Problem #1: (State Space Representation) Consider the problem of transporting N people across a river, where each person has a certain weight. Everyone starts off on the right-hand side of the river and is to be transported to the left-hand side. Suppose also that only one boat exists, which is able to support a maximum weight of WB.

A. Design a state space representation for this problem. Specify clearly the meaning of each component of the state representation.

States = (R,L,B)

R = 오른쪽에 있는 사람들의 집합

L = 왼쪽에 있는 사람들의 집합

B = 보트에 탑승해 있는 사람들의 집합

boat = 보트의 위치

boat = right (보트가 오른쪽에 위치)

boat = left (보트가 왼쪽에 위치)

Initial state = (R=[p1,p2,...pN] , L=[], B=[])

Actions = move right , move left (r->l 이동)

Transition = update (R,L,B)

Goal test = is (R, L, B) = (R=[], L=[p1,p2,...,pN] , B=[])

B. Design a complete set of operators in this domain based on the above representation. Note that these operators should also work for transporting any allowed subgroup of people in either direction. That is, they should not be overly specific to the actual problem we want to solve, but instead should allow one to traverse the search space in a general manner.

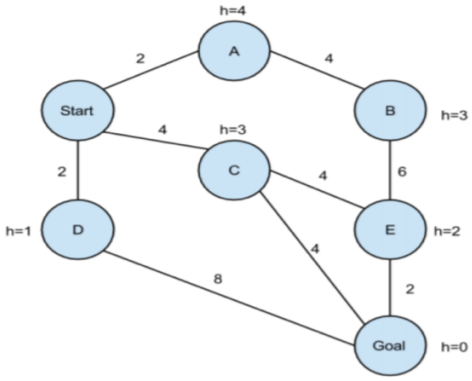
1. 탑승 연산자 GetOn()
   조건 – 보트의 위치 = 탑승 위치 / 탑승자 총 무게 <= WB
   EX) (R=[p1,p2,p3,...,pN], L=[], B=[]) → (R=[p3,p4,...,pN], L=[], B=[p1,p2]) boat=right

2. 하차 연산자 GetOff()
   조건 – 보트의 위치 = 하차 위치
   EX) (R=[p3,p4,...,pN], L=[], B=[p1,p2]) → (R=[p3,p4,...,pN], L=[P1,P2], B=[]) boat=left

3. 건너기 연산자 Cross()
   조건 – 이동하는 사람 총 무게 <= WB, 이용자의 위치 = 보트의 위치
   EX) (R=[p1,p2,p3,...,pN], L=[], B=[]) boat=right → (R=[p3,p4,...,pN], L=[P1,P2], B=[]) boat=left

Problem #2

Given the graph to the right, write down the order in which the states are visited by the following search algorithms. If a state is visited more than once, write it each time. Ties (e.g., which child to first explore in depth-first search) should be resolved according to alphabetic order (i.e. prefer A before Z). Remember to include the start and goal states in your answer. Treat the goal state as G when you break ties. Assume that algorithms execute the goal check when nodes are visited, not when their parent is expanded to create them as children.



A. Iterative deepening depth first search (ID)

깊이 제한 0

Start

깊이 제한 1

Start → A → C → D

깊이 제한 2

Start → A → B → C → E → Goal (탐색 종료!)

B. A* search, where f(n)=g(n)+h(n)

Node[f(n)]	Clsoed list
A[6]	start
C[7]	
D[3]	

Node[f(n)]	Clsoed list
A[6]	Start
C[7]	D
Goal[10]	

Node[f(n)]	Clsoed list
C[7]	start
Goal[10]	D
B[9]	A

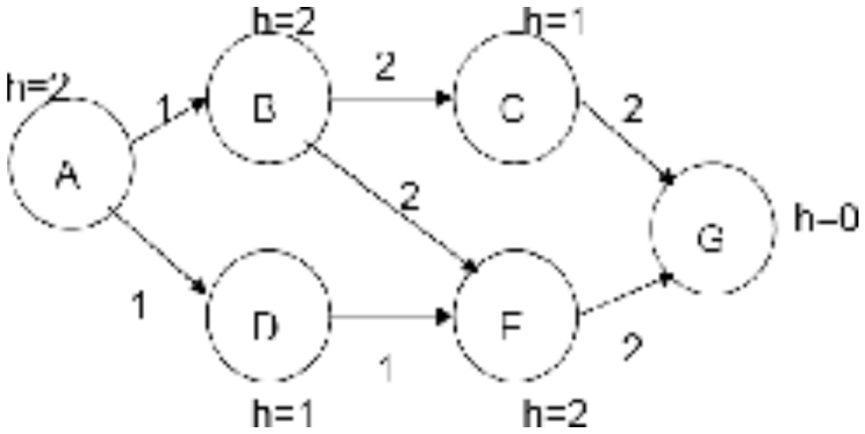
Node[f(n)]	Clsoed list
~~Goal[10]~~	start
B[9]	D
E[10]	A
Goal[8]	C

Node[f(n)]	Clsoed list
B[9]	start
E[10]	D
	A
	C
	Goal

Start → D → A → C → Goal (탐색 종료!)

Problem #3

The figure below shows a problem-space graph, where A is the initial state and G denotes the goal. Edges are labeled with their true cost. We have a heuristic function, f(), written in the standard form: f(n)=g(n)+h(n) where g(n) is the cost to get from A to n and h(n) is an estimate of the remaining distance to G. .



A. In the graph below, is f() admissible? Why or why not?

Admissible 조건 : h(n) <= h*(n)

Node A : (h(A)=2) < (h*(A)=4) → admissible
Node B : (h(B)=2) < (h*(B)=4) → admissible
Node C : (h(C)=1) < (h*(C)=2) → admissible
Node D : (h(D)=1) < (h*(D)=3) → admissible
Node E : (h(E)=2) = (h*(E)=2) → admissible
Node G : (h(G)=0) = (h*(G)=0) → admissible

→ 모든 노드가 admissible 하므로 f()은 admissible 하다.

B. Is f() monotonic? Why or why not?

Monotonic 조건 : h(n) ≤ c(n,n')+h(n')

h(A)=2,h(B)=2, h(C)=1, h(D)=1, h(E)=2, h(G)=0

Node A : 2 < (C(A,B)+h(B)=1+2=3), 2 = (C(A,D)+h(D)=1+1=2) → monotonic

Node B: 2 < (C(B,C)+h(C)=2+1=3), 2 < (C(B,E)+h(E)=2+2=4) → monotonic

Node C : 1 < (C(C,G)+h(G)=2+0=2) → monotonic

Node D : 1 < (C(D,E)+h(E)=1+2=3) → monotonic

Node E : 2 = (C(E,G)+h(G)=2+0=2 → monotonic

→ 모든 노드가 monotonic 하므로 f()은 monotonic 하다.