Simple Search Algorithms

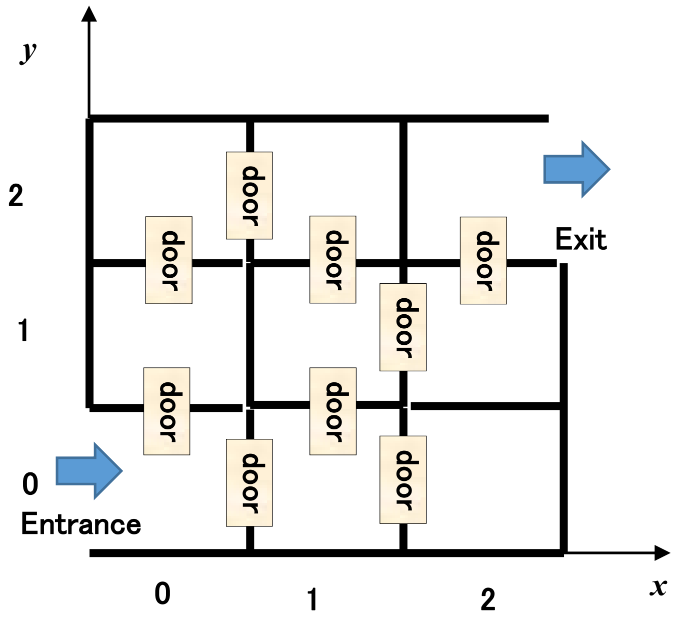
Topics of this lecture

- Random search
- Search with closed list
- Search with open list
- Depth-first and breadth-first search again
- Uniform-cost search

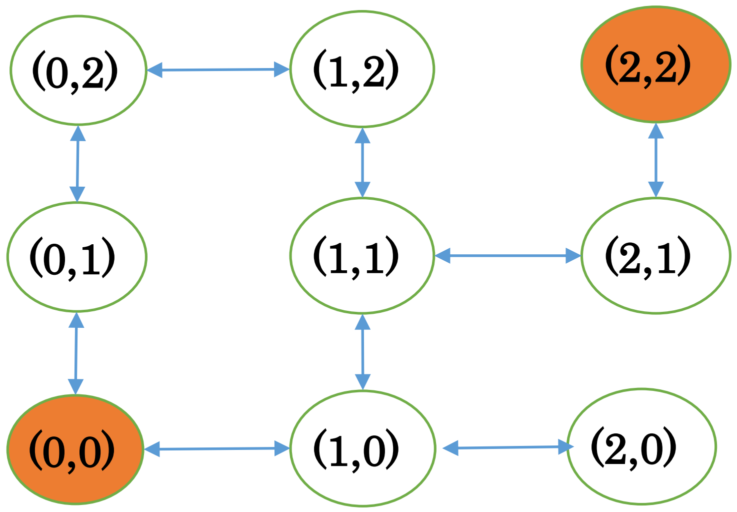
State space representation of AI problems

The maze problem

- Initial state: (0,0)
- Target state: (2,2)
- Available operations:
  - Move forward
  - Move backward
  - Move left
  - Move right
- Depends on the current state, the same operation may have different results.
- Also, an operation may not be executed for some states.



State space → search graph



- To find the solution, we can just traverse the graph, starting from (0,0), and stop when we visit (2,2).
- **The result is a path from the initial node to the target node.**
- The result can be different, depends on the order of graph traversal.

Basic considerations for solving a problem

- Many problems can be formulated using the state space representation.
- Each state corresponds to a candidate solution.
- Problem solving is to find the best solution via state transition, which is equivalent to finding a desired node via traversing a search graph.
- When the problem space is not discrete (i.e. continuous), we need more sophisticated search algorithms.

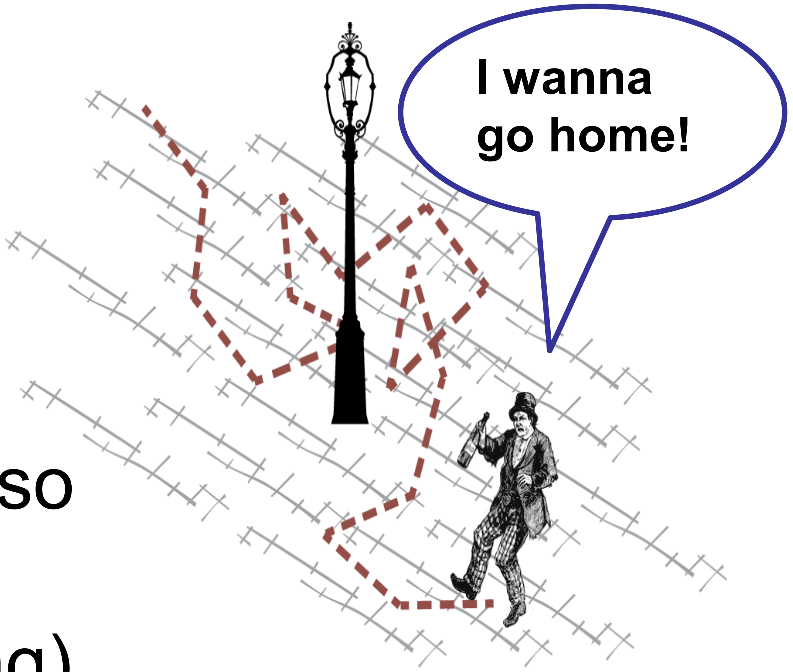
Random search

- Step 1: Current node x=initial node;
- Step 2: If x=***target node***, stop with success;
- Step 3: Expand x, and get a set S of child nodes;
- Step 4: Select a node x' from S ***at random***;
- Step 5: x=x', and return to Step 2.

ノードの展開：指定ノードの子ノードを求めること

Random search is not good

- At each step, the next node is determined at random.
- We cannot guarantee to reach the target node.
- Even if we can, the path so obtained can be very redundant (extremely long).



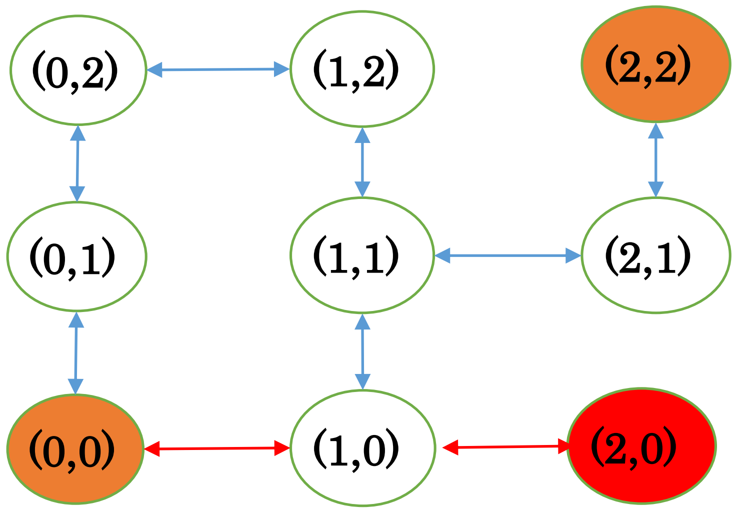
Search with a closed list

Do not visit the same node more than once!

- Step 1: Current node x=initial node;
- Step 2: If x=target node, stop with success;
- Step 3: Expand x, and get a set S of child nodes. If S is empty, stop with failure. Add x to the closed list.
- Step 4: Select from S a new node x' ***that is not in the closed list***.
- Step 5: x=x', and return to Step 2.

Closed list is not enough!

- Using a closed list, we can guarantee termination of search in finite steps.
- However, we may never reach the target node!



Search with open list

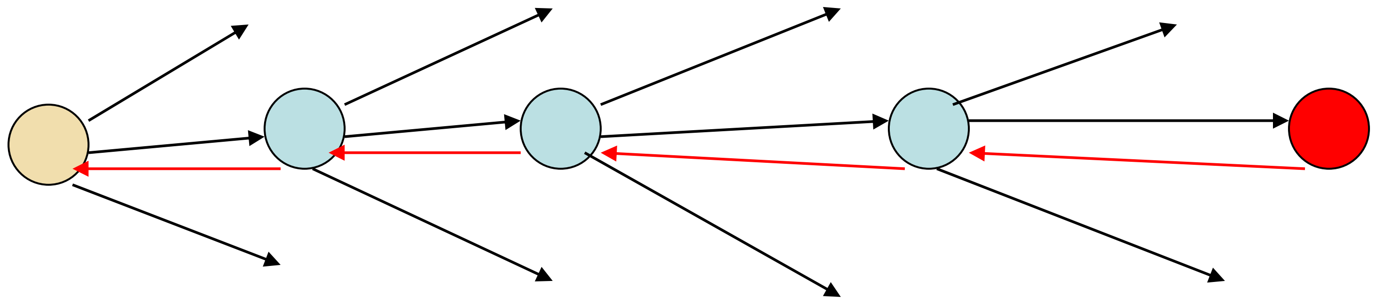
Keep all “un-visited” nodes in another list!

- Step 1: Add the initial node to the open list.
- Step 2: Take a node x from the open list from the top. If the open list is empty, stop with failure; on the other hand, if x is the target node, stop with success.
- Step 3: Expand x to obtain a set S of child nodes, and put x into the closed list.
- Step 4: For each node x’ in S, ***if it is not in the closed list, add it to the open list along with the edge (x, x’).***
- Step 5: Return to Step 2.

See Algorithm 1 in the textbook

Property of Algorithm 1

- It is known that search based on the open list is **complete** in the sense that we can always find the solution in a finite number of steps if the search graph is finite.
- The edge (x, x’) is kept in the search process for “restoring” the search path via “back-tracking”.



Depth-first search and breadth-first search

- Algorithm I is a depth-first search ***if we implement the open list using a stack***.
- Algorithm I becomes the breadth-first search ***if the open list is implemented using a queue***.
- It is known the breadth-first search is better because even for an infinite search graph, we can get the solution in finite steps, if the solution exists.

Example 2.4

(p. 20 in the textbook)

Step	Open List	Closed List
0	(0,0)	--
1	(1,0) (0,1)	(0,0)
2	(2,0) (1,1) (0,1)	(0,0) (1,0)
3	(1,1) (0,1)	(0,0) (1,0) (2,0)
4	(2,1) (1,2) (0,1)	(0,0) (1,0) (2,0) (1,1)
5	(2,2) (1,2) (0,1)	(0,0) (1,0) (2,0) (1,1) (2,1)
6	(2,2)=Target node	(0,0) (1,0) (2,0) (1,1) (2,1)

Uniform-cost search: The Dijkstra's algorithm

- Usually, the solution is not unique. It is expected to find the BEST one.
- For example, if we want to travel around the world, we may try to find the fastest route; the most economic route; or the route in which we can visit more friends.
- The uniform-cost search or Dijkstra’s algorithm is a method for solving this problem.

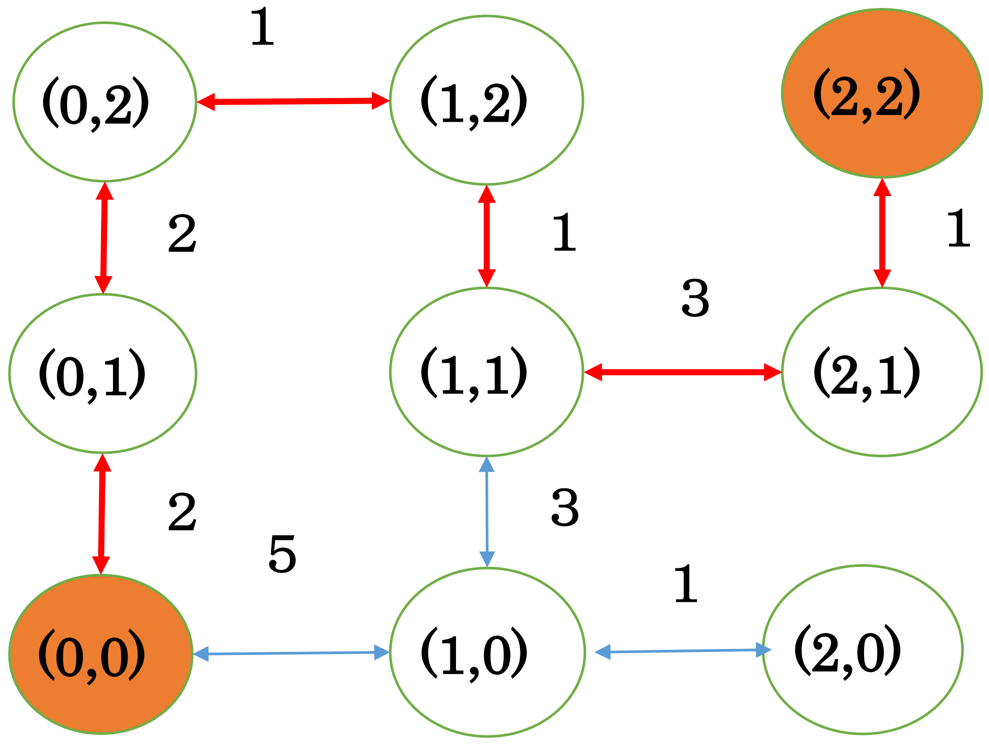
Uniform-cost search

- Step 1: Add the initial node x0 and its cost C(x0)=0 to the open list.
- Step 2: Get a node x from the top of the open list. If the open list is empty, stop with failure. If x is the target node, stop with success.
- Step 3: Expand x to get a set S of child nodes, and move x to the closed list.
- Step 4: For each x' in S but not in the closed list, find its ***accumulated cost*** C(x')=C(x)+d(x,x'); and add x', C(x'), and (x, x') to the open list. If x' is already in the open list, update its cost and link if the new cost is smaller.
- Step 5: Sort the open list based on the node costs, and return to Step 2.

Uniform-cost search

- During uniform-cost search, we can always find the best path from the initial node to the current node. That is, when search stops with success, the solution must be the best one.
- In the algorithm, c(x) is the cost of the node x accumulated from the initial node; and d(x,x’) is the cost for state transition (e.g. the distance between to adjacent cities).
- If we set d(x,x’)=1 for all edges, uniform-cost search is equivalent to the breadth-first search.

Example 2.5: Search a path with the minimum cost



Example 2.5: Search the path with the minimum cost

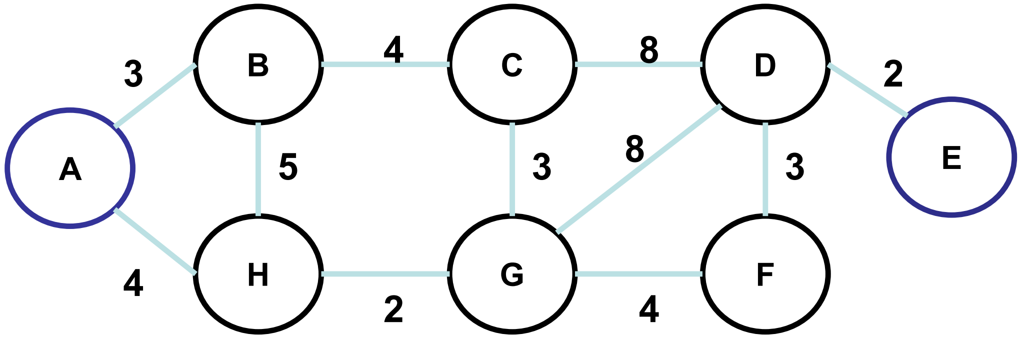
Step	Open List	Closed List
0	{(0,0),0}	--
1	{(0,1),2}, {(1,0),5}	(0,0)
2	{(0,2),4}, {(1,0),5}	(0,0) (0,1)
3	{(1,0),5}, {(1,2),5}	(0,0) (0,1) (0,2)
4	{(1,2),5},{(2,0),6}, {(1,1),8}	(0,0) (0,1) (0,2) (1,0)
5	{(2,0),6}, **{(1,1),6}**	(0,0) (0,1) (0,2) (1,0) (1,2)
6	{(1,1),6}	(0,0) (0,1) (0,2) (1,0) (1,2) (2,0)
7	{(2,1),9}	(0,0) (0,1) (0,2) (1,0) (1,2) (2,0) (1,1)
8	{(2,2),10}	(0,0) (0,1) (0,2) (1,0) (1,2) (2,0) (1,1) (2,1)
9	(2,2)=Target node	(0,0) (0,1) (0,2) (1,0) (1,2) (2,0) (1,1) (2,1)

Homework for lecture 3 (1)

- For the maze problem shown in Fig. 2.2 (p. 14 in the textbook), find a path from (0,0) to (2,2) using breadth-first search.
- Summarize the results using a table similar to Table 2.3 (p. 20 in the textbook).
- Submit the result (in hardcopy) to the TA within the exercise class.

Homework for lecture 3 (2)

- Down-load the skeleton, and make a program for depth-first search and breadth-first search.
- The graph is given below. The initial node is A, and the target node is E. The costs of the edges are also given.



- Differences between this program and the one of last class:
  - Input the node using alphabets;
  - Stop when the target node is found; and
  - Output the costs during search.

Quizzes

- What is the main problem of “random search” ?
- What is the main problem of “search with closed list”?
- How to implement the open list if we want to do breadth-first search?
- What is the relation of uniform-cost search and breadth-first search?