**Binary Method**

*// Function to calculate y^n using the binary method*

int binaryMethod(int y, int n)

{

    int result = 1;

*// While n is greater than 0*

    while (n > 0)

    {

*// If n is odd, multiply result by y*

        if (n & 1 == 1 ) *// so le*

        {

            result \*= y;

        }

*// Divide n by 2 and square y*

        n >>= 1;

        y =y\* y;

    }

*return* result;

}

Đây là một hàm để tính toán y^n sử dụng phương pháp nhị phân.

Hàm này bắt đầu bằng việc khởi tạo biến result với giá trị ban đầu là 1. Sau đó, hàm sẽ lặp qua một vòng lặp với điều kiện là n > 0. Trong mỗi lần lặp, nếu n là số lẻ (được kiểm tra bằng cách dùng toán tử & với 1), thì nó sẽ nhân kết quả hiện tại với y. Sau đó, n sẽ được chia đôi và y sẽ được bình phương. Cuối cùng, hàm sẽ trả về kết quả cuối cùng của result.

Phương pháp nhị phân này cho phép tính toán y^n một cách hiệu quả hơn so với cách tính bằng cách nhân y với chính nó bằng cách sử dụng các phép nhân và chia đôi, và có thể được sử dụng để tính toán y^n cho các số lớn.

Độ phức tạp của thuật toán trên là **O(logn)**, vì số lần lặp trong vòng lặp chính là logarit cơ số 2 của n. Vì vậy, thuật toán này có hiệu suất tốt hơn so với các thuật toán có độ phức tạp O(n), như việc tính toán y^n bằng cách nhân y với chính nó bằng cách sử dụng các phép nhân.

**Factor Method**

#include <iostream>

#include <iomanip>

#include <math.h>

#include <algorithm>

#include <string>

using namespace std;

int soNguyenTo(int n);

void phanTich(int n1, int m1, int *&*count);

int y;

unsigned long long kq; *//kết quả y^n*

int main()

{

    int n;

    cout << "P=y^n\nNhap n:";

    cin >> n;

    cout << "Nhap y: ";

    cin >> y;

    kq = y;

    int n1 = n; *//Tạo S(n1,m1)=S(n,1)*

    int m1 = 1;

    string str;

    int count = 0; *//biến tính số phép nhân*

    cout << "Phan tich: \nBan dau lay y" << endl;

    phanTich(n1, m1, count);

    cout << "So phep nhan toi thieu: " << count << endl;

    cout << kq;

*return* 0;

}

int soNguyenTo(int n) *//hàm xác định n có là số nguyên tố*

{

    if (n == 2)

*return* 0;

    for (int i = 2; i <= sqrt(n); i++)

    {

        if (n % i == 0)

        {

*return* i; *// nếu n không phải là số nguyên tố thì trả về thừa số nguyên tố bé nhất*

        }

    }

*return* 0; *// nếu n là số nguyên tố thì trả về 0*

}

void phanTich(int n1, int m1, int *&*count) *//hàm phân tích S(n1,m1)*

{

    if (n1 != 1) *//S(n1,m1)=S(1,m1)= chuỗi null, nếu không bằng chuỗi null thì thực hiện if*

    {

        if (soNguyenTo(n1) == 0) *// nếu n1 là số nguyên tố*

        {

*//S(n1,m1)=S(n1-1,m1)Xm1*

            n1 = n1 - 1;

            count++; *//với mỗi X thì tăng 1 phép nhân*

            cout << "Nhan voi y^" << m1 << endl; *//Xm1 nghĩa là "Nhân với y^m1"*

            if (m1 != 1) *//nếu m1!=1 thì không phân tích nữa*

            {

*return*;

            }

            phanTich(n1, m1, count); *//tạo ra 1 S(n1,m1) mới, tiếp tục phân tích*

            kq = kq \* y; *// tính (y^(n-1))\*y*

        }

        else if (soNguyenTo(n1) > 0) *// nếu n1 không phải số nguyên tố*

        {

*//S(n, m) = S(n/p,m)S(p, nm/p)*

            int p = soNguyenTo(n1); *//p= thừa số nguyên tố bé nhất*

            int n2 = p;

            int m2 = (n1 \* m1) / p;

            n1 = n1 / p;

            phanTich(n1, m1, count); *//gán S(n1,m1)= S(n1/p,m1) và phân tích tiếp*

            phanTich(n2, m2, count); *//gán S(n2,m2)= S(p, n1m1/p) và phân tích tiếp*

            kq = pow(kq, p); *// tính (y^p1)^p2*

        }

    }

}

*/\**

*Ví dụ cách tính của y^13:*

*y^13=y^12\*y*

*=(y^6)^2\*y*

*=((y^3)^2)^2\*y*

*=((y^2\*y)^2)^2\*y*

*=((y\*y\*y)^2)^2\*y*

*\*/*

Chương trình này nhằm tính toán giá trị của **yn** bằng cách sử dụng số phép nhân ít nhất. Để làm điều này, đầu tiên nó phân tích **n** thành các thừa số nguyên tố của nó

Hàm **soNguyenTo** có nhiệm vụ phân tích **n** thành các thừa số nguyên tố của nó.Hàm **phanTich** thực hiện điều này bằng cách chia nhiều lần **n** cho thừa số nguyên tố nhỏ nhất của nó cho đến khi nó bằng 1. Đối với mỗi phép chia, nó tăng một bộ đếm **count** để theo dõi số phép nhân được thực hiện.

Hàm **soNguyenTo** dùng để xác định xem một số có phải là số nguyên tố hay không. Nếu số đó là số nguyên tố, nó sẽ trả về 0. Nếu số không phải là số nguyên tố, nó sẽ trả về thừa số nguyên tố nhỏ nhất của số đó.

Trước tiên, hàm chính đọc các giá trị của **n**và **y**từ người dùng, sau đó gọi hàm **phanTich** để phân tách **n** và tính toán kết quả cuối cùng, kết quả này được lưu trữ trong biến **kq**. Sau đó, nó in ra kết quả và số phép nhân được thực hiện.

**Tree Method**

#include <iostream>

using namespace std;

int main()

{

*//2^r*

  int LINKU[177147] = {0}, k = 0, LINKR[177147] = {0},

  q*// nút cuối cùng thêm vào cấp k*

  , s *// biến dùng để chạy từ gốc-> nút hiện tại đang xét*

  , nm*// nút cuối cùng được thêm vào cây*

  , m*// nút cuối cùng được thêm vào mức hiện tại*

  , n; *// bien lap qua cac nut o cap do hien tai*

  LINKR[0] = 1;

  LINKR[1] = 0;

  int r; *//số mũ của y^r*

  cout<<"Nhap r:";

  cin>>r;

  while (k < r) *// vòng lặp sẽ tiếp tục cho đến cấp độ mong muốn (nút k)*

  {

    n = LINKR[0]; *// n được đặt thành nút đầu tiên của cấp độ hiện tại*

    m = 0; *// nút cuối cùng của cấp độ hiện tại*

    do

    {

      q = 0, s = n; *// nút cuối cùng thêm vào cấp k + biến nút hiện tại là n*

      do

      {

        if (LINKU[n + s] == 0) *// nút phía trên nút hiện tại có tồn tại không*

        {

          if (q == 0) *// có phải là nút cuối thêm vào cấp k hiện tại*

            nm = n + s; *// nút cuối cùng thêm vào cấp k (chỉ số)*

          LINKR[n + s] = q; *// them mot nut ben phai nut hien tai*

          LINKU[n + s] = n; *// them mot nut phia tren nut hien tại*

          q = n + s; *// nút cuối được thêm vào cấp k là n+s*

        }

        s = LINKU[s]; *// biến chạy nút ( từ gốc -> nút now) thành nút hiện tại*

      } while (s != 0); *// lặp từ gốc đến mức s ( s là biến chạy nút gốc- > nút đang xét) và hoàn thành các nút con bên trong mức k*

      if (q != 0) *// nếu ở trên có thêm bất kỹ nút nào được tao ra trong vòng lặp*

      {

        LINKR[m] = q; *// nút cuối cùng của mức độ hiện tại có giấ trị là nút cuối cùng thêm vào cấp k (k này k cuối cùng vì đã thoát loop ở trên)*

        m = nm; *// nút cuối cùng mức độ hiên tại( mức cuối ) đặt thành nút cuối cùng thêm vào cây*

      }

      n = LINKR[n]; *// n đặt thành nút mới tiếp theo ở mức k*

    } while (n != 0); *// vòng lặp liên tục cho đến khi các nút ở mức độ hiện tại được xử lí*

    LINKR[m] = 0; *// cái nút cuói cùng trong cái tầng cấp độ đã xong hiện tại có khôing có nút R (bên phải) --> để trỏ sang cấp độ tiếp theo*

    k = k + 1; *// tăng cấp độ của cây lên*

  }

  cout << m;

}

Đoạn code trên triển khai một biến thể của cấu trúc dữ liệu cây nhị phân, trong đó mỗi nút có tối đa hai nút con. Các nút của cây được đại diện bởi một mảng **LINKU**, lưu trữ các chỉ số của các nút cha của các nút và một mảng **LINKR**, lưu trữ các chỉ số của các nút con bên phải của các nút.

Hàm **main** bắt đầu bằng cách đọc giá trị của **r** từ người dùng và khởi tạo cây với một nút gốc và một nút con. Sau đó, nó đi vào một vòng lặp sẽ tiếp tục cho đến khi đạt đến mức mong muốn **k** ( lặp cho đến khi bằng **r**).

Bên trong vòng lặp, mã xác định một số biến: **n**, **m**, **q**và **s**. **n** được đặt thành nút đầu tiên của cấp độ hiện tại, **m** được đặt thành nút cuối cùng của cấp độ hiện tại, **q** được sử dụng để lưu trữ nút cuối cùng được thêm vào cấp độ **k+1**và **s** được sử dụng để lặp qua các nút của cấp độ hiện tại từ gốc đến nút nút hiện tại.

Sau đó, mã đi vào một vòng lặp lồng nhau bắt đầu tại **n** và lặp qua các nút của cấp hiện tại cho đến khi nó đến một nút không có nút cha (được biểu thị bằng giá trị 0 trong mảng **LINKU** ). Đối với mỗi nút, đoạn mã này sẽ kiểm tra xem nút phía trên nó (được biểu thị bằng chỉ mục của nó trong mảng **LINKU** ) có tồn tại hay không. Nếu nó không tồn tại, mã sẽ tạo một nút mới phía trên nút hiện tại và đặt nó làm nút con bên phải của nút hiện tại. Nếu đây là nút đầu tiên được thêm vào cấp độ **k+1**, thì nó cũng được lưu dưới dạng **nm**nút cuối cùng được thêm vào cấp độ **k+1**. Sau đó, mã tiếp tục lặp qua các nút của cấp độ hiện tại cho đến khi kết thúc.

Nếu bất kỳ nút nào được thêm vào cấp độ **k+1**trong vòng lặp bên trong, mã sẽ đặt nút con bên phải của nút cuối cùng của cấp độ hiện tại (**m**) thành nút cuối cùng được thêm vào cấp độ **k+1** ( **q** ). Sau đó, nó đặt nút cuối cùng của cấp độ hiện tại thành nút cuối cùng được thêm vào cấp độ **k+1** và tăng dần **k** để chuyển sang cấp độ tiếp theo.

Vòng lặp tiếp tục cho đến khi **k**trở thành bằng **r**, tại thời điểm đó, mã sẽ in ra các chỉ số của các nút trong cây và kết thúc. **n** được **r** nhập bởi người dùng. Điều này là do thuật toán bao gồm hai vòng lặp lồng nhau, mỗi vòng lặp qua tất cả các nút của một cấp độ của cây. Thời gian thực hiện thuật toán tăng theo phương trình bậc hai với số nút trong cây.Độ phức tạp không gian của thuật toán cũng là **O(n^2)**, vì nó liên quan đến việc tạo một mảng có kích thước **n^2** để lưu trữ các nút của cây.

Top of Form