第 6 章　贪婪算法

1. 商店售货员找给 1 个顾客 n 元，用以下 7 种面值的纸币：100 元，50 元，20 元，10 元，5 元，2 元，1 元。设计一个贪婪算法，使得找的钱币张数最少。如果商店售货员找给 1 个顾客 140 元，假设钱币的面值有 9 种：100 元，70 元，50 元，20 元，10 元，7 元，5 元，2 元，1 元。用贪婪算法得到的是该问题的最优解吗？

C/C++代码：

```
#include <stdio.h>

int main(){
    int n, temp;
    int fee[8]={0, 100, 50, 20, 10, 5, 2, 1}, result[8]={0, 0, 0, 0, 0, 0, 0, 0};
    scanf("% d", &n);
    for(int j=1; j<=7; j++){
        temp=n/fee[j];
        result[j] = temp;
        n = n - temp* fee[j];
        printf("% d元 % d张\n", fee[j], temp);
    }
    return 0;
}
```

Python 代码：

```
def main():
    n =int(input())
    fee = [0, 100, 50, 20, 10, 5, 2, 1]  # 面额数组
    result = [0] *  8  # 记录每种面额的数量

    for j in range(1, 8):   # 从第 1 种面额开始
        temp = n // fee[j]
        result[j] = temp
        n -= temp *  fee[j]
print(f"{fee[j]}元 {temp}张")

if __name__ == "__main__":
    main()
```

2. 将 n 个正整数组成的一个数列，进行如下操作：每一次删除其中的两个数 a 和 b，然后在数列中加入一个数 $a \times b+1$，如此下去直至数列中剩下一个数。在所有按这种操作方式

最后得到的数中，最大的记作 max，最小的记作 min，则该数列的极差定义为 $M = \max - \min$。

C/C++代码：

```
#include <stdio.h>
#include <stdlib.h>

//比较函数，用于升序排序
int compare_asc(const void * a, const void * b) {
    return (* (int * )a - * (int * )b);
}

//比较函数，用于降序排序
int compare_desc(const void * a, const void * b) {
    return (* (int * )b - * (int * )a);
}

//计算最大值
int calculate_max(int arr[], int n) {
    int * heap = (int * )malloc(n *  sizeof(int));
    for (int i = 0; i < n; i++) {
        heap[i] = arr[i];
    }
    qsort(heap, n, sizeof(int), compare_asc); // 升序排序
    for (int i = 0; i < n - 1; i++) {
        int a = heap[i];
        int b = heap[i +1];
        heap[i +1] = a *  b + 1; // 合并最小的两个数
    }
    int max = heap[n -1];
    free(heap);
    return max;
}

//计算最小值
int calculate_min(int arr[], int n) {
    int * heap = (int * )malloc(n *  sizeof(int));
    for (int i = 0; i < n; i++) {
        heap[i] = arr[i];
    }
    qsort(heap, n, sizeof(int), compare_desc); // 降序排序
```

```
    for (int i = 0; i < n - 1; i++) {
        int a = heap[i];
        int b = heap[i +1];
        heap[i +1] = a *  b + 1; // 合并最大的两个数
    }
    int min = heap[n -1];
    free(heap);
    return min;
}

int main() {
    int n;
    scanf("% d", &n);
    int * arr = (int * )malloc(n *  sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("% d", &arr[i]);
    }
    // 计算最大值
    int max = calculate_max(arr, n);
    // 计算最小值
    int min = calculate_min(arr, n);
    // 计算极差
    int M = max - min;
    printf("% d\n", M);
    // 释放动态分配的内存
    free(arr);
    return 0;
}
```

Python 代码：

```
#比较函数, 用于升序排序
def compare_asc(a, b):
    return a - b

#比较函数, 用于降序排序
def compare_desc(a, b):
    return b - a

#计算最大值
def calculate_max(arr):
```

```
    heap = arr[: ]
    heap.sort(key=lambda x: x)  # 升序排序

    for i in range(len(heap) - 1):
        a = heap[i]
        b = heap[i +1]
        heap[i +1] = a *  b + 1  # 合并最小的两个数

    return heap[-1]

#计算最小值
def calculate_min(arr):
    heap = arr[: ]
    heap.sort(key=lambda x: -x)  # 降序排序

    for i in range(len(heap) - 1):
        a = heap[i]
        b = heap[i +1]
        heap[i +1] = a *  b + 1  # 合并最大的两个数

    return heap[-1]

#主程序
if __name__ == "__main__":
    n =int(input())  # 输入数列长度
    arr =list(map(int, input().split()))  # 输入数列
    # 计算最大值
    max_value = calculate_max(arr)
    # 计算最小值
    min_value = calculate_min(arr)
    # 计算极差
    M = max_value - min_value
    print(M)
```

3. 设计一个算法，把一个真分数 F 表示为埃及分数之和的形式。

C/C++代码：

```
#include"stdio.h"

int main(){
    int a, b, c;
```

```
    scanf("% d % d", &a, &b);
    if(a>=b)
        printf("-1");
    else
        if(a==1 ||b% a==0)
            printf("1/% d", b/a);
        else{
            while(a! =1){
                c = b/a+1;
                a = a* c - b;
                b = b* c;
                printf("1/% d", c);
                if(a==1){
                    printf("+1/% d", b/a);
                    break;
                }
                else
                    printf("+");
                if(b% a ==0){
                    printf("1/% d", b/a);
                    a=1;
                }
            }
        }
    printf("\n");
    return 0;
}
```

Python 代码：

```
def main():
    a, b =map(int, input().split())
    if a >= b:
        print("-1")
    else:
        if a == 1 or b %  a == 0:
            print(f"1/{b // a}")
        else:
            while a ! = 1:
                c = b // a +1
                print(f"1/{c}", end="")
```

```
            a = a *  c - b
            b = b *  c
            if a == 1:
                print(f"+1/{b // a}")
                break
            else:
                print("+", end="")
            if b %  a == 0:
                print(f"1/{b // a}", end="")
                a =1
        print()  # 换行

if __name__ == "__main__":
    main()
```

4. 给定 n 个正整数，编写一个程序找出它们中出现次数最多的数。如果这样的数有多个，输出其中最小的一个。

C/C++代码：

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_NUM1000  // 假设数字的最大值为 1000

int main() {
    int n;
    scanf("% d", &n);
    int * nums = (int * )malloc(n *  sizeof(int));
    if (nums == NULL) {
        printf("内存分配失败 \n");
        return 1;
    }

    for (int i = 0; i < n; i++) {
        scanf("% d", &nums[i]);
    }

    // 创建一个哈希表来记录每个数字的出现次数
    int hash[MAX_NUM +1] = {0};

    // 统计每个数字的出现次数
```

```
    for (int i = 0; i < n; i++) {
        hash[nums[i]]++;
    }
    int max_count =0;
    int result =0;
    // 找出出现次数最多的数字
    for (int i = 1; i <= MAX_NUM; i++) {
        if (hash[i] > max_count) {
            max_count = hash[i];
            result = i;
        }
    }
    printf("% d\n", result);
    free(nums);
    return 0;
}
```

Python 代码：

```
from collections import Counter

def main():
    n =int(input())
    nums =list(map(int, input().split()))

    # 使用 Counter 统计每个数字的出现次数
    count_dict = Counter(nums)

    # 找出出现次数最多的数字(如果有多个, 选择最小的)
    result =max(count_dict.keys(), key=lambda x: (count_dict[x], -x))

    print(result)

if __name__ == "__main__":
    main()
```

5. 键盘输入一个高精度的正整数 n，去掉其中任意 s 个数字后剩下的数字按原左右次序组成一个新的正整数。编程对给定的 n 和 s，寻找一种方案使得剩下的数字组成的新的正整数最小。

C/C++代码：

```
#include <stdio.h>
#include <string.h>
```

```
void removeDigits(char * num, int s) {
    int len = strlen(num);
    int k, i, j;

    for (k = 0; k < s; k++) {
        // 找到第一个递减的位置
        for (i = 0; i < len - 1; i++) {
            if (num[i] > num[i + 1]) break;
        }
        // 删除该位置的字符
        for (j = i; j < len - 1; j++) {
            num[j] = num[j +1];
        }
        len--;
        num[len] ='\0'; // 更新字符串结束符
    }

    // 处理前导零
    i =0;
    while (i < len && num[i] == '0') i++;
    if (i == len) {
        num[0] = '0';
        num[1] = '\0';
    }else {
        for (j = 0; i < len; j++, i++) {
            num[j] = num[i];
        }
        num[j] ='\0';
    }
}

int main() {
    char num[1000]; // 假设输入数字不超过1000位
    int s;
    scanf("%s %d", num, &s);
    removeDigits(num, s);
    printf("%s\n", num);
    return 0;
}
```

Python 代码：

```
def remove_digits(num, s):
    num =list(num)  # 将字符串转换为列表，方便操作
    for _ in range(s):
        # 找到第一个递减的位置
        i =0
        while i < len(num) - 1 and num[i] <= num[i + 1]:
            i +=1
        # 删除该位置的字符
        num.pop(i)

    # 处理前导零
    i =0
    while i < len(num) and num[i] == '0':
        i +=1
    if i == len(num):   # 如果全部是零，保留一个零
        num = ['0']
    else:
        num = num[i: ]# 去掉前导零
    return ''.join(num)  # 将列表转换为字符串

def main():
    # 输入数字和 s
    num, s =input().split()
    s =int(s)
    # 删除 s 个数字
    result = remove_digits(num, s)
    # 输出结果
    print(result)

if __name__ == "__main__":
    main()
```