*[As of 02-Nov-2020, Mock Exam Solutions are now available on blackboard!]*

Hi everyone~ Please contribute to this doc if you can! **If you would like edit access and you don’t already have it - go through the UQAttic instructions (**[**UQ attic home page**](https://uqattic.net/)**). This will give you access to all docs on UQAttic, not just this one!** Please do *not* send a request to doc owner because I may not see it! Thanks <3

# Module 1

## Question 1.1

### Part a) What data structure is best used to implement breadth-first-search?

* A Queue

### Part b) What blind search algorithm or algorithms can be implemented using priority queue?

* UCS

### Part c) Explain in your own words the benefits of using iterative-deepening depth-first search over depth-first search in a search problem.

* IDDFS is guaranteed to return an optimal solution, DFS is not guaranteed to return an optimal solution.
* IDDFS avoids the issue of going down a deep rabbit hole. (Especially when the maximum depth is very large)
* IDDFS have better both space and time complexity
  + Let b be the branching factor
  + Let m be the maximum depth
  + Let d be the depth of shallowest goal node

|  | IDDFS | DFS |
| --- | --- | --- |
| Time Complexity |  |  |
| Space Complexity |  | or  if using backtracking DFS |

## Question 1.2

### Part a) What path does iterative-deepening depth-first search with depth parameter k = 3 return for this search problem?

Confirmed in Tutorial: k=3 means the depth used in every iteration (for this question the path over 2 iterations).

S -> (1, 0) -> (2, 0) -> (2, 1) [1st iteration done; 2nd iteration start] (2, 2) -> (3, 2) -> (4, 2) = G

~~S - (1, 0) - (2, 0) - (3, 0) - [Does not reach goal]~~

* ~~This path is taken because the priority ties are in this order - UP, RIGHT, LEFT, DOWN~~
  + ~~First the agent will go the up path (ending at (0, 3)), because this doesn’t reach the goal, it will pull the next from the priority queue - starting RIGHT from S~~
  + ~~Next, it will continue along the path until it branches (2, 0).~~
  + ~~Again because of tie-breaker it will go UP and end (2, 1) - didn’t reach goal go back to branch:~~
  + ~~This time go RIGHT to (3, 0), didn’t reach the goal but have exhausted all possible nodes within the k-parameter.~~
  + ~~Return the last path travelled.~~

### Part b) Now assume a constant cost of -1 per move. What path does uniform cost search return for this search problem?

S - (1, 0) - (2, 0) - (2, 1) - (2, 2) - (3, 2) - G

* When all costs are constant, UCS will behave like BFS

### Part c) Consider a heuristic for this search problem given by h = 4-x (x is the x-coordinate).

i) Is the heuristic h consistent?

For a heuristic to be consistent needs to not overestimate a step -> YES

ii) Is the heuristic h admissible?

For a heuristic to be admissible needs to not overestimate the distance to goal -> YES

Side note: A consistent heuristic must be admissible, but an admissible heuristic does not have to be consistent.

# Module 2

## Question 2.1

### Part a) What is the domain of each queen?

Domain of each Queen: Rows {1, 2, 3, 4} (columns are already assigned)

~~NOOO IGNORE THIS: Domain of each Queen: {every square on the 4x4 board}~~

### 

### Part b) List all binary constraints between variables for this CSP. How many constraints are there?

~~Let QX represent a queen that is situated in column X.~~

~~Select the leftmost queen (QA) to have NO logical constraints (can be placed anywhere in column A).~~

~~The next queen (QB/Q(A+1)) has a constraint to have a Manhattan distance with the previous queen to be greater than or equal to 3. Then the following queen (QC/Q(B+1)) must also follow this constraint. Most basic constraint that cannot be in the immediate surroundings of all previous queens. I.E. the last queen’s distance to all other queens must be greater than or equal to 3.~~

~~Each queen can’t attack each other. So it is: Q1 can’t attack Q2, Q1 can’t attack Q3, Q1 can’t attack Q4, etc… 12 total constraints. 1 & 2, 1 & 3, 1 & 4, 2 & 3, 2 & 4, 3 & 4 and their reverses. (TECHNICALLY:: if we can satisfy all six that were listed, it’s fine - the relationship is symmetric…)~~

(A != B) ^ (A != C) ^ (A != D) ^ (B != C) ^ (B != D) ^ (C != D) ^ (B + 1 != A) ^ (B - 1 != A) ^ (C + 1 != B) ^ (C - 1 != B) ^ (C + 2 != A) ^ (C - 2 != A) ^ (D + 1 != C) ^ (D - 1 != C) ^ (D + 2 != B) ^ (D - 2 != B) ^ (D + 3 != A) ^ (D - 3 != A). There are 18 constraints (some of these can be simplified). This is in conjunctive normal form.

====================================================

Archie’s Answer from 2-Nov-2020 Tutorial Session - they were titled “Partial answers” so like don’t trust it fully yet (Also, they are not in conjunctive normal form like the student answer above :D)

12 Binary Constraints Total:

6 Row Different for rows i, j in {A, B, C, D} with i ≠ j:

* I.e. ≠
* How many? From the left, there are 3 row-diff for Q-A, two more for Q-B, and one more for Q-C; Q-D’s row diff constraint are included in the rest.

6 Diagonally Different (again using same notation previously, Qi, Qj where i and j are rows and i ≠ j).

* They cannot be the same number of columns apart as they are rows apart (i, e,≠ |i - j|)
* How many? Same as row-diff; for Q-A from the left, there are 3 Diag-difffor Q-A, two more for Q-B, and one more for Q-C; Q-D’s row diff constraints are included in the rest.

[no longer Archie’s answer] What does this mean? => there are 6 constraints that are row-related (QA can’t be on the same row as QB, can’t be on the same row as QC, can’t be on the same row as QD) etc and because these are all symmetric (QA not same row as QB is the same as QB not same row as QA) there are 6 total. Then we need to account for diagonals - this is kind of cursed because all columns and rows are affected and it’s difficult to represent in CNF definitively like the above student answer did.

~~Reference:~~ [~~https://en.wikipedia.org/wiki/Binary\_constraint~~](https://en.wikipedia.org/wiki/Binary_constraint)

### Part c) Express the problem as one of logical validity in *conjunctive normal form*.

~~(manhattan(QB, QA) >= 3) AND (manhattan(QC, QB) >= 3) AND (manhattan(QC, QA) >= 3) AND (manhattan(QD, QC) >= 3) AND (manhattan(QD, QB) >= 3) AND (manhattan(QD, QA) >= 3)~~

(QA ≠ QB) ˄ (QA ≠ QC) ˄ (QA ≠ QD) ˄ (QB ≠ QC) ˄ (QB ≠ QD) ˄ (QC ≠ QD) ˄

(|QA - QB|≠1) ˄ (|QA - QC|≠2) ˄ (|QA - QD|≠3) ˄ (|QB - QC|≠1) ˄ (|QB - QD|≠2) ˄ (|QC - QD|≠1)

### 

### Part d) Assume a partial assignment is given, where Q-A is placed in row 3. Apply backtracking search starting from this partially-assigned CSP. Use the variable ordering (Q-B, Q-C, Q-D) and the variable domain order 1,2,3,4 to expand nodes in the search graph. List all variable assignment and removal operations, and any backtracking operations.

(backtrack at any inconsistency)

Select QB, put in row 1 -> no inconsistencies, remove QB, continue to next:

Select QC, put in row 1 -> inconsistency with QB (row). put in row 2 -> inconsistency with QB (diagonal). Put in row 3 -> inconsistency with QA (row). Put in row 4 -> no inconsistencies, remove QC, continue to next.

Select QD, put in row 1 -> QB inconsistency. Row 2 -> No inconsistencies, remove QD, continue to next.

No more unassigned variables -> exit.

### Part e) Give a solution to this CSP.

. Q . .

. . . Q

Q . . .

. . Q .

## 

## Question 2.2

### Part a) Construct a truth table to show that ¬(p ∨ q) is logically equivalent to (¬p ∧ ¬q).

| p | q | p v q | ¬(p v q) | ¬p | ¬q | ¬p ∧ ¬q |
| --- | --- | --- | --- | --- | --- | --- |
| T | T | T | F | F | F | F |
| T | F | T | F | F | T | F |
| F | T | T | F | T | F | F |
| F | F | F | T | T | T | T |

### Part b) Given the premises (p ⇒ q) and (r ⇒ s), use the Propositional Resolution rule to prove the conclusion (p ∨ r ⇒ q ∨ s).

### 

Courtney’s Proof by Contradiction:

Premise 1: not p or q

Premise 2: not r or s

Conclusion: not (p or r) or (q or s)

Negate Conclusion & set to TRUE: (p or r) and not (q or s) = TRUE

P or R = TRUE

Not (Q or S) = TRUE

Therefore Q = FALSE, S = FALSE

For P1 to be TRUE, not p = TRUE, p = FALSE

For P2 to be TRUE, not r = TRUE, r = FALSE

(P or R) != TRUE, (FALSE or FALSE = FALSE != TRUE)

This is a contradiction -> Good! Therefore, the statement is valid!

### 

# 

# Module 3

## Question 3.1

### The decomposability axiom of the rational preferences utility model asserts that “there is no fun in gambling.” Explain how an agent whose preferences violate the decomposability axiom can be manipulated using the concept of a money pump.

Money Pump:

If the axiom is violated and agent gets some utility from gambling -> then would be NOT indifferent to 2 lotteries with the same probabilities in the end. Set up a transitive cycle of lotteries, be manipulated by setting up in a way that keeps taking more complicated lottery and losing something each time. Leads to ability to set up a transitive cycle and suck your money :(

For this answer just explain in words.

**---- From here on out, answers are in the tutorial recording (2-Nov-2020)**

## Question 3.2

### Part a) What is the transition function for each action starting in the mud? That is, write out T(s,a,s’) for each s’ with non-zero transition probability, starting at s=(2,2).

As mentioned in the tutorial recording (2-Nov-2020), there are 4 actions and 4 ending states (successful up, successful right, successful down and stuck in the same place - note that there is an obstacle on the left). This would typically mean there are 8 rows in the table below HOWEVER the question specifically asks for transitions with non-zero probability, so there are 7 non-zero transition probabilities.

| State (s) | Action (a) | Next State (s’) | Probability |
| --- | --- | --- | --- |
| (2, 2) | UP | (2, 1) | 0.2 |
| (2, 2) | UP | (2, 2) | 0.8 |
| (2, 2) | RIGHT | (3, 2) | 0.2 |
| (2, 2) | RIGHT | (2, 2) | 0.8 |
| (2, 2) | DOWN | (2, 1) | 0.2 |
| (2, 2) | DOWN | (2, 2) | 0.8 |
| (2, 2) | LEFT | (2, 2) | 1 |

### Part b) Initialise all values to 0, and compute three iterations of (synchronous) value iteration for the gridworld above. [List only the non-zero iterates; all unstated values will be assumed to be equal to 0.] What are the value function iterates (i.e. approximate values) for each state after:

#### i. the first iteration

Only the cells next to the goal will update their values (probability is 1) so:

1 \* 10 = 10

Cells (3, 2) and (4, 3)

#### ii. the second iteration

Now expanding. Cell (4, 4) will update: 1 \* (0 + 0.8 \* 10) = 8

Cell (2, 2) is the mud cell, so there’s a 0.2 chance to move into (3, 2);:

(2, 2) = 0.2 \* (0 + 0.8 \* 10) = 1.6

#### iii. the third iteration.

Expanding again:

Cell (3, 4): 0.8 \* 8 = 6.4

Cell (2, 2): Failing to move right + staying: 0.2 \* 0.8 \* 8 + 0.8 \* 0.8\*1.6 = 2.304

Cell (2, 1): 0.8 \* (2, 2) = 1.28

Cell (2, 3): 0.8 \* (2, 2) = 1.28

# Module 4

### Part a) What values does the Q-function attain if we initialise the Q-values to 0 and replay the experience in the table exactly two times? Use a learning rate 𝛼 = 0.6, and discount factor 𝛾 = 0.8. List only the non-zero approximate Q-values; all unstated Q-values are assumed to be equal to 0.

First 3 states are all 0 (nothing updated) only 2nd run through we care about q-values. Only from (3, 2) -> (4, 2) we get a reward and key values change.

### Part b) Using these Q-values and the epsilon-greedy exploration strategy describe above, what are the probabilities of taking each action next time the SARSA agent gets stuck in the mud?

# Module 5

Answered during the last lecture~