动态规划算法

1.带备忘录的自顶向下法

```
1.求解子问题，判断子问题的解是否存在
2.如果存在直接返回
3.如果不存在，求解子问题解
public static Integer memoizedCutRod(Integer[] p, Integer n,
Integer[] r){
    Integer q = 0;
    //检查子问题解是否存在,如果存在，直接返回
    if(r[n] >= 0){
        return r[n];
    }
    //如果不存在，求解子问题解
    if(n==0){
        q = 0;
    }else if(r[n]==Integer.MIN_VALUE){
        for(int i = 1; i <= n; i++){
            Integer tmp = memoizedCutRod(p, n-i, r) + p[i-1];
            if(tmp > q){
                q = tmp;
            }
        }
    }
    return q;
}

public static void main( String[] args) {
    Integer min = Integer.MIN_VALUE;
    Integer[] p = {1, 5, 8, 9};
    Integer[] r = {min, min, min, min, min};
    Integer result = memoizedCutRod(p, 4, r);
    System.out.println(result);
}
```

2.自顶向上

```
public static Integer bottomUpCutRod(Integer[] p, Integer n,
Integer[] r){
    r[0] = 0;
    //按自然顺序，依次求解子问题
    for(int j = 1; j <= p.length; j++){
        int q = 0;
        for(int i = 1; i <= j; i++){
            int tmp = p[i-1] + r[j-i];
            if(q < tmp){
                q = tmp;
            }
        }
        r[j] = q;
    }
    return r[n];
}


public static void main(String[] args) {
    Integer min = Integer.MIN_VALUE;
    Integer[] p = {1, 5, 8, 9, 10};
    Integer[] r = {min, min, min, min, min, min};
    Integer result = bottomUpCutRod(p, 5, r);
    System.out.println(result);
}
```

3.求连续子数组的最大和

```java
public static int maxSumSubArray(int[] arr){
    int sum = arr[0]; //记录以i结尾的最大子数组之和
    int result = arr[0]; //记录最后结果
    for(int i = 1; i < arr.length; i++){
        if(sum > 0){
            sum += arr[i];
        }else{
            sum = arr[i];
        }
        if(sum > result){
            result = sum;
        }
    }
    return result;
}

public static void main(String[] args) {
    int[] arr = { 1, -2, 3, 10, -4, 7, 2, -5 };
    System.out.println(maxSumSubArray(arr));
}
```

4.求斐波那契数列

1.带备忘录自顶向下

```
public static long fibonacci(int n, long[] arr){
    if(n == 1 || n == 2){
        return arr[n-1];
    }else{
        long a = arr[n-1] > 0 ? arr[n-1] : fibonacci(n-1, arr);
        long b = arr[n-2] > 0 ? arr[n-2] : fibonacci(n-2, arr);
        arr[n] = a + b;
        return arr[n];
    }
}

public static void main(String[] args) {
    long[] arr = new long[101];
    arr[1] = 1l;
    arr[2] = 1l;
    long begin = System.currentTimeMillis();
    System.out.println(fibonacci(100, arr));
    long end = System.currentTimeMillis();
    System.out.println(end - begin);
}
```

2.自底向上

```
public static long fibonacci(int n){
    long[] arr = new long[n+1];
    for(int i = 1; i <= n; i++){
        if(i == 1 || i == 2){
            arr[i] = 1;
        }else{
            arr[i] = arr[i-1] + arr[i-2];
        }
    }
    return arr[n];
}

public static void main(String[] args) {
    long begin = System.currentTimeMillis();
    System.out.println(fibonacci(100));
    long end = System.currentTimeMillis();
    System.out.println(end - begin);
}
```