**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**🙡 🕮 🙣**



**ĐỒ ÁN TIN HỌC**

**ĐỀ TÀI: TRÒ CHƠI NIM**

Lớp: D21\_TH10

Họ và tên: Nguyễn Hoàng Anh Quân

MSSV: DH52111603

**LỜI MỞ ĐẦU**

Trò chơi Nim đã tồn tại từ hàng thế kỷ và là một trò chơi logic với tính toán cao và thử thách trí tuệ. Trò chơi này không chỉ đòi hỏi người chơi sự tư duy chiến lược, mà còn yêu cầu khả năng đọc tâm lý đối thủ và đưa ra những quyết định thông minh. Trong báo cáo này, chúng ta sẽ khám phá quy tắc chơi, các chiến lược và thuật toán liên quan đến trò chơi Nim.

**MỤC LỤC**

**I. Cấu trúc dữ liệu và giải thuật**

1. Cấu trúc dữ liệu 1

2. Giải thuật 2

**II. Chương trình trò chơi NIM**

Chương trình trò chơi NIM 3

**I. Cấu trúc dữ liệu và giải thuật:**

Trò chơi này được tạo ra dựa trên cấu trúc dữ liệu mảng một chiều và ngôn ngữ C++ cùng với thuật toán NIMSUM trong giải thuật.

**1. Cấu trúc dữ liệu:**

- Mảng một chiều trong C++:

Trong ngôn ngữ lập trình C++, mảng một chiều là một cấu trúc dữ liệu cơ bản để lưu trữ một tập hợp các phần tử có cùng kiểu dữ liệu. Mảng một chiều cho phép bạn lưu trữ và truy cập vào các phần tử bằng cách sử dụng chỉ số (index) của chúng.

Để khai báo một mảng một chiều trong C++, bạn sử dụng cú pháp sau:

*<kiểu\_dữ\_liệu> <tên\_mảng>[kích\_thước];*

Ví dụ, để khai báo một mảng một chiều chứa 5 số nguyên, bạn có thể viết như sau:

*int myArray[5];*

Sau khi khai báo, bạn có thể truy cập và gán giá trị cho các phần tử của mảng bằng cách sử dụng chỉ số của chúng. Chúng ta phải nhớ rằng chỉ số của mảng bắt đầu từ 0 đến (kích thước - 1).

Ví dụ, để gán giá trị cho phần tử thứ hai trong mảng:

*myArray[1] = 10;*

Để truy cập và in giá trị của phần tử thứ ba trong mảng:

*cout << myArray[2] << endl;*

Mảng một chiều trong C++ cũng hỗ trợ các hoạt động như duyệt qua các phần tử của mảng bằng vòng lặp, sao chép mảng, so sánh các mảng, và nhiều hoạt động khác.

Một điểm cần lưu ý là trong C++, mảng một chiều không kiểm tra việc truy cập vượt quá chỉ số hợp lệ. Việc truy cập vào phần tử không hợp lệ có thể dẫn đến lỗi thời gian chạy (run-time error). Do đó, cần đảm bảo rằng chỉ số truy cập vào mảng luôn nằm trong phạm vi hợp lệ của mảng.

Mảng một chiều trong C++ là một công cụ mạnh mẽ và phổ biến để lưu trữ và xử lý dữ liệu có cùng kiểu dữ liệu. Nó cung cấp khả năng truy cập và thao tác dễ dàng trên các phần tử của mảng.

- Ưu điểm:

+ Đơn giản và dễ sử dụng: Mảng một chiều là một cấu trúc dữ liệu đơn giản và phổ biến, dễ hiểu và dễ sử dụng trong việc lưu trữ và xử lý thông tin về trạng thái của các đống sỏi và viên sỏi.

+ Truy cập dễ dàng: Mảng một chiều cho phép truy cập vào các phần tử bằng cách sử dụng chỉ số. Điều này rất hữu ích trong trò chơi Nim khi cần lấy số lượng sỏi từ một đống sỏi cụ thể.

+ Tính linh hoạt: Mảng một chiều có thể thay đổi kích thước và được cập nhật dễ dàng. Khi người chơi lấy sỏi từ một đống sỏi, chỉ cần giảm giá trị của phần tử tương ứng trong mảng.

- Nhược điểm:

+ Khó khăn trong việc thay đổi kích thước: Khi trạng thái của trò chơi Nim thay đổi, ví dụ như xóa đi đống sỏi hoặc thêm đống sỏi mới, mảng một chiều cần được cập nhật kích thước. Điều này có thể tốn nhiều thời gian và tài nguyên tính toán.

+ Không hỗ trợ thao tác chèn và xóa hiệu quả: Trong trường hợp cần chèn thêm đống sỏi hoặc xóa đống sỏi khỏi trạng thái trò chơi, mảng một chiều không phải là cấu trúc dữ liệu hiệu quả nhất. Thao tác này yêu cầu di chuyển các phần tử trong mảng, gây tốn thời gian và tài nguyên.

+ Không lưu trữ thông tin chi tiết: Mảng một chiều chỉ lưu trữ thông tin về số lượng sỏi trong mỗi đống sỏi, không lưu trữ thông tin về vị trí và sự phân bố của các viên sỏi. Điều này có thể làm giảm tính linh hoạt trong việc phân tích và đưa ra các chiến lược phức tạp hơn.

**2. Giải thuật:**

- Trò chơi được tạo nên dựa vào thuật toán NIMSUM, trong trò chơi Nim bóc sỏi với người bóc viên cuối là thua, thuật toán Nimsum được áp dụng để xác định chiến lược chiến thắng cho người chơi đầu tiên. Mục tiêu của người chơi đầu tiên là tạo ra một trạng thái mà giá trị nim-sum của nó là 0, từ đó buộc người chơi thứ hai vào một trạng thái không thể thắng.

Thuật toán Nimsum trong trò chơi Nim bóc sỏi với người bóc viên cuối là thua có các bước thực hiện như sau:

+ Lấy tất cả các giá trị sỏi từ các hàng trong trạng thái hiện tại của trò chơi.

+ Thực hiện phép XOR trên tất cả các giá trị sỏi này.

+ Kết quả của phép XOR chính là giá trị nim-sum của trạng thái hiện tại.

+ Nếu giá trị nim-sum khác 0, có nghĩa là trạng thái hiện tại không có đường đi thắng và người chơi đầu tiên sẽ thua.

+ Người chơi đầu tiên cần tìm cách tạo ra một trạng thái mới sao cho giá trị nim-sum của nó khác 0. Để làm điều này, người chơi đầu tiên cần tìm những hàng có giá trị sỏi sao cho khi thực hiện phép XOR với giá trị nim-sum hiện tại, kết quả là một giá trị nim-sum nhỏ hơn. Người chơi đầu tiên có thể điều chỉnh số lượng sỏi trong hàng đó để đạt được điều kiện này.

+ Bằng cách tạo ra một trạng thái với giá trị nim-sum khác 0, người chơi đầu tiên buộc người chơi thứ hai vào một trạng thái không thể thắng và đạt chiến thắng.

Thuật toán Nimsum trong trò chơi Nim bóc sỏi với người bóc viên cuối là thua giúp người chơi đầu tiên xác định chiến lược chơi để đạt chiến thắng. Bằng cách tạo ra một trạng thái mà giá trị nim-sum khác 0, người chơi đầu tiên có thể đảm bảo rằng người chơi thứ hai sẽ không có cách để tránh thua cuộc.

**II. Chương trình trò chơi NIM:**

#include <iostream>

#include <cstdlib>

#include <cmath>

using namespace std;

const int MAX = 100;

struct PileOfRocks

{

int n;

int a[MAX];

};

void InputPile(PileOfRocks& p)

{

//Nhập số lượng đống ban đầu

cout << "\nEnter the number of piles of rocks: ";

while (!(cin >> p.n) || p.n <= 0 || p.n > MAX)

{

cout << "Invalid input! Please enter again within the range of 1 to " << MAX << ": ";

cin.clear();

cin.ignore(123, '\n');

}

}

void InputRocks(PileOfRocks& p)

{

//Tạo ra viên sỏi ngẫu nhiên cho mỗi đống

srand((int)time(0)); //Truyền giá trị động, thời gian hiện hành (đảm bảo mỗi lần chạy rand() sẽ cho ra kết quả khác nhau)

for (int i = 0; i < p.n; i++)

{

p.a[i] = 1 + rand() % 100;//(1+(0->99)=(1->100))

}

}

void OutputRocks(PileOfRocks p)

{

//Danh sách đống sỏi

for (int i = 0; i < p.n; i++)

{

cout << "\nPile[" << i + 1 << "]: " << p.a[i];

}

}

void PeopleTurn(PileOfRocks& p)

{

int Pile, Rocks;

cout << endl;

cout << "\n========================================================================================================================" << endl;

cout << "\n[YOUR'S TURN]";

cout << "\n========================================================================================================================" << endl;

cout << "\nRemaining: ";

OutputRocks(p);

//Nhập đống muốn bóc

cout << "\n\nChoose a pile of rocks (1-" << p.n << "): ";

while (!(cin >> Pile) || Pile < 1 || Pile > p.n || p.a[Pile - 1] == 0)

{

cout << "Invalid input! Please enter again: ";

cin.clear();

cin.ignore(123, '\n');

}

Pile--;

//Nhập số viên sỏi muốn bóc

cout << "\nChoose the number of rocks (1-" << p.a[Pile] << "): ";

while (!(cin >> Rocks) || Rocks < 1 || Rocks > p.a[Pile])

{

cout << "Invalid input! Please enter again: ";

cin.clear();

cin.ignore(123, '\n');

}

p.a[Pile] -= Rocks;

cout << "\nPeople takes " << Rocks << " rocks from Pile[" << Pile + 1 << "]" << endl;

}

int Count(PileOfRocks p)

{

//Đếm đống sỏi

int count = 0;

for (int i = 0; i < p.n; i++)

{

if (p.a[i] > 0)

count++;

}

return count;

}

bool CountA(PileOfRocks p)

{

//Kiểm tra chỉ còn 2 đống còn sỏi với 1 đống có 1 viên sỏi

for (int i = 0; i < p.n; i++)

{

if (p.a[i] == 1 && Count(p)==2)

{

return true;

}

}

return false;

}

int CountB(PileOfRocks p)

{

//Gán vị trí của đống sỏi có viên sỏi lớn hơn 1

int vitriB = 0;

for (int i = 0; i < p.n; i++)

{

if (p.a[i] > 1)

{

vitriB = i;

break;

}

}

return vitriB;

}

bool CountC(PileOfRocks p)

{

//Kiểm tra có còn 1 đống duy nhất với số sỏi lớn hơn 1

for (int i = 0; i < p.n; i++)

{

if (p.a[i] > 1 && Count(p) == 1)

{

return true;

}

}

return false;

}

void ComputerTurn(PileOfRocks& p)

{

cout << "\n========================================================================================================================" << endl;

cout << "\n[COMPUTER'S TURN]";

cout << "\n========================================================================================================================" << endl;

cout << "\nRemaining: ";

OutputRocks(p);

cout << endl;

//Thuật toán nimsum

int nims = 0;

for (int i = 0; i < p.n; i++)

{

nims ^= p.a[i];

}

if (nims == 0)

{

srand(time(NULL));

int Pile = rand() % (p.n);

//Lập tới khi tìm được đống có sỏi khác 0

while (p.a[Pile] == 0)

{

Pile = rand() % (p.n);

}

int Rocks = rand() % p.a[Pile] + 1;

p.a[Pile] -= Rocks;

cout << "\nThe computer takes " << Rocks << " rocks from Pile[" << Pile + 1 << "]" <<endl;

}

else

{

for (int i = 0; i < p.n; i++)

{

if ((p.a[i] ^ nims) < p.a[i] && p.a[i]>0)

{

//Nếu còn 2 đống và 1 trong 2 có số sỏi bằng 1

if (CountA(p))

{

//Gán đống bằng vị trí có số sỏi lớn hơn 1, lấy hết sỏi trong đống đó

int Pile = CountB(p);

int Rocks = p.a[Pile];

p.a[Pile] = 0;

cout << "\nThe computer takes " << Rocks << " rocks from Pile[" << Pile + 1 << "]" << endl;

}

//Nếu còn 1 đống và sỏi lớn hơn 1

else if (CountC(p))

{

//Gán đống bằng vị trí đống có sỏi lớn hơn 1, lấy chừa lại 1 viên sỏi

int Pile = CountB(p);

int Rocks = p.a[Pile]-1;

p.a[Pile] = 1;

cout << "\nThe computer takes " << Rocks << " rocks from Pile[" << Pile + 1 << "]" << endl;

}

else

{

int Rocks = p.a[i] - (p.a[i] ^ nims)-1;

if (Rocks < 1 && p.a[i]>0)

{

Rocks = 1;

}

p.a[i] -= Rocks;

cout << "\nThe computer takes " << Rocks << " rocks from Pile[" << i + 1<<"]" << endl;

break;

}

}

}

}

}

int main()

{

cout << "\n========================================================================================================================" << endl;

PileOfRocks p;

InputPile(p);

InputRocks(p);

OutputRocks(p);

cout << "\n========================================================================================================================" << endl;

cout << "\nCoin toss: ";

string Toss = (rand() % 2 == 0) ? "People" : "Computer";

cout << Toss << " First!";

cout << "\n========================================================================================================================" << endl;

while (Count(p) != 0)

{

if (Toss == "People")

{

PeopleTurn(p);

Toss = "Computer";

}

else

{

ComputerTurn(p);

Toss = "People";

}

OutputRocks(p);

}

if (Toss == "People")

{

cout << "\nPeople win!";

}

else

{

cout << "\nComputer win!";

}

return 0;

}

**KẾT THÚC**

Trò chơi Nim là một trò chơi cổ điển đơn giản nhưng vô cùng thú vị, đòi hỏi người chơi phải có sự tư duy chiến lược và khéo léo. Trong báo cáo này, chúng ta đã tìm hiểu về cấu trúc dữ liệu mảng một chiều và giải thuật Nimsum được áp dụng trong trò chơi Nim.

Cấu trúc dữ liệu mảng một chiều trong ngôn ngữ C++ cho phép lưu trữ và truy cập vào các phần tử một cách dễ dàng, điều này rất hữu ích trong việc lưu trữ trạng thái của các đống sỏi và viên sỏi trong trò chơi Nim. Mặc dù mảng một chiều có nhược điểm như khó khăn trong việc thay đổi kích thước và không lưu trữ thông tin chi tiết, nhưng vẫn là một công cụ mạnh mẽ cho việc xử lý dữ liệu có cùng kiểu.

Giải thuật Nimsum là một phần quan trọng trong trò chơi Nim, giúp người chơi đầu tiên xác định chiến lược chơi để đạt chiến thắng. Bằng cách tạo ra một trạng thái mà giá trị nim-sum khác 0, người chơi đầu tiên có thể buộc người chơi thứ hai vào một trạng thái không thể thắng. Thuật toán Nimsum cung cấp một phương pháp tính toán thông minh và hiệu quả để đánh bại đối thủ.

Trò chơi Nim không chỉ là một trò chơi giải trí mà còn mang trong mình những bài học quan trọng về tư duy chiến lược, tính toán và khéo léo. Bạn có thể áp dụng những nguyên tắc và kỹ năng học được từ trò chơi này vào cuộc sống hàng ngày, đặc biệt là trong việc đưa ra quyết định và tìm kiếm giải pháp cho các vấn đề phức tạp.

**Tài Liệu Tham Khảo**

1. <https://www.geeksforgeeks.org/nim-game-algorithm-winning-strategies/>
2. <https://brilliant.org/wiki/sprague-grundy-theorem/>