# Bài thực hành 4: Thực hành sử dụng các cấu trúc dữ liệu cơ bản để giải quyết các bài toán cụ thể

## Phần 1: Bài tập thực hành

### Bài tập 1: Đảo ngược một danh sách liên kết đơn

Hãy hoàn thiện các hàm thao tác trên một danh sách liên kết:

- Thêm một phần tử vào đầu danh sách liên kết
- In danh sách
- Đảo ngược danh sách liên kết (yêu cầu độ phức tạp thời gian O(N) và chi phí bộ nhớ dùng thêm O(1))

In [ ]:

```cpp
#include <iostream>

struct Node {
    int data;
    Node* next;

    Node(int data) {
        this->data = data;
        next = NULL;
    }
};

// push a new element to the beginning of the list
Node* prepend(Node* head, int data) {
    Node* temp = new Node(data);
    temp->next = head;
    return temp;
}

// print the list content on a line
void print(Node* head) {
    while (head) {
        std::cout << head->data << " ";
        head = head->next;
    }
    std::cout << std::endl;
}

// return the new head of the reversed list
Node* reverse(Node* head) {
    Node* prev = NULL;
    Node* next = NULL;
    while (head) {
        next = head->next;
        head->next = prev;
        prev = head;
        head = next;
    }
    return prev;
}

int main() {
    int n, u;
    std::cin >> n;
    Node* head = NULL;
    for (int i = 0; i < n; ++i){
        std::cin >> u;
        head = prepend(head, u);
    }

    std::cout << "Original list: ";
    print(head);

    head = reverse(head);

    std::cout << "Reversed list: ";
```

```
    print(head);

    return 0;
}
```

## Bài tập 2: Tính diện tích tam giác

Một điểm trong không gian 2 chiều được biểu diễn bằng std::pair. Hãy viết hàm tính diện tích tam giác theo tọa độ 3 đỉnh.

Tham khảo:

- http://www.cplusplus.com/reference/utility/pair/ (http://www.cplusplus.com/reference/utility/pair/)
- https://vi.wikipedia.org/wiki/Tam_gi%C3%A1c#C%C3%A1c_c%C3%B4ng_th%E1%BB%A9c_t%C3%ADnh_d (https://vi.wikipedia.org/wiki/Tam_gi%C3%A1c#C%C3%A1c_c%C3%B4ng_th%E1%BB%A9c_t%C3%ADnh_d

In [ ]:
```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <utility>

using Point = std::pair<double, double>;

double area(Point a, Point b, Point c) {
    return fabs(a.first * (b.second - c.second) +
        b.first * (c.second - a.second) +
        c.first * (a.second - b.second)) / 2;
}

int main() {
    std::cout << std::setprecision(2) << std::fixed;
    std::cout << area({1, 2}, {2.5, 10}, {15, -5.25}) << std::endl;
    return 0;
}
```

## Bài tập 3: Tính tích có hướng của 2 vector

Một vector trong không gian 3 chiều được biểu diễn bằng std::tuple<double, double, double>. Hãy viết hàm tính tích có hướng của 2 vector.

Tham khảo:

- http://www.cplusplus.com/reference/tuple/tuple/ (http://www.cplusplus.com/reference/tuple/tuple/)
- https://vi.wikipedia.org/wiki/T%C3%ADch_vect%C6%A1 (https://vi.wikipedia.org/wiki/T%C3%ADch_vect%C6%A1)

In [ ]:
```cpp
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;
using Vector = std::tuple<double, double, double>;

Vector cross_product(Vector a, Vector b) {
    return Vector(
            std::get<1>(a) * std::get<2>(b) - std::get<2>(a) * std::get<1>(b),
            std::get<0>(a) * std::get<2>(b) - std::get<2>(a) * std::get<0>(b),
            std::get<0>(a) * std::get<1>(b) - std::get<1>(a) * std::get<0>(b)
        );
}

int main() {
    cout << setprecision(2) << fixed;
    Vector a {1.2, 4, -0.5};
    Vector b {1.5, -2, 2.5};
    Vector c = cross_product(a, b);
    cout << get<0>(c) << ' ' << get<1>(c) << ' ' << get<2>(c) << endl;
    return 0;
}
```

## Bài tập 4: Thao tác với std::vector

Cho hai std::vector, hãy xóa hết các phần tử chẵn, sắp xếp giảm dần các số trong cả 2 vector và trộn lại thành một vector cũng được sắp xếp giảm dần.

Tham khảo:

- http://www.cplusplus.com/reference/vector/vector/ (http://www.cplusplus.com/reference/vector/vector/)
- http://www.cplusplus.com/reference/algorithm/remove_if/ (http://www.cplusplus.com/reference/algorithm/remove_if/)
- http://www.cplusplus.com/reference/algorithm/merge/ (http://www.cplusplus.com/reference/algorithm/merge/)
- http://www.cplusplus.com/reference/iterator/back_inserter/ (http://www.cplusplus.com/reference/iterator/back_inserter/)

In [ ]:

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

void print_vector(const std::vector<int> &a) {
    for (int v : a) std::cout << v << ' ';
    std::cout << std::endl;
}

void delete_even(std::vector<int> &a) {
    a.erase(std::remove_if(a.begin(), a.end(), [] (int x) { return x % 2 == 0;
}), a.end());
}

void sort_decrease(std::vector<int> &a) {
    std::sort(a.rbegin(), a.rend());
}

std::vector<int> merge_vectors(const std::vector<int> &a, const std::vector<in
t> &b) {
    std::vector<int> c;
    merge(a.begin(), a.end(), b.begin(), b.end(), std::back_inserter(c), std::
greater<int>());
    return c;
}

int main() {
    int m, n, u;
    std::vector<int> a, b;

    std::cin >> m >> n;
    for(int i = 0; i < m; i++){
        std:: cin >> u;
        a.push_back(u);
    }
    for(int i = 0; i < n; i++){
        std:: cin >> u;
        b.push_back(u);
    }

    delete_even(a);
    std::cout << "Odd elements of a: ";
    print_vector(a);

    delete_even(b);
    std::cout << "Odd elements of b: ";
    print_vector(b);

    sort_decrease(a);
    std::cout << "Decreasingly sorted a: ";
    print_vector(a);

    sort_decrease(b);
    std::cout << "Decreasingly sorted b: ";
    print_vector(b);
```

```
    std::vector<int> c = merge_vectors(a, b);
    std::cout << "Decreasingly sorted c: ";
    print_vector(c);

    return 0;
}
```

## Bài tập 5:

Thực hiện thuật toán DFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề.

Tham khảo: http://www.cplusplus.com/reference/stack/stack/ (http://www.cplusplus.com/reference/stack/stack/)

In [ ]:
```cpp
#include <vector>
#include <list>
#include <iostream>
#include <stack>

void dfs(std::vector< std::list<int> > adj) {
    std::stack<int> S;
    std::vector<bool> visited(adj.size());
    S.push(1);
    while (!S.empty()) {
        int u = S.top();
        if (!visited[u]) {
            visited[u] = true;
            std::cout << u << std::endl;
        }
        if (!adj[u].empty()){
            int v = adj[u].front();
            adj[u].pop_front();
            if (!visited[v]) {
                S.push(v);
            }
        } else {
            S.pop();
        }
    }
}

int main() {
    int n = 7;
    std::vector< std::list<int> > adj;
    adj.resize(n + 1);
    adj[1].push_back(2);
    adj[2].push_back(4);
    adj[1].push_back(3);
    adj[3].push_back(4);
    adj[3].push_back(5);
    adj[5].push_back(2);
    adj[2].push_back(7);
    adj[6].push_back(7);
    dfs(adj);

    return 0;
}
```

## Bài tập 6:

Thực hiện thuật toán BFS trên đồ thị biểu diễn bằng danh sách kề.

Tham khảo: http://www.cplusplus.com/reference/queue/queue/ (http://www.cplusplus.com/reference/queue/queue/)

In [ ]:
```cpp
#include <vector>
#include <list>
#include <iostream>
#include <queue>
using namespace std;

void bfs(std::vector< std::list<int> > adj) {
    std::queue<int> Q;
    std::vector<bool> was(adj.size());
    Q.push(1);
    was[1] = true;
    while (!Q.empty()) {
        int u = Q.front();
        Q.pop();
        std::cout << u << std::endl;
        for (int v : adj[u]) {
            if (!was[v]) {
                was[v] = true;
                Q.push(v);
            }
        }
    }
    std::cout << std::endl;
}

int main() {
    int n = 7;
    std::vector< std::list<int> > adj;
    adj.resize(n + 1);
    adj[1].push_back(2);
    adj[2].push_back(4);
    adj[1].push_back(3);
    adj[3].push_back(4);
    adj[3].push_back(5);
    adj[5].push_back(2);
    adj[2].push_back(7);
    adj[6].push_back(7);
    bfs(adj);

    return 0;
}
```

## Bài tập 7:

Thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng std::set

Tham khảo: http://www.cplusplus.com/reference/set/set/ (http://www.cplusplus.com/reference/set/set/)

In [ ]:

```cpp
#include <iostream>
#include <set>

template<class T>
std::set<T> set_union(const std::set<T> &a, const std::set<T> &b) {
    std::set<T> c = a;
    c.insert(b.begin(), b.end());
    return c;
}

template<class T>
std::set<T> set_intersection(const std::set<T> &a, const std::set<T> &b) {
    std::set<T> c;
    for (const T &x : a) {
        if (b.find(x) != b.end()) {
            c.insert(x);
        }
    }
    return c;
}

template<class T>
void print_set(const std::set<T> &a) {
    for (const T &x : a) {
        std::cout << x << ' ';
    }
    std::cout << std::endl;
}

int main() {
    std::set<int> a = {1, 2, 3, 5, 7};
    std::set<int> b = {2, 4, 5, 6, 9};
    std::set<int> c = set_union(a, b);
    std::set<int> d = set_intersection(a, b);

    std::cout << "Union: "; print_set(c);
    std::cout << "Intersection: "; print_set(d);

    return 0;
}
```

## Bài tập 8:

Thực hiện các phép giao và hợp của hai tập hợp mờ được biểu diễn bằng std::map.

Trong đó mỗi phần tử được gán cho một số thực trong đoạn [0..1] biểu thị độ thuộc của phần tử trong tập hợp, với độ thuộc bằng 1 nghĩa là phần tử chắc chắn thuộc vào tập hợp và ngược lại độ thuộc bằng 0 nghĩa là phần tử chắc chắn không thuộc trong tập hợp.

Phép giao và hợp của 2 tập hợp được thực hiện trên các cặp phần tử bằng nhau của 2 tập hợp, với độ thuộc mới được tính bằng phép toán min và max của hai độ thuộc.

Tham khảo:

- https://vi.wikipedia.org/wiki/T%E1%BA%ADp_m%E1%BB%9D (https://vi.wikipedia.org/wiki/T%E1%BA%ADp_m%E1%BB%9D)
- https://en.wikipedia.org/wiki/Fuzzy_set_operations (https://en.wikipedia.org/wiki/Fuzzy_set_operations)
- http://www.cplusplus.com/reference/map/map/ (http://www.cplusplus.com/reference/map/map/)

In [ ]:

```cpp
#include <iostream>
#include <map>

using namespace std;

template<class T>
std::map<T, double> fuzzy_set_union(const std::map<T, double> &a, const std::m
ap<T, double> &b) {
    std::map<T, double> c = a;
    for (const auto &e : b) {
        if (c.count(e.first)) {
            c[e.first] = max(c[e.first], e.second);
        } else {
            c.insert(e);
        }
    }
    return c;
}

template<class T>
std::map<T, double> fuzzy_set_intersection(const std::map<T, double> &a, const
std::map<T, double> &b) {
    std::map<T, double> c;
    for (const auto &x : a) {
        const auto it = b.find(x.first);
        if (it != b.end()) {
            c[x.first] = min(x.second, it->second);
        }
    }
    return c;
}

template<class T>
void print_fuzzy_set(const std::map<T, double> &a) {
    cout << "{ ";
    for (const auto &x : a) {
        std::cout << "(" << x.first << ", " << x.second << ") ";
    }
    cout << "}";
    std::cout << std::endl;
}

int main() {
    std::map<int, double> a = {{1, 0.2}, {2, 0.5}, {3, 1}, {4, 0.6}, {5, 0.7
}};
    std::map<int, double> b = {{1, 0.5}, {2, 0.4}, {4, 0.9}, {5, 0.4}, {6, 1
}};
    std::cout << "A = "; print_fuzzy_set(a);
    std::cout << "B = "; print_fuzzy_set(b);
    std::map<int, double> c = fuzzy_set_union(a, b);
    std::map<int, double> d = fuzzy_set_intersection(a, b);
    std::cout << "Union: "; print_fuzzy_set(c);
    std::cout << "Intersection: "; print_fuzzy_set(d);
}
```

## Bài tập 9:

Cài đặt thuật toán Dijkstra trên đồ thị vô hướng được biểu diễn bằng danh sách kề sử dụng std::priority_queue

Tham khảo: http://www.cplusplus.com/reference/queue/priority_queue/
(http://www.cplusplus.com/reference/queue/priority_queue/)

In [ ]:

```cpp
#include <iostream>
#include <queue>
#include <climits>

std::vector<int> dijkstra(const std::vector< std::vector< std::pair<int, int>
> >&adj) {
    // hàng đợi ưu tiên lưu các cặp (-khoảng cách từ 1, đỉnh)
    std::priority_queue< std::pair<int, int> > Q;
    std::vector<int> d(adj.size(), INT_MAX);
    d[0] = 0;
    Q.push({0, 0});
    while (!Q.empty()) {
        int du = -Q.top().first;
        int u = Q.top().second;
        Q.pop();
        if (du != d[u]) continue;
        for (auto e : adj[u]) {
            int v = e.first;
            int c = e.second;
            if (d[v] > d[u] + c) {
                d[v] = d[u] + c;
                Q.push({-d[v], v});
            }
        }
    }
    return d;
}

int main() {
    int n = 9;
    std::vector< std::vector< std::pair<int, int> > > adj(n);
    auto add_edge = [&adj] (int u, int v, int w) {
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    };

    add_edge(0, 1, 4);
    add_edge(0, 7, 8);
    add_edge(1, 7, 11);
    add_edge(1, 2, 8);
    add_edge(2, 3, 7);
    add_edge(2, 8, 2);
    add_edge(3, 4, 9);
    add_edge(3, 5, 14);
    add_edge(4, 5, 10);
    add_edge(5, 6, 2);
    add_edge(6, 7, 1);
    add_edge(6, 8, 6);
    add_edge(7, 8, 7);

    std::vector<int> distance = dijkstra(adj);
    for (int i = 0; i < distance.size(); ++i) {
        std::cout << "distance " << 0 << "->" << i << " = " << distance[i] <<
std::endl;
    }
```

```
    return 0;
}
```

## Phần 2: Bài tập về nhà

## Bài tập 10: Search Engine

Xây dựng một máy tìm kiếm (search engine) đơn giản.

Cho $N$ văn bản và $Q$ truy vấn. Với mỗi truy vấn, cần trả về văn bản khớp với truy vấn đó nhất.

Sử dụng phương pháp tính điểm TF-IDF:

- $f(t, d)$ là số lần xuất hiện của từ $t$ trong văn bản $d$
- $max_f(d)$ là giá trị lớn nhất của $f(t, d)$ với mọi $t$
- $df(t)$ là số văn bản chứa từ $t$
- $TF(t, d) = 0.5 + 0.5 \cdot \frac{f(t,d)}{max_f(t,d)}$
- $IDF(t) = log_2(\frac{N}{df(t)})$
- Điểm số của từ $t$ trong văn bản $d$ là $score(t, d) = TF(t, d) \cdot IDF(t)$, nếu từ $t$ không xuất hiện trong văn bản $d$ thì $score(t, d) = 0$.
- Điểm số của văn bản $d$ đối với truy vấn gồm các từ (có thể trùng nhau) $t_1, t_2, \ldots, t_q$ là $\sum_{i=1}^{q} score(t_i, d)$

Ta coi văn bản có điểm số càng cao thì càng khớp với truy vấn.

**Input:**

- Dòng đầu tiên chứa số $N$
- Dòng thứ $i$ trong $N$ dòng tiếp theo thể hiện văn bản $i$, mỗi dòng là một dãy các từ ngăn cách nhau bởi dấu phẩy
- Dòng tiếp theo chứa số $Q$
- Dòng thứ $i$ trong $Q$ dòng tiếp theo thể hiện truy vấn thứ $i$, mỗi dòng là một dãy các từ ngăn cách nhau bởi dấu phẩy

**Output:** Gồm $Q$ dòng, dòng thứ $i$ là chỉ số của văn bản khớp với truy vấn thứ $i$ nhất. Nếu có nhiều văn bản có điểm số bằng nhau, in ra văn bản có chỉ số nhỏ nhất.

**Ví dụ:**

Input:

```
5
k,k,ow
bb,ar,h
qs,qs,qs
d,bb,q,d,rj
ow
5
h,d,d,qs,q,q,ar
qs,qs
hc,d,ow,d,qs
ow,wl,hc,k
q,hc,q,d,hc,q
```

Output:

4
3
4
1
4

**Giới hạn:**

- $N \leq 1000$
- $Q \leq 1000$
- Số từ trong mỗi văn bản không quá $1000$
- Số từ trong mỗi truy vấn không quá $10$
- Độ dài mỗi từ không quá $10$

**Tham khảo:**

- https://en.wikipedia.org/wiki/Tf%E2%80%93idf (https://en.wikipedia.org/wiki/Tf%E2%80%93idf)