# Computer Assignment 1
# Informed and Uninformed Search

**In this computer assignment we will implement three search algorithms, BFA, IDS, and A*.**

We will try to model a problem witch our agent must collect all parts and reach to end point. But there are some special parts in the map that will create another agent in upper left corner of the map. We cannot move more than one agent at the same time and all agents must be in upper right corner of the map at the end.

## BFS

BFS (Breath First Search) is an uninformed search algorithms witch will check all possible actions to find the answer.

This algorithm is very memory hungry and relatively slow.

This is the definition of our states: position of each agent, position of parts need to be collected and position of special parts.

We define our initial state like this: one agent in (0,0), list of parts, and list of special parts.

We define our goal state like this: all agents in (n-1,m-1) and empty list of parts.

Each action is move up, down, left, and right and if there is any part our special part we will collect them and take necessary actions like creating another agent or remove from list.

In [1]:

```
import enum
import time


class Cell(enum.Enum):
    empty = 0
    wall = 1
    potion = 2
    double = 3
    start = 4
    end = 5


temple = []
m, n = 0, 0


class State:
    def __init__(self, doubles, potions, doctors, parent):
        self.doubles = doubles
        self.potions = potions
        self.doctors = doctors
        self.parent = parent

    def print_path(self):
        doc = [[] for x in range(int(len(self.doctors)))]
        current = self
        while current is not None:
            for i in range(len(current.doctors)):
                if len(doc[i]) == 0:
                    doc[i].append(current.doctors[i])
                elif current.doctors[i] != doc[i][-1]:
                    doc[i].append(current.doctors[i])
            current = current.parent
        result = ""
        for index in range(len(doc)):
            doc[index].reverse()
            result += f"Doctor {index + 1} : {doc[index]} Length = {len(doc[index]) -
        return result

    def is_done(self):
        done = True
        if len(self.potions) != 0:
            return False
        for doc in self.doctors:
            if doc != (int(n) - 1, int(m) - 1):
                return False
        return done

    def __str__(self):
        result = ""
        result += str(self.doubles)
        result += " "
        result += str(self.potions)
        result += " "
        for i in range(len(self.doctors)):
            result += f"{i} : {self.doctors[i]} "
        return result

```

```python
    def __hash__(self):
        return self.__str__().__hash__()

    def __eq__(self, other):
        return self.__hash__() == other.__hash__()


move = ((1, 0), (0, 1), (-1, 0), (0, -1))


def next_state(frontier, visited):
    ns = []
    for i in range(len(frontier)):
        for j in range(len(frontier[i].doctors)):
            for adj in move:
                pos = frontier[i].doctors[j]
                new_pos = (pos[0] + adj[0], pos[1] + adj[1])
                if pos == (int(n) - 1, int(m) - 1):
                    continue
                if new_pos[0] >= int(n) or new_pos[0] < 0 or new_pos[1] >= int(m) or n
                    continue
                if temple[new_pos[0]][new_pos[1]] == Cell.wall:
                    continue
                new_doubles = list(frontier[i].doubles)
                new_potions = list(frontier[i].potions)
                new_doctors = list(frontier[i].doctors)
                new_doctors[j] = new_pos
                new_s = State(new_doubles, new_potions, new_doctors, frontier[i])
                new_s.doctors[j] = new_pos
                new_s.parent = frontier[i]
                if new_s not in visited:
                    ns.append(new_s)
                    visited.add(new_s)

    return ns


def bfs(initial):
    visited = set()
    queue = [initial]
    while queue:
        for f in queue:
            for doc in f.doctors:
                x, y = doc
                if temple[x][y] == Cell.potion and (x, y) in f.potions:
                    f.potions.remove((x, y))
                elif temple[x][y] == Cell.double and (x, y) in f.doubles:
                    f.doctors.append((int(n) - 1, 0))
                    f.doubles.remove((x, y))
                if f.is_done():
                    return f
        queue = next_state(queue, visited)


result = []
test_time = []
for test in 1, 2, 3:
    test_time.append([])
    for r in 1, 2, 3:
        file = open(f'Tests/test{test}.in')
        n, m = file.readline().split()
```

```python
        c, k = file.readline().split()

        temple = [[Cell.empty for x in range(int(n))] for y in range(int(m))]

        temple[0][0] = Cell.start
        temple[int(n) - 1][int(m) - 1] = Cell.end

        potion = []
        for i in range(int(c)):
            x, y = file.readline().split()
            temple[int(x)][int(y)] = Cell.potion
            potion.append((int(x), int(y)))

        double = []
        for i in range(int(k)):
            x, y = file.readline().split()
            temple[int(x)][int(y)] = Cell.double
            double.append((int(x), int(y)))

        d = file.readline()

        for i in range(int(d)):
            x, y = file.readline().split()
            temple[int(x)][int(y)] = Cell.wall

        initial_state = State(double, potion, [(0, 0)], None)
        begin = time.time()
        res = bfs(initial_state)
        test_time[test - 1].append(time.time() - begin)
        if len(result) == test - 1:
            result.append(res)
    print(f"Average execution time of test{test}.in is : {sum(test_time[test - 1]) / l
    print("Optimal path is :")
    print(result[test - 1].print_path())
```

```
Average execution time of test1.in is : 0.09894331296284993 seconds
Optimal path is :
Doctor 1 : [(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3)] Length =
6
Doctor 2 : [(3, 0), (2, 0), (3, 0), (3, 1), (3, 2), (3, 3)] Length = 5

Average execution time of test2.in is : 22.268924395243328 seconds
Optimal path is :
Doctor 1 : [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (3, 3)] Length =
6
Doctor 2 : [(3, 0), (2, 0), (2, 1), (2, 2), (3, 2), (3, 3)] Length = 5

Average execution time of test3.in is : 20.906030893325806 seconds
Optimal path is :
Doctor 1 : [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (1, 2), (1, 3), (1, 4),
(1, 5), (2, 5), (3, 5), (4, 5), (5, 5)] Length = 12
Doctor 2 : [(5, 0), (5, 1), (5, 2), (5, 3), (4, 3), (5, 3), (5, 4), (5, 5)]
Length = 7
```

# IDS

IDS (Iterative Deepening Search) is ans uninformed search algorithm mixture of BFS and DFS (Depth First Search).

This algorithms runs DFS on increasing depth limit until find the goal state.

Even though at each iteration it runs a DFS search, it's optimal like BFS and can usually find the goal state without exploring all the nodes, yet it doesn't require the queue and uses much less memory than BFS.

We define our initial state, goal state, and actions as same as BFS algorithm.

## A*