****

**TEAMNOTE  
3D.PTIT**

**Contents**  
Chương I – Đồ thị

1.1 – Tìm đường đi ngắn nhất  
 1.3.1 – Dijkstra  
 1.3.2 – Floyd  
1.2 – Cây khung nhỏ nhất (Kruskal)  
1.3 – Chu trình Euler  
1.4 – Chu trình Hamilton

1.5 – Disjoinset

1.6 – Khớp, cầu

1.7 DistJoinSet

Chương II – Sắp xếp và tìm kiếm

2.1 – Quicksort

2.2 – Heap sort  
 2.2.1 – Heap min  
 2.2.2 – Heap max

2.3 – Rời rạc hóa

Chương III – Cấu trúc dữ liệu

3.1 – Cây BIT (Binary Index Tree)  
3.2 – Cây IT (Interval Tree – Segment Tree)

Chương IV – Toán học

4.1 – Làm tròn số thực  
4.2 – Euclid  
4.3 – Nhân mod và lũy thừa mod  
4.4 – Fast Pow  
4.5 – Phi hàm Euler

4.6 – Công thức hình học

Chương V – String

5.1 – So sánh chuỗi con KMP  
5.2 – Dãy con tăng dài nhất O(nLogn)  
5.3 – LCA  
5.4 – Hasing   
5.5 – RMQ

Chương VI – Bonus

6.1. Matrix

6.2 Bao lồi

6.3 Hash2d

6.4 Manacher

6.5. Cramer

6.6 Lehmar

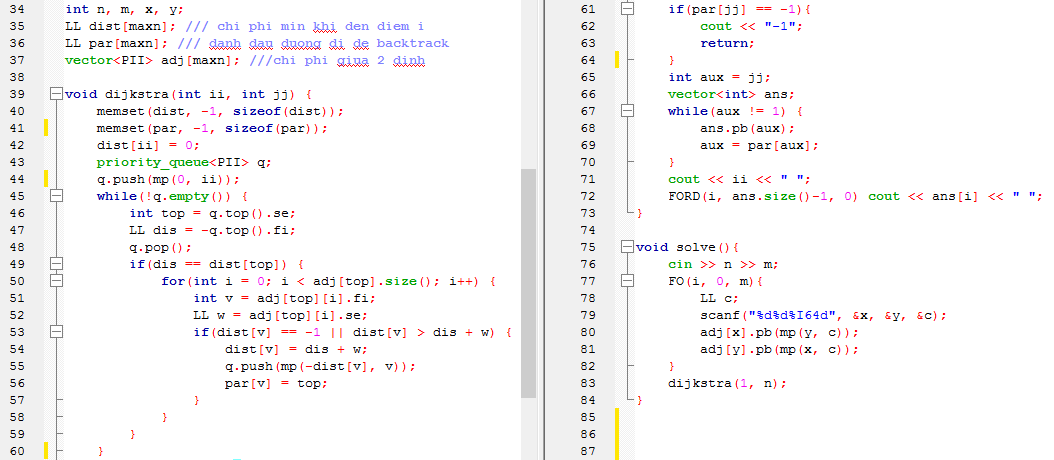
6.7 Cặp ghép cực đại

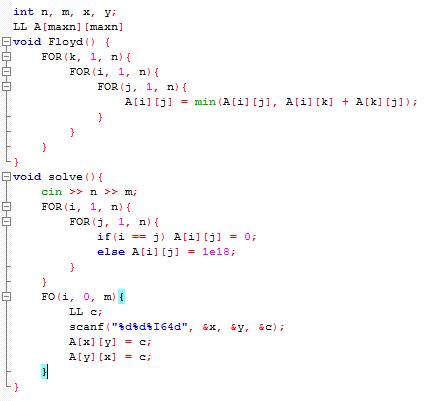
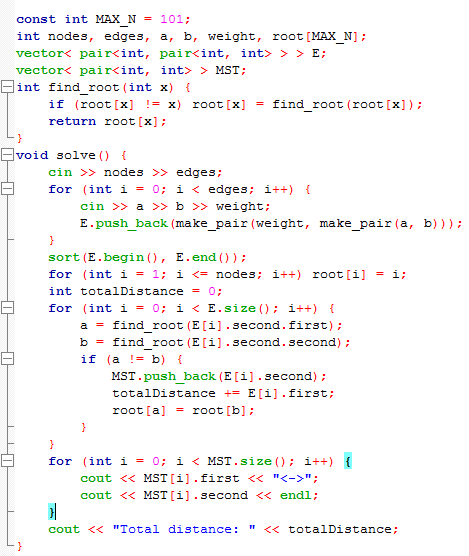
**Template**



**Chương I – Đồ Thị**

**Đồ thị n đỉnh m cạnh , k thành phần liên thông thì có chứa m-n+k chu trình cơ sở**

**1.1.1 – Dijkstra**

**1.1.2 – Floyd** **1.2 – Kruskal**

**1.3 – Chu trình Euler**

stack := (1); *//Ngăn xếp ban đầu chỉ chứa một đỉnh bất kỳ, chẳng hạn đỉnh 1*

repeat

u := Get; *//Đọc phần tử ở đỉnh ngăn xếp*

if ∃(u, v) ∈ E then *//Từ u còn đi tiếp được*

begin

Push(v);

E := E – {(u, v)}; *//Xóa cạnh (u, v) khỏi đồ thị*

end;

else *//Từ u không đi đâu được nữa*

begin

u := Pop; *//Lấy u khỏi ngăn xếp*

Output ← u; *//In ra u*

end;

until stack = ∅; *//Lặp tới khi ngăn xếp rỗng*

**1.4 – Chu trình Hamilton**

procedure Attempt(i: Integer); *//Thuật toán quay lui*

var

v: Integer;

begin

for v := 2 to n do

if avail[v] and a[x[i - 1], v] then *//Xét các đỉnh v kề x[i - 1] chưa đi qua*

begin

x[i] := v; *//Thử đi sang v*

if i = n then *//Nếu đã qua đủ n đỉnh, đến đỉnh thứ n*

begin

if a[v, 1] then found := True; *//Đỉnh thứ n quay về được 1 thì tìm ra nghiệm*

Exit; *//Thoát luôn*

end

else *//Qua chưa đủ n đỉnh*

begin

avail[v] := False; *//Đánh dấu đỉnh đã qua*

Attempt(i + 1); *//Đi tiếp*

if found then Exit; *//Nếu đã tìm ra nghiệm thì thoát ngay*

avail[v] := True;

end;

end;

end;

1. Chu trình Euler:

int A[MAX][MAX], n, u=1;

void Init(void){

freopen("CTEULER.IN","r", stdin);

cin>>n;

cout<<" So dinh cua do thi n = "<<n<<endl;

// nhap ma tran lien ke.

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

for(int j=1; j<=n;j++){

cin>>A[i][j];

}

}

}

int Kiemtra(){

int s, d;

d=0;

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

s=0;

for(int j=1; j<=n;j++)

s+=A[i][j];//đếm các bậc của các đỉnh của đồ thị

if(s%2) d++;

}

if(d>0) return(FALSE); //Nếu có 1 đỉnh bậc lẻ thì đồ thị không có chu trình Euler.

return(TRUE); //Nếu tất cả các đỉnh của đồ thị là chắn thì đồ thị có thể có chu trình Euler.

}

void Tim(){

int v, x, top, dCE;

int stack[MAX], CE[MAX];

top=1;

stack[top]=u;//thêm đỉnh u vào stack.

dCE=0;

do {

v = stack[top];//lấy đỉnh trên cùng của stack.

x=1;

while (x<=n && A[v][x]==0) //tìm trong danh sách những đỉnh kề với đỉnh v.

x++;

if (x>n) { //lấy ra khỏi stack.

dCE++;

CE[dCE]=v;//lưu đỉnh v vào mảng kết quả duyệt CE.

top--;

}

else { //đỉnh x là đỉnh kề với đỉnh v.

top++;

stack[top]=x;

A[v][x]=0;

A[x][v]=0;

}

} while(top!=0);

cout<<" Co chu trinh Euler:";

for(x=dCE; x>0; x--)

cout<<(char)(CE[x] + 'a' - 1)<<" "; //in ra kết quả dưới dạng char.

}

void main(void){

Init();

if(Kiemtra())

Tim();

else printf("\n Khong co chu trinh Euler");

\_getch();

}

1. Đường đi Euler:

#define MAX 50

#define TRUE 1

#define FALSE 0

int n;//số đỉnh của đồ thị.

int m;//số cạnh của đồ thị.

int b[MAX];//mảng b có độ dài m + 1 phần tử.

int u;//đỉnh bậc lẻ trong đồ thị

int OK;//biến kiểm tra đồ thị có đường đi EULER hay không.

int A[MAX][MAX];//ma trận kề của đồ thị.

int i;

void Init(){

freopen("DDEULER.IN", "r",stdin);

cin>>n;

cout<<"So dinh do thi:"<<n;

// nhập ma trận kề.

int s;//biến đếm số bậc của đỉnh i.

int d = 0;//biến đếm số đỉnh bậc lẻ.

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

int s = 0;

for(int j=1; j<=n;j++){

cin>>A[i][j];

s+=A[i][j];

}

if (s%2) {

d++;//tắng giá trị biến đếm đỉnh bậc lẻ.

u=i;// ghi nhớ đỉnh bậc lẻ.

}

m=m+s;

}

m=m /2;

if (d!=2) OK=FALSE;// Nếu số đỉnh bậc lẻ khác 2 thì không có đường đi Euler.

else OK=TRUE;

}

void Result(void){

cout<<"Co duong di Euler:";

for(int i=0; i<=m; i++)

cout<<(char)(b[i] + 'a' - 1)<<" "; //in ra kết quả dưới dạng char.

cout<<endl;

}

void DDEULER(int i){

for(int j=1; j<=n;j++){

if (A[b[i-1]][j]==1){

A[b[i-1]][j]=0;

A[j][b[i-1]]=0;

b[i]=j;

if(i==m){

Result();

}

else{

DDEULER(i+1);

}

A[b[i-1]][j]=1;

A[j][b[i-1]]=1;

}

}

}

void main(void){

Init();

b[0]=u;//Gán b[0] nhận giá trị là đỉnh lẻ của đồ thị.

i=1;

if(OK) DDEULER(i);

else printf("\n Khong co duong di Euler");

getch();

}

**1.5 – Disjoinset**

int find\_set(int u){

if (p[u]<0) return(u);

p[u]= find\_set(p[u]);

return(p[u]);

}

void union\_(int u, int v){

int r1,r2;

r1= find\_set(u);

r2= find\_set(v);

if (r1==r2) return;

if (p[r1]<p[r2]){

p[r1]=p[r1]+p[r2];

p[r2]=r1;

}

else{

p[r2]=p[r1]+p[r2];

p[r1]=r2;

}

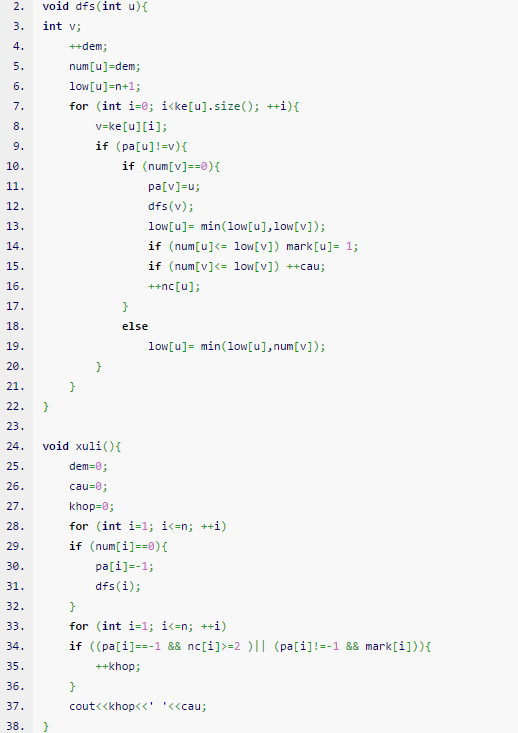
}

void makeset(){

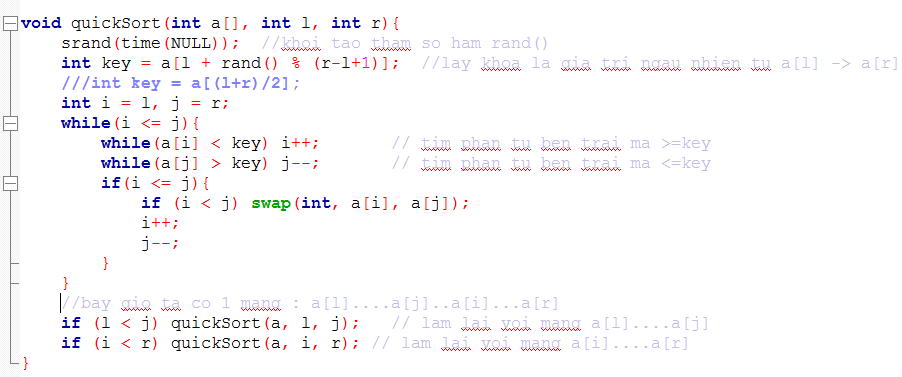
for (int i=1; i<=n; ++i) p[i]=-1;

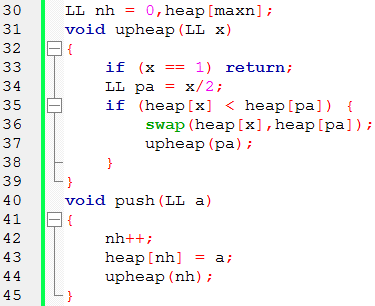
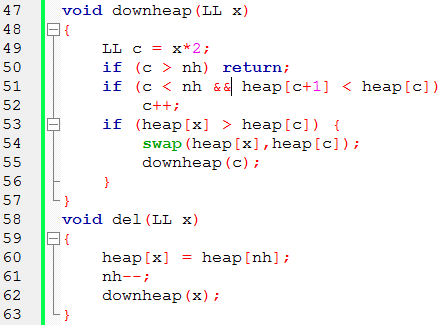
}

**1.6 – Khớp và cầu:**

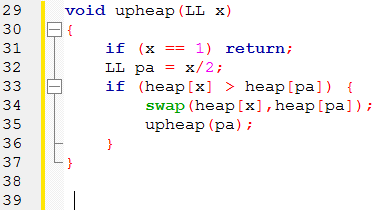
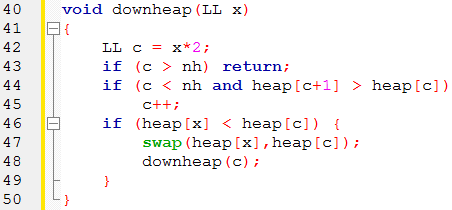


**Chương II – Sắp xếp và Tìm kiếm**

**2.1 – Quick Sort (Fastest: Nlog(N) – Slowest: N2)** **2.2.1 – Heap min**

**2.2.2 – Heap max**

**2.3 – Rời rạc hóa**

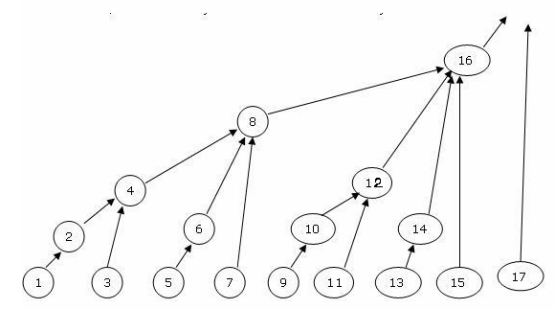
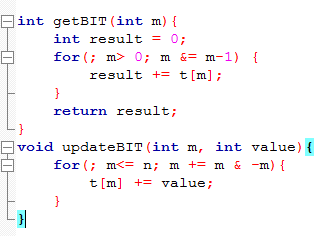
for (int i = 1 ;i <= n ;++i) x[i] = a[i];

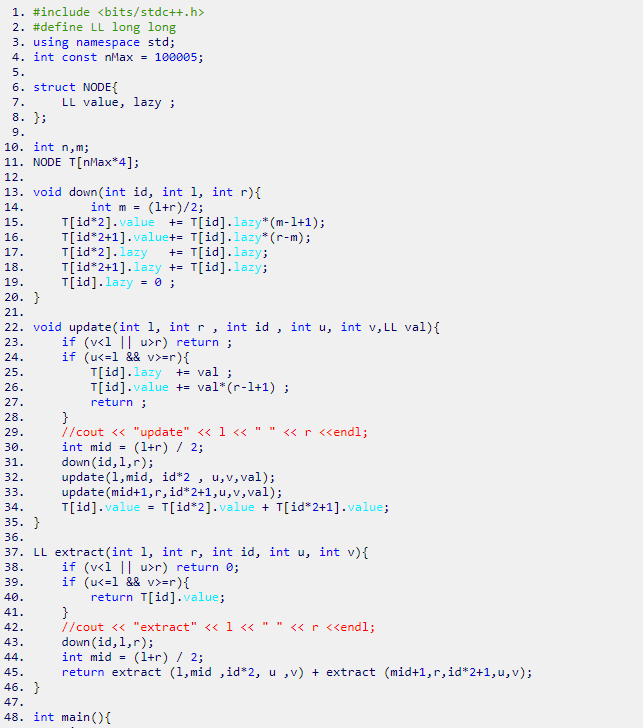
sort(x + 1 , x + n + 1);

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

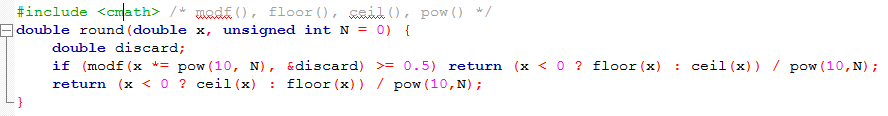
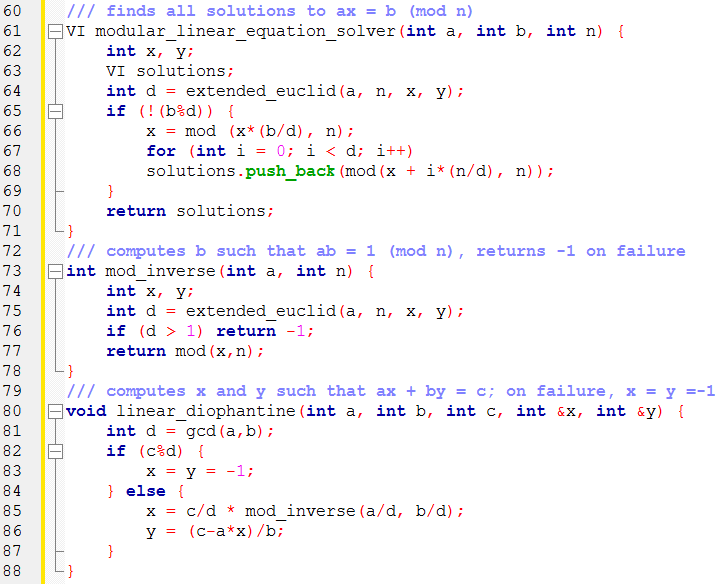
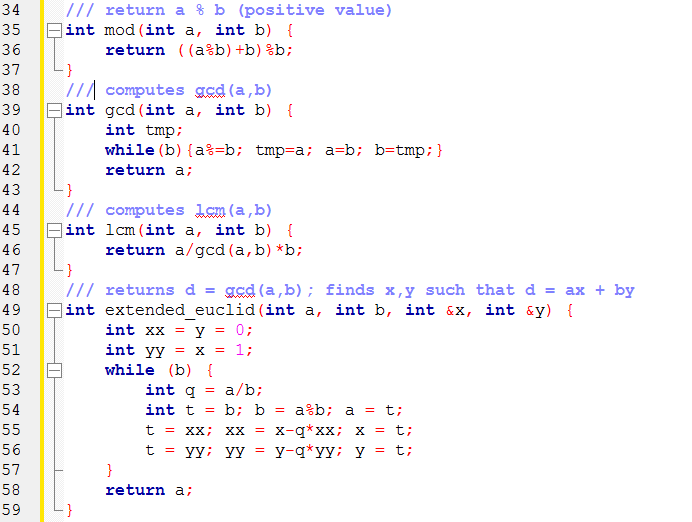
a[i] = lower\_bound(x+1,x+n+1,a[i]) - x;

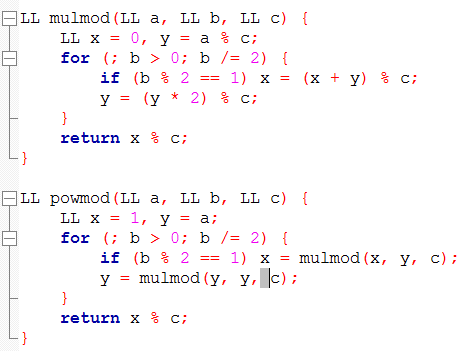
**Chương III – Cấu trúc dữ liệu**

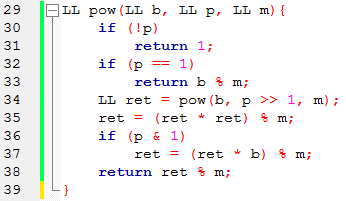
**3.1 – Cây BIT**Tổng quát, đặt m = 2k.p (với p là số lẻ). Hay nói cách khác, k là vị trí của bít 1 bên phải nhất của m. Trong cây BIT, nút có số hiệu m sẽ là nút gốc của một cây con gồm 2k nút có số hiệu từ m- 2k+1 đến m.  
8 = 23.1, vậy 8 là nút gốc của các nút 1, 2, 3, …, 8.  
Trong cây BIT, nút gốc đại diện cho tất cả các nút con của nó. Ý nghĩa của từ đại diện ở đây thường dùng là nút gốc lưu tổng giá trị của các nút con. Vì vậy khi tính toán, ta chỉ cần truy xuất nút gốc là đủ mà không cần thiết phải truy xuất đến các nút con. Xét ví dụ:  
Cho mảng gồm n phần tử a1, a2, …, an. Hãy tính tổng Am =  a1 + a2 + … + am (m ≤ n).  
Thay vì sử dụng vòng lặp từ 1 đến m để truy xuất từng phần tử ai một (độ phức tạp O(m)), ta sử dụng cấu trúc BIT như sau  
 - t1 = a1  
- t2 = a1 + a2  
- t3 = a3  
- t4 = a1 + a2 + a3 + a4  
- t5 = a5  
- t6 = a5 + a6  
- t7 = a7  
- t12 = a9 + a10 + a11 + a12

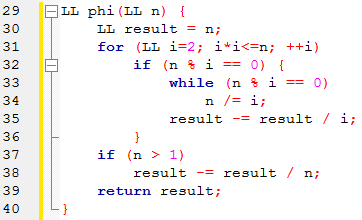
**3.2 – Cây IT với lazy update**

**CHƯƠNG IV : TOÁN HỌC**

**4.1 – Làm tròn số thực** **4.2 – Euclid**

**4.3 – Nhân mod và Lũy thừa mod**

**4.4 – Fast Pow**

**4.5 – Phi hàm Euler**

Phi hàm Euler của số n (phi(n)) được định nghĩa bằng số các số nhỏ hơn n và nguyên tố cùng nhau với n

**4.6 – Công thức hình học**

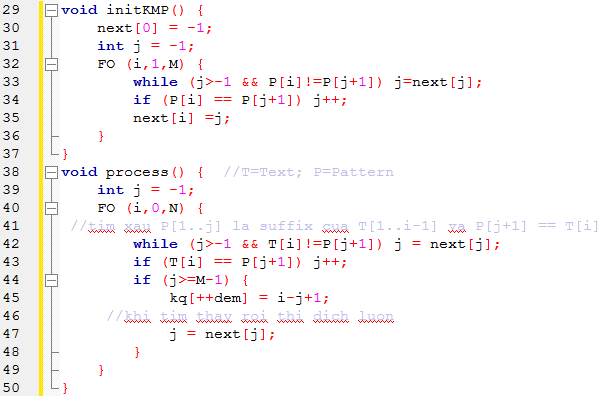
**- Diện tích đa giác :**

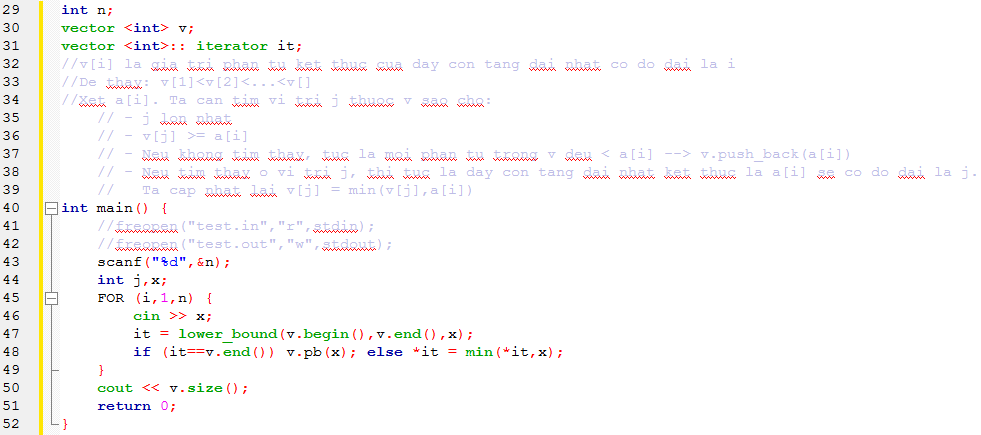
Description: http://vnschool.net/images/102010/Toantin/Toantin_15102010_h10.jpg

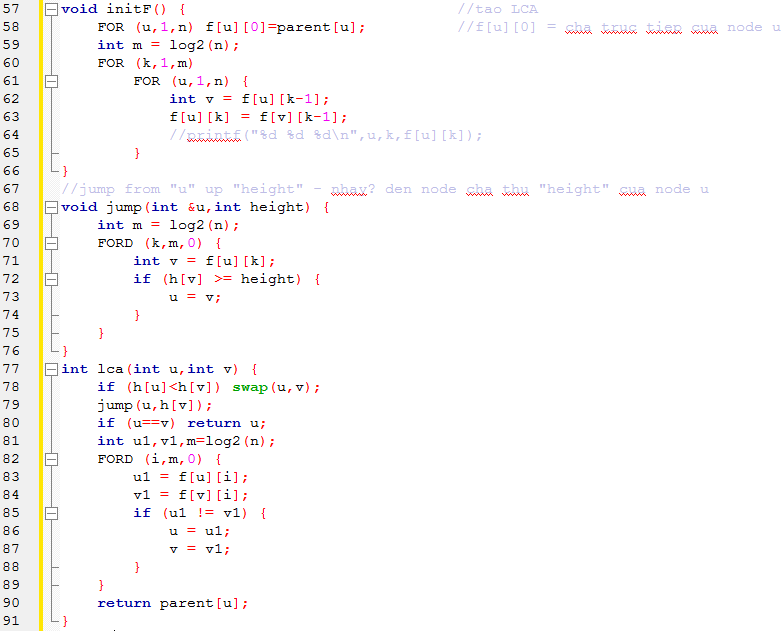
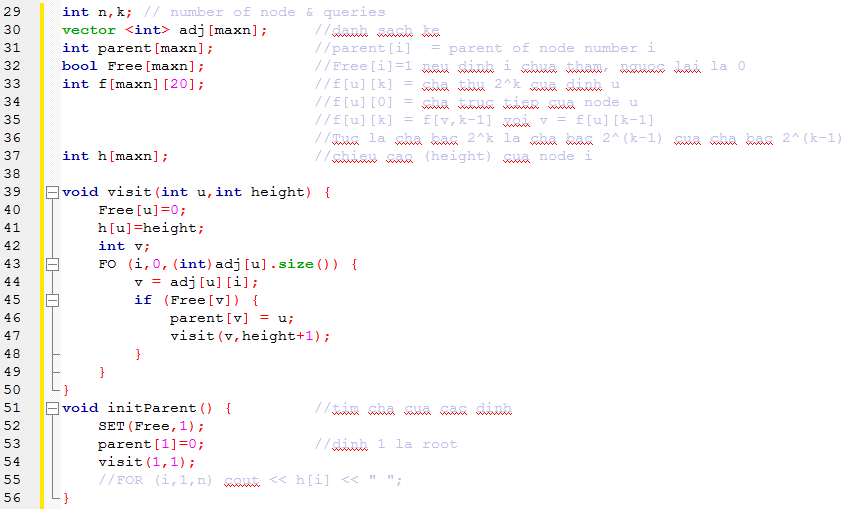
**4.7 – Xử lí BIT**

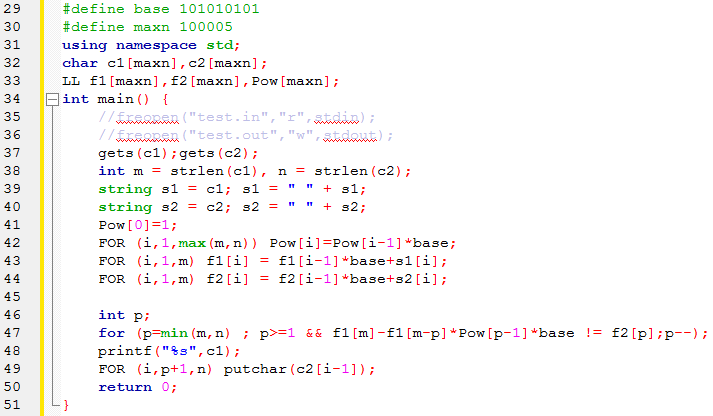


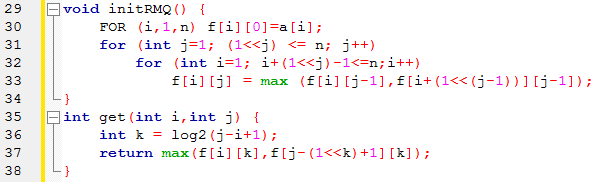
**Chương V – String**

**5.1 – So sánh chuỗi con KMP**

**5.2 – Dãy con tăng dài nhất O(nLogn)**

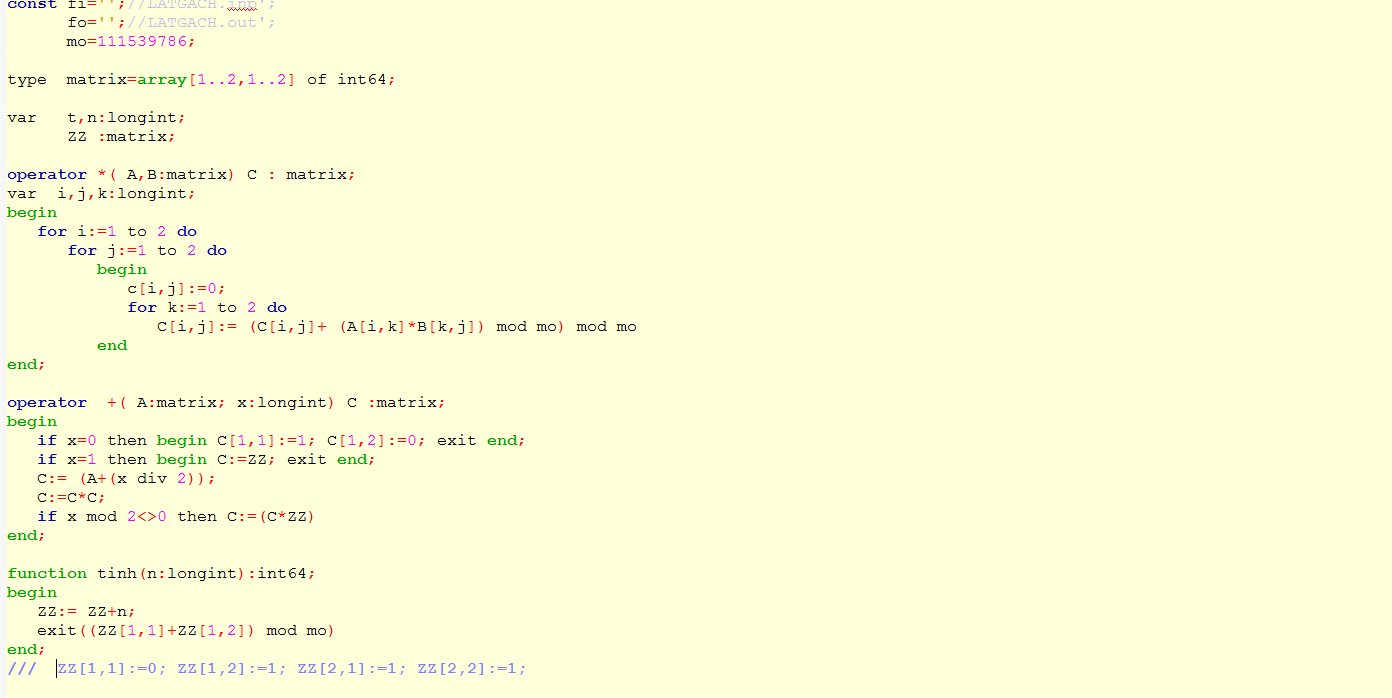
**5.3 – LCA**

**5.4 – Hashing**

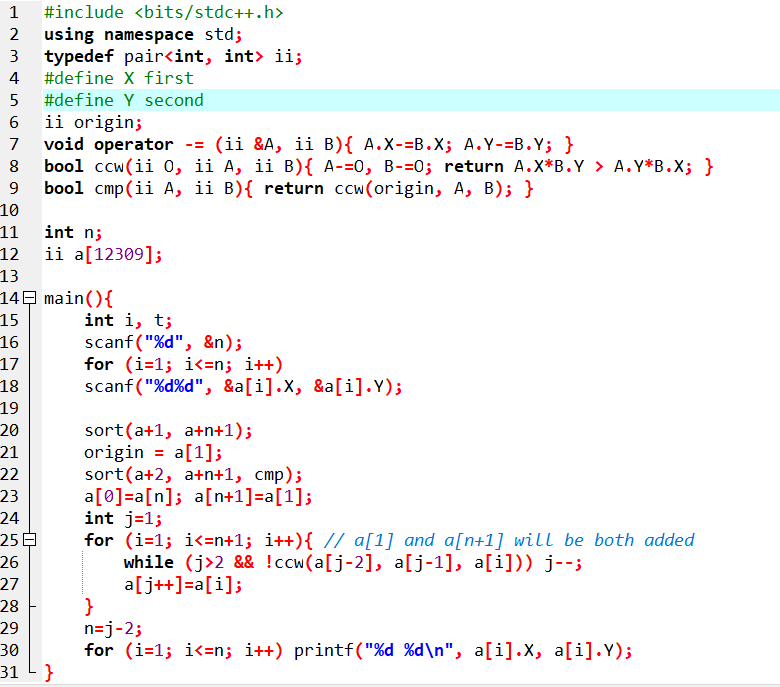
**5.6 – RMQ**

**Chương VI – BONUS**

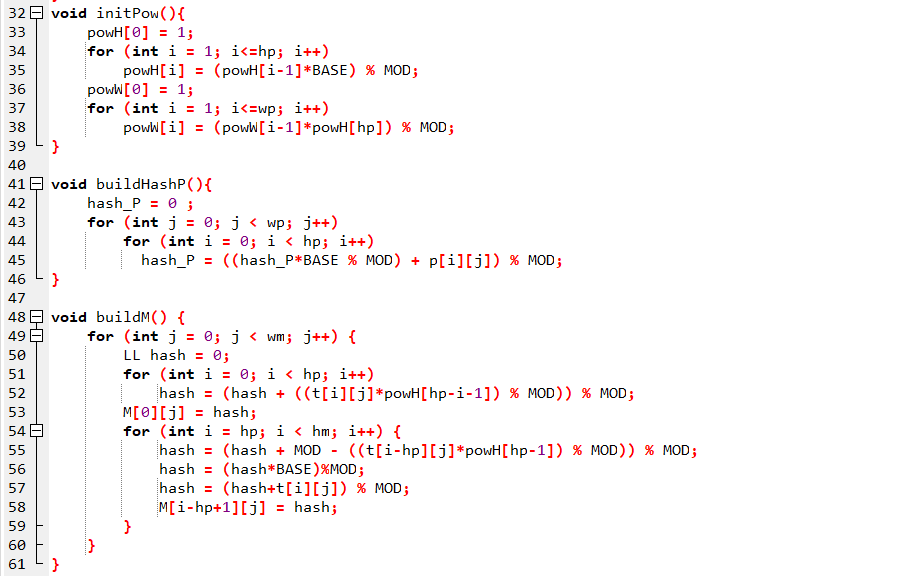
**6.1. Matrix**

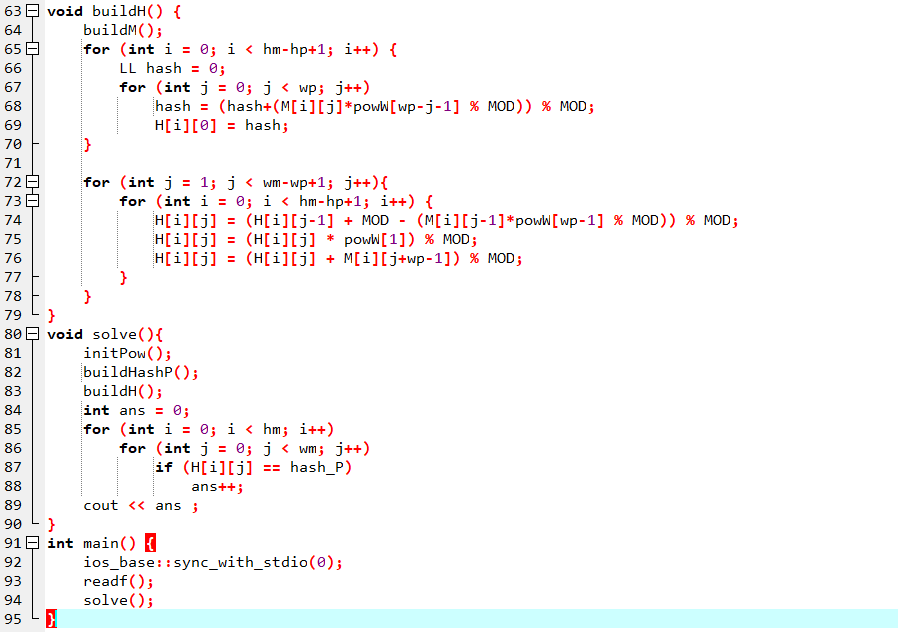
****

**6.2 Bao lồi**

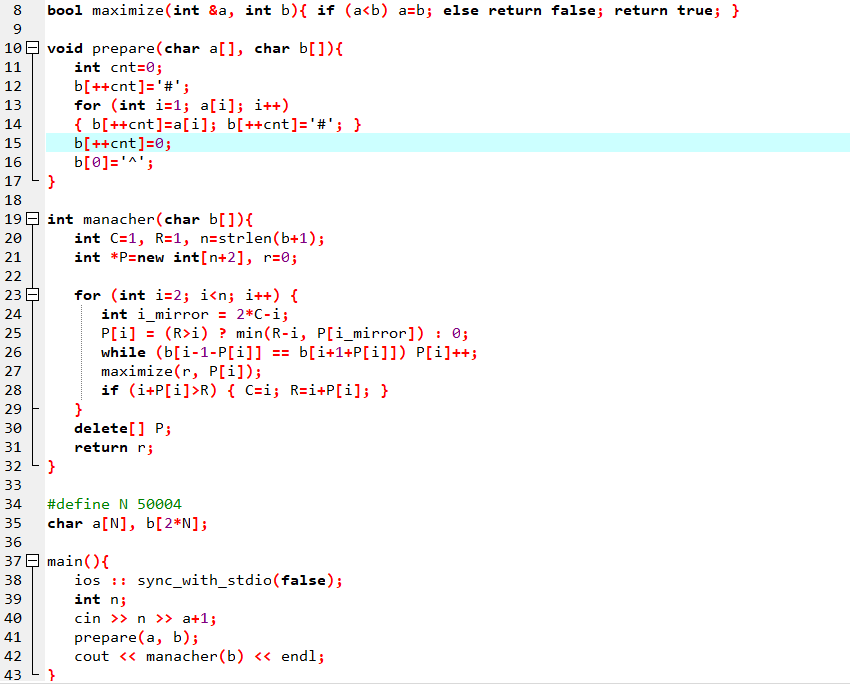


**6.3. Hash 2D**

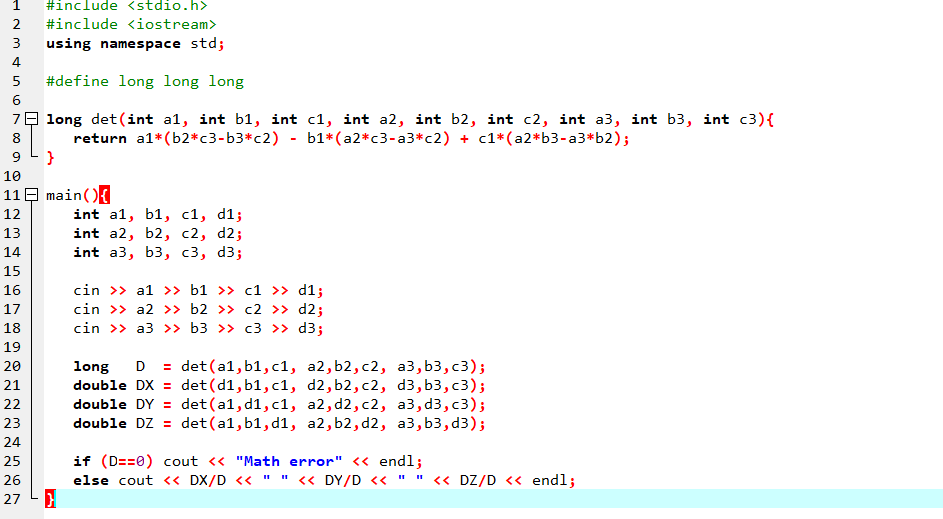




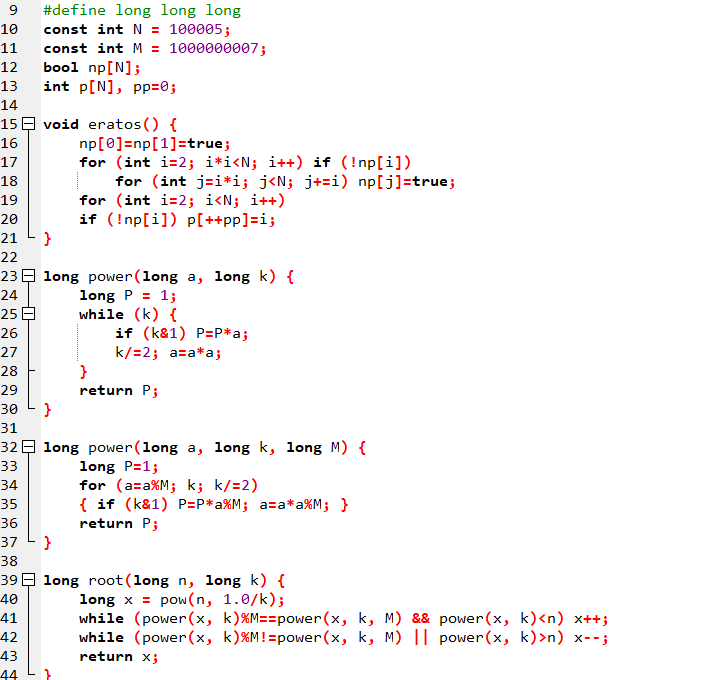
**6.4. Manacher**

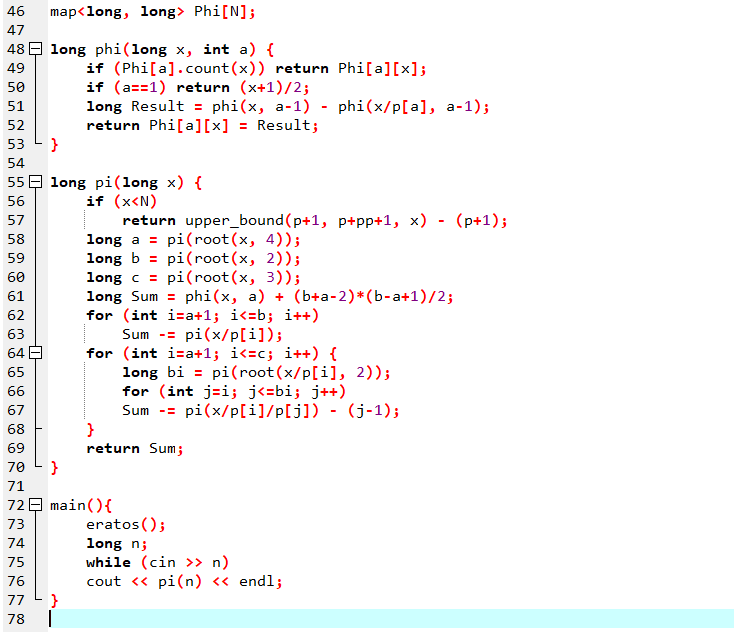


**6.5. Cramer**

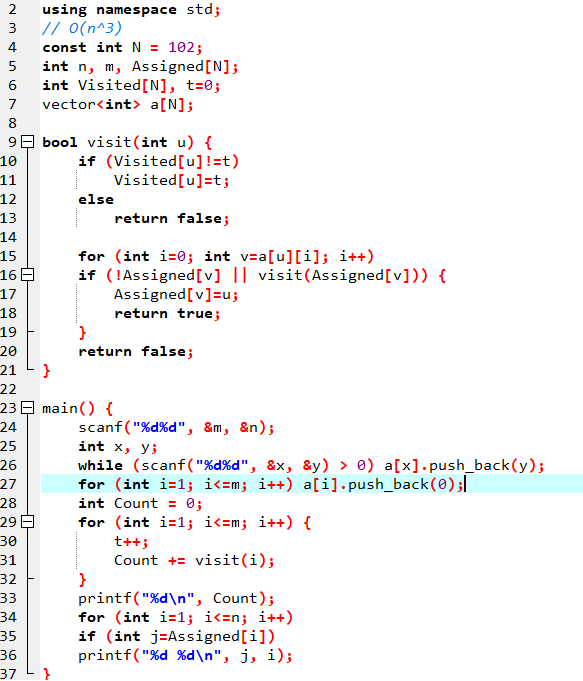


**6.6. Lehmer**



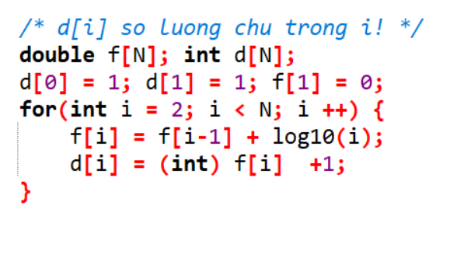
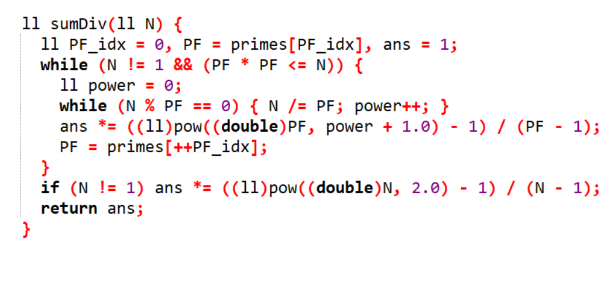


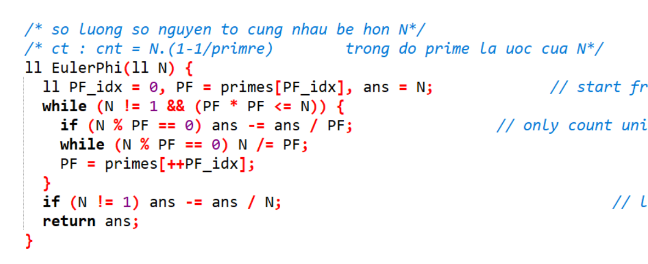
**6.8. Cặp ghép cực đại**

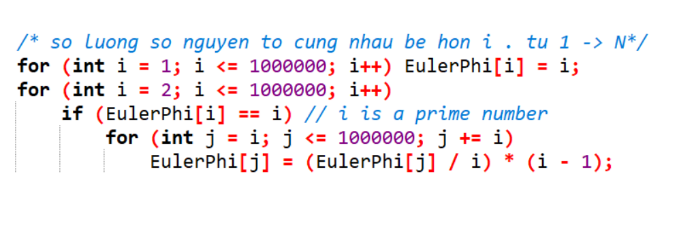


1. Sinh hoán vị:

+ next\_permutation(a,a+n); tạo ra số mới nhỏ nhất lơn hơn hoặc bằng mảng a ban đầu  
+ prev\_permutation(a,a+n); tạo ra số mới lớn nhất nhỏ hơn hoặc bằng số ban đầu.

1. Công thức tính tổng các ước của N: Số chữ số của n!
2. Sô nguyên cùng nhau





4 . LCM/GCD

