Solution Assignment 2

Võ Lê Ngọc Thịnh

Ngày 24 tháng 3 năm 2025

Mục lục

1 Basic

1.1 Bài 1: Task

Tóm tắt bài toán: Bài toán yêu cầu xác định vị trí bàn và vị trí ngồi của Bob sao cho gần Alice nhất trong một lớp học có k dãy bàn.

Mô tả bài toán:

- Bàn được đánh số từ 1 đến k từ trên xuống dưới.
- Mỗi bàn có 2 vị trí: bên trái (1) và bên phải (2).
- Alice ngồi ở bàn p với vị trí q.
- Bob muốn tìm bàn gần nhất còn trống để ngồi, ưu tiên bàn phía trên nếu có hai lựa chọn.

Thuật toán:

- Ta đánh số thứ tự vào các chỗ ngồi, với chỗ ngồi ở hàng p và vị trí q thì số thứ tự là: $idx = 2 * (p - 1) + q$
- Gọi chỗ ngồi của Alice có số thứ tự là cur thì vị trí của Bob có thể ngồi để cùng đề với Alice là: $cur + k$ hoặc $cur - k$.
- Gọi res là số thứ tự chỗ ngồi thỏa mãn yêu cầu của Bob thì hàng và vị trí ngồi của Bob tương ứng là $(res + 1)/2$ và $(2 - (res\&1))$.

Source code:

```
ll cur = 2 * (p - 1) + q;
ll _prev = cur - k;
ll _next = cur + k;

if (_prev < 1 && _next > n)
    cout << -1;
else {
    if (_prev >= 1)
        cout << (_prev+1)/2 <<" "<< ((_prev % 2)!=0 ? 1 : 2);
    else
        cout << (_next+1)/2 <<" "<< ((_next % 2)!=0 ? 1 : 2);
}
```

Độ phức tạp thuật toán: O(1)

1.2 Bài 2: Point3D

Tóm tắt bài toán: Cho N điểm có tọa độ (x, y, z) trên không gian ba chiều. Yêu cầu sắp xếp (không được sử dụng các hàm sort trong thư viện STL) và thứ tự các điểm là **tắng dần theo** x, nếu x bằng nhau thì **tăng dần theo** y, nếu y bằng nhau thì **giảm dần theo** z.

Thuật toán:

- Do $N \leq 10^5$ nên ta cần phải cài thuật toán sắp xếp chạy được với độ phức tạp O(n) hoặc O($nlog(n)$), ví dụ như QuickSort, HeapSort hay MergeSort.
- Do dữ liệu là các điểm có tọa độ (x, y, z) trên không gian ba chiều và đồng thời để thỏa mãn tiêu chí sắp xếp của bài toán, ta cần tạo một đối tượng có ba thuộc tính x, y, z và phương thức so sánh của đối tượng đó.

Source code:

```
const int N = 1e5;
struct Point3D{
    int x, y, z;
    bool operator < (const Point3D &other)  const{
        if (x == other.x){
            if (y == other.y)   return (z < other.z);
            return (y > other.y);
        }
        return (x < other.x);
    }
} pt[N + 5];
int n;

template<class T>
void quickSort(T arr[], int left, int right) {
    \\ Ham quickSort sap xep mang arr tang dan.
    ...
}
int main(){
    cin >> n;
    for (int i=1;i<= n;i++ )
        cin >> pt[i].x >> pt[i].y >> pt[i].z;
    quickSort(pt, 1, n);
    for (int i=1;i<= n;i++ )
        cout << pt[i].x << ' ' << pt[i].y << ' ' << pt[i].z
           << '\n';
}
```

Độ phức tạp thuật toán: O($NlogN$)

1.3 Bài 3: Trộn 2 mảng

Tóm tắt bài toán: Cho hai mảng được sắp xếp tăng dần. Yêu cầu tự cài đặt thuật toán MergeSort để trộn hai mảng thành một mảng cũng được sắp xếp tăng dần.

Thuật toán:

- Giả sử hai mảng đã được sắp xếp là A và B. Ta bắt đầu từ phần tử đầu tiên của cả hai mảng, ta so sánh chúng và đưa phần tử nhỏ hơn vào mảng kết quả. Quá trình này lặp lại cho đến khi duyệt hết một trong hai mảng.

- Sau đó, nếu mảng nào còn phần tử chưa được thêm, ta tiếp tục đưa toàn bộ phần còn lại vào mảng kết quả. Kết quả cuối cùng là một mảng chứa tất cả phần tử của hai mảng ban đầu theo thứ tự tăng dần.

Source code:

```
void merge(int *a, int n, int *b, int m, int *c){
    int k = 0;
    int i = 0, j = 0;
    while (i < n && j < m){
        if (a[i] < b[j])
            c[k++] = a[i++];
        else c[k++] = b[j++];
    }

    while (i < n)
        c[k++] = a[i++];
    while (j < m)
        c[k++] = b[j++];
}
```

Độ phức tạp thuật toán: $O(N + M)$

1.4 Bài 4: Find Mex

Tóm tắt bài toán: Tìm MEX của một mảng (MEX được định nghĩa là số tự nhiên nhỏ nhất không xuất hiện trong mảng). Yêu cầu cài đặt không sử dụng các hàm sort và các CTDL set, map của thư viện STL.

Thuật toán:

- Nhận xét: MEX của một mảng gồm N số không âm có giá trị trong đoạn $[0, N]$.

- Với nhận xét trên, ta nghĩ đến việc duyệt qua từng giá trị có thể của MEX theo thứ tự tăng dần và kiểm tra nó có tồn tại trong mảng hay không. Giá trị đầu tiên mà không tìm thấy trong mảng chính là giá trị MEX của mảng.

- Để việc kiểm tra một phần tử có xuất hiện trong mảng, ta có thể sử dụng hàm binary_search với điều kiện là mảng đã được sắp xếp. Để sắp xếp mảng ta có thể sử dụng các thuật toán sắp xếp có độ phức tạp $O(Nlog(N))$ để phù hợp với giới hạn của bài toán như QuickSort, HeapSort, MergeSort.

Source code:

```
const int N = 1e5;

template<class T>
void QuickSort(T arr[], int _left,int _right){
    if (_left >= _right)    return;
    int mid = (_left + _right) >> 1;
    T pivot = arr[mid];

    int i = _left, j = _right;
    while (i < j){
        while (arr[i] < pivot)  i++;
        while (arr[j] > pivot)  j--;
        if (i <= j){
            swap(arr[i], arr[j]);
            i++, j--;
        }
    }

    if (_left < j)  QuickSort(arr, _left, j);
    if (i < _right) QuickSort(arr, i, _right);
}

int main(){
    int n;  cin >> n;
    ll *arr = new ll[n];
    for (int i=0;i< n;i++ )
        cin >> arr[i];

    QuickSort(arr, 0, n - 1);
    for (int i=0;i<= n;i++ )
        if (!binary_search(arr, arr + n, i))
            return void(cout << i),0;
}
```

Độ phức tạp của thuật toán: $O(Nlog(N))$

1.5 Bài 5: Point2D

Tóm tắt bài toán: Cho một tập gồm N điểm có tọa độ (x, y) trên mặt phẳng Oxy. Yêu cầu sắp xếp và in ra các điểm tăng dần theo x, nếu x bằng nhau thì giảm dần theo y và không được sử dụng các hàm sort trong thư viện STL.

Thuật toán:

- Do $N \leq 10^6$ nên ta cần phải dùng các thuật toán sắp xếp chạy được với độ phức tạp $O(N)$ hoặc $O(Nlog(N))$, ví dụ như QuickSort, HeapSort hay MergeSort.

Source code:

```
template<class T>
void QuickSort(T arr[], int _left,int _right){
    if (_left >= _right)    return;
    int mid = (_left + _right) >> 1;
    T pivot = arr[mid];

    int i = _left, j = _right;
    while (i < j){
        while (arr[i] < pivot)  i++;
        while (pivot < arr[j])  j--;
        if (i <= j){
            swap(arr[i], arr[j]);
            i++, j--;
        }
    }

    if (_left < j)  QuickSort(arr, _left, j);
    if (i < _right) QuickSort(arr, i, _right);
}

struct Point2D{
    int x, y;
    bool operator < (const Point2D &other)  const{
        return ((x == other.x) ? y > other.y : x < other.x);
    }
};

int main(){
    int n;  cin >> n;
    Point2D *arr = new Point2D[n];
    for (int i=0;i< n;i++ )
        cin >> arr[i].x >> arr[i].y;

```

```
        QuickSort(arr, 0, n - 1);
        for (int i=0;i< n;i++ )
            cout << arr[i].x << ' ' << arr[i].y <<'\n';
    }
```

Độ phức tạp thuật toán: $O(NlogN)$

1.6 Bài 6: VU33_MaxStr

Tóm tắt thuật toán: Cho một xâu kí tự gồm N chữ số ($2 < N \leq 10^5$). Yêu cầu tìm ra số lớn nhất mà chia hết cho 3 bằng cách xóa hoặc di chuyển các chữ số (nếu cần thiết).

Thuật toán:

- Để thỏa mãn tiêu chí là số lớn nhất, ta có thể dễ dàng thấy rằng nên xếp các chữ số lớn nhất ở đầu dãy và theo thứ tự giảm dần đến chữ số nhỏ nhất ở cuối dãy. Để thực hiện việc sắp xếp này ta có thể sử dụng thuật toán CoutingSort với độ phức tạp $O(N)$.
- Để thỏa mãn rằng số đó chia hết cho 3, ta sẽ tiếp cận bằng giải thuật tham lam:
  - Gọi M là tổng các chữ số của xâu sau khi chia lấy dư cho 3.
  - Nếu $M = 0$ thì kết quả chính là chuỗi đã sắp xếp hiện tại.
  - Nếu $M = 1$, ta cần loại bỏ một chữ số trong số các chữ số chia 3 dư 1. Nếu không có số nào chia 3 dư 1 trong dãy, ta sẽ loại bỏ hai chữ số trong số các chữ số chia 3 dư 2.
  - Nếu $M = 2$, ta cần loại bỏ một chữ số trong số các chữ số chia 3 dư 2. Nếu không có số nào chia 3 dư 2 trong dãy, ta sẽ loại bỏ hai chữ số trong số các chữ số chia 3 dư 1.
  - Khi loại bỏ chữ số, ta ưu tiên loại bỏ các chữ số có giá trị thấp nhất trước.

Source code:

```
const int N = 1e5;
int cnt[10];
int main(){
    string str; cin >> str;
    int n = str.length();
    int mod = 0;
    for (int i=0;i< n;i++ ){
        int id = str[i] - '0';
        cnt[id]++;
        mod = (mod + id) % 3;
```

```
    }
    int mod_1[] = {1, 4, 7};
    int mod_2[] = {2, 5, 8};

    if (mod == 1){
        for (int d : mod_1)
            while (mod > 0 && cnt[d] > 0)
                cnt[d]--, mod--;
    }
    else if (mod == 2){
        for (int d : mod_2){
            while (mod > 0 && cnt[d] > 0)
                cnt[d]--, mod -= 2;
        }
    }

    if (mod == 2){
        for (int d : mod_1){
           while (mod > 0 && cnt[d] > 0){
                cnt[d]--;
                mod = (mod - 1 + 3) % 3;
           }
        }
    }
    else if (mod == 1){
        for (int d : mod_2){
            while (mod > 0 && cnt[d] > 0){
                cnt[d]--;
                mod = (mod - 2 + 3) % 3;
            }
         }
    }

    for (int i=9;i>= 0;i--)
        while (cnt[i] > 0){
            cout << i;
            cnt[i]--;
        }
}
```

Độ phức tạp thuật toán: $O(Nlog(N))$

1.7 Bài 7: VQ44_FLOWERS

Tóm tắt bài toán: Cho tập N bóng đèn, mỗi bóng đèn thứ i có màu là a_i. Hãy chọn ra k bóng đèn sao số lượng màu khác nhau trong tập k bóng đèn này là nhiều nhất. Và yêu cầu không sử dụng các thuật toán sort và các CTDL set, map của thư viện STL.

Thuật toán:

- Vì ta muốn số lượng các màu khác nhau là lớn nhất, nên ta duyệt một lần qua toàn bộ N bóng đèn và lấy tất cả các màu khác nhau trong tập N bóng đèn. Ta lưu lại các mảng *check* với $check[i]$ có ý nghĩa là bóng đèn có màu loại i đã được chọn chưa và *usedPos* với $usedPos[i]$ có ý nghĩa là bóng đèn ở vị trí thứ i đã được sử dụng chưa.

- Nếu các bóng đèn lựa ra ở trên vẫn chưa đủ số lượng k bóng, ta duyệt lại một lần nữa là có thế bất kì bóng đèn nào mà chưa được sử dụng ở bước trên.

- Để thuận tiện trong việc lưu trữ, ta có thể rời rạc hóa các giá trị a_i.

Source code:

```
int main(){
    int n,k;    cin >> n >> k;
    int *arr = new int[n];
    for (int i=0;i< n;i++ )
        cin >> arr[i];
    QuickSort(arr, 0, n-1);
    vector<int> value;
    for (int i=0;i< n;i++ )
        value.push_back(arr[i]);
    UNIQUE(value);
    for (int i=0;i< n;i++ )
        arr[i] = lower_bound(ALL(value), arr[i]) - value.
           begin();

    bool *check = new bool[n];
    bool *isUsed = new bool[n];

    for (int i=0;i< n;i++ ) check[i] = isUsed[i] = false;

    for (int i=0;i< n & k > 0;i++ ){
        if (check[arr[i]] == false){
            cout << value[arr[i]] << ' ';
            check[arr[i]] = true;
            isUsed[i] = true;
            k--;
```

```
            }
        }
        for (int i=0;i< n && k > 0;i++ ){
            if (!isUsed[i]){
                cout << value[arr[i]] << ' ';
                isUsed[i] = true;
                k--;
            }
        }
    }
```

Độ phức tạp thuật toán: $O(NlogN)$

1.8 Bài 8: Kiểm kê

Tóm tắt bài toán: Cho một tập gồm N $(1 \leq N \leq 5.10^4)$ xâu chữ số. Hãy đếm số lượng các xâu chữ số khác nhau. Và yêu cầu không sử dụng các thuật toán sort và các CTDL set, map của thư viện STL.

Thuật toán:

- Đơn giản ta chỉ cần sắp xếp N xâu chữ số này theo thứ tự từ điển và duyệt qua để đếm số xâu chữ số khác nhau.

Source code:

```
template<class T>
void QuickSort(T arr[], int _left,int _right){
    ....
}

int main(){
    int n;  cin >> n;
    string *str = new string[n];
    for (int i=0;i< n;i++ )
        cin >> str[i];
    QuickSort(str, 0, n-1);
    int res = 1;
    for (int i=1;i< n;i++ )
        if (str[i] != str[i - 1])
            res++;

    cout << res;
}
```

Độ phức tạp thuật toán: $O(NlogN)$

1.9 Bài 9: MergeSort

Tóm tắt bài toán: Yêu cầu thực hiện thuật toán MergeSort trên dãy N $(0 < N < 20)$ số và sau mỗi lần thực hiện thao tác merge phải in ra mảng để theo dõi quá trình merge.

Thuật toán: Đơn giản sau hàm merge trong thuật toán MergeSort, ta chỉ cần in ra toàn bộ mảng.

Source code:

```
template <class T>
void merge(T arr[], int _left, int mid, int _right){
    int left_size = mid - _left + 1;
    int right_size = _right - mid;

    T *left_arr = new T[left_size];
    for (int i=0;i< left_size;i++ )
        left_arr[i] = arr[_left + i];

    T *right_arr = new T[right_size];
    for (int i=0;i< right_size;i++ )
        right_arr[i] = arr[mid + i + 1];

    int i = 0, j = 0, k = _left;
    while (i < left_size && j < right_size){
        if (left_arr[i] <= right_arr[j])
            arr[k++] = left_arr[i++];
        else    arr[k++] = right_arr[j++];
    }

    while (i < left_size)
        arr[k++] = left_arr[i++];

    while (j < right_size)
        arr[k++] = right_arr[j++];

    delete[] left_arr;
    delete[] right_arr;
}

template<class T>
void MergeSort(T *arr, int _left, int _right){
    if (_left >= _right)    return;
    int mid = (_left + _right) >> 1;
    MergeSort(arr, _left, mid);
    MergeSort(arr, mid+1, _right);
```

```
        merge(arr, _left, mid, _right);
        for (int i=0;i< n;i++ ){
            if (i == _left) cout << "[ ";
            cout << arr[i] << ' ';
            if (i == _right)    cout << "] ";
        }
        cout << '\n';
    }
```

1.10 Bài 10: InsertionSort

Tóm tắt bài toán: Yêu cầu cài đặt thuật toán InsertionSort để sắp xếp tăng dần một mảng N phần tử. Nếu có hành động *swap* xảy ra, hãy in ra mảng sau hành động *swap* đó.

Thuật toán: Đơn giản mỗi lần *swap* trong thuật toán InsertionSort, ta chỉ cần in ra toàn bộ mảng.

Source code:

```
template<class T>
void print(T *arr, int n){
    for (int i=0;i< n;i++ )
        cout << arr[i] << ' ';
    cout << '\n';
}

int main(){
    int n; cin >> n;
    int *arr = new int[n];
    for (int i=0;i< n;i++ ) cin >> arr[i];

    for (int i=1;i< n;i++ ){
        int val = arr[i];
        int j = i;
        while (j > 0 && arr[j - 1] > val){
            arr[j] = arr[j - 1];
            print(arr, n);
            j--;
        }
        arr[j] = val;
        print(arr, n);
    }
}
```

1.11 Bài 11: BubbleSort

Tóm tắt bài toán: Yêu cầu cài đặt thuật toán BubbleSort để sắp xếp tăng dần một mảng N phần tử. Nếu có hành động *swap* xảy ra, hãy in ra mảng sau hành động *swap* đó.

Thuật toán: Đơn giản mỗi lần *swap* trong thuật toán BubbleSort, ta chỉ cần in ra toàn bộ mảng.

Source code:

```
template<class T>
void print(T *arr, int n){
    for (int i=0;i< n;i++ )
        cout << arr[i] << ' ';
    cout << '\n';
}

int main(){
    int n;  cin >> n;
    int *arr = new int[n];
    for (int i=0;i< n;i++ ) cin >> arr[i];

    for (int turn=0;turn < n;turn++ ){
        for (int i=0;i + 1< n;i++ ){
            if (arr[i] > arr[i + 1]){
                swap(arr[i], arr[i + 1]);
                print(arr, n);
            }
        }
    }
}
```

1.12 Bài 12: SelectionSort

Tóm tắt bài toán: Yêu cầu cài đặt thuật toán BubbleSort để sắp xếp tăng dần một mảng N phần tử. Nếu có hành động *swap* xảy ra, hãy in ra mảng sau hành động *swap* đó.

Thuật toán: Đơn giản mỗi lần *swap* trong thuật toán BubbleSort, ta chỉ cần in ra toàn bộ mảng.

Source code:

```
template<class T>
void print(T *arr, int n){
    for (int i=0;i< n;i++ )
        cout << arr[i] << ' ';
    cout << '\n';
}

int main(){
    int n;  cin >> n;
    int *arr = new int[n];
    for (int i=0;i< n;i++ ) cin >> arr[i];

    for (int i=0;i< n;i++ ){
        int pMin = i;
        for (int j=i + 1;j< n;j++ ){
            if (arr[pMin] > arr[j])
                pMin = j;
        }

        if (pMin != i){
            swap(arr[pMin], arr[i]);
            print(arr, n);
        }
    }
}
```

2 Advance

2.1 Bài 1: MaxMinSum

Tóm tắt bài toán: Cho dãy gồm N phần tử, tỉnh tổng trọng số của các dãy con có K phần tử. Trong đó, trọng số của một dãy con được định nghĩa là hiệu số giữa phần tử lớn nhất và phần tử bé nhất trong dãy.

Thuật toán:

- Ta không thể tính tổng các trọng số của các dãy con có K phần tử bằng các phương pháp duyệt thông thường.
- Nhận xét: Với mỗi phần tử a_i, ta sẽ xem xét a_i đóng vai trò là giá trị max trong những dãy con nào, là giá trị min trong những dãy con nào, thì tổng giá trị mà a_i đóng góp vào kết quả là: a_i*(số dãy con có a_i là max - số dãy có a_i là min).
- Ta sắp xếp các phần tử theo giá trị tăng dần. Với $i \geq k$, số dãy con a_i đóng vai trò là max là $\binom{i-1}{k-1}$. Với $i \leq n-k+1$, số dãy con a_i đóng vai trò là min: $\binom{n-i}{k-1}$.

Source code:

```
const int N = 1e5;
const ll mod = 1e9 + 7;
ll fact[N + 5], inv[N + 5];
ll arr[N + 5];

ll binaryExpo(ll x, ll y){
    ll res = 1;
    for (;y > 0;y >>= 1){
        if (y & 1)  res = (res * x) % mod;
        x = (x * x) % mod;
    }
    return res;
}
ll C(int n, int k){
    if (k > n)  return 0;
    if (k == 0 || n == k)   return 1;
    return ((1LL*((1LL*fact[n]*inv[k])%mod)*inv[n-k])%mod);
}
int n, k;
int main(){
    cin >> n >> k;
    for (int i=1;i<= n;i++ )    cin >> arr[i];
    sort(arr + 1, arr + n + 1);

```

```
        fact[0] = 1;
        for (int i=1;i<= N;i++ )
            fact[i] = (1LL*fact[i - 1]*i) % mod;
        inv[N] = binaryExpo(fact[N], mod - 2);
        for (int i=N;i>= 1;i-- )
            inv[i - 1] = (1LL*inv[i]*i) % mod;

        ll res = 0;
        for (int i=1;i<= n;i++ ){
            if (i >= k)
                res = (res + 1LL*C(i - 1, k - 1)*arr[i]) % mod;
            if (i <= n - k + 1)
                res = (res - 1LL*C(n - i, k - 1)*arr[i]) % mod;
        }
        cout << (res + mod) % mod;
    }
```

Độ phức tạp của thuật toán: $O(NlogN)$

2.2 Bài 2: khangtd.Login1

Tóm tắt bài toán: Cho N cặp (tài khoản, mật khẩu). Với mỗi tên tài khoản được truy vấn, xuất mật khẩu ở lần cập nhật gần nhất hoặc "Chua Dang Ky!" nếu không tồn tại.

Thuật toán:

- Sử dụng CTDL map để ánh xạ tên tài khoản sang một ID duy nhất.
- Dùng vector *database* để lưu trữ danh sách mật khẩu của từng tài khoản.
- Khi nhập dữ liệu, nếu một tài khoản chưa tồn tại, ta sẽ tạo ID mới và lưu vào *database*.
- Khi truy vấn, nếu tài khoản không tồn tại, in "Chua Dang Ky!". Nếu tồn tại, lấy mật khẩu gần nhất (lớn nhất theo thứ tự nhập).

Source code:

```
map<string, int> getID;
struct User{
    vector<string> password;
    User(){}
};
vector<User> database;

```

```
int findID(const string &str){
    if (getID.find(str) == getID.end()){
        int sz = (int)getID.size();
        getID[str] = sz;
        database.emplace_back(User());
    }
    return getID[str];
}

int main(){
    int n, q;   cin >> n >> q;
    for (int i=1;i<= n;i++ ){
        string str;     cin >> str;
        int id = findID(str);
        string pass;    cin >> pass;
        database[id].password.push_back(pass);
    }

    for (int i=1;i<= q;i++ ){
        string name;    cin >> name;
        if (getID.find(name) == getID.end())
            cout << "Chua Dang Ky!\n";
        else{
            int id = getID[name];
            cout << database[id].password.back() << '\n';
        }
    }
}
```

Độ phức tạp của thuật toán: $O((N+Q)logN)$

2.3 Bài 3: khangtd.Login2

Tóm tắt bài toán: Cho N cặp (tài khoản, mật khẩu). Với mỗi tên tài khoản được truy vấn, xuất ra tất cả mật khẩu theo thứ tự thời gian hoặc "Chua Dang Ky!" nếu không tồn tại.

Thuật toán:

- Về cách xử lý dữ liệu thì bài 3 giống như bài 2, nhưng với mỗi truy vấn ta sẽ in ra tất cả các mật khẩu theo thứ tự thời gian.

Source code:

```
map<string, int> getID;
struct User{
```

```
    vector<string> password;
    User(){}
};
vector<User> database;

int findID(const string &str){
    if (getID.find(str) == getID.end()){
        int sz = (int)getID.size();
        getID[str] = sz;
        database.emplace_back(User());
    }
    return getID[str];
}

int main(){
    int n, q;   cin >> n >> q;
    for (int i=1;i<= n;i++ ){
        string str;     cin >> str;
        int id = findID(str);
        string pass;    cin >> pass;
        database[id].password.push_back(pass);
    }

    for (int i=1;i<= q;i++ ){
        string name;    cin >> name;
        if (getID.find(name) == getID.end())    cout << "Chua
           Dang Ky!\n";
        else{
            int id = getID[name];
            for (string pass : database[id].password)
                cout << pass << ' ';
            cout << '\n';
        }
    }
}
```

Độ phức tạp của thuật toán: $O(NlogN)$

2.4 Bài 4: khangtd.DetectVirus2

Tóm tắt thuật toán: Cho xâu S và T ($|S|, |T| \leq 10^6$) gồm các chữ cái thường. Kiểm tra nếu xâu S có trong xâu T thì in ra "YES" và các vị trí xuất hiện của xâu T trong xâu S. Ngược lại, ghi ra "NO".

Thuật toán:

- Do độ dài của mỗi xâu lên tới 10^6 kí tự, nên ta không thể duyệt và so sánh các xâu một cách ngây thơ được, ta sẽ sử dụng thuật toán KMP.
- Đặt xâu $P = T + "\#" + S$ và tính toán mảng pi là hàm tiền tố của xâu P với thuật toán KMP.
- Với vị trí i trên xâu P, nếu $pi[i] = |T|$ thì vị trí xuất hiện xâu T trên S là $(i - 2 * |T| + 1)$.

Source code:

```
void prefix_function(const string &str, int *pi){
    for (int i=1;i< str.length();i++ ){
        int j = pi[i - 1];
        while (j > 0 && str[i] != str[j])   j = pi[j - 1];
        if (str[i] == str[j])   j++;
        pi[i] = j;
    }
}

int main(){
    string str, pattern;    cin >> str >> pattern;
    int n = pattern.length(), m = str.length();
    string T = pattern + "#" + str;
    int *pi = new int[n + m + 1];
    for (int i=0;i< n + m + 1;i++ )
        pi[i] = 0;
    prefix_function(T, pi);

    vector<int> value;
    for (int i=0;i< T.length();i++ )
        if (pi[i] == n){
            value.push_back(i - (n + 1));

    if ((int)value.size()){
        cout << "YES\n";
        for (int i : value)
            cout << i - n + 2 << ' ';
    }
    else cout << "NO";
}
```

Độ phức tạp của thuật toán: $O(|S| + |T|)$

2.5 Bài 5: khangtd.DetectVirus2

Tóm tắt bài toán: Cho xâu S và T ($|S|, |T| \leq 10^3$) gồm các chữ cái thường. Kiểm tra nếu xâu S có trong xâu T thì in ra "YES" và các vị trí xuất hiện của xâu T trong xâu S. Ngược lại, ghi ra "NO".

Thuật toán:

- Do giới hạn ở bài toán này khá nhỏ ($|S| \leq 10^3$) nên ta có thể sử dụng thuật toán duyệt qua từng xâu con liên tiếp có độ dài $|T|$ bắt đầu ở vị trí i, ta so khớp xâu con này với xâu T.

Source code:

```
int main(){
    string S,T; cin >> S >> T;
    vector<int> ans;
    for(int i = 0; i + (int)T.size() <= (int)S.size(); i++){
        if(S.substr(i, T.size()) == T) ans.push_back(i + 1);
    }
    if(ans.empty()) cout << "NO";
    else {
        cout << "YES\n";
        for(int x : ans) cout << x << " ";
    }
}
```

Độ phức tạp của thuật toán: $O(|S| * |T|)$

2.6 Bài 6: Linear Search 4

Tóm tắt bài toán: Alice có N sinh viên, mỗi sinh viên i có chuyên môn về môn học a_i. Alice cần phải chia N sinh viên thành hai đội, sao cho số lượng các môn học khác nhau mà ở mỗi đội có sinh viên chuyên môn về môn đó chính xác bằng K. Hỏi rằng Alice có thể chia sinh viên thành hai đội thỏa mãn điều kiện đó được không?

Thuật toán:

- Gọi $type_1$ là số lượng môn học khác nhau mà mỗi môn chỉ có duy nhất một sinh viên có chuyển môn về nó, $type_2$ là số lượng các loại môn học khác nhau còn lại.

- Nếu $type_1 + type_2 > 2 * k$, tức số lượng các môn phân biệt lớn hơn $2 * k$ thì ta không thể chia sao cho mỗi đội có đúng k môn học phân biệt.

- Nếu $type_1 + 2 * type_2 < 2 * k$, tức là số lượng các môn học khả dụng không thể chia cho hai bên đội sao cho cả hai đội đều đủ k môn học phân biệt.

Source code:

```cpp
int numTest;
map<int, int> cnt;

int main(){
    cin >> numTest;
    while (numTest-- ){
        int n, k;  cin >> n >> k;
        for (int i=1;i<= n;i++ ){
            int x;  cin >> x;
            cnt[x]++;
        }
        int type_1 = 0, type_2 = 0;
        for (pii it : cnt)
            (it.se >= 2 ? type_2++ : type_1++);

        if (((type_1 + type_2) > 2*k) || ((type_1 + 2*type_2) <
           2*k))
            cout << "NO\n";
        else cout << "YES\n";
        cnt.clear();
    }
}
```

Độ phức tạp thuật toán: $O(\sum_{i=1}^{T} N_i)$

2.7 Bài 7: Vượt mức Pickleball v2

Tóm tắt bài toán: Cho N ($1 \leq N \leq 2.10^5$) lần đánh bóng, mỗi lần có một điểm số a_i ($0 \leq a_i \leq 200$). Với mỗi lần đánh bóng thứ i, để được xem là vượt mức Pickleball, điểm số của lần đó phải lớn hơn hoặc bằng 2 lần giá trị trung vị của d lần đánh bóng trước đó. (Nếu d là chẵn, trung vị được tính là trung bình cộng của hai phần tử giữa). Nhiệm vụ là đếm số lần đánh bóng vượt mức Pickleball và in ra kết quả.

Thuật toán:

- Sử dụng mảng $cnt[0..200]$ để lưu số lần xuất hiện của từng điểm số trong phạm vi d phần tử gần nhất và ta có thể tính nhanh trung vị của d phần tử trước đó.
- Khi duyệt đến phần tử thứ i, kiểm tra xem nó có vượt mức Pickleball hay không bằng cách so sánh với trung vị của d phần tử trước đó.
- Sau khi kiểm tra xong phần tử thứ i, thêm phần tử i vào cnt và loại bỏ phần tử id để đảm bảo chỉ giữ lại d phần tử gần nhất.

Source code:

```cpp
const int N = 2e5;
const int A = 200;
int n, d, res;
int arr[N + 5];
int cnt[A + 5];

int findMedian(int pos){
    for (int i=0;i<= 200;i++ )
        if (pos > cnt[i])   pos -= cnt[i];
        else    return i;
}
int main(){
    cin >> n >> d;
    for (int i=1;i<= n;i++ )    cin >> arr[i];
    for (int i=1;i<= n;i++ )
        if (i <= d){
            cnt[arr[i]]++;
            continue;
        }
        else{
            if (d % 2 == 1){
                int median = findMedian(d/2 + 1);
                if (arr[i] >= 2*median)     res++;
            }
            else{
                int median_l = findMedian(d/2);
                int median_r = findMedian(d/2 + 1);
                if (arr[i] >= (median_l + median_r))    res++;
            }
            cnt[arr[i]]++, cnt[arr[i - d]]--;
        }
    cout << res;
}
```

Độ phức tạp thuật toán: $O(200N)$

2.8 Bài 8: Bốn ông điên

Tóm tắt bài toán: Cho một mảng gồm N ($1 \leq N \leq 10^5$) phần tử. Hãy tìm số phép hoán đổi vị trí hai phần tử để được mảng có trật tự.

Thuật toán:

- Do giá trị của a_i lớn ($1 \leq a_i \leq 2.10^9$), ta cần rời rạc hóa giá trị a_i để tối ưu cho

việc xử lý.

- Ta xử lý cho trường hợp mảng tăng dần, số bước swap tối thiểu chính là tổng số cạnh trong các chu trình hoán vị:
  - Với mỗi phần tử a_i, ta lưu vị trí đúng của a_i sau khi mảng được sắp xếp tăng dần.
  - Ta duyệt qua từng phần tử a_i với $(1 \leq i \leq n)$.
  - Nếu phần tử đó đã ở đúng vị trí hoặc đã thuộc trong một chu trình khác thì bỏ qua.
  - Nếu chưa, ta bắt đầu một chu trình tại i. Từ i, ta nhảy đến j là vị trí đúng của a_i sau khi mảng được sắp xếp. Và tiếp tục ta cứ xét phần tử tại vị trí j. Tiếp tục như thế cho đến khi chu trình khép kín.
  - Với chu trình có k đỉnh, cần $k - 1$ lần swap.

Source code:

```
const int N = 1e5;
int n, arr[N + 5], sorted[N + 5];
vector<int> value, idx[N + 5];
bool used[N + 5];

ll solve(){
    for (int i=0;i<= n;i++ )    vector<int> ().swap(idx[i]);
    memset(used, false, sizeof(used));
    for (int i=1;i<= n;i++ )
        if (arr[i] == sorted[i])    used[i] = true;
        else    idx[sorted[i]].push_back(i);

    ll ret = 0;
    for (int i=1;i<= n;i++ ){
        int cur = i, cnt = 0;
        while (!used[cur]){
            used[cur] = true;
            cnt++;
            int _next = idx[arr[cur]].back();
            idx[arr[cur]].pop_back();
            cur = _next;
        }
        ret += (cnt > 0 ? cnt - 1 : 0);
    }
    return ret;
}
```

```
27
28      int main(){
29          cin >> n;
30          for (int i=1;i<= n;i++ ){
31              cin >> arr[i];
32              value.push_back(arr[i]);
33          }
34          sort(ALL(value));
35          UNIQUE(value);
36          for (int i=1;i<= n;i++ )
37              arr[i] = lower_bound(ALL(value), arr[i]) - value.
                    begin();
38          for (int i=1;i<= n;i++ )
39              sorted[i] = arr[i];
40
41          ll res = LLONG_MAX;
42          sort(sorted + 1, sorted + n + 1);
43          res = min(res,solve());
44          sort(sorted + 1, sorted + n + 1, greater<int>());
45          res = min(res,solve());
46          cout << res;
47      }
```

Độ phức tạp thuật toán: $O(NlogN)$

2.9 Bài 9: Huấn luyện chuột

Tóm tắt bài toán: Linh muốn huấn luyện N ($1 \leq N \leq 10^{12}$) chú chuột hamster bằng một trò chơi rút thăm số từ một bộ có N^2 tấm thẻ số (N số 1, N số 2, ...). Mỗi lần huấn luyện, Linh sẽ:

- Rút ra một tập hợp các số từ 1 đến N.
- Nếu dãy mới không giảm dần và khác hoàn toàn những dãy trước đó, lần huấn luyện đó được xem là thành công.

Linh muốn biết sau T lần thử nghiệm, số lần huấn luyện thành công là bao nhiêu. Kết quả cần được lấy modulo p ($2 \leq p \leq 10^5$).

Thuật toán:

- Nhận xét: Nếu ta bốc bất kì N số bất kì từ N^2 tấm thẻ số thì ta chỉ có duy nhất 1 cách sắp xếp cho dãy số tăng dần (giả sử những tấm thẻ có số bằng nhau là không phân biệt).

- Nhờ nhận xét trên, ta có thể thấy đáp án chính là số tổ hợp lặp chập N của N phần tử hay là $\binom{2N-1}{N}$.
- Do bài cho N rất lớn ($1 \leq N \leq 10^{12}$) và phải modulo cho p nhỏ ($2 \leq p \leq 10^5$) nên ta sẽ sử dụng Lucas Theorem để tính $\binom{2N-1}{N}$ mod p.

Source code:

```
const int N = 1e5;

ll mod;
ll fact[N + 5], inv[N + 5];

ll binaryExpo(ll x, ll y){
    ll res = 1;
    for (;y > 0;y >>= 1){
        if (y & 1)  res = (res * x) % mod;
        x = (x * x) % mod;
    }
    return res;
}

ll C(int n, int k){
    if (k > n)  return 0;
    if (k == 0 || n == k)   return 1;
    return ((1LL*((1LL*fact[n]*inv[k])%mod)*inv[n - k]) % mod);
}

ll lucas_theorem(ll n, ll k){
    if (k == 0) return 1;
    return 1LL*C(n % mod, k % mod)*lucas_theorem(n/mod, k/mod) %
       mod;
}

int main(){
    int numTest;    cin >> numTest;
    cin >> mod;
    fact[0] = 1;
    for (int i=1;i< mod;i++ )
        fact[i] = (1LL*fact[i - 1]*i) % mod;
    inv[mod - 1] = binaryExpo(fact[mod - 1], mod - 2);
    for (int i=mod - 1;i>= 1;i-- )
        inv[i - 1] = (1LL*inv[i]*i) % mod;

```

```
36      while (numTest-- ){
37          ll n;    cin >> n;
38          cout << lucas_theorem(2*n - 1, n) << '\n';
39      }
40  }
```

Độ phức tạp thuật toán: $O(p + T \log_p N)$

3 Source code

Source code của tất cả các bài.