Part 1

1.

Initial state: MonkeyAt(A) ^ MonkeyAt(Low) ^ BoxAt(B) ^ ¬holds()
Goal state: MonkeyAt(C) ^ MonkeyAt(High) ^ BoxAt(C) ^ holds()

2.

ACTION1 Go(x,y)
PRECOND: MonkeyAt(x) ^ MonkeyAt(Low) ^ (x!=y)
EFFECT: MonkeyAt(y) ^ MonkeyAt(Low) ^ ¬MonkeyAt(x)

ACTION2 Push(x,y)
PRECOND: MonkeyAt(x) ^ MonkeyAt(Low) ^ BoxAt(x) ^ (x!=y)
EFFECT: MonkeyAt(y) ^ MonkeyAt(Low) ^ BoxAT(y) ^ ¬MonkeyAt(x)

ACTION3 ClimbUp(x)
PRECOND: MonkeyAt(x) ^ MonkeyAt(Low) ^ BoxAt(x)
EFFECT: MonkeyAt(x) ^ MonkeyAt(High) ^ BoxAT(x)

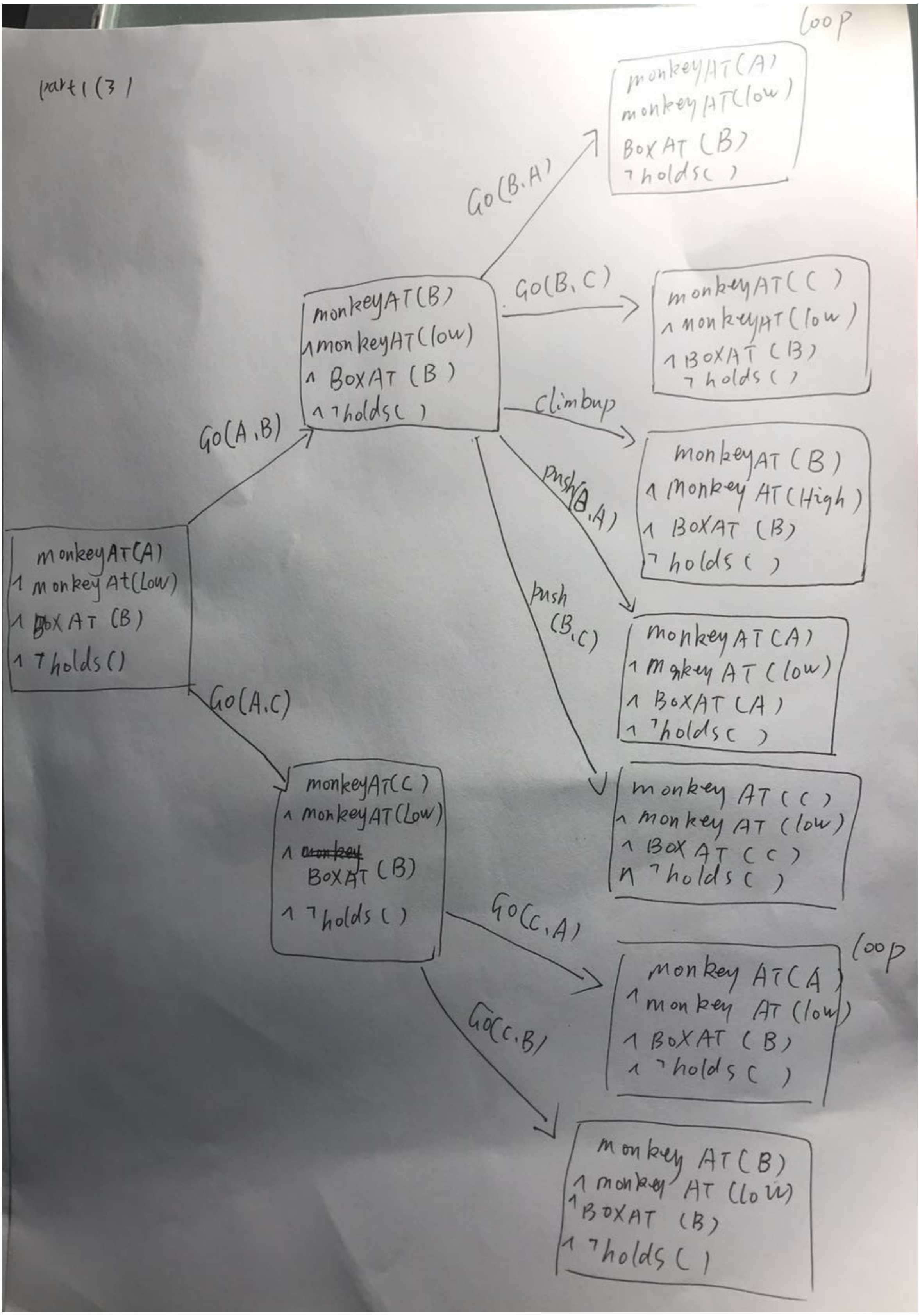
ACTION4 ClimbDown(x)
PRECOND: MonkeyAt(x) ^ MonkeyAt(High) ^ BoxAt(x)
EFFECT: MonkeyAt(x) ^ MonkeyAt(Low) ^ BoxAT(x)

ACTION5 Grasp()
PRECOND: MonkeyAt(C) ^ MonkeyAt(High) ^ BoxAt(C)
EFFECT: MonkeyAt(C) ^ MonkeyAt(High) ^ BoxAT(C) ^ holds()

ACTION6 Ungrasp()
PRECOND: MonkeyAt(C) ^ MonkeyAt(High) ^ BoxAt(C) ^ holds()
EFFECT: MonkeyAt(C) ^ MonkeyAt(High) ^ BoxAT(C) ^ ¬holds()

3.

part1(3)



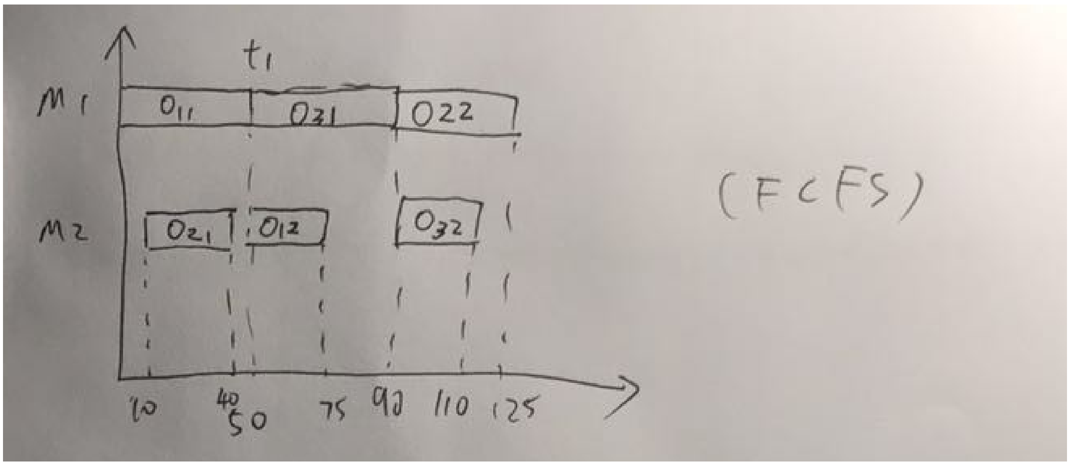
monkeyAT(A)
∧ monkeyAt(Low)
∧ BoxAT (B)
∧ ¬holds()
Go(A,B)
Go(A,C)
monkeyAT(B)
∧monkeyAT(low)
∧ BoxAT (B)
∧ ¬holds( )
Go(B,A)
loop
monkeyAT(A)
monkeyAT(low)
BoxAT (B)
¬holds( )
Go(B,C)
monkeyAT(C)
∧ monkeyAT(low)
∧ BoxAT (B)
¬holds( )
climbup
monkeyAT (B)
∧ Monkey AT(High)
∧ BoxAT (B)
¬holds( )
push(B,A)
monkeyAT(A)
∧ monkey AT (low)
∧ BoxAT (A)
∧ ¬holds( )
push (B,C)
monkey AT (C)
∧ monkey AT (low)
∧ Box AT (C)
∧ ¬holds( )
monkeyAT(C)
∧ monkeyAT(Low)
∧ ~~monkey~~ BoxAT (B)
∧ ¬holds( )
Go(C,A)
loop
monkey AT(A)
∧ monkey AT (low)
∧ BoxAT (B)
∧ ¬holds( )
Go(C,B)
monkey AT(B)
∧ monkey AT (low)
∧ BoxAT (B)
∧ ¬holds( )

4.

Initial state: MonkeyAt(A) ^ MonkeyAt(Low)^ BoxAt(B) ^ ¬holds()
Action 1: Go(A,B)
State 1: MonkeyAt(B) ^ MonkeyAt(Low) ^ BoxAt(B) ^ ¬holds()
Action 2: Push(B,C)
State 2: MonkeyAt(C) ^ MonkeyAt(Low) ^ BoxAt(C) ^ ¬holds()
Action 3: ClimbUp()
State 3: MonkeyAt(C) ^ MonkeyAt(High) ^ BoxAt(C) ^ ¬holds()
Action 4: Grasp()
State 4 (goal state): MonkeyAt(C) ^ MonkeyAt(High) ^ BoxAt(C) ^ holds()

part2

1.



t1 = 0, t2 = 10, t3 = 50, t4 = 50, t5 = 90, t6 = 90

2.

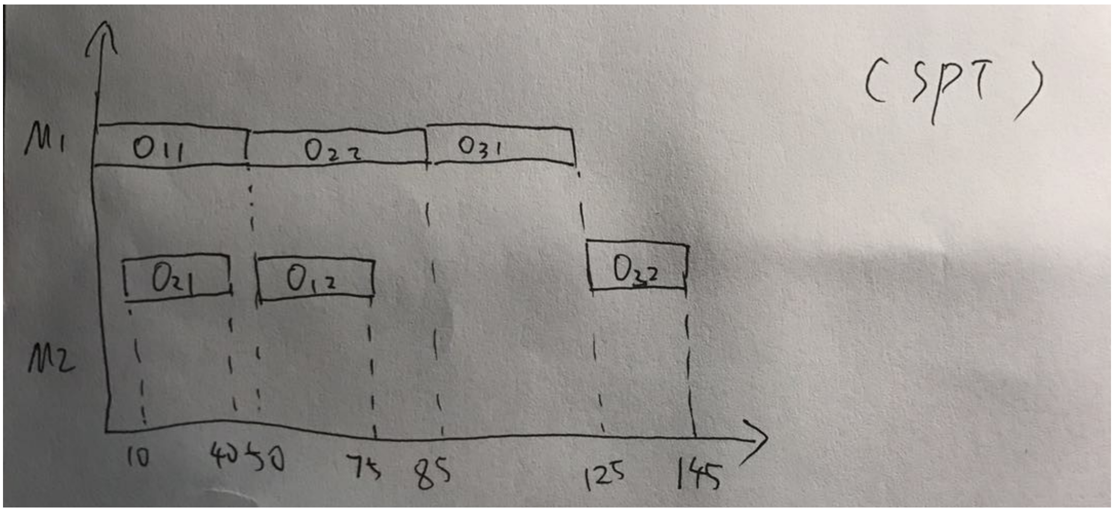
job1 finish time: 75

job2 finish time: 125

job3 finish time: 110

makespan is maximal time to complete three jobs which is 125.

3.



Step1:

Partial solution: P(O11,M1,0)

earliestIdleTime(M1)=50, earliestIdleTime(M2)=0

~~earliestReadyTime(O21)=10~~,earliestReadyTime(O12)=50

earliestReadyTime(O21)=10, earliestReadyTime(O22)=∞

earliestReadyTime(O31)=20, earliestReadyTime(O32)=∞

Step2:

Partial solution: P(O11,M1,0)-->P(O21,M2,10)

earliestIdleTime(M1)=50, earliestIdleTime(M2)=40

~~earliestReadyTime(O11)=0~~, earliestReadyTime(O12)=50

~~earliestReadyTime(O21)=10~~, earliestReadyTime(O22)=40

earliestReadyTime(O31)=20, earliestReadyTime(O32)=∞

Step3:

Partial solution: P(O11,M1,0)-->P(O21,M2,10)-->P(O12,M2,50)

earliestIdleTime(M1)=50, earliestIdleTime(M2)=75

~~earliestReadyTime(O11)=0, earliestReadyTime(O12)=50~~

~~earliestReadyTime(O21)=10~~, earliestReadyTime(O22)=50

earliestReadyTime(O31)=20, earliestReadyTime(O32)=∞

Final solution: P1(O11,M1,0)-> P2(O21,M2,10)-> P3(O12,M2,50)-> P4(O22,M1,50)-> P5(O31,M1,85)-> P6(O32,M2,125)

4.

job1 finish time: 75

job2 finish time: 85

job3 finish time: 145

makespan is maximal time to complete three jobs which is 145.

5.

In this case, the makespan of FCFS is faster than that of SPT, but it does not mean FCFS is better all the time. It depends on different cases.

Advantage of FCFS is that simplicity and inherent fairness.

SPT is that results in better customer service levels.

Part3

1.

R1(1,2,3,5,1)

R2(1,6,8,4,1)

R3(1,7,9,10,1)

2.

R1: 1+1+1+2.24= 5.24

R2: 1.41+1.41+1.41+3.16=7.39

R3: 2.23+3.16+2+5.39=12.78

total length: R1+R2+R3=25.41

3.

Function set:

1. multiply 2. divide 3. add 4. subtract 5. min

because those function set speed up the search process. Finding the nearest neighbour by min which need select smallest distance between two nodes in Euclidean matrix.

Others is basic function sign to calculate result.

Terminal set:

1. distance to nearest neigbour(distance of node)
2. graph(demand of depot) which is distance of depot
3. vehicle(capacity of truck)

we need ensure capacity of vehicle. And find best node by distance of nearest node. At same time, graph which combined with distance of nearest node which help us to find shortest way.

Fitness function:

Euclidean distance

fitness function contributes to compare and evaluate the routing to find shortest way to goal. I choose Euclidean distance algorithm which help us figure out best solution in graph.