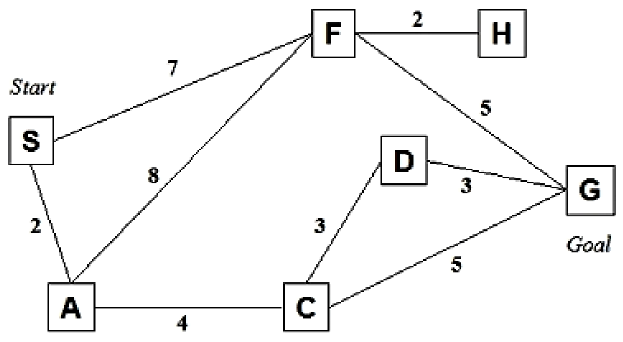
CS 380/480 – Foundations of Artificial Intelligence
Winter 2007
Assignment 2 – Solutions to Selected Problems

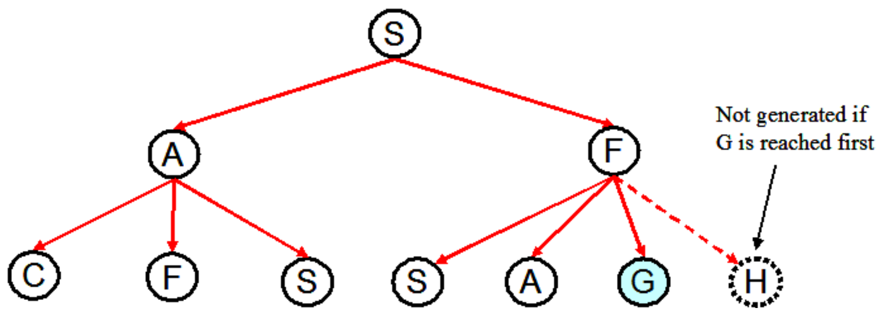
1. Search trees for the state-space graph given below:



Solution:

We only show the search trees corresponding to Breadth-First Search and Uniform Cost Search. It should be noted that for this problem we are no assuming repeated state checking. Thus, expanding a node may generate nodes that were previously visited in the search.

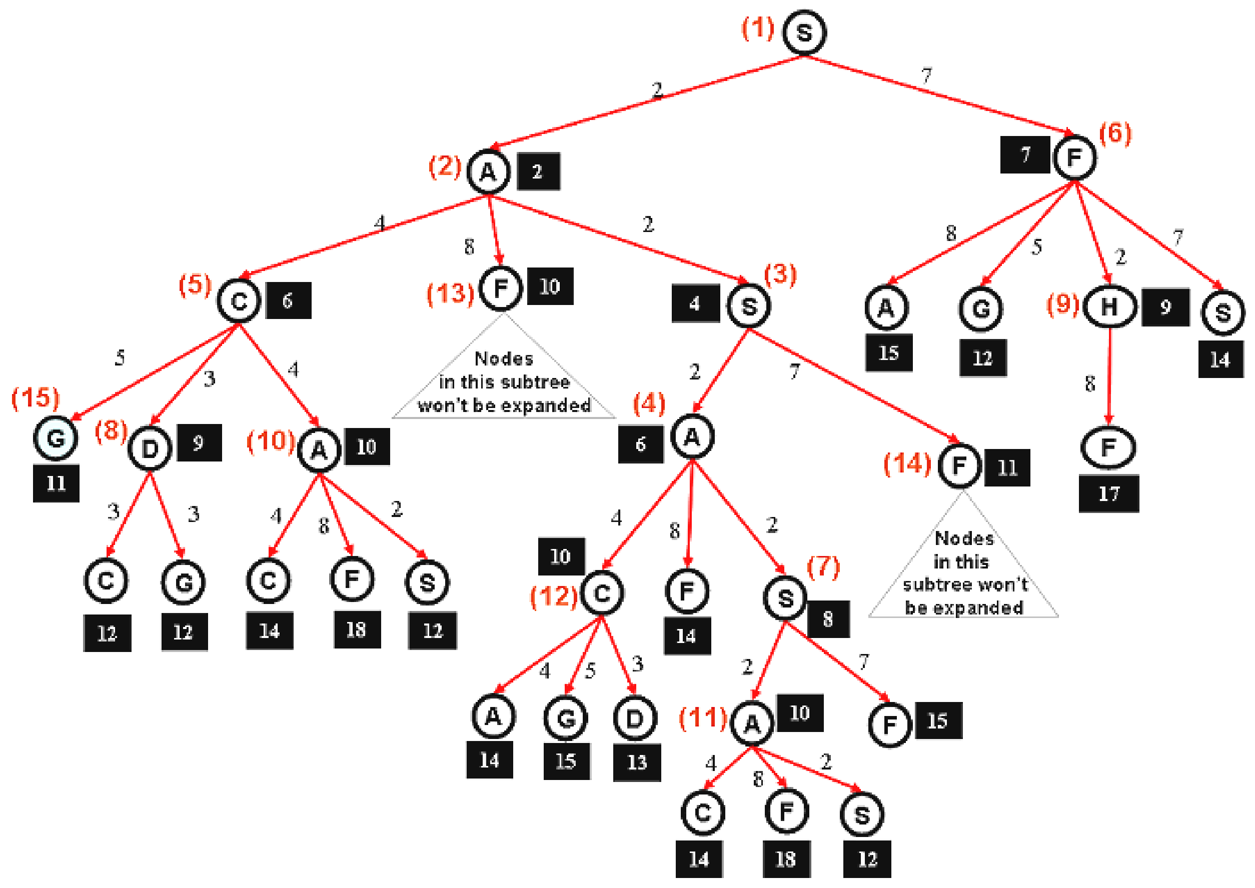
Tree for Breadth-First Search:



Note that the problem statement assumes the goal test occurs as soon as the nodes are generated (not when the nodes are selected from the queue for expansion). So, no additional nodes are generated once G is generated as a child of F. Also, the tree structure will depend on which nodes are expanded first. For example, at level 1, if F were expanded before A, then G would have been generated (and the search terminated) before expanding A.

You should note, however, that, in the standard BFS algorithm, the goal test occurs once a node is selected for expansion not when it is generated due to expanding its parent. So, the above formulation, is a non-standard formulation of the problem.

Tree for Uniform-Cost Search:


Nodes in this subtree won't be expanded
Nodes in this subtree won't be expanded

In the above tree, each edge is labeled with the actual cost of the move. The numbers in black boxes represent the $g(n)$ value for each node n. Recall that $g(n)$ is the total cost of the path to node n from the root of the tree. The (red) numbers in parentheses show the order of expansion. Specifically, nodes are expanded as follows: S → A → S → A → C → F → S → D → H → A → A → C → F → F → G. In cases of ties in $g(n)$ values, the node whose letter is smaller alphabetically is chosen for expansion (e.g., A was chosen before C after expanding node S at level 2). For the sake of clearer presentation, for some nodes whose children will not eventually be expanded, the subtrees haven been omitted (these are the two triangles in the figure under two instance of F). Note that according to the problem specifications, in contrast to the BFS, DFS, and IDS searches for parts (a) through (c), in this case we only perform a goal test when a node is selected for expansion (not immediately when it is generated during the search). This is, in fact, the standard search procedure discussed in class.

2. Heuristic Search

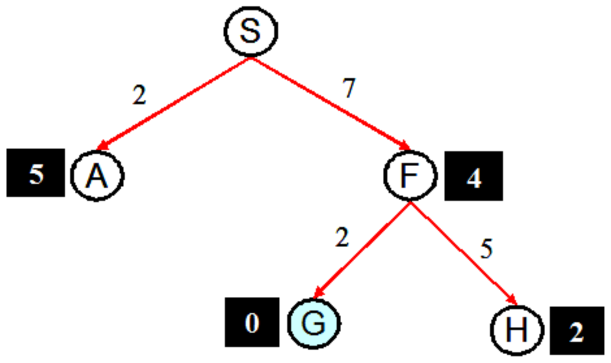
In this problem we use the same state-space graph as the previous problem. In addition, we are given a heuristic evaluation function *h*, defined according to the following table.

State	*S*	*A*	*C*	*D*	*F*	*G*	*H*
h	10	5	4	3	4	0	2

Also, note that in this problem we assume there is repeated-state checking, and, as in the case of Uniform-Cost search in problem 1, we only perform a goal test when a node is selected for expansion.
<u>Solution:</u>

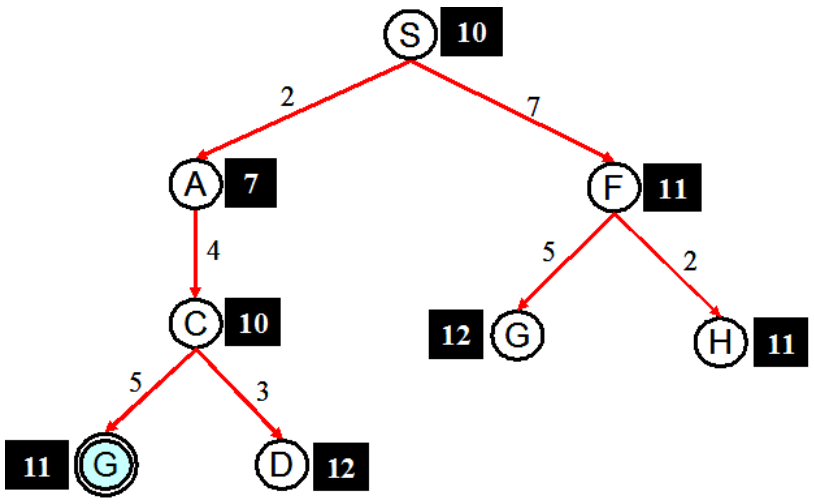
First, we note that the heuristic function *h* given above is indeed an **admissible** heuristic. This is because *h* never over-estimates the actual cost of the optimal path to the goal node G. In other words, for each node *n* (where n is once of S, A, C, D, F, G, or H), *h*(*n*) is less than or equal to the cost of the path from *n* to the goal node.

Tree for Greedy Best-First Search:



In the above tree the numbers in boxes next to each node represent the heuristic value, *h*, of the node. The costs for each move are also shown as labels on the edges; however these are not taken into account in Greedy Best-First search. In this example, the greedy search goes directly to the goal node without expanding extraneous nodes. However it does not succeed in finding the optimal (lowest cost) solution for the problem (which is S→A→C→G).

Tree for A* Search:



In the above tree, the boxes next to each node depict the evaluation function *f*(*n*) = *g*(*n*) + *h*(*n*). For example, the *g*(*n*) for node C in the left branch is 2 + 4 = 6 and h(n) for C is 4. This results in the *f* value of 10 for C. Here the tie between G on the left branch and F in the right branch was broken in favor of F. Note that despite the expansion of some extraneous nodes, A* does find the optimal solution to the problem. This, in fact, is guaranteed by A* so long as we use an admissible heuristic.

6. Problem 7.8 on Page 237 of Russell and Norvig:

<u>Solution:</u>

(a) ***Smoke* => *Smoke***

This is a valid sentence. This can be verified using truth tables, but it is easier to observe that the sentence is equivalent to **¬*Smoke*** ∨ ***Smoke*** which is true always true.

(b) ***Smoke* => *Fire***

This is just implication with two propositional variables, so we can refer to the truth table for "=>". It is satisfiable since the implication is true when ***Smoke*** is false or when ***Fire*** is true.

(c) **(*Smoke* => *Fire*) => (¬*Smoke* => ¬*Fire*)**

Again we can use truth tables, but let's try simplification using equivalence rules:

(*Smoke* => *Fire*) => (¬*Smoke* => ¬*Fire*)
➔ (¬*Smoke* ∨ *Fire*) => (*Smoke* ∨ ¬*Fire*) [elimination the inside implications]
➔ ¬ (¬*Smoke* ∨ *Fire*) ∨ (*Smoke* ∨ ¬*Fire*) [elimination the remaining implication]
➔ (*Smoke* ∧ *Fire*) ∨ (*Smoke* ∨ ¬*Fire*) [moving negation inside - DeMorgan's laws]
➔ (*Smoke* ∨ *Smoke* ∨ ¬*Fire*) ∧ (*Fire* ∨ *Smoke* ∨ ¬*Fire*) [distributivity]
➔ (*Smoke* ∨ ¬*Fire*) ∧ *True* [simplification - see also part (d)]
➔ (*Smoke* ∨ ¬*Fire*) [note: *P* ∧ *True* is equivalent to *P*]
➔ (*Fire* => *Smoke*) [converting back to implication]

So, we have reduced the formula to ***Fire* => *Smoke*** which is satisfiable using similar reasoning as in part (b).

(d) **(*Smoke* ∨ Fire ∨ ¬*Fire*)**

This is just equivalent to ***True***, since it contains ***Fire*** ∨ **¬*Fire*** which is valid (always true). Disjunction of a true statement with anything is still true. So the above formula is valid.

(e) **((*Smoke* ∧ *Heat*) => *Fire*) <=> ((*Smoke* => *Fire*) ∨ (*Heat* => *Fire*))**

This is valid. You can show this validity by constructing truth tables for the two sides of logical equivalence <=> and verifying this equivalence (by showing that for any assignment of truth values to the variables ***Smoke***, ***Heat***, and ***Fire***, the two formulas are either both true or both false.

6. Suppose we have a knowledge base KB containing the following 4 sentences:

KB1: $(\boldsymbol{A} \lor \neg \boldsymbol{C}) \Rightarrow \boldsymbol{B}$
KB2: $\neg \boldsymbol{C} \lor \boldsymbol{D}$
KB3: $\boldsymbol{D} \Rightarrow \boldsymbol{A}$
KB4: $\boldsymbol{C}$

Using the above knowledge base and inference rules for propositional logic, give a derivation (proof) for the sentence $\boldsymbol{A} \land \boldsymbol{B}$.

<u>Solution:</u>

1.	$\boldsymbol{C}$	KB4
2.	$\boldsymbol{D}$	Step 1, KB2, using Resolution rule
3.	$\boldsymbol{A}$	Step 2, KB3, using Modus Ponens
4.	$\boldsymbol{A} \lor \neg \boldsymbol{C}$	Step 3, using OR-Introduction (introducing $\neg \boldsymbol{C}$)
5.	$\mathbf{B}$	Step 4, KB1, using Modus Ponens
6.	$\boldsymbol{A} \land \boldsymbol{B}$	Step 1, Step 5, using AND-Introduction