Cấu trúc dữ liệu và giải thuật

Nhóm 15

BÁO CÁO THU HOẠCH MÔ HÌNH THUẬT TOÁN SINH KẾ TIẾP

Họ và tên	**Mã sinh viên**
Vũ Công Tuấn Dương	B22DCKH024

Ngày nộp : 23/03/2024

1 Sinh kế tiếp

1. Điều kiện áp dụng :
   - Xác định được thứ tự trên tập các cấu hình cần liệt kê. Biết cấu hình đầu tiên và cấu hình cuối cùng.
   - Từ một cấu hình chưa phải cuối cùng, ta xây dựng được thuật toán sinh ra cấu hình ngay sau nó.
2. Mô hình thuật toán:

```
Generation( ){
Bước 1: Khởi tạo
    < Thiết lập cấu hình đầu tiên > ;
Bước 2: Lặp
    while (< Lặp khi chưa phải cấu hình cuối cùng >) do
        < Đưa ra cấu hình hiện tại > ;
        < Sinh ra cấu hình kế tiếp > ;
    }
}
```

3. Nhận xét: Ta thấy với điều kiện để áp dụng mô hình thuật toán sinh kế tiếp là chúng ta phải xác định được thứ tự trên tập các cấu hình rồi sau đó tư duy để tìm ra được thuật toán để sinh ra cấu hình ngay sau nó

2 Sinh nhị phân

1. Xác định thứ tự :
   - Ta sẽ sắp xếp tập các cấu hình theo thứ tự tăng dần, so sánh như sau: đi sang bên phải từ vị trí đầu tiên bên trái , nếu gặp vị trí đầu tiên mà tại đó giá trị khác nhau thì xâu có giá trị tại vị trí đó lớn hơn thì lớn hơn.
   - Ví dụ với xâu nhị phân có độ dài bằng 3, ta cần so sánh "011" và "001" thì ta so sánh như sau:
     - Tại vị trí đầu tiên, 2 xâu giống nhau (cùng bằng 0)
     - Tại vị trí thứ hai, xâu thứ nhất có giá trị 1 và xâu thứ hai có giá trị là 0 -> xâu thứ nhất lớn hơn xâu thứ hai
   - Cấu hình đầu tiên: Xâu có độ dài n gồm n phần tử 0
   - Cấu hình cuối cùng: Xâu có độ dài n gồm n phần tử 1
2. Tư duy tìm thuật toán: Ta thấy tại một cấu hình bất kỳ chưa phải cấu hình cuối cùng thì để sinh ra cấu hình ngay sau nó cần thoả mãn đồng thời 2 điều kiện:
   - Cấu hình sau lớn hơn cấu hình hiện tại-tức là cần phải tăng xâu hiện tại
   - Cấu hính sau phải là cấu hình ngay kế tiếp cấu hình hiện tại-tức là xâu tăng và tăng nhỏ nhất có thể

Từ hai điều kiện này, ta có thể tư duy như sau: Để tăng thì ta phải chọn vị trí phần tử 0 để tăng lên 1. Tuy nhiên, chúng ta cần thoả mãn điều kiện thứ hai là tăng nhỏ nhất có thể. Do đó, ta đi từ phải sang trái để tìm vị trí đầu tiên có phần tử 0 để tăng(để bảo đảm điều kiện 2), sau khi tăng thì giữ nguyên phần bên trái số 0 và phần bên phải số 0 phải giảm về 0(để bảo đảm điều kiện 2) . Từ đó, ta có được thuật toán sinh nhị phân kế tiếp

3. Biểu diễn thuật toán:

```
#include <bits/stdc++.h>
using namespace std;
int a[101],n;
bool ok = true;
void init();
void display();
void genNext();
int main () {
    init();
    while (ok) {
        display();
        genNext();
    }
    return 0;
}
void genNext() {
    int i = n;
    while (i > 0 && a[i] == 1) {
        a[i] = 0;
        i--;
    }
    if (i > 0) {
        a[i] = 1;
    } else {
        ok = false;
    }
}
void display() {
    for (int i = 1; i <= n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
}
void init() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        a[i] = 0;
    }
}
```

4. Áp dụng :DSA01008: XÂU NHỊ PHÂN CÓ K BIT 1

- Tiếp cận; Ta sẽ sử dụng thuật toán sinh nhị phân với một chút sửa

đổi cho bài tập này. Ta vẫn sinh ra các xâu nhị phân như bài toán gốc nhưng sẽ có thêm bước kiểm tra xem có thoa mãn có k bit 1 không.

- Code bài toán:

```
#include <bits/stdc++.h>
using namespace std;
int a[101], n, k;
bool ok;
void init();
void display();
void genNext();
void testCase();
bool check();
int main () {
    int t;
    cin >> t;
    while (t--) {
        testCase();
        // cout << endl;
    }

    return 0;
}
bool check() {
    int count1s = 0;
    for (int i = 1; i <= n; i++) {
        if (a[i] == 1) {
            count1s ++;
        }
    }
    return (count1s == k);
}
void testCase() {
    init();
    while (ok) {
        display();
        genNext();
    }
}
void genNext() {
    int i = n;
    while (i > 0 && a[i] == 1) {
        a[i] = 0;
        i--;
    }
    if (i > 0) {
        a[i] = 1;
    } else {
        ok = false;
    }
}
```

```
void display() {
    if (!check()) {
        return;
    }
    for (int i = 1; i <= n; i++) {
        cout << a[i];
    }
    cout << endl;
}
void init() {
    cin >> n >> k;
    ok = true;
    for (int i = 1; i <= n; i++) {
        a[i] = 0;
    }
}
```

5. Áp dụng: DSA01020:XÂU NHỊ PHÂN TRƯỚC

- Tiếp cận: Ta sẽ áp dụng thuật toán sinh nhị phân nhưng làm ngược lại để sinh ra được cấu hình ngay trước nó
- Code:

```
#include <bits/stdc++.h>
using namespace std;
void testCase();
int main() {
    // Write your code here
    int t;
    cin >> t;
    while (t--) {
        testCase();
        cout << endl;
    }
    return 0;
}
void testCase() {
    string s;
    cin >> s;
    int n = s.size();
    int i = n - 1;
    while (i >= 0 && s[i] == '0') {
        s[i] = '1';
        i--;
    }
    if (i >= 0) {
        s[i] = '0';
        cout << s;
        return;
    } else {
        for (int j = 0; j < n; j++) {
```

```
                cout << "1";
            }
        }

    }
```

3 Sinh tổ hợp

1. Xác định thứ tự : Tương tự sinh nhị phân, tuy nhiên ta có thêm ràng buộc tại một cấu hình bất kỳ thì các giá trị được sắp xếp tăng dần, ví dụ "124" chứ không phải là "421"

    - Cấu hình đầu tiên: Tại vị trí i thì có giá trị là i. Ví dụ với tổ hợp chập 3 của 5 thì cấu hình đầu tiên là "123"
    - Cấu hình cuối cùng: Với tổ hợp chập k của n thì cấu hình cuối cùng là k phần tử lớn nhất nhỏ hơn hoặc bằng n. Ví dụ với tổ hợp chập 3 của 5 thì cấu hình cuối cùng là "345"

2. Tư duy tìm thuật toán: Tương tự như sinh nhị phân, ta có hai điều kiện cần thoả mãn.

    - Đầu tiên, ta cần tìm vị trí ở đó để tăng giá trị lên, ta có thể hỏi vị trí nào không thể tăng lên được? Dó là vị trí mà tại đó giá trị giống giá trị tại cấu hình cuối cùng. Ví dụ với tổ hợp chập 3 của 5 : xâu "125" thì vị trí 5 không thể tăng lên do tại vị trí đó, cấu hình cuối cùng "345" cũng là 5. Từ đó ta có thể tìm ra vị trí đầu tiên tính từ bên phải sang (để thoả mãn điều kiện nhỏ nhất có thể) mà tại vị trí đó giá trị khác với giá trị của cấu hình cuối cùng và tăng giá trị lên 1
    - Tiếp theo, để cho sinh ra được cấu hình kế tiếp nhỏ nhất có thể thì ta cần giữ nguyên những phần tử bên trái vị trí đó, còn những phần tử bên phải thì phải nhỏ nhất có thể, vì thế chỉ có một cách duy nhất là mỗi vị trí bằng vị trí đứng trước nó cộng thêm 1

3. Biểu diễn thuật toán:

```
#include <bits/stdc++.h>
using namespace std;
int a[101], n, k;
bool ok = true;
void init();
void display();
void genNext();
int main () {
    init();
    while (ok) {
        display();
        genNext();
    }
    return 0;
}
void genNext() {
```

```
    int t = k;
    while (t > 0 && a[t] == (n - k + t)) {
        t--;
    }
    if (t <= 0) {
        ok = false;
        return;
    }
    a[t]++;
    for (int i = t + 1; i <= k; i++) {
        a[i] = a[i - 1] + 1;
    }
}
void display() {
    for (int i = 1; i <= k; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
}
void init() {
    cin >> n >> k;
    for (int i  = 1; i <= k; i++) {
        a[i] = i;
    }
}
```

4. Áp dụng: DSA01023:SỐ THỨ TỰ TỔ HỢP

- Tiếp cận: Ta vẫn sinh tổ hợp như bình thường nhưng sẽ có một biến đếm và mỗi lần sinh kế tiếp xong ta sẽ kiểm tra xem đó có phải là cấu hình đầu vào hay không, nếu có thì ta dừng vòng lặp và in ra số thứ tự
- Code:

```
#include <bits/stdc++.h>
using namespace std;
int a[101], n, k, stt;
int x[16];
bool ok;
void init();
void display();
void genNext();
int main () {
    int t;
    cin >> t;
    while (t--) {
        init();
        while (ok) {
            display();
            genNext();
        }
```

```
    }

    return 0;
}
void genNext() {

    int t = k;
    while (t > 0 && a[t] == (n - k + t)) {
        t--;
    }
    if (t <= 0) {
        ok = false;
        return;
    }
    a[t]++;
    for (int i = t + 1; i <= k; i++) {
        a[i] = a[i - 1] + 1;
    }
}
void display() {
    stt++;
    for (int i = 1; i <= k; i++) {
        if (a[i] != x[i]) {
            return;
        }
    }
    cout << stt << endl;
    ok = false;
    return;
}
void init() {
    cin >> n >> k;
    ok = true;
    stt = 0;
    for (int i  = 1; i <= k; i++) {
        cin >> x[i];
        a[i] = i;
    }
}
```

5. Áp dụng:DSA01028:LIỆT KÊ TỔ HỢP

- Tiếp cận: Ta sẽ tiền xử lý dữ liệu với input bằng cách tạo ra mảng x gồm các phần tử khác nhau của mảng đầu vào rồi tạo mảng a và thực hiện sinh tổ hợp chập k của n với n là độ dài mảng x chứ không phải n đầu vào. Mỗi lần sinh ra cấu hình tiếp theo thì giá trị của mỗi vị trí của mảng a sẽ là chỉ số của đến mảng x và in ra kết quả
- Code:

```
#include <bits/stdc++.h>
using namespace std;
```

```
int a[101], n, k;
int x[101];
set<int> se;
bool ok;
void init();
void display();
void genNext();
int main () {
    init();
    while (ok) {
        display();
        genNext();
    }
    return 0;
}
void genNext() {
    int t = k;
    while (t > 0 && a[t] == (n - k + t)) {
        t--;
    }
    if (t <= 0) {
        ok = false;
        return;
    }
    a[t]++;
    for (int i = t + 1; i <= k; i++) {
        a[i] = a[i - 1] + 1;
    }
}
void display() {
    for (int i = 1; i <= k; i++) {
        cout << x[a[i]] << " ";
    }
    cout << endl;
}
void init() {
    cin >> n >> k;
    ok = true;
    for (int i = 1; i <= n; i++) {
        cin >> x[i];
        se.insert(x[i]);
    }
    int j = 1;
    for (auto temp : se) {
        x[j++] = temp;
    }
    n = se.size();
    for (int i  = 1; i <= k; i++) {
        a[i] = i;
    }
```

```
}
```

4 Sinh hoán vị

1. Xác định thứ tự : Tương tự như thuật toán sinh nhị phân. Với sinh hoán vị thì do hoán vị là các cách sắp xếp n số cho trước nên ta không có ràng buộc về thứ tự các phần tử của một cấu hình bất kỳ

   - Cấu hình đầu tiên: Xâu n phần tử sắp xếp tăng dần
   - Cấu hình cuối cùng : Xâu n phần tử sắp xếp giảm dần

2. Tư duy tìm thuật toán: Tương tự như sinh nhị phân, ta có hai điều kiện cần thoả mãn.

   - Đầu tiên, ta cần tìm vị trí ở đó để tăng giá trị lên. Do đây là hoán vị nên thực chất là ta chỉ sắp xếp lại các chữ số sao cho đạt được cấu hình ngay kế tiếp.Việc đầu tiên đó là tìm vị trí để tăng lên. Ta thấy thực chất việc tăng giá trị lên này thực chất là hoán đổi vị trí với phần tử khác do tính chất của hoán vị. Vậy giá trị nào cần tìm để hoán đổi vị trí? Dễ thấy không thể luôn là giá trị nhỏ nhất(1), Ta thấy cần tìm vị trí gần nhất với phần giảm dần, do cấu hình cuối cùng có dạng được sắp xếp giảm dần. Do đó, ta cần tìm vị trí đầu tiên tính từ bên phải sang mà phần tử tại vị trí đó nhỏ hơn phần tử đứng sau nó. Bước tiếp theo là cần tìm phần tử để hoán đổi vị trí với phần tử này, đó phải là phần tử nhỏ nhất mà lớn hơn phần tử này(để bảo đảm điều kiện 2). Ví dụ, ta có xâu "32541" thì phần tử cần tìm là 2, phần tử cần hoán đổi với 2 là 4.
   - Tiếp theo, để cho sinh ra được cấu hình kế tiếp nhỏ nhất có thể thì ta phải biến phần có dạng giảm dần thành tăng dần. Ví dụ "32541" thì sau khi hoán đổi thành "34521" thì ta lật ngược phần giảm dần để thành phần tăng dần: "34125"

3. Biểu diễn thuật toán:

```
#include <bits/stdc++.h>
using namespace std;
int a[101], n;
bool ok = true;
void init();
void display();
void genNext();
int main () {
    init();
    while (ok) {
        display();
        genNext();
    }
    return 0;
}
void genNext() {
    int j = n - 1;
```

```
        while (j > 0 && a[j] > a[j + 1]) {
            j--;
        }
        if (j > 0) {
            int k = n;
            while (a[k] < a[j]) {
                k--;
            }
            swap(a[k], a[j]);
            int l = j + 1, r = n;
            while (l <= r) {
                swap(a[l], a[r]);
                l++;
                r--;
            }
        } else {
            ok = false;
        }
    }
    void display() {
        for (int i = 1; i <= n; i++) {
            cout << a[i] << " ";
        }
        cout << endl;
    }
    void init() {
        cin >> n;
        for (int i  = 1; i <= n; i++) {
            a[i] = i;
        }
    }
```

4. Áp dụng:DSA01022:SỐ THỨ TỰ HOÁN VỊ

   - Tiếp cận: Ta vẫn sinh hoán vị như bình thường nhưng ta sẽ có biến đếm kiểm soát và với mỗi cấu hình được sinh ra, ta kiểm tra với cấu hình đầu vào, nếu nó giống với cấu hình đầu vào thì dừng vòng lặp và in ra số thứ tự hoán vị
   - Code:

```
#include <bits/stdc++.h>
using namespace std;
int a[101], n, stt;
int x[101];
bool ok;
void init();
void display();
void genNext();
void testCase();
int main () {
    int t;
```

```
    cin >> t;
    while (t--) {
        testCase();
        cout << endl;
    }

    return 0;
}
void testCase() {
    init();
    while (ok) {
        display();
        genNext();
    }

}
void genNext() {
    int j = n - 1;
    while (j > 0 && a[j] > a[j + 1]) {
        j--;
    }
    if (j > 0) {
        int k = n;
        while (a[k] < a[j]) {
            k--;
        }
        swap(a[k], a[j]);
        int l = j + 1, r = n;
        while (l <= r) {
            swap(a[l], a[r]);
            l++;
            r--;
        }
    } else {
        ok = false;
    }
}
void display() {
    stt++;
    for (int i = 1; i <= n; i++) {
        if (a[i] != x[i]) {
            return;
        }
    }
    cout << stt;
}
void init() {
    cin >> n;
    ok = true;
    stt = 0;
```

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

5. Áp dung:DSA01027:HOÁN VỊ DÃY SỐ

- Tiếp cận: Ta có sẽ sắp xếp mảng ban đầu theo thứ tự tăng dần rồi tạo thêm mảng tiếp theo cho việc sinh hoán vị, mỗi phần tử của mảng này sẽ là chỉ số của mảng ban đầu. Mỗi lần sinh ra được cấu hình hoán vị kế tiếp ta sẽ in ra kết quả
- Code:

```
#include <bits/stdc++.h>
using namespace std;
int a[101], n;
int x[101];
bool ok;
void init();
void display();
void genNext();
int main () {
    init();
    while (ok) {
        display();
        genNext();
    }
    return 0;
}
void genNext() {
    int j = n - 1;
    while (j > 0 && a[j] > a[j + 1]) {
        j--;
    }
    if (j > 0) {
        int k = n;
        while (a[k] < a[j]) {
            k--;
        }
        swap(a[k], a[j]);
        int l = j + 1, r = n;
        while (l <= r) {
            swap(a[l], a[r]);
            l++;
            r--;
        }
    } else {
        ok = false;
    }
}
```

```
void display() {
    for (int i = 1; i <= n; i++) {
        cout << x[a[i]] << " ";
    }
    cout << endl;
}
void init() {
    cin >> n;
    ok = true;
    for (int i = 1; i <= n; i++) {
        cin >> x[i];
        a[i] = i;
    }
    sort(x + 1, x + n + 1);
}
```