CS671 homework 5 Shuowen Wei

1. Performance Measure: goal accomplished (pick up food/gold and store into granary/bank, or kill red/blue dragon and destroy the castle of the same kind), time/steps cost to achieve the goal

Environments: As state in the question: food, gold, granary, bank, dragon (red/blue), castle (red/blue), doors (only one door is open at some time point).

Actuators: move up, move down, move left, move right, open doors, pick up, put down, attack.

Sensors: eyes or camera sensor

For simple reflex agent:

It will randomly choose a direction to move and open the door, if meets food or golden, pick it up, if meets dragon or castle, attack it, if meets granary or bank, put food or gold down if it has, otherwise just keep moving randomly to the next place, since simple agent doesn't maintain a goal test function, thus he will never know he has already finished the goal or not. But we can let him stop once he put down food at granary of gold at bank (since he can't remember he had killed a dragon or destroy the castle or not before, we can't let him stop after doing one of the two things)

For reflex agent (With state):

This agent will basically do the same thing like simple agent, but he can keep an internal state of the world he had already explored and update it when he explores something new (like moving to the unknown area) or changes something in the world (like, picking up food/gold or killing the dragon, then the food, gold or dragon are gone). Since he also don't have a goal function as guidance, he will also never know whether he has finished the goal or not, but he will trend to explore the unknown area instead of just moving randomly, in the case, he'll trend to explore unknown area when he picks up food although the position of granary is already in his internal state. We can set him stop once he finishes one of the two goals (since he has a memory of killing the dragon and destroying the castle of not.)

For the goal-based agent:

Everything is the same with reflex agent except that he maintain a goal function in addition to keep the internal state. So, he will randomly move and explore the world at the beginning, then when he meets a dragon and kills it, based on the goal function he has, he will trend to go back to the corresponding castle and destroy it if the position of that castle is already in his internal state, otherwise he will explore the

unknown area to try to find the castle, if he picks food and gold on his way to find the castle, he will pick them up and go back to the granary or bank and put them down if their locations are already in his internal state (this is very likely just because food and gold are either in the corner or on the edge.). He will stop once finishing the goal.

For utility-based agent:

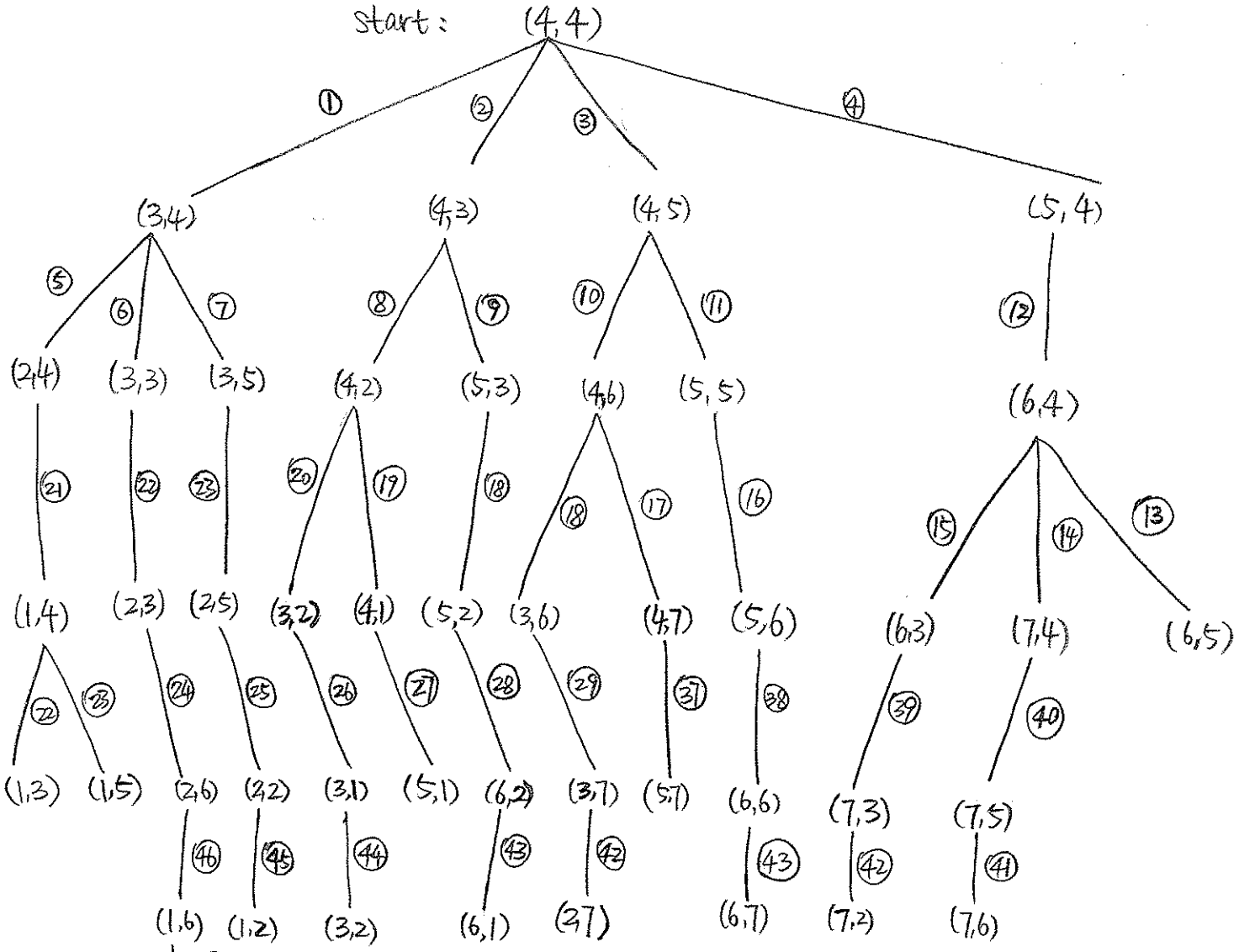
This agent is smarter than goal-based agent, since he also maintains a utility function in order to maximize the expected utilities by, in this case, minimizing the cost of cost/time. If this agent meets the dragon and kills it, it will try to find the shortest path to the corresponding castle if it's in his internal, thus he will not pick up food or gold on his way to find the castle since "pick up" action is a step-cost of -1. In the other, if he already picks up food and gold, he will not kill the dragon if met and will try to decide to go to the granary or bank first, whichever is closer. And he will stop once finished the goal.

The utility-based agent will perform the best in this situation.

2. Goal: collect the food and store it in the nearest granary.

Notatain: since this the 7×7 table, denote every position as (x, y). $x, y = 1, 2, \cdots, 7$, thus the agent's start position is $(4,4)$, food is at $(1,7)$ and $(7,1)$, the nearest granary to food $(1,7)$ is $(4,7)$; the nearest granary to food $(7,1)$ is at $(6,3)$. [Assume the distance in this problem is Manhatten distance.]

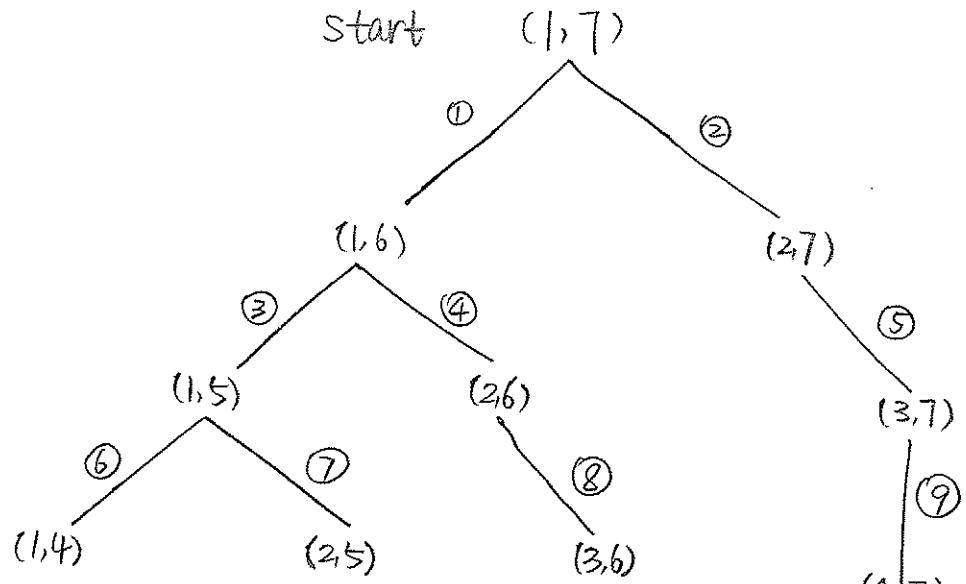
BFS (under graph search:)



⇒ find the food at position (1,7), pick it up

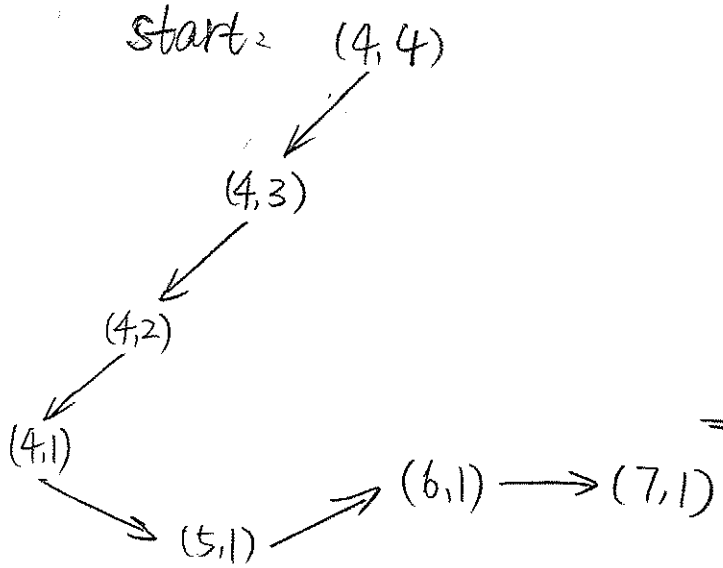
After picking the food up at (1,7), the agent now is looking for nearest granary.

BFS (under graph search)



⇒ find the granary, and since this is the only one granary found in the 4th level and assume each position is 1 from its neighbour, so that this is the nearest granary to food (1,7), thus the goal is achieved.

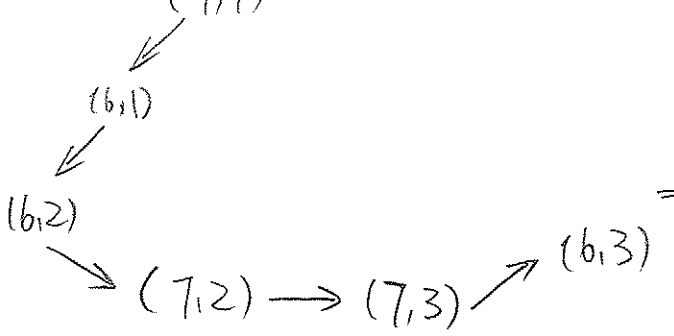
- DFS (under graph search)



⇒ find the food by DFS and pick it up

Then, start from (7,1) to look for the granary.

start: (7,1)



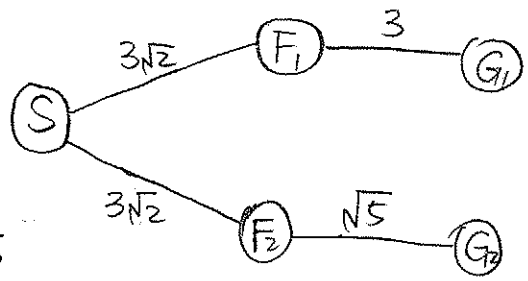
⇒ find the granary, and put the food down.

(Now, it's more convenient to use euclidean distance since A* is optimal)

Heuristic design: denote food in (1,7) and (7,1) as F_1 and F_2

denote the granary in (4,7) and (6,3) as G_1 and G_2

then:



$h(S) = 5$

$h(F_1) = 1$

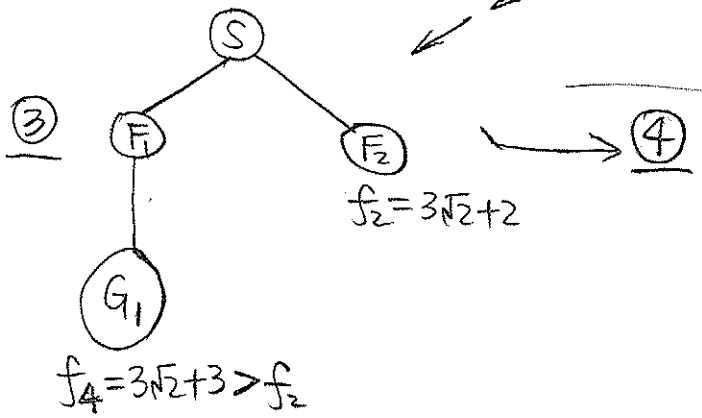
$h(F_2) = 2$

$h(G_1) = 0$

$h(G_2) = 0$

thus, this heuristic design is admissible.

A* search: ① S $f_1 = 5+0 = 5$



Optimal solution: $S \to F_2 \to G_2$

3. a.

$\forall t, [\exists x, [door(x) \wedge time(t) \wedge openat(x,t)] \rightarrow \exists y, [door(y) \wedge time(t) \wedge (x \neq y) \wedge \neg openat(y,t)]$

b. $\forall x,\ door(y) \wedge Walkthrough(agent, y) \rightarrow open(x)$

c. $\forall x, y,\ direction(x) \wedge direction(y) \wedge WalkToward(agent, x) \wedge OpenToward(door, y) \rightarrow (x = y)$

d. $\forall x, y,\ resource(x) \wedge Have(agent, x) \wedge location(y) \wedge sametype(x, y) \rightarrow store(x, y)$

e. $\forall x,\ resource(x) \leftrightarrow (x = food) \vee (x = gold)$

d. $location(G1) \wedge In(gold, G1) \wedge location(G2) \wedge In(gold, G2)$
$Storage(B1) \wedge store(gold, B1) \wedge Storage(B2) \wedge Store(gold, B2)$

f. $location(F1) \wedge In(food, F1) \wedge location(F2) \wedge In(food, F2)$
$storage(S1) \wedge Store(food, S1) \wedge storage(S2) \wedge store(food, S2)$

GMP.

→ in the next page.

proof: $\forall x,\ resource(x) \iff (x = food) \vee (x = gold)$

$Have(agent, gold) \wedge In(gold, G1) \wedge location(G1)$

$Storage(B1) \wedge store(gold, B1) \wedge Storage(B2) \wedge store(gold, B2)$

$\forall x, y,\ resource(x) \wedge Have(agent, x) \wedge location(y) \wedge Sametype(x, y) \Rightarrow store(x, y)$

$[storage(B1) \wedge store(gold, B1)] \vee [storage(B2) \wedge store(gold, B2)]$

thus, it should store the gold either at B1 or B2