資料結構

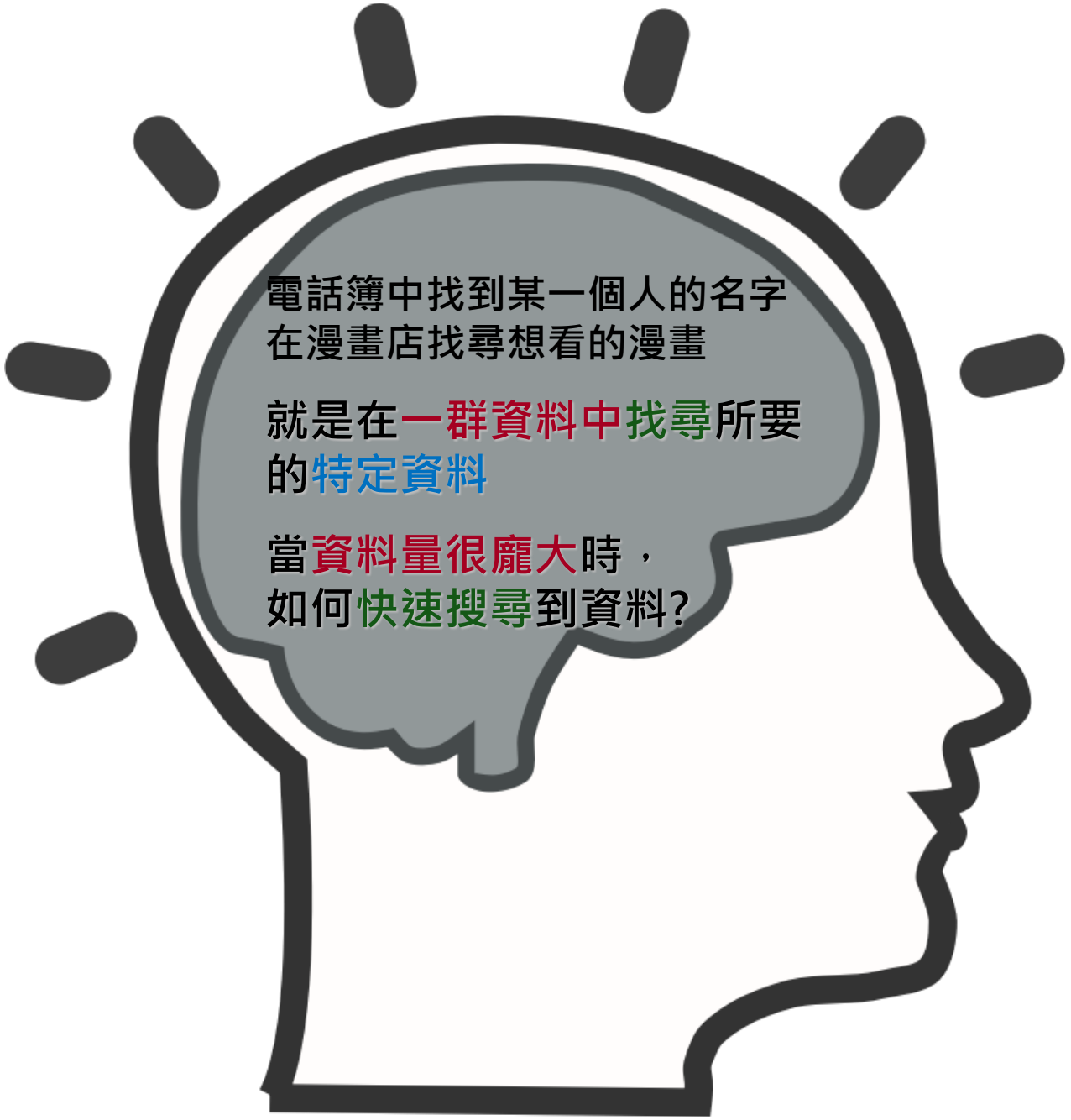
演算法

搜　　尋

S e a r c h

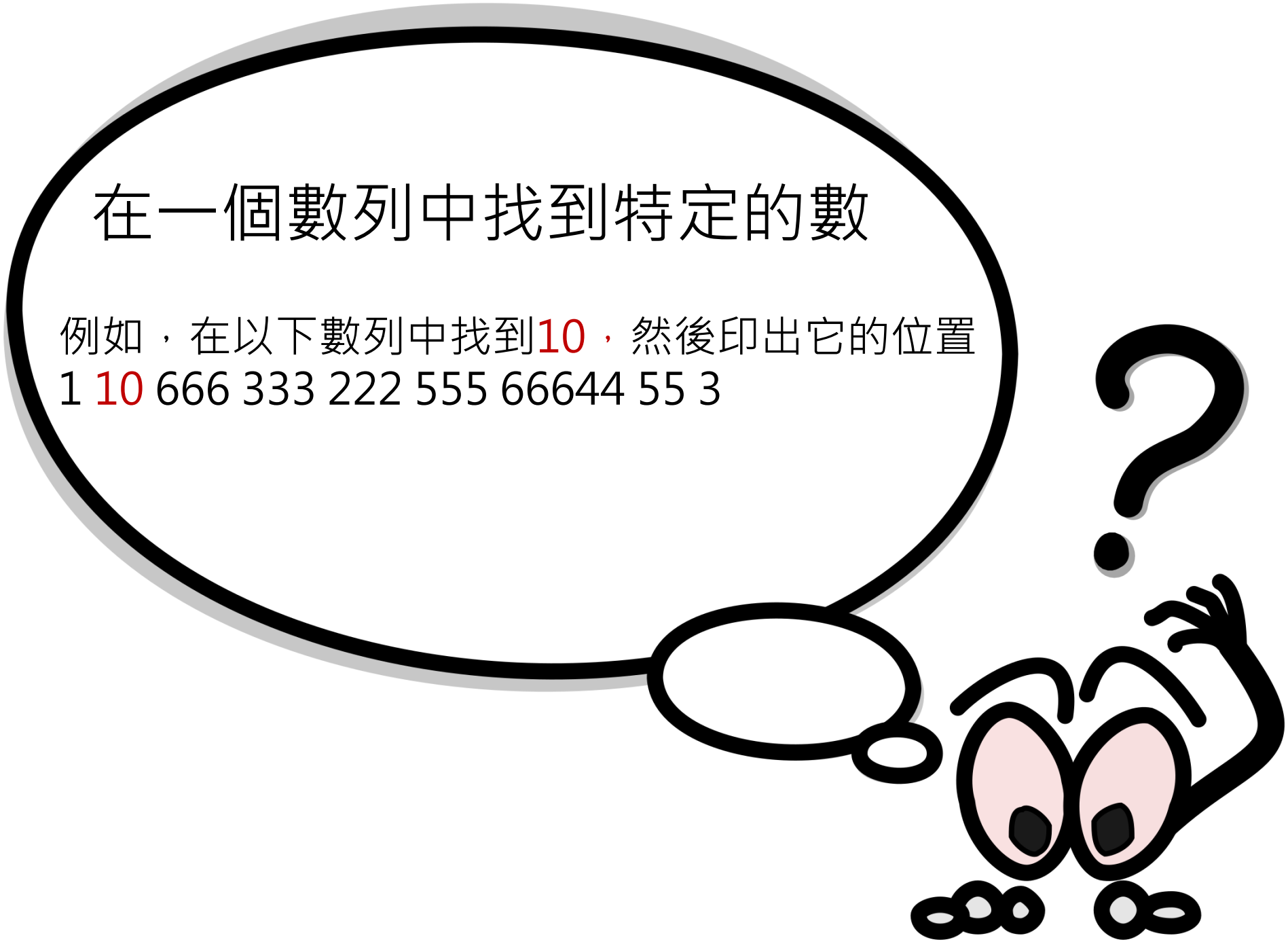
搜尋 Search

搜尋的目的


電話簿中找到某一個人的名字
在漫畫店找尋想看的漫畫
就是在一群資料中找尋所要的特定資料
當資料量很龐大時，
如何快速搜尋到資料?

在一個數列中找到特定的數

例如，在以下數列中找到10，然後印出它的位置
1 10 666 333 222 555 66644 55 3



可以這樣找...

方法:

拿10去比對數列中的數，
如果相等，
就印出值。

將方法變成程式碼

```c
#include <stdio.h>
int main()
{
  int i, j, k, arr[10];
  for(j=0; j<10; j++)
    scanf("%d", &arr[j]);
  scanf("%d", &k);

  for(j=0; j<10; j++) {
    if(k==arr[j]) {
      printf("在第 %d 個位置發現%d", j+1, k);
      break;
    }
  }

  if(j==i)
    printf("找不到");
  return 0;
}
```

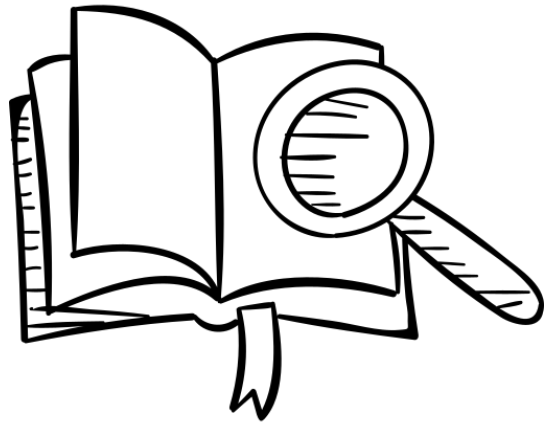
一筆一筆找

這是線性搜尋法

- 從第一個資料項開始依序取出與「目的資料項(鍵值Key)」相互比較，直到找出所要的元素或所有資料均已找完為止

- 也就是 **全部一個一個** 比較

- 循序搜尋法(Sequential Search)也稱為線性搜尋法(Linear Search)



搜尋也可以這樣找...

方法:

先將資料排序，
然後先跟中間比，
比中間值大就往下找，
比中間值小就往上找

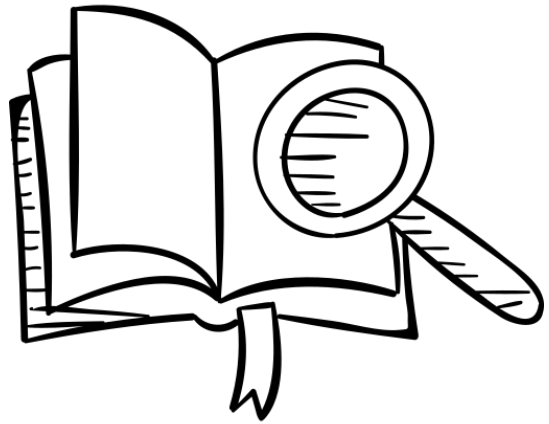
將方法變成程式碼

```c
#include <stdio.h>
int main()
{
  int i, low=0, high=9, mid, ans=7;
  int arr[10]={1,2,3,4,5,6,7,8,9,10};

  while(low<=high) {
    mid=(low+high)/2;
    if(arr[mid]==ans) {
      printf("在第 %d 個位置找到 %d", mid+1, ans);
      break;
    } else if(arr[mid] > ans) {
      high=mid-1;
    } else if(arr[mid]<ans) {
      low=mid+1;
    }
  }

  if(low>high)
   printf("找不到");

  return 0;
}
```

從中間比，根據比較結果往上(下)繼續找

這是二分搜尋法

- 步驟 1: 10筆資料要先排序
- 步驟 2: 將資料分割成兩部份，再利用「搜尋值」與「中間值」來比較大小，如果搜尋值小於中間值，則可確定要找的資料在前半段的元素中，反之則在後半段。
- 步驟 3: 持續分割直到找到或者不能再分割為止。



除了

- 循序搜尋
- 二分搜尋

還有

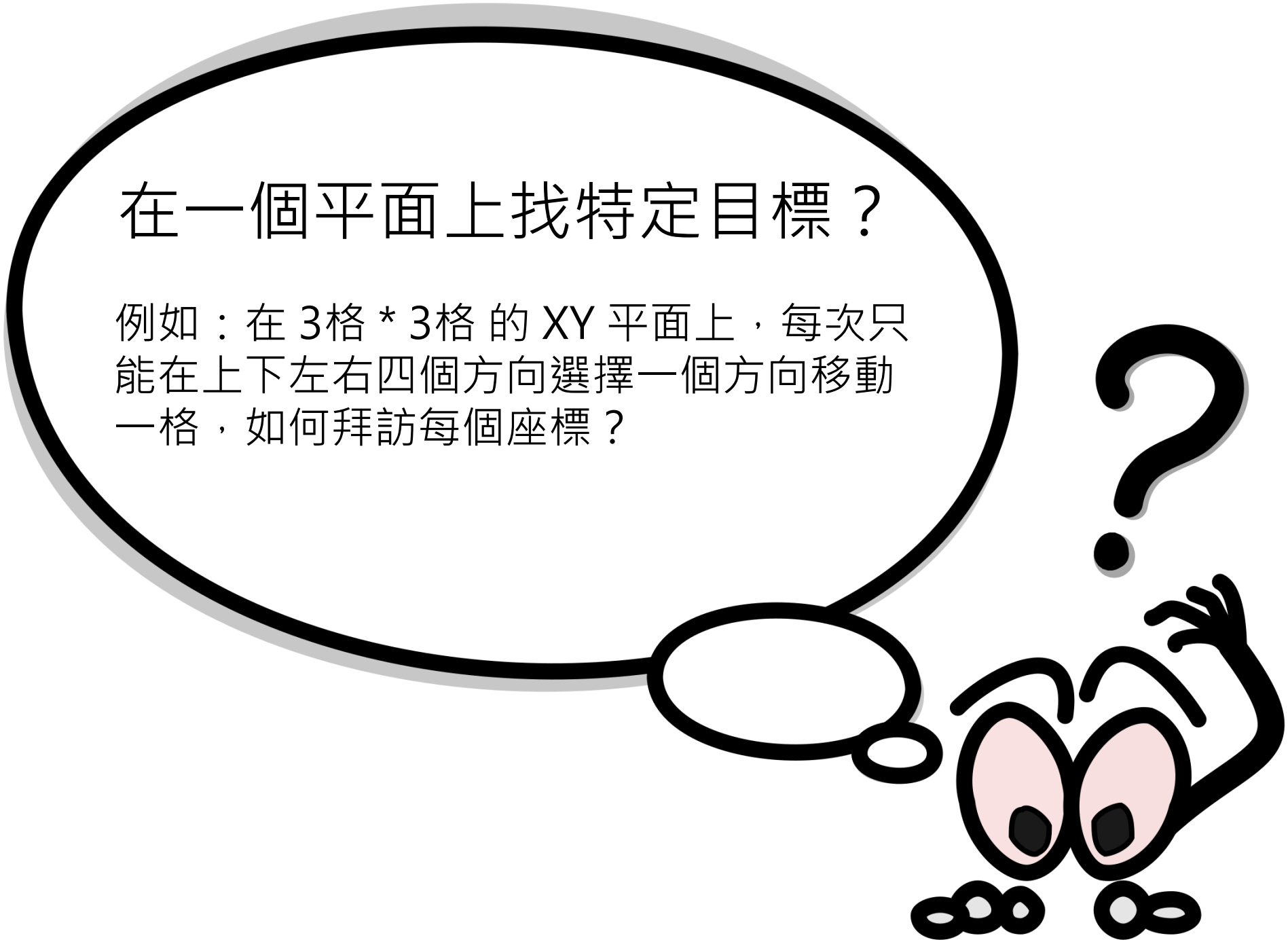
- 二元樹搜尋
- 雜湊搜尋
- 費式搜尋法

如果是
有限制的搜尋

循序搜尋法、二分搜尋法如何使用呢

在一個平面上找特定目標？

例如：在 3格 * 3格 的 XY 平面上，每次只能在上下左右四個方向選擇一個方向移動一格，如何拜訪每個座標？



可以這樣找...?

方法:

如同線性搜尋一樣
從起始點出發
走過每一點

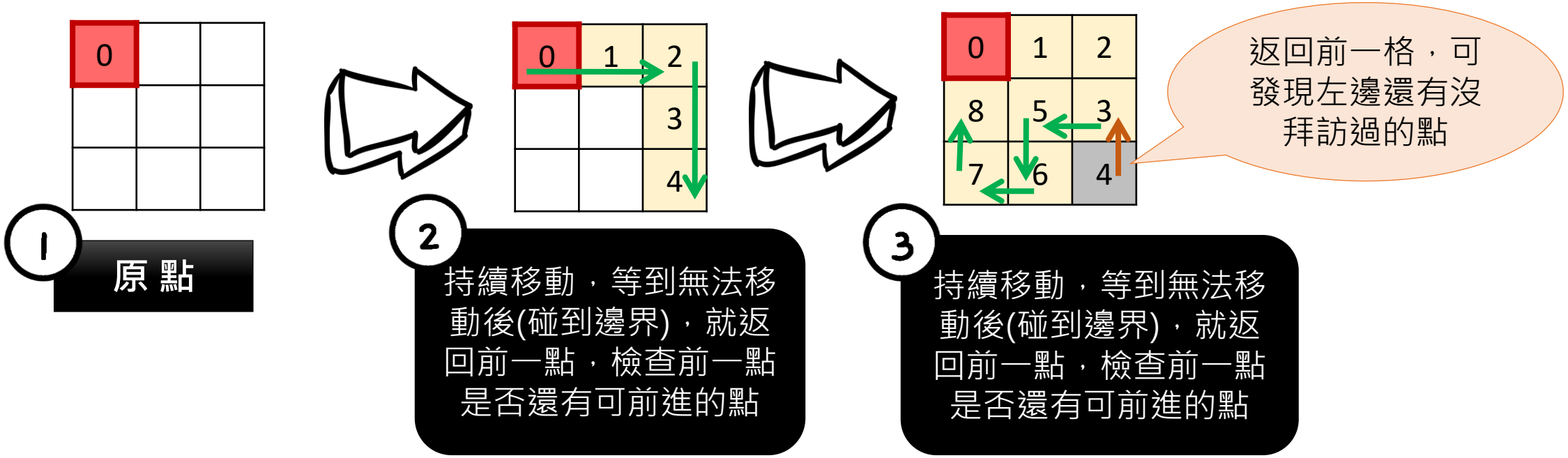
問題是，

要如何規律的線性走過每一點，
且避免同一點一直重複經過？

也就是說，要如何擴展呢？

第一種擴展方式

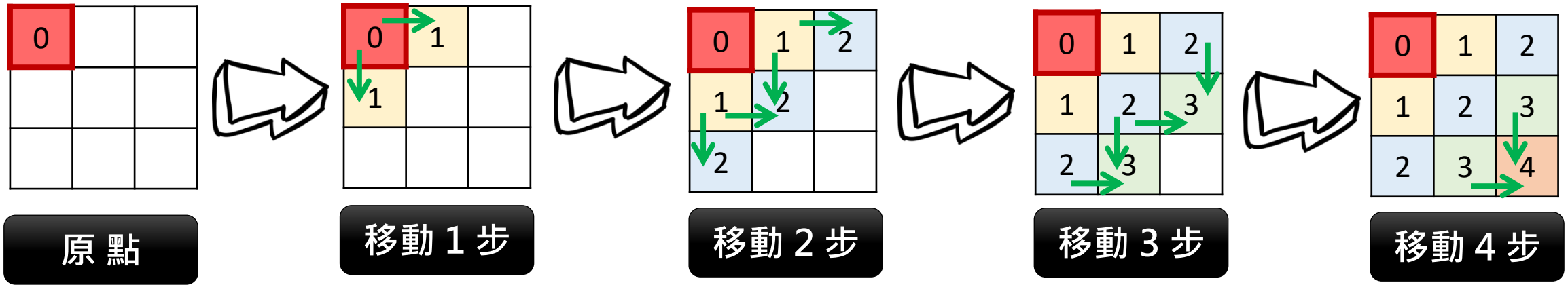
- 只要旁邊還有沒拜訪過的格子，就**持續前進**！
- 如果沒有可前進的格子，就往回退一格，找尋可前進的點



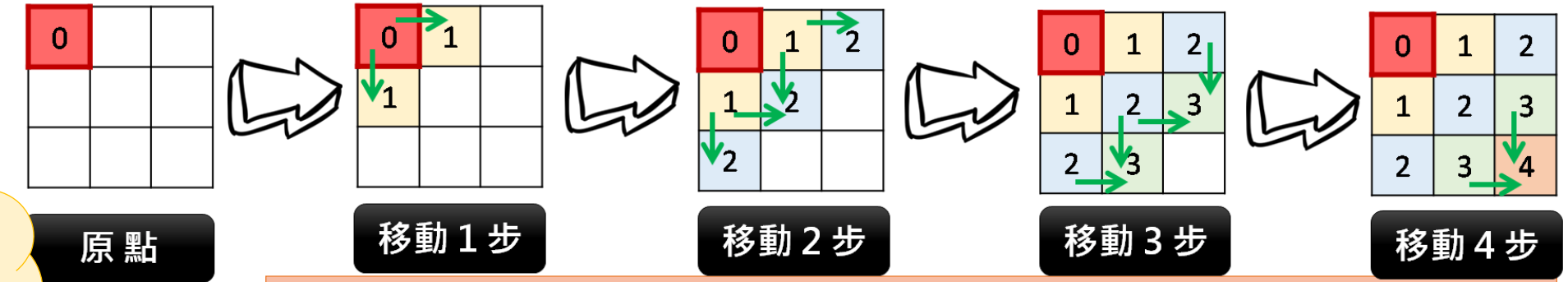
如何線性搜尋？

第二種擴展方式

- 先訪問從目前位置移動 1 步可到達的點
- 再訪問移動 2 步可到達的點，3 步可到達的點，以此類推

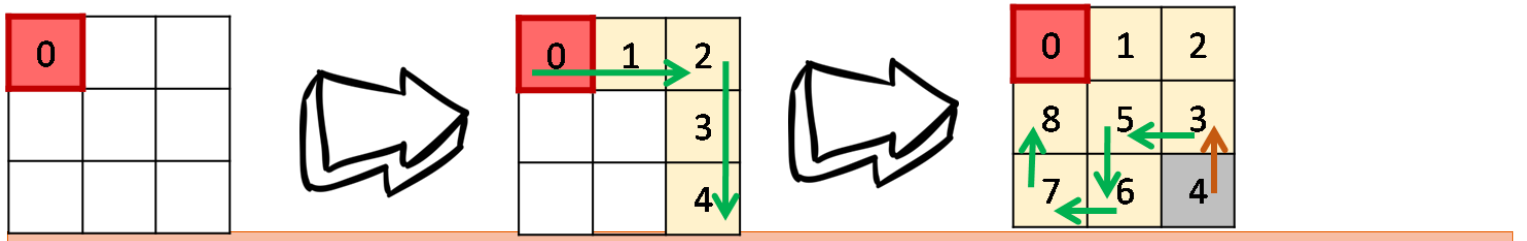


兩種擴展方式



由**中心往外擴展**的方式

廣度優先搜尋（Breadth-First Search, BFS）



深度優先搜尋（ Depth-First Search, DFS ）

持續前進的擴展方式

深度優先搜尋 DFS
depth-First Search

持續前進的擴展

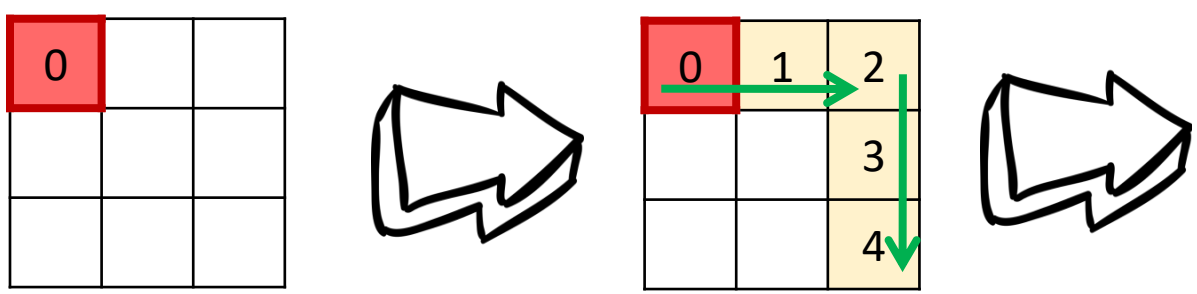
先求有再求好

堆疊與遞迴

深度優先搜尋：只看下一步

- 擴展方式：
  - 只看下一步
  - 只關心當下這一步可以往下走哪一步

由於每一點的執行重點都是在找下一點，因此透過每次都是提供當下位置做為參數的『**遞迴**』演算法是最適合的



實作深度優先搜尋 (1) – 基本流程

```
void dfs(int x, int y, int step)
{
  // 1. 結束條件

  // 2. 還沒結束就繼續下一步
  dfs(nextX, nextY, step+1);

  return;
}
```

重點 1：決定結束條件

定義好遞迴的結束條件，才不會造成函式無限呼叫的問題。

重點 2：找出下一步

相鄰的點都可以做為下一步，但哪些為相鄰，哪一個又是第一個優先拜訪的點，都必須定義清楚，讓每一次的遞迴都可遵循相同規則。

實作深度優先搜尋 (2) – 基本 DFS

```
void dfs(int x, int y)
{
  int i, skip = 0;
  int nextX, nextY;

  if (x < 0 || y < 0 || x >= WIDTH || y >= HEIGHT || visit[x][y] == 1) {
    return;
  }

  visit[x][y] = 1;
  printf("visit: %d, %d\n", x, y);

  for (i=0; i<DIRECTIONS; i++) {
    nextX = x + dir[i][0];
    nextY = y + dir[i][1];

    dfs(nextX, nextY);
  }
}
```

結束條件：在一個 WIDTH * HEIGHT 的平面內探索，當超出範圍時，自然就不需要再繼續拜訪。

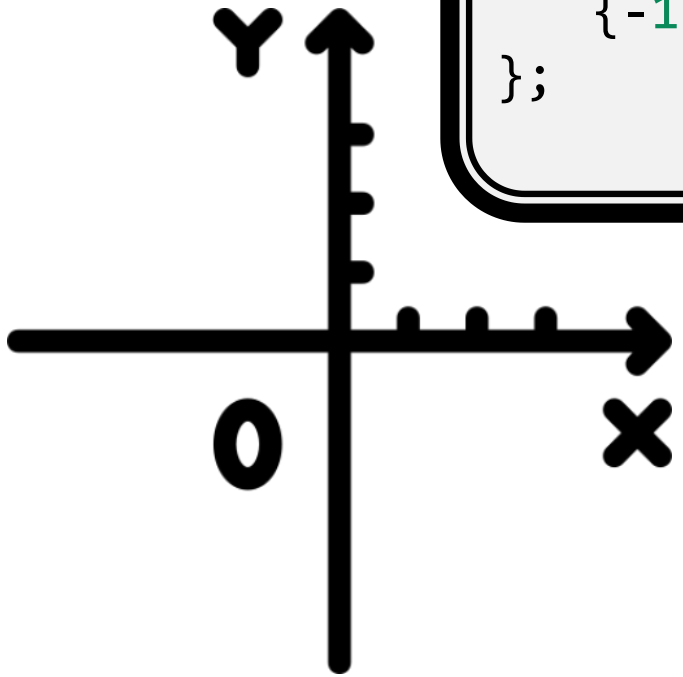
下一步規則：上下左右四個方向是平面上的下一步拜訪對象。

```
void main() {
  initMaze();
  initStack();

  dfs(0, 0);
}
```

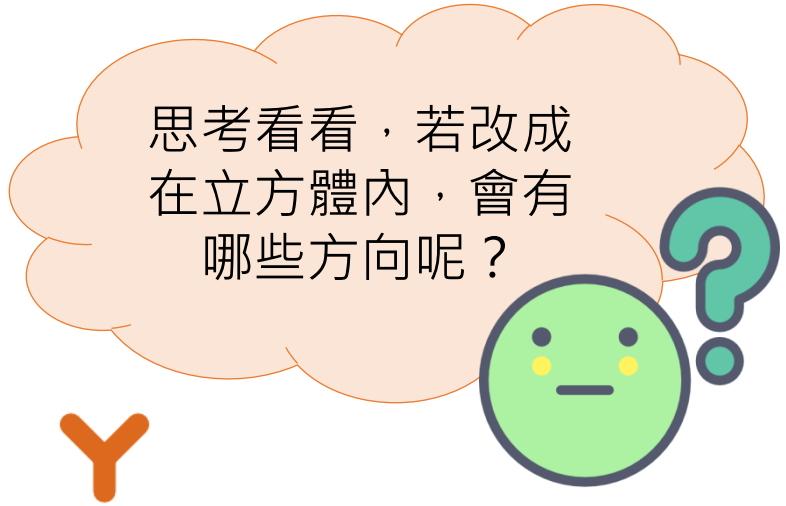
下一步？

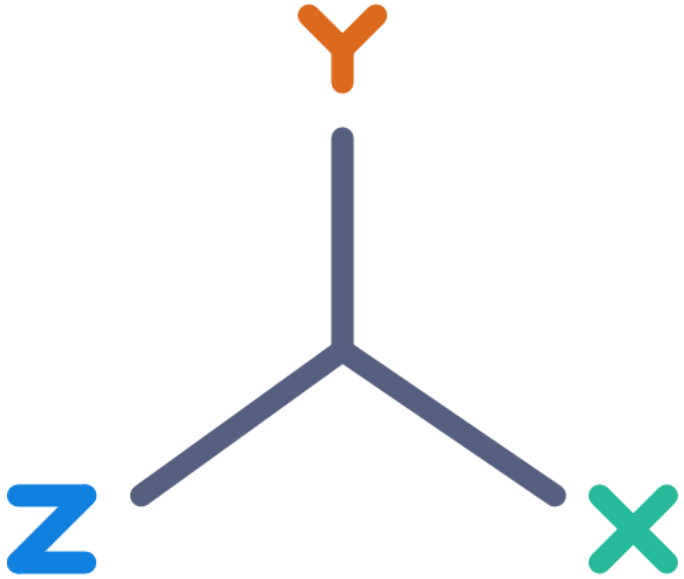
```
dir[4][2] =
{
    {0, 1},
    {1, 0},
    {0, -1},
    {-1, 0}
};
```

平面：四方向



立方？





實作深度優先搜尋 – 搭配堆疊

顯示走過的路徑

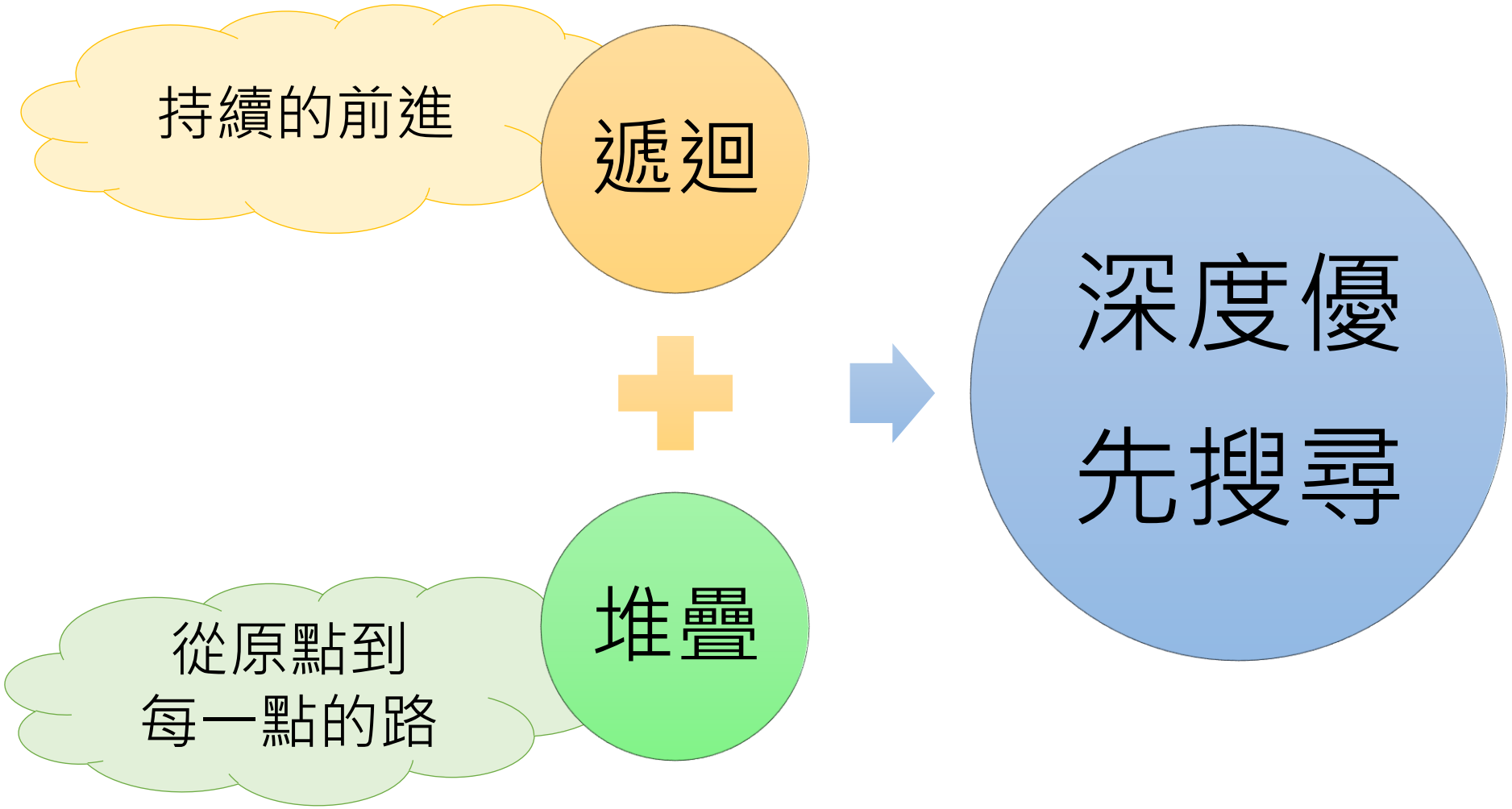
```
void dfs(int x, int y)
{
  int i, skip = 0;
  int nextX, nextY;
  visit[x][y] = 1;
  printf("visit: %d, %d\n", x, y);
  if (x == WIDTH-1 && y == HEIGHT-1) {
    showStack();
    printf("Boundary. x:%d, y: %d\n", x, y);
    return ;
  }
  for (i=0; i<DIRECTIONS; i++) {
    nextX = x + dir[i][0];
    nextY = y + dir[i][1];
    if (nextX < 0 || nextY < 0 || nextX >= WIDTH || nextY >= HEIGHT || visit[nextX][nextY]
== 1) {
      continue;
    }
    stackPush(nextX, nextY);
    dfs(nextX, nextY);
    printf("back to: %d, %d\n", x, y);
    stackPop();
  }
}
```

1. 結束條件

2. 還沒結束就繼續下一步

透過堆疊記路走過的路徑

深度優先搜尋



廣度優先搜尋 BFS
Breadth-First Search

由中心往外的擴展

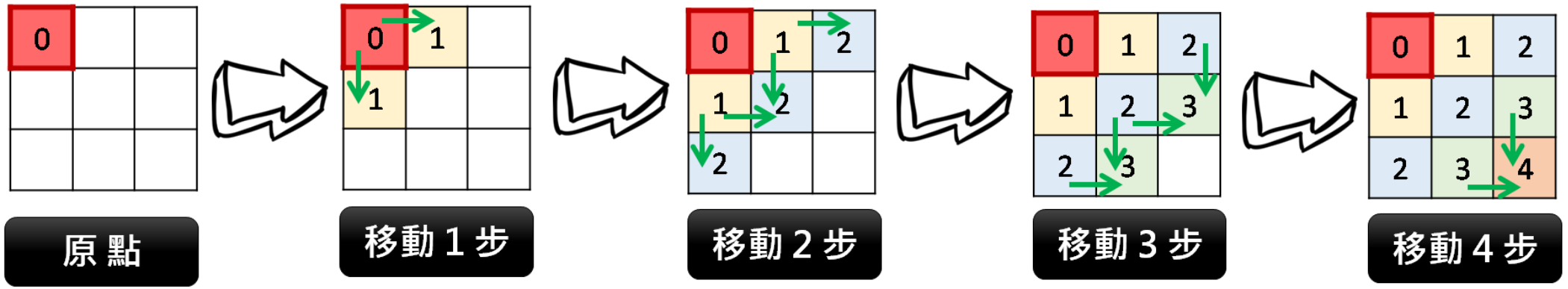
一次到位弄到好

佇列與迴圈

廣度優先搜尋：四面兼顧

- 擴展方式：
  - 相鄰點都要依序拜訪

使用『**迴圈**』逐一檢視每個相鄰點是最適合廣度優先搜尋的！



實作廣度優先搜尋 (1) – 基本流程

```
Void bfs(int x, int y, int step)
{
  bfs(0, 0, 0);

  while(deQueue(&x, &y, &step)) {
     for (i=0; i<DIRECTIONS; i++) {
       enQueue(nextX, nextY, step+1);
     }
  }

  return;
}
```

1

步驟 1：先將起點放到佇列

2

步驟 2：從佇列取出目前要拜訪的點

3

步驟 3：將目前點的相鄰點放道佇列等待拜訪

實作廣度優先搜尋 (2) – 基本 BFS

```
void bfs()
{
  int i, step;
  int x, y;
  int nextX, nextY;
  while(deQueue(&x, &y, &step)) {
    printf("visit: (%d, %d), %d\n", x, y, step);
    for (i=0; i<DIRECTIONS; i++) {
      nextX = x + dir[i][0];
      nextY = y + dir[i][1];
      if (nextX < 0 || nextY < 0 || nextX >= WIDTH || nextY >= HEIGHT)
        continue;
      }
      if (visit[nextX][nextY] >= 0) {
        continue;
      }
      enQueue(nextX, nextY, step+1);
    }
  }
}
```

使用迴圈，將佇列內的點都逐一拜訪，直到佇列內沒有資料時，就代表廣度優先搜尋完成。

```
void main() {
  initMaze();
  initStack();
  enQueue(0, 0, 0);
  bfs(0, 0);
}
```

將所有需要拜訪的點都加到佇列裡

廣度優先搜尋