알고리즘 과제1

2018203040 노석규

개발언어 : cpp

소스코드에 함수에 관한 설명과 어떠한 방법으로 흘러가는지에 대한 설명은 주석처리로 표기하였습니다.

\*소스코드\*

#include <iostream>

#include <string>

using namespace std;

enum CATEGORY { NATIONALITY, COLOR, BEVERAGE, CIGARETTE, PET };

enum COLOR { BLUE, GREEN, RED, WHITE, YELLOW };

enum NATIONALITY { BRIT, GERMAN, SWEDE, NORWEGIAN, DANE };

enum BEVERAGE { BEER, COFFEE, MILK, TEA, WATER };

enum CIGARETTE { BLUEMASTER, DUNHILL, PALLMALL, PRINCE, BLEND };

enum PET { CAT, BIRD, DOG, FISH, HORSE };

string Einstein\_table[5][5] = {

{ "Brit\t", "German\t", "Swede\t", "Norwegian", "Dane\t" },

{ "Blue\t", "Green\t", "Red\t", "White\t", "Yellow\t" },

{ "Beer\t", "Coffee\t", "Milk\t", "Tea\t", "Water\t" },

{ "Bluemaster", "Dunhill\t", "PallMall", "Prince\t", "Blend\t" },

{ "Cat\t", "Bird\t", "Dog\t", "Fish\t", "HORSE\t" }

};

bool set\_table(int[5][5], int line, int pos);

bool check\_hint(int[5][5], int line);

void print\_table(int[5][5]);

int count\_num = 0; // 노드 수 확인을 위한 변수 count\_num 초기화

int main()

{

int table[5][5];

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

table[i][j] = j;

}

}

/\*

table 초기화

table[CATEGORY][ITEM]=집의 순서 로 초기화

첫번째 집 ~ 다섯 번째 집 => 0 ~ 4

\*/

set\_table(table, 0, 0);

cout << "explored node count : " << count\_num << endl;

}

bool set\_table(int table[5][5], int n, int pos)

{

/\*

table 인자값은 main에서 생성된 2차원 배열

n 인자값은 몇번째 CATEGORY까지 해결했는지 카운트

pos 인자값은 몇번째 인덱스까지 섞였는지 카운트

\*/

if (n == 5) // 모든 CATEGORY가 해결되었으므로 출력함수 호출 후 함수 종료

{

print\_table(table);

return 1;

}

/\*

pos 가 5와 같으면 해당 CATEGORY의 모든 값이 섞였다는 것이다.

그러므로 해당 CATEGORY까지의 값을 검사 후 조건에 대해 false가 발생하지 않는다면

다음 CATEGORY로 넘어간다.

조건을 만족시키지 못 했을 경우 백트래킹을 한다.

\*/

if (pos == 5)

{

if (check\_hint(table, n))

{

if (set\_table(table, n + 1, 0))

return 1;

}

return 0;

}

/\*

table의 n번째 CATEGORY를 섞어주는 반복문

pos는 이전까지 섞인 인덱스의 개수를 의미하므로 i = pos부터 섞는다.

table[n][pos]와 table[n][i]를 스왑한 후,

다음 pos로 넘어가, 해결이 가능한지 확인한다.

만약 해결이 가능하다면 함수를 종료한다.

해결이 불가능하다면, 다시 swap하여 원래의 값으로 변경해놓은 후 다음으로 넘어간다.

\*/

for (int i = pos; i < 5; i++)

{

swap(table[n][pos], table[n][i]);

if (set\_table(table, n, pos + 1))

return 1;

swap(table[n][pos], table[n][i]);

count\_num++;

}

return 0;

}

bool check\_hint(int table[5][5], int n) {

/\*

# promissing #

table이 조건에 대해 거짓이 있는지 판별한다.

거짓일 경우 false, 참일 경우 true를 반환한다.

\*/

if (n == 0) {

// 국적의 정보를 통해 판별이 가능한 경우

if (table[NATIONALITY][NORWEGIAN] != 0) // hint 9

return 0;

}

else if (n == 1) {

// 국적과 집 색깔의 정보를 통해 판별이 가능한 경우

if (table[NATIONALITY][BRIT] != table[COLOR][RED]) // hint 1

return 0;

if (table[COLOR][GREEN] != table[COLOR][WHITE] - 1) // hint 4

return 0;

if (table[NATIONALITY][NORWEGIAN] != table[COLOR][BLUE] - 1 && table[NATIONALITY][NORWEGIAN] != table[COLOR][BLUE] + 1) // hint 14

return 0;

}

else if (n == 2) {

// 국적과 집 색깔, 음료의 정보를 통해 판별이 가능한 경우

if (table[NATIONALITY][DANE] != table[BEVERAGE][TEA]) // hint 3

return 0;

if (table[COLOR][GREEN] != table[BEVERAGE][COFFEE]) // hint 5

return 0;

if (table[BEVERAGE][MILK] != 2) // hint 8

return 0;

}

else if (n == 3) {

// 국적과 집 색깔, 음료와 담배 정보를 통해 판별이 가능한 경우

if (table[COLOR][YELLOW] != table[CIGARETTE][DUNHILL]) // hint 7

return 0;

if (table[CIGARETTE][BLUEMASTER] != table[BEVERAGE][BEER]) // hint 12

return 0;

if (table[NATIONALITY][GERMAN] != table[CIGARETTE][PRINCE]) // hint 13

return 0;

if (table[CIGARETTE][BLEND] != table[BEVERAGE][WATER] - 1 && table[CIGARETTE][BLEND] != table[BEVERAGE][WATER] + 1) // hint 15

return 0;

}

else if (n == 4) {

// 모든 정보를 통해 판별이 가능한 경우

if (table[NATIONALITY][SWEDE] != table[PET][DOG]) // hint 2

return 0;

if (table[CIGARETTE][PALLMALL] != table[PET][BIRD]) // hint 6

return 0;

if (table[CIGARETTE][BLEND] != table[PET][CAT] - 1 && table[CIGARETTE][BLEND] != table[PET][CAT] + 1) // hint 10

return 0;

if (table[PET][HORSE] != table[CIGARETTE][DUNHILL] - 1 && table[PET][HORSE] != table[CIGARETTE][DUNHILL] + 1) // hint 11

return 0;

}

return 1;

}

void print\_table(int table[5][5]) // table 저장된 값을 통해 Einstein\_table 출력한다.

{

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

for (int k = 0; k < 5; k++)

{

if (table[i][k] == j)

{

cout << Einstein\_table[i][k] << "\t";

break;

}

}

}

cout << endl;

}

cout << endl;

}

카테고리를 행으로, 아이템을 열로하는 2차원 배열을 구상하였습니다. 첫번째 행부터 아이템의 위치를 swap하여 promising 조건을 만족시키면 다음 행으로 넘어갈 수 있게 하였습니다.

check\_hint 함수에서 행별로 쓰일 수 있는 힌트들의 정보를 통해 promising한지 판별하였습니다.

국적으로만 판별할 수 있는 힌트.

'첫 번째 집에 노르웨이 사람이 살고 있다' 조건에서 첫 행을 배치할 때 첫 번째 집에 노르웨이 사람이 있을 때만 promising한 상태로 다음 행도 promising한지 판별하였습니다.

그 다음 국적과 집의 색으로 판별할 수 있는 힌트 -> 국적, 집의 색, 음료 -> 국적, 집의 색, 음료, 담배 -> 국적, 집의 색, 음료, 담배, 애완동물의 순으로 각 조건마다 promising 할때마다 다음 추가로 새로운 정보를 가지고 promising한지 안한지 판별하였습니다.