**Artificial Intelligence (CZ3005) Lab 1**

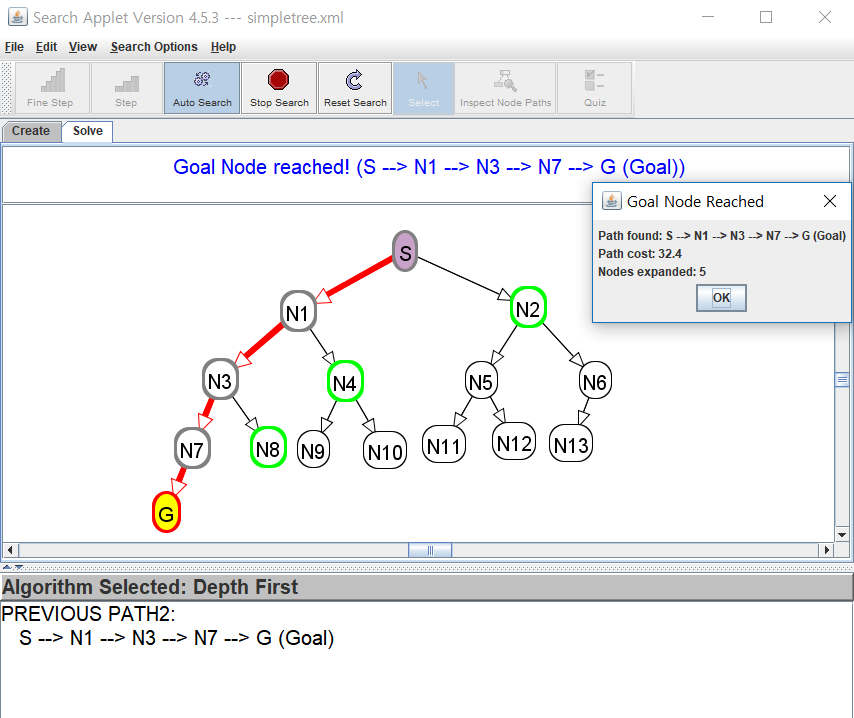
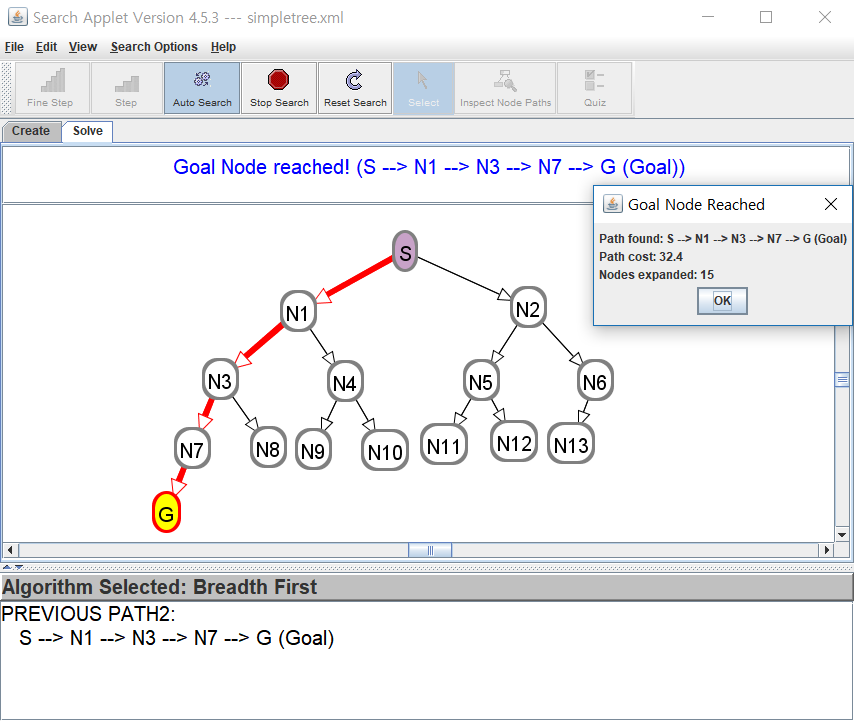
Name : LEE SUYEON

ID : N1801612A

Group : TS2

**1. Question One [20 marks]**

**(a) Give a graph where depth-first search (DFS) is much more efficient (expands fewer nodes) than breadth-first search (BFS).**

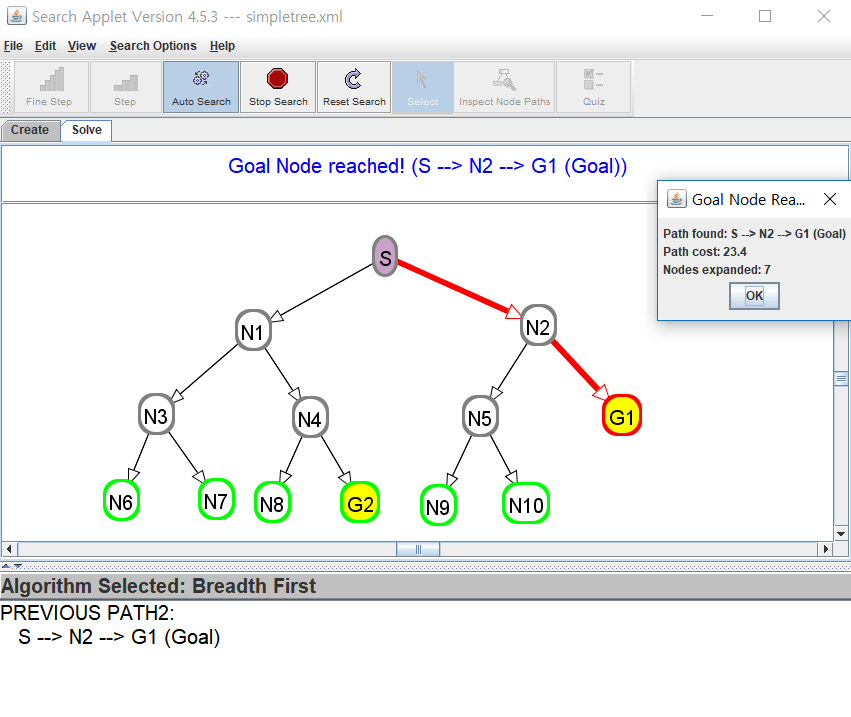
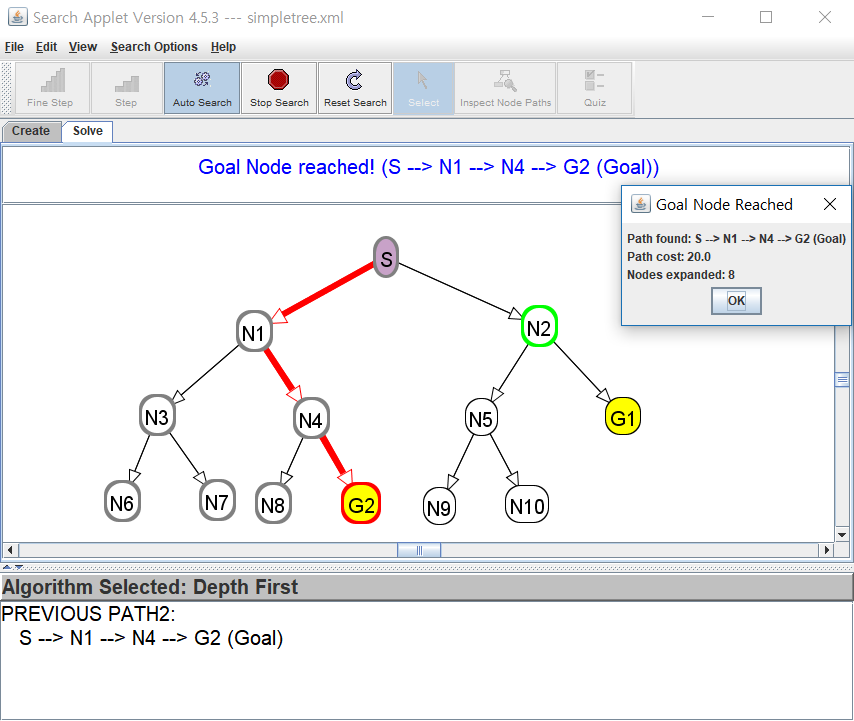
 

(left : DFS / right : BFS)

In this case, DFS is much better since the only goal is located at greatest depth. We have to examine all nodes if we use BFS.

* Path found : S-N1-N3-N7-G
* DFS) node expanded : S-N1-N3-N7-G (5)
* BFS) node expanded : S-N1-N2-N3-N4-N5-N6-N7-N8-N9-N10-N11-N12-N13-G (15)

**(b) Give a graph where BFS is much better than DFS.**

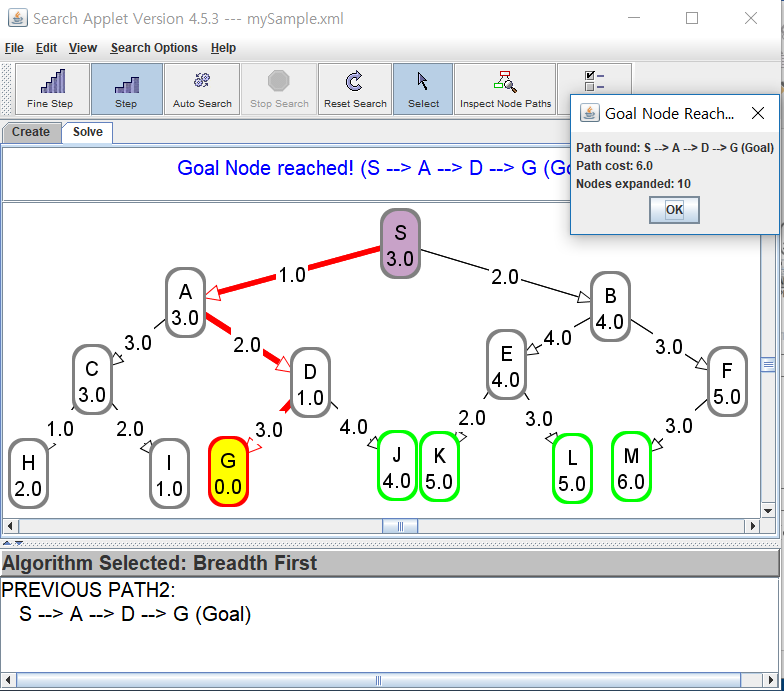
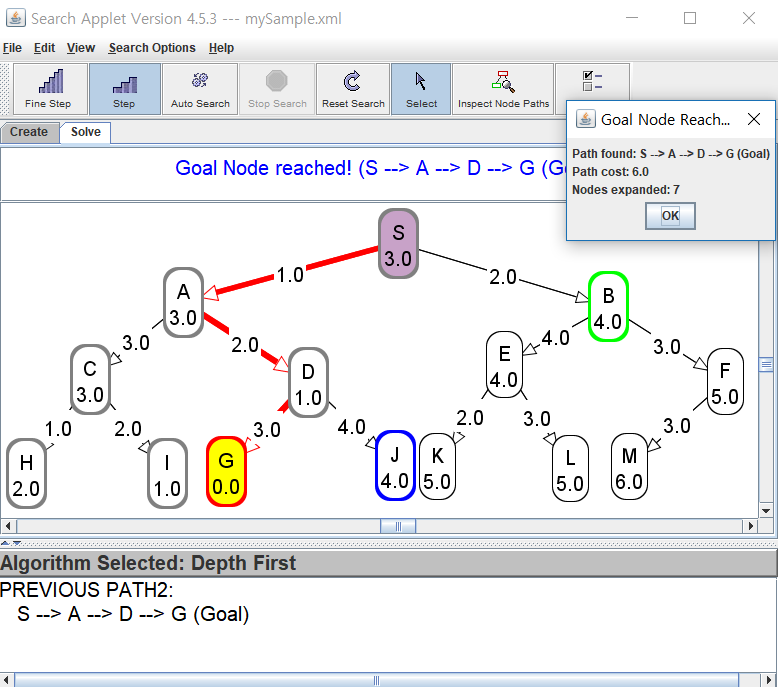


(left : DFS / right : BFS)

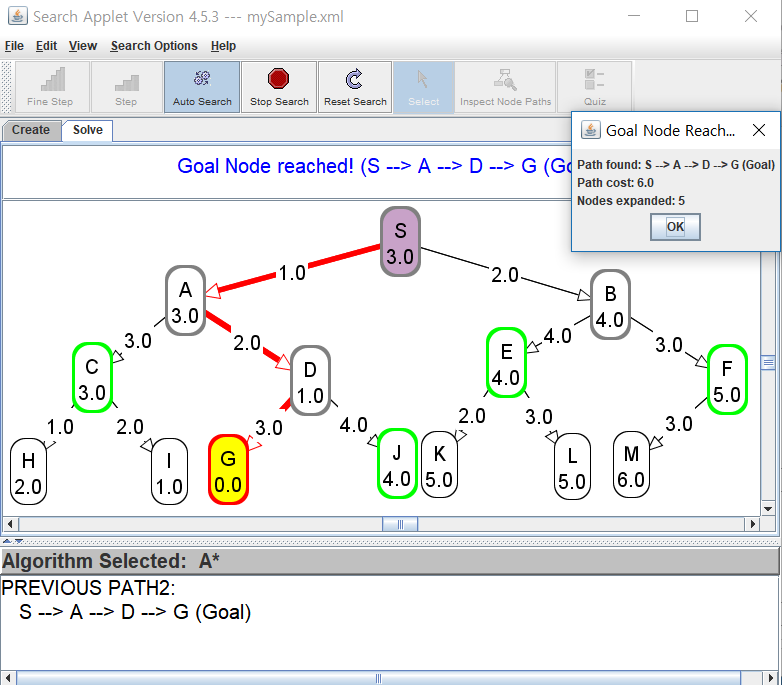
In this case, BFS is much better since one of goal(G1) is located in more shallow depth than another one(G2). If we use DFS, we might find goal G2 first rather than goal G1, which is not optimal. Also expanded nodes are fewer in BFS than DFS.

* Path found : S-N1-N4-G2 (DFS) / S-N2-G1 (BFS)
* DFS) node expanded : S-N1-N3-N6-N7-N4-N8-G2 (8)
* BFS) node expanded : S-N1-N2-N3-N4-N5-G1 (7)

**(c) Give a graph where A\* search is more efficient than either DFS or BFS.**



(left : DFS / right : BFS)

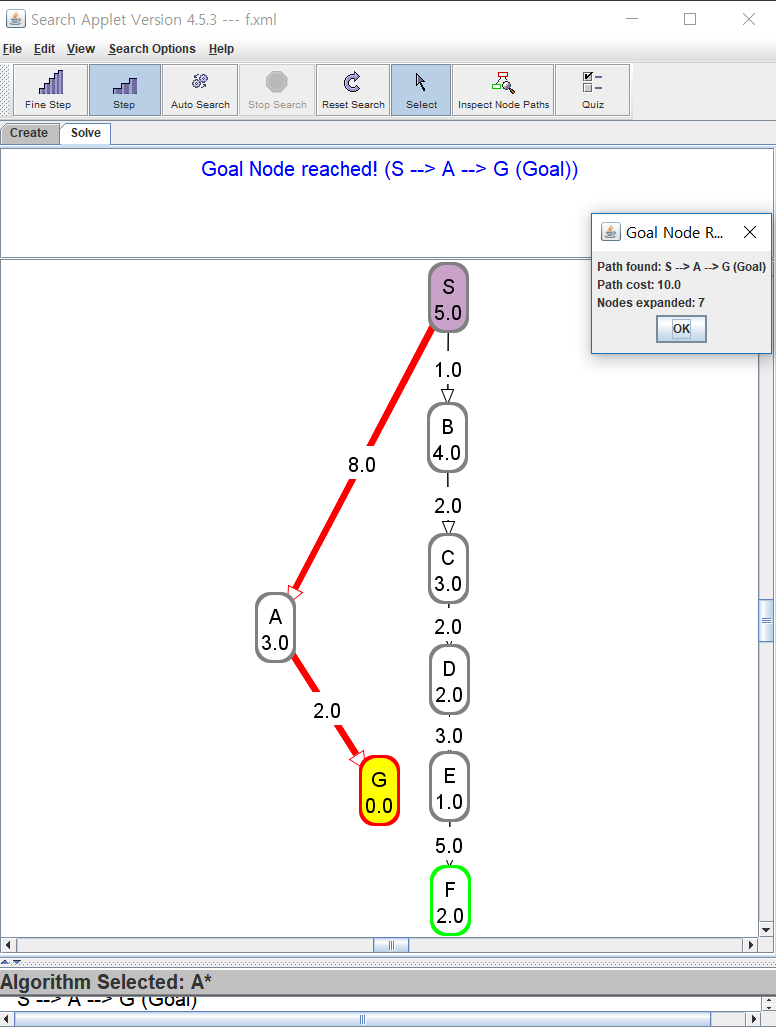


(A\*)

In this case, A\* search is more efficient (expands fewer nodes) rather than both DFS and BFS.

* Path found : S-A-D-G / Cost : 1+2+3 = 6
* DFS) node expanded : S-A-C-H-I-D-G (7)
* BFS) node expanded : S-A-B-C-D-E-F-H-I-G (10)
* A\*) node expanded : S-A-D-B-G (5)

**(d) Give a graph where DFS and BFS are both more efficient than A\* search.**



(A\*)

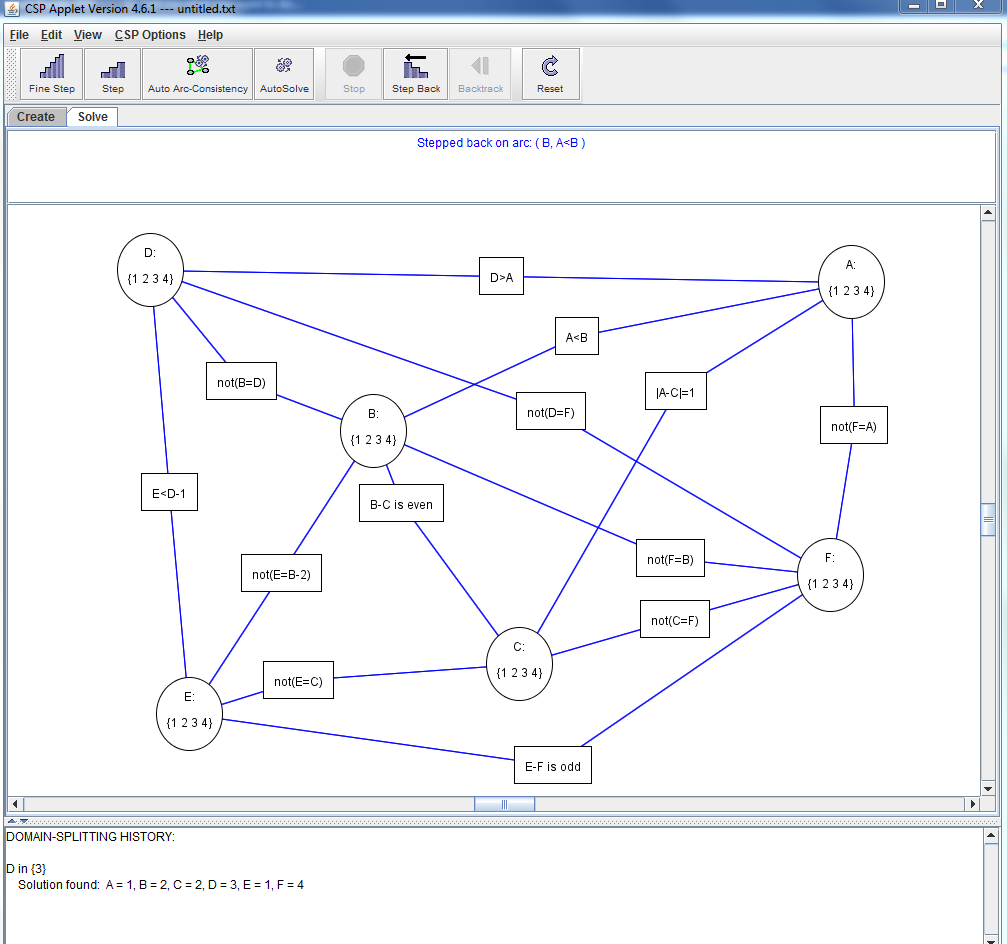
If we decide heuristic as a straight distance (Euclidian distance) and if tree graph is like given above, DFS and BFS are both more efficient than A\* search. When A\* is not efficient, it means that it has bad heuristic. So in these case, we should decide heuristic depends on other criteria.

* Path found : S-A-G / Cost : 8 + 2 = 10
* DFS) node expanded: S-A-G (3)
* BFS) node expanded: S-A-B-G (4)
* A\*) node expanded: S-B-C-D-E-A-G (7)

(next page continue)

**2. Question Two [20 marks]**

**(a) Draw a graph in AIspace.org as a CSP problem (constraint graph).**



**(b) For the first 5 instances of arc consistency, show which elements of a domain are deleted at each step, and which arc is responsible for removing the element.**

- 1 removed from the domain of B because of arc ( B, A<B )

- 4 removed from the domain of A because of arc ( A, A<B )

- Arc ( C, |A-C|=1 ) is consistent

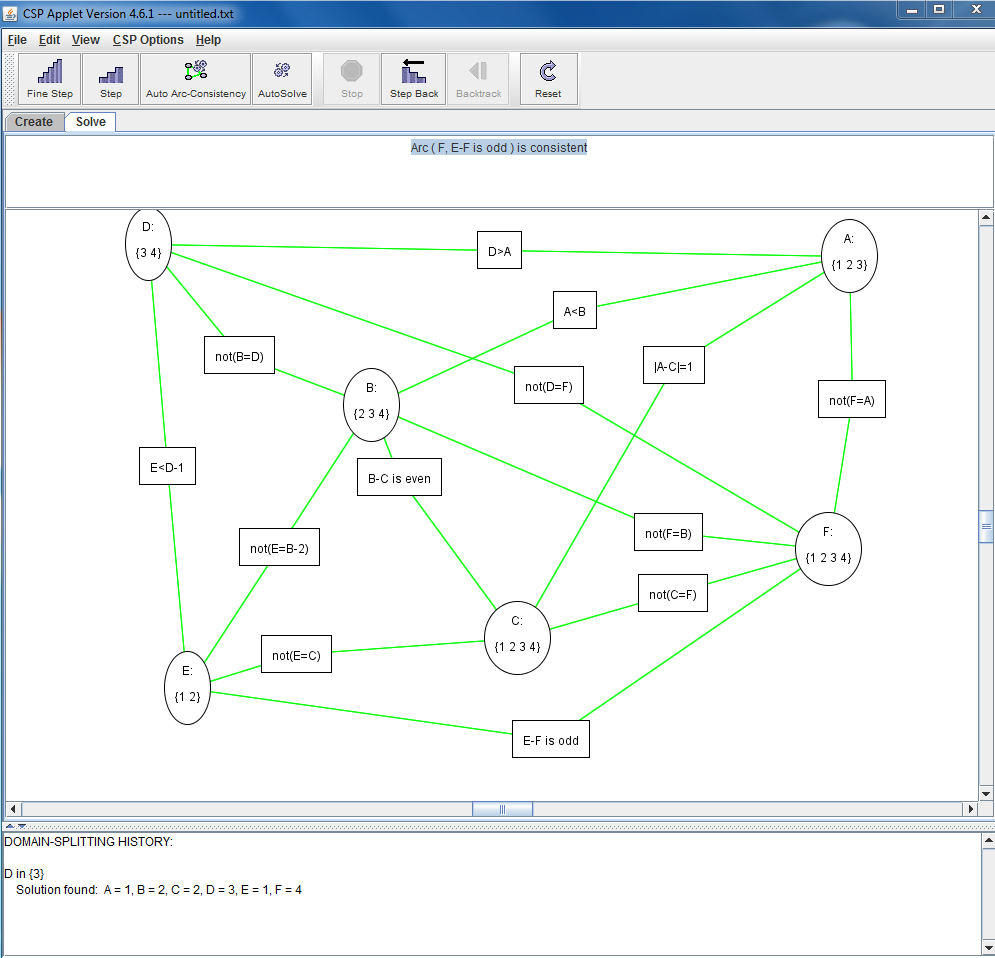
- Arc ( A, |A-C|=1 ) is consistent

- Arc ( C, B-C is even ) is consistent

//from this line, it is rest instances of arc consistency

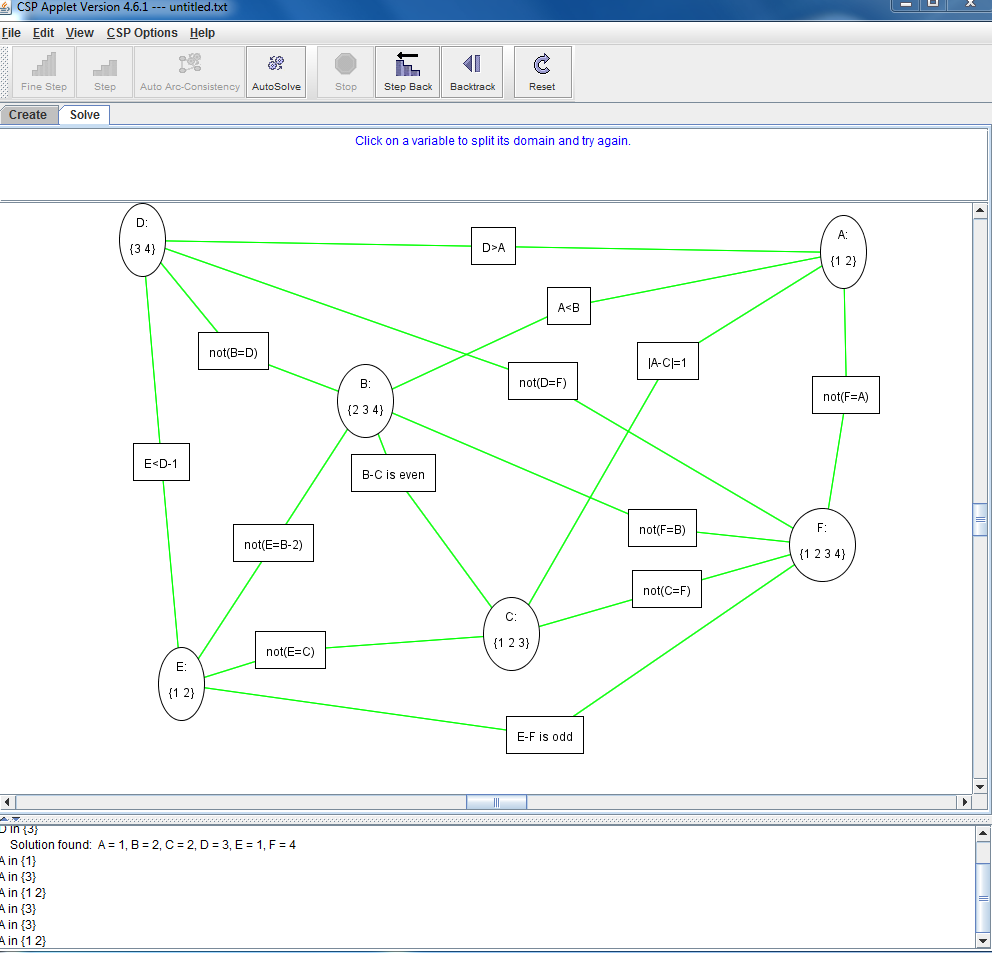
* Arc ( B, B-C is even ) is consistent
* Arc ( D, not(B=D) ) is consistent
* Arc ( B, not(B=D) ) is consistent
* Arc ( A, D>A ) is consistent
* 1 removed from the domain of D because of arc ( D, D>A ) => in D-cons-B, cons-B side edge becomes not visited(=have to revisit) because D has changed => check “not(B=D)” constraints again in next step
* Arc ( B, not(B=D) ) is consistent
* Arc ( C, not(E=C) ) is consistent
* Arc ( E, not(E=C) ) is consistent
* 3 4 removed from the domain of E because of arc ( E, E<D-1 ) => in E-cons-C, cons-C side edge becomes not visited(=have to revisit) because E has changed
* Arc ( C, not(E=C) ) is consistent
* 2 removed from the domain of D because of arc ( D, E<D-1 ) => in D-cons-A, cons-A side edge becomes not visited(=have to revisit) because D has changed // in D-cons-B, cons-B side edge becomes not visited(=have to revisit) because D has changed
* Arc ( B, not(B=D) ) is consistent
* Arc ( A, D>A ) is consistent
* Arc ( E, not(E=B-2) ) is consistent
* Arc ( B, not(E=B-2) ) is consistent
* Arc ( A, not(F=A) ) is consistent
* Arc ( F, not(F=A) ) is consistent
* Arc ( B, not(F=B) ) is consistent
* Arc ( F, not(F=B) ) is consistent
* Arc ( F, not(C=F) ) is consistent
* Arc ( C, not(C=F) ) is consistent
* Arc ( F, not(D=F) ) is consistent
* Arc ( D, not(D=F) ) is consistent
* Arc ( E, E-F is odd ) is consistent
* Arc ( F, E-F is odd ) is consistent => all edge visited

**(c) Show explicitly the constraint graph after arc consistency has stopped.**



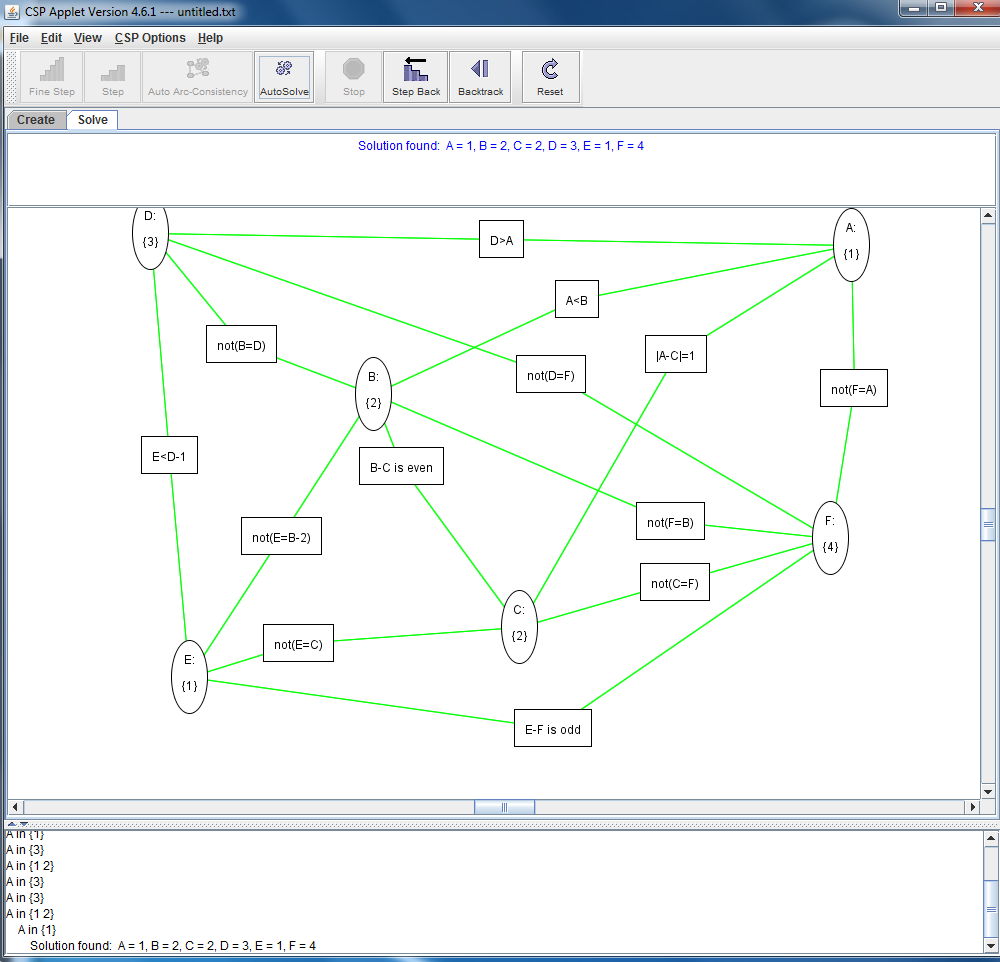
**(d) Show how splitting domains can be used to solve this problem. Draw the tree of splits and show the solutions.**

(continues from (c), starts splitting domains after arc consistency has stopped)

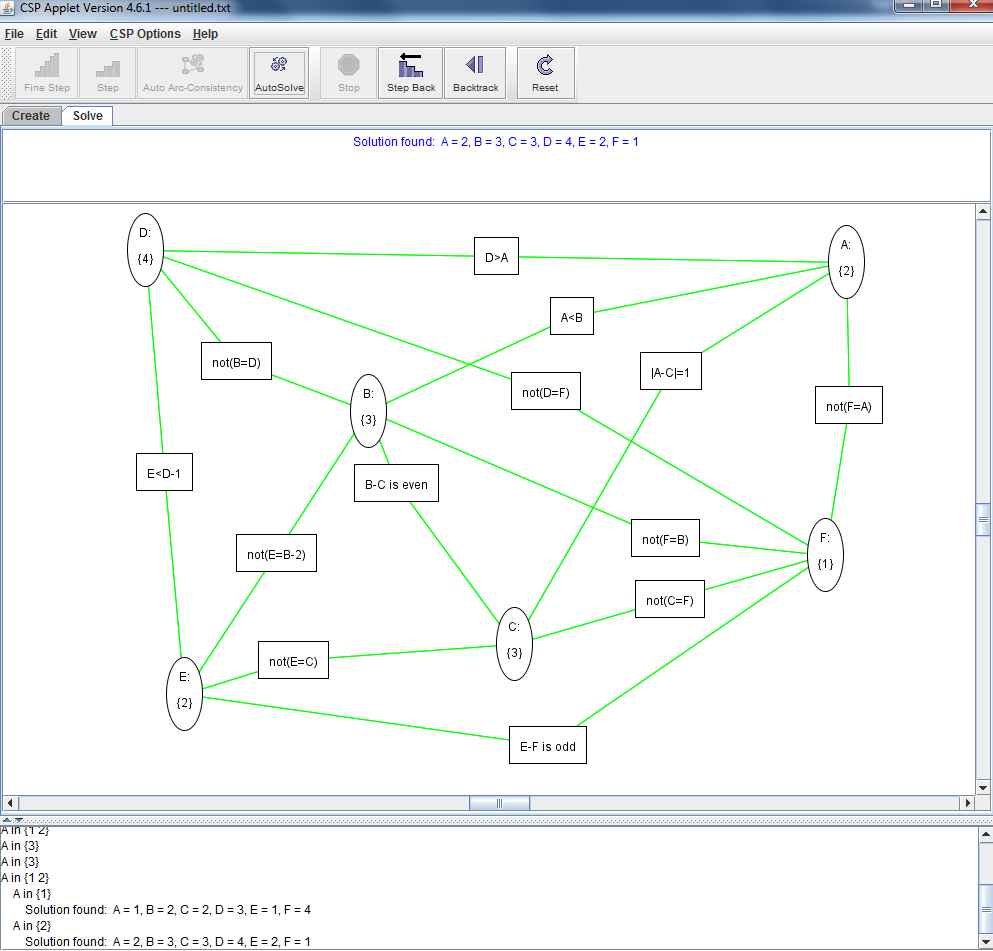


Split {1,2} from A

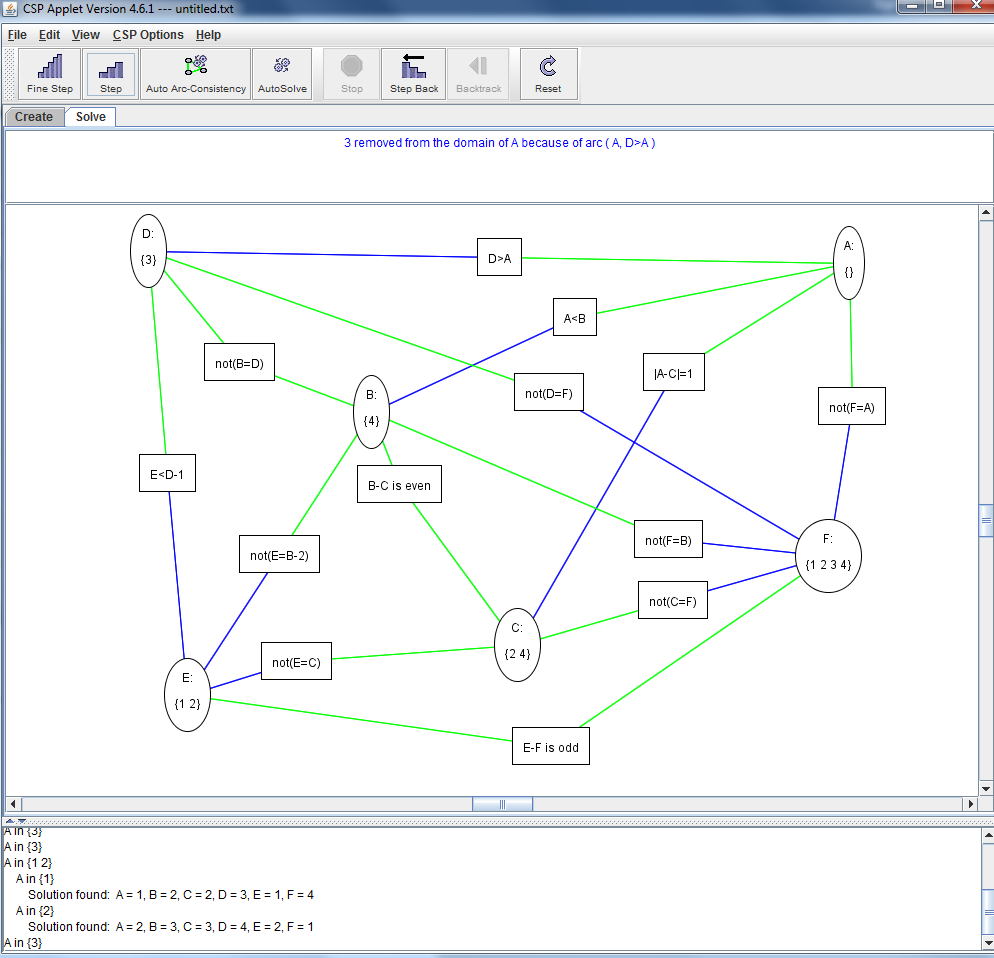
(next page continue)

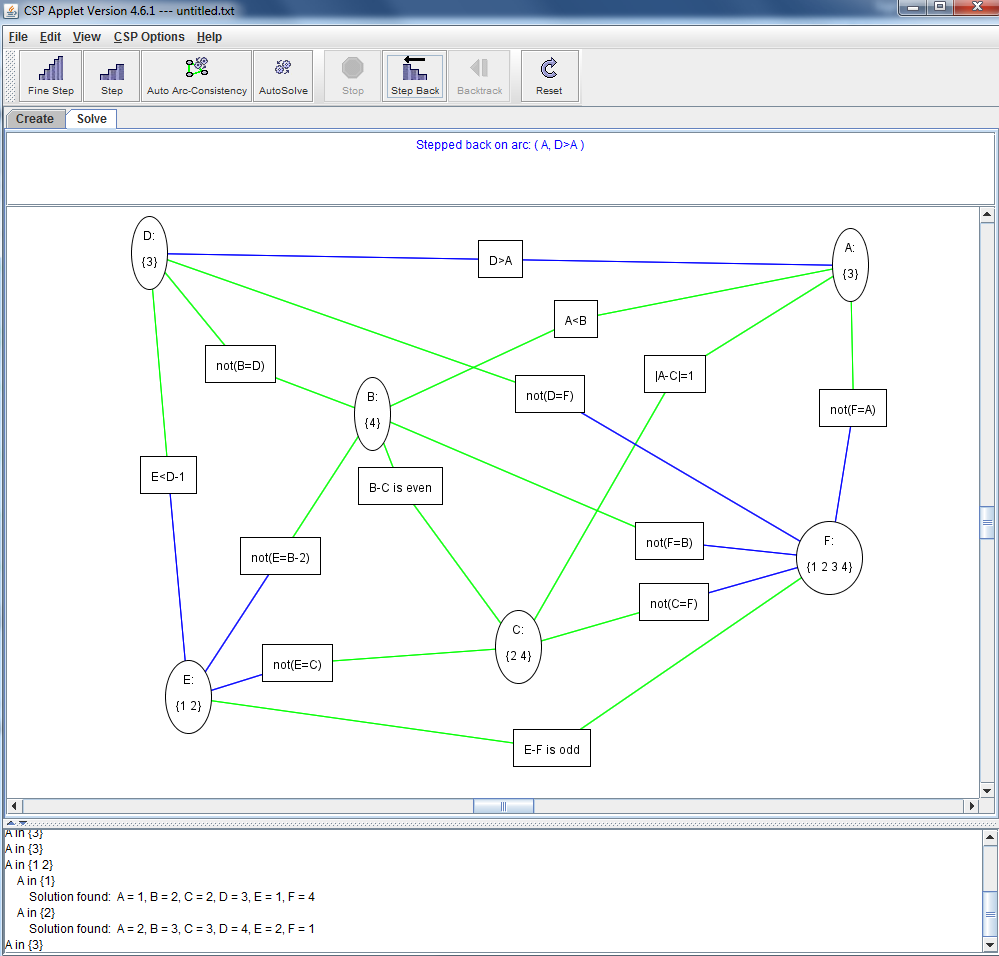


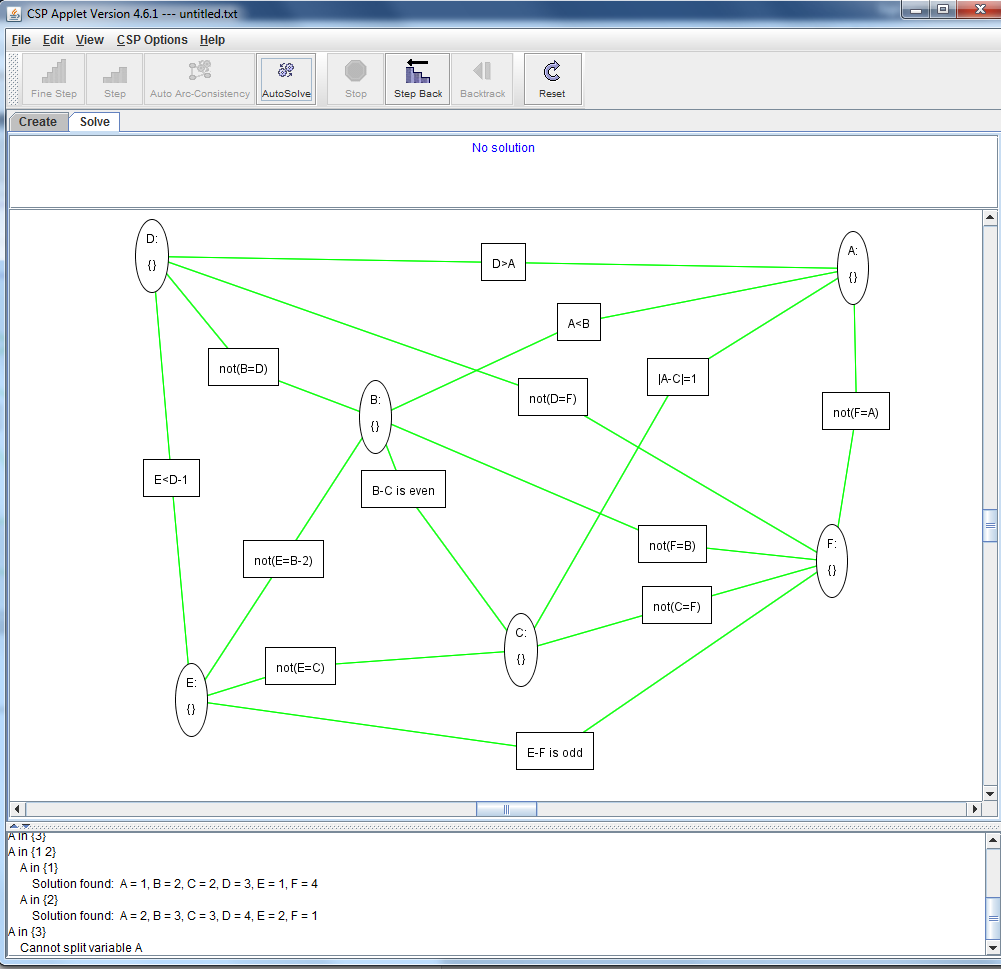
Split {1} from {1,2} => Solution found : A=1, B=2, C=2, d=3, E=1, F=4



Backtrack => split 2 from {1,2} => Solution found : A=2, B=3, C=3, D=4, E=2, F=1







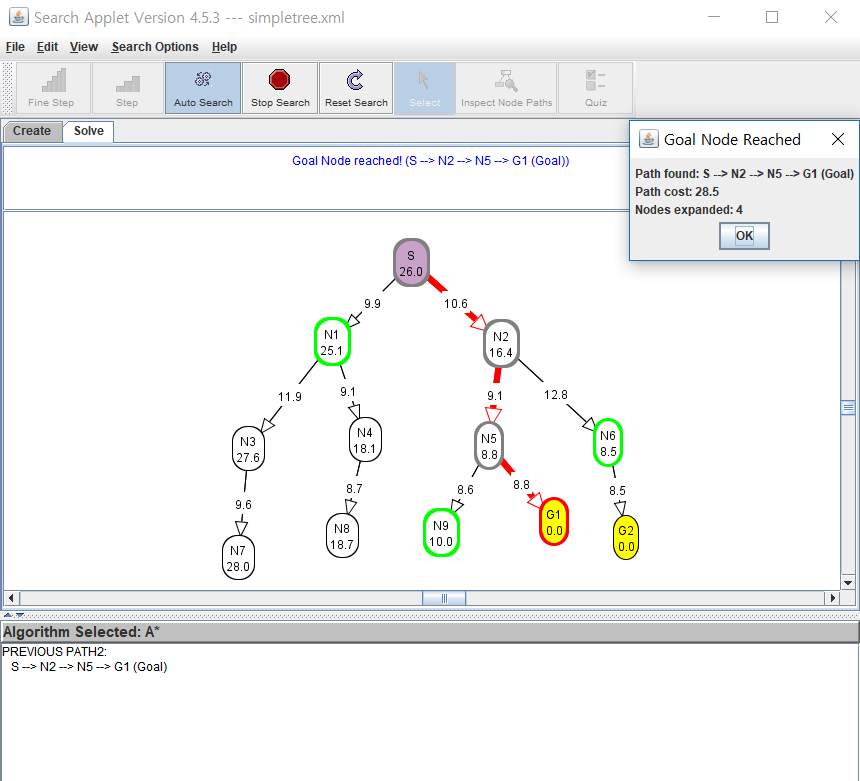
Backtrack => split 3 from A => A becomes {} => all variables becomes {} => no solution => can’t split A more => done

∴ solutions Found : A=1, B=2, C=2, d=3, E=1, F=4 / A=2, B=3, C=3, D=4, E=2, F=1

(next page continue)

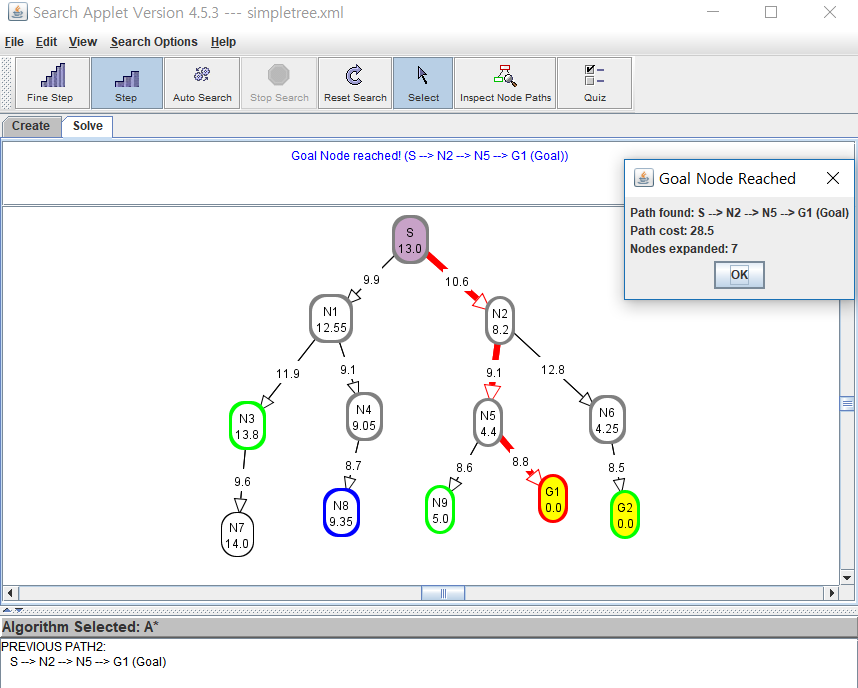
**3. Question Three [15 marks]**

**(a) What is the effect of reducing h(n) when h(n) is already an underestimate?**



Before reducing h(n) (already underestimated)

* Path found : S-N2-N5-G1 / Cost : 10.6 + 9.1 + 8.8 = 28.5
* Node expanded : S-N2-N5-G1 (4)

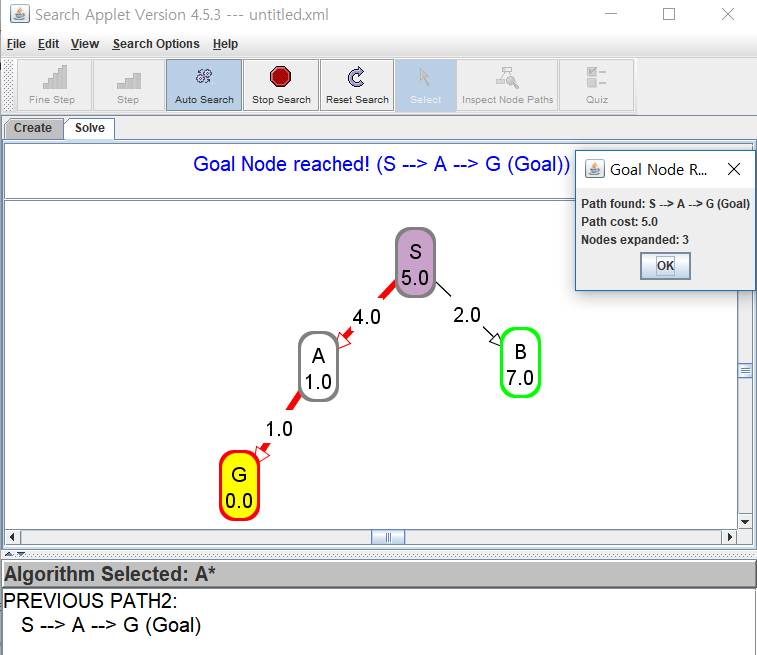


After multiplying each h(n) 0.5 (each node’s h(n) reduced to half)

* Path found : S-N2-N5-G1 / Cost : 28.5
* Node expanded : S-N2-N1-N5-N6-N4-G1 (7)

∴ Even we reduced h(n) when they are already underestimates, we can still find optimal solution(optimal path). However, the lower h(n) is, the more node A\* expands, therefore making search slower. In other words, if we further reduce h(n) even they are underestimates, it becomes inefficient.

**(b) How does A\* perform when h(n) is the exact distance from n to a goal?**

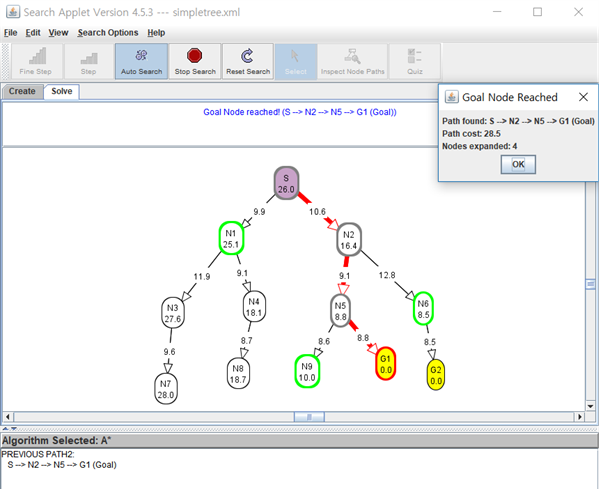


* Path found : S-A-G / Cost : 4 + 1 = 5 / Node expanded : S-A-G (3)

If heuristic is the exact distance from current node to goal, the graph becomes like above example. It means h(n) is exactly equal to the cost of moving from n to the goal, therefore A\* will only follow the best path and never expand anything else. In other words, it would guarantee to find optimal solution and always works better. In this situation, start node’s heuristic value would be same as optimal solution’s path cost.

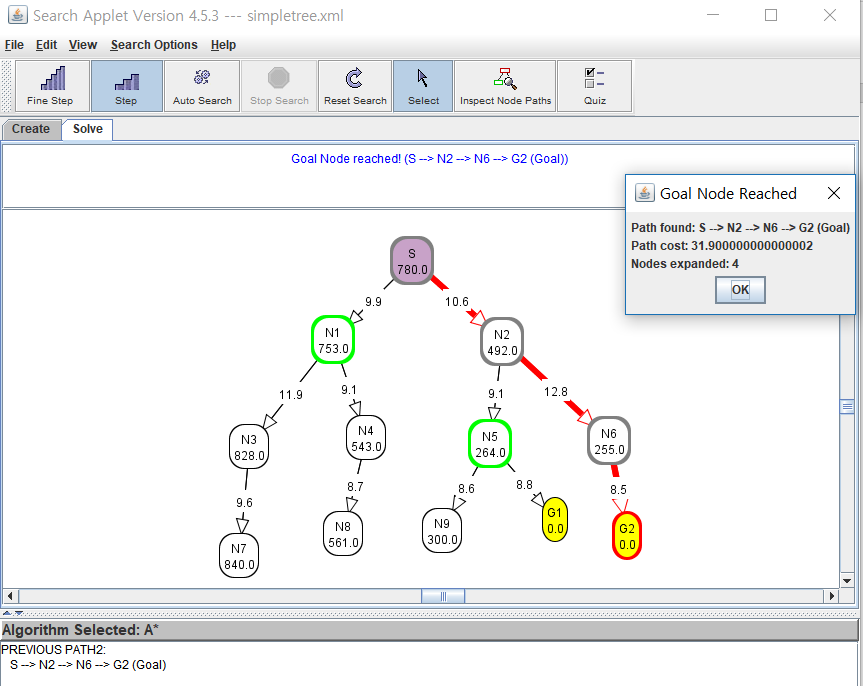
(next page continue)

**(c) What happens if h(n) is not an underestimate?**



Before reducing h(n) (already underestimated)

* Path found : S-N2-N5-G1 / Cost : 10.6 + 9.1 + 8.8 = 28.5
* Node expanded : S-N2-N5-G1 (4)



After multiplying each h(n) 30 (each node’s h(n) increased to 30 times) => h(n) are not underestimates

* Path found : S-N2-N6-G2 / Cost : 10.6 + 12.8 + 8.5 = 31.9
* Node expanded : S-N2-N6-G2 (4)

∴ When we increased each h(n) and made them not underestimates, it didn’t reach to optimal solution(optimal path). If h(n) gets sometimes greater than the cost of moving from n to the goal, it considers h(n) more rather than g(n), so that it will just expand node which has smaller h(n). This leads to conclusion that this case is not guaranteed to find a shortest path, but it might run faster(end search faster) since there would be fewer expanded nodes.

**4. Question Four [0 mark]**

**To help us in future planning with the lab exercise, please mention how much time you spent on each of the questions.**

* Q1 : 2 hours
* Q2 : 2 hours
* Q3 : 3 hours

(End)