执行测验: Homework 1

测试信息

描述 我们将在本次作业中允许多次尝试，不限制提交次数。请注意：

- 作业将使用最后一次尝试的成绩作为最终成绩；
- 未提交的尝试将被记为0分；
- 当开始新的尝试时，所填入的答案将被完全清除。

因此，当决定提交作业时，请在其他设备上妥善保存已经完成的答案；否则，请保存答案但不要提交。在截止日期之前，请确保作业的最后一次尝试已经提交。在截止日期之后，如果发现作业成绩有任何问题，可以随时联系助教处理。

FAQ

1. 作业有grace day吗?

BB作业没有grace day，Autolab编程作业有5个grace day。

2. 我忘记提交作业了，可以请助教帮忙提交吗?

在同时满足以下条件时，你可以联系助教在ddl之后为你提交作业：
a. 你的当前作业没有成绩，没有提交记录；
b. 你的作业完成记录显示你的所有操作在ddl之前完成。

注意，BB会记录助教的所有操作，这些操作也都将需要归档。

说明 注意：本作业不会自动提交。请在完成作业检查无误后，单击右下角“保存并提交”按钮提交作业。逾期未提交的作业不会被保存或计分。

多次尝试 此测试允许进行多次尝试。

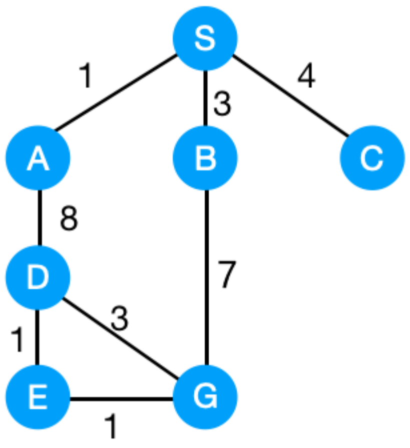
强制完成 本测试可保存并可稍后继续。

问题完成状态：

问题 1 10 分 已保存

Graph Search Part 1

Consider a graph search from S to G on the graph below. Edges are labeled with action costs. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A.)

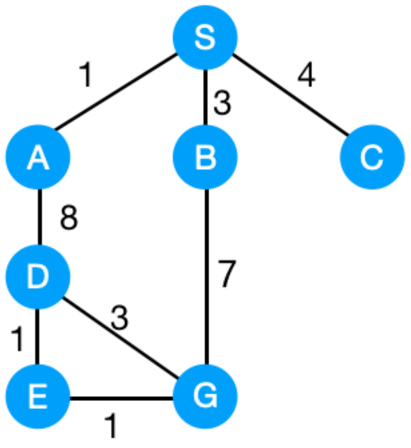


Which search strategy will return the path S-B-G?

- [✓] Breadth-First Search
- [ ] Depth-First-Search
- [✓] Uniform Cost Search
- [ ] None

问题 2 10 分 已保存

Graph Search Part 2



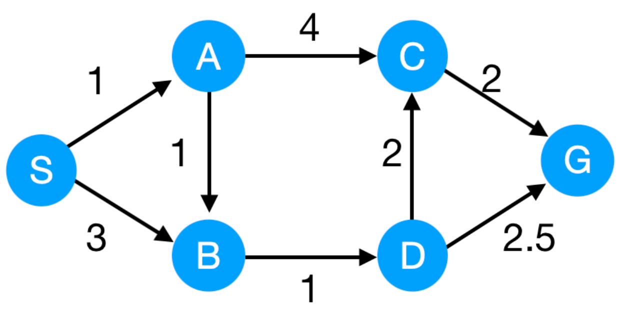
Continue with Graph Search Part 2, which search strategy will return the path S-A-D-E-G?

- [ ] Breadth-First Search
- [✓] Depth-First Search
- [ ] Uniform Cost Search
- [ ] None

问题 3 10 分 已保存

Graph Search: Uniform Cost Search

Consider uniform cost graph search from S to G on the graph below. Arcs are labeled with action costs. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A.



Rank the nodes according to the visiting sequence during uniform cost search.
Note: nodes visited during UCS might not be included in the path returned by UCS.

2.
A

5.
C

4.
D

6.
G

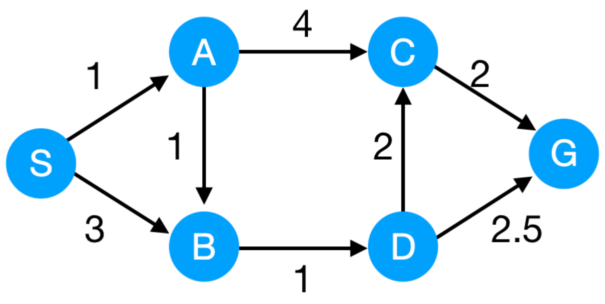
1.
S

3.
B

问题 4 10 分 已保存

Graph Search: A*

Given the heuristic for each node. Consider A* search from S to G on the graph below. Arcs are labeled with action costs. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A.



	S	A	B	C	D	G
heuristic	5	3	2	1	1.5	0

Choose the path returned by A* search.

○ S-A-C-G

○ S-B-D-G

○ S-A-B-D-C-G

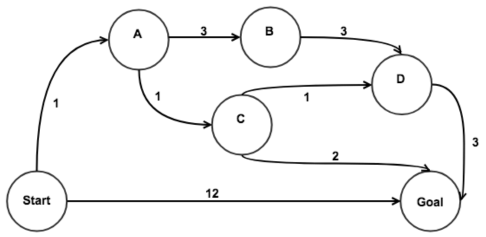
[✓ S-A-B-D-G]

○ S-B-D-C-G

问题 5 10 分 已保存

Graph Search: Uniform Cost Search

Consider uniform cost graph search on the graph below. Arcs are labeled with action costs. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A.



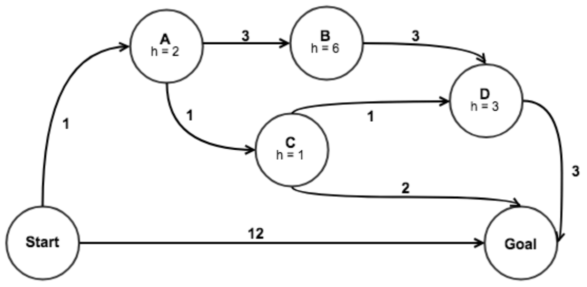
In what order are states expanded by uniform cost graph search? You may find it helpful to execute the search on scratch paper.

- ○ Start, A, B, C, D, Goal
- ○ Start, A, B, D, C, Goal
- ○ Start, A, C, B, D, Goal
- ● Start, A, C, D, B, Goal
- ○ Start, A, D, C, B, Goal
- ○ Start, A, C, Goal

问题 6 10 分 已保存

A* Search

Consider A* graph search on the graph below. Arcs are labeled with action costs and states are labeled with heuristic values. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A.



In what order are states expanded by A* graph search? You may find it helpful to execute the search on scratch paper.

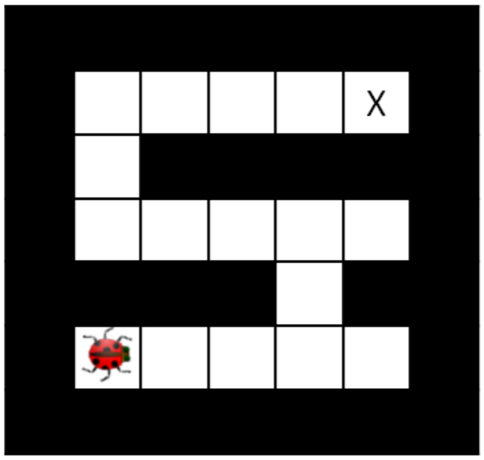
- ○ Start, A, B, C, D, Goal
- ○ Start, A, B, D, C, Goal
- ○ Start, A, C, B, D, Goal
- ○ Start, A, C, D, B, Goal
- ○ Start, A, D, C, B, Goal
- ● Start, A, C, Goal

问题 7

10 分 已保存

HIVE MINDS Part 1

You control one or more insects in a rectangular maze-like environment with dimensions $M*N$, as shown in the figure below.



At each time step, an insect can move into an adjacent square if that square is currently free, or the insect may stay in its current location. Squares may be blocked by walls, but the map is known. **Optimality is always in terms of time steps; all actions have cost 1 regardless of the number of insects moving or where they move**.

For each question, you should answer for a general instance of the problem, not simply for the example maps shown.

You control a single insect as shown in the maze below, which must reach a designated target location X, also known as the hive. There are no other insects moving around.

Which of the following is a *minimal* correct state space representation?

- ○ An integer d encoding the Manhattan distance to the hive.
- [✓ A tuple (x, y) encoding the x and y coordinates of the insect.]
- ○ A tuple (x,y,d) encoding the insect's x and y coordinates as well as the Manhattan distance to the hive.
- ○ This cannot be represented as a search problem.

问题 8

10 分 已保存

HIVE MINDS Part 2

Only in this question, suppose there is a pellet somewhere in the maze (not on the walls). Before reaching the designated location X, the insect must eat this pellet.

By adding this assumption, please give a tight upper bound on the size of the state space.

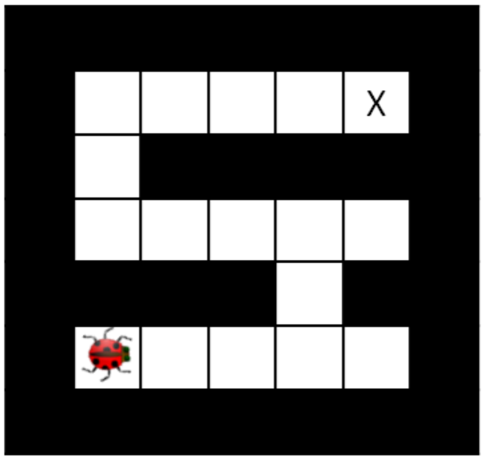
- ○ $2^{M \times N + 1}$
- [✓ $M \times N \times 2$]
- ○ $M \times N$
- ○ $(M + N) \times 2$

问题 9

10 分 已保存

HIVE MINDS Part 3

You control a single insect as shown in the maze below, which must reach a designated target location X, also known as the hive. There are no other insects moving around.



Which of the following heuristics are admissible (if any)?

☑ Manhattan distance from the insect’s location to the hive.

☑ Euclidean distance from the insect’s location to the hive.

☐ Number of steps taken by the insect.

问题 10

10 分 已保存

Memory Efficient Graph Search

Recall from lecture the general algorithm for GRAPH-SEARCH reproduced below.

```
function GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

Using GRAPH-SEARCH, when a node is expanded it is added to the closed set. This means that even if a node is added to the fringe multiple times it will not be expanded more than once. Consider an alternate version of GRAPH-SEARCH, MEMORY-EFFICIENT-GRAPH-SEARCH, which saves memory by (a) not adding node n to the fringe if STATE[n] is in the closed set, and (b) checking if there is already a node in the fringe with last state equal to STATE[n]. If so, rather than simply inserting, it checks whether the old node or the new node has the cheaper path and then accordingly leaves the fringe unchanged or replaces the old node by the new node.

By doing this the fringe needs less memory, however insertion becomes more computationally expensive.

More concretely, MEMORY-EFFICIENT-GRAPH-SEARCH is shown below with the changes highlighted.

```
function MEMORY-EFFICIENT-GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← SPECIAL-INSERT(child-node, fringe, closed)
            end
    end


function SPECIAL-INSERT(node, fringe, closed) return fringe
    if STATE[node] not in closed set then
        if STATE[node] is not in STATE[fringe] then
            fringe ← INSERT(node, fringe)
        else if STATE[node] has lower cost than cost of node in fringe reaching STATE[node] then
            fringe ← REPLACE(node, fringe)
```

Now, we've produced a more memory efficient graph search algorithm. However, in doing so, we might have affected some properties of the algorithm. Assume you run MEMORY-EFFICIENT-GRAPH-SEARCH with the A* node expansion strategy and a consistent heuristic, select all statements that are true.

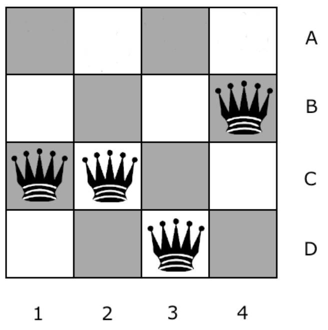
- ✓ The EXPAND function can be called at most once for each state.
- ✓ The algorithm is complete.
- ✓ The algorithm will return an optimal solution.

问题 11 10 分 已保存

4-Queens Part 1

The min-conflicts algorithm attempts to solve CSPs iteratively. It starts by assigning some value to each of the variables, ignoring the constraints when doing so. Then, while at least one constraint is violated, it repeats the following: (1) randomly choose a variable that is currenly violating a constraint, (2) assign to it the value in its domain such that after the assignment the total number of constraints violated is minimized (among all possible selections of values in its domain).

In this question, you are asked to execute the min-conflicts algorithm on a simple problem: the 4-queens problem in the figure shown below. Each queen is dedicated to its own column (i.e. we have variables Q_ 1, Q_2, Q_3 and Q_4 and the domain for each one of them is {A, B, C, D}). In the configuration shown below, we have Q_1=C, Q_2=C, Q_3=D, Q_4=B. Two queens are in conflict if they share the same row, diagonal, or column (though in this setting, they can never share the same column).



You will execute min-conflicts for this problem three times, starting with the state shown in the figure above. When selecting a variable to reassign, min-conflicts chooses a conflicted variable at random. For this problem, assume that your random number generator always chooses the leftmost conflicted queen. When moving a queen, move it to the square in its column that leads to the fewest conflicts with other queens. If there are ties, choose the topmost square among them.

We recommend you work out the solutions to the following questions on a sheet of scratch paper, and then enter your results below.

(a) Starting with the queens in the configuration shown in the above figure, which queen will be moved, and where will it be moved to?

(a-1) Queen

● 1
○ 2
○ 3
○ 4

问题 12 10 分 已保存

4-Queens Part 2

(a) Starting with the queens in the configuration shown in the above figure, which queen will be moved, and where will it be moved to?

(a-2) Position

● A
○ B
○ C
○ D

问题 13 10 分 已保存

4-Queens Part 3

(b) Continuing off of Part 1-2, which queen will be moved, and where will it be moved to?

(b-1) Queen

○ 1
● 2
○ 3
○ 4

问题 14 10 分 已保存

4-Queens Part 4

(b) Continuing off of Part 1-2, which queen will be moved, and where will it be moved to?

(b-2) Position

● A
○ B
○ C
○ D

问题 15 10 分 已保存

4-Queens Part 5

(c) Continuing off of Part 3-4, which queen will be moved, and where will it be moved to?

(c-1) Queen

● 1
○ 2
○ 3

○ 4

问题 16 10 分 已保存

4-Queens Part 6

(c) Continuing off of Part 3-4, which queen will be moved, and where will it be moved to?

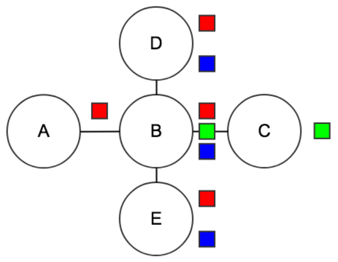
(c-2) Position

○ A
○ B
[✓ C]
○ D

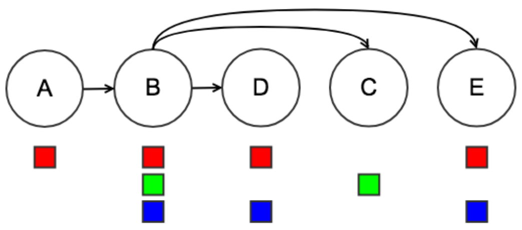
问题 17 10 分 已保存

Solving Tree-Structured CSPs Part 1

Consider the following tree-structured CSP that encodes a coloring problem in which neighboring nodes cannot have the same color. The domains of each node are shown.



The algorithm for solving tree-structured CSPs starts by picking a root variable. We can pick any variable for this. For this exercise, we will pick A. There are several linearizations consistent with A as the root; we will use the one shown below.



Step 1: Remove Backward

In this step we start with the right-most node (E), enforce arc-consistency for its parent (B),then do the same for the second-to-right-most node (C) and its parent (B), and so on. Execute this process, and then mark the remaining values for each variable below.

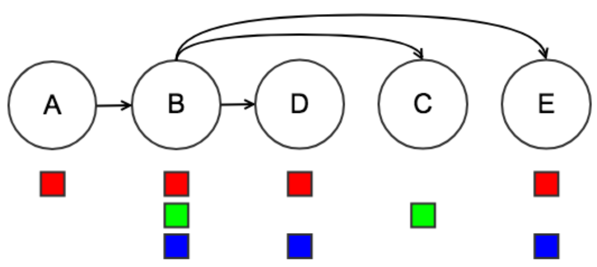
- [x] A:red
- [ ] A:green
- [ ] A:blue
- [x] B:red
- [ ] B:green
- [x] B:blue

- [ ] C:red
- [x] C:green
- [ ] C:blue
- [x] D:red
- [ ] D:green
- [x] D:blue
- [x] E:red
- [ ] E:green
- [x] E:blue

问题 18

10 分 已保存

Solving Tree-Structured CSPs Part 2



Step 2: Assign Forward

Now that all domains have been pruned, we can find the solution in a single forward pass (i.e. no need for backtracking). This is done by starting at the left-most node (A), picking any value remaining in its domain, then going to the next variable (B), picking any value in its domain that is consistent with its parent, and continue left to right, always picking a value consistent with its parent's assignment.

If at any given node there are multiple colors left that are consistent with its parent's value, break ties by picking red over green, and then green over blue.

What is the solution found by running the algorithm?

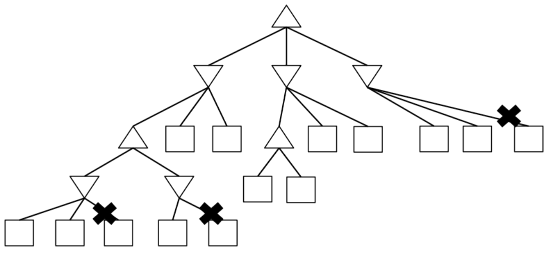
- [x] A:red
- [ ] A:green
- [ ] A:blue
- [ ] B:red
- [ ] B:green
- [x] B:blue
- [ ] C:red
- [x] C:green
- [ ] C:blue
- [x] D:red
- [ ] D:green
- [ ] D:blue
- [x] E:red
- [ ] E:green
- [ ] E:blue

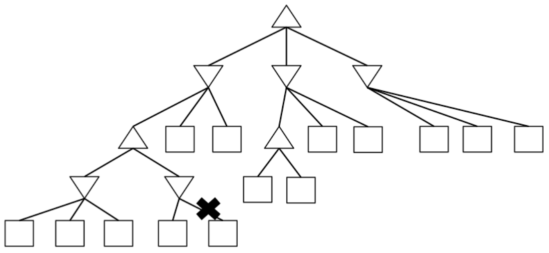
问题 19

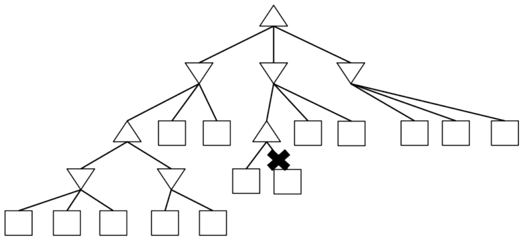
10 分 已保存

Possible Pruning

Assume we run α-β pruning, expanding successors from left to right, on a game with trees as shown below.


(a)


(b)


(c)

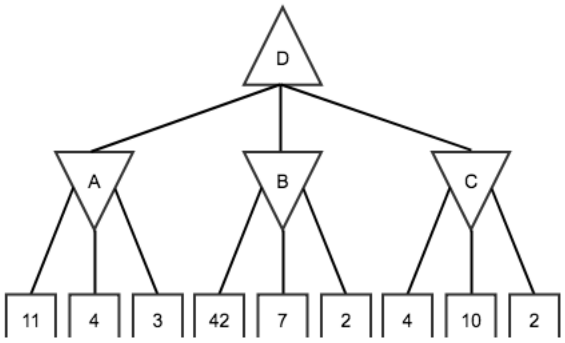
- [ ] There exists an assignment of utilities to the terminal nodes such that the pruning shown in Figure (a) will be achieved.
- [✓] There exists an assignment of utilities to the terminal nodes such that the pruning shown in Figure (b) will be achieved.
- [ ] There exists an assignment of utilities to the terminal nodes such that the pruning shown in Figure (c) will be achieved.
- [ ] None of the above.

问题 20

10 分 已保存

Minimax

Consider the zero-sum game tree shown below. Triangles that point up, such as at the top node (root), represent choices for the maximizing player; triangles that point down represent choices for the minimizing player. Outcome values for the maximizing player are listed for each leaf node, represented by the values in squares at the bottom of the tree. Assuming both players act optimally, carry out the minimax search algorithm. Enter the values for the letter nodes in the boxes below the tree.



A	3
B	2
C	2
D	3

单击“保存并提交” 以保存并提交。单击“保存所有答案” 以保存所有答案。

保存所有答案 保存并提交