贪心算法

贪心算法（Greedy Algorithm）是解决最优化问题的一种常用方法。

在每一步选择中，它总是做出在当前看来最好的选择，而不考虑之后的影响。这种策略希望通过一系列的局部最优选择，最终导致全局的最优解。

核心特征：

- 局部最优选择：每一步都只关注当前状态下的最优方案
- 无后效性：一旦做出了某个选择，这个选择就不会受后续决策的影响，也不会改变已经确定的结果。
- 最优子结构：问题的最优解包含其子问题的最优解。

贪心算法的基本步骤：

- 建立数学模型
- 制定贪心策略
- 迭代求解
- 合并解

贪心算法标准代码模板

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// 节点结构体，存储每个任务的信息
struct Node {
    int id;       // 任务编号
    long long a;  // 任务参数a（如开始时间）
    long long b;  // 任务参数b（如结束时间）
};

// 自定义比较函数：按a值从小到大排序
bool cmp(Node x, Node y) {
    return x.a < y.a;
}

int main() {
    ios::sync_with_stdio(false);  // 关闭C++与C输入输出同步，提高速度
    cin.tie(0);                   // 解除cin与cout的绑定，进一步提高速度

    int n;  // 任务数量
    // 循环处理多组数据（直到输入结束）
    while (cin >> n) {
        vector<Node> nums(n);  // 创建任务数组

        // 读入每个任务的数据
        for (int i = 0; i < n; i++) {
            cin >> nums[i].a >> nums[i].b;
            nums[i].id = i;  // 记录原始编号
        }

        // 按a值从小到大排序
        sort(nums.begin(), nums.end(), cmp);
```

```
        long long ans = 0;           // 最大可完成任务数
        long long last_state = -1;   // 记录上一个被选任务的a值

        // 贪心遍历：优先选择a值小的任务
        for (int i = 0; i < n; i++) {
            // 如果当前任务的b值 >= 上一个选择任务的a值，则选择当前任务
            if (nums[i].b >= last_state) {
                ans++;                   // 计数增加
                last_state = nums[i].a;  // 更新上一个选择的a值
            }
        }

        cout << ans << endl;  // 输出结果
    }

    return 0;
}
```

题目：小明的衣服

题目描述

小明买了 n 件白色的衣服，他觉得所有衣服都是一种颜色太单调，希望对这些衣服进行染色，每次染色时，他会将某种颜色的**所有**衣服寄去染色厂，第 i 件衣服的邮费为 a_i 元，染色厂会按照小明的要求将其中一部分衣服染成同一种任意的颜色，之后将衣服寄给小明，请问小明要将 n 件衣服染成不同颜色的最小代价是多少?

输入描述

第一行为一个整数 n，表示衣服的数量。

第二行包括 n 个整数 $a_1, a_2 \ldots a_n$ 表示第 i 件衣服的邮费为 a_i 元。

$(1 \leq n \leq 10^5, 1 \leq a_i \leq 10^9)$

输出描述

输出一个整数表示小明所要花费的最小代价。

输入输出样例

示例 1

输入

```
5
5 1 3 2 1
```

输出

```
25
```

运行限制

- 最大运行时间：1s
- 最大运行内存：256M

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>

using namespace std;

typedef long long ll;

int main()
{
  ios::sync_with_stdio(false);
  cin.tie(0);

  int n;
  while(cin >> n){
    priority_queue<ll, vector<ll>, greater<ll>> pq;

    for(int i=0; i<n; i++){
      ll cost;
      cin >> cost;
      pq.push(cost);
    }

    ll total_cost = 0;
    while(pq.size() > 1){
      ll first = pq.top();
      pq.pop();

      ll second = pq.top();
      pq.pop();

      ll new_pile = first + second;

      total_cost += new_pile;

      pq.push(new_pile);
    }

    cout << total_cost << endl;
  }
  return 0;
}
```