无信息搜索

PPT制作及讲解：罗玲

目录

- 理论课回顾
  - 形式化一个搜索问题
    - 问题的定义
    - 问题的解的定义
    - 为什么要搜索算法
  - 问题求解算法的性能
  - 无信息搜索策略
    - 宽度优先搜索(BFS)
    - 一致代价搜索 (UCS)
    - 深度优先搜索(DFS)
    - 深度受限搜索
    - 迭代加深搜索
    - 双向搜索
- 思考题&实验任务

形式化一个搜索问题

1.1 问题的定义 [1]

用5个组件形式化定义一个search problem：

1) initial state 初始状态
2) Actions 行动
3) transition model 转移模型
4) goal test 目标测试
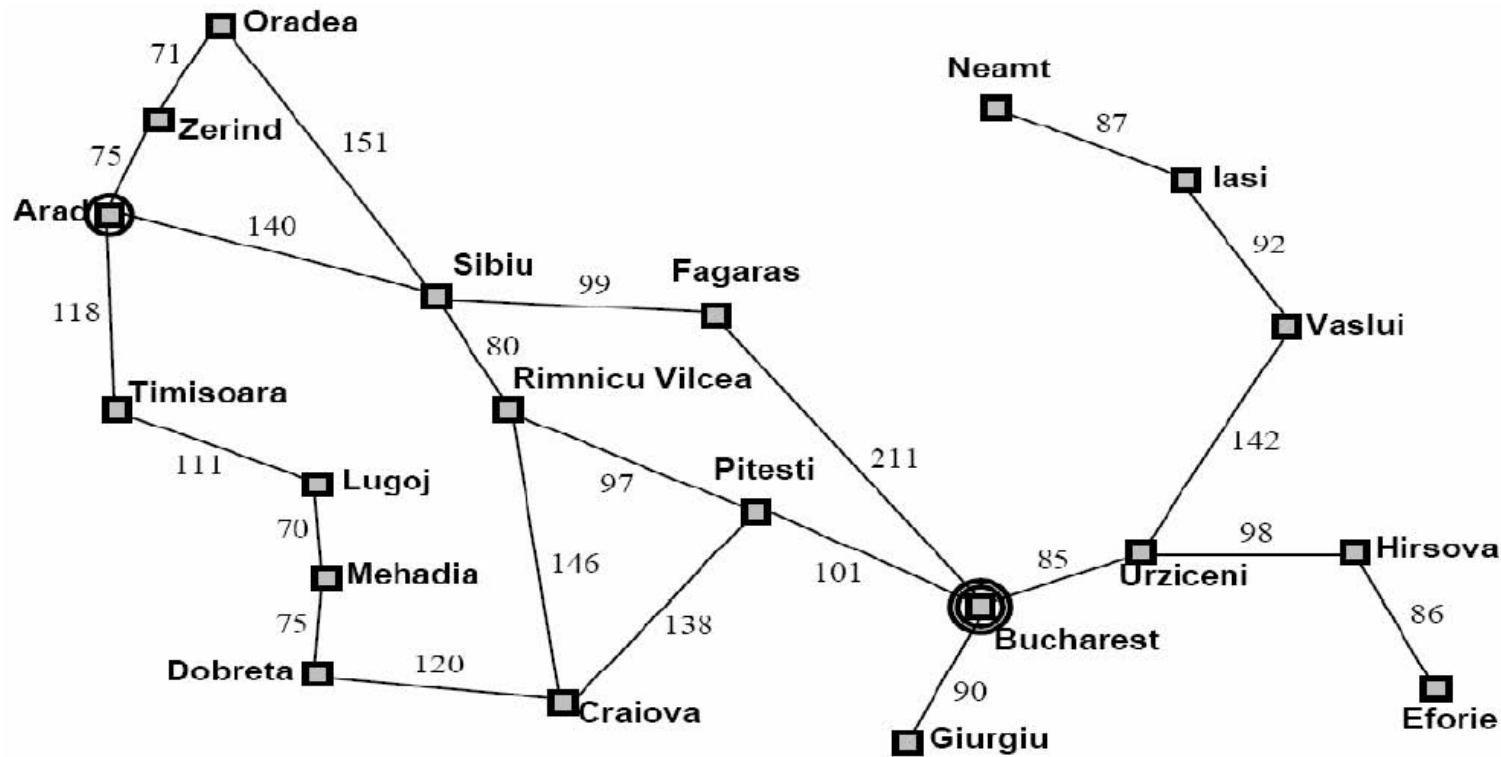5) path cost 路径耗散

1.2 问题的解（solution）的定义

A solution to a problem is an action sequence that leads from the initial state to a goal state.

即：一个问题的解是：一个**行动**序列

举例

Currently in Arad, need to get to Bucharest

- ➤ **States:** the various cities you could be located in.
- ➤ **Actions:** drive between neighboring cities.
- ➤ **Initial state:** in Arad
- ➤ **Goal:** in Bucharest
- ➤ **Solution:** the route, the sequence of cities to travel through to get to Bucharest.

通过搜索求解

function TREE-SEARCH(*problem*) returns a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

function GRAPH-SEARCH(*problem*) returns a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
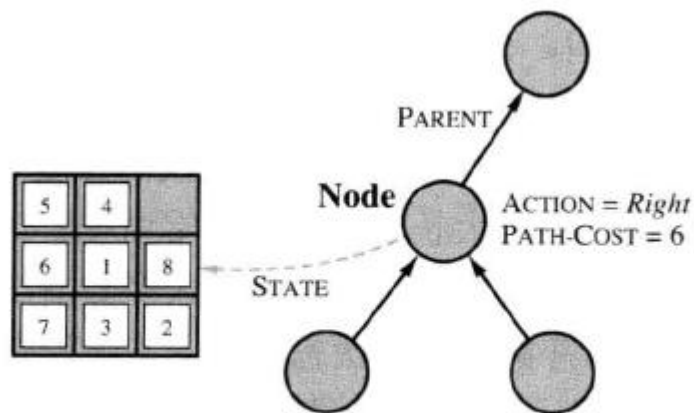   *only if not in the frontier or explored set*

图 3.7　一般的树搜索和图搜索算法的非形式化描述。用粗斜体标出的图搜索算法的部分内容是指需要额外处理重复状态

搜索的数据结构

对树中每个结点n，一般定义如下数据结构

- *n*.STATE：对应状态空间中的状态；
- *n*.PARENT：搜索树中产生该结点的结点（即父结点）；
- *n*.ACTION：父结点生成该结点时所采取的行动；
- *n*.PATH-COST：代价，一般用 $g(n)$ 表示，指从初始状态到达该结点的路径消耗；

结点是数据结构，由搜索树构造。每个结点都有一个父结点、一个状态和其他域。箭头由子结点指向父结点

求解算法的性能

四个方面：

- 完备性（当问题有解时，这个算法能否保证找到解）

- 最优性（搜索策略能否找到最优解）

- 时间复杂度（找到解要花多长时间）也叫搜索代价
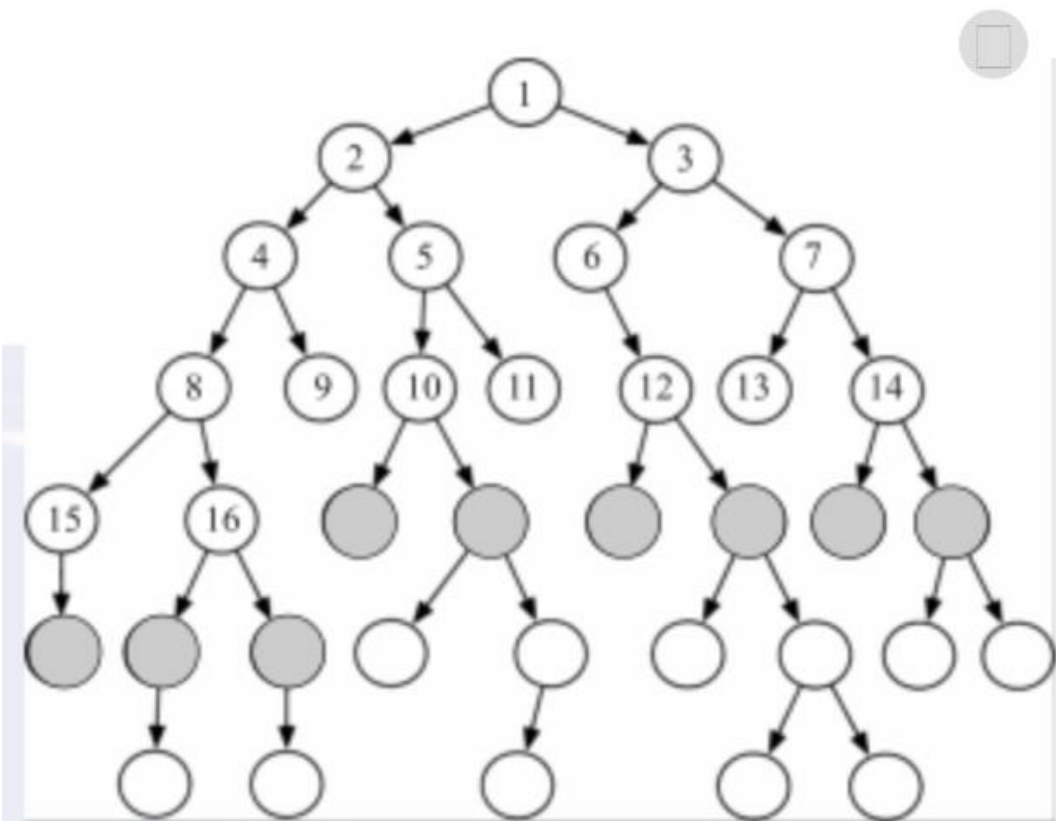
- 空间复杂度（在执行搜索的过程中需要多少内存）
  总代价=搜索代价+内存使用（空间复杂度）

无信息搜索策略

- 宽度优先搜索
- 一致代价搜索
- 深度优先搜索
- 深度受限搜索
- 迭代加深搜索
- 双向搜索

宽度优先搜索(BFS)

宽度优先搜索中节点的扩展顺序

- 完备性；
- 非最优；
- 时间复杂度O(b^d);
- 空间复杂度O(b^d);

- 节点扩展顺序与目标节点的位置无关；
- 边界（frontier）用一个先进先出（FIFO）队列实现;

一致代价搜索Uniform-cost search (UCS)

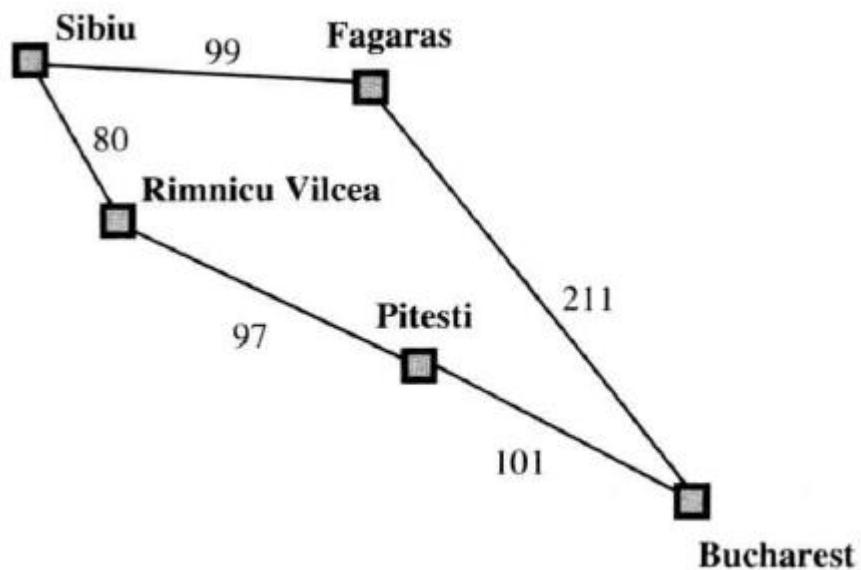
图 3.15　罗马尼亚问题的部分状态空间，用于描述一致代价搜索

Search Strategy：扩展最低代价的未扩展节点。

Implementation：队列，按路径代价排序，最低优先

Complexity：$O(b^{1+[C^*/e]})$，

一致代价搜索Uniform-cost search (UCS)

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST, with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                frontier ← INSERT(child, frontier)
            else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child
```
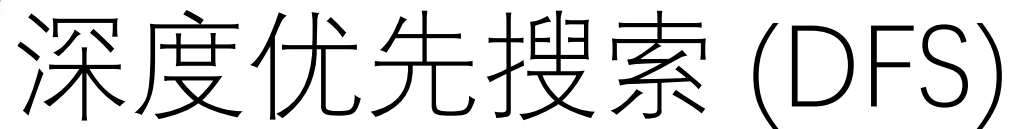
深度优先搜索 (DFS)
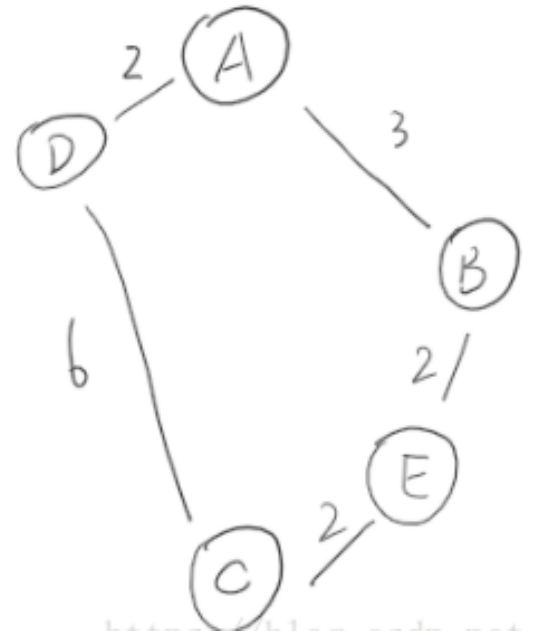
图 3-5 深度优先搜索中节点的扩展顺序

深度受限搜索 (Depth-limited Search)

```
1  function Depth-Limited-Search(problem, limit) returns a solution, or failure/cutoff
2      if problem.Goal-Test(node.State) then return Solution(node)
3      if limit = 0 then return cutoff       //no solution
4      cutoff_occurred? ← false
5      for each action in problem.Action(node.State) do
6          child ← Child-Node(problem, node, action)
7          result ← Recusive-DLS(child, problem, limit - 1)
8          if result = cutoff then cutoff_occurred ? ← true
9          else if result ≠ failure then return result
10     if cutoff_occurred ? then return cutoff    //no solution
11     else return failure
```

迭代加深搜索（Iterative Deepening Search）
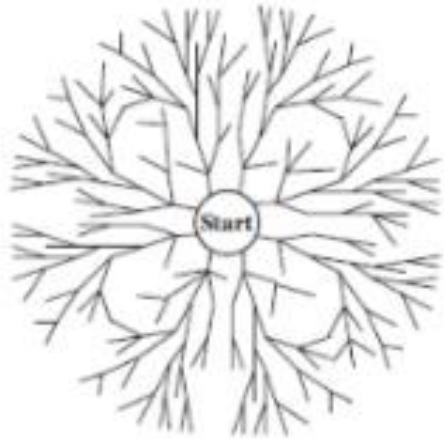
```
1   function Iterative-Deepening-Search(problem, limit) returns a solution, or failure
2       for depth = 0 to ∞ do
3           result ← Depth-Limited-Search(problem,  depth)
4           if result ≠ cutoff then return result
```

双向搜索 （Bidirectional search）

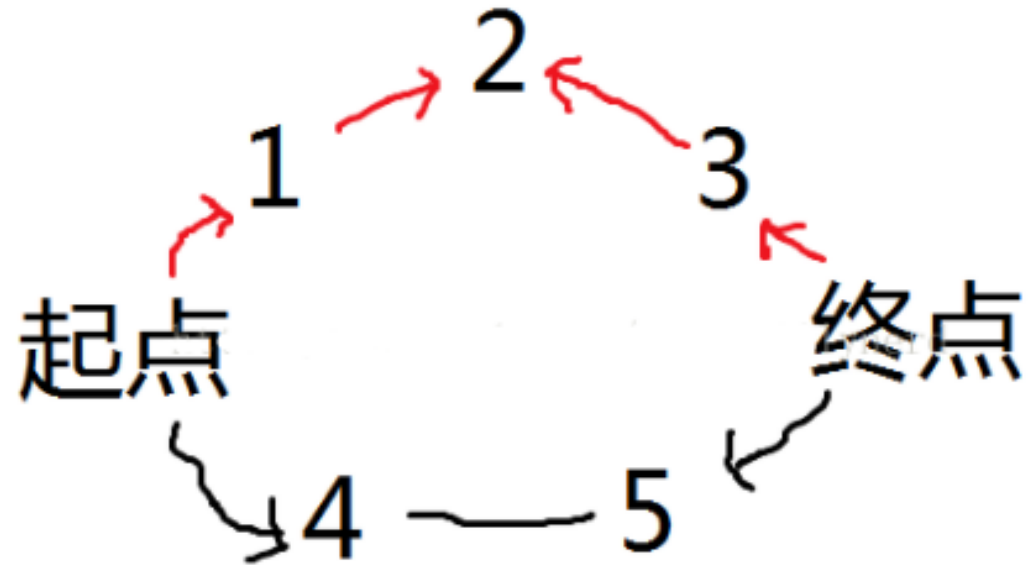它同时进行两个搜索：一个是从初始状态向前搜索，二另一个则从目标向后搜索。当两者在中间相遇时停止。

forward tree

backward tree

双向搜索 （Bidirectional search）

无信息树搜索策略评价

Evaluation of Uninformed Tree-search Strategies

无信息树搜索策略评价

Criterion	Breadth First	Uniform Cost	Depth First	Depth Limited	Iterative Deepening	Bidirectional
Complete	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Where

- b -- maximum branching factor of the tree
- d -- depth of the shallowest solution
- m -- maximum depth of the tree
- l -- the depth limit

- a -- complete if b is finite
- b -- complete if step costs ϵ for positive
- c -- optimal if step costs are all identical
- d -- if both directions use breadth-first search
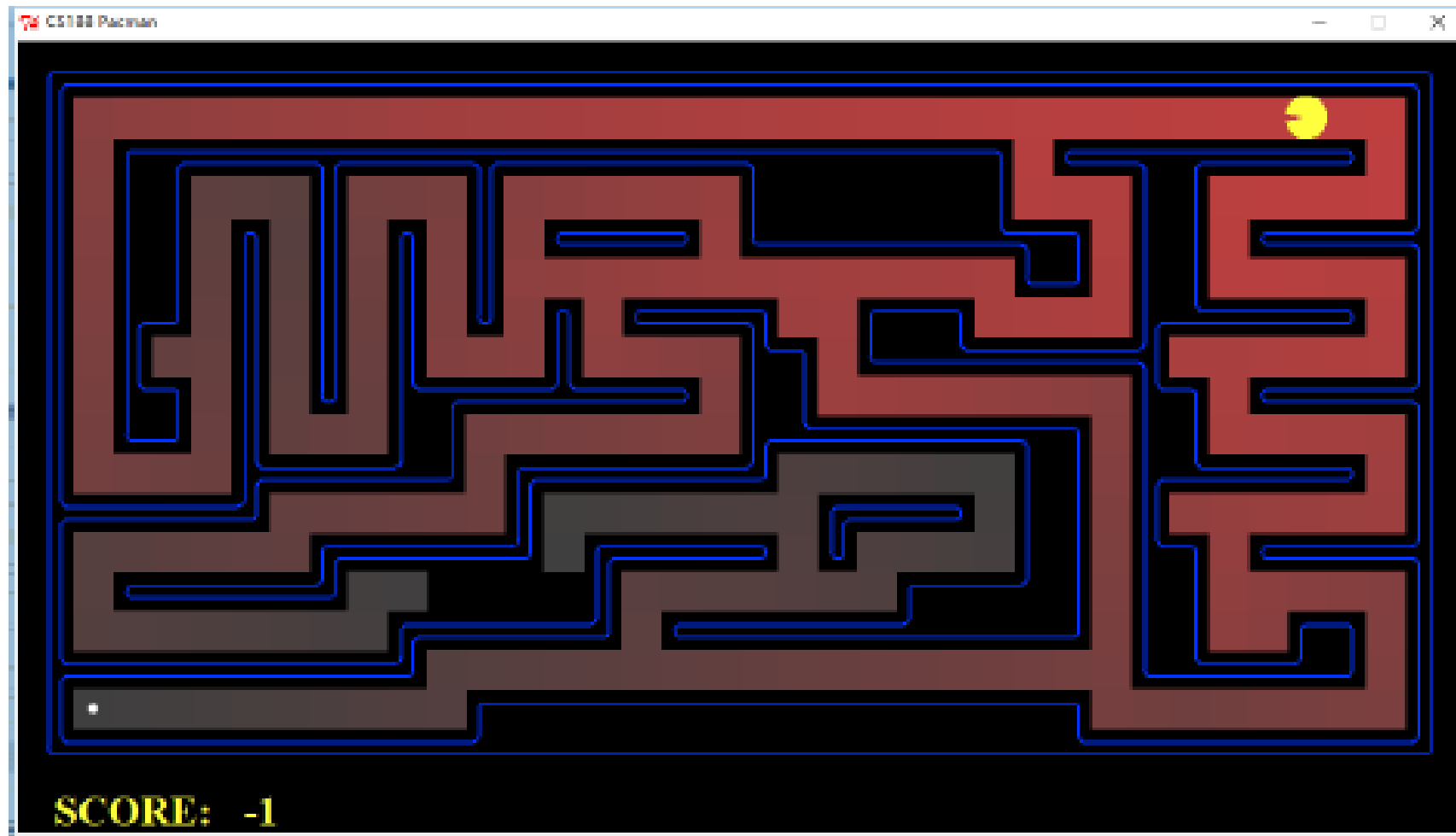
思考题

- 这些策略的优缺点是什么？它们分别适用于怎样的场景？

实验要求

- 从给定类型一和类型二中分别选择一个策略解决迷宫问题。
  - 类型一：BFS、DFS
  - 类型二：一致代价、迭代加深、双向搜索
- 代码要求使用python或者C++
- 报告要求：报告中①需要有对实现的策略（两个）的原理解释，②需要两种策略的实验效果（四个方面的算法性能）的对比和分析，③以及自己对不同策略优缺点，适用场景的理解和认识。
- 提交文件
  - 实验报告：16******_wangxiaoming.pdf。
  - 代码：16******_wangxiaoming.zip。如果代码分成多个文件，最好写份readme。
- DDL: 2018-11-06 23:00:00

```
11111111111111111111111111111111111
1000000000000000000000000000000000S1
10111111111111111111111101111111101
10110001001000001111111000110000001
10110101010101111011111111011011111
10110101010100000000000011011000001
10110101010101011110011100011111101
10100101010001000110111111110000001
10110101011111110110000000011011111
10110100110000000111111110110000001
10000111110111111100000011011111101
11111100000010000000111101101000001
10000001111110111110100001101101111
10111111000000100000011110110000001
10000000011111101111111111011001101
11111111100000000000000000011111101
1E0000000011111111111111111000000001
11111111111111111111111111111111111111
```