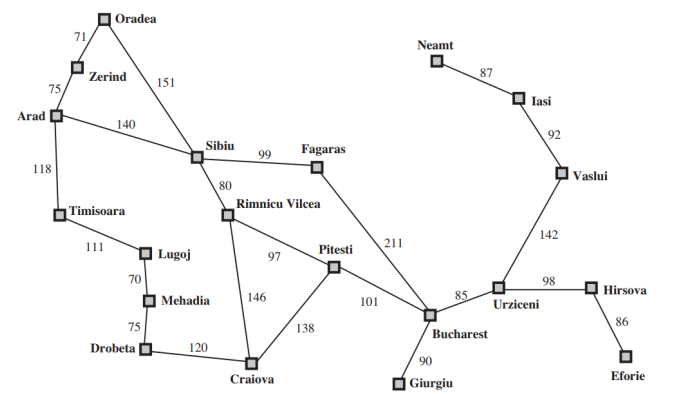
**Question 1. Suppose two friends live in different cities on a map, such as the Romania map shown in below Figure**

****

**On every turn, we can simultaneously move each friend to a neighboring city on the map. The amount of time needed to move from city i to neighbor j is equal to the road distance d(i, j) between the cities, but on each turn the friend that arrives first must wait until the other one arrives (and calls the first on his/her cell phone) before the next turn can begin. We want the two friends to meet as quickly as possible.**

**a. Write a detailed formulation for this search problem.**

Denote the cities as *C*1​,*C*2​,…,*Cn*​. We can represent the state of the problem as the pair of current cities for the two friends, where i and j are the current cities of the two friends. The goal is to find a sequence of states that minimizes the total time for the two friends to meet.

The initial state is (*C*start1​,*C*start2​), where *C*start1​ and *C*start2​ are the initial cities of the two friends.

The goal state is where *C*goal​ is the city where the two friends meet.

We need.

**State Space**: The state space consists of all possible combinations of cities where each friend is located.

**Initial State**: The initial state is the current locations of the two friends. *S*=(*Ci*​,*Cj*​)

**Goal State**: The goal state is achieved when both friends are at the same city. (*C*goal​,*C*goal​)

**Transition Model**: The state transition is determined by the simultaneous movement of Friend A and Friend B to neighboring cities.

**Cost Function**: The cost of moving from one city to another is equal to the road distance between those cities.*d(i, j)*

* Let *S* be the set of all possible states.
* Let *A* and *B* represent the locations of Friend A and Friend B, respectively.
* The initial state ​ is the starting locations of both friends.
* The goal state ​ is the state where both friends have the same location.
* The action *A->*​*B* represents Friend A moving to a neighboring city.
* The action *B->*​*A* represents Friend B moving to a neighboring city.
* The cost function *C*(*s*,*a*) is the road distance between the current location and the destination for each friend.

**b. Let D(i, j) be the straight-line distance between cities i and j. Which of the following heuristic functions are admissible?**

**• D(i, j)**

**• 2 · D(i, j)**

**• D(i, j)/2**

Among these, D(i,j), D(i,j)/2 is admissible because D(i,j) calculates the straight-line distance between the current state and the goal state, which is a lower bound on the actual road distance. Furthermore, D(i,j)/2 is less or equal to D(i,j)

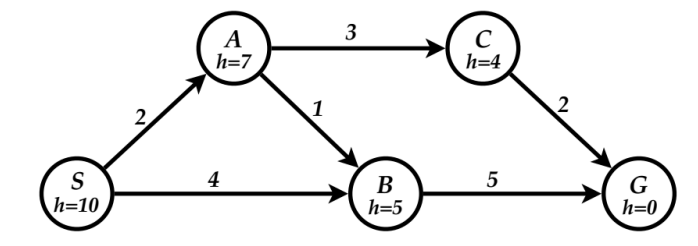
**c. Are there completely connected maps for which no solution exists?**

In a completely connected map, where every city is directly connected to every other city, the solution always exists. As long as there is a road connecting every pair of cities, the friends can eventually reach each other.

**d. Are there any maps in which all solutions require one friend to visit the same city twice?**

If there are cities with loops or multiple paths leading to the same city, it's possible that a solution requires one friend to visit the same city twice. This situation may occur if the road distances are such that it's more efficient for one friend to backtrack and meet the other friend at a later point.

**Question 2. Consider the following graph, in which S and G are the initial and goal states, respectively. The heuristic values are shown under the vertices’ names, while path costs are shown on every edges.**

****

**For each of the search strategies listed below,**

**(a) list, in order, the states expanded,**

**(b) list, in order, the states included in the found path, and**

**(c) show the final content of the frontier (recall that a state is expanded when it is removed from the frontier)**

**When all else is equal, nodes should be expanded in alphabetical order.**

**Uniform-cost search (UCS)**

List of expanded nodes: [S, A, B, C, G]

Path found: S -> A -> C -> G

Frontier = { }

**Depth-first search (DFS) (Avoid loops by remembering nodes on the current path).**

List of expanded nodes: [S, A, B, G]

Path found: S, A, B, G

Or

List of expanded nodes: [S, B, G]

Path found: S -> B -> G

**Iterative deepening search (IDS)**

List of expanded nodes for each limit: {S} {S, A, B} {S, A, B, C, G}

Path found: S -> B -> G

**Greedy best first search (GBFS)**

List of expanded nodes: { S, B, G }

Path found: S -> B -> G

Frontier: { }

**A\* search**

List of expanded nodes: { S, A, B, C, G }

Path found: S ->A -> B -> G

Frontier: { C (f=9) }

Or

List of expanded nodes: { S, B, G }

Path found: S -> B -> G

Frontier: { } (due to tie break of A (f=9) and B (f=9))

From S to G: The actual cost is 7 (S -> A -> C -> G), and the heuristic value is 10. The heuristic is not admissible for S.

From A to G: The actual cost is 5 (A -> C -> G), and the heuristic value is 7. The heuristic is not admissible for A.

From B to G: The actual cost is 5 (B -> G), and the heuristic value is 5. The heuristic is admissible for B.

From C to G: The actual cost is 2 (C -> G), and the heuristic value is 4. The heuristic is not admissible for C.

From G to G: The actual cost is 0 (already at the goal), and the heuristic value is 0. The heuristic is admissible for G.

Since the heuristic values are not less than or equal to the actual costs for all states, the given heuristic is not admissible.

Mathematically: *h*(*n*)≤*c*(*n*,*a*,*n*′)+*h*(*n*′) for every state *n*, action *a*, and successor state *n*′.

S to A: h(S) = 10, c(S, S->A, A) = 2, h(A) = 7. It holds (10 is not <= 2 + 7) is not consistent

S to B: h(S) = 10, c(S, S->B, B) = 4, h(B) = 5. It holds (10 is not <= 4 + 5) is not consistent

A to B: h(A) = 7, c(A, A->B, B) = 1, h(B) = 5. It holds (7 is not <= 1 + 5) is not consistent

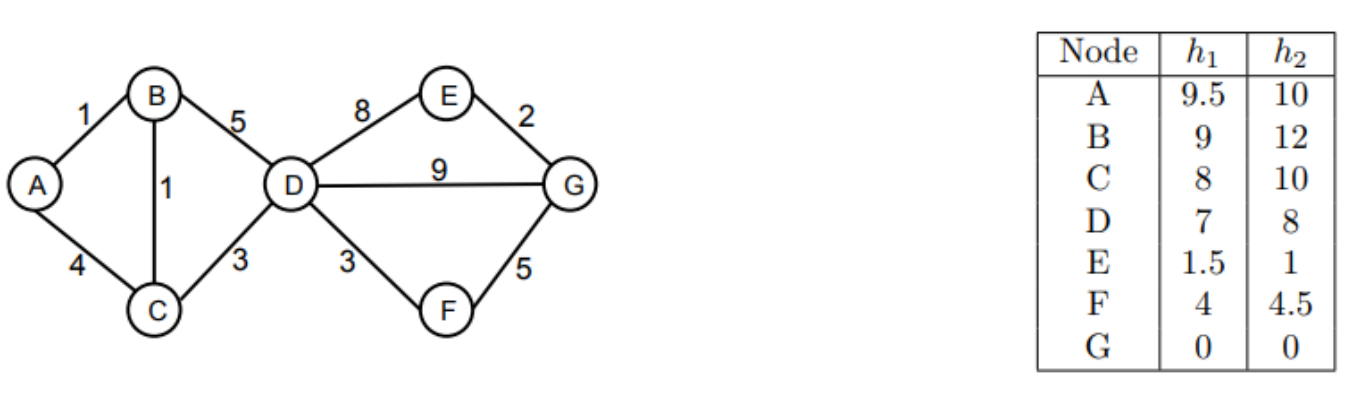
A to C: h(A) = 7, c(A, A->C, C) = 3, h(C) = 4. It holds (7 <= 3 + 4) is consistent

B to G: h(B) = 5, c(B, B->G, G) = 5, h(G) = 0. It holds (5 <= 5 + 0) is consistent

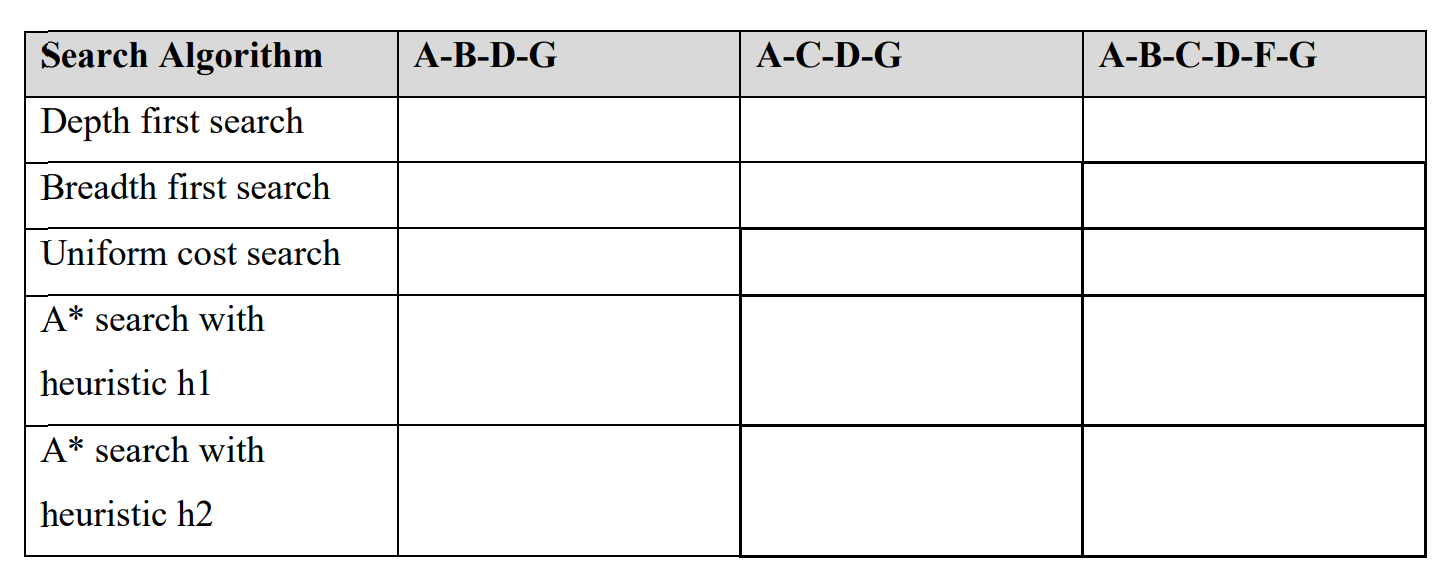
C to G: h(C) = 4, c(C, C->G, G) = 2, h(G) = 0. It holds (4 is not <= 2 + 0) is not consistent

Since the heuristic values are not less than or equal to the sum of cost and next state heuristic, the given heuristic is not consistent.

**Question 3 Consider the state space graph shown above. A is the start state and G is the goal state. The costs for each edge are shown on the graph. Each edge can be traversed in both directions. Note that the heuristic h1 is consistent but the heuristic h2 is not consistent.**

****

**1. Possible paths returned For each of the following graph search strategies (do not answer for tree search), mark which, if any, of the listed paths it could return. Note that for some search strategies the 4 specific path returned might depend on tie-breaking behavior. In any such cases, make sure to mark all paths that could be returned under some tie-breaking scheme**

****

X

X

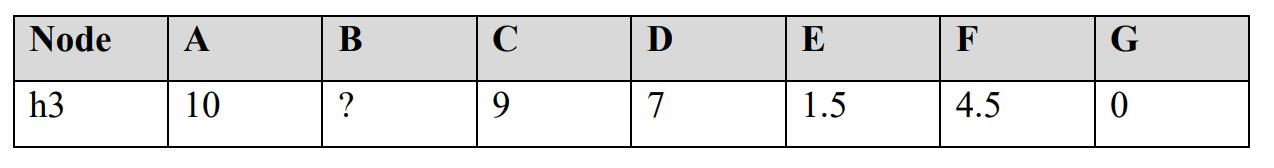
X

X

X

X

**2. Heuristic function properties Suppose you are completing the new heuristic function h3 shown below. All the values are fixed except h3(B).**

****

**For each of the following conditions, write the set of values that are possible for h3(B). For example, to denote all non-negative numbers, write [0, ∞], to denote the empty set, write ∅, and so on.**

**• What values of h3(B) make h3 admissible?**

From A to G: The actual cost is 13 (B -> D -> F -> G), and the heuristic must be <=13 to be admissible

**• What values of h3(B) make h3 consistent?**

B to A: c(B, B->A, A) = 1, h(A) = 10. It holds (h(B) <= 10 + 1)

B to C: c(B, B->C, C) = 1, h(C) = 9. It holds (h(B) <= 1 + 9)

B to D: c(B, B->D, D) = 5, h(D) = 7. It holds (h(B) <= 7 + 5)

h(B) must be <= 10 to be consistent

**• What values of h3(B) will cause A\* graph search to expand node A, then node C, then node B, then node D in order?**

A expand to B and C  
A to C: c(C, C->A, A) = 4, h(C) = 9. It holds (h(B) <= 10 + 1)

For A expand to C -> A to B must be > 13 -> h(B) > 13 – 1 =12

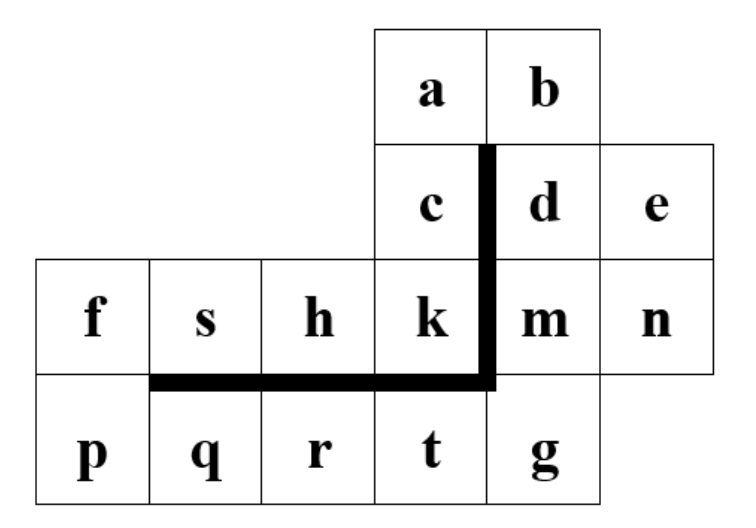
C expand to B and D

C to D: c(C, C->D, D) = 7, h(D) = 7. It holds (h(B) <= 10 + 1)

For C expand to B -> C to B must be < 14 -> h(B) < 14 -5 = 9

Because it requires h(B) > 12 and h(B) < 9. Therefore, h(B) = ∅

**Question 4 Given the following maze. The bold line is wall which you cannot get pass.**

****

**Searching for the path from s to g. Give the answer with the format: <b1, b2,..,bn> with bi is explored node**

**a. BFS**

Expanded cell at step 1: queue(<s,f>, <s,h>)

Expanded cell at step 2: queue(<s,h>, <s,f,p>)

Expanded cell at step 2: queue(<s,f,p>, <s,h,k>)

Expanded cell at step 3: queue(<s,h,k>, <s,f,p,q>)

Expanded cell at step 3: queue(<s,f,p,q>, <s,h,k,c>)

Expanded cell at step 4: queue(<s,h,k,c>, <s,f,p,q,r>)

Expanded cell at step 4: queue(<s,f,p,q,r>, <s,h,k,c,a>)

Expanded cell at step 5: queue(<s,h,k,c,a>, <s,f,p,q,r,t>)

Expanded cell at step 5: queue(<s,f,p,q,r,t>, <s,h,k,c,a,b>)

Expanded cell at step 6: queue(<s,h,k,c,a>, <s,f,p,q,r,t>)

Expanded cell at step 6: queue(<s,f,p,q,r,t>, <s,h,k,c,a,b>)

Expanded cell at step 7: queue(<s,h,k,c,a,b>, <s,f,p,q,r,t,g>)

Expanded cell at step 7: queue(<s,f,p,q,r,t,g>, <s,h,k,c,a,b,d>)

Expanded cell at step 8: queue(<s,h,k,c,a,b,d>)

Goal reach: so the optimal path is s->f->p->q->r->t->g

**b. DFS with a cycle check to prevent loop. The order of the operators is up, left, right, then down**

Expanded cell at step 1: stack (<s,f>) (left)

Expanded cell at step 2: stack (<s,f,p>) (down)

Expanded cell at step 3: stack (<s,f,p,q>) (right)

Expanded cell at step 4: stack (<s,f,p,q,r>) (right)

Expanded cell at step 5: stack (<s,f,p,q,r,t>) (right)

Expanded cell at step 6: stack (<s,f,p,q,r,t,g>) (right)

Goal reach: so the optimal path is s->f->p->q->r->t->g

**c. Greedy best first search with Manhattan distances. The Manhattan distance between two points is the distance in the x-direction plus the distance in the y-direction. It corresponds to the distance traveled along city streets arranged in a grid. Ex: h(k) = 2, h(s) = 4, h(g) = 0**

Expanded cell at step 1: queue(<s,h>, h(h) = 3)

Expanded cell at step 2: queue(<s,h,k>, h(k) = 2)

Expanded cell at step 3: queue(<s,h,k,c>, h(c) = 3)

Expanded cell at step 4: queue(<s,h,k,c,a>, h(a) = 4)

Expanded cell at step 5: queue(<s,h,k,c,b>, h(b) = 3)

Expanded cell at step 6: queue(<s,h,k,c,b,d>, h(d) = 2)

Expanded cell at step 7: queue(<s,h,k,c,b,d,m>, h(m) = 1)

Expanded cell at step 8: queue(<s,h,k,c,b,d,m,g>, h(g) = 0)

Goal reach: so the optimal path is s->h->k->c->a->b->d->m->g

**d. A\* with the above heuristic**

Expanded cell at step 1: queue(<s,h>, g+h(h) = 4)

Expanded cell at step 2: queue(<s,h,k>, g+h(k) = 4)

Expanded cell at step 3: queue(<s,h,k,c>, g+h(c) = 6)

Expanded cell at step 4: queue(<s,h,k,c,a>, g+h(a) = 8)

Expanded cell at step 4: queue(<s,f>, g+h(f) = 6; <s,h,k,c,a>, g+h(a) = 8)

Expanded cell at step 5: queue(<s,f,p>, g+h(p) = 6; <s,h,k,c,a>, g+h(a) = 8)

Expanded cell at step 6: queue(<s,f,p,q>, g+h(q) = 6; <s,h,k,c,a>, g+h(a) = 8)

Expanded cell at step 7: queue(<s,f,p,q,r>, g+h(r) = 6; <s,h,k,c,a>, g+h(a) = 8)

Expanded cell at step 8: queue(<s,f,p,q,r,t>, g+h(t) = 6; <s,h,k,c,a>, g+h(a) = 8)

Expanded cell at step 9: queue(<s,f,p,q,r,t,g>, g+h(g) = 6; <s,h,k,c,a>, g+h(a) = 8)

Goal reach: so the optimal path is s->f->p->q->r->t->g