1. 硬體規格

- 筆電
  - CPU : Intel(R) Core(TM) i7-10510U CPU @ 1.60GHz (8核)
  - 顯卡 : NVIDIA GeForce MX250
  - RAM : 12GB
- 桌電
  - CPU : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz (8核)
  - 顯卡：NVDIA GeForce GTX 1060 3GB
  - RAM : 16GB

桌電大概七年前買的，顯卡後來有更新過，筆電是兩年前買的。作業系統皆為 Windows10 ，沒有其他規格的電腦可以選了，如果筆電顯卡跑不動實驗，會讓桌機去跑。

電話：0901228989

2. 輸入檔

easy_input.txt

```
5 4
1 2 2 3
1 2 2 3
5 5 0 6
4 8 0 6
4 7 910
```

medium_input.txt

```
5 4
0 1 2 2
0 1 2 2
5 5 6 3
4 8 6 3
4 7 910
```

hard_input.txt

```
5 4
1 2 2 0
1 2 2 3
5 5 8 3
4 9 0 6
4 710 6
```

no_answer_input.txt

```
5 4
3 2 2 1
3 2 2 1
4 5 5 6
4 0 0 6
7 8 910
```

製作方式皆為隨便移動步數，越少的越簡單

3. IDS

使用方式：

1. terminal 輸入 python3 ids.py 後會跳出以下訊息

```
Please enter your input file name(ex. input.txt):
```

2. 按照要求輸入輸入檔名稱

```
Please enter your input file name(ex. input.txt): easy_input.txt
```

3. 輸出結果，結果會自動存取於 output.txt

```
λ python3 ids.py

Please enter your input file name(ex. input.txt): easy_input.txt
complete reading input, use time: 0.0 seconds

starting ids, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDS

Total run time = 0.004985809326171875 seconds.
An optimal solution has 4 moves:
5R 4U 7L 8D
```

source code:

```python
import time

num_map = []
ans_map = [[1, 2, 2, 3], [1, 2, 2, 3], [4, 5, 5, 6], [4, 0, 0, 6], [7, 8, 9, 10]]
# decide max depth for IDS
max_depth = 12

height = 0
width = 0

# check if the move is valid
def checkMove(num_map, num, dir):
    for y in range(height):
        for x in range(width):
```

```
            if dir == 'R':
                if num_map[y][x] == num and x == width - 1:
                    return False
                if num_map[y][x] == num and (num_map[y][x + 1] != num and num_map[y][x + 1] != 0):
                    return False
            elif dir == 'L':
                if num_map[y][x] == num and x == 0:
                    return False
                if num_map[y][x] == num and (num_map[y][x - 1] != num and num_map[y][x - 1] != 0):
                    return False
            elif dir == 'U':
                if num_map[y][x] == num and y == 0:
                    return False
                if num_map[y][x] == num and (num_map[y - 1][x] != num and num_map[y - 1][x] != 0):
                    return False
            elif dir == 'D':
                if num_map[y][x] == num and y == height - 1:
                    return False
                if num_map[y][x] == num and (num_map[y + 1][x] != num and num_map[y + 1][x] != 0):
                    return False
    return True

# move the number to the direction
def move(num_map, num, dir):
    ret_map = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
    for y in range(height):
        for x in range(width):
            if num_map[y][x] != num and num_map[y][x] != 0:
                ret_map[y][x] = num_map[y][x]
            elif num_map[y][x] == num:
                if dir == 'L':
                    ret_map[y][x - 1] = num
                elif dir == 'R':
                    ret_map[y][x + 1] = num
                elif dir == 'U':
                    ret_map[y - 1][x] = num
                elif dir == 'D':
                    ret_map[y + 1][x] = num
    return ret_map

def DLS(num_map, limit, sol):
    if limit == 0 and num_map == ans_map:
        return True, sol
    elif limit == 0:
        return False, sol

    possible_move = []
    # find all possible moves in number 1-10
    for num in range(1, 11):
        if checkMove(num_map, num, 'R'):
            possible_move.append([str(num),'R'])
        if checkMove(num_map, num, 'L'):
            possible_move.append([str(num),'L'])
```

```
        if checkMove(num_map, num, 'U'):
            possible_move.append([str(num),'U'])
        if checkMove(num_map, num, 'D'):
            possible_move.append([str(num),'D'])
    # check every possible move can lead to final answer or not
    for act in possible_move:
        next_num_map = move(num_map, int(act[0]), act[1])
        fix, ret_sol = DLS(next_num_map, limit - 1, sol + [act])
        if fix:
            return True, ret_sol

    return False, sol


def IDS(num_map):
    for i in range(0, max_depth):
        fix, sol = DLS(num_map, i, [])
        if fix:
            return sol
    return []

# reading input
print('\nPlease enter your input file name(ex. input.txt): ', end='')
input_path = input()

start = time.time()
with open(input_path, 'r') as f:
    tmp = f.read()
    input_list = tmp.split('\n')
    for i in range(len(input_list)):
        if i == 0:
            height = int(input_list[0][0:2])
            width = int(input_list[0][2:4])
        else:
            l = []
            for j in range(0, len(input_list[i]), 2):
                num = int(input_list[i][j:j + 2])
                l.append(num)
            num_map.append(l)

now_time = time.time()

print('complete reading input, use time: ' + str(now_time - start) + '
seconds')

print('\nstarting ids, static ans: ')
for i in ans_map:
    for j in i:
        print(j, end = ' ')
    print()
print()

sol = IDS(num_map)
print('finish IDS\n')
end = time.time()

with open('output.txt', 'w') as f:
    f.write('Total run time = ' + str(end - start) + ' seconds.\n')
```

```
125     print('Total run time = ' + str(end - start) + ' seconds.')
126 
127     if len(sol) == 0:
128         f.write('There\'s no optimal solution in ' + str(max_depth) + '
    moves\n')
129         print('There\'s no optimal solution in ' + str(max_depth) + '
    moves')
130     else:
131         f.write('An optimal solution has ' + str(len(sol)) + ' moves:\n')
132         print('An optimal solution has ' + str(len(sol)) + ' moves:')
133         for i in sol:
134             f.write(str(i[0])+i[1]+' ')
135             print(str(i[0])+i[1]+' ', end = '')
136         print()
```

時間複雜度為 $O(b^d)$，空間複雜度為 $O(b \times d)$

使用到的資料結構只有 list。

流程為每次找尋盤面上可以移動的方塊，並儲存在一個 list 裡，接著對於每個可以移動的動作進行下一步的探索。由於會造訪所有可能的移動方塊，且深度由淺至深，所以結果必為 optimal，一定能找到最小步數。

上方 source code 裡 max_depth = 12 是為了方便展示，由於時間複雜度很大，若用太多層時間會不夠跑，以下是針對每種測資的結果。

easy:

```
λ python3 ids.py

Please enter your input file name(ex. input.txt): easy_input.txt
complete reading input, use time: 0.0 seconds

starting ids, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDS

Total run time = 0.0050442218780517578 seconds.
An optimal solution has 4 moves:
5R 4U 7L 8D
```

medium:

```
λ python3 ids.py

Please enter your input file name(ex. input.txt): medium_input.txt
complete reading input, use time: 0.0 seconds

starting ids, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDS

Total run time = 0.05189371109008789 seconds.
An optimal solution has 9 moves:
1L 2L 3U 3U 6R 5R 4U 7L 8D
```

hard:

```
λ python3 ids.py

Please enter your input file name(ex. input.txt): hard_input.txt
complete reading input, use time: 0.0 seconds

starting ids, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDS

Total run time = 109.48106980323792 seconds.
An optimal solution has 11 moves:
3U 6U 8D 5R 4U 7L 9D 8L 10R 9R 8D
```

no answer:

```
λ python3 ids.py

Please enter your input file name(ex. input.txt): no_answer_input.txt
complete reading input, use time: 0.00099921226501464 84 seconds

starting ids, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDS

Total run time = 192.04747772216797 seconds.
There's no optimal solution in 12 moves
```

4. IDA*

使用方式：

1. terminal 輸入 python3 ida.py 後會跳出以下訊息

```
Please enter your input file name(ex. input.txt):
```

2. 按照要求輸入輸入檔名稱

```
Please enter your input file name(ex. input.txt): easy_input.txt
```

3. 輸出結果，結果會自動存取於 output.txt

```
λ python3 ida.py

Please enter your input file name(ex. input.txt): easy_input.txt
complete reading input, use time: 0.0 seconds

starting ida, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDA

Total run time = 0.0010004043579101562 seconds.
An optimal solution has 4 moves:
5R 4U 7L 8D
```

source code:

```python
import time

num_map = []
ans_map = [[1, 2, 2, 3], [1, 2, 2, 3], [4, 5, 5, 6], [4, 0, 0, 6], [7, 8,
9, 10]]
```

```
ans_coor = [[0, 0], [0, 1], [0, 3], [2, 0], [2, 1], [2, 3], [4, 0], [4, 1], [4, 2], [4, 3]]
# decide max depth for IDA
max_limit = 12

height = 0
width = 0

# count the H() of the num_map
def getH(num_map):
    num_apper = [False]*10
    cnt = 0
    for y in range(height):
        for x in range(width):
            num = num_map[y][x]
            if num != 0:
                if num_apper[num - 1] == False:
                    num_apper[num - 1] = True
                    coor = ans_coor[num - 1]
                    cnt += abs(y - coor[0]) + abs(x - coor[1])
    return cnt


# check if the move is valid
def checkMove(num_map, num, dir):
    for y in range(height):
        for x in range(width):
            if dir == 'R':
                if num_map[y][x] == num and x == width - 1:
                    return False
                if num_map[y][x] == num and (num_map[y][x + 1] != num and num_map[y][x + 1] != 0):
                    return False
            elif dir == 'L':
                if num_map[y][x] == num and x == 0:
                    return False
                if num_map[y][x] == num and (num_map[y][x - 1] != num and num_map[y][x - 1] != 0):
                    return False
            elif dir == 'U':
                if num_map[y][x] == num and y == 0:
                    return False
                if num_map[y][x] == num and (num_map[y - 1][x] != num and num_map[y - 1][x] != 0):
                    return False
            elif dir == 'D':
                if num_map[y][x] == num and y == height - 1:
                    return False
                if num_map[y][x] == num and (num_map[y + 1][x] != num and num_map[y + 1][x] != 0):
                    return False
    return True

# move the number to the direction
def move(num_map, num, dir):
    ret_map = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
    for y in range(height):
```

```
        for x in range(width):
            if num_map[y][x] != num and num_map[y][x] != 0:
                ret_map[y][x] = num_map[y][x]
            elif num_map[y][x] == num:
                if dir == 'L':
                    ret_map[y][x - 1] = num
                elif dir == 'R':
                    ret_map[y][x + 1] = num
                elif dir == 'U':
                    ret_map[y - 1][x] = num
                elif dir == 'D':
                    ret_map[y + 1][x] = num
    return ret_map

def DFS(num_map, flimit, sol, g, next_f):
    h = getH(num_map)
    F = g + h
    if num_map == ans_map:
        return sol, min(next_f, F)
    if F > flimit:
        return [], min(next_f, F)

    possible_move = []
    # find all possible moves in number 1-10
    for num in range(1, 11):
        if checkMove(num_map, num, 'R'):
            possible_move.append([str(num),'R'])
        if checkMove(num_map, num, 'L'):
            possible_move.append([str(num),'L'])
        if checkMove(num_map, num, 'U'):
            possible_move.append([str(num),'U'])
        if checkMove(num_map, num, 'D'):
            possible_move.append([str(num),'D'])
    # check every possible move can lead to final answer or not
    for act in possible_move:
        next_num_map = move(num_map, int(act[0]), act[1])
        ret_sol, ret_F = DFS(next_num_map, flimit, sol + [act], g + 1,
next_f)
        if ret_F <= flimit:
            return ret_sol, min(ret_F, next_f)


    return sol, min(F, next_f)


def IDA(num_map):
    flimit = getH(num_map)
    while True:
        now_time = time.time()
        if now_time - start > 200.0:
            return []
        sol, new_flimit = DFS(num_map, flimit, [], 0, max_limit)
        if len(sol) != 0 and new_flimit == flimit:
            return sol
        flimit = min(new_flimit, max_limit)
        if flimit >= max_limit:
            return []

```

```
# reading input
print('\nPlease enter your input file name(ex. input.txt): ', end='')
input_path = input()

start = time.time()
with open(input_path, 'r') as f:
    tmp = f.read()
    input_list = tmp.split('\n')
    for i in range(len(input_list)):
        if i == 0:
            height = int(input_list[0][0:2])
            width = int(input_list[0][2:4])
        else:
            l = []
            for j in range(0, len(input_list[i]), 2):
                num = int(input_list[i][j:j + 2])
                l.append(num)
            num_map.append(l)

now_time = time.time()

print('complete reading input, use time: ' + str(now_time - start) + '
seconds')

print('\nstarting ida, static ans: ')
for i in ans_map:
    for j in i:
        print(j, end = ' ')
    print()
print()

sol = IDA(num_map)
print('finish IDA\n')
end = time.time()

with open('output.txt', 'w') as f:
    f.write('Total run time = ' + str(end - start) + ' seconds.\n')
    print('Total run time = ' + str(end - start) + ' seconds.')

    if len(sol) == 0:
        f.write('Can\'t find optimal solution in 200 seconds\n')
        print('Can\'t find optimal solution in 200 seconds\n')
    else:
        f.write('An optimal solution has ' + str(len(sol)) + ' moves:\n')
        print('An optimal solution has ' + str(len(sol)) + ' moves:')
        for i in sol:
            f.write(str(i[0])+i[1]+' ')
            print(str(i[0])+i[1]+' ', end = '')
        print()
```

時間複雜度為 $O(b^d)$，空間複雜度為 $O(b^d)$

使用到的資料結構只有 list。

H() 的設定為每個方塊與最終位置的絕對位置距離相加。

流程為每次找尋盤面上可以移動的方塊，並儲存在一個 list 裡，接著對於每個可以移動的動作進行下一步的探索。計算 F() 值若沒有超過 limit，則繼續下一步的探索。

除了設定最大步數外，在遇到無解答案時，用探索的時間做為 cutoff，若超過 200 秒則強制停止。以下是各測資的結果。

easy:

```
λ python3 ida.py

Please enter your input file name(ex. input.txt): easy_input.txt
complete reading input, use time: 0.0 seconds

starting ida, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDA

Total run time = 0.0010004043579101562 seconds.
An optimal solution has 4 moves:
5R 4U 7L 8D
```

medium:

```
λ python3 ida.py

Please enter your input file name(ex. input.txt): medium_input.txt
complete reading input, use time: 0.0 seconds

starting ida, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDA

Total run time = 0.0019948482513427734 seconds.
An optimal solution has 9 moves:
1L 2L 3U 3U 6R 5R 4U 7L 8D
```

hard:

```
λ python3 ida.py

Please enter your input file name(ex. input.txt): hard_input.txt
complete reading input, use time: 0.0 seconds

starting ida, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDA

Total run time = 0.0019941329956054688 seconds.
An optimal solution has 11 moves:
3U 6U 8D 5R 4U 7L 9D 8L 10R 9R 8D
```

no answer:

```
λ python3 ida.py

Please enter your input file name(ex. input.txt): no_answer_input.txt
complete reading input, use time: 0.0 seconds

starting ida, static ans:
1 2 2 3
1 2 2 3
4 5 5 6
4 0 0 6
7 8 9 10

finish IDA

Total run time = 200.00044631958008 seconds.
Can't find optimal solution in 200 seconds
```

5.

由於不會根據輸入的長、寬還有數字找到最終盤面，所以統一以作業二範例的方塊和最終盤面當作測資和目標。

沒有進行過多的輸入檢查，所以如果長寬不是 5×4 、或是數字不是 1-10 就會出錯。