**App01: 2dna.\***

**Vấn đề:** Cho 2 chuỗi ADN (A, B). Kiểm tra xem chuỗi ADN dài nhất trong B mà cũng nằm trong A.

**Ý tưởng:**

* Sử dụng mảng băm để kiểm tra sự giống nhau của 2 chuỗi ký tự
* Cây tiềm kiếm nhị phân câng bằng để tiềm kiếm kết quả (chiều dài)

**Code:**

a) Hằng số:

Base **=** 5**,** M **=** 1e9 **+** 7

b) Lớp hash chứa các hàm cho việc băm chuỗi ADN:

**Hash\_DNA(string&** str**)** **{**

m\_size **=** str**.**size**();**

hash **=** **new** int**[**str**.**size**()** **+** 5**];**

p **=** **new** int**[**str**.**size**()** **+** 5**];**

p**[**0**]** **=** 1**;**

hash**[**0**]** **=** func**(**str**[**0**]);**

**for** **(**int i **=** 1**;** i **<** m\_size**;** **++**i**)** **{**

p**[**i**]** **=** 1LL **\*** p**[**i **-** 1**]** **\*** Base **%** M**;**

hash**[**i**]** **=** **(**1LL **\*** hash**[**i **-** 1**]** **\*** Base **+** func**(**str**[**i**]))** **%**M**;**

**}**

**}**

p**[**i**]** : mảng lũy thừa của Base (p**[**i**] =** Base**\*\***i)

hash**[**i**]** : mảng băm của chuỗi ADN từ đầu chuỗi đến ký tự thứ i

(vd: ACTGC, hash**[**2**]** = A\*52 + C\*5 + T, hash**[**3**]** = A\*53 + C\*52 + T\*5 + G)

c) Hàm lấy giá trị băm từ ký tự thứ l đến ký tự thứ r trong mảng băm:

int get**(**int l**,** int r**)** **{**

**if** **(**l **==** 0**)** **return** hash**[**r**];**

**return** **(**1LL **\*** M **\*** M **+** hash**[**r**]** **-** 1LL **\*** hash**[**l **-** 1**]** **\*** p**[**r **-** l **+** 1**])** **%** M**;**

**}**

vd: ACTGC, hash**[**1**]** = A\*5 + C, hash**[**3**]** = A\*53 + C\*52 + T\*5 + G

hash(2, 3): hash**[**3**] -** hash**[**1**]\***p**[**2**]<=>**(A\*53 + C\*52 + T\*5 + G) – (A\*5 + C)\*52 = T\*5 + G

d) Hàm kiểm tra với chiều dài len có tồn tại 1 chuỗi ADN cùng thuộc A và B không:

bool check**(Hash\_DNA&** h1**,** **Hash\_DNA&** h2**,** int len**)** **{**

map **<**int**,** int**>** mp**;**

**for** **(**int i **=** 0**;** i **+** len **-** 1 **<** h1**.**size**();** **++**i**)** **{**

int hash **=** h1**.**get**(**i**,** i **+** len **-** 1**);**

mp**[**hash**]** **=** i**;**

**}**

**for** **(**int i **=** 0**;** i **+** len **-** 1 **<** h2**.**size**();** **++**i**)** **{**

int hash **=** h2**.**get**(**i**,** i **+** len **-** 1**);**

**if** **(**mp**.**find**(**hash**)** **!=** mp**.**end**()) return** **true;**

**}**

**return** **false;**

**}**

map **<**int**,** int**>** mp: Cây tìm kiếm nhị phân câng bằn (std::map)

e) Tìm kiếm nhị phân chiều dài (len):

int l **=** 0**,** r **=** h2**.**size**()** **+** 1**;**

**while** **(**r **>** l **+** 1**)** **{**

int mid **=** **(**l **+** r**)** **>>** 1**;**

**if** **(**check**(**h1**,** h2**,** mid**))** l **=** mid**;**

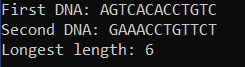
**else** r **=** mid**;**

**}**

**Độ phức tạp:**

* Băm 2 chuỗi : O(n) (n là chiều dài của 2 chuỗi A, B)
* Kiểm tra 2 chuỗi có giống nhau : O(1)
* Tìm kiếm độ dài : O(log2n)
* Kiểm tra tất cả các chuỗi của B có nằm trong A : O(nlog2n)
* Độ phức tạp : O(nlog22n)

**Chạy thử:**

****

Giải thích: Chuỗi dài nhất: ACCTGT

**App02: line-up.\***

**Vấn đề:** Cho 1 hàng người với độ cao khác nhau (1->300cm), hỏi xem trước người thứ i có bao nhiều người cao hơn người đó (người thứ 0 là người đứng đầu hàng)

**Ý tưởng:**

* Sử dụng cấu trúc dữ liệu cây nhị phân phân đoạn (Interval Tree - IT)
  + 1 Node sẽ quản lý số người trong khoảng [a, b]
  + 1 Node sẽ có 2 node con quản lý lần lượt các đoạn [a, mid] và [mid + 1, b] (với mid là trung vị của a và b)
  + Node quản lý đoạn [a, a] là node lá và sẽ là N nếu có N người có độ cao là a
* Biểu diễn cây nhị phân bằng mảng:
  + Node gốc có id = 1
  + Id Node con của Node id = a là 2 \* a và 2 \* a + 1 (vd: Node 4 có 2 node con là 8 và 9)
* Tại người thứ i, dùng cây IT để tính số người có độ cao trong khoảng [ai, max\_height] (ai là chiều cao của người thứ i)

**Code:**

a) Hằng số:

maxHeight **=** 300

b) Hàm tìm kiếm số người trong khoảng [s, f]:

int get**(**int s**,** int f**,** int l**,** int r**,** int id**)** **{**

**if** **(**l **>** f **||** r **<** s**)** **return** 0**;**

**if** **(**l **>=** s **&&** r **<=** f**)** **return** data**[**id**];**

int mid **=** **(**l **+** r**)** **>>** 1**;**

**return** get**(**s**,** f**,** l**,** mid**,** 2 **\*** id**)** **+** get**(**s**,** f**,** mid **+** 1**,** r**,** 2 **\*** id **+** 1**);**

**}**

c) Hàm cập nhật người có độ cao x vào cây:

void update**(**int x**,** int l**,** int r**,** int id**)** **{**

**if** **(**l **>** x **||** r **<** x**)** **return;**

**if** **(**l **>=** x **&&** r **<=** x**)** **{**

data**[**id**]++;** **return;**

**}**

int mid **=** **(**l **+** r**)** **>>** 1**;**

update**(**x**,** l**,** mid**,** 2 **\*** id**);**

update**(**x**,** mid **+** 1**,** r**,** 2 **\*** id **+** 1**);**

data**[**id**]** **=** data**[**2 **\*** id**]** **+** data**[**2 **\*** id **+** 1**];**

**}**

d) Tại người thứ i, lấy số người cao hơn trong khoảng [ai + 1, max\_height], sau có cập nhật người đó vào cây:

cout **<<** "The number of people standing in front of P is higher than P.\n"**;**

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

cout **<<** i **+** 1 **<<** ": "**;**

cout **<<** tree**.**get**(**height**[**i**]** **+** 1**,** maxHeight**,** 1**,** maxHeight**,** 1**)** **<<** "\n"**;**

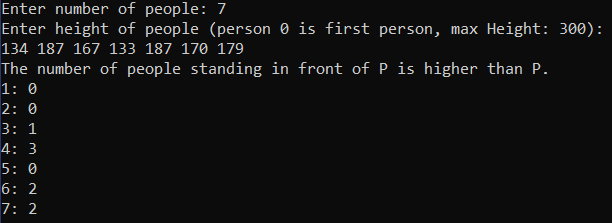
tree**.**update**(**height**[**i**],** 1**,** maxHeight**,** 1**);**

**}**

**Độ phức tạp:**

* Tìm kiếm trên cây IT : O(log2n)
* Cập nhật cây IT : O(log2n)
* Duyệt độ cao từng người : O(n)
* Độ phức tạp : O(nlog2n)

**Chạy thử:**



Giải thích: độ cao của người đứng trước i cao hơn i là

3: 187

4: 134 187 167

6: 187 187

7: 187 187

**App03: power.\***

**Vấn đề:** Một trường học có 1 danh sách các học sinh, các học sinh này đều có 1 chỉ số sức mạnh về học tập (poweri) khác nhau. Tuy nhiên để tránh việc trường cho 1 đội toàn học sinh ưu tú đi thi, cuộc thi yêu cầu chọn ra 1 dãy liên tục các học sinh mà tổng chỉ số sức mạnh không vượt quá ngưỡng cho phép M. Hỏi trường này có bao nhiêu cách chọn dãy liên tục học sinh được phép tham gia cuộc thi.

**Ý tưởng:**

* Sử dụng cấu trúc dữ liệu queue
  + Chứa danh sách các học sinh liên tục
* Duyệt danh sách từ trên xuống (1->n) với học sinh thứ i, pop ra khỏi queue các học sinh khi mà tổng sức mạnh của các học sinh trong queue + poweri vượt quá M
* Sẽ có queue.size() dãy học sinh thỏa yêu cầu của cuộc thi với học sinh i là học sinh cuối cùng

**Code:**

a) Các hàm của queue:

bool empty**()** **{** **return** l **>** r**;** **}**

int size**()** **{** **return** r **-** l **+** 1**;** **}**

int top**()** **{** **return** data**[**l**];** **}**

void pop**()** **{** l**++;** **}**

void push**(**int value**)** **{** data**[++**r**]** **=** value**;** **}**

b) Tính tổng sức mạnh của các học sinh:

**for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** **{**

cout **<<** "Power of student " **<<** i **<<** ": "**;**

cin **>>** power**[**i**];**

power**[**i**]** **+=** power**[**i **-** 1**];**

**}**

power**[**i**]** : là tổng sức mạnh từ học sinh đầu tiên (1) tới học sinh thứ i

c) Đếm số dãy thỏa mãn yêu cầu:

**for** **(**int i **=** n**;** i **>** 0**;** **--**i**)** **{**

que**.**push**(**i**);**

**while** **(!**que**.**empty**()** **&&** power**[**que**.**top**()]** **-** power**[**i **-** 1**]** **>** M**)**

que**.**pop**();**

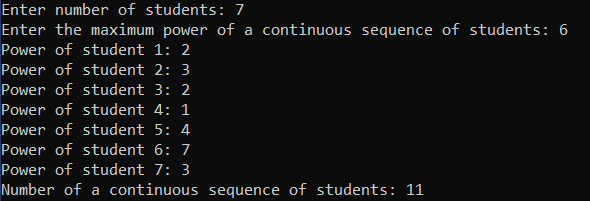
Result **+=** que**.**size**();**

**}**

**Độ phức tạp:**

* Tổng chi phí push và pop của queue : O(n)
* Duyệt sức mạnh từng học sinh : O(n)
* Độ phức tạp : O(n)

**Chạy thử:**



Giải thích: học sinh tham gia và tổng sức mạnh của nhóm

1. 1 (2)

2. 2 (3)

3. 1 2 (5)

4. 3 (2)

5. 2 3 (5)

6. 4 (1)

7. 3 4 (3)

8. 2 3 4 (6)

9. 5 (4)

10. 4 5 (5)

11. 7 (3)

**App04: recycle.\***

**Vấn đề:** Có một hàng rào gỗ là các ván gỗ có cùng chiều rộng, được đóng chặt vào nhau. Do thời gian, các ván gỗ này không còn nguyên vẹn (phần trên các ván gỗ bị nứt mẻ hoặc gãy mất). Người chủ muốn tái chế lại hàng rào này bằng cách tìm một hình chữ nhật có diện tích lớn nhất cho 1 tác phẩm nghệ thuật của ông (Độ cao của các tấm ván là độ cao sau khi được gọt phẳng các vết nứt bên trên)

**Ý tưởng:**

* Với mỗi tấm ván, ta cố định chiều cao theo chiều cao của tấm ván đó, sau đó tìm diện tích 2 bên trái và phải của nó.
* Sử dụng cấu trúc dữ liệu stack, chứ vị trí tấm ván gần nhất 1 phía (trái hoặc phải) mà tấm ván đó có độ cao nhỏ hơn độ cao tấm ván đang xét
* Duyệt 2 lần với mỗi tấm ván thứ i, ta có được ví trí 2 tấm ván như trên ở 2 phía (trái và phải), diện tích tối đa tấm ván này đạt được là khoảng cách 2 tấm ván nhân cho độ cao tấm ván thứ i

**Code:**

a) Các hàm của stack:

bool empty**()** **{ return** m\_top **==** **-**1**;** **}**

int top**()** **{** **return** data**[**m\_top**];** **}**

void clear**()** **{** m\_top **=** **-**1**;** **}**

void pop**()** **{** m\_top**--;** **}**

void push**(**int value**)** **{** data**[++**m\_top**]** **=** value**;** **}**

b) Tìm ván gỗ gần nhất (trái) có độ cao bé hơn ván gỗ thứ i (gọi ván gỗ 0 là ảo có độ cao = -1):

s**.**push**(**0**);**

height**[**0**]** **=** **-**1**;**

**for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** **{**

**while** **(!**s**.**empty**()** **and** height**[**s**.**top**()]** **>=** height**[**i**])** s**.**pop**();**

lenght**[**i**]** **=** s**.**top**()** **+** 1**;**

s**.**push**(**i**);**

**}**

c) Tìm ván gỗ gần nhất (phải) có độ cao bé hơn ván gỗ thứ i (gọi ván gỗ n + 1 là ảo có độ cao = -1):

s**.**push**(**n **+** 1**);**

height**[**n **+** 1**]** **=** **-**1**;**

**for** **(**int i **=** n**;** i **>=** 1**;** **--**i**)** **{**

**while** **(!**s**.**empty**()** **and** height**[**s**.**top**()]** **>=** height**[**i**])** s**.**pop**();**

long plank\_lenght **=** s**.**top**()** **-** lenght**[**i**];**

**if** **(**maxArea **<** plank\_lenght **\*** height**[**i**])** **{**

maxArea **=** plank\_lenght **\*** height**[**i**];**

l **=** lenght**[**i**];**

r **=** s**.**top**()** **-** 1**;**

h **=** height**[**i**];**

**}**

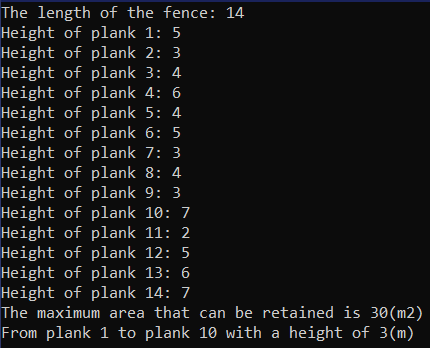
s**.**push**(**i**);**

**}**

**Độ phức tạp:**

* Tìm kiếm ván gỗ bên trái : O(n)
* Tìm kiếm ván gỗ bên phải : O(n)
* Duyệt độ cao từng ván gỗ : O(n)
* Độ phức tạp : O(n)

**Chạy thử:**



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**App05: service-network.\***

**Vấn đề:** Có một mạng cáp quang kết nối các khu dân cư, công ty, nhà dịch vụ mạng, ... (thành phần mạng). Có một số nhà dịch vụ mạng cung cấp dịch A hoặc B hoặc đồng thời cả 2. Các kết nối là các đường cáp nối trực tiếp 2 thành phần mạng. Hỏi cáp kết nối nào là quan trọng cần nâng cao bảo trì (cáp quan trọng khi mà nó bị sự cố, sẽ có 1 số thành phần mạng không tiếp cận được với dịch vụ A hoặc B)

**Ý tưởng:**

* Mô hình hóa hệ thống mạng thành đồ thị với V là đại diện cho 1 thành phần mạng, cạng (U, V) là cáp nối trực tiếp giữa U và V.
* Sử dụng thuật toán Tarjan để tìm cầu trong đồ thị (cầu là cạnh mà khi bỏ nó đi sẽ tăng số thành phần liên thông của đồ thị)
* Với mỗi cầu tìm được, tính xem 1 phía của cầu nếu không có bất cứ nhà cung cấp dịch vụ A/B nào hoặc có tối đa số nhà cung cấp dịch vụ A/B (tương đương với phía còn lại là 0 nhà dịch vụ) thì cầu này là một kết nối quan trọng

**Code:**

a) Lưu đồ thị dạng danh sách kề:

**for** **(**int i **=** 1**,** u**,** v**;** i **<=** m**;** **++**i**)** **{**

cout **<<** "Enter edge " **<<** i **<<** " (u, v): "**;**

cin **>>** u **>>** v**;**

a**[**u**].**push\_back**({** v**,** cnt**++** **});**

a**[**v**].**push\_back**({** u**,** cnt**++** **});**

**}**

cnt: đại diện cho id 2 hướng của cạnh u,v (dùng để block cạnh còn lại khi 1 cạnh đã được duyệt)

b) Thuật toán tarjan tìm cạnh cầu và xử lý cạnh có phải là cạnh quan trọng:

void DFS**(**int u**)** **{**

Num**[**u**]** **=** **++**cnt**;** Lowc**[**u**]** **=** n **+** 1**;**

**for** **(**pii**&** i **:** a**[**u**])** **{**

**if** **(!**fg**[**i**.**second**])** **{**

int v **=** i**.**first**;**

//edge i.second and i.second^1 is 2 two-ways of 1 edge

fg**[**i**.**second **^** 1**]** **=** 1**;**

**if** **(**Num**[**v**])**

// tarjan algorithm

Lowc**[**u**]** **=** min**(**Lowc**[**u**],** Num**[**v**]);**

**else** **{**

DFS**(**v**);**

// tarjan algorithm

Lowc**[**u**]** **=** min**(**Lowc**[**u**],** Lowc**[**v**]);**

A**[**u**]** **+=** A**[**v**];**

B**[**u**]** **+=** B**[**v**];**

**if** **(**Lowc**[**v**]** **>=** Num**[**v**])**

//bridge has been found

**if** **(!**A**[**v**]** **||** A**[**v**]** **==** K **||** **!**B**[**v**]** **||** B**[**v**]** **==** L**)**

edge**.**push\_back**({** u**,** v **});**

**}**

**}**

**}**

**}**

void Tarjan**()** **{**

//Init

cnt **=** 0**;**

fg **=** Init**<**bool**>(**2 **\*** m **+** 5**);**

Num **=** Init**<**int**>(**n **+** 5**);**

Lowc **=** Init**<**int**>(**n **+** 5**);**

DFS**(**1**);**

//Release

**delete[]** fg**;**

**delete[]** Num**;**

**delete[]** Lowc**;**

**}**

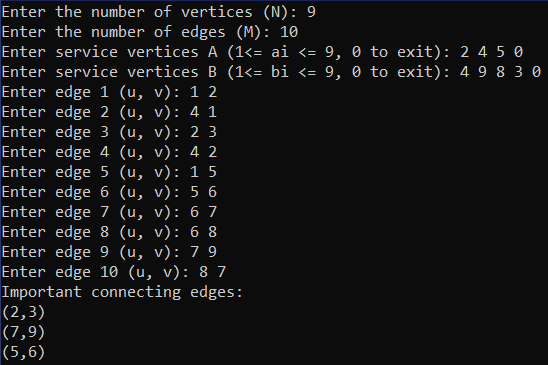
K, L: Lần lượt là số nhà cung cấp dịch vụ A và dịch vụ B

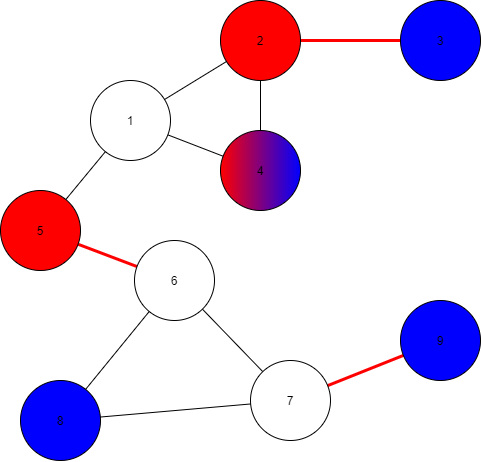
A**[**u**]**, B**[**u**]** : Số nhà cung cấp dịch vụ A/B xét tại u đã được duyệt bằng dps (theo thứ tự topo)

**Độ phức tạp:**

* Duyệt các đỉnh : O(n) (n là số đỉnh của đồ thị)
* Duyệt các đỉnh kề : O(m) (m là số cạnh của đồ thị)
* Kiểm tra 1 cạnh có phải là cạnh quan trọng : O(1)
* Độ phức tạp : O(n + m)

**Chạy thử:**





Các cạnh quan trọng:

* (2, 3): nếu trục trặc: 3 không thể tiếp cận dv A
* (7, 9): nếu trục trặc: 9 không thể tiếp cận dv A
* (5, 6): nếu trục trặc: 6, 7, 8, không thể tiếp cận dịch vụ A