CS208 Lab5 Practice

12312110 李轩然

DDL: Apr.20

Problem 1

Description

n piles of stones are arranged around a playground. The goal is to merge all the stones into a single pile in a sequence of steps. At each step, two piles of stones can be merged into one pile, with the **cost of each merge** is the **sum of** stones in the two piles. The total cost is the sum of all individual merge costs throughout the process.

Design an algorithm to calculate both the maximum and minimum total cost required to merge the n piles into one pile.

Analysis

According to the **Greedy algorithm**, we have to merge the minimum two piles every time, so that we can get the minimum total cost; similarly merge the maximum two piles every time, so that we can get the maximum total cost. To achieve this, we can use **min-heap** and **max-heap**, every time we delete the last two nodes and push into a node whose value is the sum of them, until there only exists root and the value of root is the answer.

C++ Code

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

using namespace std;

int mincost(vector<int>& piles) {
    priority_queue<int, vector<int>, greater<int>> minheap;
    for (int pile : piles)
```

```
        minheap.push(pile);

    int sum = 0;
    while (minheap.size() > 1)
    {
        int x = minheap.top();
        minheap.pop();
        int y = minheap.top();
        minheap.pop();

        sum += x + y;
        minheap.push(x + y);
    }

    return sum;
}

int maxcost(vector<int>& piles) {
    priority_queue<int, vector<int>, less<int>> maxheap;
    for (int pile : piles)
        maxheap.push(pile);

    int sum = 0;
    while (maxheap.size() > 1)
    {
        int x = maxheap.top();
        maxheap.pop();
        int y = maxheap.top();
        maxheap.pop();

        sum += x + y;
        maxheap.push(x + y);
    }

    return sum;
}

int main()
{
    int n;
    cin >> n;
    vector<int> piles(n);
    for (int i = 0; i < n; i++)
        cin >> piles[i];

    int minn = mincost(piles);
    int maxx = maxcost(piles);
    cout << minn << " " << maxx;
    return 0;
}
```

Problem 2

Description

n piles of stones are arranged around a playground. The goal is to merge all the stones into a single pile in a sequence of steps. At each step, **at least** 2 piles and **at most** k piles can be merged into one, with the cost of each merge is the sum of the stones in the merged piles. The total cost is the sum of of all individual merge costs throughout the process.

Design an algorithm implement it in code to calculate both the maximum and minimum total cost required to merge the n piles into one pile.

Analysis

For the minimum case, appearently, we need to choose the smallest few piles each time to get the minimum total cost. But what is the number of piles we have to choose? If we choose k piles each time, it's easy to find that we get the minimum total cost if there are k piles left at last. If there are less than k piles left, there exists at least one pile, whose cost is added more times than the minimum condition (for it can be added at the last operation). Thus by the **Greedy algorithm**, we have to leave appropriate k piles at last. Using the same method, the back step must merge k piles as well, until the first step. Suppose we need to merge x piles at the first step, for each time there will be $k-1$ piles disappeared, then:

$$(x-1)+t(k-1)+k=n$$

thus $x=((n-k) \mod (k-1))+1$, or $x=((n-1) \mod (k-1))+1$.

For the maximum case, still merge the largest two piles each time is the best way.

C++ Code

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

```cpp
using namespace std;

int mincost(vector<int>& piles, int n, int k) {
    priority_queue<int, vector<int>, greater<int>> minheap;
    for (int pile : piles)
        minheap.push(pile);

    int sum = 0, x = (n - 1) % (k - 1) + 1, mergecost = 0;
    if (x > 1) {
        for (int i = 0; i < x; i++)
        {
            mergecost += minheap.top();
            minheap.pop();
        }
        minheap.push(mergecost);
        sum += mergecost;
    }

    while (minheap.size() > 1)
    {
        mergecost = 0;
        for (int i = 0; i < k; i++)
        {
            mergecost += minheap.top();
            minheap.pop();
        }
        minheap.push(mergecost);
        sum += mergecost;
    }

    return sum;
}

int maxcost(vector<int>& piles) {
    priority_queue<int, vector<int>, less<int>> maxheap;
    for (int pile : piles)
        maxheap.push(pile);

    int sum = 0;
    while (maxheap.size() > 1)
    {
        int x = maxheap.top();
        maxheap.pop();
        int y = maxheap.top();
        maxheap.pop();

        sum += x + y;
        maxheap.push(x + y);
    }
```

```
    return sum;
}

int main()
{
    int T;
    cin >> T;
    while (T--)
    {
        int n, k;
        cin >> n >> k;
        vector<int> piles(n);
        for (int i = 0; i < n; i++)
            cin >> piles[i];

        int minn = mincost(piles, n, k);
        int maxx = maxcost(piles);
        cout << maxx << " " << minn << endl;
    }

    return 0;
}
```