[BÁO CÁO ASSIGNMENT 4]

Bùi Huỳnh Tây

MSSV: 24521589

06/05/2025

Cấu trúc dữ liệu và giải thuật
`IT003.P21.CTTN`

GV HDTH: Trần Đình Khang

Nội dung báo cáo Assignment 4

Mục lục

1 [H007] - Alpha Problem

1.1 Ý tưởng

Để giải bài toán, ta thực hiện các bước sau:

- Nhập số nguyên dương a và cơ số x.
- Chuyển a sang hệ cơ số x bằng cách liên tục chia a cho x, lưu các chữ số dư (tức $a\%x$) vào một vector.
- Sau khi chuyển đổi, đảo ngược vector để có thứ tự đúng (từ chữ số cao nhất đến thấp nhất).
- In lần lượt các chữ số trong vector để được biểu diễn của a trong hệ cơ số x.

1.2 Code mẫu

```
ll a;
cin >> a;

int x;
cin >> x;

vector<int> ar;
while(a){
    ar.pb(a % x);
    a /= x;
}

reverse(all(ar));
for(int c : ar)
cout << c;
```

2 [H006] - Word Merging

2.1 Ý tưởng

Để giải bài toán, ta thực hiện các bước sau:

1. Nhập chuỗi s.
2. Sử dụng một stack để xử lý chuỗi: duyệt qua từng ký tự trong s.
3. Nếu stack không rỗng và ký tự hiện tại giống với ký tự ở đỉnh stack, ta xóa ký tự ở đỉnh stack (pop).
4. Ngược lại, ta thêm ký tự hiện tại vào stack (push).
5. Sau khi duyệt hết chuỗi, số ký tự còn lại trong stack chính là số lượng chữ cái không thể nối (kết quả cần tìm).

6. In ra kích thước của stack.

2.2 Code mẫu

```cpp
#include <bits/stdc++.h>
using namespace std;
#define Size(x) (int)x.size()
int main() {
    string s;
    cin >> s;
    stack<char> st;
    for(char c : s) {
        if(Size(st) && st.top() == c)
            st.pop();
        else
            st.push(c);
    }
    cout << Size(st);
return 0;
}
```

3 [H004] - Wood cutting

3.1 Ý tưởng

Để tối thiểu hóa chi phí, ta nên ưu tiên thực hiện các nhát cắt trên những đoạn gỗ ngắn hơn trước.

Ta có thể mô phỏng quá trình cắt bằng cách sử dụng một hàng đợi ưu tiên (min-heap). Ban đầu, ta đưa tất cả các chiều dài $a_1, a_2, \ldots, a_n$ vào hàng đợi ưu tiên. Sau đó, ta thực hiện các bước sau:

1. Lấy ra hai phần tử nhỏ nhất từ hàng đợi ưu tiên. Gọi chúng là x và y.
2. Tính chi phí để ghép hai phần này lại thành một đoạn gỗ mới có chiều dài $x + y$. Cộng chi phí này vào tổng chi phí.
3. Đưa đoạn gỗ mới có chiều dài $x + y$ trở lại hàng đợi ưu tiên.
4. Lặp lại các bước trên cho đến khi trong hàng đợi chỉ còn lại một phần tử duy nhất (có chiều dài bằng S).

Tổng chi phí tích lũy được trong quá trình này chính là chi phí cắt gỗ tối thiểu.

3.2 Code mẫu

```cpp
int n, s;
cin >> s >> n;
vector<int> ar;
for(int i = 1; i <= s; i++){
    int x;
```

```
    cin >> x;
    ar.pb(x);
}

sort(all(ar), greater<int>());

ll res = 0;
for(int i = 1; i <= s; i++)
    res += 1ll * i * ar[i - 1];

cout << res;
```

4 [H002] - Tam giác cân

4.1 Ý tưởng

Bài toán yêu cầu tìm số lượng bộ ba chỉ số (i, j, k) thỏa mãn $1 \leq i < j < k \leq N$ sao cho các phần tử tương ứng a_i, a_j, a_k là độ dài ba cạnh của một tam giác cân. Tam giác cân là tam giác có ít nhất hai cạnh bằng nhau.

Để giải quyết bài toán, ta cần xét tất cả các bộ ba chỉ số (i, j, k) thỏa mãn điều kiện $1 \leq i < j < k \leq N$. Với mỗi bộ ba chỉ số này, ta kiểm tra xem ba cạnh có độ dài a_i, a_j, a_k có tạo thành một tam giác cân hay không.

Một bộ ba độ dài (x, y, z) tạo thành một tam giác cân nếu một trong các điều kiện sau đúng:

- $x = y$ và $2x > z$ (bất đẳng thức tam giác).
- $x = z$ và $2x > y$.
- $y = z$ và $2y > x$.

Lưu ý rằng trường hợp tam giác đều $(x = y = z)$ cũng được bao gồm trong định nghĩa tam giác cân.

Do đó, thuật toán sẽ là:

1. Khởi tạo một biến đếm ‘count‘ bằng 0.
2. Duyệt qua tất cả các bộ ba chỉ số (i, j, k) sao cho $1 \leq i < j < k \leq N$. Điều này có thể được thực hiện bằng ba vòng lặp lồng nhau.
3. Với mỗi bộ ba chỉ số (i, j, k), lấy ra ba giá trị tương ứng a_i, a_j, a_k.
4. Kiểm tra xem ba giá trị này có thỏa mãn một trong các điều kiện tạo thành tam giác cân ở trên hay không.
5. Nếu thỏa mãn, tăng biến đếm ‘count‘ lên 1.
6. Sau khi duyệt qua tất cả các bộ ba, giá trị của ‘count‘ chính là kết quả cần tìm.

4.2 Code mẫu

```cpp
int n;
cin >> n;

map<int,int> Map;
for(int i = 0; i < n; i++){
    int x;
    cin >> x;
    Map[x]++;
}

vector<pii> ar;
for(auto x : Map){
    ar.pb(x);
}

ll res = 0;
int p = -1;
ll sum = 0;

for(auto x : Map){
    while(p + 1 < Size(ar) && ar[p + 1].fi < 2 * x.fi) p++,
       sum+=ar[p].se;
    res += 1ll * x.se * (x.se - 1) * (x.se - 2) / 6;
    res += 1ll * x.se * (x.se - 1) / 2 * (sum - x.se);
}
cout << res;
```

5 [H002] - Difference in Height

5.1 Ý tưởng

Bài toán yêu cầu tính tổng chênh lệch độ cao giữa tất cả các cặp tòa nhà trong thành phố Alpha. Cho n tòa nhà với độ cao lần lượt là $a_1, a_2, \ldots, a_n$. Độ chênh lệch độ cao giữa hai tòa nhà i và j là $|a_i - a_j|$. Ta cần tính tổng $\sum_{1 \le i < j \le n} |a_i - a_j|$.

Để giải bài toán này một cách hiệu quả, ta có thể nhận thấy rằng việc tính trực tiếp tổng trên bằng cách duyệt qua tất cả các cặp (i, j) với $i < j$ sẽ có độ phức tạp $O(n^2)$, có thể không đủ nhanh với giới hạn $n \le 2 \times 10^5$.

Một cách tiếp cận tốt hơn là sắp xếp mảng độ cao a theo thứ tự không giảm. Giả sử sau khi sắp xếp, ta có mảng ar với các phần tử $ar[0] \le ar[1] \le \cdots \le ar[n-1]$.

Vậy, tổng chênh lệch độ cao có thể được tính bằng cách duyệt qua mảng đã sắp xếp ar. Tại mỗi vị trí i, ta tính tổng của các phần tử đứng trước nó (gọi là sum) và áp dụng công thức: $res += i \times ar[i] - sum$. Biến res sẽ tích lũy tổng chênh lệch.

Các bước thực hiện:

1. Đọc số lượng tòa nhà n.
2. Đọc độ cao của n tòa nhà và lưu vào một vector ar.
3. Sắp xếp vector ar theo thứ tự không giảm.
4. Khởi tạo một biến res (kết quả) và một biến sum (tổng các phần tử đã duyệt) bằng 0.
5. Duyệt qua vector ar từ đầu đến cuối (với chỉ số i từ 0 đến $n-1$).
6. Tại mỗi vị trí i, cập nhật kết quả: $res = res + (i \times ar[i] - sum)$.
7. Cập nhật tổng: $sum = sum + ar[i]$.
8. In ra giá trị của res.

5.2 Code mẫu

```
int n;
cin >> n;

vector<int> ar;
for(int i = 1; i<= n; i++){
    int x;
    cin >> x;
    ar.pb(x);
}

sort(all(ar));

ll res = 0;
ll sum = 0;
for(int i = 0; i < n; i++){
    res += 1ll * i * ar[i] - sum;
    sum += ar[i];
}
cout << res;
```

6 [H001] - Tom's Currency

6.1 Ý tưởng

Đếm số lượng mệnh giá tiền khác nhau mà Tom có. Sử dụng một map để lưu trữ tần số xuất hiện của mỗi mệnh giá tiền. Duyệt qua danh sách các mệnh giá tiền Tom có, với mỗi mệnh giá, tăng số lượng xuất hiện của nó trong map. Cuối cùng, số lượng các khóa duy nhất trong map chính là số lượng mệnh giá tiền phân biệt.

6.2 Code mẫu

```
int n;
cin >> n;
map<int,int> Map;
for(int i = 1; i <= n; i++){
    int x;
    cin >> x;
    Map[x]++;
}

cout << Size(Map) << '\n';
```

7 Hashing: VQ44 FLOWERS

7.1 Ý tưởng

Tìm k màu sao cho số lượng hoa khác màu lấp được là nhiều nhất. Đếm tần số xuất hiện của mỗi màu trong mảng đầu vào A bằng cách sử dụng một map 'Hash'. Tạo một vector 'res'. Duyệt qua mảng A, nếu một màu xuất hiện lần đầu tiên (tần số trong 'Hash' là 1), thêm màu đó vào 'res'. Sau đó, duyệt qua map 'Hash', với mỗi màu và tần số của nó, thêm màu đó vào 'res' số lần bằng tần số trừ 1. Cuối cùng, lấy k phần tử đầu tiên của 'res' làm kết quả.

7.2 Code mẫu

```
vector<int> get_ans(const vector<int>& A,int K){
    map<int,int> Hash;

    vector<int> res;
    for(int x : A){
        Hash[x]++;
        if(Hash[x] == 1) res.push_back(x);
    }

    for(auto x : Hash){
        for(int i = 1; i < x.second; i++)
            res.push_back(x.first);
    }

    vector<int> ans;
    for(int i = 0; i < K; i++) ans.push_back(res[i]);
    return ans;
}
```

8 Hashing: KiemKe

8.1 Ý tưởng

Đếm số lượng mã hàng khác nhau trong danh sách các mã hàng đã nhập. Sử dụng một map ‘Hash‘ để lưu trữ tần số xuất hiện của mỗi mã hàng (string). Duyệt qua vector các mã hàng ‘ids‘. Với mỗi mã hàng, tăng số lượng xuất hiện của nó trong map ‘Hash‘. Cuối cùng, trả về kích thước của map ‘Hash‘, đây chính là số lượng mã hàng khác nhau.

8.2 Code mẫu

```
int count_distinct(const vector<string>& ids){
    map<string,int> Hash;
    for(string x: ids){
        Hash[x]++;
    }
    return (int)Hash.size();
}
```

Trên đây là bài báo cáo của em, cảm ơn quý thầy/cô đã đọc ạ!