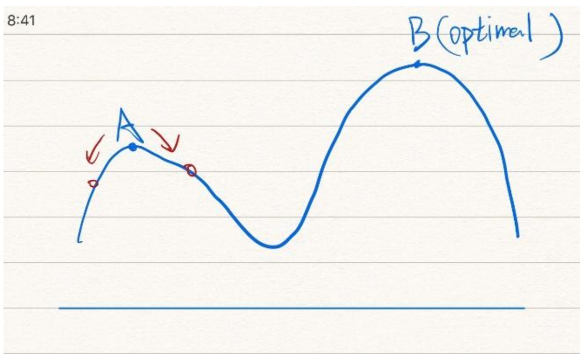
2. **Describe the *Simulated Annealing* algorithm. When does one prefer to use the algorithm? What is the role of the parameter "temperature" in the algorithm?**

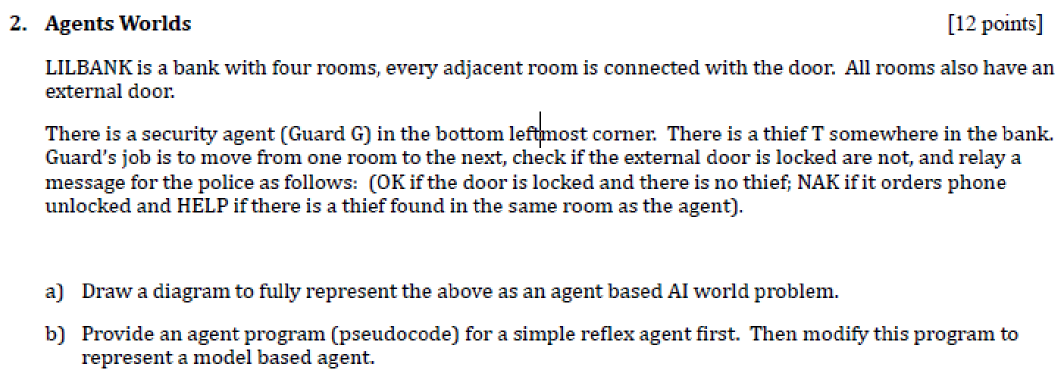
SIMULATED ANNEALING: If J(Y(i+1))>= J(Y(i)) (that is, a better solution is obtained after the move), the move is always accepted. If J(Y(i+1))< J(Y(i)) (that is, the solution after the movement is worse than the current solution), the movement is accepted with a certain probability, and this probability gradually decreases over time (gradually Only by lowering can it tend to be stable). The calculation of "a certain probability" here refers to the annealing process of metal smelting, which is also the origin of the name of the simulated annealing algorithm.
WHEN: When the hill climbing method finds the local optimal solution. To find the maximum value of f(x), the mountain climbing algorithm will stop searching when it reaches point A, because the values around point A are less than the value of point A. The solution adopted by the simulated annealing algorithm is to select the points on both sides of A with a certain probability, although the points on both sides of A are not locally optimal solutions, so there is a certain probability to search for point D, and then search for point B, and finally obtain Global optimal solution.
TEMPERATURE: When the temperature is T, the probability of a temperature drop with an energy difference of dE is P(dE), expressed as: P(dE) = exp(dE/(kT)) where k is a constant, exp represents the natural exponent, and dE <0. The higher the temperature, the greater the probability of a cooling with an energy difference of dE; the lower the temperature, the smaller the probability of cooling. Also, since dE is always less than 0, dE/kT <0, so the value range of P(dE) is (0,1). As the temperature T decreases, P(dE) will gradually decrease. By slowly lowering the temperature, the algorithm may eventually converge to the global optimal solution

3. **When does *Simulated Annealing* perform better than *Hill Climbing*? How is this better performance achieved? Would you ever prefer *HC* to *SA*? If yes, when?**



When there are the local optimal solutions as shown, SA is better as HC. Because HC may stop when find a local optimal solution.
The simulated annealing algorithm has asymptotic convergence, and it has been proved in theory to be a global optimization algorithm that converges to the global optimal

solution with probability l;The first step is to generate a new solution in the solution space from the current solution by a generating function, and the second step is to calculate the difference of the objective function corresponding to the new solution. The third step is to judge whether the new solution is accepted. The judgment is based on an acceptance criterion. The most commonly used acceptance criterion is the Metropolis criterion. The fourth step is to replace the current solution with the new solution when the new solution is confirmed to be accepted.
However, because the simulated annealing algorithm is a random algorithm, when there is no local optimal solution but only a global optimal solution, the efficiency of the hill climbing algorithm is higher than that of the simulated annealing algorithm



2. Agents Worlds [12 points]

LILBANK is a bank with four rooms, every adjacent room is connected with the door. All rooms also have an external door.

There is a security agent (Guard G) in the bottom leftmost corner. There is a thief T somewhere in the bank. Guard's job is to move from one room to the next, check if the external door is locked are not, and relay a message for the police as follows: (OK if the door is locked and there is no thief; NAK if it orders phone unlocked and HELP if there is a thief found in the same room as the agent).

a) Draw a diagram to fully represent the above as an agent based AI world problem.

b) Provide an agent program (pseudocode) for a simple reflex agent first. Then modify this program to represent a model based agent.

a)
agent: Guard G
variable: Thief and Door

	thief	Door locked(no T)	Door opened(no T)
G at 1	HELP	OK	NAK
G at 2	HELP	OK	NAK
G at 3	HELP	OK	NAK
G at 4	HELP	OK	NAK

There are total 64 state in the world.

b)
```
FUNCTION Reflex-Agent([door,thief]) returns a message
    If door = locked and thief = 0 returns OK
    Else if door = unlock and thief = 0 returns NAK
    Else if thief = 1 returns HELP
```

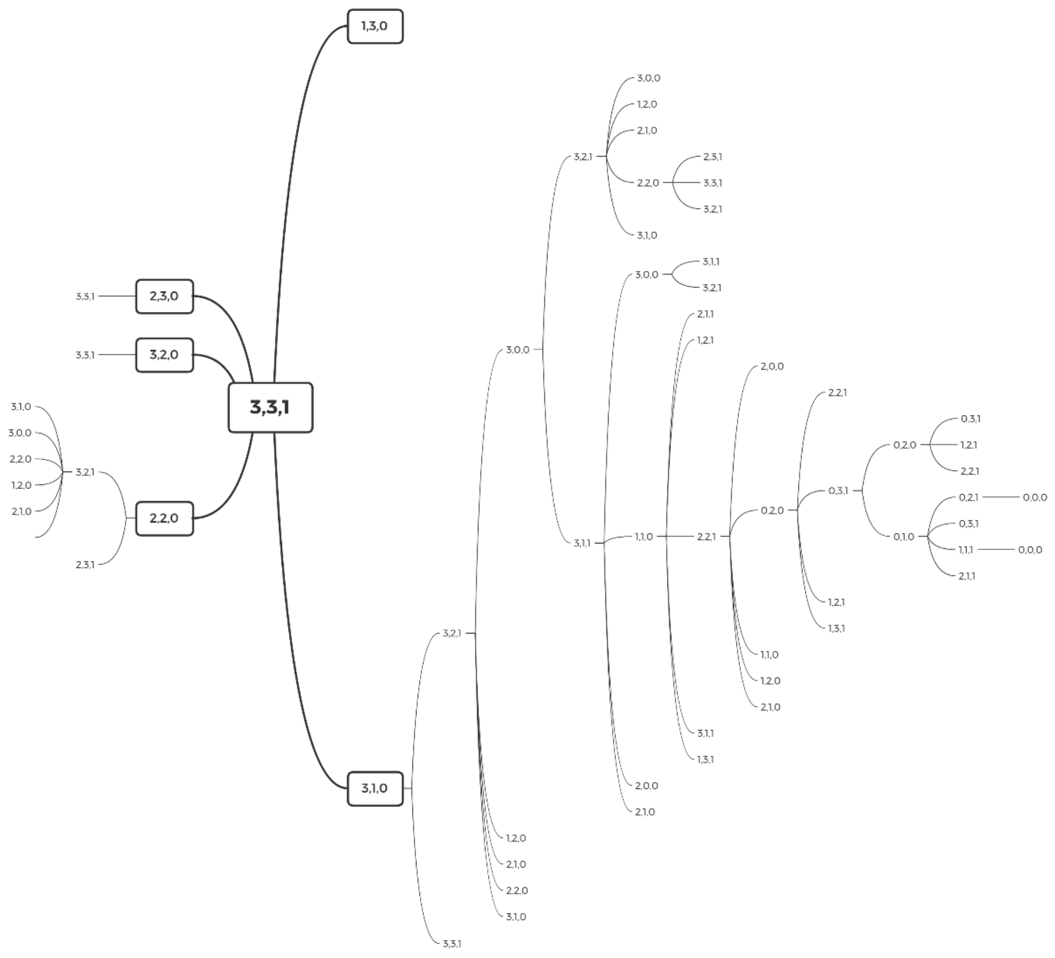
Modified:
```
FUNCTION Model-Based-Reflex-Agent(percept) returns a message
    Persistent:
        state: the agent's current conception of the world state
        model: a description of how the nest state depends on current and action
        rules: a set of condition-action rules
        message: the most recent message, initially none
    state<- UPDATE-STATE(STATE,MSG,PERCEPT,MODEL)
    rule<- RULE-MATCH(STATE,RULES)
    MSG<-rule:MSG
    Return MSG

FUNCTION rule(state):
    If door = locked and thief = 0 returns OK
    Else if door = unlock and thief = 0 returns NAK
    Else if thief = 1 returns HELP
```

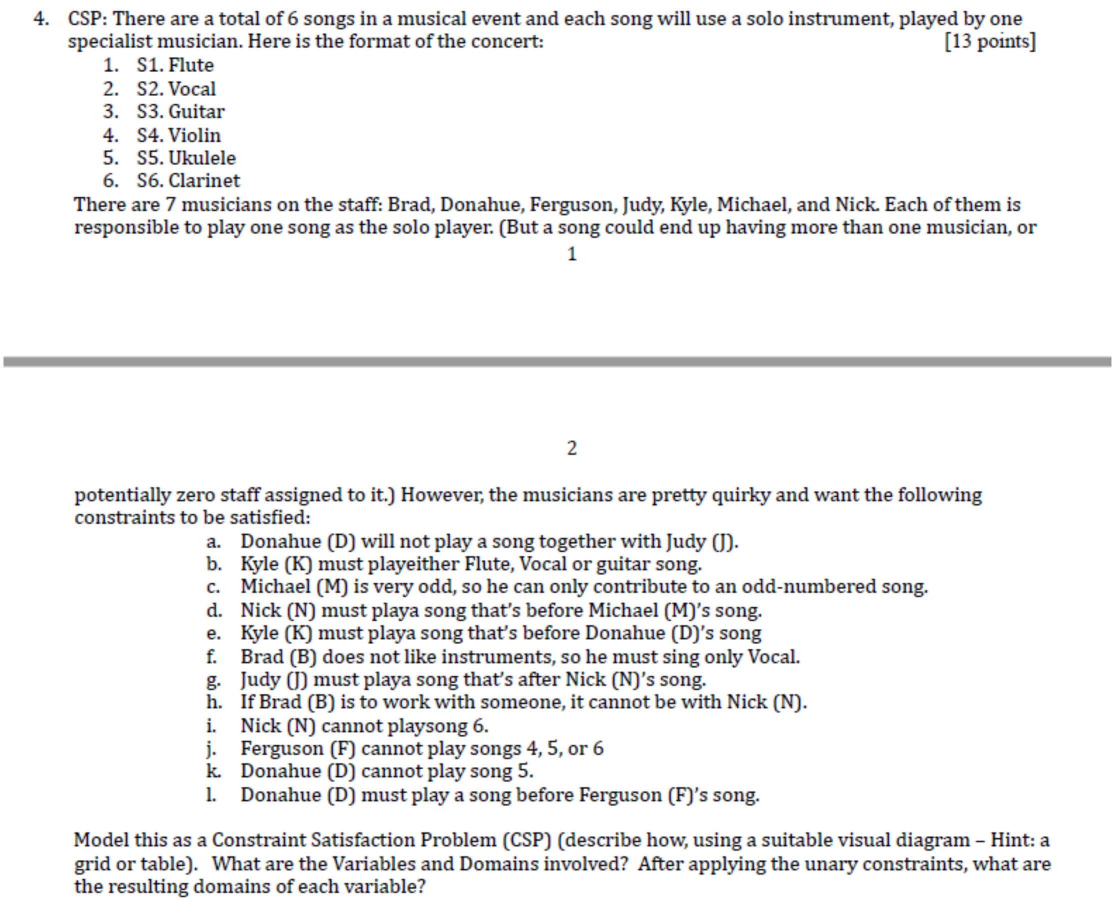
3. **What is AND-OR search? Give an example where and AND-OR Search tree function would be useful in solving the Missionaries and Cannibals problem from Question 1 above. Show with a figure. [7 points]**

In a deterministic environment, the branch is formed by the agent's choice in each state. We call these nodes or nodes. In an uncertain environment, the formation of branches may be the consequence of each action chosen by the environment. These nodes are called AND nodes. These two types of nodes form an AND-OR tree. The solution of the AND-OR search problem is a subtree: (1) There is a target node on each leaf, (2) an activity is regulated on the OR node, and (3) all possible consequences are included on the AND node.

Define a triple array, the first item represents the number of missionaries in the left side, and the second item represents the number of cannibals in the left side, and the third item represents whether the boat is in left side. The tree may be like:



1,3,0
2,3,0
3,3,1
3,2,0
3,3,1
3,3,1
3,1,0
3,0,0
2,2,0
1,2,0
2,1,0
3,2,1
2,2,0
2,3,1
3,1,0
3,2,1
3,0,0
3,2,1
3,0,0
1,2,0
2,1,0
2,2,0
2,3,1
3,3,1
3,2,1
3,1,0
3,0,0
3,1,1
3,2,1
3,1,1
1,1,0
2,1,1
1,2,1
2,2,1
2,0,0
0,2,0
2,2,1
0,3,1
0,2,0
0,3,1
1,2,1
2,2,1
0,1,0
0,2,1
0,0,0
0,3,1
1,1,1
0,0,0
2,1,1
1,2,1
1,3,1
1,1,0
1,2,0
2,1,0
3,1,1
1,3,1
2,0,0
2,1,0
1,2,0
2,1,0
2,2,0
3,1,0
3,3,1



4. CSP: There are a total of 6 songs in a musical event and each song will use a solo instrument, played by one specialist musician. Here is the format of the concert: [13 points]
   1. S1. Flute
   2. S2. Vocal
   3. S3. Guitar
   4. S4. Violin
   5. S5. Ukulele
   6. S6. Clarinet

   There are 7 musicians on the staff: Brad, Donahue, Ferguson, Judy, Kyle, Michael, and Nick. Each of them is responsible to play one song as the solo player. (But a song could end up having more than one musician, or

potentially zero staff assigned to it.) However, the musicians are pretty quirky and want the following constraints to be satisfied:

a. Donahue (D) will not play a song together with Judy (J).
b. Kyle (K) must playeither Flute, Vocal or guitar song.
c. Michael (M) is very odd, so he can only contribute to an odd-numbered song.
d. Nick (N) must playa song that's before Michael (M)'s song.
e. Kyle (K) must playa song that's before Donahue (D)'s song
f. Brad (B) does not like instruments, so he must sing only Vocal.
g. Judy (J) must playa song that's after Nick (N)'s song.
h. If Brad (B) is to work with someone, it cannot be with Nick (N).
i. Nick (N) cannot playsong 6.
j. Ferguson (F) cannot play songs 4, 5, or 6
k. Donahue (D) cannot play song 5.
l. Donahue (D) must play a song before Ferguson (F)'s song.

Model this as a Constraint Satisfaction Problem (CSP) (describe how, using a suitable visual diagram – Hint: a grid or table). What are the Variables and Domains involved? After applying the unary constraints, what are the resulting domains of each variable?

	S1	S2	S3	S4	S5	S6
B		√				
D	√	√	√	√		√
F	√	√	√			
J	√	√	√			
K						
M	√		√		√	
N	√	√	√	√	√	

K before D, D before F, N before M, J after N,B not with N

Variables: the song every musician played

Domains: 1-6

Resulting domain:

	S1	S2	S3	S4	S5	S6
B		√				
D		√				
F			√			
J		√	√			
K	√					
M			√		√	
N	√					

5. MINI-MAX SEARCH IN GAME TREES:

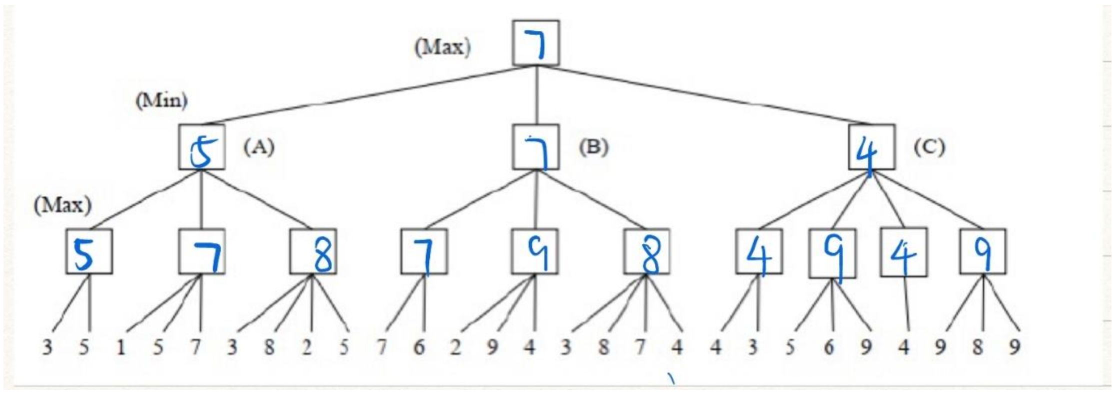
The game tree below illustrates a position reached in the game. Process the tree left-to-right. It is Max's turn to move. At each leaf node is the estimated score returned by the heuristic static evaluator. [20 points]

a) Fill in each blank square with the proper mini-max search value.

b) What is the best move for Max? (write A, B, or C)

c) What score does Max expect to achieve?

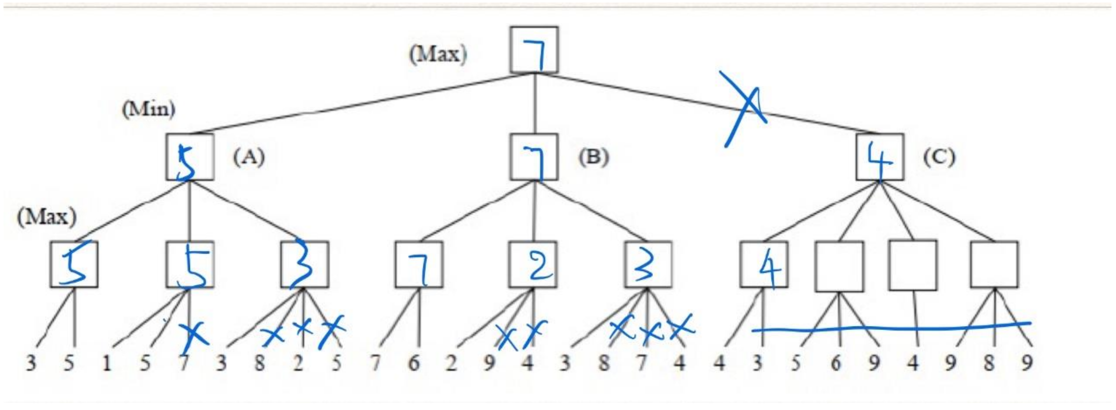
a)



b) B

c) 8

6. **ALPHA-BETA PRUNING:** Process the tree left-toright. This is the same tree as above. You do not need to indicate the branch node values again. [25 points]



7. For each English sentence below, write the letter corresponding to its best or closest FOPC (FOL) sentence (wff, or well-formed formula). The first one is done for you, as an example. [15 points]

"Every butterfly likes some flower."
A. ∀x ∀y Butterfly(x) ∧ Flower(y) ∧ Likes(x, y)
B. ∀x ∃y Butterfly(x) ∧ Flower(y) ∧ Likes(x, y)
C. ∀x ∀y Butterfly(x) ⇒ (Flower(y) ∧ Likes(x, y))
D. ∀x ∃y Butterfly(x) ⇒ (Flower(y) ∧ Likes(x, y))

D

"All butterflies are insects."
A. ∀x Butterfly(x) ∧ Insect(x)
B. ∀x Butterfly(x) ⇒ Insect(x)
C. ∃x Butterfly(x) ∧ Insect(x)
D. ∃x Butterfly(x) ⇒ Insect(x)

B

"For every flower, there is a butterfly that likes that flower."
A. ∀x ∃y Flower(x) ∧ Butterfly(y) ∧ Likes(y, x)
B. ∀x ∃y [Flower(x) ∧ Butterfly(y)] ⇒ Likes(y, x)
C. ∀x ∃y Flower(x) ⇒ [Butterfly(y) ∧ Likes(y, x)]
D. ∀x ∀y Flower(x) ∧ Butterfly(y) ∧ Likes(y, x)

C

"Every butterfly likes every flower."
A. ∀x ∀y [Butterfly(x) ∧ Flower(y)] ⇒ Likes(x, y)
B. ∀x ∀y Butterfly(x) ⇒ [Flower(y) ∧ Likes(x, y)]
C. ∀x ∀y Butterfly(x) ∧ Flower(y) ∧ Likes(x, y)
D. ∀x ∃y [Butterfly(x) ∧ Flower(y)] ⇒ Likes(x, y)

A

"There is some butterfly in Irvine that is pretty."
A. ∀x Butterfly(x) ∧ In(x, Irvine) ∧ Pretty(x)
B. ∃x Butterfly(x) ∧ In(x, Irvine) ∧ Pretty(x)
C. ∀x [Butterfly(x) ∧ In(x, Irvine)] ⇒ Pretty(x)
D. ∃x Butterfly(x) ⇒ [In(x, Irvine) ∧ Pretty(x)]

B