人工智慧導論 Program assignment#2

學號：411086006　　學生：張愷和

所有程式皆使用 C++，並用 wsl 上 linux 環境中的 gcc 編譯檔案

介紹一些宣告在 global scope 的參數：

```
const int N = 8;
int max_saved = 0;
```

N 棋盤大小

max_saved 儲存 state 狀態大小

```
class obj
{
public:
    int cost = 0;
    int board[N][N] = {};
    int hvalue = 0;
    void print()
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
            {
                if (board[i][j] == 1)
                    cout << "Q ";
                else
                    cout << ". ";
            }
            cout << endl;
        }
    }
};
```

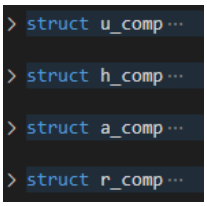
物件的一些參數

cost:state 到達的步數

board:state 現在的盤面

hvalue:state 的 heuristic 數值

print():印出 state 盤面，以 Q 表示 queen 位置，以 ． 表示空位



各個 search 的 priority_queue comp function

```
vector<obj> expand(obj node) // enpand 1 row
{
    vector<obj> child_nodes;
    int c = node.cost;
    int a[N] = {};
    for (int i = 0; i < N; i++) // 判斷哪個row上面已經有queen
    {
        for (int j = 0; j < N; j++)
        {
            if (node.board[i][j] == 1)
            {
                a[i] = 1;
                break;
            }
        }
    }
    obj child;
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            child.board[i][j] = node.board[i][j]; // 複製parent盤面
        }
    }
    for (int i = 0; i < N; i++)
    {
        if (a[i]) // i row 上有queen跳過,直到第i row上沒有queen,expand那行
            continue;
        else
        {
            for (int y = 0; y < N; y++)
            {
                child.board[i][y] = 1;
                child.cost = c + 1;
                child_nodes.push_back(child);
                child.board[i][y] = 0;
                max_saved++;
            }
        }
        break;
    }
    return child_nodes;
}
```

將輸入的 parent node 展開為 child node，將以展開的 child 輸入 vector<obj> child_node 回傳至各個 search 去判斷，首先先使用一個 array 紀錄有哪些 row 已經有放 queen，接著將 child 複製 parent 盤面，對 child 下一行(沒有 queen 的那行)放置 queen，因此假設為 8*8 的盤面，就會產生 8 個 child，我在對放置 queen 再多做一下解釋，假設第一行以及第三行有放 queen，那就會先從第二行 expand，假設第一行以及第二行有放 queen，就會先從第三行放 queen。

```
bool is_valid(obj node)
{
    for (int i = 0; i < N; i++)
    {
        int f = 0;
        for (int j = 0; j < N; j++)
        {
            if (node.board[i][j] == 1)
            {
                f = j;
                break;
            }
            else
                f = -1;
        }
        if (f != -1)
        {
            for (int x = 0; x < N; x++) // 判斷row有沒有queen…
            for (int y = 0; y < N; y++) // 判斷column有沒有queen…
            int x = i;
            int y = f;
            while (x != -1 && y != -1) // 判斷左上對角有沒有queen…
            x = i;
            y = f;
            while (x != -1 && y != N) // 判斷左下對角有沒有queen…
            x = i;
            y = f;
            while (x != N && y != -1) // 判斷右上對角有沒有queen…
            x = i;
            y = f;
            while (x != N && y != N) // 判斷右下對角有沒有queen…
        }
    }
    return true;
}
```

判斷盤面上的每個 queen 會不會互相攻擊，f 是記錄在第 i row 上的 queen 在哪一 column。

```
bool is_goal(obj node)
{
    int count = 0;
    if (!is_valid(node))
        return false;
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (node.board[i][j] == 1)
                count++;
        }
    }
    if (count == N)
        return true;
    return false;
}
```

判斷有沒有達到 goal 的狀態，先判斷 state 的盤面有沒有解，有解就判斷 queen 的數量有多少，如果數量到達 8 個(8*8)或 15 個(15*15)，就回傳有解。

```
int heuristic(obj node)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (node.board[i][j] == 1)
                node.hvalue++;
        }
    }
    return -1;
}
```

計算現在盤面上有多少 queen，越多 queen 就先 search，當作 heuristic value。

```
int ucs(obj node)
{
    priority_queue<obj, vector<obj>, u_comp> pq;
    pq.push(node);
    max_saved++;
    while (!pq.empty())
    {
        int sz = pq.size();
        for (int i = 0; i < sz; i++)
        {
            obj cur = pq.top();
            pq.pop();
            if (is_goal(cur) && is_valid(cur))
            {
                cur.print();
                cout << "Step : " << cur.cost << endl;
                cout << "Maximum saved state : " << max_saved << endl;
                cout << endl;
                max_saved = 0;
                return 1;
            }

            vector<obj> child = expand(cur);
            for (auto i : child)
            {
                if (is_valid(i))
                    pq.push(i);
            }
        }
    }
    cout << "No solution" << endl;
    return -1;
}
```

Ucs ，先將 start 輸入進 priority_queue，將 max_state 加 1，紀錄初始盤面，接著先去判斷有沒有解，如果有解就把當前 state 經過的 cost、expand 紀錄的盤面數值以及 goal state 盤面輸出，如果沒有解，就把 parent 展開，接著判斷每個 child 是否有解，如果有解就 push 進 pq，如果沒解就忽略，到最後 priority_queue search 完沒有解，輸出 No solution。

```
int ids(obj node)
{
    int res;
    for (int l = N; l < 30; l++)
    {
        res = dls(node, l);
        cout << "The limit is " << l << endl;
        if (res == 1)
        {
            cout << endl;
            break;
        }
        if (l == 29)
            cout << "No solution" << endl;
    }
    return res;
}
```

Ids ，限制 dls 深度，如果在限定深度沒有找到解，就將深度+1，然後繼續重新 search。

```
int dls(obj node, int limit)
{
    stack<obj> s({node});
    while (!s.empty())
    {
        int sz = s.size();
        for (int i = 0; i < sz; i++)
        {
            obj cur = s.top();
            s.pop();
            if (is_goal(cur) && is_valid(cur))
            {
                cur.print();
                cout << "Step : " << cur.cost << endl;
                cout << "Maximum saved state : " << max_saved << endl;
                max_saved = 0;
                return 1;
            }

            if (cur.cost <= limit)
            {
                vector<obj> child = expand(cur);
                for (auto i : child)
                {
                    if (is_valid(i))
                        s.push(i);
                }
            }
        }
    }
    return -1;
}
```

dls ，先使用 stack 紀錄 start state，將 max_state 加 1，紀錄初始盤面，用 stack 可以達到讓第一個 state 一直往下搜索的狀況，接著先去判斷有沒有解，如果有解就把當前 state 經過的 cost、expand 紀錄的盤面數值以及 goal state 盤面輸出，如果沒有解，接著判斷 state search 的深度是否到了限制深度，如果沒有就把 parent 展開，接著判斷每個 child 是否有解，如果有解就 push 進 stack，如果沒解就忽略，到最後 stack search 完沒有解，輸出 No solution。

```
int greedy(obj node)
{
    priority_queue<obj, vector<obj>, h_comp> pq;
    pq.push(node);
    max_saved++;
    while (!pq.empty())
    {
        int sz = pq.size();
        for (int i = 0; i < sz; i++)
        {
            obj cur = pq.top();
            pq.pop();
            if (is_goal(cur) && is_valid(cur))
            {
                cur.print();
                cout << "Step : " << cur.cost << endl;
                cout << "Maximum saved state : " << max_saved << endl;
                cout << endl;
                max_saved = 0;
                return 1;
            }

            vector<obj> child = expand(cur);
            for (auto i : child)
            {
                if (is_valid(i))
                {
                    heuristic(i);
                    pq.push(i);
                }
            }
        }
    }
    cout << "No solution" << endl;
    return -1;
}
```

Greedy search，search 的方法跟 ucs 差不多，只是將 cost 的優先大小排列，改成以 heuristic value 優先大小排列，而 heuristic 計算的方法，是在判斷要不要儲存 expand 出來的 child 時，如果 child 有解，就先計算 child 的 heuristic 數值，然後 push 進 pq 繼續 search。

```
int Astar(obj node)
{
    priority_queue<obj, vector<obj>, a_comp> pq;
    pq.push(node);
    max_saved++;
    while (!pq.empty())
    {
        int sz = pq.size();
        for (int i = 0; i < sz; i++)
        {
            obj cur = pq.top();
            pq.pop();
            if (is_goal(cur) && is_valid(cur))
            {
                cur.print();
                cout << "Step : " << cur.cost << endl;
                cout << "Maximum saved state : " << max_saved << endl;
                cout << endl;
                max_saved = 0;
                return 1;
            }

            vector<obj> child = expand(cur);
            for (auto i : child)
            {
                if (is_valid(i))
                {
                    heuristic(i);
                    pq.push(i);
                }
            }
        }
    }
    cout << "No solution" << endl;
    return -1;
}
```

Astar search ，跟 greedy search 差不多，只是在 pq 排列的方法，由 hvalue 小到大優先排列，更改為 hvlaue+cost 大小，由小到大優先排列，排列完後就將最前面的 state 進行 search。

```
int rbfs(obj node, int f_limit)
{
    if (is_goal(node) && is_valid(node))
    {
        node.print();
        cout << "Step : " << node.cost << endl;
        cout << "Maximum saved state : " << max_saved << endl;
        cout << endl;
        max_saved = 0;
        return 1;
    }
    priority_queue<obj, vector<obj>, r_comp> pq;
    vector<obj> child = expand(node);
    for (auto i : child)
    {
        if (is_valid(i))
        {
            heuristic(i);
            pq.push(i);
        }
    }
    if (pq.empty())
        return 1e9;
    while (!pq.empty())
    {
        obj best = pq.top();
        pq.pop();
        if (best.hvalue > f_limit)
            return best.hvalue;
        if (best.hvalue == 1e9)
            return 1e9;
        obj alternative = pq.top();
        int res = rbfs(best, min(f_limit, alternative.hvalue));
        if (res == 1)
            return 1;
        best.hvalue = res;
        pq.push(best);
    }
    return 1e9;
}
```

RBFS，主要的概念是將 pq 中最佳以及第二佳解的 state 留下去用 recursive expand，而剩下的拋棄掉，不過在這題目中，拋棄掉的狀況有可能是最佳解，因此最後找到的解有可能不是最佳解，來介紹一下程式好了，首先判斷有沒有解，如果沒解就對 node 進行 expand，接著判斷 child

有沒有解，有解就 push 進 pq search，接著就進 while 迴圈進行挑選最佳以及第二佳的 state，接著將這兩個 state 進入 recursive 下去 expand，直到找到解，如果沒有找到解就 return 1e9，return 1e9 到 RBFS function，輸出 No solution。

```
int main()
{
    int n;
    cout << "Enter the number of queen:" << endl;
    cin >> n;
    obj start;
    cout << "Enter the location of each queen:" << endl;
    for (int i = 0; i < n; i++)
    {
        int x, y;
        cin >> x >> y;
        start.board[x][y] = 1;
    }
    start.print();
    cout << endl;
    if (!is_valid(start))
        cout << "No solution!" << endl;
    else
    {
        cout << "UCS:" << endl;
        ucs(start);
        cout << "IDS:" << endl;
        ids(start);
        cout << "Greedy search:" << endl;
        greedy(start);
        cout << "Astar search:" << endl;
        Astar(start);
        cout << "RBFS:" << endl;
        RBFS(start);
    }
    return 0;
}
```

在 main function 裡就輸入有多少 queen 以及排列 queen 的位置，接著判斷初始盤面有沒有解，如果有解就進行每一個 search，如果沒有解，就輸出 No solution。

至於 15-queen，就只是把 global scope 宣告的 N = 8 改成 N = 15，15-queen 就可以解了。