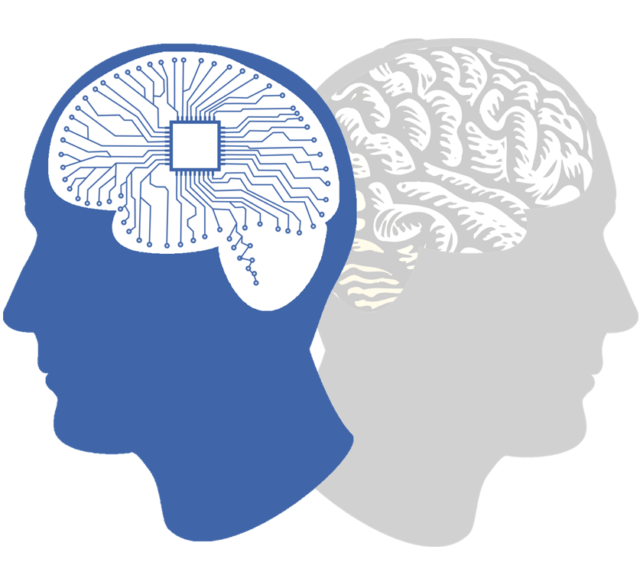
Khánh Hòa - 2022



BÁO CÁO THỰC HÀNH TRÍ TUỆ NHÂN TẠO

Sinh viên thực hiện: Võ Tấn Khuê

Mã số sinh viên: 61133814

Lớp: 61-CNTT2

# LỜI CẢM ƠN

Để hoàn thành báo cáo thực hành này, em xin gửi đến thầy Nguyễn Đình Cường, người đã tận tình hướng dẫn, giúp đỡ em hoàn thành báo cáo thực hành này lời cảm ơn sâu sắc nhất.

Mình cũng xin gửi lời cảm ơn tới các bạn sinh viên trong lớp trí tuệ nhân tạo Trường Đại học Nha Trang đã nhiệt tình chia sẻ những kinh nghiệm bài tập thực hành, hỗ trợ đóng góp trong quá trình làm báo cáo.

Trong quá trình nghiên cứu thực hiện báo cáo khó tránh khỏi sai sót, rất mong thầy bỏ qua. Đồng thời do kiến thức cũng như kinh nghiệm thực tiễn của bản thân còn hạn chế nên bài báo cáo này khó thể không tránh khỏi những thiếu sót, em rất mong nhận được ý kiến đóng góp từ quý thầy để em học hỏi thêm được nhiều kinh nghiệm, cũng như kỹ năng cần thiết.

Em xin chân thành cảm ơn!

# TÓM TẮT BÁO CÁO

Thuật toán có thể thực hiện được bằng máy tính, thường để giải quyết một lớp các vấn đề hoặc để thực hiện một phép tính. Trong bài báo cáo này em sẽ mô phỏng một số thuật toán dùng trong trí tuệ nhân tạo bằng ngôn ngữ lập trình.

Do khả năng của bản thân còn hạn chế nên các các bài mô phỏng thuật toán vẫn chưa hoàn thiện. Rất mong nhận được sự thông cảm và đóng góp ý kiến từ Quý Thầy/Cô và các bạn.

# MỤC LỤC

[LỜI CẢM ƠN 2](#_Toc111543137)

[TÓM TẮT BÁO CÁO 3](#_Toc111543138)

[MỤC LỤC 4](#_Toc111543139)

[Chương 1. CÁC PHƯƠNG PHÁP TÌM KIẾM HEURISTIC 8](#_Toc111543140)

[1.1 TÌM KIẾM THEO CHIỀU SÂU (DEPTH FIRST SEARCH) 8](#_Toc111543141)

[1.1.1 Tổng quan về DFS 8](#_Toc111543142)

[1.1.2 Code chương trình 8](#_Toc111543143)

[1.1.3 Kết quả chạy chương trình 9](#_Toc111543144)

[1.2 TÌM KIẾM THEO CHIỀU RỘNG (BREADTH FIRST SEARCH) 10](#_Toc111543145)

[1.2.1 Tổng quan về BFS 10](#_Toc111543146)

[1.2.2 Code chương trình 11](#_Toc111543147)

[1.2.3 Kết quả chạy chương trình 12](#_Toc111543148)

[1.3 TÌM KIẾM ƯU TIÊN TỐI ƯU (BEST-FIRST SEARCH) 13](#_Toc111543149)

[1.3.1 Thuật giải AKT 13](#_Toc111543150)

[1.3.1.1 Tổng quan 13](#_Toc111543151)

[1.3.1.2 Code chương trình 13](#_Toc111543152)

[1.3.1.3 Kết quả chạy chương trình 16](#_Toc111543153)

[1.3.2 Thuật giải A\* 17](#_Toc111543154)

[1.3.2.1 Tổng quan 17](#_Toc111543155)

[1.3.2.2 Code chương trình và cài đặt 18](#_Toc111543156)

[1.3.2.3 Kết quả chạy chương trình 20](#_Toc111543157)

[Chương 2. CÁC BÀI TOÁN TRÒ CHƠI 21](#_Toc111543158)

[2.1 ĐỔI TIỀN (VÉT CẠN VÀ HEURISTIC) 21](#_Toc111543159)

[2.1.1 Tổng quan 21](#_Toc111543160)

[2.1.2 Code chương trình 21](#_Toc111543161)

[2.1.3 Kết quả chạy chương trình 22](#_Toc111543162)

[2.2 TIC TAC TOE THUẬT TOÁN MINIMAX 22](#_Toc111543163)

[2.2.1 Tổng quan 22](#_Toc111543164)

[2.2.2 Code chương trình 23](#_Toc111543165)

[2.2.3 Kết quả chạy chương trình 23](#_Toc111543166)

[2.3 BÀI TOÁN TSP 24](#_Toc111543167)

[2.3.1 Tổng quan 24](#_Toc111543168)

[2.3.2 Code chương trình 24](#_Toc111543169)

[2.3.3 Kết quả chạy chương trình 26](#_Toc111543170)

[2.4 BÀI TOÁN QUA SÔNG 27](#_Toc111543171)

[2.4.1 Tổng quan 27](#_Toc111543172)

[2.4.2 Code chương trình 27](#_Toc111543173)

[2.4.3 Kết quả chạy chương trình 29](#_Toc111543174)

[2.5 XẾP HẬU VỚI HOÁN VỊ 30](#_Toc111543175)

[2.5.1 Tổng quan 30](#_Toc111543176)

[2.5.2 Code chương trình 31](#_Toc111543177)

[2.5.3 Kết quả chạy chương trình 31](#_Toc111543178)

[Chương 3. BIỂU DIỄN TRI THỨC 33](#_Toc111543179)

[3.1 CHỨNG MINH LOGIC MỆNH ĐỀ BẰNG PHƯƠNG PHÁP SỐ 33](#_Toc111543180)

[3.1.1 Tổng quan 33](#_Toc111543181)

[3.1.2 Code chương trình 33](#_Toc111543182)

[3.1.3 Kết quả chạy chương trình 33](#_Toc111543183)

[3.2 SUY DIỄN TIẾN 34](#_Toc111543184)

[3.2.1 Tổng quan 34](#_Toc111543185)

[3.2.2 Code chương trình 35](#_Toc111543186)

[3.2.3 Kết quả chạy chương trình 37](#_Toc111543187)

[3.3 SUY DIỄN LÙI 37](#_Toc111543188)

[3.3.1 Tổng quan 37](#_Toc111543189)

[3.3.2 Code chương trình 39](#_Toc111543190)

[3.3.3 Kết quả chạy chương trình 41](#_Toc111543191)

[3.4 THUẬT GIẢI VƯƠNG HẠO 43](#_Toc111543192)

[3.4.1 Tổng quan 43](#_Toc111543193)

[3.4.2 Code chương trình 43](#_Toc111543194)

[3.4.3 Kết quả chạy chương trình 43](#_Toc111543195)

[3.5 THUẬT GIẢI ROBINSON 44](#_Toc111543196)

[3.5.1 Tổng quan 44](#_Toc111543197)

[3.5.2 Code chương trình 44](#_Toc111543198)

[3.5.3 Kết quả chạy 45](#_Toc111543199)

[3.6 TAM GIÁC MẠNG NGỮ NGHĨA 46](#_Toc111543200)

[3.6.1 Tổng quan 46](#_Toc111543201)

[3.6.2 Code chương trình 46](#_Toc111543202)

[3.6.3 Kết quả chạy chương trình 46](#_Toc111543203)

[Chương 4. GIẢI THUẬT DI TRUYỀN 47](#_Toc111543204)

[4.1.1 Tổng quan 47](#_Toc111543205)

[4.1.2 Kết quả chạy chương trình 50](#_Toc111543206)

[Chương 5. BÀI TOÁN SỬ DỤNG PROLOG 50](#_Toc111543207)

[5.1 CON KHỈ VÀ NẢI CHUỐI 50](#_Toc111543208)

[5.1.1 Tổng quan 50](#_Toc111543209)

[5.1.2 Các bước thực hiện 51](#_Toc111543210)

[5.2 NGƯỜI QUA SÔNG 53](#_Toc111543211)

[5.2.1 Tổng quan 53](#_Toc111543212)

[5.2.2 Các bước thực hiện 54](#_Toc111543213)

[Chương 6. THUẬT GIẢI QUINLAND TRONG WEKA 54](#_Toc111543214)

[6.1 Tổng quan 54](#_Toc111543215)

[6.2 Các bước thực hiện 55](#_Toc111543216)

[Chương 7. VOICE TO SPEECH 59](#_Toc111543217)

[7.1 Các bước thực hiện 59](#_Toc111543218)

[7.2 Code chường trình 59](#_Toc111543219)

[7.3 Chạy 59](#_Toc111543220)

[Chương 8. IOT 59](#_Toc111543221)

[8.1 XÂY DỰNG HỆ THỐNG ĐÈN ĐƯỜNG THÔNG MINH 59](#_Toc111543222)

[8.1.1 Tổng quan 59](#_Toc111543223)

[8.1.2 Sơ đồ thiết kế 60](#_Toc111543224)

[8.1.3 Đặc điểm của linh kiện 60](#_Toc111543225)

[8.1.4 Code chương trình 60](#_Toc111543226)

[8.2 BẬT TẮT ĐÈN LED VỚI CẢM BIẾN HC-SR04 61](#_Toc111543227)

[8.2.1 Mô tả 61](#_Toc111543228)

[8.2.2 Sơ đồ thiết kế 61](#_Toc111543229)

[8.2.3 Đặc điểm của linh kiện 61](#_Toc111543230)

[8.2.4 Code chương trình 61](#_Toc111543231)

[8.3 SMART HOME MINI 62](#_Toc111543232)

[8.3.1 Mô tả 62](#_Toc111543233)

[8.3.2 Sơ đồ thiết kế 63](#_Toc111543234)

[8.3.3 Cấu hình mạng và luật 65](#_Toc111543235)

[Chương 9. MACHING LEARNING 68](#_Toc111543236)

[9.1 KNN 68](#_Toc111543237)

[9.1.1 Tổng quan 68](#_Toc111543238)

[9.1.2 Cách hoạt động 69](#_Toc111543239)

[9.1.3 Code chương trinh 69](#_Toc111543240)

[9.1.4 Kết quả chạy chương trình 70](#_Toc111543241)

[9.2 TRỢ LÝ ẢO (SPEECH RECONITION) 70](#_Toc111543242)

[9.2.1 Hướng dẫn cài đặt và sử dụng 70](#_Toc111543243)

[9.2.2 Kết quả thực hiện chương trình 70](#_Toc111543244)

[9.3 CAR NEURAL NETWORK 73](#_Toc111543245)

[9.3.1 Hướng dẫn cài đặt và sử dụng 73](#_Toc111543246)

[9.3.2 Kết quả thực hiện chương trình 74](#_Toc111543247)

[TÀI LIỆU THAM KHẢO 75](#_Toc111543248)

# CÁC PHƯƠNG PHÁP TÌM KIẾM HEURISTIC

## TÌM KIẾM THEO CHIỀU SÂU (DEPTH FIRST SEARCH)

### Tổng quan về DFS

Tìm kiếm ưu tiên chiều sâu hay tìm kiếm theo chiều sâu (tiếng Anh: Depth-first search - DFS) là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị. Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

Ý tưởng thuật toán:

1. DFS trên đồ thị vô hướng cũng giống như khám phá mê cung với một cuộn chỉ và một thùng sơn đỏ để đánh dấu, tránh bị lạc. Trong đó mỗi đỉnh s trong đồ thị tượng trưng cho một cửa trong mê cung.
2. Ta bắt đầu từ đỉnh s, buộc đầu cuộn chỉ vào s và đánh đấu đỉnh này "đã thăm". Sau đó ta đánh dấu s là đỉnh hiện hành u.
3. Bây giờ, nếu ta đi theo cạnh (u,v) bất kỳ.
4. Nếu cạnh (u,v) dẫn chúng ta đến đỉnh "đã thăm" v, ta quay trở về u.
5. Nếu đỉnh v là đỉnh mới, ta di chuyển đến v và lăn cuộn chỉ theo. Đánh dấu v là "đã thăm". Đặt v thành đỉnh hiện hành và lặp lại các bước.
6. Cuối cùng, ta có thể đi đến một đỉnh mà tại đó tất cả các cạnh kề với nó đều dẫn chúng ta đến các đỉnh "đã thăm". Khi đó, ta sẽ quay lui bằng cách cuộn ngược cuộn chỉ và quay lại cho đến khi trở lại một đỉnh kề với một cạnh còn chưa được khám phá. Lại tiếp tục quy trình khám phá như trên.
7. Khi chúng ta trở về s và không còn cạnh nào kề với nó chưa bị khám phá là lúc DFS dừng. [1]

### Code chương trình

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  int n, m;  //mang vector de luu danh sach ke  vector<int> adj[1001];  // kiem tra mot dinh da duoc tham hay chua  bool visited[1001];  void inp(){      cin >> n >> m;      for (int i = 0; i< m; i++)      {          int x, y;          cin >> x >> y;          adj[x].push\_back(y);          // co huong thi bo dong nay          adj[y].push\_back(x);      }      memset(visited, false, sizeof(visited));  }  void dfs(int u){      cout << u << " ";;      //danh dau la u da duoc tham      visited[u] = true;      //dung foreach de duyet tung diem ke cua U      for(int v : adj[u]){          //veu diem v chua dc tham thi          if(!visited[v]){              dfs(v);          }      }  }  int main(){      inp();      dfs(1);  } |

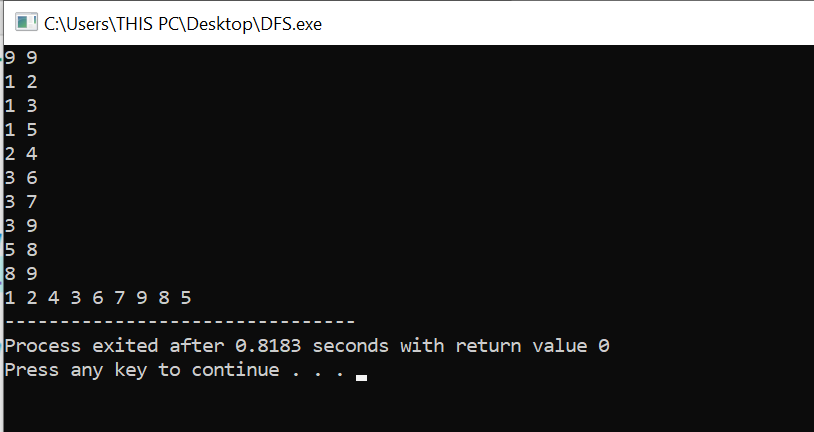
### Kết quả chạy chương trình

Ta có đồ thị như trong hình sau, ta sẽ duyệt theo chiều sâu bắt đầu từ đỉnh số 1.



Hình 1.1.1. Đồ thị gồm 9 đỉnh và 9 cạnh

Kết quả duyệt ta theo thứ tự là **1 2 4 3 6 7 9 8 5**



Hình 1.1.2. Kết quả cài đặt

## TÌM KIẾM THEO CHIỀU RỘNG (BREADTH FIRST SEARCH)

### Tổng quan về BFS

Trong lý thuyết đồ thị, tìm kiếm theo chiều rộng (BFS) là một thuật toán tìm kiếm trong đồ thị trong đó việc tìm kiếm chỉ bao gồm 2 thao tác: (a) cho trước một đỉnh của đồ thị; (b) thêm các đỉnh kề với đỉnh vừa cho vào danh sách có thể hướng tới tiếp theo. Có thể sử dụng thuật toán mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác.

Ý tưởng thuật toán:

Thuật toán sử dụng một cấu trúc dữ liệu hàng đợi để lưu trữ thông tin trung gian

1. Chèn đỉnh gốc vào hàng đợi (đang hướng tới)
2. Lấy ra đỉnh đầu tiên trong hàng đợi và quan sát nó

* Nếu đỉnh này chính là đỉnh đích, dừng quá trình tìm kiếm và trả về kết quả.
* Nếu không phải thì chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được quan sát trước đó vào hàng đợi.

1. Nếu hàng đợi là rỗng, thì tất cả các đỉnh có thể đến được đều đã được quan sát – dừng việc tìm kiếm và trả về "không thấy".
2. Nếu hàng đợi không rỗng thì quay về bước 2.

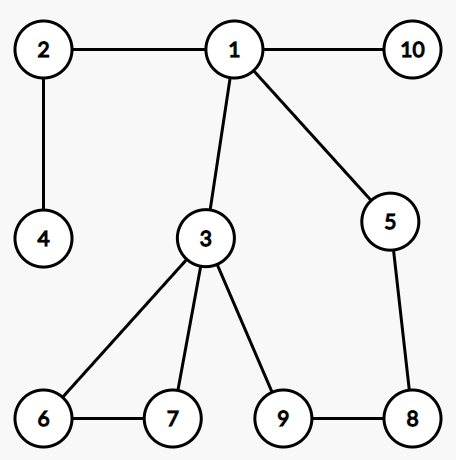
Ghi chú: Nếu sử dụng một ngăn xếp thay vì hàng đợi thì thuật toán trở thành thuật toán tìm kiếm theo chiều sâu. [2]

### Code chương trình

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  int n,m;  // mang vecto de luu danh sach ke  vector<int> adj[1001];  // nhap input va chuyen thanh ds ke  bool visited[1001];  void inp(){      cin >> n >> m;      for(int i=0; i<m; i++){          int x, y;          cin >> x >> y;          adj[x].push\_back(y);          adj[y].push\_back(x);      }      memset(visited, false, sizeof(visited));  }  void bfs(int u){      // Buoc khoi tao      queue<int> q;      q.push(u);      visited[u] = true;        // Buoc Lap      while(!q.empty()){          // lay dinh o dau hang doi          int v = q.front();          // lay ra va xoa no di          q.pop();          cout << v << " ";          for(int x : adj[v]){              //visited[x] == false              if(!visited[x]){                  q.push(x);                  visited[x] = true;              }          }      }}  int main(){      inp(); bfs(1); return 0;  } |

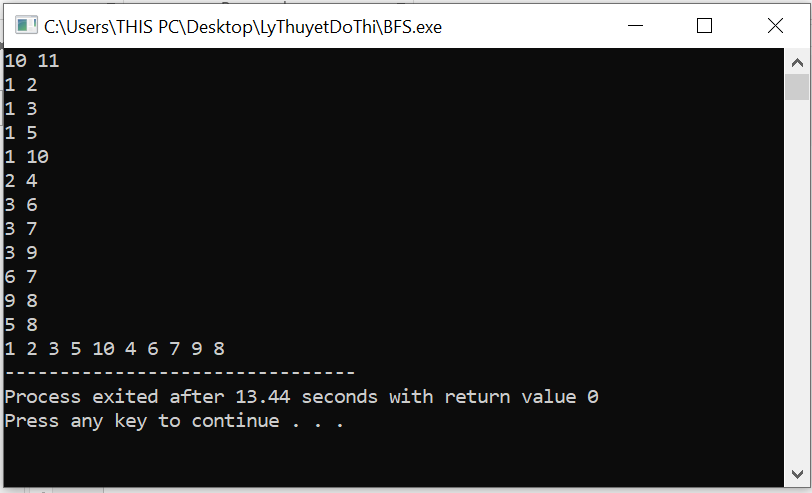
### Kết quả chạy chương trình

Ta có đồ thị như trong hình sau, ta sẽ duyệt theo chiều sâu bắt đầu từ đỉnh số 1.



Hình 1.2.1. Đồ thị gồm 10 đỉnh và 11 cạnh

Kết quả duyệt ta theo thứ tự là **1 2 3 5 10 4 6 7 9 8**



Hình 1.2.2. Kết quả cài đặt

## TÌM KIẾM ƯU TIÊN TỐI ƯU (BEST-FIRST SEARCH)

### Thuật giải AKT

#### Tổng quan

Thuật giải AKT mở rộng AT bằng cách sử dụng thêm thông tin ước lượng h’. Độ tốt của một trạng thái f là tổng của hai hàm g và h’.

***Phương pháp:***

1. Đặt OPEN chứa trạng thái khởi đầu.
2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện:
3. Chọn trạng thái (Tmax) có giá trị f nhỏ nhất trong OPEN (và xóa Tmax khỏi OPEN)
4. Nếu Tmax là trạng thái kết thúc thì thoát.
5. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện:

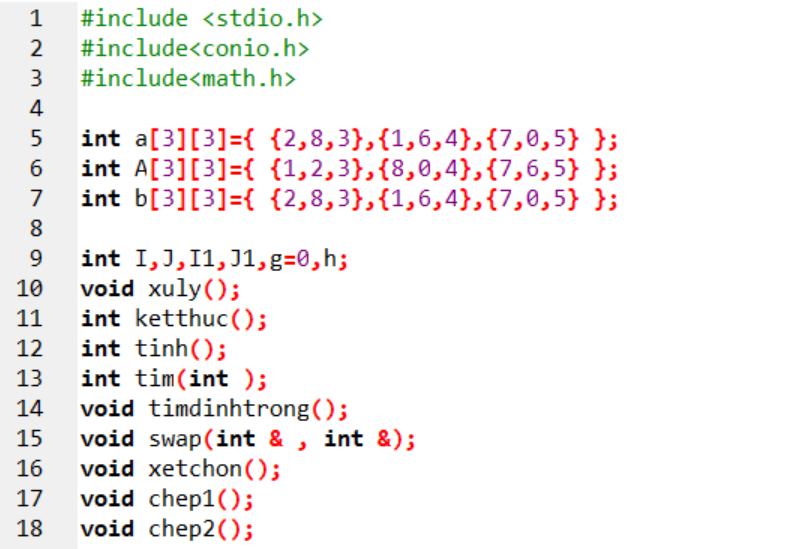
g(Tk) = g(Tmax) + cost(Tmax, Tk);

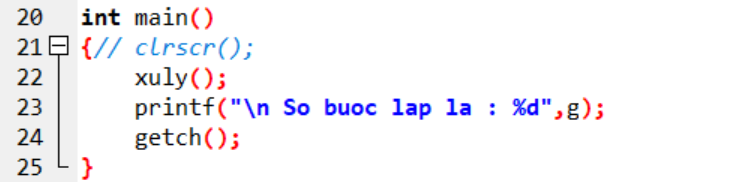
*Tính h’(Tk)*

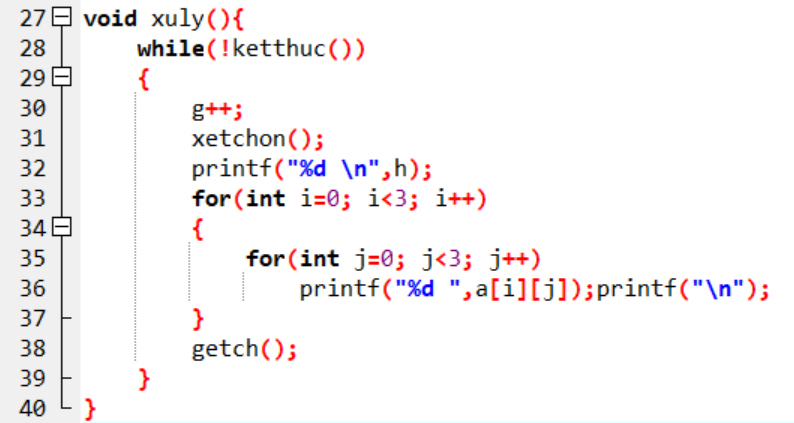
*f(Tk) = g(Tk) + h’(Tk);*

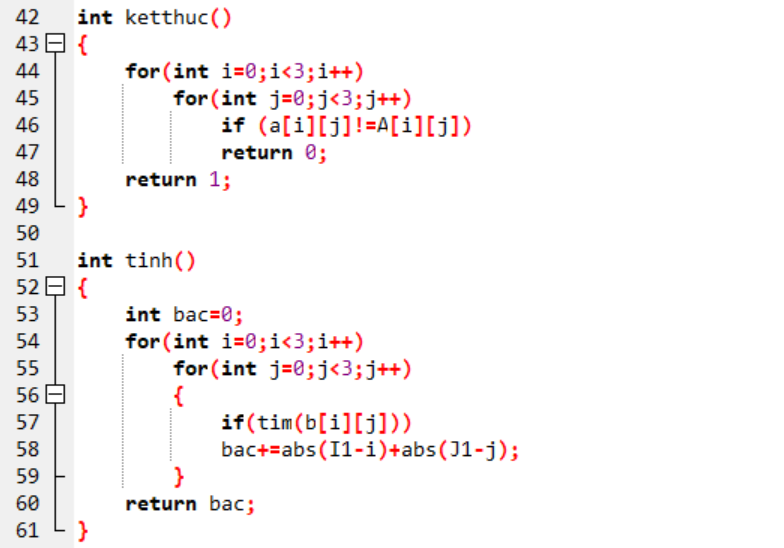
Thêm Tk vào OPEN.

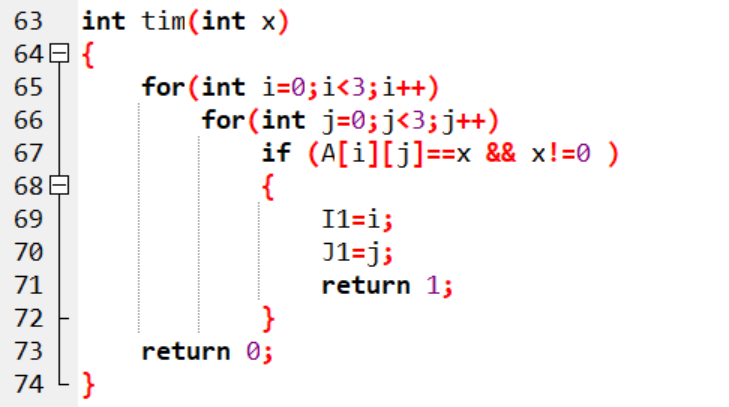
#### Code chương trình

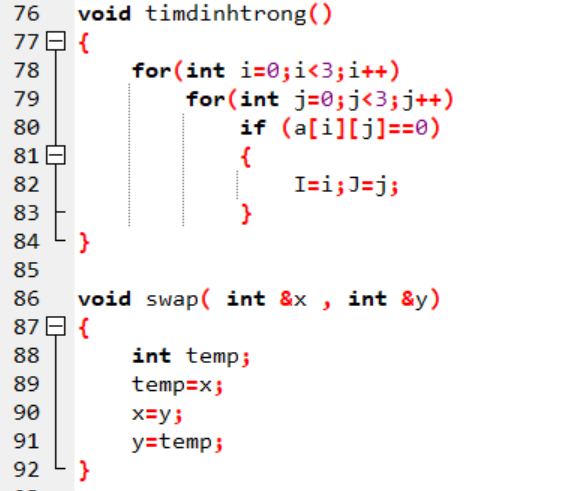


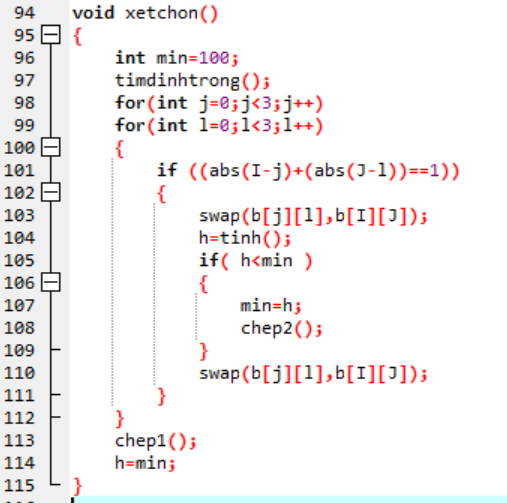


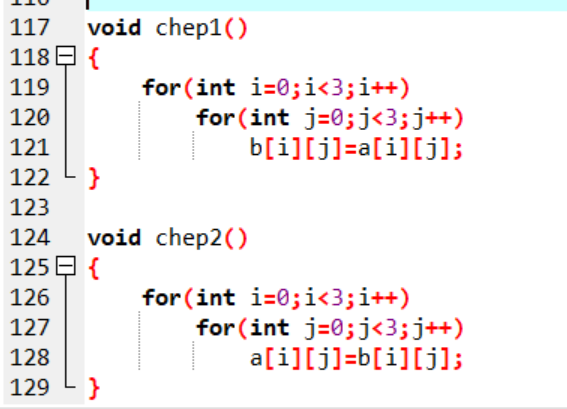






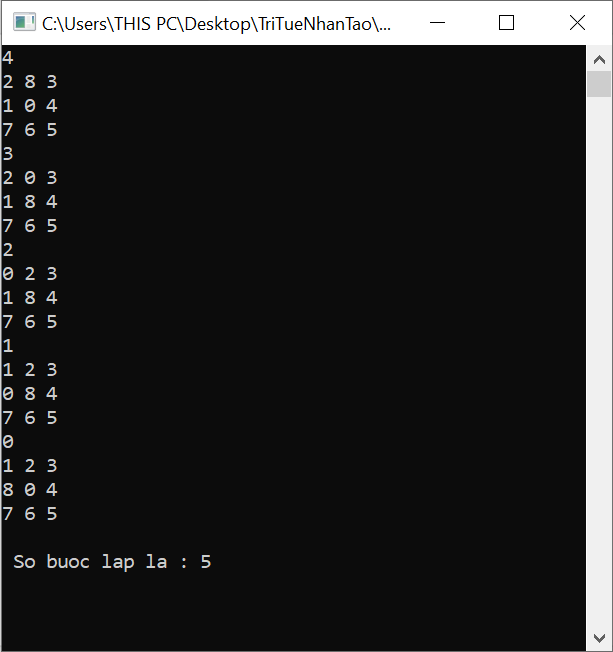






#### Kết quả chạy chương trình

Áp dụng giải thuật AKT vào bài toán sắp xếp các số trên ma trận 3x3 theo thứ tự nhất định..



Hình 1.3.1. Kết quả chạy chương trình

### Thuật giải A\*

#### Tổng quan

***Ý tưởng:*** Sử dụng hàm heuristics chấp nhận được, áp dụng tìm kiếm theo chiều rộng và loại bỏ những đường đi có chi phí cao. [3]

Hàm lượng giá: *f(u) = g(u) + h(u)*

Trong đó:

* *g(u)* là chi phí để đến u
* *h(u)* là lượng giá từ u đến đích
* *f(u)* là ước lượng tổng giá đến đích qua u

***Phương pháp:***

* A\* là một phiên bản đặc biệt của AKT áp dụng cho trường hợp đồ thị.
* Thuật giải A\* có sử dụng thêm tập hợp CLOSE để lưu trữ những trường hợp đã được xét đến.
* A\* mở rộng AKT bằng cách bổ sung cách giải quyết trường hợp khi "mở" một nút mà nút này đã có sẵn trong OPEN hoặc CLOSE.
* Khi xét đến một trạng thái Ti bên cạnh việc lưu trữ 3 giá trị cơ bản g, h’, f’ để phản ánh độ tốt của trạng thái đó, A\* còn lưu trữ thêm hai thông số sau:

1. Trạng thái cha của trạng thái Ti (ký hiệu là Cha(Ti))

2. Danh sách các trạng thái kế tiếp của Ti

***Mã giả:***

*Procedure A \*;*

*Begin*

1. *Khởi tạo danh sách L chỉ chứa trạng thái ban đầu*
2. *Loop do*
3. *if L rỗng then {thông báo thất bại; stop;}*
4. *Loại trạng thái u ở đầu danh sách L;*
5. *if u là trạng thái đích then {thông báo thành công; stop;}*
6. *for mỗi trạng thái v kề u do*

*{*

*g(v) g(u) + k(u, v);*

*f(v) g(v) + h(v);*

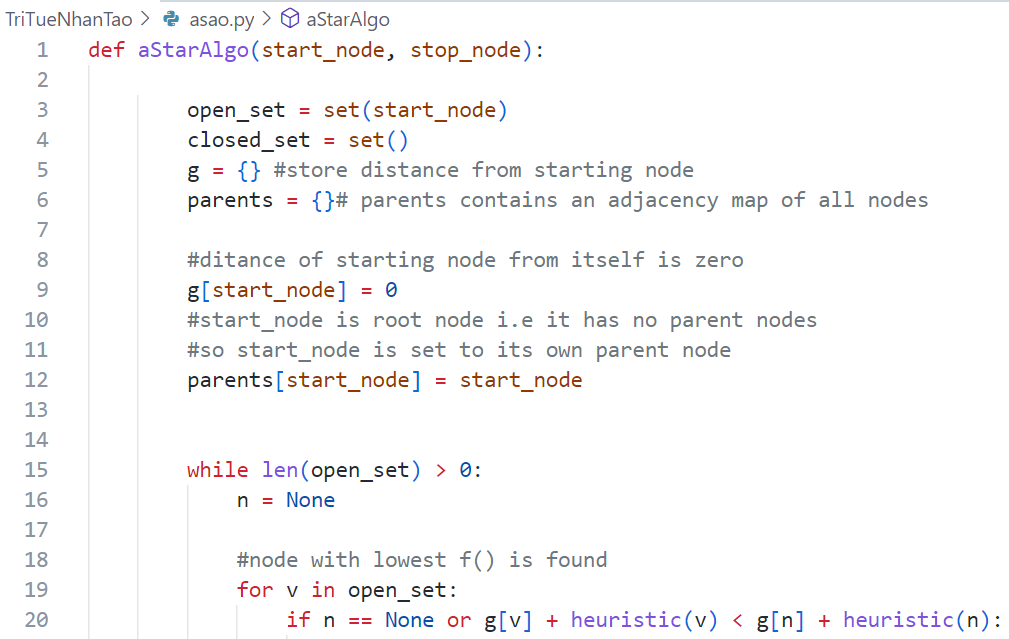
*Đặt v vào danh sách L;*

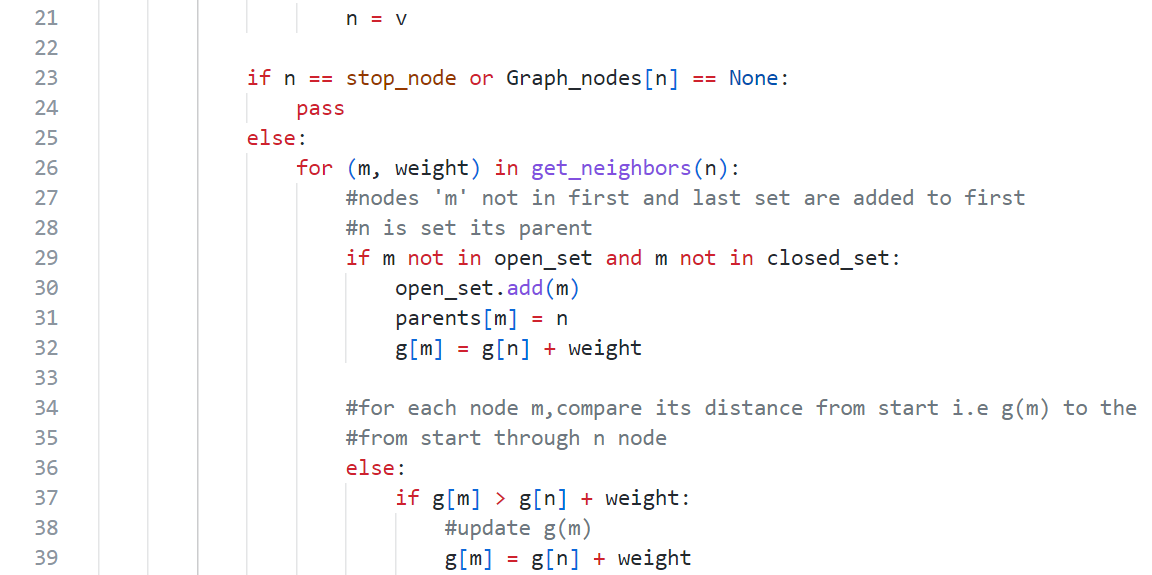
*}*

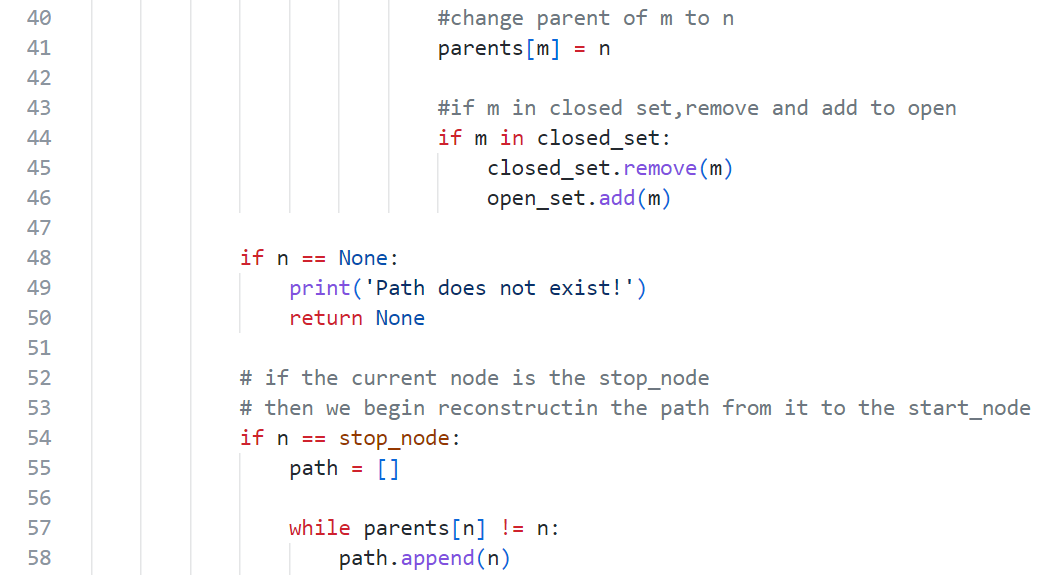
1. *Sắp xếp L theo thứ tự giảm dần của hàm f sao cho trạng thái có giá trị của hàm f nhỏ nhất ở đầu danh sách;*

*End;*

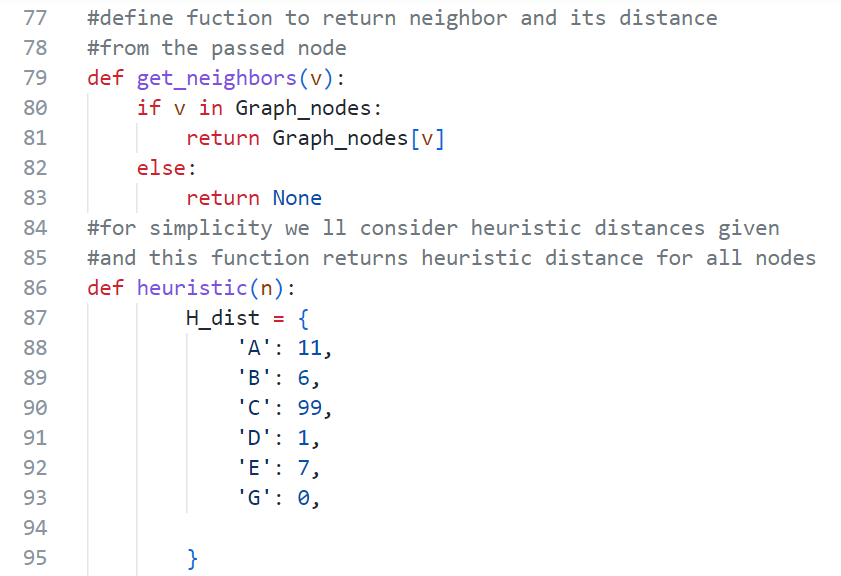
#### Code chương trình và cài đặt

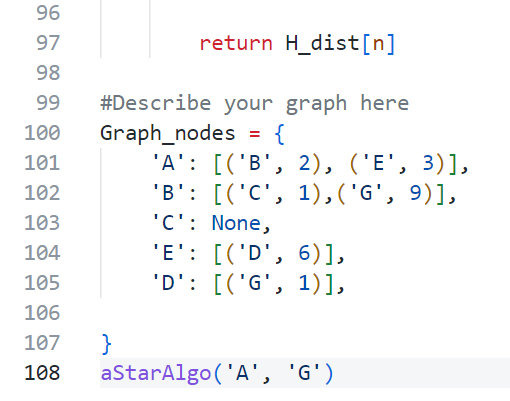






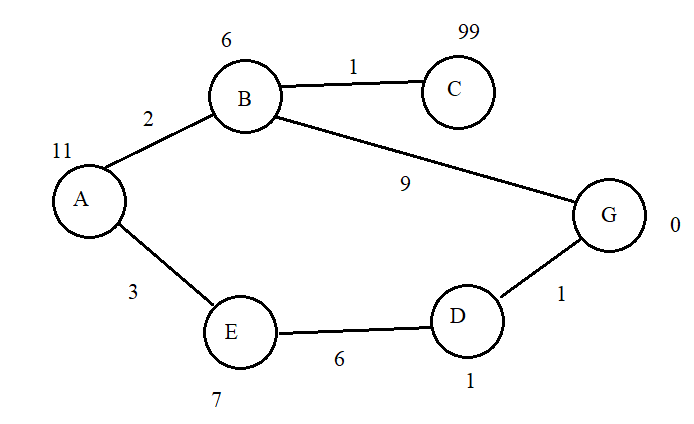






#### Kết quả chạy chương trình

Ta có hình đồ thị như sau:



Kết quả đường truyền dữ liệu tối ưu khi chạy thuật toán: A -> E -> D -> G



Hình 1.3.2. Kết quả chạy

***Nhận xét:***

Nếu không gian các trạng thái là hữu hạn và có giải pháp để tránh việc xét lặp lại các trạng thái thì giải thuật A \* là hoàn chỉnh (tìm được lời giải), nhưng không đảm bảo tối ưu. Nếu không gian các trạng thái là vô hạn thì giải thuật A \* có thể không cho ra lời giải.

# CÁC BÀI TOÁN TRÒ CHƠI

## ĐỔI TIỀN (VÉT CẠN VÀ HEURISTIC)

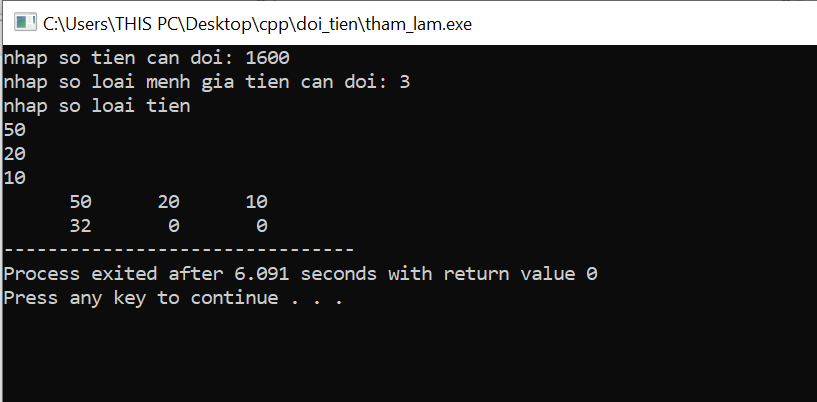
### Tổng quan

Yêu cầu: nhập 1 số n đại diện cho số tiền cần đổi và số lượng mệnh giá tiền cần đổi, loại tiền cần đổi dựa trên số lượng giá tiền cần đổi. Hãy viết chương trình cho phép đổi số tiền n thành các tờ tiền thuộc mệnh giá kể trên với số lượng tờ tiền là ít nhất.

### Code chương trình

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <algorithm>  using namespace std;  int main(int argc, char\*\* argv)  {      int n;      cout << "nhap so tien can doi: ";      cin >> n;      int i = 0, x = 8;      int smg;      cout << "nhap so loai menh gia tien can doi: ";      cin >> smg;      int sl;      int arr[100];      cout << "nhap so loai tien" << endl;      for (int i = 0; i < smg; i++)          cin >> arr[i];      sort(arr, arr + smg, greater<int>());      for (int i = 0; i < smg; i++)          cout << setw(x) << arr[i] << setw(x);      cout << endl;      while (n >= 0 && i != smg)      {          sl = n / arr[i];          n = n % arr[i];          i++;          cout << sl << setw(x);      }  } |

### Kết quả chạy chương trình



Hình 2.1.1. Kết quả chạy chương trình

## TIC TAC TOE THUẬT TOÁN MINIMAX

### Tổng quan

Tic-tac-toe là một trò chơi phổ biến dùng viết trên bàn cờ giấy có chín ô, 3x3. Hai người chơi, người dùng ký hiệu O, người kia dùng ký hiệu X, lần lượt điền ký hiệu của mình vào các ô. Người thắng là người thể tạo được đầu tiên một dãy ba ký hiệu của mình, ngang dọc hay chéo đều được.

Người Việt thường chơi trò tương tự, gọi là Cờ ca-rô, bàn cờ không giới hạn trong 9 ô, có thể vẽ thêm ô, để kéo rộng ra cho đến khi người nào đạt được một dãy 5 thì thắng cuộc. [4]

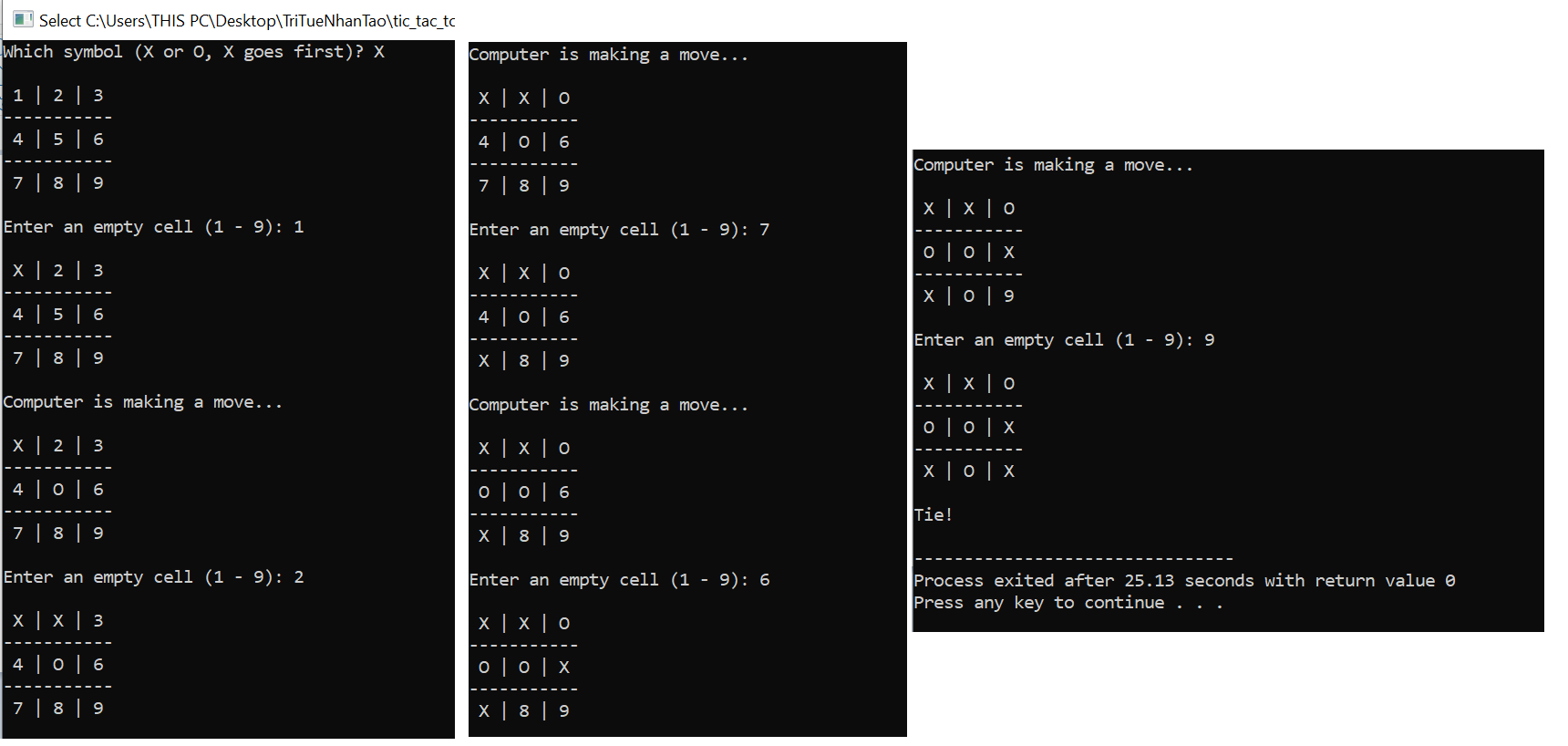


Hình 2.2.1. Bàn thắng thuộc về người O

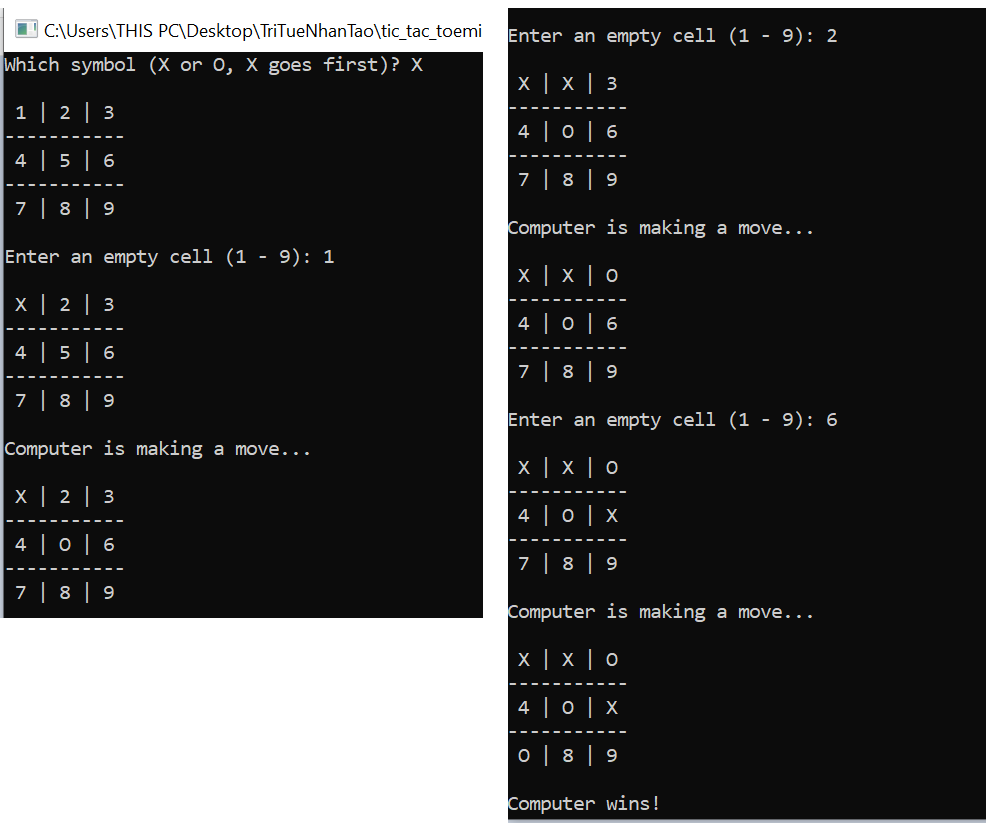
### Code chương trình

File tic\_tac\_toeminmax.cpp trong thư mục đính kèm.

### Kết quả chạy chương trình



Hình 2.2.2. Kết quả người và máy hòa



Hình 2.2.2. Kết quả khi người thua máy thắng

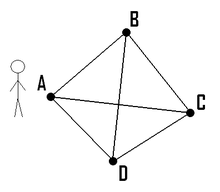
## BÀI TOÁN TSP

### Tổng quan

Bài toán người bán hàng (tiếng Anh: travelling salesman problem - TSP) là một bài toán NP-khó thuộc thể loại tối ưu rời rạc hay tổ hợp được nghiên cứu trong vận trù học hoặc lý thuyết khoa học máy tính. Bài toán được phát biểu như sau. Cho trước một danh sách các thành phố và khoảng cách giữa chúng, tìm chu trình ngắn nhất thăm mỗi thành phố đúng một lần.

Bài toán được nêu ra lần đầu tiên năm 1930 và là một trong những bài toán được nghiên cứu sâu nhất trong tối ưu hóa. Nó thường được dùng làm thước đo cho nhiều phương pháp tối ưu hóa. Mặc dù bài toán rất khó giải trong trường hợp tổng quát, có nhiều phương pháp giải chính xác cũng như heuristic đã được tìm ra để giải quyết một số trường hợp có tới hàng chục nghìn thành phố.

Ngay trong hình thức phát biểu đơn giản nhất, bài toán TSP đã có nhiều ứng dụng trong lập kế hoạch, hậu cần, cũng như thiết kế vi mạch. [5]

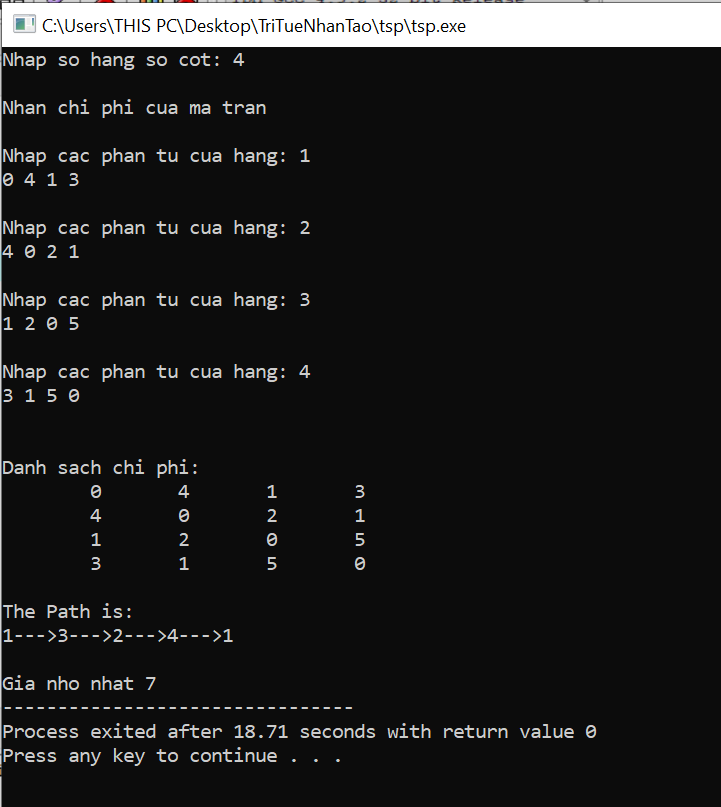


Nếu người bán hàng xuất phát từ điểm A, và nếu khoảng cách giữa hai điểm bất kì được biết thì đâu là đường đi ngắn nhất mà người bán hàng có thể thực hiện được sao cho đi hết tất cả các điểm mỗi điểm một lần để quay về lại điểm A ban đầu?

### Code chương trình

|  |
| --- |
| #include <iostream>  using namespace std;  int ary[10][10], completed[10], n, cost = 0;  void takeInput()  {      int i, j;      cout << "Nhap so hang so cot: ";      cin >> n;      cout << "\nNhan chi phi cua ma tran\n";      for (i = 0; i < n; i++)      {          cout << "\nNhap cac phan tu cua hang: " << i + 1 << "\n";          for (j = 0; j < n; j++)              cin >> ary[i][j];          completed[i] = 0;      }      cout << "\n\nDanh sach chi phi:";      for (i = 0; i < n; i++)      {          cout << "\n";          for (j = 0; j < n; j++)              cout << "\t" << ary[i][j];      }  }  int least(int c)  {      int i, nc = 999;      int min = 999, kmin;      for (i = 0; i < n; i++)      {          if ((ary[c][i] != 0) && (completed[i] == 0))              if (ary[c][i] + ary[i][c] < min)              {                  min = ary[i][0] + ary[c][i];                  kmin = ary[c][i];                  nc = i;              }      }      if (min != 999)          cost += kmin;      return nc;  }  void mincost(int city)  {      int i, ncity;      completed[city] = 1;      cout << city + 1 << "--->";      ncity = least(city);      if (ncity == 999)      {          ncity = 0;          cout << ncity + 1;          cost += ary[city][ncity];          return;      }      mincost(ncity);  }  int main()  {      takeInput();      cout << "\n\nThe Path is:\n";      mincost(0); // passing 0 because starting vertex      cout << "\n\nGia nho nhat " << cost;      return 0;  } |

### Kết quả chạy chương trình



Hình 2.3.1. Kết quả chạy thành công

## BÀI TOÁN QUA SÔNG

### Tổng quan

Người nông dân muốn vượt qua sông với một con sói, một con dê và một cái bắp cải. Ông chỉ có một chiếc thuyền nhỏ và nó chỉ có thể chở theo ông ta cùng một trong ba thứ: con sói, con dê hoặc cái bắp cải.

Vấn đề nằm ở chỗ nếu để sói và dê ở lại bờ, sói sẽ ăn thịt dê. Nếu để dê với bắp cải ở lại bờ, dê sẽ ăn bắp cải. Hãy tìm ra cách vận chuyển giúp người nông dân có thể mang được cả sói, dê và bắp cải qua sông. [6]



### Code chương trình

|  |
| --- |
| import time  names = {"N": "Nông dân",           "S": "Sói",           "C": "Cừu",           "B": "Bắp cải"}  forbidden\_states = [{"S", "C"}, {"C", "B"}, {"C", "B", "S"}]  def print\_story():      input("Nhấn enter để bắt đầu")  def clear():      print("\*" \* 60, "\n")  def print\_state(state):      left\_bank, right\_bank = state      print("#### QUA SÔNG ####")      print()      left\_bank\_display = [names[item] for item in left\_bank]      right\_bank\_display = [names[item] for item in right\_bank]      print(left\_bank\_display, "|", right\_bank\_display if right\_bank else "[]")      print()  def get\_move():      print("Bạn sẽ qua sông cùng với")      answer = ""      while answer.upper() not in ["N", "S", "C", "B"]:          answer = input("Nông dân (N), Sói (S), Cừu (C) hoặc Bắp cải (B)? ")      return answer.upper()  def process\_move(move, state):      temp\_state = [state[0].copy(), state[1].copy()]      containing\_set = 0 if move in state[0] else 1      if "N" not in state[containing\_set]:          print("Di chuyển sai")          print()          time.sleep(1)          return state      if containing\_set == 0:          temp\_state[0].difference\_update({move, "N"})          temp\_state[1].update([move, "N"])      elif containing\_set == 1:          temp\_state[1].difference\_update({move, "N"})          temp\_state[0].update([move, "N"])      if temp\_state[0] not in forbidden\_states and temp\_state[1] not in forbidden\_states:          state = [temp\_state[0].copy(), temp\_state[1].copy()]      else:          print("Di chuyển sai.")          print()          time.sleep(1)      print()      return state  def is\_win(state):      return state[1] == {"N", "S", "C", "B"}  def main():      left\_bank = {"N", "S", "C", "B"}      right\_bank = set()      state = [left\_bank, right\_bank]      print\_story()      while not is\_win(state):          clear()          print\_state(state)          move = get\_move()          state = process\_move(move, state)      print("Chúc mừng bạn qua sông thành công")  main() |

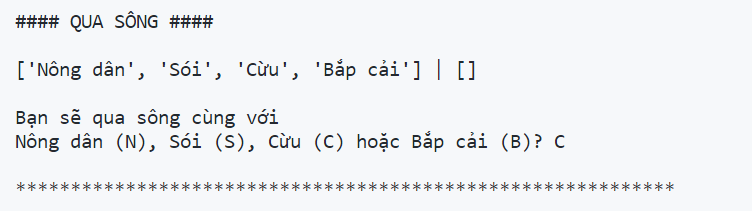
### Kết quả chạy chương trình

Nhân Enter để bắt đầu trò chơi

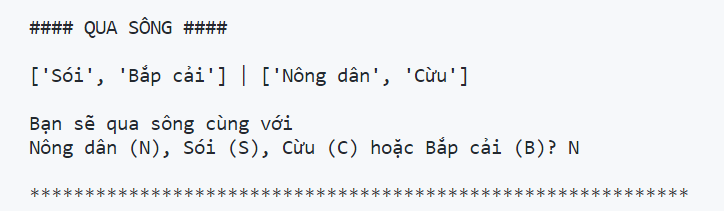


Thứ tự qua sông

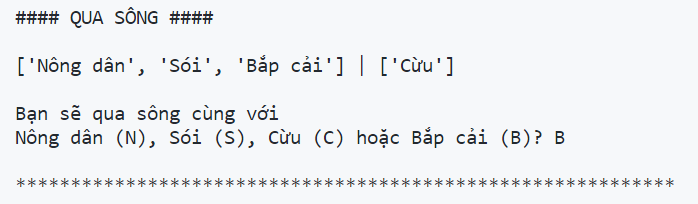
Cừu



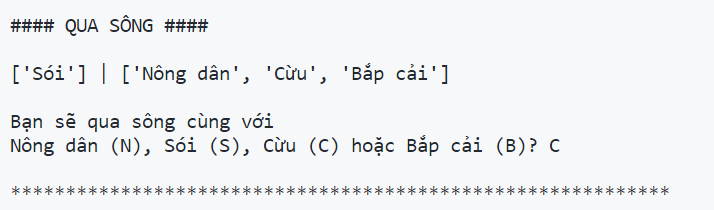
Nông dân



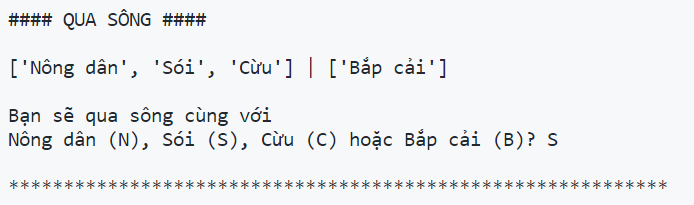
Bắp cải



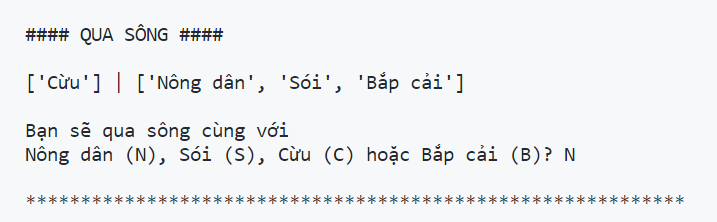
Cừu



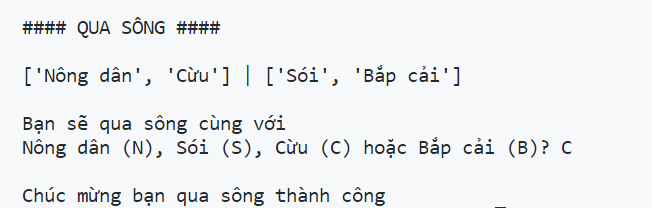
Sói



Nông dân



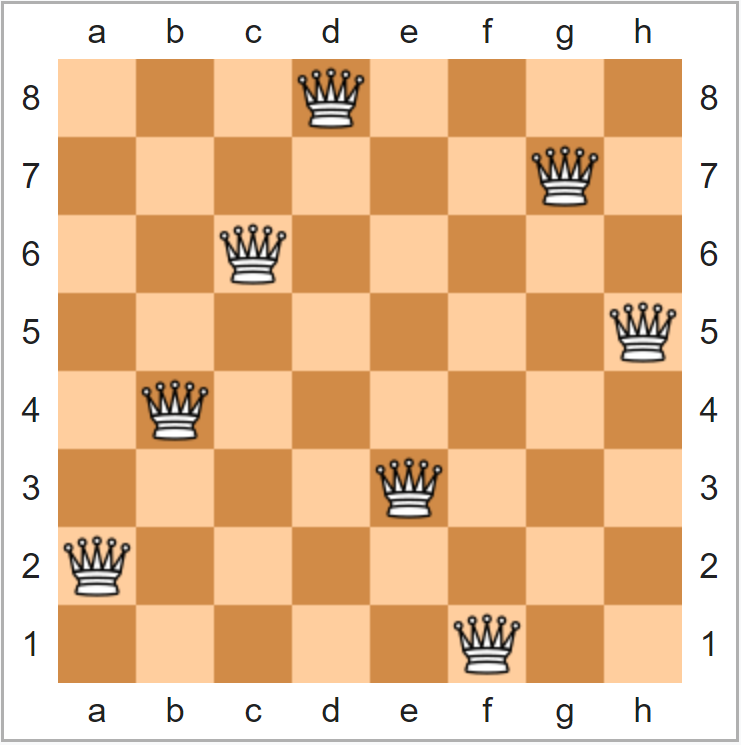
Kết quả



## XẾP HẬU VỚI HOÁN VỊ

### Tổng quan

Bài toán tám quân hậu là bài toán đặt tám quân hậu trên bàn cờ vua kích thước 8×8 sao cho không có quân hậu nào có thể "ăn" được quân hậu khác, hay nói khác đi không quân hậu nào có để di chuyển theo quy tắc cờ vua. Màu của các quân hậu không có ý nghĩa trong bài toán này. Như vậy, lời giải của bài toán là một cách xếp tám quân hậu trên bàn cờ sao cho không có hai quân nào đứng trên cùng hàng, hoặc cùng cột hoặc cùng đường chéo. Bài toán tám quân hậu có thể tổng quát hóa thành bài toán đặt n quân hậu trên bàn cờ n×n(n ≥ 4). [7]



Hình 2.5.1. Một trong 12 lời giải

### Code chương trình

|  |
| --- |
| from itertools import permutations  N = int(input("Nhập số quân hậu cần xếp: "))  sol = 0  cols = range(N)  for combo in permutations(cols):      if N == len(set(combo[i]+i for i in cols)) == len(set(combo[i]-i for i in cols)):          sol += 1          print('Solution '+str(sol)+': '+str(combo)+'\n')          print("\n".join(' o ' \* i + ' X ' + ' o ' \* (N-i-1)                for i in combo) + "\n\n\n\n") |

### Kết quả chạy chương trình

Lưu ý: Với n lớn thì phải mất rất nhiều thời gian để có thể tìm toàn bộ các cách xếp



Hình 2.5.2. Kết quả khi n=4

# BIỂU DIỄN TRI THỨC

## CHỨNG MINH LOGIC MỆNH ĐỀ BẰNG PHƯƠNG PHÁP SỐ

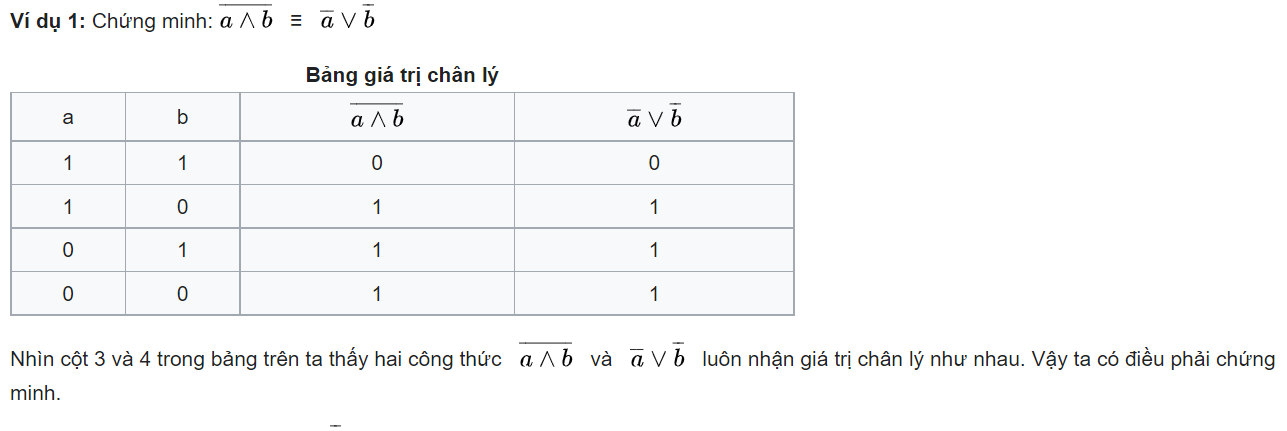
### Tổng quan

Trong logic toán, một phân ngành logic, cơ sở của mọi ngành toán học, mệnh đề, hay gọi đầy đủ là mệnh đề logic là một khái niệm nguyên thủy, không định nghĩa.

Thuộc tính cơ bản của một mệnh đề là giá trị chân lý của nó, được quy định như sau:

Mỗi mệnh đề có đúng một trong hai giá trị chân lý 0 hoặc 1. Mệnh đề có giá trị chân lý 1 là mệnh đề đúng, mệnh đề có giá trị chân lý 0 là mệnh đề sai.

Để chứng minh một đẳng thức trong logic mệnh đề ta thường dùng phương pháp lập bảng giá trị chân lý. [7]



Hình 2.3.1. Ví dụ bảng chân lý

### Code chương trình

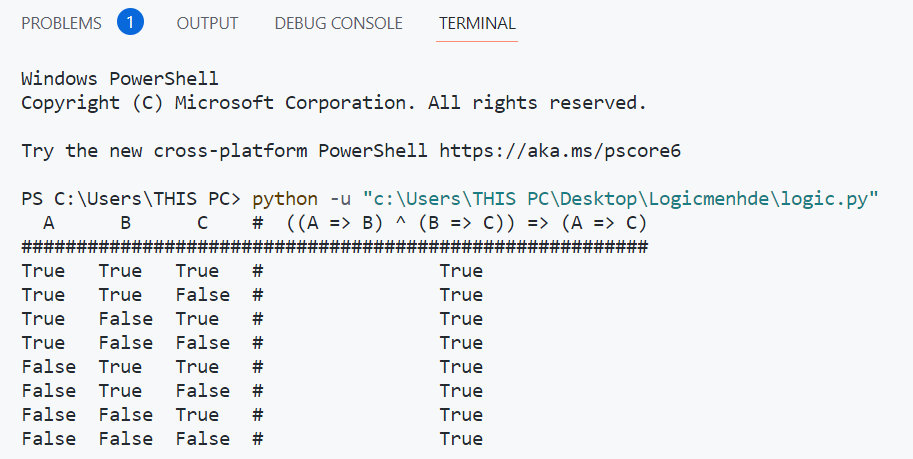
File logic.py trong thư mục đình kèm trong đó hàm chính

|  |
| --- |
| if \_\_name\_\_ == "\_\_main\_\_":      A,B,C = vars('A', 'B', 'C')      formula1 = (A >> B & B >> C) >> (A >> C)      formula1.print\_truth\_table()      formula1.is\_tautology() |

### Kết quả chạy chương trình

Chứng minh bài toán

((A => B) ^ (B => C)) => (A => C)



Hình 2.3.1. Kết quả chạy thành công

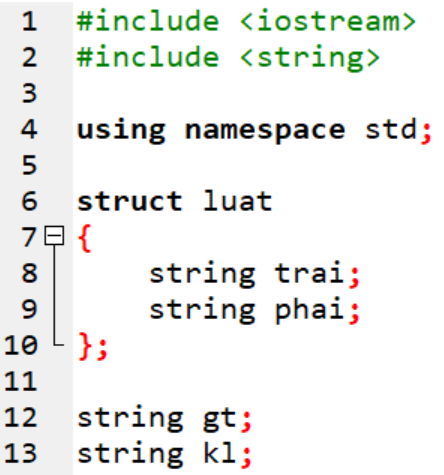
## SUY DIỄN TIẾN

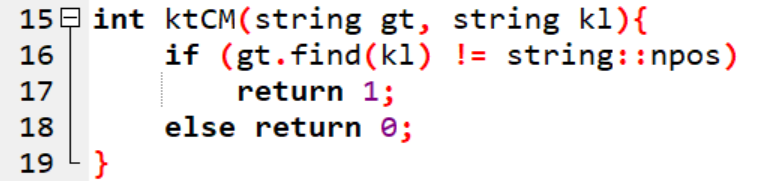
### Tổng quan

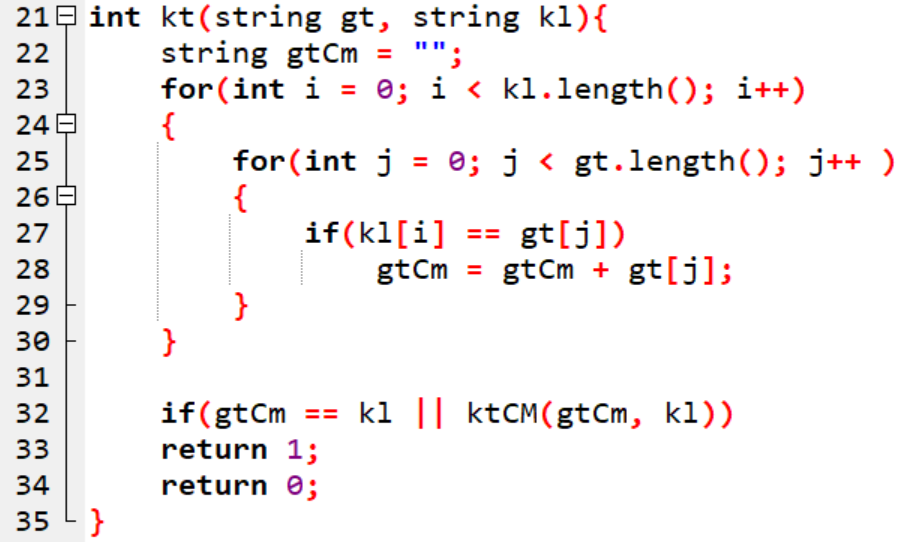
Table

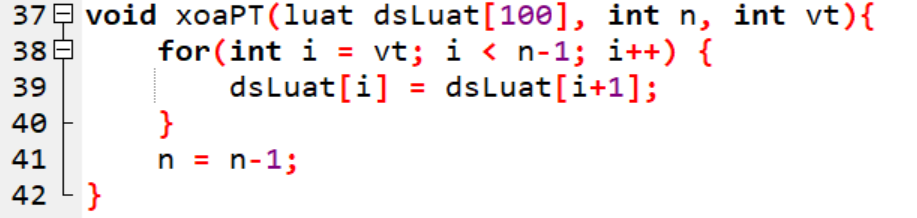
Description automatically generated

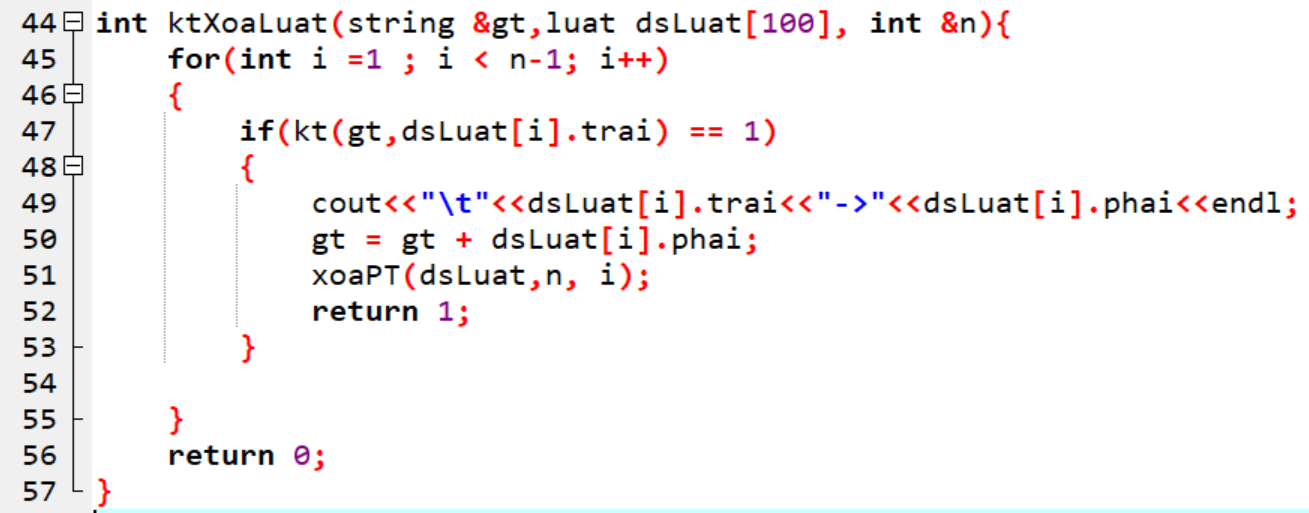
### Code chương trình

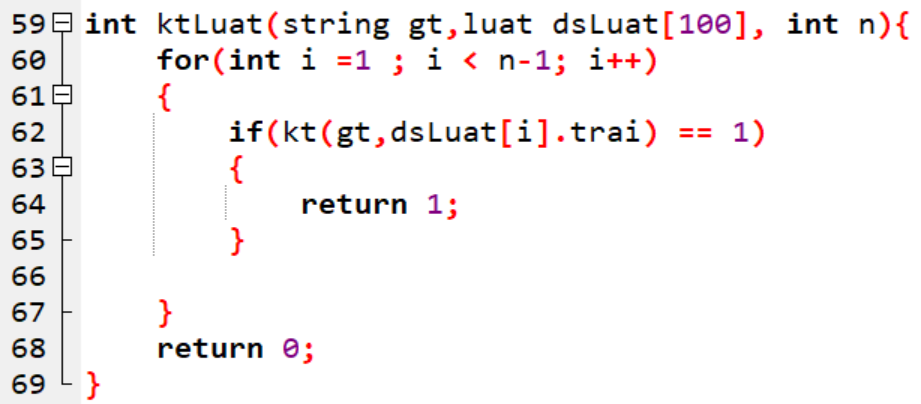


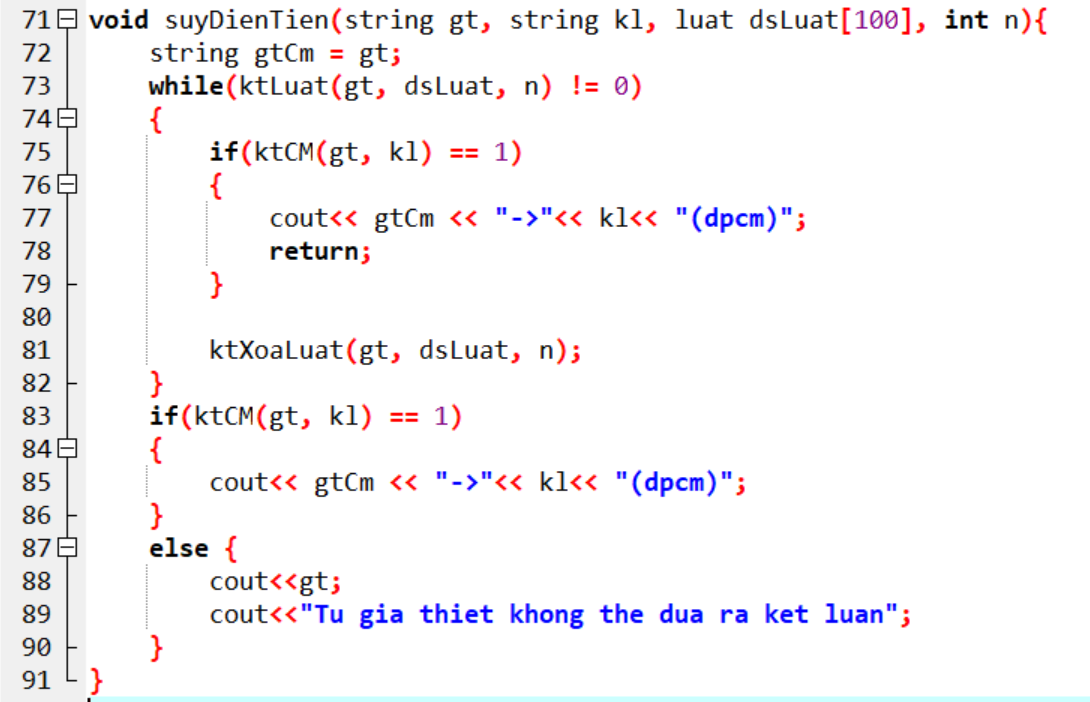


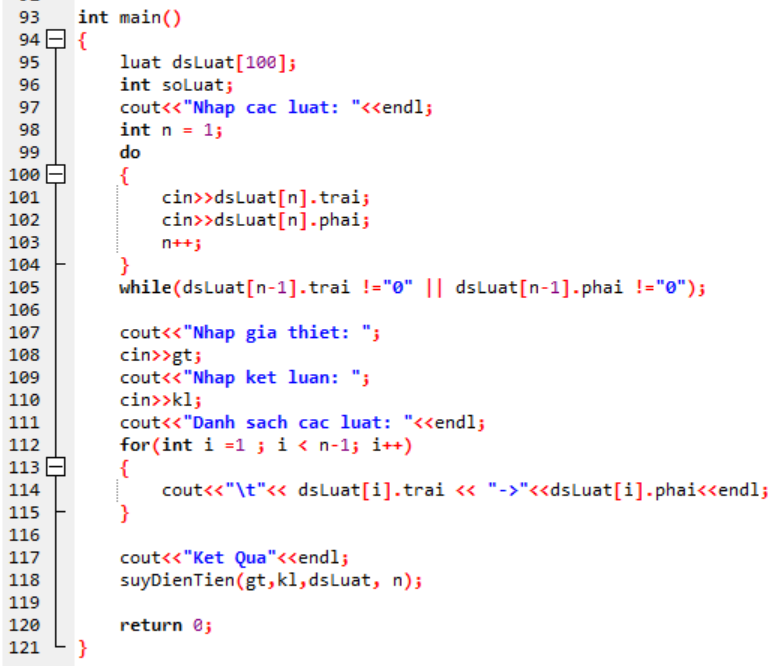












### Kết quả chạy chương trình

Ta có danh sách các luật như

(bc -> e), (a -> c), (e ->f) ,(a ->b)

Cần phải chứng minh

(a -> f)

Kết quả chạy chương trình như sau:

Chú ý 0 0 là dùng để yêu cầu đã đủ các luật



Hình 2.3.1. Kết quả chạy thành công

## SUY DIỄN LÙI

### Tổng quan

Chart, diagram

Description automatically generated

Suy diễn lùi : là quá trình suy luận ngược xuất phát từ một số sự kiện ban đầu, ta tìm kiếm các sự kiện đã "sinh" ra sự kiện này. Một ví dụ thường gặp trong thực tế là xuất phát từ các tình trạng của máy tính, chẩn đoán xem máy tính đã bị hỏng hóc ở đâu.

Ví dụ :

Tập các sự kiện :

Ổ cứng là "hỏng" hay "hoạt động bình thường"

Hỏng màn hình.

Lỏng cáp màn hình.

Tình trạng đèn ổ cứng là "tắt" hoặc "sáng"

Có âm thanh đọc ổ cứng.

Tình trạng đèn màn hình "xanh" hoặc "chớp đỏ"

Không sử dụng được máy tính.

Điện vào máy tính "có" hay "không"

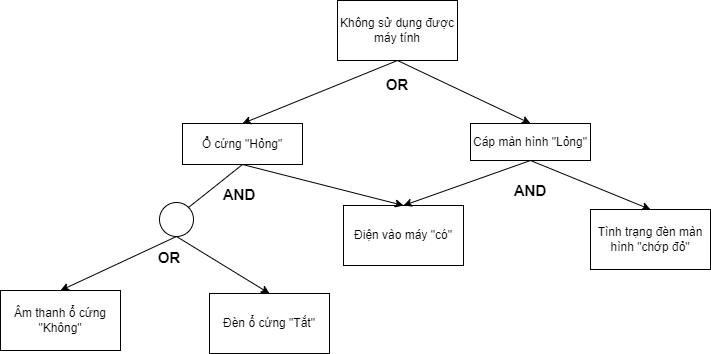
Tập các luật :

R1. Nếu ( (ổ cứng "hỏng") hoặc (cáp màn hình "lỏng")) thì không sử dụng được máy tính.

R2. Nếu (điện vào máy là "có") và ( (âm thanh đọc ổ cứng là "không") hoặc tình trạng đèn ổ cứng là "tắt")) thì (ổ cứng "hỏng").

R3. Nếu (điện vào máy là "có") và (tình trạng đèn màn hình là "chớp đỏ") thì (cáp màn hình "lỏng").

Để xác định được các nguyên nhân gây ra sự kiện "không sử dụng được máy tính", ta phải xây dựng một cấu trúc đồ thị gọi là đồ thị AND/OR như sau:



Như vậy là để xác định được nguyên nhân gây ra hỏng hóc là do ổ cứng hỏng hay cáp màn hình lỏng, hệ thống phải lần lượt đi vào các nhánh để kiểm tra các điều kiện như điện vào máy "có", âm thanh ổ cứng "không"…Tại một bước, nếu giá trị cần xác định không thể được suy ra từ bất kỳ một luật nào, hệ thống sẽ yêu cầu người dùng trực tiếp nhập vào. Chẳng hạn như để biết máy tính có điện không, hệ thống sẽ hiện ra màn hình câu hỏi "Bạn kiểm tra xem có điện vào máy tính không (kiểm tra đèn nguồn)? (C/K)". Để thực hiện được cơ chế suy luận lùi, người ta thường sử dụng ngăn xếp (để ghi nhận lại những nhánh chưa kiểm tra).

### Code chương trình

Một số hàm chính trong file suydienlui.py trong thư mục file đính kèm

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

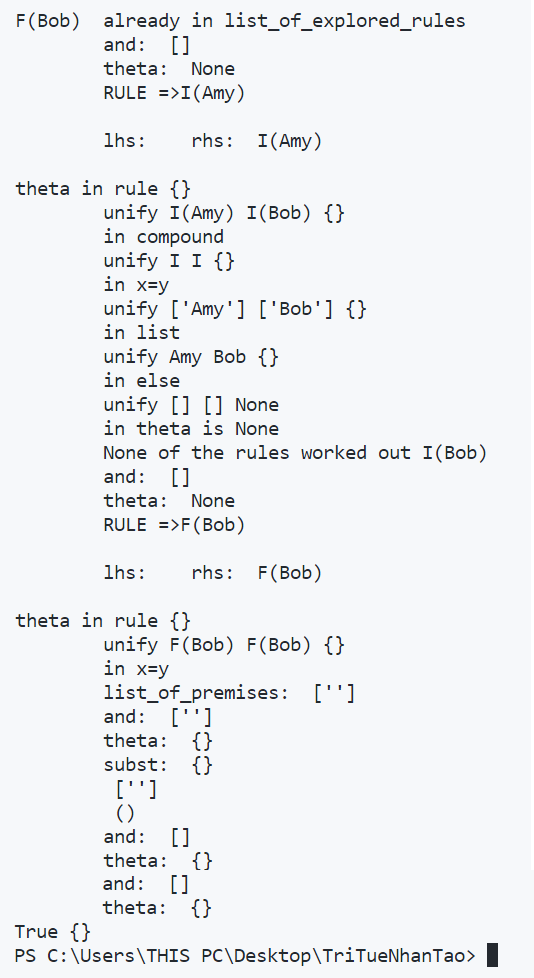
### Kết quả chạy chương trình

***Cài đặt:*** [8]

File input.txt

Ảnh có chứa văn bản

Mô tả được tạo tự động



Hình 3.3.1. Kết quả chạy thành công

## THUẬT GIẢI VƯƠNG HẠO

### Tổng quan

Text

Description automatically generated

Hình 3.4.1. Giải thuật Vương Hạo

### Code chương trình

File vuong\_hao.py trong Thư mục đính kèm

### Kết quả chạy chương trình

Hàm main

|  |
| --- |
| menhde1 = tuyen(phu(val('p')), val('q'))  menhde2 = tuyen(phu(val('q')), val('r'))  menhde3 = tuyen(phu(val('p')), val('r'))  mybieuthuc = bieuthuc([menhde1, menhde2], [menhde3])  VuongHao(mybieuthuc).run() |

Trong đó

* Khai báo các mệnh đề:  
  + menhde1 = tuyen(tuyen(phu(val('a')), phu(val('b'))), val('c'))

+ menhde2 = tuyen(tuyen(phu(val('b')), phu(val('c'))), val('d'))

+ menhde3 = tuyen(phu(val('a')), val('b'))

* Khai báo biểu thức: mybieuthuc = bieuthuc([menhde1, menhde2],[menhde3])
* Chạy thuật toán cần chứng minh: VuongHao(mybieuthuc).run()



Hình 3.4.1. Kết quả chạy thành công

## THUẬT GIẢI ROBINSON

### Tổng quan

Text

Description automatically generated

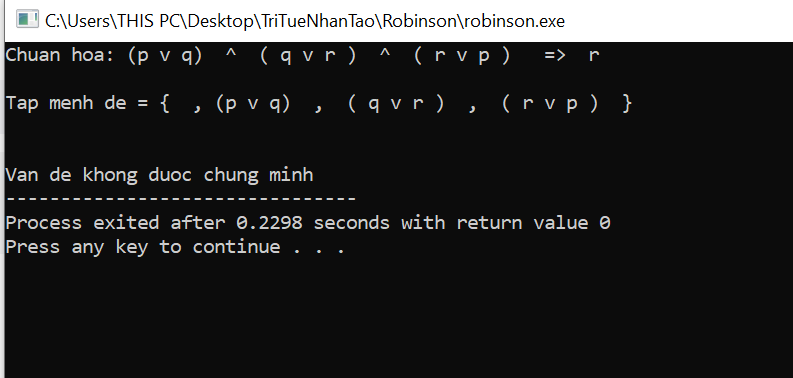
Hình 3.5.1. Giải thuật Robinson

### Code chương trình

File robinson.cpp trong Thư mục đính kèm

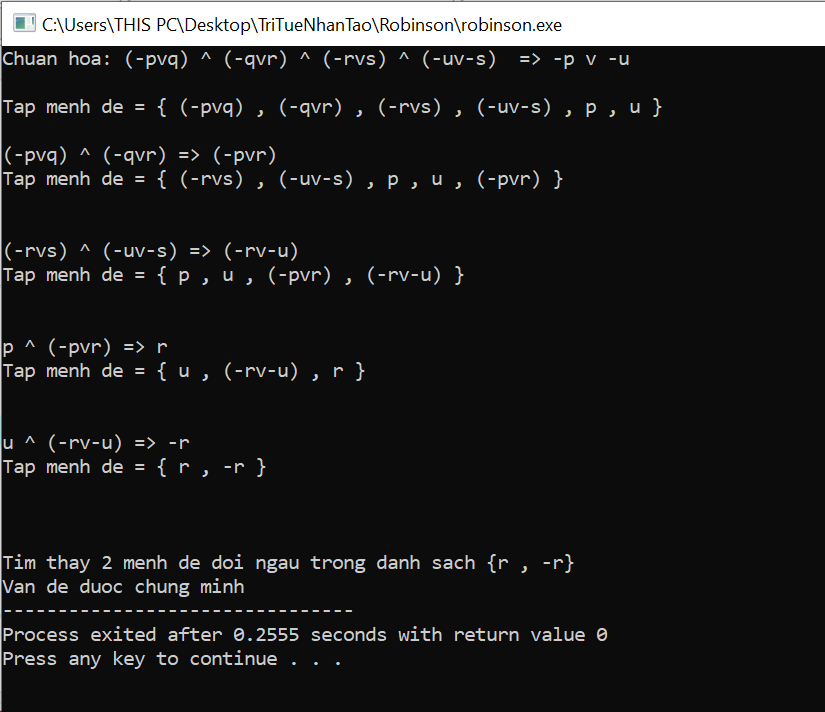
### Kết quả chạy

(p v q) ^ ( q v r ) ^ ( r v p ) > r



Hình 3.4.1. Kết quả chạy thành công

(-pvq)^(-qvr)^(-rvs)^(-uv-s)>-pv-u



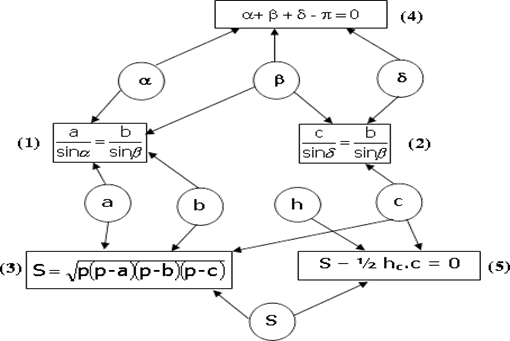
Hình 3.4.1. Kết quả chạy thành công

## TAM GIÁC MẠNG NGỮ NGHĨA

### Tổng quan

Có 22 yếu tố liên quan đến cạnh và góc của tam giác. Để xác định một tam giác thì ta phải có 3 yếu tố. Trong đó phải có yếu tố cạnh. Như vậy có khoảng vài ngàn cách để xây dựng hay xác định một tam giác. Theo thống kê thì có khoảng trên 200 công thức liên quan đến cạnh và góc của tam giác.

Để giải bài toán này bằng mạng ngữ nghĩa, ta phải sử dụng khoảng 200 đỉnh để chứa công thức và khoảng 22 đỉnh để chứa các yếu tố của tam giác. [10]



### Code chương trình

File tamgiacmangngunghia.py trong thư mục đính kèm

### Kết quả chạy chương trình



Hình 3.4.1. Kết quả chạy thành công

# GIẢI THUẬT DI TRUYỀN

### Tổng quan

GAs là một kỹ thuật tối ưu hóa hoặc tìm kiếm giải pháp heuristic, ban đầu được thúc đẩy bởi nguyên tắc tiến hóa của Darwin thông qua chọn lọc (di truyền). GA sử dụng một phiên bản trừu tượng cao của các quy trình tiến hóa để đưa ra các giải pháp cho các vấn đề đã cho. Mỗi GA hoạt động trên một quần thể nhiễm sắc thể nhân tạo (artificial chromosomes). Đây là những chuỗi trong một bảng chữ cái hữu hạn (thường là nhị phân). Mỗi nhiễm sắc thể đại diện cho một giải pháp cho một vấn đề và có một thích hợp, một số thực là thước đo mức độ tốt của một giải pháp cho một vấn đề cụ thể.

GA được xây dựng từ một số thành phần riêng biệt. Đây là một điểm mạnh đặc biệt vì nó có nghĩa là các thành phần tiêu chuẩn có thể được sử dụng lại, với sự thích ứng nhỏ trong nhiều GA khác nhau, do đó, việc triển khai dễ dàng hơn. Các thành phần chính là mã hóa nhiễm sắc thể (chromosome encoding), hàm thích hợp (the fitness function), chọn lọc (selection), tái tổ hợp (recombination) và sơ đồ tiến hóa (the evolution scheme).

Một thiết kế điển hình cho GA cổ điển sử dụng thay thế hoàn toàn bằng các toán tử di truyền chuẩn có thể như sau:

(1) Tạo ngẫu nhiên một quần thể nguồn ban đầu của P nhiễm sắc thể.

(2) Tính thể trạng của từng nhiễm sắc thể c trong quần thể nguồn bằng hàm F(c).

(3) Tạo một quần thể kế thừa trống và sau đó lặp lại các bước sau cho đến khi tạo ra P nhiễm sắc thể.

(a) Sử dụng phương pháp chọn lọc theo tỉ lệ thuận, chọn hai nhiễm sắc thể là c1 và c2 từ quần thể nguồn.

(b) Áp dụng trao đổi chéo một điểm với c1 và c2 với tỷ lệ trao đổi chéo pc để thu được nhiễm sắc thể con c.

(c) Áp dụng đột biến đồng hợp với c có tỷ lệ đột biến pm để tạo ra c’.

(d) Thêm c’ vào quần thể kế thừa.

(4) Thay thế quần thể nguồn bằng quần thể kế thừa.

(5) Nếu không gặp điều kiện dừng, quay lại Bước 2.

<https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/>



|  |
| --- |
| # genetic algorithm search of the one max optimization problem  from numpy.random import randint  from numpy.random import rand    # objective function  def onemax(x):      return -sum(x)    # tournament selection  def selection(pop, scores, k=3):      # first random selection      selection\_ix = randint(len(pop))      for ix in randint(0, len(pop), k-1):          # check if better (e.g. perform a tournament)          if scores[ix] < scores[selection\_ix]:              selection\_ix = ix      return pop[selection\_ix]    # crossover two parents to create two children  def crossover(p1, p2, r\_cross):      # children are copies of parents by default      c1, c2 = p1.copy(), p2.copy()      # check for recombination      if rand() < r\_cross:          # select crossover point that is not on the end of the string          pt = randint(1, len(p1)-2)          # perform crossover          c1 = p1[:pt] + p2[pt:]          c2 = p2[:pt] + p1[pt:]      return [c1, c2]    # mutation operator  def mutation(bitstring, r\_mut):      for i in range(len(bitstring)):          # check for a mutation          if rand() < r\_mut:              # flip the bit              bitstring[i] = 1 - bitstring[i]    # genetic algorithm  def genetic\_algorithm(objective, n\_bits, n\_iter, n\_pop, r\_cross, r\_mut):      # initial population of random bitstring      pop = [randint(0, 2, n\_bits).tolist() for \_ in range(n\_pop)]      # keep track of best solution      best, best\_eval = 0, objective(pop[0])      # enumerate generations      for gen in range(n\_iter):          # evaluate all candidates in the population          scores = [objective(c) for c in pop]          # check for new best solution          for i in range(n\_pop):              if scores[i] < best\_eval:                  best, best\_eval = pop[i], scores[i]                  print(">%d, new best f(%s) = %.3f" % (gen,  pop[i], scores[i]))          # select parents          selected = [selection(pop, scores) for \_ in range(n\_pop)]          # create the next generation          children = list()          for i in range(0, n\_pop, 2):              # get selected parents in pairs              p1, p2 = selected[i], selected[i+1]              # crossover and mutation              for c in crossover(p1, p2, r\_cross):                  # mutation                  mutation(c, r\_mut)                  # store for next generation                  children.append(c)          # replace population          pop = children      return [best, best\_eval]    # define the total iterations  n\_iter = 100  # bits  n\_bits = 20  # define the population size  n\_pop = 100  # crossover rate  r\_cross = 0.9  # mutation rate  r\_mut = 1.0 / float(n\_bits)  # perform the genetic algorithm search  best, score = genetic\_algorithm(onemax, n\_bits, n\_iter, n\_pop, r\_cross, r\_mut)  print('Done!')  print('f(%s) = %f' % (best, score)) |

### Kết quả chạy chương trình



Hình 3.4.1. Kết quả chạy thành công

# BÀI TOÁN SỬ DỤNG PROLOG

## CON KHỈ VÀ NẢI CHUỐI

### Tổng quan

Trong phòng có con khỉ, chiếc ghế và nải chuối…

Có thể giải bài toán nhờ vào các vị từ:

1. in\_room(X) : có X trong phòng

2. dexterous(X) : X là loài khéo léo

3. tall(X) : X thì cao

4. can\_climb(X,Y) : X có thể leo lên Y

5. can\_move(X,Y,Z) : X có thể di chuyển Y đền gần Z

6. get\_on(X,Y) : X đứng trên Y

7. under(X,Y) : X ở dưới Y

8. close(X,Y) : X ở gần Y

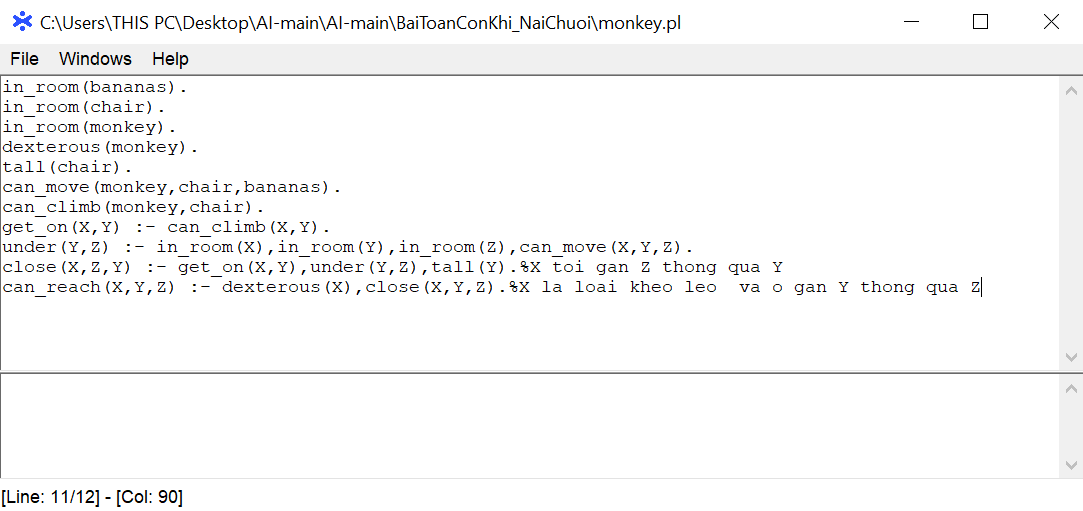
9. can\_reach(X,Y) : X có thể cầm lấy Y

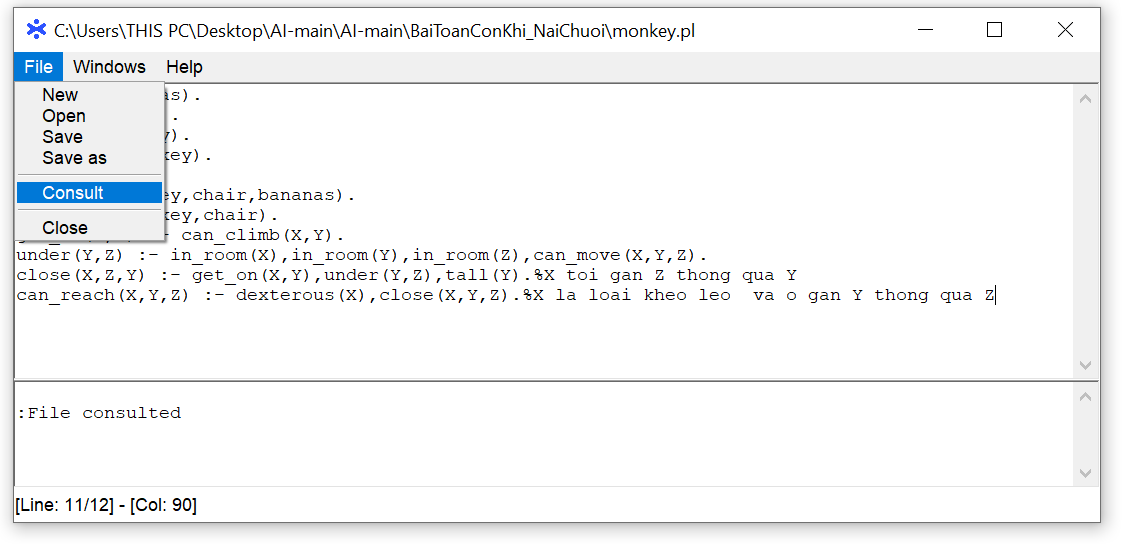
Hỏi con khỉ có thể lấy được nải chuối hay không?

### Các bước thực hiện



Vào file -> Open





can\_reach(monkey, bananas, chair).



Hình 3.4.1. Kết quả chạy thành công

## NGƯỜI QUA SÔNG

### Tổng quan

Bài toán đưa người qua sông (prolog): Có 3 người dân và 3 con sói. Làm sao đưa cả 6 qua sông. Biết rằng có 1 chiếc thuyền, thuyền chở 1 lần nhiều nhất là 2, ít nhất là 1. Chú ý rằng số người dân ở mỗi bờ không ít hơn số sói, nếu không sói sẽ ăn thịt người dân.

\* Các trạng thái:

- Trạng thái ban đầu: cả ba con sói và cả ba người dân đều ở bờ phía bên

trái. Tương ứng [3,3, 0, 0, 1].

- Trạng thái mục tiêu: Cả 3 người dân và cả ba con sói đều ở bờ bên phải,

tương ứng với [0, 0, 3, 3, 0]

\* Thông tin về số lượng người dân và sói ở hai bên bờ:

[A,B,C,D] trong đó:

+ A,B là số sói, số dân ở bờ bên trái

+ C, D là số dân, số sói ở bờ bên phải.

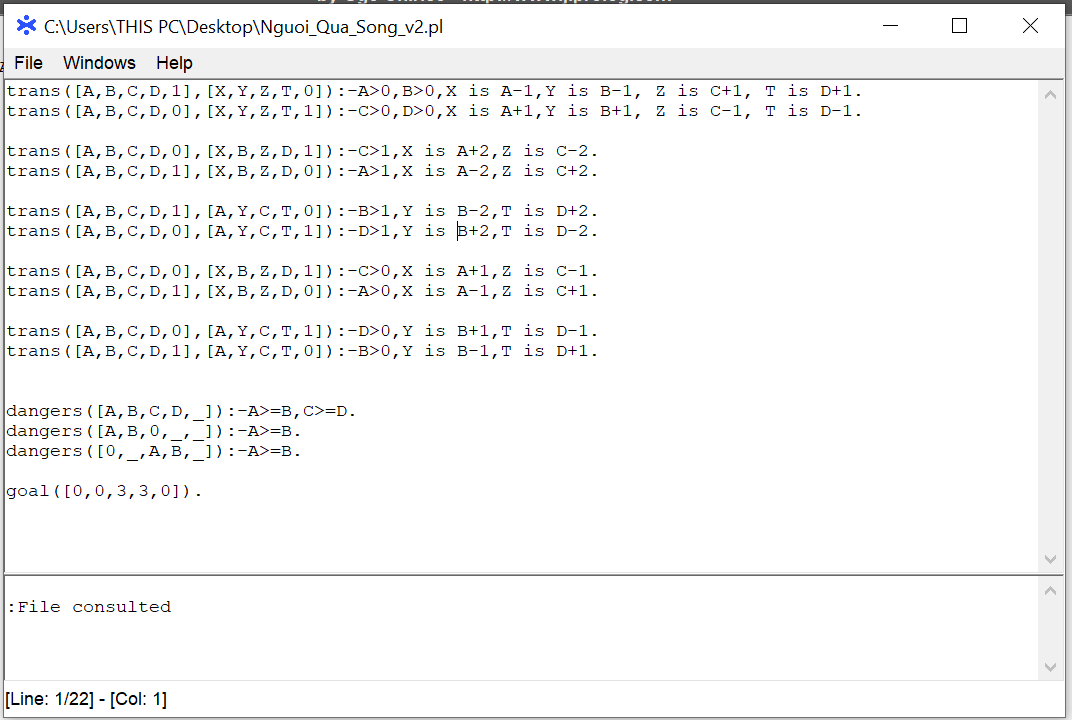
\* Ràng buộc: Số sói phải luôn nhỏ hơn hoặc bằng số dân ở mỗi bờ trái hoặc phải.

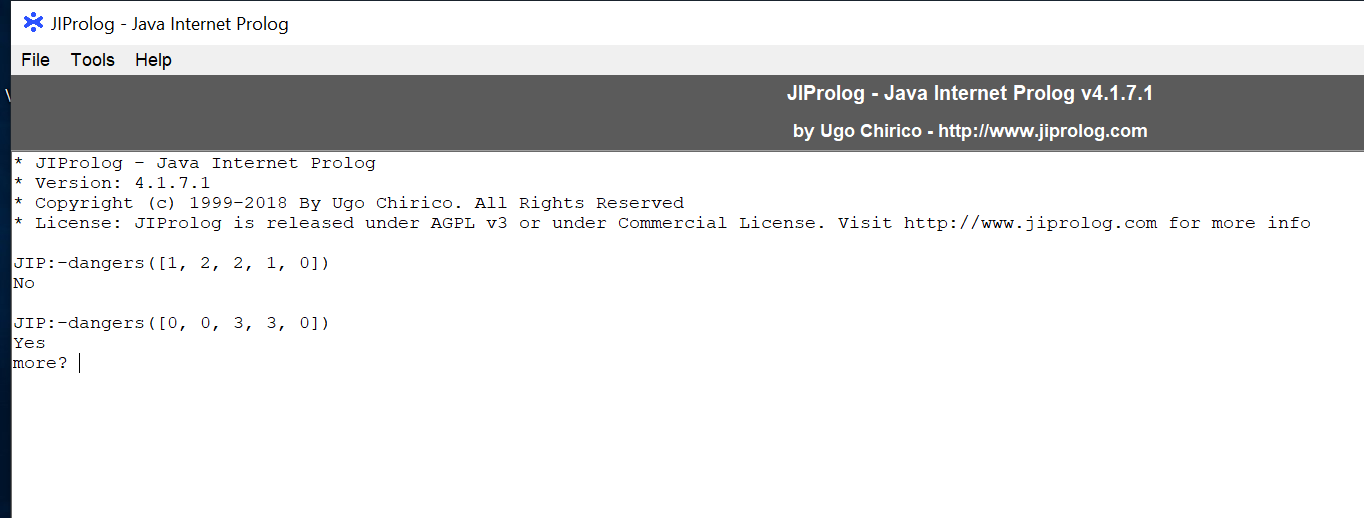
\* Ví dụ:

- Nếu hỏi: dangers([1, 2, 2, 1, 0]) thì kết quả sẽ là NO

- Nếu hỏi: dangers([0, 0, 3, 3, 0]) thì kết quả sẽ là YES

### Các bước thực hiện





Hình 3.4.1. Kết quả chạy thành công

# THUẬT GIẢI QUINLAND TRONG WEKA

## Tổng quan

Thuật toán Quinlan quyết định thuộc tính phân hoạch bằng cách xây dựng các vector đặc trưng cho mỗi giá trị của từng thuộc tính dẫn xuất và thuộc tính mục tiêu.

Bước 1: Phân loại thuộc tính dẫn xuất và thuộc tính mục tiêu

Thuộc tính mục tiêu: là thuộc tính quan tâm.

Thuộc tính dẫn xuất: là thuộc tính quan quan sát.

Bước 2: Với mỗi thuộc tính dẫn xuất A, tính vector đặc trưng

VA(j) = ( T(j , r1), T(j , r2) , …, T(j , rn) )

T(j, ri) = (tổng số phần tử trong phân hoạch có giá trị thuộc tính dẫn xuất A là j và có giá trị thuộc tính mục tiêu là ri ) / ( tổng số phần tử trong phân hoạch có giá trị thuộc tính dẫn xuất A là j )

\* r1, r2, … , rn là các giá trị của thuộc tính mục tiêu

Bước 3: Chọn thuộc tính có nhiều vector đơn vị nhất để phân hoạch.

Vector đơn vị là vector có duy nhất một thành phần có giá trị 1 và những thành phần khác có giá trị 0.

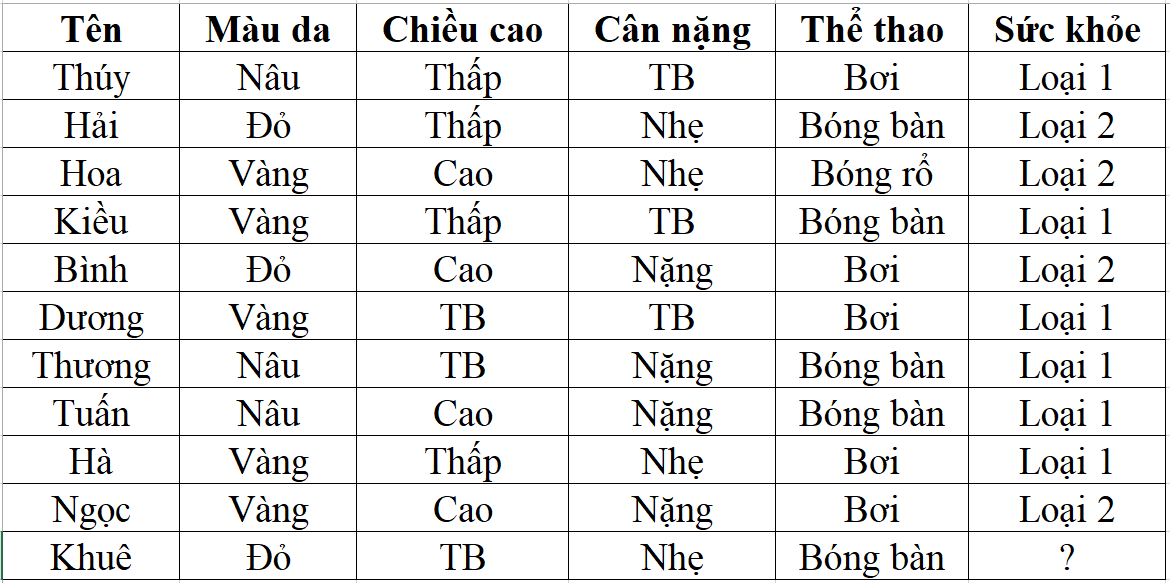
Bước 4: Loại bỏ các thuộc tính đã được phân hoạch.

Nếu vẫn còn thuộc tính đẫn xuất quay lại bước 2 để tính vector đặc trưng cho các thuộc tính dẫn xuất.

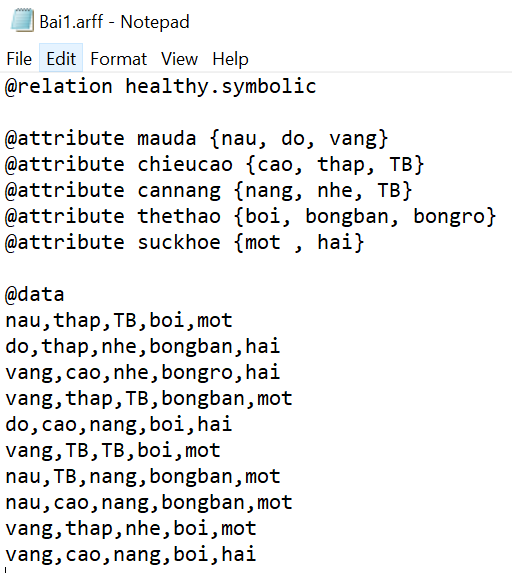
Ngược lại, kết thúc thuật toán. [11]

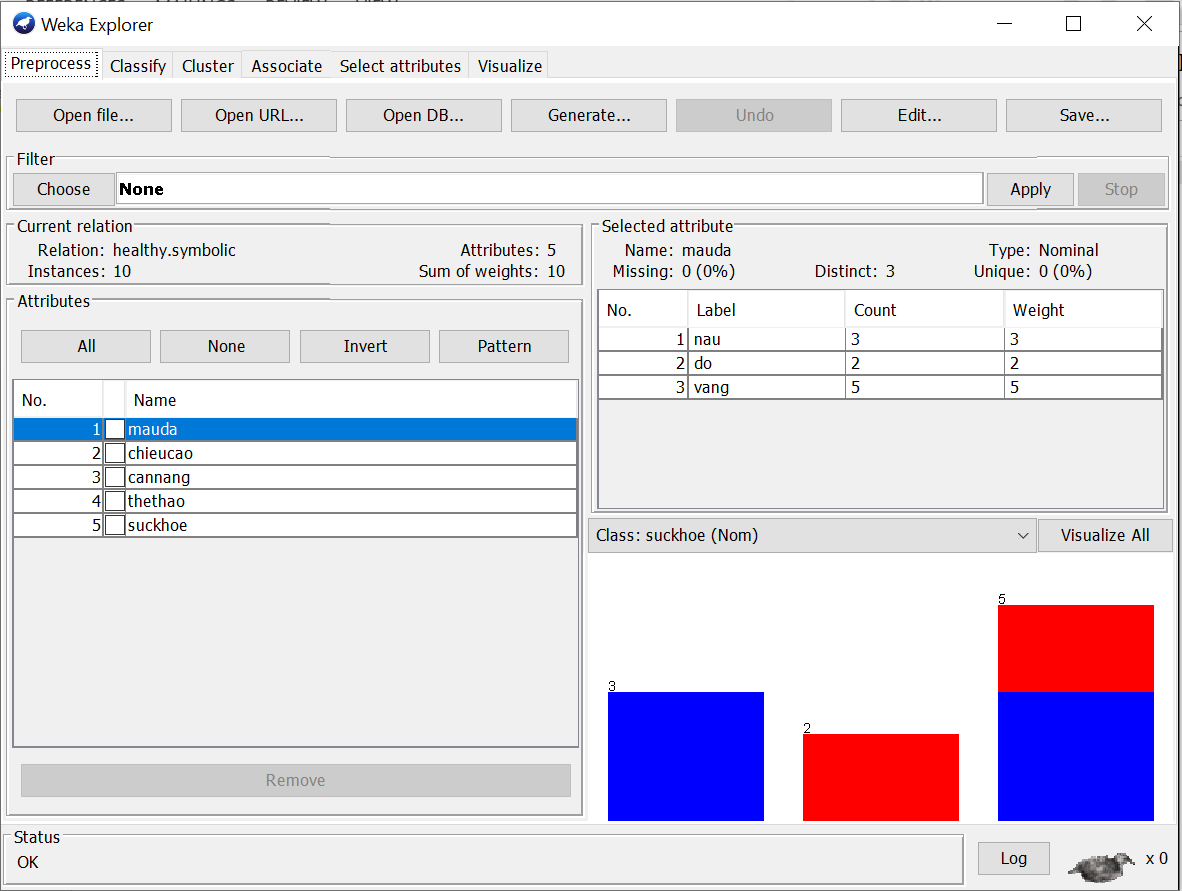
## Các bước thực hiện

Ví dụ ta có bảng dữ liệu quan sát như sau:

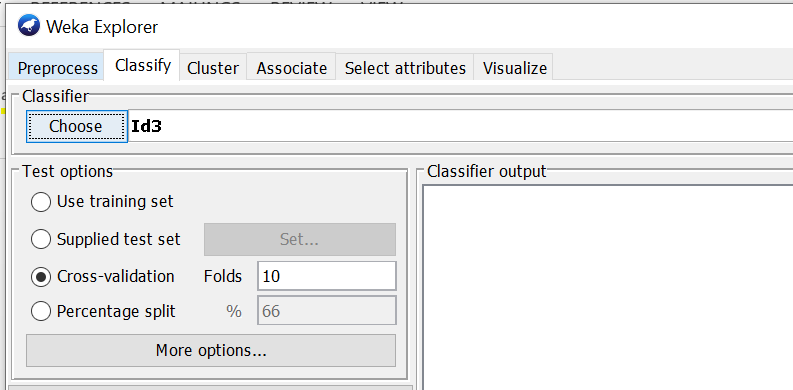


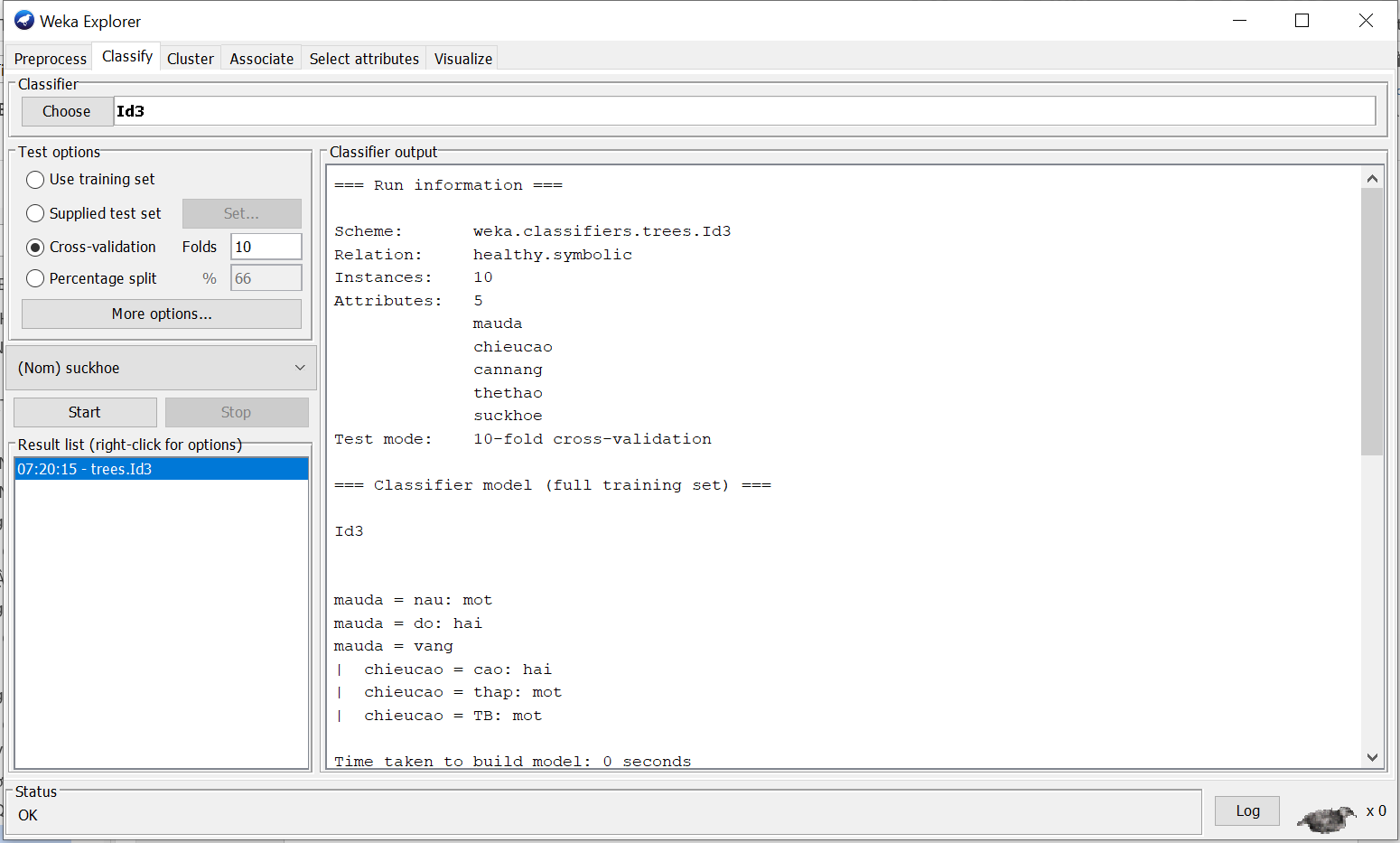
Ta có file .arff được nhập dữ liệu từ bản trên





Chọn thuật toán ID3 tương đương với Quinlan





VẬY mau da nâu : một

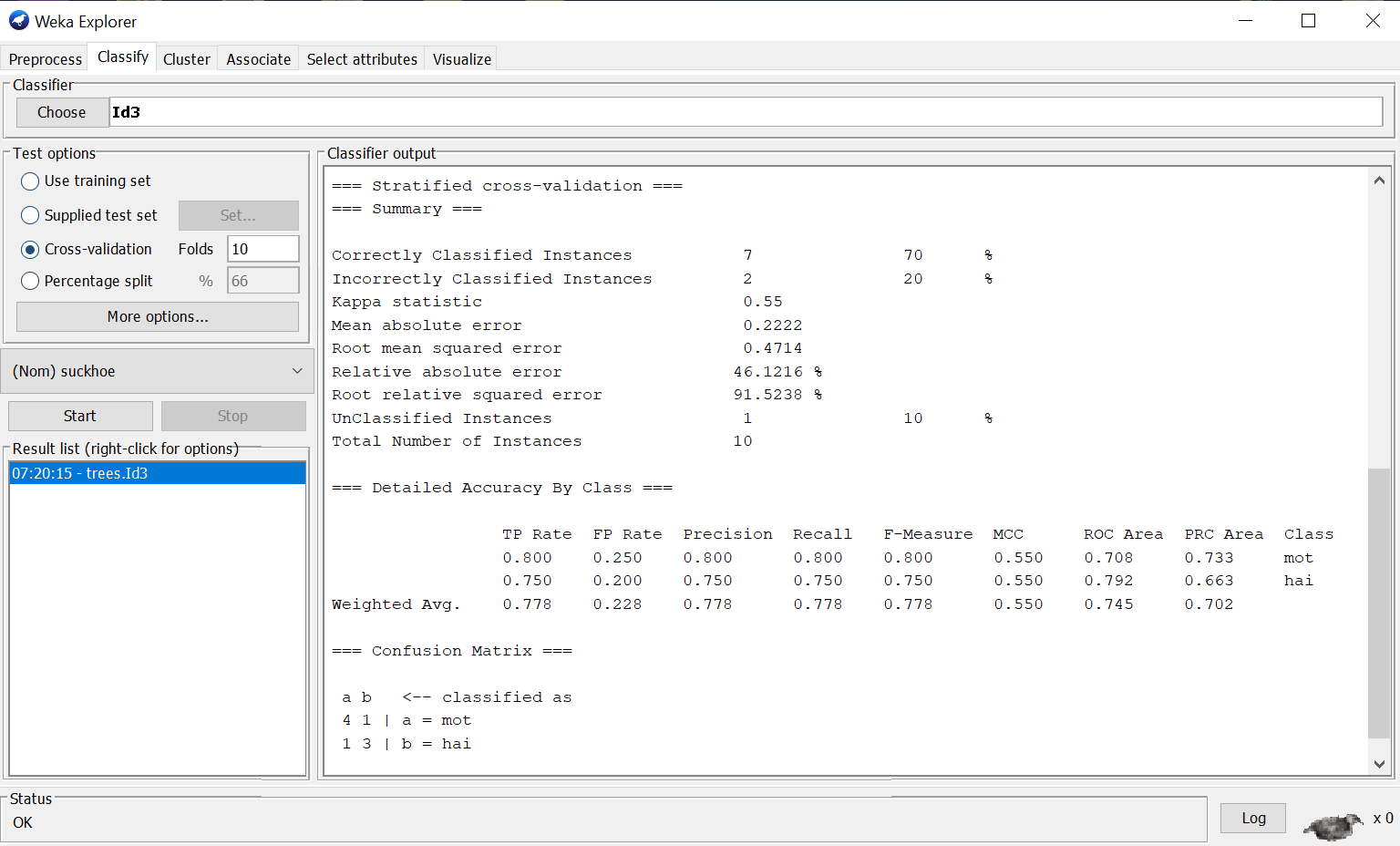
Màu da đỏ: hai

Màu da vàng

Chiều cao = cao thì là hai

Chiều cao = thấp là một

Chiều cao TB là một



Hình 3.4.1. Kết quả chạy thành công

# VOICE TO SPEECH

## Các bước thực hiện

## Code chường trình

|  |
| --- |
| import speech\_recognition as sr  def main():        r = sr.Recognizer()      with sr.Microphone() as source:          r.adjust\_for\_ambient\_noise(source)          print("Nói gì đó")          audio = r.listen(source)          print("Đang ghi")            try:              print("Bạn đã nói \n" + r.recognize\_google(audio))              print("Chuyển thành công \n ")            except Exception as e:              print("Lỗi :  " + str(e))            # write audio          with open("ghi\_am.wav", "wb") as f:              f.write(audio.get\_wav\_data())    if \_\_name\_\_ == "\_\_main\_\_":      main() |

## Chạy

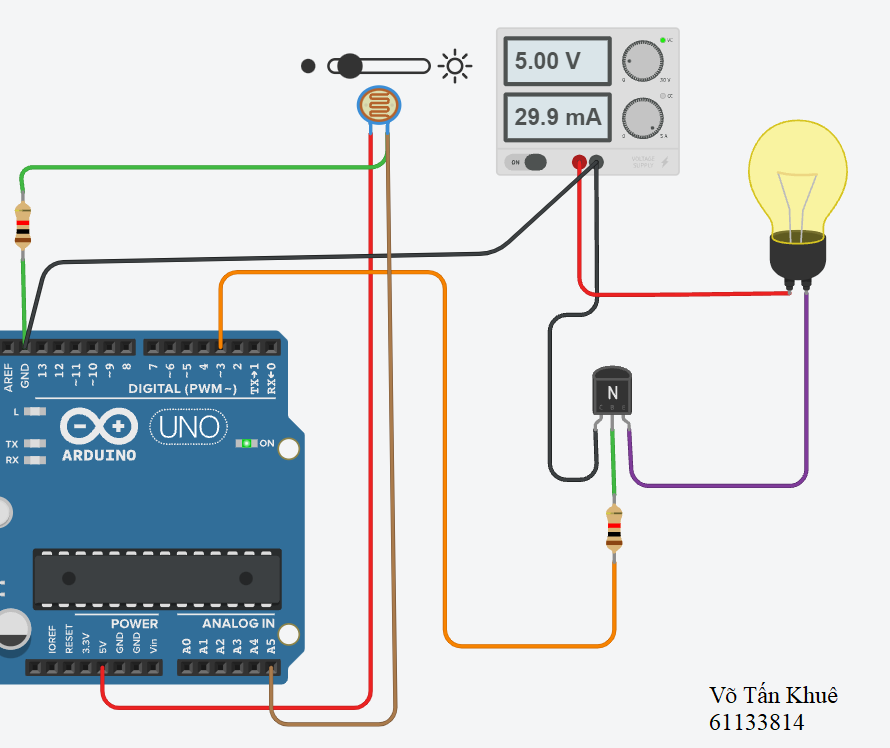
# IOT

## XÂY DỰNG HỆ THỐNG ĐÈN ĐƯỜNG THÔNG MINH

### Tổng quan

Hệ thống được thiết kế bật tắt đèn đường khi có ánh sáng chiếu vào

### Sơ đồ thiết kế



### Đặc điểm của linh kiện

* 2 điện trở: 1 kΩ
* 1 mạch Arduino
* 1 Bulb
* 1 Power supply
* 1 LDR (Light-Dependent Resistor)
* NPN Transistor

### Code chương trình

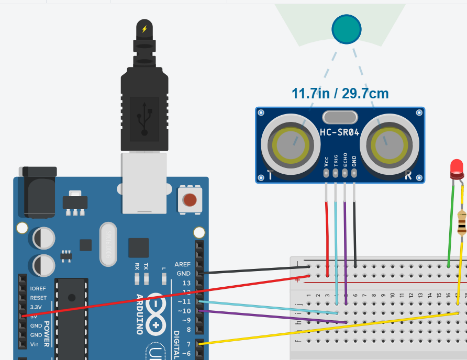
|  |
| --- |
| int ldr=A5;  int ldr\_giaTri;  void setup()  {  pinMode(3, OUTPUT);  pinMode(ldr, INPUT);  }  void loop()  {  ldr\_giaTri=analogRead(ldr);  if (ldr\_giaTri>512)  digitalWrite(3, LOW);  else  digitalWrite(3, HIGH);  } |

## BẬT TẮT ĐÈN LED VỚI CẢM BIẾN HC-SR04

### Mô tả

Hệ thống được thiết kế bật/tắt đèn Led với HC-SR04 được gắn vào chân 10, 11 của Arduino. Đèn chỉ sáng nếu cảm biến nhận thấy vật dưới 50cm.

### Sơ đồ thiết kế



Hình 74. Sơ đồ mạch Proteus

### Đặc điểm của linh kiện

* 1 đèn Led
* 1 điện trở: 100 Ω
* 1 mạch Arduino Uno
* 1 cảm biến HC-SR04

### Code chương trình

|  |
| --- |
| int trig = 11, echo = 10, led = 7; //Khai báo  float distance;  void **setup**()  {  pinMode(trig,OUTPUT);  pinMode(echo,INPUT);  pinMode(led, OUTPUT);  Serial.begin(9600);  }  float **getDistance**(){  digitalWrite(trig, HIGH);  delayMicroseconds(5);  digitalWrite(trig,LOW);  int timer = pulseIn(echo, HIGH);  return timer/58.3f;  }  void **loop**()  {  distance = getDistance();  if(distance < 50){ //Nếu khoảng cách dưới 50cm  digitalWrite(led, HIGH); // Bật đèn  }  else digitalWrite(led, LOW); // tắt đèn  } |

## SMART HOME MINI

### Mô tả

Hệ thống được thiết kế Smart Home Mini gồm có các phòng khách, nhà bếp, nhà xe, cửa ra vào. Cửa sẽ mở khóa nếu dùng RFID Card có giá trị là 225.

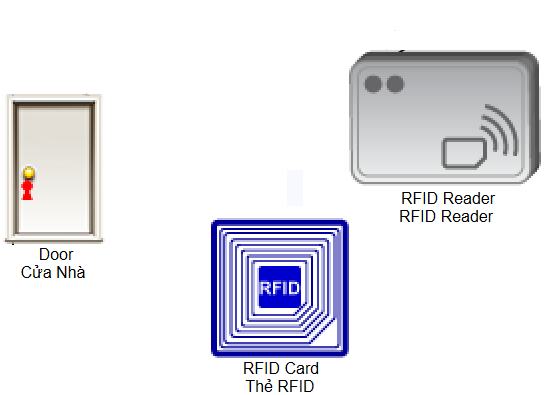
Phòng khách có hệ thống cảm biến nhiệt độ. Nếu nhiệt độ trong phòng lớn hơn 20oC thì cửa sổ sẽ mở, quạt sẽ được bật, đèn sẽ tắt. Nếu nhiệt độ phòng nhỏ hơn 20oC thì cửa sổ sẽ đóng, quạt sẽ tắt, đèn sẽ bật.

Nhà xe có hệ thống cảm biến chuyển động, khói, CO2. Nếu có chuyển động hoặc có khói lớn hơn 0.1 thì cửa nhà xe sẽ được mở. Nếu không có chuyển động và không có khói thì cửa sẽ đóng.

Nhà bếp có hệ thống cảm biến chống cháy. Nếu cảm biến phát hiện đám cháy sẽ kích hoạt vòi phun nước tự động.

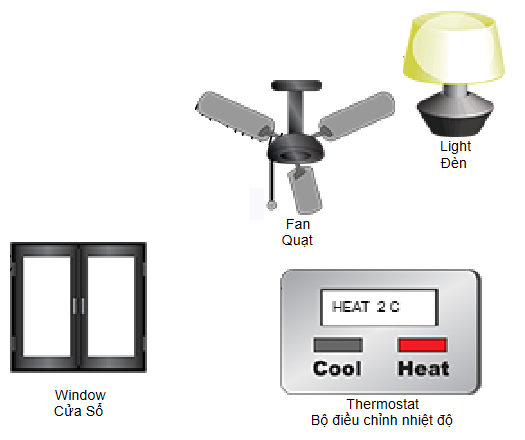
### Sơ đồ thiết kế

* Cửa ra vào



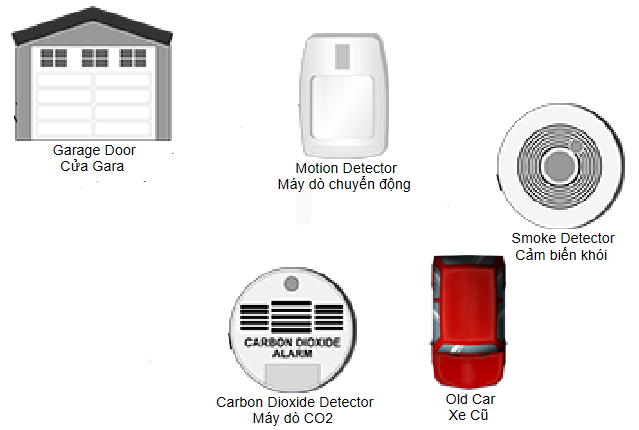
Hình 136. Sơ đồ mạch Cisco cửa ra vào

* Phòng khách



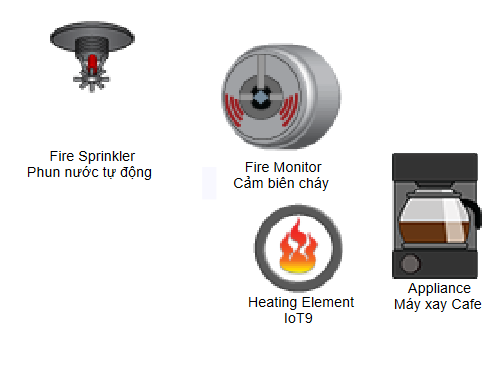
Hình 137. Sơ đồ mạch Cisco phòng khách

* Nhà xe

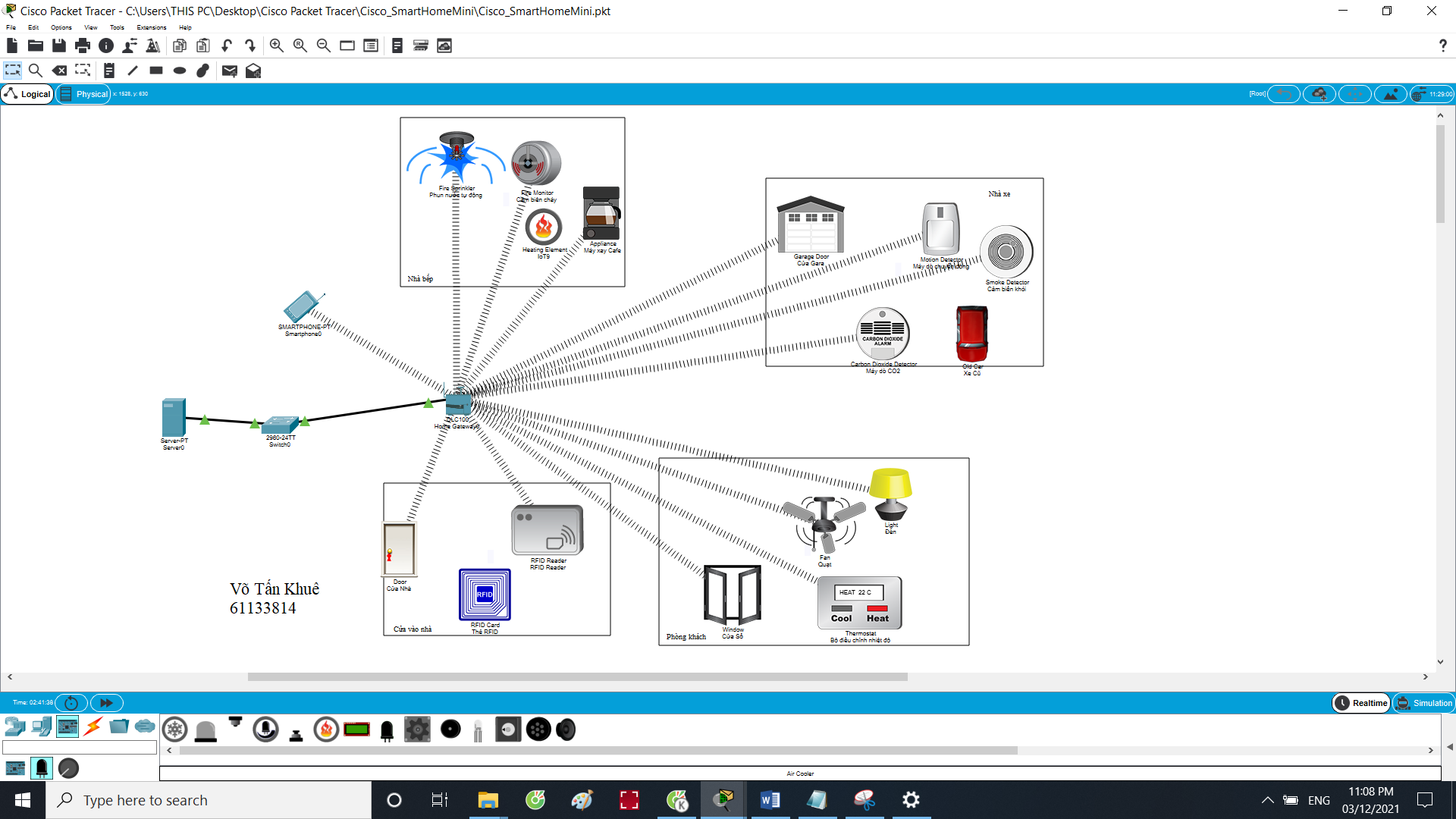


Hình 138. Sơ đồ mạch Cisco nhà xe

* Nhà bếp



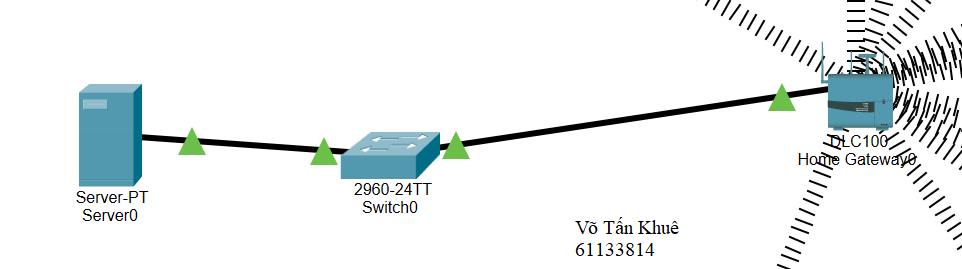
Hình 139. Sơ đồ mạch Cisco nhà bếp



Hình 140. Sơ đồ mạch Cisco Packet Tracer

### Cấu hình mạng và luật

* **Câu hình sever**

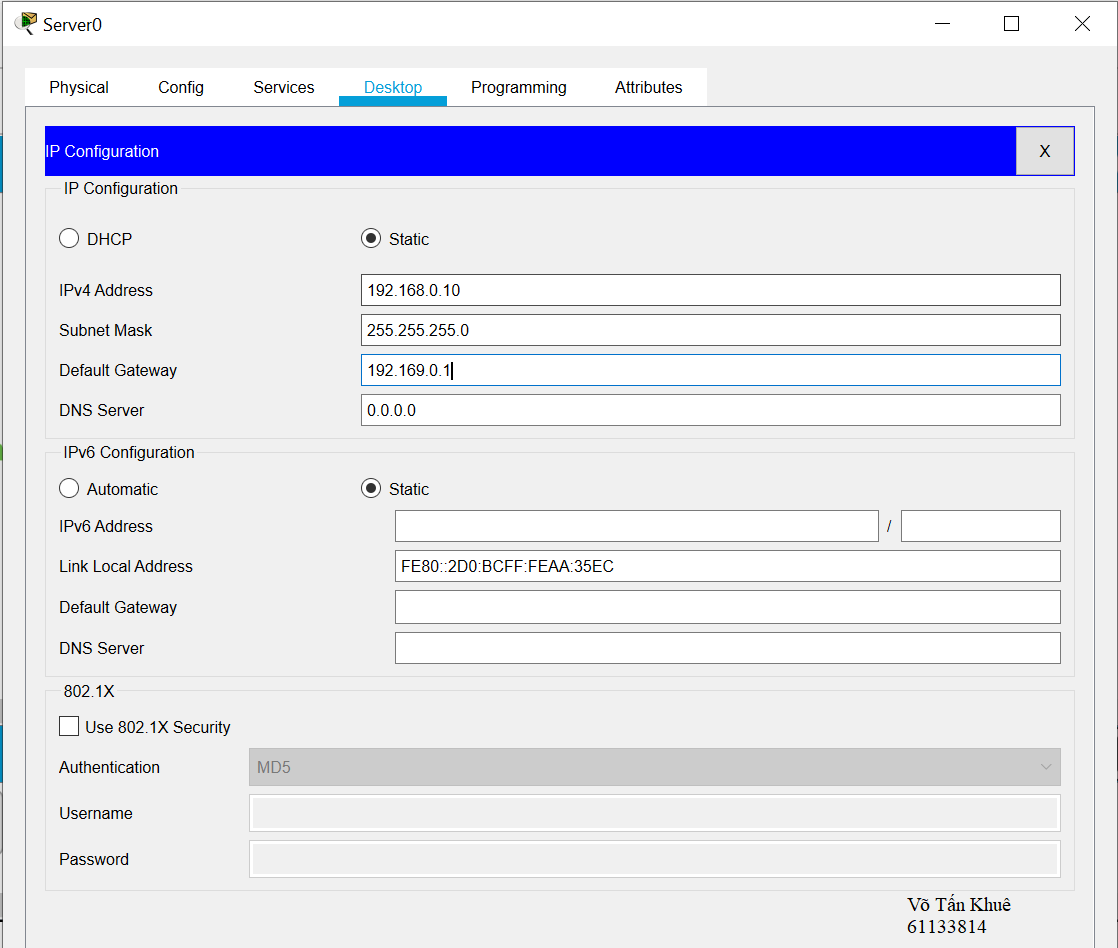


Hình 141. Sơ đồ mạch Cisco Server, Switch, Home Gateway

* Cấu hình Server-PT

+ Địa chỉ IPv4: 192.168.0.10

+ Gateway: 192.169.0.1

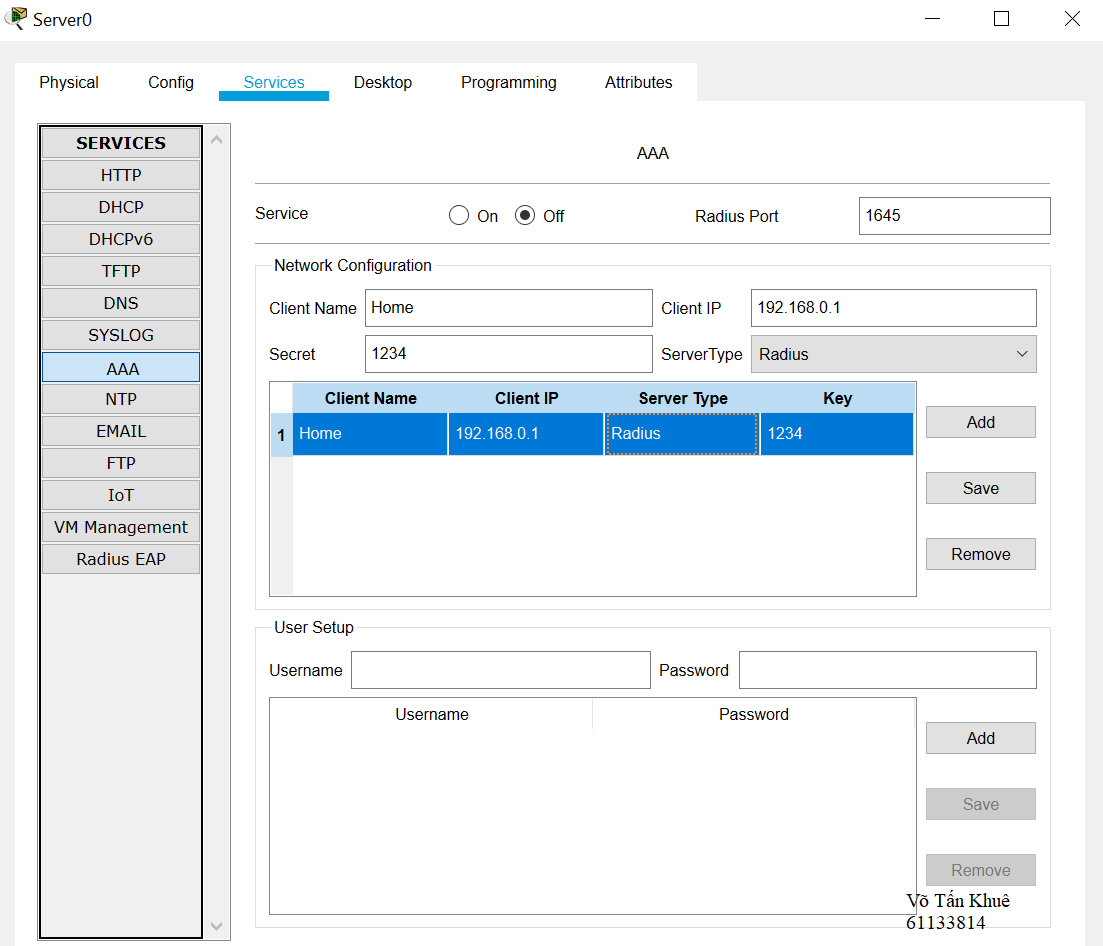


Hình 142. Cấu hình sever

+ Client: Home

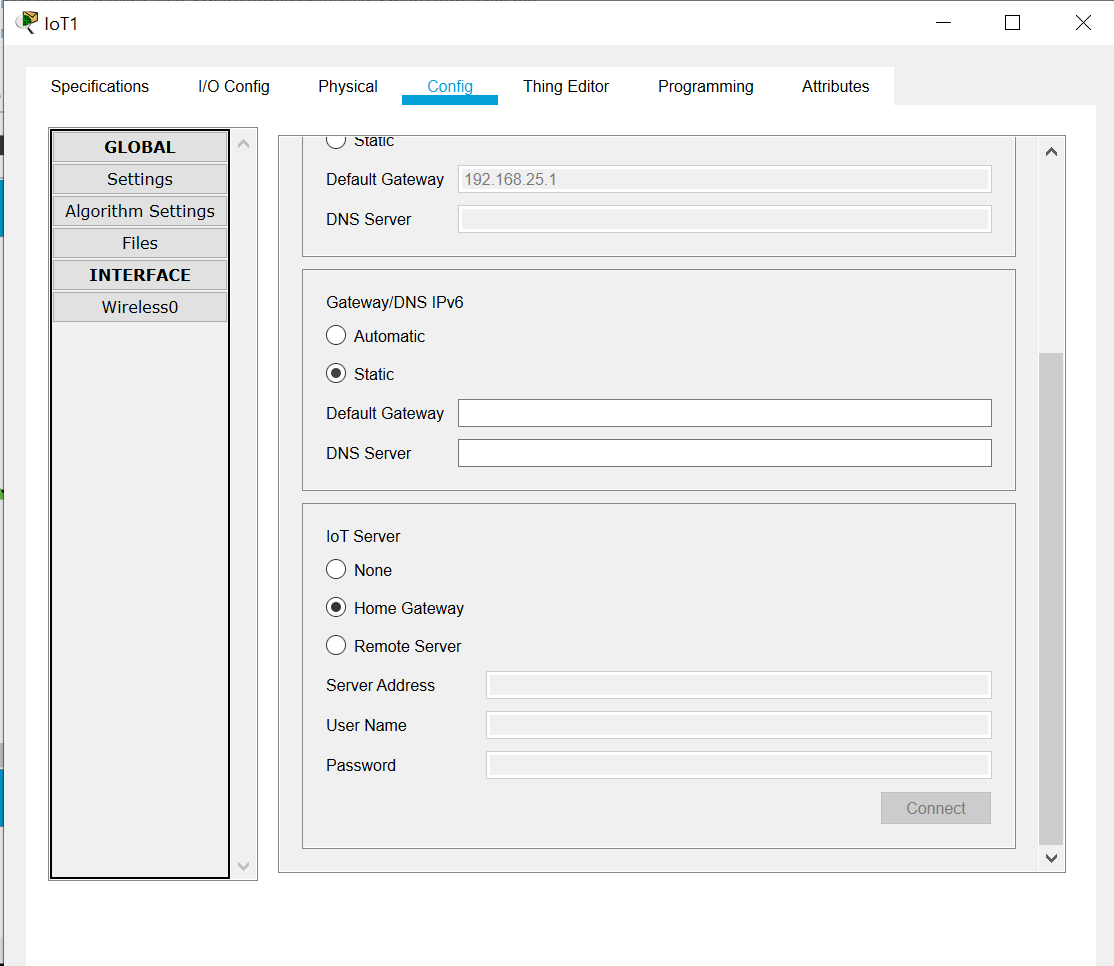
+ Client IP: 192.168.0.1

+ Secret: 1234



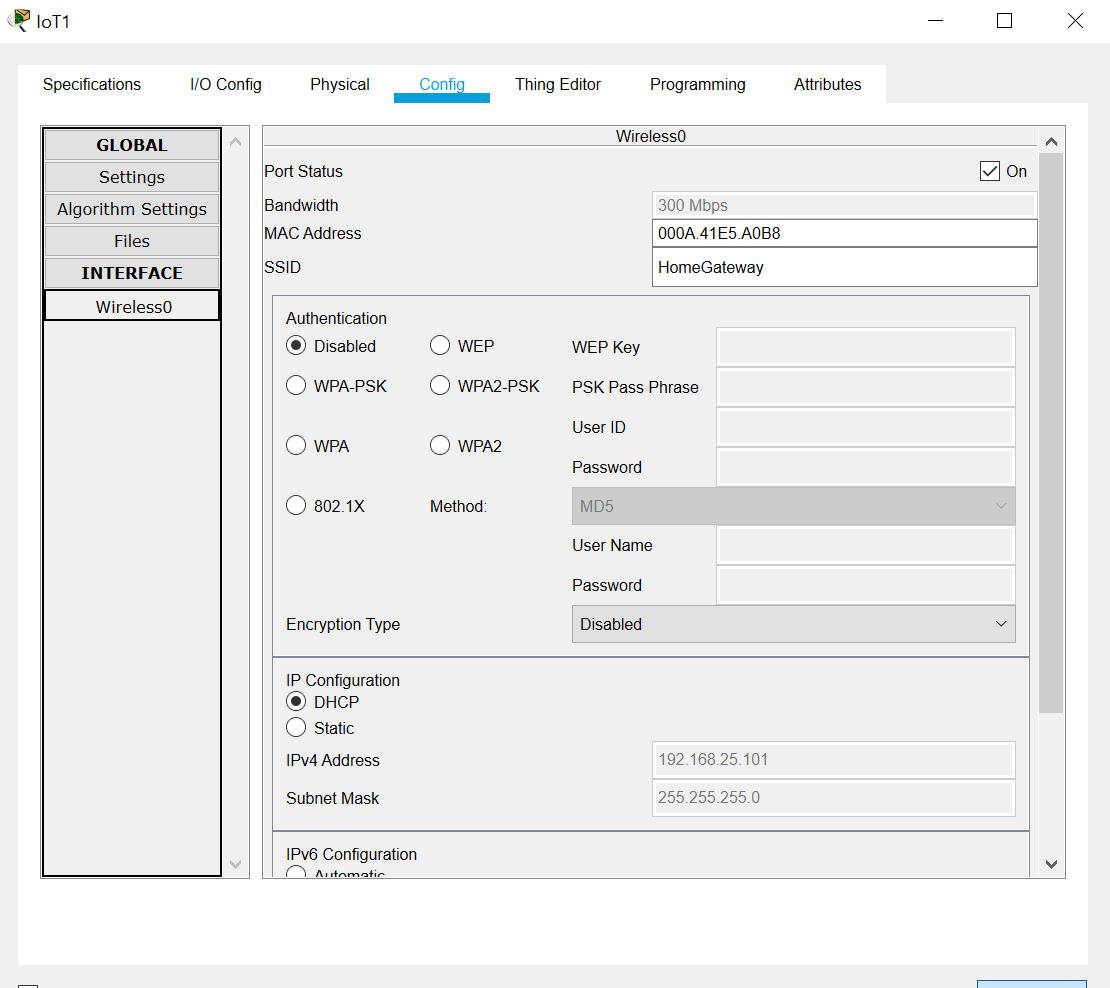
Hình 143. Cấu hình sever

* Chọn Iot Sever là **Home Gateway** cho tất cả các thiết bị trong nhà



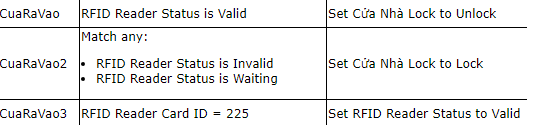
Hình 144. Chọn Home Gateway cho Iot Server

* Kết nối SSID cho tất cả thiết bị trong nhà là **HomeGateway**



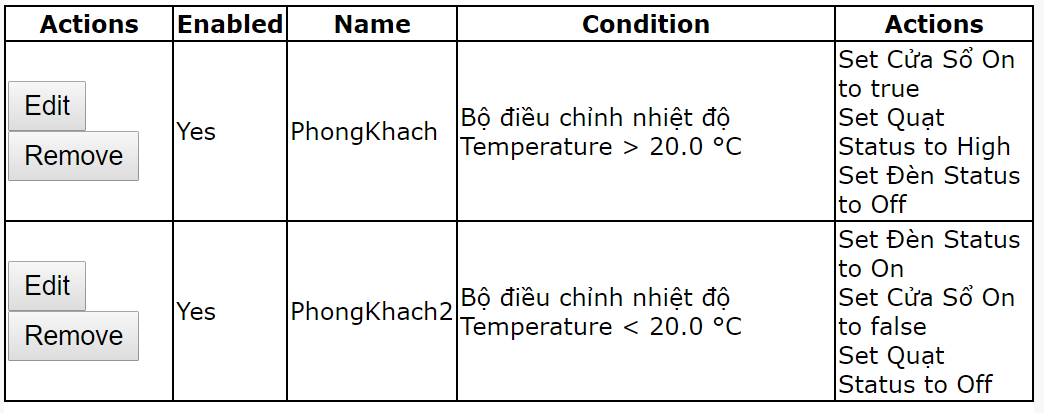
Hình 145. Kết nối SSID

* **Luật**
* Cửa ra vào



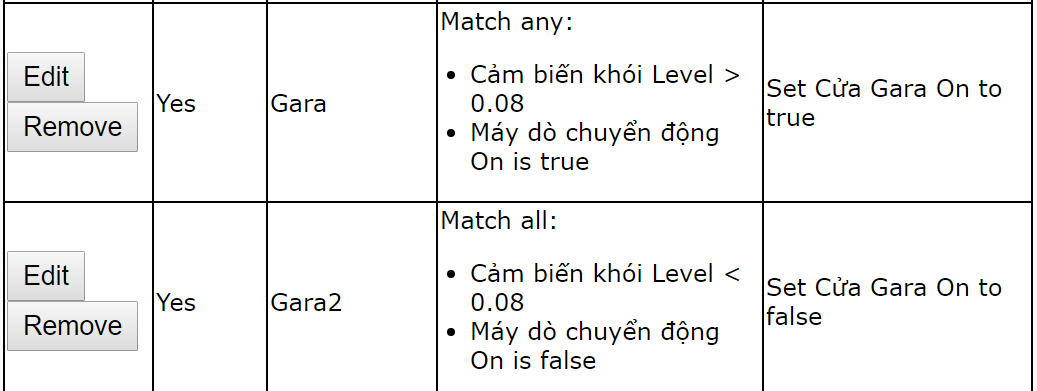
Hình 146. Luật cửa ra vào

* Phòng Khách



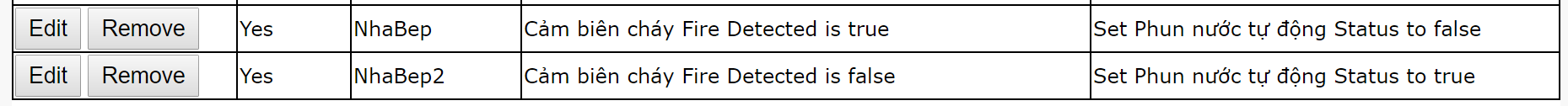
Hình 147. Luật phòng khách

* Nhà Xe



Hình 148. Luật nhà xe

* Nhà bếp



Hình 149. Luật nhà bếp

# MACHING LEARNING

## KNN

### Tổng quan

KNN là một thuật toán máy học có giám sát (ML) đơn giản, có thể được sử dụng cho các nhiệm vụ phân loại hoặc hồi quy - và cũng thường được sử dụng trong việc nhập giá trị bị thiếu. Nó dựa trên ý tưởng rằng những quan sát gần nhất với một điểm dữ liệu nhất định là những quan sát "tương tự" nhất trong một tập dữ liệu và do đó chúng ta có thể phân loại các điểm không lường trước được dựa trên giá trị của các điểm hiện có gần nhất. Bằng cách chọn K , người dùng có thể chọn số lượng quan sát gần đó để sử dụng trong thuật toán. [9]

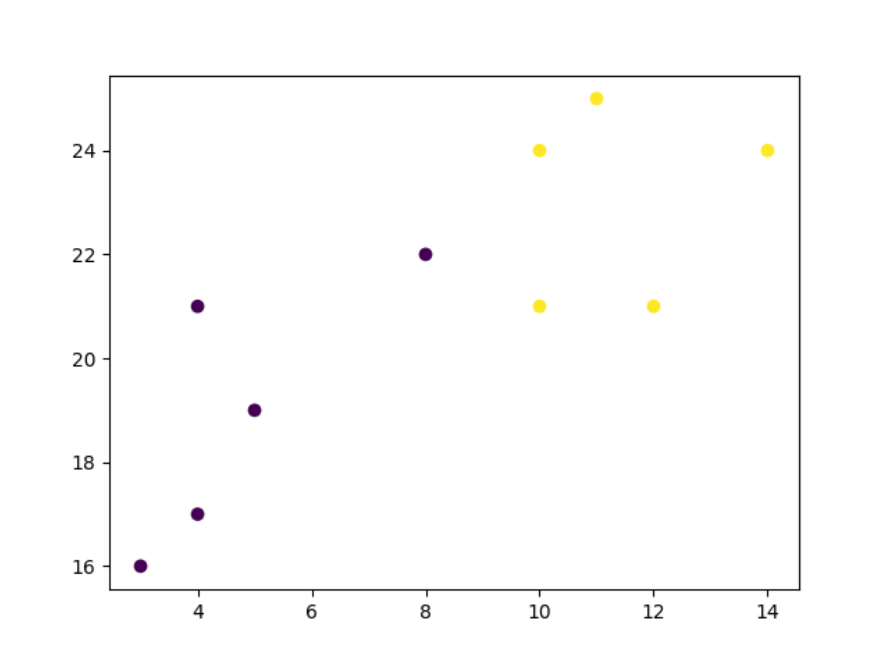
### Cách hoạt động

K là số láng giềng gần nhất để sử dụng. Đối với phân loại, đa số phiếu được sử dụng để xác định loại quan sát mới nên rơi vào. Các giá trị lớn hơn của K thường mạnh hơn đối với các giá trị ngoại lệ và tạo ra các ranh giới quyết định ổn định hơn các giá trị rất nhỏ ( K = 3 sẽ tốt hơn K = 1 , điều này có thể tạo ra kết quả không mong muốn).

### Code chương trinh

|  |
| --- |
| #Three lines to make our compiler able to draw:  import sys  import matplotlib  matplotlib.use('Agg')  import matplotlib.pyplot as plt  x = [4, 5, 10, 4, 3, 11, 14 , 8, 10, 12]  y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]  classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]  plt.scatter(x, y, c=classes)  plt.show()  #Two  lines to make our compiler able to draw:  plt.savefig(sys.stdout.buffer)  sys.stdout.flush() |

### Kết quả chạy chương trình



Hình 3.4.1. Kết quả chạy thành công

## TRỢ LÝ ẢO (SPEECH RECONITION)

### Hướng dẫn cài đặt và sử dụng

Cài đặt các thư viện hỗ trợ thu âm và phát giọng nói bằng cách mở cmd và nhập

**pip install SpeechRecognition gTTS**

**pip install playsound==1.2.2**

Mở và chạy file assistant.py trong thư mục speech\_recognition.

### Kết quả thực hiện chương trình

Khi chạy chương trình trợ lý ảo sẽ nói.



Nếu trong lời nói có chứa ‘tên’ trợ lý ảo sẽ trả lời tên của mình.



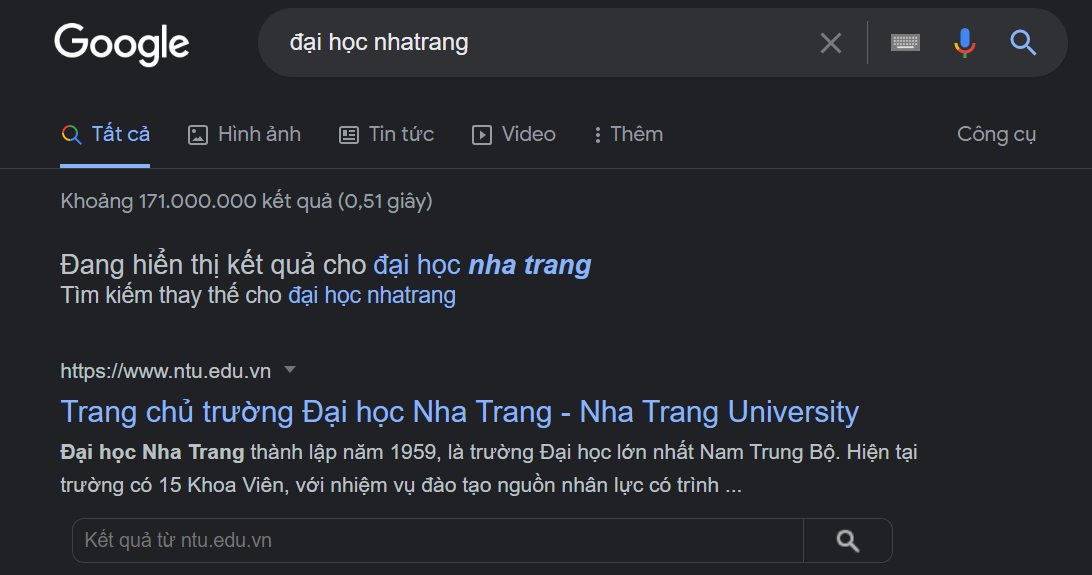
Nếu trong lời nói có chứa ‘giờ’ trợ lý ảo sẽ trả lời giờ, ngày, tháng, năm hiện tại.

.

Nếu trong lời nói có chứa ‘tìm kiếm’ và chờ một lúc sau đó nói thứ mình muốn tìm.







Nếu trong lời nói có chứa ‘vị trí’ và chờ một lúc sau đó nói vị trí muốn tìm.



  
Graphical user interface, text, application

Description automatically generated

Nếu trong lời nói có chứa ‘tạm biệt’ trợ lý ảo sẽ dùng.



## CAR NEURAL NETWORK

### Hướng dẫn cài đặt và sử dụng

Cài đặt 2 thư viện pygame và neat bằng cách mở cmd nhập **pip install pygame neat-python**.

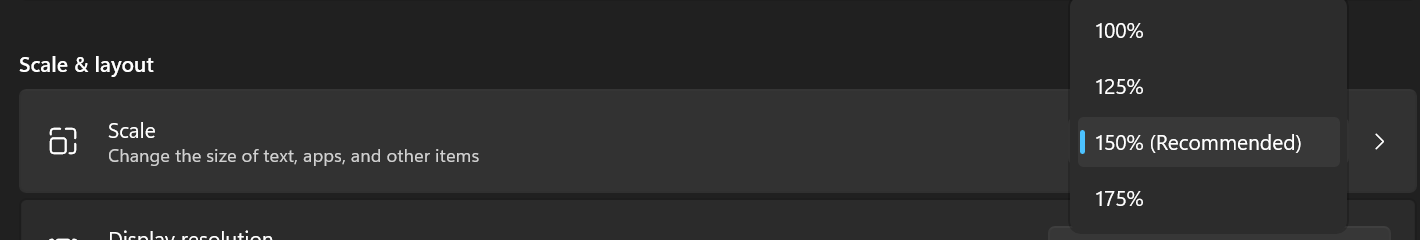
Đưa đường dẫn làm việc đến thư mục chứa file main.py trong thư mục car\_neural\_network trong phần terminal của vscode. Và chạy file main.py



Lưu ý: Màn hình có thể bị tràn, nếu bị tràn cần chỉnh lại bằng cách

Vào display settings





Chỉnh xuống 100% hoặc 125%.

### Kết quả thực hiện chương trình

Logo

Description automatically generated

Graphical user interface, text

Description automatically generated

# TÀI LIỆU THAM KHẢO

|  |  |  |
| --- | --- | --- |
| [1] |  | Wikipedia tiếng việt, “Tìm kiếm theo chiều sâu,” 26 07 2022. [Trực tuyến]. Available: https://vi.wikipedia.org/wiki/T%C3%ACm\_ki%E1%BA%BFm\_theo\_chi%E1%BB%81u\_s%C3%A2u. [Đã truy cập 26 07 2022]. |
| [2] |  | Wikipedia tiếng việt, "Tìm kiếm theo chiều rộng," 03 08 2022. [Online]. Available: https://vi.wikipedia.org/wiki/T%C3%ACm\_ki%E1%BA%BFm\_theo\_chi%E1%BB%81u\_r%E1%BB%99ng. [Accessed 03 08 2022]. |
| [3] |  | Mygreatlearning, "What is A\* Search Algorithm?," 16 08 2022. [Online]. Available: https://www.mygreatlearning.com/blog/a-search-algorithm-in-artificial-intelligence/. [Accessed 16 08 2022]. |
| [4] |  | Wikipedia tiếng việt, "Tic-tac-toe," 16 08 2022. [Online]. Available: https://vi.wikipedia.org/wiki/Tic-tac-toe. [Accessed 16 08 2022]. |
| [5] |  | Wikipedia tiếng việt, "Bài toán người bán hàng," 16 08 2022. [Online]. Available: https://vi.wikipedia.org/wiki/B%C3%A0i\_to%C3%A1n\_ng%C6%B0%E1%BB%9Di\_b%C3%A1n\_h%C3%A0ng. [Accessed 16 08 2022]. |
| [6] |  | Vnexpress, "Đưa dê qua sông bằng cách nào để không bị sói ăn thịt?," 16 08 2022. [Online]. Available: https://vnexpress.net/dua-de-qua-song-bang-cach-nao-de-khong-bi-soi-an-thit-3622489.html. [Accessed 16 08 2022]. |
| [7] |  | Wikipedia tiếng việt, "Bài toán tám quân hậu," 16 08 2022. [Online]. Available: https://vi.wikipedia.org/wiki/B%C3%A0i\_to%C3%A1n\_t%C3%A1m\_qu%C3%A2n\_h%E1%BA%ADu. [Accessed 16 08 2022]. |
| [8] |  | Wikipedia tiếng việt, "Mệnh đề toán học," 16 08 2022. [Online]. Available: https://vi.wikipedia.org/wiki/M%E1%BB%87nh\_%C4%91%E1%BB%81\_to%C3%A1n\_h%E1%BB%8Dc. [Accessed 16 08 2022]. |
| [9] |  | nowshad7, "Backward Chaining," 16 08 2022. [Online]. Available: https://nowshad7.blogspot.com/2019/09/backward-chaining.html. [Accessed 16 08 2022]. |
| [10] |  | Thanh Cuong, "Giải bài toán tam giác sử dụng mạng ngữ nghĩa – Solution for triangle using semantic network," 16 08 2022. [Online]. Available: https://thanhcuong.wordpress.com/2010/12/16/gi%E1%BA%A3i-bi-ton-tam-gic-s%E1%BB%AD-d%E1%BB%A5ng-m%E1%BA%A1ng-ng%E1%BB%AF-nghia-solution-for-triangle-using-semantic-network/. [Accessed 16 08 2022]. |
| [11] |  | Nguyễn Văn Hiếu, "Ví dụ về thuật toán Quinlan," 16 08 2022. [Online]. Available: https://sinhvientot.net/vi-du-ve-thuat-toan-quinlan/. [Accessed 16 08 2022]. |
| [12] |  | w3schools, "Machine Learning - K-nearest neighbors (KNN)," 16 08 2022. [Online]. Available: https://www.w3schools.com/python/python\_ml\_knn.asp. [Accessed 16 08 2022]. |