Lời giải [IT003.P21.CTTN.1] Assigment6 (Basic)

Bảo Quý Định Tân - 24520028

Ngày 20 tháng 5 năm 2025

Mục lục

1 Basic

1.1 Longest Increasing Sequence (LIS)

Ở bài toán này, vì n khá nhỏ ($n \leq 5 \times 10^3$), nên ta có thể làm thuật toán $O(n^2)$.

Ta gọi dp_i = độ dài dãy con tăng dài nhất kết thúc tại a_i. Công thức sẽ là $dp_i = max(1, dp_j + 1)$ với mọi $j < i$ có $(a_j < a_i)$. Đáp án sẽ lại max của mọi dp_i $(1 \leq i \leq n)$.

Để có thể truy xuất đáp án, ta lưu thêm mảng trc_i = vị trí của số liền trước a_i trong dãy con tăng dài nhất kết thúc tại a_i.

```
int ans = 0;
int ansPos = 0;
for (int i = 1; i <= n; i++) {
    dp[i] = 1;
    for (int j = 1; j < i; j++) {
        if (a[j] < a[i] && maximize(dp[i], dp[j] + 1)) {
            trc[i] = j;
        }
    }
    if (maximize(ans, dp[i])) {
        ansPos = i;
    }
}
```

Từ đó ta xuất đáp án.

```
cout << ans << '\n';
vector<int> lis;
while (ansPos) {
    lis.pb(a[ansPos]);
    ansPos = trc[ansPos];
}

reverse(all(lis));
for (int x : lis) {
    cout << x << ' ';
}
```

1.2 Knapsack

Đây là một bài toán quy hoạch động cái túi thông thường thôi.

Ta gọi $dp_{i,j}$ = giá trị lấy được nhiều nhất trong số i cái túi đầu với sức chứa là j.

Ta có công thức $dp_{i,j} = max(dp_{i-1,j}, dp_{i-1,j-w_i} + v_i)$. Đáp án sẽ là $dp_{n,W}$.

```
cin >> n >> w;
for (int i = 1; i <= n; i++) {
    int a, b;
    cin >> a >> b;
    for (int j = w; j >= a; j--) {
        maximize(dp[j], dp[j - a] + b);
    }
}

cout << *max_element(dp, dp + w + 1) << '\n';
```

1.3 Sum of Four Values

Ở bài toán này, ta sẽ tiếp cận theo hướng là với mọi i $(1 \leq i \leq n)$, giả sử ta đã lưu được các cặp và tổng tương ứng có chỉ số $< i$.

Ta sẽ for trâu các vị trí $j < i$, ghép với i và cần tìm thêm 2 phần tử nữa để có tổng X, ta sẽ lấy 2 phần tử này trong tập các cặp số cùng tổng tương ứng mà mình đã chuẩn bị. Và nhớ phải kiểm tra 2 phần tử lấy ra phải khác j.

Nghĩa là với mọi $j < i$, ta cần tìm thêm 2 phần tử có tổng là $X - a_i - a_j$.

Làm cách này sẽ đảm bảo bao quát được mọi trường hợp và không bị lặp vị trí.

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j < i; j++) {
        ll need = x - a[i] - a[j];
        if (m.count(need)) {
            if (m[need].first != j && m[need].second != j) {
                cout << i << " " << j << " " << m[need].first <<
                    " " << m[need].second;
                return 0;
            }
        }
    }

    for (int j = 1; j < i; j++) {
        m[a[i] + a[j]] = {i, j};
    }
}
```

1.4 Meet in the middle

Đây là một bài toán hay thường gặp khi học về meet in the middle.

Vì $N \leq 40$, nên ta sẽ chia mảng này ra làm hai nửa. Với nửa đầu, ta chuẩn bị trước mọi trường hợp lấy có thể xảy ra và tổng tương ứng, độ phức tạp khi làm việc này sẽ là $O(2^n)$, đây là lý do phải chia N làm nửa, tại khi này kích thước mỗi phần sẽ ≤ 20, phù hợp để thực hiện độ phức tạp $O(2^n)$.

```
vector<ll> left, right;

for (int mask = 0; mask < (1 << n); mask++) {
    ll sum = 0;
    for (int j = 0; j < n; j++) {
        if ((mask >> j) & 1) {
            sum += a[j];
        }
    }
    left.emplace_back(sum);
}

for (int mask = 0; mask < (1 << m); mask++) {
    ll sum = 0;
    for (int j = 0; j < m; j++) {
        if ((mask >> j) & 1) {
            sum += b[j];
        }
    }
    right.emplace_back(sum);
}

sort(left.begin(), left.end());
sort(right.begin(), right.end());
```

Với nửa sau, ta cũng liệt kê ra tất cả các trường hợp. Với mỗi trường hợp, ta sẽ tìm cách ghép với một vài số bên nửa đầu. Giả sử tổng hiện tại là *sum* thì ta cần tìm thử có cách nào để tạo ra tổng $x - sum$ bên nửa một không.

Hoặc ta có thể làm ngược lại.

```
ll ans = 0;
for (auto it : left) {
    // it + ?? = x <=> ?? = x - it
    int leftPos = lower_bound(right.begin(), right.end(), x - it)
        - right.begin();
    int rightPos = upper_bound(right.begin(), right.end(), x - it
        ) - right.begin();
    ans += rightPos - leftPos;
}

cout << ans;
```

1.5 Tìm đường đi trên đồ thi (BFS)

Ở bài toán này, ta chỉ BFS như thông thường và thêm mảng *trc* để xây dựng lại đường đi.

```
trc[u] = 0;
while (!q.empty()) {
    int x = q.front();
    q.pop();
    for (int y : adj[x]) {
        if (trc[y] == -1) {
            q.push(y);
            trc[y] = x;
        }
    }
}
if (~trc[v]) {
    vector<int> path;
    while (v != u) {
        path.pb(v);
        v = trc[v];
    }
    path.pb(u);
    reverse(all(path));
    for (int x : path) {
        cout << a[x] << ' ';
    }
    cout << "\n";
} else {
    cout << "no_path\n";
}
```

1.6 Tìm đường đi trên đồ thi (DFS)

Ở bài này, ta cũng DFS như thông thường và thêm mảng *trc* để có thể xây dựng lại đường đi.

```
void dfs(int u) {
    for (int v : adj[u]) {
        if (trc[v] == -1) {
            trc[v] = u;
            dfs(v);
        }
    }
}
```

```
trc[u] = 0;
```

```
dfs(u);
if (~trc[v]) {
    vector<int> path;
    while (v != u) {
        path.pb(v);
        v = trc[v];
    }
    path.pb(u);
    reverse(all(path));
    for (int x : path) {
        cout << a[x] << ' ';
    }
    cout << "\n";
} else {
    cout << "no_path\n";
}
```

1.7 khangtd.ConnectedComponents

Ta bắt đầu từ đỉnh được cho và đi đến tất cả các đỉnh còn lại cùng thành phần liên thông bằng BFS/DFS và bỏ các đỉnh đó vào một vector để xuất.

```
int n, m;
vector<int> adj[MAX_N];
vector<int> ns;
bool marked[MAX_N];

void dfs(int u) {
    ns.pb(u);
    for (int v : adj[u]) {
        if (!marked[v]) {
            marked[v] = true;
            dfs(v);
        }
    }
}
```

```
dfs(x);

sort(all(ns));
ns.erase(unique(all(ns)), ns.end());
cout << ns.size() << '\n';
for (int u : ns) {
    cout << u << ' ';
}
```

1.8 Đồ thị vô hướng - Đếm số đỉnh cô lập

Ở bài này, ta sẽ đếm bậc của từng đỉnh. Những đỉnh bị cô lập là những đỉnh có bậc là 0.

```
cin >> n >> m;
for (int i = 1; i <= m; i++) {
    int u, v;
    cin >> u >> v;
    deg[u]++;
    deg[v]++;
}

int ans = 0;
for (int i = 0; i < n; i++) {
    ans += !deg[i];
}

cout << ans;
```

1.9 Đồ thị vô hướng - Đếm số thành phần liên thông

Ở bài này, với mỗi đỉnh chưa đi qua, ta sẽ đi bắt đầu từ đỉnh đó tới các đỉnh cùng thành phần liên thông và đánh dấu lại xong tăng biến đếm số lượng thành phần liên thông lên một.

```
queue<int> q;
for (int i = 0; i < n; i++) {
    if (visited[i]) continue;
    q.push(i);
    visited[i] = true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : adj[u]) {
            if (!visited[v]) {
                visited[v] = true;
                q.push(v);
            }
        }
    }
    cc++;
}
cout << cc;
```

1.10 Đồ thị vô hướng - Liệt kê các đỉnh có thể tới từ đỉnh S

Bài này khá giống một bài ở trên, ta duyệt qua tất cả các đỉnh có thể tới được từ đỉnh được cho bằng BFS/DFS rồi lưu lại trong một vector để xuất.

```
void dfs(int u) {
    for (int v : adj[u]) {
        if (!marked[v]) {
            marked[v] = true;
            dfs(v);
            ns.pb(v);
        }
    }
}
```

```
marked[0] = true;
dfs(0);

sort(all(ns));
ns.erase(unique(all(ns)), ns.end());

if (ns.empty()) {
    cout << "KHONG\n";
    return 0;
}

for (int u : ns) {
    cout << u << ' ';
}
```

1.11 Đồ thị vô hướng - Kiểm tra có đường đi từ đỉnh S tới đỉnh E

Bắt đầu từ đỉnh được cho, ta đi qua mọi đỉnh chung thành phần liên thông, đỉnh nào đi qua thì ta đánh dấu lên để trả lời truy vấn.

```
int n, m;
vector<int> adj[MAX_N];
vector<int> ns;
bool marked[MAX_N];

void dfs(int u) {
    for (int v : adj[u]) {
        if (!marked[v]) {
            marked[v] = true;
```

```
                dfs(v);
                ns.pb(v);
            }
        }
    }
```

```
marked[0] = true;
dfs(0);

sort(all(ns));
ns.erase(unique(all(ns)), ns.end());

if (ns.empty()) {
    cout << "KHONG\n";
    return 0;
}

for (int u : ns) {
    cout << u << ' ';
}
```

1.12 Wildcard

Ở bài này ta sẽ tạo hai cây trie, một cây để lưu prefix, một cây để lưu suffix.

```
struct TrieNode {
    int child[64]{};
};

struct Trie {
    vector<TrieNode> data;

    Trie() {
        data.pb(TrieNode());
    }
};

Trie forwardTrie, backwardTrie;
```

Và mảng đếm $cnt_{i,j}$ = số lượng xét tới node i trên cây trie prefix và node j trên cây trie suffix và mảng $totalCnt_i$ là số lượng sâu đi đến node i trên cây trie prefix.

```
int cnt[10001][10001];
```

Với mỗi từ có sẵn, ta sẽ đưa lên cây trie và cập nhật mảng cnt.

```
// Setup ID mapping
for (char h = 'a'; h <= 'z'; h++) id[h] = h - 'a';
for (char h = 'A'; h <= 'Z'; h++) id[h] = h - 'A' + 26;
for (char h = '0'; h <= '9'; h++) id[h] = h - '0' + 52;
id[' '] = 62;
id['*'] = 63;

cin >> n >> m;
getline(cin, s); // consume the newline

vector<string> strings(n + 1);
for (int k = 1; k <= n; k++) {
    getline(cin, s);

    int p = 0;
    for (int i = 0; i < s.size(); i++) {
        if (!forwardTrie.data[p].child[id[s[i]]]) {
            forwardTrie.data[p].child[id[s[i]]] = forwardTrie.
                data.size();
            forwardTrie.data.pb(TrieNode());
        }
        p = forwardTrie.data[p].child[id[s[i]]];

        int p1 = 0;
        cnt[p][0]++;
        for (int j = s.size() - 1; j > i; j--) {
            if (!backwardTrie.data[p1].child[id[s[j]]]) {
                backwardTrie.data[p1].child[id[s[j]]] =
                    backwardTrie.data.size();
                backwardTrie.data.pb(TrieNode());
            }
            p1 = backwardTrie.data[p1].child[id[s[j]]];
            cnt[p][p1]++;
        }
    }
    totalCnt[p]++;

    int p1 = 0;
    for (int j = s.size() - 1; j >= 0; j--) {
        if (!backwardTrie.data[p1].child[id[s[j]]]) {
            backwardTrie.data[p1].child[id[s[j]]] = backwardTrie.
                data.size();
            backwardTrie.data.pb(TrieNode());
        }
        p1 = backwardTrie.data[p1].child[id[s[j]]];
        cnt[0][p1]++;
    }
    cnt[0][0]++;
```

```
}
```

Khi truy vấn, ta sẽ chia ra các trường hợp:

- Không có dấu $*$: Ta sử dụng mảng $totalCnt$ để tính đáp án.
- Chuỗi chỉ có duy nhất một dấu $"*"$: đáp án là n.
- Chuỗi dạng $"*suff"$: ta lấy đáp án trên cây trie suff.
- Chuỗi dạng $"pref*"$: ta lấy đáp án trên cây trie pref.
- Chuỗi dạng $"pref*suff"$: ta đi đến đỉnh i tương ứng là phần pref của chuỗi trên cây trie pref và đi đến đỉnh j tương ứng là phần suff của chuỗi trên cây trie suff. Đáp án sẽ là $cnt_{i,j}$.

```
for (int i = 1; i <= m; i++) {
    getline(cin, s);
    int cntStar = count(all(s), '*');

    if (cntStar == 0) {
        int p = 0;
        for (int j = 0; j < s.size(); j++) {
            if (!forwardTrie.data[p].child[id[s[j]]]) {
                p = -1;
                break;
            }
            p = forwardTrie.data[p].child[id[s[j]]];
        }
        cout << (~p ? totalCnt[p] : 0) << '\n';
        continue;
    }


    if (s.size() == 1 && s[0] == '*') {
        cout << n << '\n';
        continue;
    }

    if (s[0] == '*') {
        s.erase(s.begin());
        int p = 0;
        for (int j = s.size() - 1; j >= 0; j--) {
            if (!backwardTrie.data[p].child[id[s[j]]]) {
                p = -1;
                break;
            }
            p = backwardTrie.data[p].child[id[s[j]]];
```

```
        }
        cout << (~p ? cnt[0][p] : 0) << '\n';
        continue;
    }

    if (s.back() == '*') {
        s.pop_back();
        int p = 0;
        for (int i = 0; i < s.size(); i++) {
            if (!forwardTrie.data[p].child[id[s[i]]]) {
                p = -1;
                break;
            }
            p = forwardTrie.data[p].child[id[s[i]]];
        }
        cout << (~p ? cnt[p][0] : 0) << '\n';
        continue;
    }

    // '*' in the middle
    int p = 0;
    for (int i = 0; i < s.size(); i++) {
        if (s[i] == '*') {
            break;
        }
        if (!forwardTrie.data[p].child[id[s[i]]]) {
            p = -1;
            break;
        }
        p = forwardTrie.data[p].child[id[s[i]]];
    }
    if (~p) {
        int p1 = 0;
        for (int j = s.size() - 1; j > 0; j--) {
            if (s[j] == '*') {
                break;
            }
            if (!backwardTrie.data[p1].child[id[s[j]]]) {
                p1 = -1;
                break;
            }
            p1 = backwardTrie.data[p1].child[id[s[j]]];
        }
        cout << (~p1 ? cnt[p][p1] : 0) << '\n';
    } else {
        cout << 0 << '\n';
    }
}
```