Solution for Assignment #2

Problem 1 (10%)

The TSP can be formulated as a search problem by specifying the followings:

- **States**: a set of nodes that have been visited once (the starting node might be visited twice).
- **Initial State**: the salesman is at the starting point, and no other node has been visited.
- **Operators**: the salesman moves from the current node to an adjacent unvisited node. The cost of the operator is the cost of edge between the two nodes.
- **Goal test**: the salesman is at the starting point and all other nodes have been visited exactly once.

Problem 2 (10%)

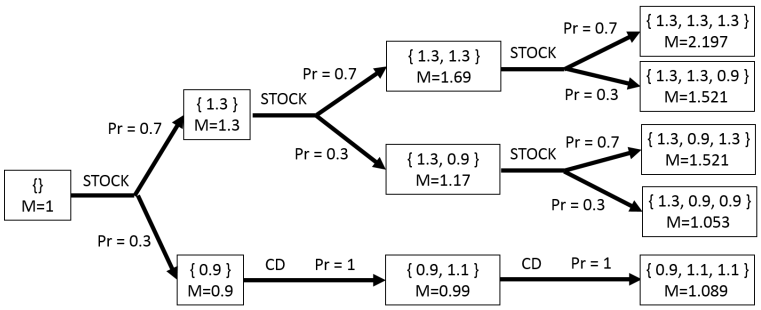
One of the possible ways to formulate *constructing crossword puzzle* as an assignment (CSP) problem is:

- **Variables**: one for each row r_i and one for each column c_i.
- **Domains**: the set of words in the dictionary.
- **Constraints**:
  - For each r_i and c_j, an equation $cross(i,j) = (m,n)$, meaning that the row r_i intersects with column c_j at the mth position for r_i and nth position for c_j. The position starts at 1, so if they do not intersect, then $m = n = 0$.
  - For each word w, an equation $w(i) = l$, where l is a letter, meaning that the ith position of the word w is letter l. Again we assume the position starts at 1.
  - For each row r_i and column c_j, if $cross(i,j) = (m,n)$ and m is not 0, then the constraint $r_i(m) = c_j(n)$.

Problem 3 (10%)

- States:
  - a special *end* state.
  - tuples $(x_1, ..., x_k)$, $0 \le k \le 3$, $x_i \in \{1.1, 1.3, 0.9\}$. In this state, k actions have been performed since the beginning, and the returns of these actions are $x_1, ..., x_k$, respectively. So in this state, you have $\Pi_i x_i$ units of money. In the following, if $s = (x_1, ..., x_k)$, then we denote by $M(s)$ the product $\Pi_i x_i$. If $s = ()$, then $M(s) = 1$.
- $S_{start} = ()$; $End(s)$ iff $s = end$.
- $T(s, CD, s.append(1.1)) = 1$ if $|s| < 3$, where $s.append(1.1)$ means append 1.1 to the end of the tuple s (e.g. $(1.3, 1.1).append(1.1) = (1.3, 1.1, 1.1)$); $T((x,y,z), a, end) = 1$ for all action a; if $|s| < 3$, then $T(s, Stocks, s.append(1.3)) = 0.7$ and $T(s, Stocks, s.append(0.9)) = 0.3$.
- $Rewards(s, a, end) = 0$ if $M(s) \ge 1$, and $Rewards(s, a, end) = -10$ if $M(s) < 1$; If $s' \ne end$, then $Rewards(s, a, s') = M(s') - M(s)$.

The optimal policy π should be that taking *Stock* only if losing 10% can be made up later on by CD's. That is, if $\pi(s) = Stocks$, then there are still enough CD actions left to make $s.append(0.9)$ into a state s' such that $M(s') \geq 1$. The following is the execution tree of this optimal policy:



Problem 4 (15%)

There are two ways to read this question: the robot has the same 8 sensors as in our running example or it is sensor-impaired or even blind. The default is to use the same 8 sensors as before. Under this assumption, the main component of a robot's state is the sensor reading vector. While the robot's actions are deterministic (no uncertainty), its knowledge about it in terms of what the sensor readings will be after an action is incomplete, so need to be defined as a probabilistic transition relation. The main point of the question is for you to think about how to specify the desired behavior by the reward function.

A "cheating" way is to give rewards according to the production system that we know will make the robot do the right thing:

- States: vectors of 8 sensors.
- Starting state: S_0, when all sensors are 0 (nothing around it).
- End state: no end state.
- Discount factor = 1.
- Rewards: $R(s, a, s')$ is 1 if a is the action to do according to the boundary-following production system, and -1 otherwise.
- Transition: $T(s, a, s')$: the probability that the robot's sensor vector will be s' given that it is currently s and it performs a. This probablity can only be fully specified if the environment is known. But there are certain properties that we can deduce about it. For example, if $s_2 = 1$ in s, then $T(s, North, s) = 1$ (it bumps into something so stay where it is).

Another way is to record the entire history in the state, and reward actions that extend the current history in good ways. But for this example, there is no need to keep the entire history, just last action performed is enough:

- States: (a, s) - last action a, and current sensor reading s.
- Starting state: (nil, S_0).
- No end state.
- Discount factor = 1.
- Transition: similar to above.
- Rewards: $R((A, s), a, (a', s'))$ is 1 if
  1. either $s' \neq S_0$ or $s = S_0$ (don't move away from an obstacle),
  2. $a = a'$, and

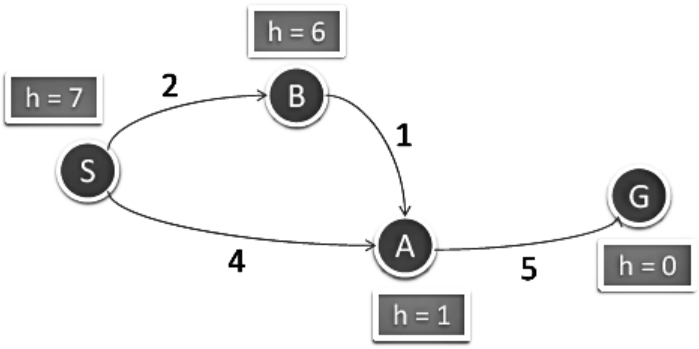
3. (A, a) is not a loop (North and South or East and West);

and 0 otherwise. Notice that here we assume no tight space so that the robot does not have to choose which obstacle to follow.

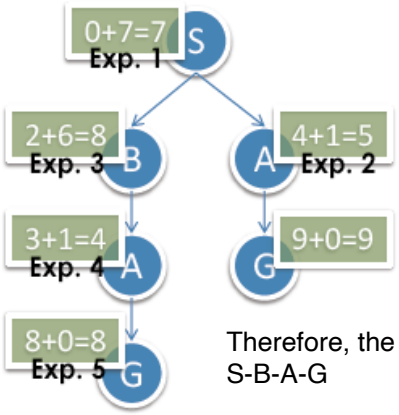
To make it into reinforcement problem, you can drop T, *Rewards* or both. Without *Rewards*, you can only train using the whole epoch, i.e. decide how to distribute rewards at the end of the 'game', in this case, like a maximum number of steps you want the robot to run.

Problem 5 (15%)

The search problem can be illustrated by the graph below:



A^* by tree



Therefore, the final solution will be S-B-A-G

Problem 6 (10%)

The values of the nodes: $I = 0$, $F = 5$, $G = 8$, $C = 4$, $B = 3$, $A = 4$.

Problem 7 (10%)

From left to right: nodes N and L are pruned. From right to left: nodes K and I are pruned.