**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO TOÁN RỜI RẠC**

**CHỦ ĐỀ: TỔNG QUAN VỀ CÁC KỸ THUẬT QUY HOẠCH ĐỘNG, NHÁNH CẬN**

**Học phần: Toán rời rạc**

**Nhóm HP: 63.CNTT-2**

**Họ và tên: Nguyễn Tuấn Tú**

**MSSV: 63135967**

**Giảng viên: GV. Nguyễn Hải Triều**

2024

**MỤC LỤC**

**CHƯƠNG 1: LÝ THUYẾT**

**I. Quy hoạch động:**

**1. Giới thiệu chung:**

Phương pháp quy hoạch động dùng để giải bài toán tối ưu có bản chất đệ quy, tức là việc tìm phương án tối ưu cho bài toán ban đầu có thể đưa về tìm phương án tối ưu của 1 số hữu hạn các bài toán con.

Trong giải thuật đệ quy, có một số bài toán con bị giải nhiều lần dẫn đến tốn rất nhiều thời gian.

Giải pháp: Mỗi bài toán con chỉ giải 1 lần và lưu trữ kết quả bài toán vào 1 biến rồi sử dụng lại kết quả đã được lưu trữ mà không cần phải giải lại bài toán con lại nữa.

Tuy nhiên có những bài toán chứa số bài toán con rất lớn nên cần sử dụng 1 bảng (mảng) để lưu trữ kết quả các bài toán con thay vì biến.

Tạo bảng bằng cách:

* + Gán giá trị cho 1 ô nào đó.
  + Gán giá trị cho các ô khác nhờ vào giá trị của ô trước đó

Kết quả của bài toán ban đầu cũng nằm trong bảng này, do đó ta chỉ cần tra bảng để xác định kết quả của bài toán ban đầu

**2. Phạm vi áp dụng:**

Các bài toán tối ưu như: tìm xâu con chung dài nhất, tổng dãy con bằng S, tì dãy con đơn điệu tăng dần, bài toán balo…

Các bài toán có công thức truy hồi.

**3. Các bước giải bài toán quy hoạch động:**

* B1: Xác định hàm mục tiêu.
* B2: Xác định nghiệm tối ưu của bài toán trong trường hợp đơn giản nhất.
* B3: Tìm các công thức truy hồi biểu diễn nghiệm tối ưu của bài toán lớn thông qua nghiệm tối ưu của các bài toán con.
* B4: Tính nghiệm các bài toán con và ghi lại các nghiệm của các bài toán con đã tính để sử dụng sau này.
* B5: Tra bảng để tìm ra kết quả.

**4. Ví dụ:**

Bài toán Tính dãy Fibonacci: f(0) = 0; f(1) = 1; f(n) = f(n-1) + f(n-2), ∀𝑛≥2

B1: Hàm mục tiêu: f(n)

B2: Nghiệm của bài toán trong trường hợp đơn giản nhất: f(0) = 0 , f(1) = 1

B3: Tìm các công thức truy hồi biểu diễn nghiệm tối ưu của bài toán lớn thông qua nghiệm tối ưu của các bài toán con: f(n) = f(n-1) + f(n-2), ∀𝑛≥2

B4: Lưu vào bảng phương án: N=5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 |
| F[i] | 0 | 1 | 2 | 3 | 5 | 8 |

B5: Tra bảng phương án F(5) là F[5]=8

**II. Nhánh cận:**

**1. Giới thiệu chung:**

Là 1 sự cải tiến của thuật toán quay lui dựa trên cơ sở xác định cận dưới/trên tương ứng với bải toán min/max cho từng phương án bộ phận. Thay vì xây dựng toàn bộ cây thì ta sử dụng giá trị cận để hạn chế bớt các nhánh, nếu ta đến 1 vị trí mà giá trị tại các hàm mục tiêu tại đó và các điểm về sau chắc chắn không tốt nhất thì quay lại.

Cây tìm kiếm phương án có nút gốc biểu diễn cho tập tất cả các phương án, mỗi nút lá biểu diễn cho 1 phương án nào đó. Nút n có các nút con tương ứng với các khả năng có thể lựa chọn tập phương án xuất phát từ n. Kỹ thuật này gọi là phân nhánh.

Với mỗi nút trên cây ta sẽ xác định 1 giá trị cận. Giá trị cận là 1 giá trị gần với giá của các phương án, với bài toán tìm min ta sẽ xác định cận dưới, tìm max ta sẽ xác định cận trên.

Cận dưới là giá trị nhỏ hơn hoặc bằng giá của phương án, cận trên là giá trị lớn hơn hoặc bằng giá của phương án.

**2. Phạm vi áp dụng:**

Phương phánh nhánh cận thường được áp dụng cho các bài toán tối ưu, đặc biệt là các bài toán tổ hợp như tìm kiếm trong không gian trạng thái hoặc tìm kiếm trong không gian các phương án.

**3. Các bước giải bải toán nhánh - cận:**

B1: Xác định hàm mục tiêu.

B2: Xác định hàm giới hạn (cận).

B3: Phân nhánh.

B4: Kiểm tra và cắt nhánh không cần thiết.

B5: Đánh giá và chọn lựa

B6: Lặp lại quá trình

**3. Ví dụ:**

Bài toán Người đi du lịch, tìm min của hàm để người đi quãng đường ngắn nhất:

f(x1, x2, …, xn) = c[1, x2] + c[x2, x3] + … + c[x(n-1), xn] 🡪 min

Với điều kiện (x1, x2, …, xn) là hoán vị của các số 2,3,…,n.

Ký hiệu: cmin = min{c[i, j], i, j = 1,2,…,n,i ≠ j }

Giả sử ta có phương án bộ phận (u1,u2,u3,…,uk) tương ứng với hành trình bô phận qua k thành phố. T1 🡪 T(u2) 🡪 … 🡪 T(u(k-1)) 🡪 T(uk)

Vì vậy, chi phí phải trả theo hành trình bộ phận qua k thành phố

σ = c[1, u2] + c[u2, u3] +…+ c[u(k-1), uk]

Cận dưới của phương án bộ phận: g(u1, u2, u3, … , uk) = σ + (n – k + 1)\*cmin

Giải bài toán người đi du lịch:

c =

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 3 | 14 | 18 | 15 |
| 3 | 0 | 4 | 22 | 20 |
| 17 | 9 | 0 | 16 | 4 |
| 6 | 2 | 7 | 0 | 12 |
| 9 | 15 | 11 | 5 | 0 |

Vậy cmin = 3

**III. So sánh ưu, nhược điểm:**

|  |  |  |
| --- | --- | --- |
| Kỹ thuật | Quy hoạch động | Nhánh cận |
| Ưu điểm | - Chương trình thực hiện nhanh  - Có thể vận dụng giải các bài toán tối ưu, các bài toán có công thức truy hồi | - Không quét qua toàn bộ các nghiệm có thể có của bài toán  - Giảm chi phí: do loại bỏ những bước đi không cần thiết ( nhờ đánh giá cận) |
| Nhược điểm | - Không tìm được công thức truy hồi.  - Số lượng bài toán con cần giải quyết và lưu trữ kết quả là rất lớn.  - Sự kết họp lời giải của các bài toán con chưa chắc đã giải được bài toán ban đầu | - Khó khăn trong việc làm thế nào để đánh giá được các nghiệm mở rộng (cận). Nếu đánh giá tốt sẽ bỏ nhiều nghiệm không cần thiết phải xét (nhánh). |

**CHƯƠNG 2: LẬP TRÌNH**

**I. Câu 1:**

**1. Đề bài:**

Lập trình cài đặt thuật toán quay lui giải bài toán N-Queens.

**2. Ý tưởng giải thuật:**

Bài toán yêu cầu đặt N quân hậu lên bàn cờ nxn sao cho chúng không đớp lẫn nhau khi được đặt trên cùng 1 hàng, 1 cột hay 1đường chéo.

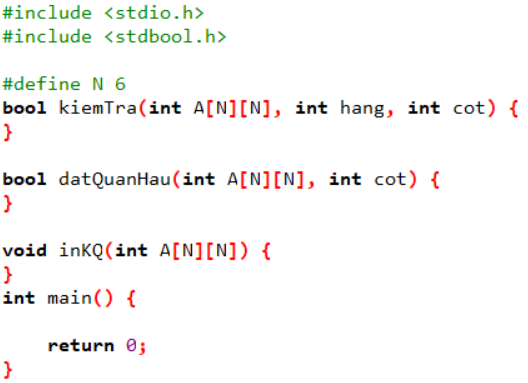
Ý tưởng giải thuật của bài toán N-Queens là sử dụng phương pháp quay lui để thử từng cách xếp quân hậu một cách tuần tự trên bàn cờ sao cho chúng không ăn được lẫn nhau.

Cụ thể, giải thuật bắt đầu với một bàn cờ trống và thử từng cột một để đặt quân hậu. Trong mỗi cột, đệ quy sẽ được sử dụng để thử từng hàng một và kiểm tra xem quân hậu có thể được đặt ở vị trí đó mà không bị tấn công bởi các quân hậu đã được đặt trước đó hay không.

Khi một vị trí an toàn được tìm thấy, quân hậu sẽ được đặt tại đó và giải thuật sẽ di chuyển đến cột tiếp theo để đặt quân hậu tiếp theo. Nếu không thể tìm thấy một vị trí an toàn cho quân hậu trong cột hiện tại, giải thuật sẽ quay lui lại và thử lại các lựa chọn khác trong cột trước đó.

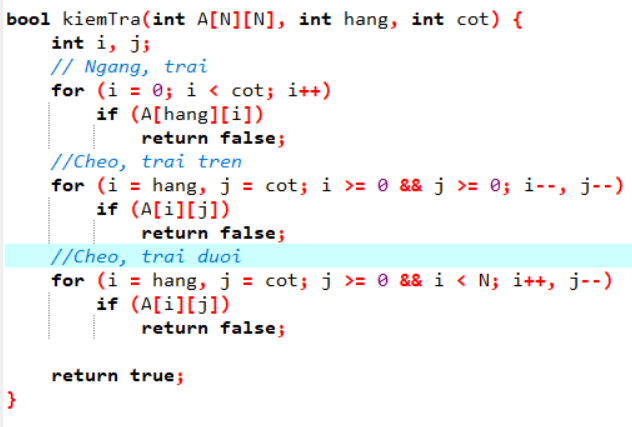
Quá trình này được lặp lại cho đến khi tất cả các quân hậu đã được đặt trên bàn cờ hoặc không có cách nào để đặt quân hậu mà vẫn đảm bảo an toàn. Kết quả cuối cùng sẽ là một cách xếp quân hậu hợp lệ trên bàn cờ, hoặc thông báo rằng không có cách xếp nào thỏa mãn yêu cầu.

**3. Code chương trình:**

****

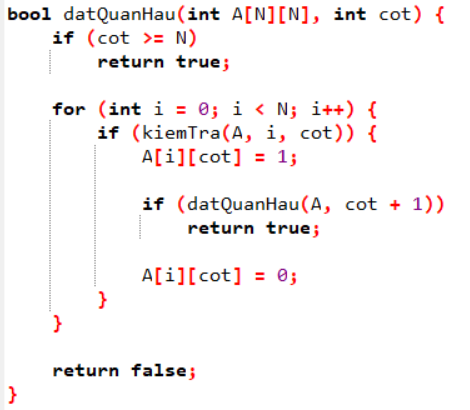
Code gồm 3 hàm chính:

* + Hàm datQuanHau: dùng để đặt quân hậu.
  + Hàm kiemTra: dùng để kiểm tra vị trí quân hậu có đặt được hay không theo 3 hướng.
  + Hàm inKQ: dùng để in ra vị trí của quân hậu trong mảng 2 chiều ( số 1 là vị trí đặt, 0 là trống).



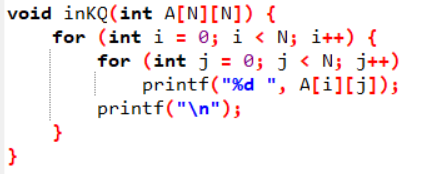
Hàm kiemTra gồm:

* Kiểm tra theo hướng ngang: Kiểm tra những cột phía trước cột tại vị trí muốn đặt quân hậu mới. Nếu đã tồn tại thì ta không đặt tại vị trí này được 🡪 Trả về false.
* Kiểm tra theo hướng trái trên: Chạy vòng lặp hàng và cột giảm dần tạo thành đường chéo. Nếu đã tồn tại thì ta không đặt tại vị trí này được 🡪 Trả về false.
* Kiểm tra theo hướng trái dưới: Chạy vòng lặp hàng tăng và cột giảm dần tạo thành đường chéo. Nếu đã tồn tại thì ta không đặt tại vị trí này được 🡪 Trả về false.
* Nếu kiểm tra tất cả những trường hợp trên mà không bị false lần nào thì ta có thể đặt quân hậu ở đó 🡪 Trả về true.

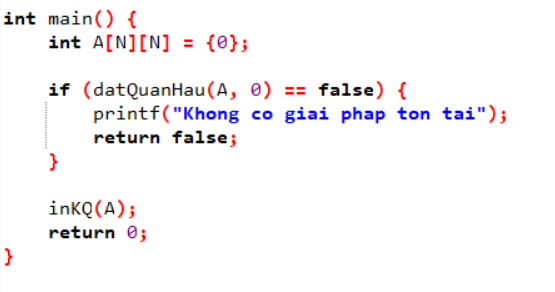


Hàm datQuanHau gồm:

* Điều kiện dừng: Nếu giá trị cột > giá trị N thì ta đã đặt được hết tất cả quân hậu 🡪 Trả về true, kết thúc chương trình.
* Sau khi kiểm tra lần đầu tiên thì chưa thể thực hiện điệu kiện dừng nên ta sẽ sử dụng vòng lập đặt các giá trị theo N cột rồi dùng hàm kiemTra để kiểm tra. Nếu A[i][cot] = 1 thì ta đặt được hậu.
* Ta tiếp tục kiểm tra vị trí đặt hậu ở cột kế tiếp cot+1 thì trả về true còn nếu không thì ta xét lại giá trị A[i][cot] = 0 🡨 Thực hiện kỹ thuật quay lui.

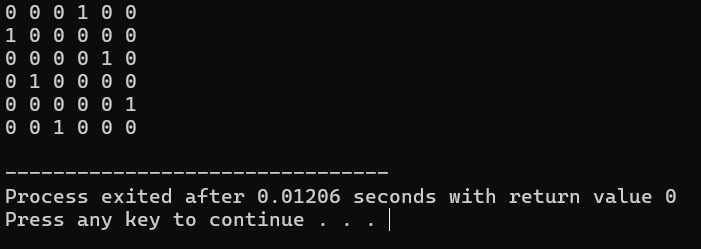


Hàm inKQ: Hiện thị giá trị hàng và cột của mảng.



Hàm main: Thông báo nếu sau khi thực hiện vòng for trong datQuanHau nếu không trả về true được thì thông báo lỗi.

**4. Chạy chương trình:**

****

**II. Câu 2:**

**1. Đề bài:**

Lập trình cài đặt thuật toán quy hoạch động giải bài toán Xếp ba lô 1.

**2. Ý tưởng giải thuật:**

Đầu tiên cần tạo mảng 2 chiều để lưu trữ giá trị tối đa có thể đạt được cho mỗi trọng lượng và số lượng của vật và được gán giá trị 0.

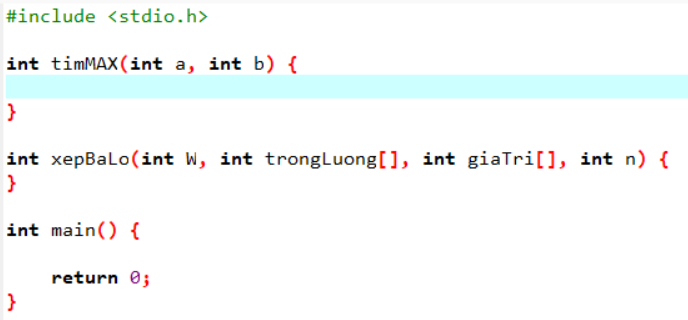
Sử dụng vòng lặp để duyệt qua từng vật và giá trị của chúng.

Tại mỗi ô của mảng, ta tính giá trị tối đa của vật dựa trên 2 trường hợp:

* + Nếu trọng lượng của vật hiện tại nhỏ hơn hoặc bằng trọng lượng đang xét, ta có thể chọn giữa việc bỏ qua vật này hoặc bao gồm vật này và sử dụng giá trị tối đa từ phần còn lại của ba lô.
  + Nếu trọng lượng của vật hiện tại lớn hơn trọng lượng đang xét, ta bỏ qua vật này và sử dụng giá trị tối đa từ phần còn lại của ba lô.

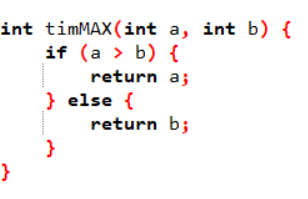
Trả về kết quả là giá trị max ở ô cuối cùng, cột bên phải của mảng.

**3. Code chương trình:**

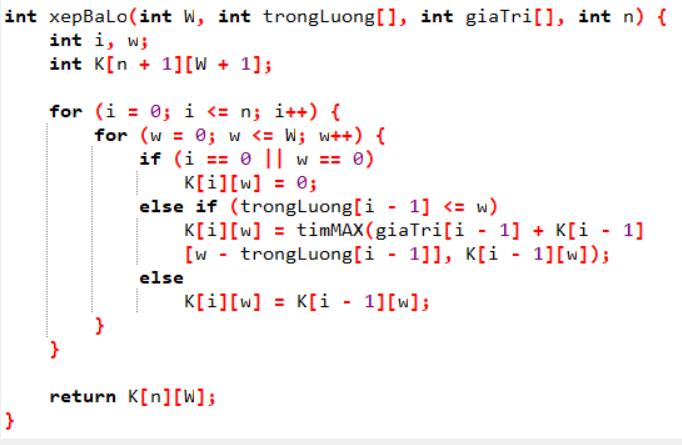
****

Code gồm 2 hàm chính:

* + Hàm timMAX: chứa 2 số nguyên a, b để tìm giá trị lớn nhất của vật.
  + Hàm xepBaLo: dùng để tính toán tổng giá trị lớn nhất của vật có thể xếp vào ba lô với các thông số là trọng lượng tối đa của ba lô W, số lượng vật n và mảng chứa trọng lượng và giá trị của vật

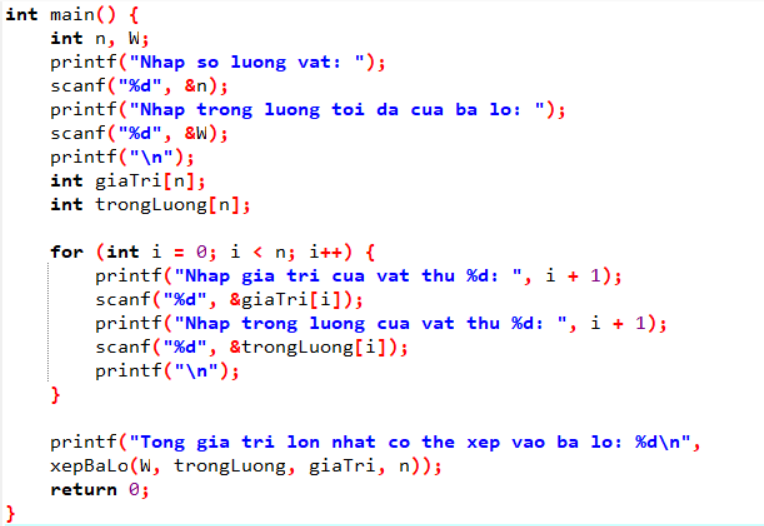
****

Hàm timMAX: tìm giá trị lớn nhất và trả về giá trị đó



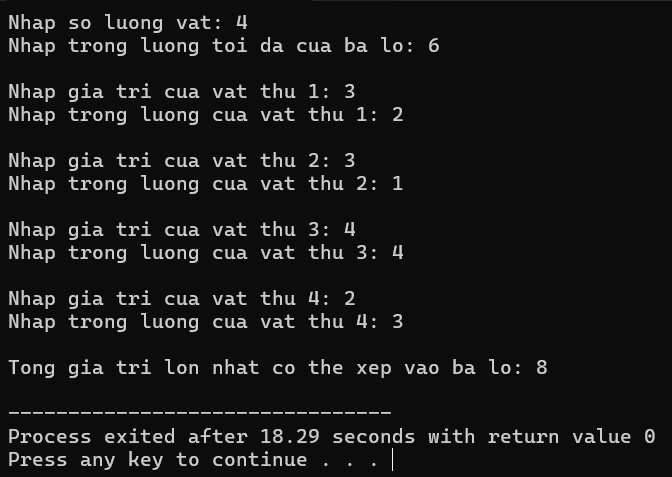
Hàm xepBaLo:

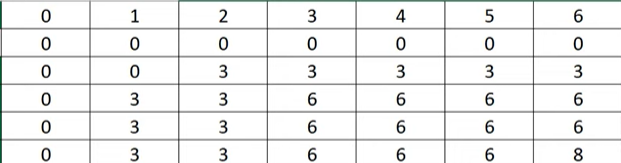
* + Tạo mảng 2 chiều K để lưu trữ giá trị tối đa của trọng lượng có thể đạt được rồi dùng vòng lặp for để duyệt qua từng vật và trọng lượng của chúng.
  + Tại mỗi ô của K, ta tính toán giá trị tối đa có thể đạt được 🡨 Phương pháp quy hoạch động
  + Nếu trọng lượng của vật hiện tại nhỏ hơn hoặc bằng trọng lượng đang xét, hàm tính toán giá trị tối đa giữa việc bỏ qua vật này hoặc bao gồm vật này và sử dụng giá trị tối đa từ phần còn lại của ba lô.
  + Nếu trọng lượng của vật hiện tại lớn hơn trọng lượng đang xét, hàm bỏ qua vật này và sử dụng giá trị tối đa từ phần còn lại của ba lô.
  + Cuối cùng ta trả về giá trị lớn nhất có thể đạt được, lưu ở vị trí cuối cùng trong bảng tương ứng với giá trị tối đa mà ba lô chứa được.



Hàm main: Cho phép nhập các giá trị của mảng và in kết quả

**4. Chạy chương trình:**

****

****

**III. Câu 3:**

**1. Đề bài:**

Lập trình cài đặt thuật toán nhánh - cận giải bài toán Người đi du lịch (TSP).

**2. Ý tưởng giải thuật:**

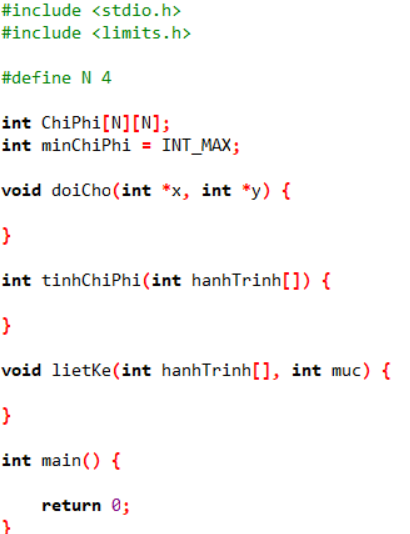
Bắt đầu bằng việc nhập ma trận chi phí từ bàn phím. Tiếp đến khởi tạo một biến để lưu chi phí nhỏ nhất rồi khởi tạo một mảng để lưu trữ hành trình du lịch.

Về thuật toán ta sử dụng một hàm đệ quy để thử tất cả các hành trình có thể. Hàm đệ quy này sẽ chọn lần lượt các thành phố để đi qua và tính toán chi phí của hành trình tương ứng.

Để tránh việc duyệt lại các hành trình đã xét, sử dụng kỹ thuật nhánh cận: Nếu chi phí hiện tại vượt quá chi phí nhỏ nhất, dừng việc duyệt và thoát khỏi hàm.

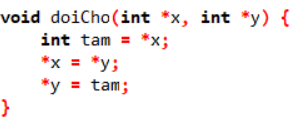
Khi duyệt qua tất cả các thành phố, kiểm tra chi phí của hành trình: Nếu nhỏ hơn chi phí nhỏ nhất thì cập nhật nó và lưu lại hành trình đó.

**3. Code chương trình:**

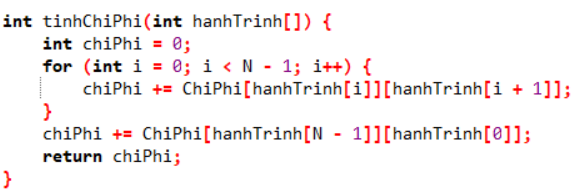
****

Code gồm 3 hàm chính:

* + Hàm doiCho: thực hiện đổi chỗ cho 2 con trỏ x, y
  + Hàm tinhChiPhi: tính tổng chi phí của 1 hành trình du lịch dựa trên 1 mảng các thành phố.
  + Hàm lietKe: Dùng phương pháp đệ quy để liệt kê tất cả các hành trình du lịch

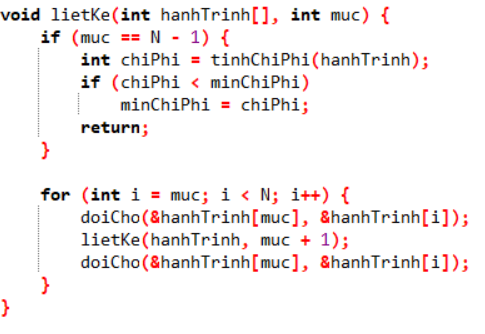


Hàm doiCho: đổi giá trị 2 biến sau này 2 con trỏ trỏ tới



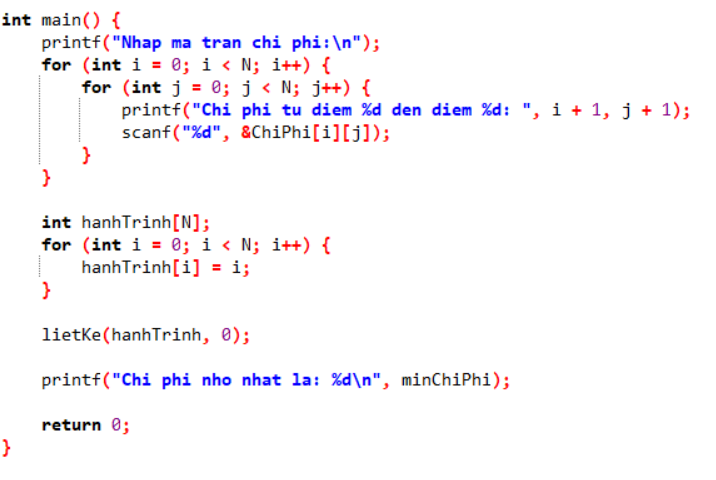
Hàm tinhChiPhi:

* Hàm tinhChiPhi tính tổng chi phí của hành trình du lịch dựa trên các chỉ số của các thành phố trong một mảng.
* Đầu tiên, hàm duyệt qua từng cặp thành phố liên tiếp trong hành trình và cộng thêm chi phí từ thành phố hiện tại đến thành phố kế tiếp.
* Cuối cùng, hàm cộng thêm chi phí từ thành phố cuối cùng trở về thành phốxuất phát và trả về tổng chi phí của hành trình.



Hàm lietKe:

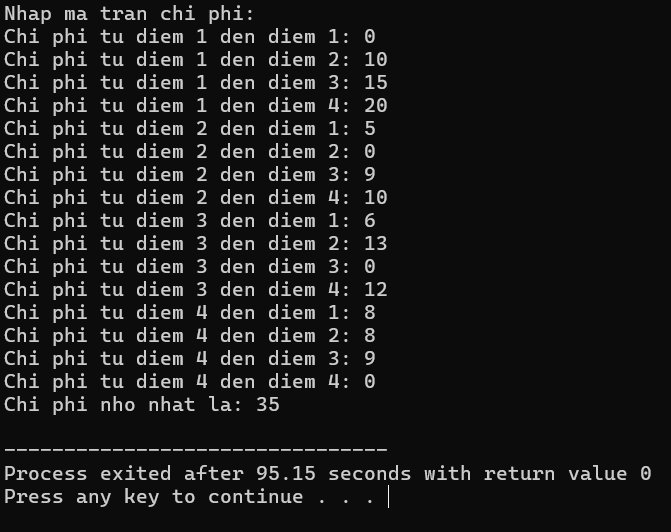
* Điều kiện dừng: Nếu muc bằng N - 1, tức là đã chọn tất cả các thành phố trong hành trình, ta tính chi phí của hành trình hiện tại bằng cách gọi hàm tinhChiPhi.
* Sau đó, so sánh chi phí này với minChiPhi. Nếu nhỏ hơn, ta cập nhật minChiPhi bằng giá trị mới này.
* Sau đó, hàm duyệt qua các lựa chọn để chọn thành phố tiếp theo trong hành trình. Đối với mỗi lựa chọn, ta thực hiện đổi chỗ vị trí các thành phố và tiếp tục đệ quy để tìm các hành trình con.
* Cuối cùng, sau khi quay lui từ hành trình con, trạng thái của mảng hanhTrinh được phục hồi để tiếp tục tìm kiếm các hành trình khác.



Hàm main:

* Cho phép nhập thông tin ma trận. Sử dụng 2 vòng lặp lồng nhau để cho phép nhập chi phí từ thành phố này sang thành phố khác rồi lưu vào mảng chiPhi
* Khởi tạo mảng hanhTrinh và gán giá trị ban đầu cho mỗi thành phố theo thứ tự từ 0 đến N-1. Mỗi thành phố sẽ có một chỉ số tương ứng trong mảng.
* In chi phí nhỏ nhất tìm được thông qua biến minChiPhi

**4. Chạy chương trình:**

****

**IV. Link Github:**

Xem toàn bộ code tại Github: [ngtuantu1202/63135967\_ToanRoiRac (github.com)](https://github.com/ngtuantu1202/63135967_ToanRoiRac)