Cấu trúc dữ liệu và giải thuật

Nhóm 15

BÁO CÁO THU HOẠCH NHÓM MÔ HÌNH THUẬT TOÁN ĐỆ QUY - QUAY LUI - NHÁNH CẬN

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

Ngày nộp : 07/04/2024

1 Đệ quy

1. Hàm được gọi là đệ quy nếu nó được gọi trực tiếp hoặc gián tiếp(thông qua một hàm thứ 3) đến chính nó. Điều kiện áp dụng:

   - Phân tích được: Có thể giải được bài toán P nếu bài toán P' giống như P. P và P' chỉ khác ở dữ liệu đầu vào
   - Điều kiện dừng: Dãy các bài toán P' giống như P là hữu hạn và sẽ dừng tại một bài toán xác định nào đó

2. Mô hình thuật toán:

```
Thuật toán Recursion ( P ) {
    1. Nếu P thỏa mãn điều kiện dừng:
        <Giải P với điều kiện dừng>;
    2. Nếu P không thỏa mãn điều kiện dừng:
        <Giải P' giống như P:Recursion(P')>;
}
```

2 Quay lui

1. Dùng để liệt kê tất cả các cấu hình thoả mãn yêu cầu cho trước.Giả sử ta cần xác định bộ X =(x1, x2,..,xn) thỏa mãn một số ràng buộc nào đó. Ứng với mỗi thành phần xi ta có ni khả năng cần lựa chọn. Ứng với mỗi khả năng j thuộc ni dành cho thành phần xi ta cần thực hiện:

   - Kiểm tra xem khả năng j có được chấp thuận cho thành phần xi hay không? Nếu khả năng j được chấp thuận thì ta xác định thành phần xi theo khả năng j. Nếu i là thành phần cuối cùng (i=n) ta ghi nhận nghiệm của bài toán. Nếu i chưa phải cuối cùng ta xác định thành phần thứ i +1.
   - Nếu không có khả năng j nào được chấp thuận cho thành phần xi thì ta quay lại bước trước đó (i-1) để thử lại các khả năng còn lại.

2. Biểu diễn thuật toán:

```
Thuật toán Back-Track ( int i ) {
    for ( j =<Khả năng 1>; j <=ni; j++ ){
        if (<chấp thuận khả năng j>) {
        X[i] = <khả năng j>;
        if ( i ==n) Result();
        else Back-Track(i+1);
        }
    }
}
```

3. Áp dụng: MÃ SỐ

   - Tiếp cận; Ta sẽ sử dụng quay lui để sinh hoán vị cho 2 chữ cái đầu và liệt kê tất cả các khả năng cho 2 số ở 2 vị trí còn lại rồi kết hợp lại với nhau

- Code bài toán:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int n;
vector<int> a;
vector<int> x;
vector<string> res;
vector<string> res2;
bool unused[101];
void testCase();
void init();
void TryChar(int i);
void TryNum(int i);
int main() {
    testCase();
    return 0;
}
void TryChar(int i) {
    for (int j = 1; j <= n; j++)  {
        if (unused[j]) {
            unused[j] = false;
            a[i] = j;
            if (i == n - 1) {
                string s = "";
                for (int k = 0; k < n; k++) {
                    s += (char)(a[k] + 'A' - 1);
                }
                res.push_back(s);
            } else {
                TryChar(i + 1);
            }
            unused[j] = true; // Reset unused flag
        }
    }
}
void TryNum(int i) {
    for (int j = 1; j <= n; j++) {
        x[i] = j;
        if (i == n - 1) {
            string s = "";
            for (int k = 0; k < n; k++) {
                s += to_string(x[k]);
            }
            res2.push_back(s);
        } else {
            TryNum(i + 1);
        }
    }
```

```
}
void init() {
    res.clear();
    res2.clear();
    cin >> n;
    a.resize(n);
    x.resize(n);
    for (int i = 1; i <= n; i++) {
        unused[i] = true;
    }
}
void testCase() {
    init();
    TryChar(0);
    TryNum(0);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < res2.size(); j++) {
            cout << res[i] + res2[j] << endl;
        }
    }
}
```

4. Áp dụng: TỔ HỢP

- Tiếp cận: Ta sẽ áp dụng quay lui để liệt tổ hợp chập k của n phần tử với mảng x là chỉ số của xâu đầu vào và sử dụng map để lọc những xâu trùng nhau
- Code:

```
#include <bits/stdc++.h>
using namespace std;
int k;
string s;
int n;
string input;
map<string, int> mp;
int x[101];
void testCase();
void Try(int i);
int main() {
    // Write your code here
    int t;
    cin >> t;
    while (t--) {
        testCase();
        // cout << endl;
    }
    return 0;
}
void Try(int i) {
    for (int j = x[i - 1] + 1; j <= (n - k + i); j++) {
```

```
            x[i] = j;
            if (i == k) {
                string temp = "";
                for (int t = 1; t <= k; t++) {
                    temp += s[x[t] - 1];
                    // cout << temp;
                }
                mp[temp]++;
            } else {
                Try(i + 1);
            }
        }
    }
    void testCase() {
        cin >> s >> k;
        n  = s.size();
        mp.clear();
        x[0] = 0;
        for (int i = 1; i <= k; i++) {
            x[i] = i;
        }
        Try(1);
        for(auto temp : mp) {
            cout << temp.first << endl;
        }
        mp.clear();
    }
```

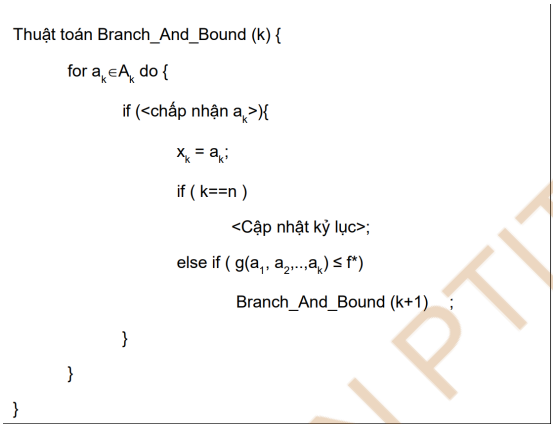
3 Nhánh cận

1. Thuật toán nhánh cận thường được dùng trong việc giải quyết các bài toán tối ưu tổ hợp. Để giải quyết bài toán tối ưu, ta có thể dùng thuật toán quay lui duyệt các phần tử. Sau đó ta xây dựng được hàm g xác định trên tất cả phưng án bộ phận cấp k của bài toán. Khi ta đã có hàm g (được gọi là hàm cận trên hoặc cận dưới tuỳ bài toán), để giảm bớt khối lượng duyệt trên tập phương án, bằng thuật toán quay lui ta xác định được X* là phương án làm cho hàm mục tiêu có giá trị nhỏ nhất trong các phương án tìm được. Ta gọi X* là phương án tốt nhất hiện có, f* là kỷ lục hiện tại. Điều này có nghĩa tập D hiện tại chắc chắn không chứa phương án tối ưu. Trong trường hợp này ta không cần phải triển khai phương án bộ phận(a1, a2,..., ak). Tập D cũng bị loại bỏ khỏi quá trình duyệt. Nhờ đó, số các phương án cần duyệt nhỏ đi trong quá trình tìm kiếm.

2. Biểu diễn thuật toán: 1

3. Áp dụng:DSA05027:CÁI TÚI

    - Tiếp cận: Ta sẽ sắp xếp lại danh sách theo thứ tự giảm dần giá trị mỗi khối lượng đồ vật rồi xây dựng hàm cận dưới.
    - Code:



Hình 1: Mô hình thuật toán Branch and Bound

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int n, v;
vector<pair<int, int>> a;
vector<int> x;
int res, deltak, bk, gk;
void testCase();
void init();
void Try(int i);
bool cmp(const pair<int, int> &a, const pair<int, int> &b);
int main() {
    int t;
    cin >> t;
    while (t--) {
        testCase();
        cout << endl;
    }
    return 0;
}
bool cmp(const pair<int, int> &a, const pair<int, int> &b) {
    double x = static_cast<double>(a.second) / a.first;
    double y = static_cast<double>(b.second) / b.first;
    return y < x;
}
void Try(int i) {
    for (int j = 1; j >= 0; j--) {
        x[i] = j;
        deltak = deltak + x[i] * a[i].second;
        bk = bk - x[i] * a[i].first;
        if (bk < 0) {
            bk += x[i] * a[i].first;
            deltak -= x[i] * a[i].second;
            continue;
        }
```

```
        if (i == n - 1) {
            res = max(res, deltak);
        } else {
            gk = deltak + static_cast<double>
            (bk * a[i + 1].second) / a[i + 1].first;
            if (gk > res) {
                Try(i + 1);
            }
        }
        deltak -= x[i] * a[i].second;
        bk += x[i] * a[i].first;
    }
}
void init() {
    res = 0;
    cin >> n >> v;
    a.resize(n);
    x.resize(n);
    deltak = 0;
    bk = v;
    for (int i = 0; i < n; i++) {
        cin >> a[i].first;
    }
    for (int i = 0; i < n; i++) {
        cin >> a[i].second;
    }
    sort(a.begin(), a.end(), cmp);
}
void testCase() {
    init();
    Try(0);
    cout << res << endl;
}
```