人工智慧 project1

實驗過程:

大致上跟老師所解釋之做法相同，用stack存資料、在一個個pop出來，比較不同的是，我每個node只記錄了variable + value，並沒有記錄之前node所用之variable + value，相反的，我用了一個board紀錄之前的紀錄，並在pop時更新這個board。

實驗結果:

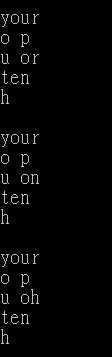
1. puzzle1 visited node = 75,

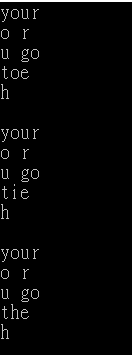
puzzle2 visited node = 11,

puzzle3 visited node = 2686,

puzzle4 visited node = 35366,

1. 可以找出所有解，以puzzle1為例:





最終visited node = 1006853

共求得280685組解

1. 可以每次都產生不同解，只需把每次的input English word打亂即可。
2. 我在push node時，只會把能符合所有條件的node push進stack，某種程度上使用了forward check，不過我不確定是否算是有使用，因為我實在想不出該如何把這些演算法應用在實作上的方法，所以我並沒有特別作比較。

觀察:

我認為這次作業有兩個特稱，第一個是資料量龐大，例如我一開始也想在每個點都儲存之前node的紀錄，但memory爆掉了，所以我決定還是用自己的方法。第二，數據間會互相影響，所以問題就會變得較為複雜。

學到的東西:

思緒還是不夠全面，coding途中還是不停的塗塗改改，架構也沒確定好，非常沒效率。但也學到了ai的一些基本觀念。

剩餘的問題:

如何更好地把資料分類、去除不必要的可能。

未來的想法:

希望作業內容能講清楚一些，希望能具體說明該如何把演算法(forward check、ac3)融入到作業中。

0416313 何元斌

附件(code)

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class puzzle\_info {

public:

int start\_x;

int start\_y;

int length;

int direction;

};

class node {

public:

int puzzle\_index;

string word;

};

class binary\_constraint {

public:

int puzzle1;

int puzzle2;

int index1;

int index2;

int ban\_index;

char ban\_list[5];

};

puzzle\_info p\_info[50];

node stack[10000];

string word\_info[3000];

binary\_constraint bin[20];

int puzzle\_count;

int bin\_count;

int board[100][100];

int index[21];

int top, x, y;

int visited\_node;

void init() {

visited\_node = 0;

top = -1;

for (int i = 0; i < 100; i++)

for (int j = 0; j < 100; j++)

board[i][j] = 0;

for (int i = 0; i < 20; i++) {

bin[i].ban\_index = 0;

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

bin[i].ban\_list[j] = 0;

}

}

int read\_puzzle(int num) {

fstream file;

int i;

char buf[200];

file.open("puzzle.txt", ios::in);

for (int i = 0; i < num;i++) file.getline(buf, sizeof(buf));

for (i = 0; buf[i] != 0; i = i + 10) {

p\_info[i / 10].start\_x = atoi(&buf[i]);

p\_info[i / 10].start\_y = atoi(&buf[i+2]);

p\_info[i / 10].length = atoi(&buf[i+4]);

if(buf[i+6] == 'A') p\_info[i / 10].direction = 0;

else p\_info[i / 10].direction = 1;

}

file.close();

for (i = 0; p\_info[i].length != 0; i++) {}

return i;

}

void read\_word() {

fstream file;

string buf;

file.open("English words 3000.txt", ios::in);

for (int i = 0; i < 3000; i++) {

getline(file, buf);

word\_info[i] = buf;

}

}

void count\_sort() {

string word\_info\_sorted[3000];

int index\_tmp[21];

for (int i = 0; i <= 20; i++) index[i] = 0;

for (int i = 0; i < 3000; i++) {

switch (word\_info[i].size())

{

case 1 : index[ 1]++; break;

case 2 : index[ 2]++; break;

case 3 : index[ 3]++; break;

case 4 : index[ 4]++; break;

case 5 : index[ 5]++; break;

case 6 : index[ 6]++; break;

case 7 : index[ 7]++; break;

case 8 : index[ 8]++; break;

case 9 : index[ 9]++; break;

case 10: index[10]++; break;

case 11: index[11]++; break;

case 12: index[12]++; break;

case 13: index[13]++; break;

case 14: index[14]++; break;

case 15: index[15]++; break;

case 16: index[16]++; break;

case 17: index[17]++; break;

case 18: index[18]++; break;

case 19: index[19]++; break;

case 20: index[20]++; break;

default:

break;

}

}

for (int i = 2; i <= 20; i++) {

index[i] = index[i] + index[i - 1];

}

for (int i = 20; i >= 0; i--) {

index[i] = index[i - 1];

}

index[1] = 0;

for (int i = 1; i <= 20; i++) {

index\_tmp[i] = index[i];

}

for (int i = 0; i < 3000; i++) {

word\_info\_sorted[index\_tmp[word\_info[i].size()]] = word\_info[i];

index\_tmp[word\_info[i].size()]++;

}

for (int i = 0; i < 3000; i++) word\_info[i] = word\_info\_sorted[i];

}

int com\_binary\_constraint() {

int x, y, tmp, count;

count = 0;

for (int i = 0; p\_info[i].length != 0; i++) {

x = p\_info[i].start\_x;

y = p\_info[i].start\_y;

if (p\_info[i].direction == 0) {

for (int j = 0; j < p\_info[i].length; j++) {

if (board[x + j][y] == 0) {

board[x + j][y] = i \* 100 + j;

}

else {

tmp = board[x + j][y]; //tmp / 100 = puzzle\_index, tmp % 100 = word\_index

for (int w = 0; w < count; w++) {

if ((bin[count].puzzle1 == i) && (bin[count].puzzle2 == (tmp / 100))) continue;

}

bin[count].puzzle1 = tmp / 100;

bin[count].index1 = tmp % 100;

bin[count].puzzle2 = i;

bin[count].index2 = j;

count++;

}

}

}

else {

for (int j = 0; j < p\_info[i].length; j++) {

if (board[x][y + j] == 0) {

board[x][y + j] = i \* 100 + j;

}

else {

tmp = board[x][y + j];

for (int w = 0; w < count; w++) {

if (bin[count].puzzle1 == i && bin[count].puzzle2 == tmp / 100) continue;

}

bin[count].puzzle1 = tmp / 100;

bin[count].index1 = tmp % 100;

bin[count].puzzle2 = i;

bin[count].index2 = j;

count++;

}

}

}

}

init();

return count;

}

string select\_word(int word\_num) {

string word = word\_info[word\_num];

return word;

}

int check\_board(int puzzle\_num, int word\_num) {

puzzle\_info puzzle = p\_info[puzzle\_num];

string word = select\_word(word\_num);

x = puzzle.start\_x;

y = puzzle.start\_y;

if (puzzle.direction == 0) {

for (int i = 0; i < puzzle.length; i++) {

if ((board[x + i][y] == 0) || (board[x + i][y] >= puzzle\_num \* 1000) || (board[x + i][y] % 1000 == word[i])) {}

else return 0;

}

}

else {

for (int i = 0; i < puzzle.length; i++) {

if ((board[x][y + i] == 0) || (board[x][y + i] >= puzzle\_num \* 1000) || (board[x][y + i] % 1000 == word[i])) {}

else return 0;

}

}

return 1;

}

void update\_board(int puzzle\_num, string word) {

puzzle\_info puzzle = p\_info[puzzle\_num];

x = puzzle.start\_x;

y = puzzle.start\_y;

//cout << word << endl;

if (puzzle.direction == 0) {

for (int i = 0; i < puzzle.length; i++) {

if(board[x + i][y] % 1000 != word[i]) board[x + i][y] = puzzle\_num \* 1000 + word[i];

}

}

else {

for (int i = 0; i < puzzle.length; i++) {

if (board[x][y + i] % 1000 != word[i]) board[x][y + i] = puzzle\_num \* 1000 + word[i];

}

}

}

void forward\_info(node current\_node) {

int puzzle\_num = current\_node.puzzle\_index;

string word = current\_node.word;

for (int i = 0; i < bin\_count; i++) {

cout << i << ' ' << bin[i].ban\_index << ' ' << bin[i].index1 << endl;

if (bin[i].puzzle1 == puzzle\_num) {

for (int j = 0; j < bin[i].ban\_index; j++)

if (bin[i].ban\_list[j] == word[bin[i].index1])

return;

bin[i].ban\_list[bin[i].ban\_index] = word[bin[i].index1];

bin[i].ban\_index++;

}

else if (bin[i].puzzle2 == puzzle\_num) {

for (int j = 0; j < bin[i].ban\_index; j++)

if (bin[i].ban\_list[j] == word[bin[i].index1])

return;

bin[i].ban\_list[bin[i].ban\_index] = word[bin[i].index2];

bin[i].ban\_index++;

}

}

}

int check\_forward(int puzzle\_num, int word\_num) {

puzzle\_info puzzle = p\_info[puzzle\_num];

string word = select\_word(word\_num);

for (int i = 0; i < bin\_count; i++) {

if (bin[i].puzzle1 == puzzle\_num) {

for (int j = 0; j < bin[i].ban\_index; j++) {

if (word[bin[i].index1] == bin[i].ban\_list[j])

return 0;

}

}

else if (bin[i].puzzle2 == puzzle\_num) {

for (int j = 0; j < bin[i].ban\_index; j++) {

if (word[bin[i].index2] == bin[i].ban\_list[j])

return 0;

}

}

}

return 1;

}

node pop() {

node current\_node;

current\_node = stack[top];

top--;

//forward\_info(current\_node);

update\_board(current\_node.puzzle\_index, current\_node.word);

return current\_node;

}

void push(node current\_node, int word\_num) {

node tmp = current\_node;

string word = select\_word(word\_num);

top++;

stack[top].puzzle\_index = current\_node.puzzle\_index + 1;

stack[top].word = word;

}

void push\_child(node current\_node) {

puzzle\_info tmp = p\_info[current\_node.puzzle\_index + 1];

if (tmp.length != 0) {

for (int i = index[tmp.length]; i < index[tmp.length + 1]; i++) { //push all childs

if (check\_board(current\_node.puzzle\_index + 1, i)) {

push(current\_node, i);

}

}

}

}

void print\_solution() {

for (int j = 0; j < 6; j++) {

for (int i = 0; i < 25; i++) {

if (board[i][j] != 0) cout << char(board[i][j] % 1000);

else cout << ' ';

}

cout << endl;

}

}

int match() {

int sol = 0;

node current\_node;

current\_node.puzzle\_index = -1;

push\_child(current\_node);

while (1) {

current\_node = pop();

if (current\_node.puzzle\_index == puzzle\_count - 1) {

print\_solution();

sol++;

break;

}

else if (top <= 0) {

break;

}

else {

push\_child(current\_node);

}

visited\_node++;

//cout << current\_node.puzzle\_index << ' ' << top << endl;

}

return sol;

}

int main(void){

int sol;

puzzle\_count = read\_puzzle(4);

read\_word();

count\_sort();

init();

bin\_count = com\_binary\_constraint() + 1;

sol = match();

if (sol) {

cout << "solution found!!" << endl << endl;

cout << "visited node:" << visited\_node << endl;

cout << sol << endl;

}

else {

cout << "no solution found!!" << endl;

}

system("pause");

return 0;

}