int max_sum(int e[], int n, int m)
{
	int *curr_best;
	int *prev_best;
	int max_sum, i, j;

	curr_best = (int*)malloc(sizeof(int) * (n + 1));
	prev_best = (int*)calloc(n + 1, sizeof(int));

	*curr_best = 0;
	e--;

	for(i = 1; i <= m; ++i)
	{
		max_sum = MIN_SUM;
		for(j = i; j <= n; ++j)
		{
			if(curr_best[j - 1] < prev_best[j - 1])
				curr_best[j] = prev_best[j - 1] + e[j];
			else
				curr_best[j] = curr_best[j - 1] + e[j];
			prev_best[j - 1] = max_sum;
			if(max_sum < curr_best[j])
				max_sum = curr_best[j];
		}
		prev_best[j - 1] = max_sum;
	}

	free(prev_best);
	free(curr_best);

	return max_sum;
}

 int main()
{
	int n, m, i, *data;
	while(scanf("%d%d", &m, &n) == 2 && n > 0 && m > 0)
	{
		data = (int*)malloc(sizeof(int) * n);
		for(i = 0; i < n; ++i)
			scanf("%d", &data[i]);
		printf("%d\n", max_sum(data, n, m));
		free(data);
	}
	return 0;
```

```
51 }
```