一个数组有 N 个元素，求连续子数组的最大和。 例如：[-1,2,1]，和最大的连续子数组为[2,1]，其和为 3

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
	int n;
	cin>>n;
	vector<int> in(n);
	for(int i=0;i<n;i++)
		cin>>in[i];
	vector<int> dp(n);
	dp[0]=in[0];
	int maxVal=in[0];
	for(int i=1;i<n;i++)
	{
		dp[i]=max(in[i],dp[i-1]+in[i]);
		if(dp[i]>maxVal)
			maxVal=dp[i];
	}
	cout<<maxVal<<endl;
	return 0;
}
```

某餐馆有 n 张桌子，每张桌子有一个参数：a 可容纳的最大人数； 有 m 批客人，每批客人有两个参数:b 人数，c 预计消费金额。 在不允许拼桌的情况下，请实现一个算法选择其中一部分客人，使得总预计消费金额最大

```
import java.util.*;

public class Main {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		while (sc.hasNext()) {
			int n = sc.nextInt();
			int m = sc.nextInt();
			int[] disk = new int[n]; //桌子数组
			for (int i = 0; i < n; i ++) {
				disk[i] = sc.nextInt();
			}
			Arrays.sort(disk); // 桌子容纳量从小到大排序
			PriorityQueue<Customer> queue = new PriorityQueue<>(); // 将客人按消费额降序加入优先级队列
			for (int i = 0; i < m; i ++) {
```

```
                int b = sc.nextInt();
                int c = sc.nextInt();
                if(b <= disk[n - 1]) queue.add(new Customer(b, c)); // 如果人数小于桌子最大容纳量,加入队列
            }
            boolean[] visited = new boolean[n]; // 记录桌子是否被占用
            long sum = 0; // 记录总盈利
            int count = 0; // 记录已使用的桌子数
            while ( ! queue.isEmpty()) {
                Customer customer = queue.poll();
                for (int i = 0; i < n; i ++) { // 为客人分配桌子
                    if(customer.peopleCount <= disk[i] && ! visited[i]) {
                        sum += customer.moneyCount;
                        visited[i] = true;
                        count ++;
                        break;
                    }
                }
                if(count == n) break;
            }
            System.out.println(sum);
        }
    }

    static class Customer implements Comparable<Customer> {
        private int peopleCount;
        private int moneyCount;

        public Customer(int peopleCount, int moneyCount) {
            this.peopleCount = peopleCount;
            this.moneyCount = moneyCount;
        }

        @Override
        public int compareTo(Customer o) {
            if(o.moneyCount > this.moneyCount) return 1;
            else if(o.moneyCount < this.moneyCount) return - 1;
            return 0;
        }
    }
}
```

小青蛙有一天不小心落入了一个地下迷宫,小青蛙希望用自己仅剩的体力值 P 跳出这个地下迷宫。为了让问题简单,假设这是一个 n*m 的格子迷宫,迷宫每个位置为 0 或者 1,0 代表这个位置有障碍物,小青蛙达到不了

这个位置;1 代表小青蛙可以达到的位置。小青蛙初始在(0,0)位置,地下迷宫的出口在(0,m-1)(保证这两个位置都是 1,并且保证一定有起点到终点可达的路径),小青蛙在迷宫中水平移动一个单位距离需要消耗 1 点体力值,向上爬一个单位距离需要消耗 3 个单位的体力值,向下移动不消耗体力值,当小青蛙的体力值等于 0 的时候还没有到达出口,小青蛙将无法逃离迷宫。现在需要你帮助小青蛙计算出能否用仅剩的体力值跳出迷宫(即达到(0,m-1)位置)。

```
mport java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Scanner;
public class Main {
private static int startx,starty,endx,endy,n,m,a[][],book[][],p; //分别为起点的坐标和出口的坐标,行和列
private static int min=Integer.MAX_VALUE;//体力值,默认最大
private static LinkedList<Point> linkedList = new LinkedList<>();
private static int next[][]=new int[][]{{0,1},{1,0},{0,-1},{-1,0}};//分别为右，下，左，上移动时需要加的坐标
private static String path = "";
    public static void main(String []args){
            Scanner sc=new Scanner(System.in);
              n=sc.nextInt();//n 行
              m=sc.nextInt();//m 列
              p=sc.nextInt();//体力
              startx=0;starty=0;
              endx=0;endy=m-1;
            a=new int[n][m];
            book=new int[n][m];
            for(int i=0;i<n;i++){
                  for(int j=0;j<m;j++){
                        a[i][j]=sc.nextInt();
                  }
            }//输入行和列
            dfs(0, 0, 0);
            if(min==Integer.MAX_VALUE){
                  System.out.println("Can not escape!");
            }else {
                  System.out.println(path.substring(0,path.length()-1));
        }

            }
    public static void dfs(int curx,int cury,int curt){
          linkedList.add(new Point(curx, cury));
                                if(curt>p){
                                      return;
                                }
```

```
                    if(curx==endx&&cury==endy&&curt<min){
                                min=curt;
                                savePath();
                                return;
                        }
                    int tx=0;
                    int ty=0;
        for(int i=0;i<=3;i++){//遍历四个方向,规定顺时针遍历
                        tx=curx+next[i][0];
                    ty=cury+next[i][1];
                    if(tx<0||tx>=n||ty<0||ty>=m){
                            continue;
                    }
                    if(a[tx][ty]==1&&book[tx][ty]==0){
                                book[tx][ty]=1;
                            switch (i) {
                            case 0:
                                curt+=1;
                                break;
                            case 1:
                                break;
                            case 2:
                                curt+=1;
                            case 3:
                                curt+=3;
                                break;
                            default:
                                break;
                            }
                                dfs(tx, ty, curt);
                                linkedList.removeLast();
                                 book[tx][ty]=0;
                    }

        }
}

private static class Point{
        int x, y;
        public Point(int x,int y){
                this.x=x;
                this.y=y;
        }
}//点类
```

```
    private static void savePath() {
            Iterator<Point> iterator = linkedList.iterator();
        StringBuilder sb = new StringBuilder();
        while (iterator.hasNext()) {
                Point point = iterator.next();
        sb.append("[").append(point.x).append(",").append(point.y).append("],");
    }
        path = sb.toString();

        }
1
}
```

输入一个正整数 n,求 n!(即阶乘)末尾有多少个 0？ 比如: n = 10; n! = 3628800,所以答案为 2

```
#include <stdlib.h>
#include <stdio.h>
using namespace std;

int main()
    {
    int n;
    scanf("%d",&n);
    //算法
    int count = 0;
    while(n){
        count += n/5;
        n /= 5;
    }
    printf("%d",count);
    return 0;
1   }
```

给定一个十进制数 M，以及需要转换的进制数 N。将十进制数 M 转化为 N 进制数

```
importjava.util.Stack;

importjava.util.Scanner;

publicclassMain{
```

```
    publicstaticvoidmain(String[] args){
        intn,base;
        Scanner scanner = newScanner(System.in);
        Stack S=newStack();
        while(scanner.hasNextInt()){
        n=scanner.nextInt();
        base=scanner.nextInt();
        chardigit[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
        while(n>0){
            S.push(digit[n%base]);
            n/=base;
        }
        while(!S.empty()){
            System.out.print(S.pop());
        }
            System.out.println();
        }
    }
}
```

给定一个有 n 个正整数的数组 A 和一个整数 sum,求选择数组 A 中部分数字和为 sum 的方案数。
当两种选取方案有一个数字的下标不一样,我们就认为是不同的组成方案。

```
/*
 给定一个有 n 个正整数的数组 A 和一个整数 sum,求选择数组 A 中部分数字和为 sum 的方案数。
```

当两种选取方案有一个数字的下标不一样,我们就认为是不同的组成方案。

解：此题使用递归的遍历方法也可以解决，但是会超时

dp 解决：

以每个物品作为横轴，以背包容量作为纵轴

0 1 2 3 4 5 6.....

0 1 0 0 0 0 0 0.....

5 1 0 0 0 0 0 1 0

其中 1 表示前 n 件物品放入容量为 M 的背包有 1 种方法，（5，0）表示重量为 5 的物品放入容量为 0 的背包的背包有 1 中方法，即不放入。0 表示恰好放满背包的方法为 0

当 M>weight[i]时，dp[M]=dp[M]+dp[M-weight[i]];意义是：放入物品 i 和不放入物品 i 的方法总和

```
 */
import java.util.*;
public class Main{
    public static long bag(int []weight,int n,int sum){
        long dp[]=new long[sum+1];
        dp[0]=1;
        int i,j;
        for(i=0;i<n;i++){
            for(j=sum;j>=weight[i];j--){
                dp[j]=dp[j-weight[i]]+dp[j];
            }
        }
        return dp[sum];
    }
    public static void main(String args[]){
        Scanner s=new Scanner(System.in);
        int n=s.nextInt();
        int sum=s.nextInt();
        int i,j;
        int arr[]=new int[n];
        for(i=0;i<n;i++){
            arr[i]=s.nextInt();
        }
        System.out.println(bag(arr,n,sum));
    }
}
```