**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 [ ]:

#include *<iostream>*

**using** **namespace** std;

**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) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

*// print the list content on a line*

void print(Node\* head) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

*// return the new head of the reversed list*

Node\* reverse(Node\* head) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

int main() {

int n, u;

cin >> n;

Node\* head = NULL;

**for** (int i = 0; i < n; ++i){

cin >> u;

head = prepend(head, u);

}

cout << "Original list: ";

print(head);

head = reverse(head);

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 pair. Hãy viết hàm double area(Point a, Point b, Point c) tính diện tích tam giác theo tọa độ 3 đỉnh. Trong đó, Point là kiểu được định nghĩa sẵn trong trình chấm như sau: using Point = pair<double, double>;

Tham khảo:

* <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_di%E1%BB%87n_t%C3%ADch_tam_gi%C3%A1c>

In [ ]:

*// #include <iostream>*

*// #include <cmath>*

*// #include <iomanip>*

*// #include <utility>*

*// using namespace std;*

*// using Point = pair<double, double>;*

double area(Point a, Point b, Point c) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

*// int main() {*

*// cout << setprecision(2) << fixed;*

*// cout << area({1, 2}, {2.5, 10}, {15, -5.25}) << endl;*

*// return 0;*

*// }*

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

Một vector trong không gian 3 chiều được biểu diễn bằng tuple<double, double, double>. Hãy viết hàm Vector cross\_product(Vector a, Vector b) tính tích có hướng của 2 vector. Trong đó Vector là kiểu dữ liệu được định nghĩa sẵn trong trình chấm như sau: using Vector = tuple<double, double, double>;

Tham khảo:

* <http://www.cplusplus.com/reference/tuple/tuple/>
* <https://vi.wikipedia.org/wiki/T%C3%ADch_vect%C6%A1>

In [ ]:

*// #include <iostream>*

*// #include <cmath>*

*// #include <iomanip>*

*// using namespace std;*

*// using Vector = tuple<double, double, double>;*

Vector cross\_product(Vector a, Vector b) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

*// 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 vector**

Cho hai 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/algorithm/remove_if/>
* <http://www.cplusplus.com/reference/algorithm/merge/>
* <http://www.cplusplus.com/reference/iterator/back_inserter/>

In [ ]:

#include *<iostream>*

#include *<vector>*

#include *<algorithm>*

**using** **namespace** std;

void print\_vector(**const** vector<int> &a) {

**for** (int v : a) cout << v << ' ';

cout << endl;

}

void delete\_even(vector<int> &a) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

void sort\_decrease(vector<int> &a) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

vector<int> merge\_vectors(**const** vector<int> &a, **const** vector<int> &b) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

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);

cout << "Odd elements of a: ";

print\_vector(a);

delete\_even(b);

cout << "Odd elements of b: ";

print\_vector(b);

sort\_decrease(a);

cout << "Decreasingly sorted a: ";

print\_vector(a);

sort\_decrease(b);

cout << "Decreasingly sorted b: ";

print\_vector(b);

vector<int> c = merge\_vectors(a, b);

cout << "Decreasingly sorted c: ";

print\_vector(c);

**return** 0;

}

**Bài tập 5:**

Viết hàm thực hiện thuật toán DFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề vector< list<int> > . Đồ thị có n đỉnh được đánh số từ 1 đến n. Thuật toán DFS xuất phát từ đỉnh 1. Các đỉnh được thăm theo thứ tự ưu tiên từ trái sang phải trong danh sách kề. Yêu cầu hàm trả ra thứ tự các đỉnh được thăm (những đỉnh không thể thăm từ đỉnh 1 thì không phải in ra).

Tham khảo: <http://www.cplusplus.com/reference/stack/stack/>

In [ ]:

void dfs(vector< list<int> > adj) {

stack<int> S;

vector<bool> visited(adj.size());

S.push(1); *// Bắt đầu từ đỉnh số 1*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

**Bài tập 6:**

Viết hàm thực hiện thuật toán BFS không sử dụng đệ quy trên đồ thị biểu diễn bằng danh sách kề vector< list<int> > . Đồ thị có n đỉnh được đánh số từ 1 đến n. Thuật toán BFS xuất phát từ đỉnh 1. Các đỉnh được thăm theo thứ tự ưu tiên từ trái sang phải trong danh sách kề. Yêu cầu hàm trả ra thứ tự các đỉnh được thăm (những đỉnh không thể thăm từ đỉnh 1 thì không phải in ra).

Tham khảo: <http://www.cplusplus.com/reference/queue/queue/>

In [ ]:

void bfs(vector< list<int> > adj) {

queue<int> Q;

vector<bool> visited(adj.size());

Q.push(1); *// Bắt đầu từ đỉnh số 1*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

**Bài tập 7:**

Viết các hàm thực hiện các phép giao và hợp của hai tập hợp được biểu diễn bằng set.

Tham khảo: <http://www.cplusplus.com/reference/set/set/>

In [ ]:

*// #include <iostream>*

*// #include <set>*

*// using namespace std;*

**template**<**class** **T**>

set<T> set\_union(**const** set<T> &a, **const** set<T> &b) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

**template**<**class** **T**>

set<T> set\_intersection(**const** set<T> &a, **const** set<T> &b) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

*// 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:**

Viết các hàm thực hiện các phép giao và hợp của hai tập hợp mờ được biểu diễn bằng 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://en.wikipedia.org/wiki/Fuzzy_set_operations>
* <http://www.cplusplus.com/reference/map/map/>

In [ ]:

*// #include <iostream>*

*// #include <map>*

*// using namespace std;*

**template**<**class** **T**>

map<T, double> fuzzy\_set\_union(**const** map<T, double> &a, **const** map<T, double> &b) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

**template**<**class** **T**>

map<T, double> fuzzy\_set\_intersection(**const** map<T, double> &a, **const** map<T, double> &b) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

*// 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 priority\_queue Cụ thể, bạn cần cài đặt hàm vector<int> dijkstra(const vector< vector< pair<int, int> > >&adj) nhận đầu vào là danh sách kề chứa các cặp pair<int, int> biểu diễn đỉnh kề và trọng số tương ứng của cạnh. Đồ thị gồm n đỉnh được đánh số từ 0 tới n-1. Hàm cần trả vector<int> chứa n phần tử lần lượt là khoảng cách đường đi ngắn nhất từ đỉnh 0 tới các đỉnh 0, 1, 2, ..., n-1.

Tham khảo: <http://www.cplusplus.com/reference/queue/priority_queue/>

In [ ]:

*// #include <iostream>*

*// #include <queue>*

*// #include <climits>*

*// using namespace std;*

vector<int> dijkstra(**const** vector< vector< pair<int, int> > >&adj) {

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# YOUR CODE HERE #*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

}

*// int main() {*

*// int n = 9;*

*// vector< vector< 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);*

*// vector<int> distance = dijkstra(adj);*

*// for (int i = 0; i < distance.size(); ++i) {*

*// cout << "distance " << 0 << "->" << i << " = " << distance[i] << 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 NN văn bản và QQ 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)f(t,d) là số lần xuất hiện của từ tt trong văn bản dd
* maxf(d)maxf(d) là giá trị lớn nhất của f(t,d)f(t,d) với mọi tt
* df(t)df(t) là số văn bản chứa từ tt
* TF(t,d)=0.5+0.5⋅f(t,d)maxf(t,d)TF(t,d)=0.5+0.5⋅f(t,d)maxf(t,d)
* IDF(t)=log2(Ndf(t))IDF(t)=log2(Ndf(t))
* Điểm số của từ tt trong văn bản dd là score(t,d)=TF(t,d)⋅IDF(t)score(t,d)=TF(t,d)⋅IDF(t), nếu từ tt không xuất hiện trong văn bản dd thì score(t,d)=0score(t,d)=0.
* Điểm số của văn bản dd đối với truy vấn gồm các từ (có thể trùng nhau) t1,t2,...,tqt1,t2,...,tq là ∑qi=1score(ti,d)∑i=1qscore(ti,d)

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

**Input:**

* Dòng đầu tiên chứa số NN
* Dòng thứ ii trong NN dòng tiếp theo thể hiện văn bản ii, mỗi dòng là một dãy các từ ngăn cách nhau bởi dấu phẩy
* Dòng tiếp theo chứa số QQ
* Dòng thứ ii trong QQ dòng tiếp theo thể hiện truy vấn thứ ii, mỗi dòng là một dãy các từ ngăn cách nhau bởi dấu phẩy

**Output:** Gồm QQ dòng, dòng thứ ii là chỉ số của văn bản khớp với truy vấn thứ ii 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≤1000N≤1000
* Q≤1000Q≤1000
* Số từ trong mỗi văn bản không quá 10001000
* Số từ trong mỗi truy vấn không quá 1010
* Độ dài mỗi từ không quá 1010

**Tham khảo:**

* <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>

**Bài tập 11. Bảo vệ lâu đài**

Bức tường bao quanh một lâu đài nọ được cấu thành từ nn đoạn tường được đánh số từ 1 đến nn. Quân giặc lên kế hoạch tấn công lâu đài bằng cách gửi aiai tên giặc đánh vào đoạn tường thứ ii. Để bảo vệ lâu đài có tất cả ss lính.

Do các đoạn tường có chất lượng khác nhau nên khả năng bảo vệ tại các đoạn tường cũng khác nhau. Cụ thể tại đoạn tường thứ ii, mỗi lính có thể đẩy lùi tấn công của kiki tên giặc.

Giả sử đoạn tường thứ ii có xixi lính. Khi đó nếu số tên giặc không vượt quá xi×kixi×ki thì không có tên giặc nào lọt vào được qua đoạn tường này. Ngược lại sẽ có ai−xi×kiai−xi×ki tên giặc lọt vào lâu đài qua đoạn tường này.

Yêu cầu hãy viết chương trình phân bố lính đứng ở các đoạn tường sao cho tổng số lính là ss và tổng số lượng tên giặc lọt vào lâu đài là nhỏ nhất.

**Dữ liệu vào:**

Dòng thứ nhất chứa các số nguyên nn và ss (1≤n≤100000;1≤s≤1091≤n≤100000;1≤s≤109).

nn dòng tiếp theo chứa hai số nguyên aiai và kiki lần lượt là số tên giặc tấn công đoạn tường thứ ii và khả năng chống trả của một lính ở đoạn tường thứ ii (1≤ai,ki≤1091≤ai,ki≤109).

**Kết quả:**

Ghi ra một số nguyên duy nhất là số lượng tên giặc tối thiểu có thể lọt vào lâu đài.

**Ví dụ:**

Dữ liệu vào:

3 3  
4 2  
1 1  
10 8

Kết quả:

3

**Bài tập 12. Lược đồ**

Cho một lược đồ gồm nn cột chữ nhật liên tiếp nhau có chiều rộng bằng 1 và chiều cao lần lượt là các số nguyên không âm h1,h2,…,hnh1,h2,…,hn. Hãy xác định hình chữ nhật có diện tích lớn nhất có thể tạo thành từ các cột liên tiếp.

**Dữ liệu vào:**  
Dòng thứ nhất chứa số nguyên dương nn (1≤n≤1061≤n≤106).  
Dòng thứ hai chứa nn số nguyên không âm h1,h2,…,hnh1,h2,…,hn cách nhau bởi dấu cách (0≤hi≤1090≤hi≤109).

**Kết quả:**  
In ra số nguyên duy nhất là diện tích hình chữ nhật lớn nhất có thể tạo thành từ các cột liên tiếp của lược đồ.

**Ví dụ:**  
Dữ liệu vào:  
7  
6 2 5 4 5 1 6

Kết quả:  
12



**Bài tập 13: Đếm xâu con**

Cho một xâu nhị phân độ dài nn. Hãy viết chương trình đếm số lượng xâu con chứa số ký tự 0 và số ký tự 1 bằng nhau.

**Dữ liệu vào:**  
Một dòng duy nhất chứa một xâu nhị phân độ dài nn (1≤n≤1061≤n≤106).

**Kết quả:**  
Ghi ra một số nguyên duy nhất là số lượng xâu con có số ký tự 0 và số ký tự 1 bằng nhau.

**Ví dụ:**

Dữ liệu vào:  
1001011

Kết quả:  
8

Giải thích: Trong ví dụ trên có 8 xâu con thỏa mãn là 10, 01, 10, 01, 1001, 0101, 100101, 001011