FOR INTERNAL SCRUTINY (date of this version: 8/1/2018)

Specimen Answers

Part A

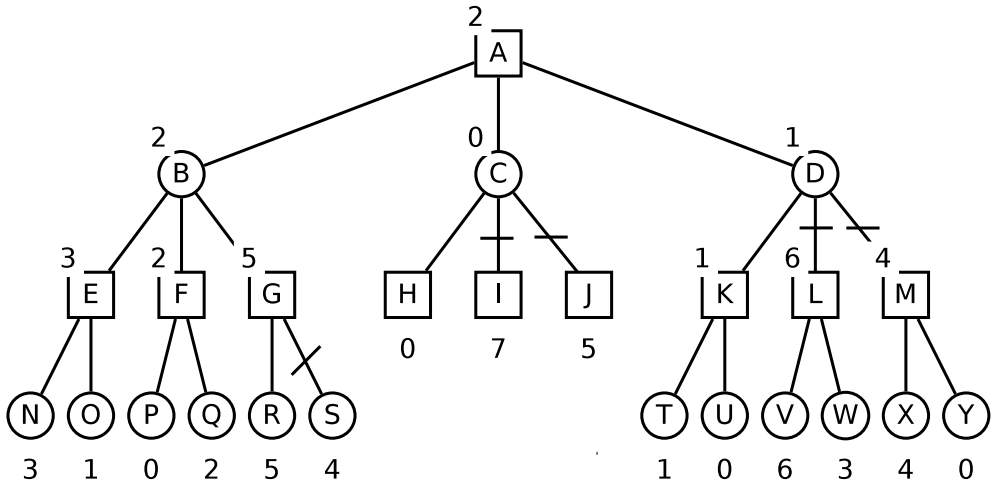
1. d
2. c
3. c
4. b
5. e
6. b
7. a
8. e
9. e
10. a
11. d
12. c
13. b
14. e
15. a
16. e
17. c
18. e
19. d
20. d

Part B

1. Adversarial Search

   (a) Scores are propagated from bottom to top. MAX nodes select the maximum score of its children while MIN nodes select the minimum score of its children. The recommended next move is from the root to one of its children that shares its score.

   The propagated scores for the given game tree are shown below. The recommended move for MAX will be to node B.



   3 marks for explaining the minimax algorithm. 2 marks for specifying the non-leaf node scores and 1 mark for identifying the recommended move.

   (b) From Russell & Norvig:

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

2 mark for first function, with correct initial values for α and β. 2 marks for MAX-VALUE and 2 marks for MIN-VALUE. The algorithms do not have to be given exactly as above. Any reasonable description of all the important aspects is acceptable.

(c) The figure above shows which nodes are pruned (crossed). These are nodes S, I, J, L, M.

- S: MAX chooses at least 5 (R) but MIN chooses at most 2 (F).
- I/J: MAX chooses at least 2 (B) but MIN chooses at most 0 (H).
- L/M: MAX chooses at least 2 (B) but MIN chooses at most 1 (K).

4 marks for specifying pruned nodes. 4 marks for explaining why.

(d)
- The opponent plays optimally, i.e., it optimises score in opposite direction without mistakes.
- Minimax will never obtain worse scores because it optimises against worst-case opponent. Minimax may achieve higher scores against a non-optimal opponent because opponent makes mistakes.

2 marks for first question. 3 marks for second question.

2. Propositional Logic

(a)
- $\neg A$ is true iff A is false

- $A \wedge B$ is true iff A is true and B is true
- $A \vee B$ is true iff A is true or B is true
- $A \Rightarrow B$ is true iff A is false or B is true
- $A \Leftrightarrow B$ is true iff $A \Rightarrow B$ is true and $B \Rightarrow A$ is true

1.5 marks for $\neg, \wedge, \vee$. 1.5 marks for $\Rightarrow, \Leftrightarrow$.

(b)
- A sentence is in CNF if it is a conjunction of clauses. A clause is a disjunction of literals. A literal is a proposition or negated proposition (symbol).
- Every sentence in propositional logic has an equivalent CNF sentence.

1.5 marks for defining CNF. 1.5 marks for stating that every sentence has equivalent CNF.

(c)
- $(A \vee B) \wedge (A \vee C) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee \neg C)$.
- Two sentences are equivalent if every model of one sentence is also a model of the other sentence ($\varphi \equiv \psi$ iff $\varphi \vDash \psi$ and $\psi \vDash \varphi$).

2 marks for giving CNF. 1 mark for defining equivalence.

(d) From Russel & Norvig:

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, [])

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols*–P, [P = *value*|*model*])
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols*–P, [P = *value*|*model*])
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, [P = *true*|*model*]) **or**
    DPLL(*clauses*, *rest*, [P = *false*|*model*])

1 mark for first function, 4 marks for second function. The algorithms do not have to be given exactly as above. Any reasonable description of all the important aspects is acceptable.

(e)
- Early termination: A clause is true if one of its literals is true, and a sentence is false if any of its clauses is false. Speed-up because don't have to check remaining literals/clauses.

- Pure symbol: Make pure literals true. Speed-up because it can never make any clause false.
- Unit clause: Make literal in unit clause true. Speed-up because it has to be true for satisfiability.

2 marks for giving the heuristics. 3 marks for explaining speed-ups.

(f) The sentence is satisfiable. A model is given by $A = true, B = false, C = false$.
3 marks for stating that sentence is satisfiable and giving a model.

(g)
- A satisfiability algorithm is complete if it terminates (i.e. returns either true or false after finite time) for every sentence.
- The algorithm is sound if any returned result is correct.
- DPLL is sound and complete.

1 mark for explaining completeness, 1 mark for explaining soundness, 1 mark for stating that DPLL has both properties.

Part C

1. Planning

(a)

S:	constants, Shakey
$R1, R2, R3, R4, C$:	constants (the rooms and corridor)
$S1, S2, S3, S4$:	constants (the switches)
$B1, B2, B3, B4$:	constants (the boxes)
$in(x, r)$:	switch, box, or door is in room y
$at(x, y)$:	x (Shakey or box) is at location y (box or switch)
$room(x)$:	x is a room
$connected(x, y)$:	rooms x and y are connected by a door.
$box(x)$:	x is a box
$switch(x)$:	x is a switch
$at(x, y)$:	x is at location y
$onfloor$:	proposition; Shakey is on the floor (i.e., not on a box)
$on(x, y)$:	x is on y
$down(x)$:	switch x is down.

(There are other options; they are fine if they fully describe the state and action space. Penalise the language if it fails to do this, or if it distinguishes states that don't need to be distinguished (i.e., it's not a sufficiently compact representation). For instance, you don't need to refer to the doors in this planning problem.).

(b)

$Action(move(r, y),$
PRECOND:$onfloor, room(r), in(y, r), in(S, r)$
EFFECT:$at(S, y)$(**when** $at(S, x) : \neg at(S, x)$))

$Action(move\text{-}rooms(r1, r2),$
PRECOND:$onfloor, room(r1), room(r2), in(S, r1), connected(r1, r2)$
EFFECT:$in(S, r2), \neg in(S, r1),$ (**when** $at(S, x) : \neg at(S, x)$))

$Action(push(x, r, y),$
PRECOND:$onfloor, box(x), room(r), at(S, x), in(x, r), in(S, r), in(y, r)$
EFFECT:$at(x, y), at(S, y),$ (**when** $at(S, z), switch(z) : \neg at(S, z)$))

$Action(push\text{-}rooms(x, r1, r2),$
PRECOND:$onfloor, at(S, x), in(x, r1), in(S, r1), connected(r1, 2)$
EFFECT:$in(x, r2), in(S, r2), \neg in(x, r1), \neg in(S, r1))$

$Action(ClimbUp(x),$
PRECOND:$box(x), at(S,x), onfloor$
EFFECT:$\neg onfloor, on(S,x))$

$Action(ClimbDown(x),$
PRECOND:$box(x), on(S,x)$
EFFECT:$\neg on(S,x), at(S,x), onfloor)$

$Action(Flip(x),$
PRECOND:$at(S,x)$
EFFECT:(**when** $Down(x) : \neg Down(x)), (\textbf{when } \neg Down(x) : Down(x)))$

(Getting the move and push actions correct is really tricky, because you have to make sure you delete appropriate at relations. Penalise them 2 points if they don't get this part right. Don't penalise the same mistakes more than once.)

(c) Initial state is as follows:

$room(R1) \wedge room(R2) \wedge room(R3) \wedge room(R4) \wedge$
$switch(S1) \wedge switch(S2) \wedge switch(S3) \wedge switch(S4) \wedge$
$box(B1) \wedge box(B2) \wedge box(B3) \wedge box(B4) \wedge$
$in(S1,R1) \wedge in(S2,R2) \wedge in(S3,R3) \wedge in(S4,R4) \wedge$
$in(B1,R1) \wedge in(B2,R1) \wedge in(B3,R1) \wedge in(B4,R1) \wedge$
$connected(R1,C) \wedge connected(R2,C) \wedge connected(R3,C) \wedge connected(R4,C) \wedge$
$connected(C,R1) \wedge connected(C,R2) \wedge connected(C,R3) \wedge connected(C,R4) \wedge$
$in(S,R3) \wedge \neg down(S1) \wedge down(S2) \wedge down(S3) \wedge \neg down(S4) \wedge onfloor$

Goal state is:

$$in(B3,R4) \wedge \neg down(S2)$$

Give 2 points for getting the initial state right, and 1 for the goal. Penalise if any of the properties of the initial state are missing. Note how you cannot assume that connected is symmetric! Penalise if they didn't get this right. Penalise also if the goal is over-specified.

(d) A valid plan is as follows:

move-rooms$(S,R3,C)$; *move-rooms*$(S,C,R1)$; *move*$(S,B3,R1)$;
push-rooms$(B3,R1,C)$; *push-rooms*$(B3,C,R2)$; *push*$(B3,R2,S2)$;
ClimbUp$(B3)$; *Flip*$(S2)$;
ClimbDown$(B3)$; *push-rooms*$(B3,R2,C)$; *push-rooms*$(B3,C,R4)$

(e)

$$\begin{array}{l} Action(ClimbUp(x), \\ \quad \textsc{Precond}{:}box(x), at(S,x), onfloor \\ \quad \textsc{Effect}{:}onfloor \vee (\neg onfloor \wedge on(S,x))) \end{array}$$

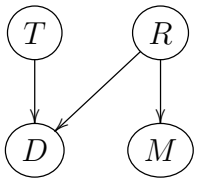
$$\begin{array}{l} move\text{-}rooms(S, R3, C); move\text{-}rooms(S, C, R1); move(S, B3, R1); \\ \quad push\text{-}rooms(B3, R1, C); push\text{-}rooms(B3, C, R2); push(B3, R2, S2); \\ \quad\quad L : [ClimbUp(B3); \\ \quad\quad\quad \textbf{if } on(S, B3) \\ \quad\quad\quad \textbf{then } Flip(S2)\, ClimbDown(B3); \\ \quad\quad\quad\quad push\text{-}rooms(B3, R2, C); push\text{-}rooms(B3, C, R4) \\ \quad\quad\quad \textbf{else } L] \end{array}$$

2. Bayesian Inference

(a) Random Variables are:

T	time of day, boolean: $< 6pm$
R	route: 1, 2, 3
D	decks, boolean: s, $\neg s$
M	destinations, boolean: m, $\neg m$

The Bayes Net is as shown:



The student may have omitted the variable corresponding to M; don't deduct points for this since its CPT is deterministic. Deduct 1 point for each wrong connection or superfluous/missing variables. Deduct 2 points for not defining the random variables.

(b)

$P(T)$
0.5

R	$P(R)$
1	0.6
2	0.3
3	0.1

T	R	$P(D)$
$< 6pm$	1	0.9
$> 6pm$	1	1
$< 6pm$	2	1
$> 6pm$	2	0.1
$< 6pm$	3	0.2
$> 6pm$	3	0.2

R	M
1	0
2	0
3	1

Deduct 0.5 points for each wrong cell.

(c)

$$
\begin{aligned}
P(\neg m|s,>6pm) &= \textstyle\sum_r P(\neg m, R|s,>6pm) \\
&= \textstyle\sum_r P(\neg m|R,s,>6pm)P(R|s,>6pm) \\
&= \textstyle\sum_r P(\neg m|R)P(R|s,>6pm) \\
&= P(\neg m|1)P(1|s,>6pm) + P(\neg m|2)P(2|s,>6pm) + \\
&\quad P(\neg m|3)P(3|s,>6pm) \\
&= P(1|s,>6pm) + P(2|s,>6pm) \\
P(1|s,>6pm) &\propto P(1)P(s|1,>6pm)P(>6pm) \\
&= 0.6 * 1 * 0.5 \\
&= 0.3 \\
P(2|s,>6pm) &\propto P(2)P(>6pm)P(s|2,>6pm) \\
&= 0.3 * 0.1 * 0.5 \\
&= 0.015 \\
P(3|s,>6pm) &\propto P(3)P(>6pm)P(s|3,>6pm) \\
&= 0.1 * 0.2 * 0.5 \\
&= 0.01 \\
P(1|s,>6pm) &= \tfrac{0.3}{0.3+0.015+0.1} = 0.72 \\
P(2|s,>6pm) &= \tfrac{0.015}{0.3+0.015+0.1} = 0.04 \\
P(\neg m|s,>6pm) &= 0.72 + 0.04 \\
&= 0.76
\end{aligned}
$$

If the original Bayes Net didn't include the variable M, then the probability the student needs to calculate is $P(\neg 3|s,>6pm)R$. If they have done this correctly then that is fine. Deduct 1 point for each error in the derivation, or if a crucial step in the derivation is missing, but don't penalise repeated errors.