7 Temporal Models (20 points)

1. For the Rain-Umbrella HMM model (from class and Figure 15.2 in AIMA) with 2 time periods, show how **variable elimination** operates to answer the following query $P(R_1|U_1 = T, U_2 = T)$. You do not need to do numerical calculations.

   We need to eliminate R_0 and R_2, the initial factors are

$$P(R_0), P(R_1|R_0), P(R_2|R_1), P(U_1 = T|R_1), P(U_2 = T|R_2)$$

   Eliminating R_0

$$F_1(R_1) = \sum_{R_0} P(R_0)P(R_1|R_0)$$

   Eliminate R_2

$$F_2(R_1, U_2 = T) = \sum_{R_2} P(U_2 = T|R_2)P(R_2|R_1)$$

   The final answer is the product of these three factors:

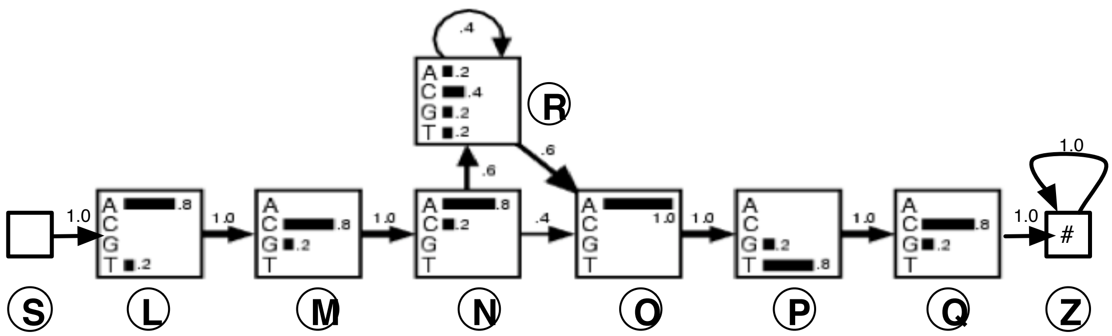
$$F_1(R_1)P(U_1 = T|R_1)F_2(R_1, U_2 = T)$$

2. Explain the relationship between the variable elimination computation and the **forward/backward algorithm**. Be specific, referring to the factors obtained during variable elimination.

   The forward message in a smoothing computation would be: $P(R_1|U_1 = T)$ which is $F_1(R_1)P(U_1 = T|R_1)$, the backward message would be $P(U_2 = T|R_1)$ which is $F_2(R_1, U2 = T)$

5 Hidden Markov Models (20 points)

The following picture specifies a Hidden Markov Model for a DNA sequence "pattern", of length at least 6. There are two special states, S which is the start state and Z which is the end state. We always start in state S, that is, the state distribution at time 0 has all the probability on state S, there are no observations in state S. Once we transition into state Z, all the observations are of an "end of string" symbol (which is not relevant in this problem). The other states show the distribution over the observations possible in that state (one of the four bases of DNA - A,C,G,T). The transition probabilities between the states are given on the arcs between the states.



1. (10 pts) What is the distribution over the states after seeing "ACAAA", i.e. $P(X_5|E_{1:5} = ACAAA)$. Compute the probabilities and write them in the 5th table column. Write the probabilities as sums and products of entries in the diagram, do **not** multiply or add the numbers. So, an entry in the table could be $0.3 \times 0.4 \times 0.2 + 0.5 \times 0.2 \times 0.2$. The table is big enough to keep track of intermediate results, but you do not need to fill in the whole table. If you do, you might want to label important entries with letters, e.g. a and b, so you can reuse them. In this range of time, S and Z have 0 probability, so they're not shown.

State	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$
L	1	0	0	0	0
M	0	1	0	0	0
N	0	0	1	0	0
O	0	0	0	0.4*1	a*(0.6*1.0)
P	0	0	0	0	0
Q	0	0	0	0	0
R	0	0	0	0.6*0.2 = a	a*(0.4*0.2)

Each column should be normalized.

2. (1 pt) Which algorithm should be used for computing the distribution above?

   Forward (Filtering)

3. (4 pts) How would you expect the distribution you computed above (for X_5) to change after seeing "ACAAAATC", that is, $P(X_5|E_{1:8} = ACAAAATC)$? You do not need to compute the new distribution numerically; indicate the qualitative changes to the state probabilities.

   The probability of being in state R at time 5 should be 1.

4. (1 pt) Which algorithm should be used for computing this new distribution?

   Forward-Backward (smoothing)

5. (4 pts) Explain how this algorithm would arrive at the expected change in the distribution.

   The backwards part of the computation would result in non-zero probabilities for states N and R. The forward part of the computation would have non-zero probabilities for states O and R. The product of these probabilities will result in only R having non-zero probability. When we normalize, this probability will become 1.

8 Search Algorithms (29 points)

1. What are to pros (if any) and cons (if any) to using an expanded list with Uniform Cost Search (as opposed to not keeping track of visited or expanded states)? Explain; consider both time and space.

```
If all the states are in memory, keeping track of expanded states
does not take much space; if states are generated then remembering
can take lots of space.
But, remembering states can reduce the running time enormously,
think of a grid where there are many paths with the same cost
to the same state.
```

2. What are the pros (if any) and cons (if any) to using A* versus Uniform Cost Search? Explain; consider both time and space.

```
Evaluating the heuristic in A* can take extra time, but if the
heuristic is good (informed) it can cut down the number of
expanded states a lot (which helps running time and space).
```

3. What are the pros (if any) and cons (if any) to using Breadth First versus Progressive Deepening? Explain; consider both time and space.

```
PD uses much less space (since it's running DFS which takes time
proportional to the depth of the tree) but it does take more time
than BFS, since it repeats the search of the upper layers of the
tree.  However, in a very large tree, the running time is
dominated by the last search.  So, PD can cut the space by a lot
in exchange for a relatively modest increase in time.
```

4. If you take an admissible heuristic $h(s)$ with A* and modify it so that it returns $2h(s)$, what could be the impact on A*? Indicate possible positive and negative impacts.

```
The new heuristic will generally not be admissible, so A* will
no longer be guaranteed to find the optimal answer.  In most
cases, you would expect that this would make the running time
smaller since the resulting search is "greedier".
```

5. Imagine that you are developing an algorithm to plan the motion of a mobile robot moving among obstacles. You have a map that indicates feasible intermediate locations between the start and the goal. Assume your map gives the coordinates of each location as well as which locations are reachable from which other locations. Each location has a position and also a "clearance", indicating how tight the passage is. Lower clearance will require the robot to slow down to go through there. So, we want to minimize a combination of the length of the path and the sum of (1/clearance) for every location along the path, for example, $\sum LinkLength + \alpha \sum \frac{1}{clearance}$, with α used to control the relative importance of the two measures. Give a (non-zero) admissible heuristic for this problem and discuss (briefly) its strengths and weaknesses.

```
Using euclidean distance to goal is admissible, but not very
informed if alpha is not small -- 1/clearance could be very big.
```

6. You are given a graph corresponding to the structure of the complete World Wide Web. A node corresponds to a Web page and there are directed links connecting the nodes. If you want to find the shortest sequence of links from a start page to an end page, what search algorithm should you use? Explain.

```
Certainly not depth-first search, since it could go (nearly)
forever.  All the costs are the same and no reasonable heuristic,
so not UCS or A*.  That leaves BFS or PD.  Given the big branching
factor and likely large depth, PD is probably best.
```

7. When using the BT-FC algorithm for CSP problems you need to test any new assignment to a variable at the leaf of the search tree for consistency with previous assignments along the path. Is this True or False? Explain.

   This is False. The domains of "future" variables have already been pruned by "past" assignments, so the only values left in the domain are already consistent.

8. Briefly, explain why BT-FC is often more efficient than simple backtracking.

   It avoids examining sub-trees in the search space that are doomed to fail due to inconsistency with an assignment early on in the search. BT would generate (and test) all the intermediate assignments before detecting the failure.

9. Circle TRUE or FALSE next to each statements below:

   (a) Alpha-Beta and MIN-MAX always compute the same score.
   True **False** True

   (b) In the best case, on a fixed tree, Alpha-Beta cuts the running time of MIN-MAX in half.
   True **False** False - cuts the exponent in half

   (c) Alpha-Beta computes the same score independent of the ordering of the leaves of the game tree.
   True **False** True - see (a)

   (d) The running time of Alpha-Beta is independent of the ordering of the leaves of the game tree.
   True **False** False - order matters a lot

   (e) A good chess-playing program will never need more than 13-ply look-ahead.
   True **False**
   False - in some cases much deeper look-ahead is needed, Deep Blue sometimes goes to 30.

4 Constraints (20 points)

1. (8 pts) We have three variables: X, Y, and Z, each with the domain = $\{1, 2, 3, 4, 5\}$. The constraint between each pair of variables is that the sum of the variable assignments be odd. Assume you use BT-FC with the variable and value orderings are as given above. For each variable, how many tentative assignments would you need to try before discovering that it is impossible to find a satisfying solution to this problem.

   - Number of tentative assignments to X = 5
   - Number of tentative assignments to Y = 12
   - Number of tentative assignments to Z = 0

2. (3 pts) **True** or **False**: We could improve the efficiency of solving CSPs if we could develop a good heuristic function, so that we could use A^* instead of depth-first search.

   False. In CSP all the goals are at the same depth of the tree, so the choice among "paths" is not affected by distance to the goal.

3. (3 pts) **True** or **False**: Full constraint propagation may not always find a unique solution to a CSP, but it will always tell you when a solution does not exist.

   False. When multiple elements remain in the domain, we don't know anything about the number of solutions.

4. (3 pts) **True** or **False**: When doing backtracking with forward-checking, we never need to check the consistency of a new assignment with previous assignments.

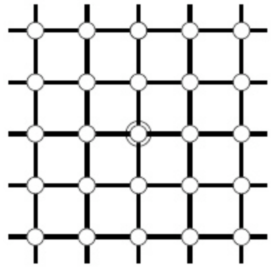
   True. All inconsistent assignments have already been eliminated.

5. (3 pts) **True** or **False**: Randomized hill-climbing methods such as GSAT can be viewed as a depth-first search in the space of partial assignments, in which the descendant of a search node is chosen randomly.

   False. GSAT operates on complete assignments.

5 Search (35 points)

Consider the **unbounded**, i.e, extending infinitely, regular 2D grid state space shown below; the cost of each link is 1. The start state is the origin (in the center of the grid) and the goal state is at (x, y). Assume, for simplicity, that $x \geq 0, y \geq 0$.



1. (1 pt) What is the branching factor b in this state space?

2. (3 pts) How many **distinct states** are there at depth k (for $k > 0$)? Circle all the true choices.

   (a) 4^k

   (b) $4k$ ← *Correct, assuming we don't "double back"*

   (c) $4k^2$ ← *If we "double back" it is closer to this.*

3. (3 pts) What is the length of the shortest path to the goal? And, is there unique such path?

   Length is $|x| + |y|$; all the paths in a rectangle of sides x and y have this length.

4. (3 pts) Let d be the length of the shortest path to the goal. Breadth-First search **without** an expanded or visited list expands (in the worst case) a number of nodes that grows as

   (a) 4^d ← *Correct*

   (b) d

   (c) d^2

   nodes before terminating. Circle all the true choices.

5. (3 pts) Let d be the length of the shortest path to the goal. Breadth-First search **with** an expanded list expands (in the worst case) a number of nodes that grows as

   (a) 4^d

   (b) d

   (c) d^2 ← *Correct, see comment about rectangle of paths above*

   nodes before terminating. Circle all the true choices.

6. (3 pts) **True** or **False**: $h = |u - x| + |v - y|$ is an admissible heuristic for a state at (u, v).

   True. This is exactly equal to path length.

7. (3 pts) **True** or **False**: A* search with a strict expanded list using h expands a number of nodes before terminating that grows linearly with $x + y$.

   False. It would grow as $x * y$ since all the paths in the rectangle described above could be on optimal paths.

8. (3 pts) **True** or **False**: h is admissible if some links are removed from the grid.

   True. This can only make the actual best path longer, so h remains an underestimate.

9. (3 pts) **True** or **False**: h is admissible if some links are added between non-adjacent states.

   False. The links could make the actual best path shorter.

10. (10 pts) Consider the following four search methods: depth-first (DF), breadth-first (BF), progressive deepening (PD), and uniform-cost (UC). Each of these methods can optionally be paired with an expanded list (EL). For each of these 8 methods (4 basic methods with and without EL), indicate those that **guarantee** the following properties on a grid state-space like above but **where the links have unequal costs**. You can give answers of the form "all except BF" or "all that use EL" or a list of methods "DF+EL, PD+EL" or "None".

   - Find a path to the goal: *All except DF and DF+EL*

   - Find path with fewest states: *BF, PD, UC, with or without EL*

   - Find minimal cost path: *UC, UC+EL*

   - Visit each state at most once: *None*

   - Expand each state at most once: *DF+EL, BF+EL, UC+EL*

6.034 Quiz 1, Spring 2005

1 Search Algorithms (16 points)

1.1 Games

The standard alpha-beta algorithm performs a depth-first exploration (to a pre-specified depth) of the game tree.

1. Can alpha-beta be generalized to do a breadth-first exploration of the game tree and still get the optimal answer? Explain how or why not. If it can be generalized, indicate any advantages or disadvantages of using breadth-first search in this application.

   No. The alpha-beta algorithm is an optimization on min-max. Min-max inherently needs to look at the game-tree nodes below the current node (down to some pre-determined depth) in order to assign a value to that node. A breadth-first version of min-max does not make much sense. Thinking about alpha-beta instead of min-max only makes it worse, since the whole point of alpha-beta is to use min-max values from one of the earlier (left-most) sub-trees to decide that we do not need to explore some later (right-most) subtrees.

   Some answers suggested that min-max inherently needs to go all the way down to the leaves of the game tree, where the outcome of the game is known. This is not true. Typically one picks some depth of look-ahead depth and searches to that depth, using the static evaluator to compute a score for the board position at that depth.

2. Can alpha-beta be generalized to do a progressive-deepening exploration of the game tree and still get the optimal answer? Explain how or why not. If it can be generalized, indicate any advantages or disadvantages of using progressive-deepening search in this application.

   Yes. Progressive-deepening involves repeated depth-first searches to increasing depths. This can be done trivially with min-max and alpha-beta as well, which also involve picking a maximum depth of lookahead in the tree. PD does waste some work, but as we saw in the notes, the extra work is a small fraction of the work that you would do anyways, especially when the branching factor is high, as it is in game trees. The advantage is that in timed situations you guarantee that you always have a reasonable move available.

1.2 Algorithms

1. You are faced with a path search problem with a very large branching factor, but where the answers always involve a relative short sequence of actions (whose exact length is unknown). All the actions have the same cost. What search algorithm would you use to find the optimal answer? Indicate under what conditions, if any, a visited or expanded list would be a good idea.

   Progressive deepening (PD), with no visited or expanded list would probably be the best choice. All the costs are the same, so breadth-first (BF) and PD both guarantee finding the shortest path in that situation, without the overhead of uniform-cost search. Since the branching factor is high, space will be an issue, which is why we prefer PD over BF. If we were to use a visited list with PD, the space cost would be the same as BF and it would not make sense to pay the additional run-time cost of PD (repeated exploration of parts of the tree) if we give up the space advantage.

2. You are faced with a path search problem with a very large branching factor, but where the answers always involve a relative short sequence of actions (whose exact length is unknown). These actions, however, have widely varying costs. What search algorithm would you use to find the optimal answer? Indicate under what conditions, if any, a visited or expanded list would be a good idea.

   Since we have variable link costs, we should use Uniform Cost search to guarantee the optimal answer. The fact that the costs are highly variable is good, since we expect that we might be able to avoid exploring sub-trees with high cost. Note that we don't necessarily have a useful heuristic and so A* may not be applicable. Using an expanded list would make sense if the search space involves lots of loops, which would lead us to re-visit the same state many times. However, since we know that there's a relatively short path to the goal, it might not be worth the extra space.

2 Constraints (16 points)

Consider assigning colors to a checkerboard so that squares that are adjacent vertically or horizontally do not have the same color. We know that this can be done with only two colors, say red (R) and black (B). We will limit our discussion to **five squares** on a 3x3 board, numbered as follows:

```
1 | 2 | 3
---------
4 | 5 |
---------
  |   |
```

Let's look at the CSP formulation of this problem. Let the squares be the variables and the colors be the values. All the variables have domains { R, B }.

1. If we run full constraint propagation on the initial state, what are the resulting domains of the variables?

   None of the variable domains change:

$$\begin{array}{lll} 1 = \{R, B\} & 2 = \{R, B\} & 3 = \{R, B\} \\ 4 = \{R, B\} & 5 = \{R, B\} & \end{array}$$

2. Say, instead, the initial domain of variable 5 is restricted to { B }, with the other domains as before. If we now run full constraint propagation, what are the resulting domains of the variables?

$$\begin{array}{lll} 1 = \{B\} & 2 = \{R\} & 3 = \{B\} \\ 4 = \{R\} & 5 = \{B\} & \end{array}$$

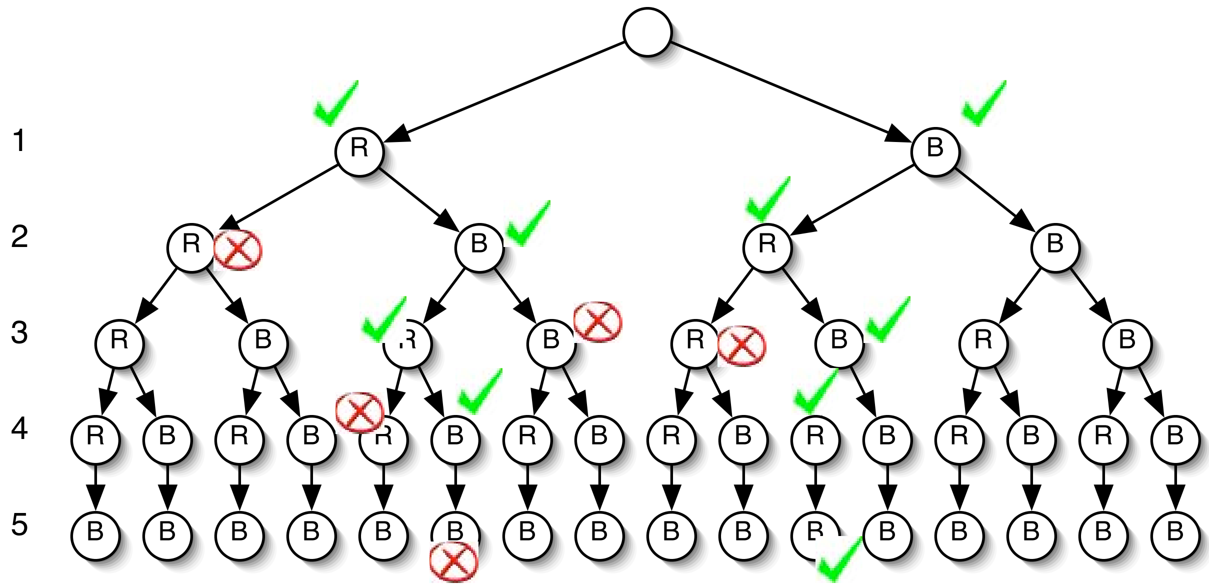
3. If in the initial state (all variables have domains { R, B }), we assign variable 1 to R and do forward checking, what are the resulting domains of the other variables?

   Forward checking is defined as a single iteration of constraint propagation only on those edges that terminate at the variable whose value was just set, and that do not originate from variables which have already been set. Therefore, after we set $1 = R$, forward checking affects the domains of variables 2 and 4 since they are adjacent to variable 1 (and have not yet been assigned).

$$\begin{array}{lll} 1 = \{R\} & 2 = \{B\} & 3 = \{R, B\} \\ 4 = \{B\} & 5 = \{R, B\} & \end{array}$$

   Forward checking only does one step of propagation, only to the immediate neighbors of the assigned variable.

4. Assume that during backtracking we first attempt assigning variables to R and then to B. Assume, also, that we examine the variables in numerical order, starting with 1. Also, let the domain of variable 5 be { B }, the other domains are { R, B }. In the following tree, which shows the space of assignments to the 5 variables we care about, indicate how pure backtracking (BT) would proceed by placing a check mark next to any assignment that would be attempted during the search and crossing out the nodes where a constraint test would fail. Leave unmarked those nodes that would never be explored.



5. If we use backtracking with forward checking (BT-FC) in this same situation, give a list of all the assignments attempted, in sequence. Use the notation variable = color for assignments, for example, 1=R.

We must keep track of the variable domains as we search since forward checking modifies these domains based on the current assignment, and we will need to restore the domain of earlier search nodes if we have to backtrack to them. We fail at a node if (1) the current assignments violate some constraint, or (2) if forward checking after the present assignment causes the domain of some variable to become empty. The following lists (in order from left to right) each attempted assignments and the resulting variable domains *after* forward checking.

Assignment:	None	$1 = R$	$2 = B$	$1 = B$	$2 = R$	$3 = B$	$4 = R$	$5 = B$
Domain of 1:	$\{R, B\}$	**R**	**R**	**B**	**B**	**B**	**B**	**B**
Domain of 2:	$\{R, B\}$	$\{B\}$	**B**	$\{R\}$	**R**	**R**	**R**	**R**
Domain of 3:	$\{R, B\}$	$\{R, B\}$	$\{R\}$	$\{R, B\}$	$\{B\}$	**B**	**B**	**B**
Domain of 4:	$\{R, B\}$	$\{B\}$	$\{B\}$	$\{R\}$	$\{R\}$	$\{R\}$	**R**	**R**
Domain of 5:	$\{B\}$	$\{B\}$	$\{\}$	$\{B\}$	$\{B\}$	$\{B\}$	$\{B\}$	**B**
			$\Downarrow$ **FAIL**					

Note that when we fail at $2 = B$, since there are no further values to try in the

domain of variable 2, we backtrack to the assignment of variable 1. When this happens, we restore the domains from *before* variable 1 was assigned, i.e. the ones listed above under "None".

6. If we use backtracking with forward checking (BT-FC) but with dynamic variable ordering, using the most-constrained-variable strategy, give a list of all the variable assignments attempted, in sequence. If there is a tie between variables, use the lowest-numbered one first. Use the notation variable = color for assignments, for example, 1=R.

   Use of the most-constrained-variable strategy entails assigning the variable first whose domain is smallest. This ordering is not only performed at the start of the search. Rather, it is updated after each variable is assigned and forward checking modifies the domains of unassigned variables. For this problem, variable 5 has the smallest domain initially. After assigning variable 5, the domains of variable 2 and 4 become smaller than those of variables 3 and 5. Since variable 2 has the lowest index, it is assigned next. And so on, as shown below:

Assignment:	None	$5 = B$	$2 = R$	$1 = B$	$3 = B$	$4 = R$
Domain of 1:	$\{R, B\}$	$\{R, B\}$	$\{B\}$	**B**	**B**	**B**
Domain of 2:	$\{R, B\}$	$\{R\}$	**R**	**R**	**R**	**R**
Domain of 3:	$\{R, B\}$	$\{R, B\}$	$\{B\}$	$\{B\}$	**B**	**B**
Domain of 4:	$\{R, B\}$	$\{R\}$	$\{R\}$	$\{R\}$	$\{R\}$	**R**
Domain of 5:	$\{B\}$	**B**	**B**	**B**	**B**	**B**

3 Constraint satisfaction (24 points)

You are trying to schedule observations on the space telescope. We have m scientists who have each submitted a list of n telescope observations they would like to make. An observation is specified by a target, a telescope instrument, and a time slot. Each scientist is working on a different project so the targets in each scientist's observations are different from those of other scientists. There are k total time slots, and the telescope has three instruments, but all must be aimed at the same target at the same time.

The greedy scientists cannot all be satisfied, so we will try to find a schedule that satisfies the following constraints:

C1. Exactly two observations from each scientist's list will be made (the choice of the two will be part of the solution).

C2. At most one observation per instrument per time slot is scheduled.

C3. The observations scheduled for a single time slot must have the same target.

Note that for some set of requested observations, there may not be any consistent schedule, but that's fine.
Consider the following three formulations of the problem.

A. The variables are the $3k$ instrument/time slots.

B. The variables are the m scientists.

C. The variables are the mn scientists' requests.

For each formulation, specify

1. The value domain for the variables.

2. The size of the domain for the variables (in terms of k, m, and n).

3. Which of the constraints are necessarily satisfied because of the formulation.

4. Whether the constraints can be specified as binary constraints in this formulation. If they can, explain how. If not, provide a counterexample.

Formulation A: The variables are the $3k$ instrument/time slots.
1. Domain: **for each instrument/time slot, the set of observations requesting that instrument and time slot and the value "empty"**
2. Size of domain: **at most m*n+1 per variable**
3. Satisfied constraints: **C2, since each variable (instrument/time) gets at most one value, an observation.**
4. Binary constraints?:

- **C1 is not a binary constraint in this formulation. It requires checking all the variable assignments at once to make sure that exactly two observations from each scientist's list are made.**
- **C3 is a binary constraint in this formulation. Place a constraint between the 3 variables with the same time slot and require that the targets of the assigned observation be equal if they are both non-empty.**

Formulation B: The variables are the m scientists.
1. Domain: **for each scientist, the set of all pairs of observations that scientist requested.**
2. Size of domain: $\binom{n}{2}$**, approximately** $n^2/2$**.**
3. Satisfied constraints: **C1, since we will guarantee that exactly two of the scientist's observations are scheduled.**
4. Binary constraints?:

- **C2 is a binary constraint in this formulation. Place a constraint between every pair of variables and require that the instrument/time slot requests don't conflict.**
- **C3 is a binary constraint in this formulation. Place a constraint between every pair of variables and require that the targets for observations with the same time slot don't conflict.**

Formulation C: The variables are the mn scientists' requests.
1. Domain: {Granted, Rejected}
2. Size of domain: 2
3. Satisfied constraints: None
4. Binary constraints?:

- **C1 is not a binary constraint in this formulation. It requires checking all the variable assignments of Granted observations at once to make sure that exactly two observations from each scientist's list are granted.**
- **C2 is a binary constraint in this formulation. Place a constraint between every pair of variables and require that the instrument/time slot requests don't conflict between any two Granted requests.**
- **C3 is a binary constraint in this formulation. Place a constraint between every pair of variables and require that the targets of the Granted observations with the same time slot don't conflict.**

4 Search Problem formulation (23 points)

Consider a Mars rover that has to drive around the surface, collect rock samples, and return to the lander. We want to construct a plan for its exploration.

- It has batteries. The batteries can be charged by stopping and unfurling the solar collectors (pretend it's always daylight). One hour of solar collection results in one unit of battery charge. The batteries can hold a total of 10 units of charge.
- It can drive. It has a map at 10-meter resolution indicating how many units of battery charge and how much time (in hours) will be required to reach a suitable rock in each square.
- It can pick up a rock. This requires one unit of battery charge. The robot has a map at 10-meter resolution that indicates the type of rock expected in that location and the expected weight of rocks in that location. Assume only one type of rock and one size can be found in each square.

The objective for the rover is to get one of each of 10 types of rocks, within three days, while minimizing a combination of their total weight and the distance traveled. You are given a tradeoff parameter α that converts units of weight to units of distance. The rover starts at the lander with a full battery and must return to the lander.

Here is a list of variables that might be used to describe the rover's world:

- types of rocks already collected
- current rover location (square on map)
- current lander location (square on map)
- weight of rocks at current location (square on map)
- cost to traverse the current location (square on map)
- time since last charged
- time since departure from lander
- current day
- current battery charge level
- total battery capacity
- distance to lander
- total weight of currently collected rocks

1. Use a set of the variables above to describe the rover's state. Do not include extraneous information.

   - **types of rocks already collected**
   - **current rover location (square on map)**
   - **time since departure from lander**
   - **current battery charge level**
   - **total weight of currently collected rocks (optional, depending on your choice of cost function)**

2. Specify the goal test.

   - **All types of rocks have been collected**
   - **rover at lander location**
   - **time since departure less than 3 days**

3. Specify the actions. Indicate how they modify the state and any preconditions for being used.

   charge **: precondition: none; effects: increases battery voltage by 1 unit, increases time-since-departure by 1 hour**

   move **: precondition: enough battery voltage to cross square; effects: decreases battery voltage by amount specified in map; increases time by amount specified in map; changes rover location**

   pick-up-rock **: precondition: enough battery voltage; effects: decreases battery voltage by 1 unit; changes types of rocks already collected**

4. Specify a function that determines the cost of each action.

   charge **: 0**

   move **: 10 meters**

   pick-up-rock **:** α ***** **weight-of-rocks-at-current-location**

5. This can be treated as a path search problem. We would like to find a heuristic. Say whether each of these possible heuristics would be useful in finding the optimal path or, if not, what's wrong with them. Let l be the number of rocks already collected.

   H1: The sum of the distances (in the map) from the rover to the $10 - l$ closest locations for the missing types of rocks.

   This heuristic is inadmissible.

   H2: The length of the shortest tour through the $10 - l$ closest locations for the missing types of rocks.

   This heuristic would take an impractical amount of time to compute; and while more reasonable than H1 is also inadmissible.

   H3: The distance back to the lander.

   This heuristic is admissible, but very weak.

5 Search traces (21 points)

Consider the graph shown in the figure below. We can search it with a variety of different algorithms, resulting in different search trees. Each of the trees (labeled G1 though G7) was generated by searching this graph, but with a different algorithm. Assume that children of a node are visited in alphabetical order. Each tree shows all the nodes that have been visited. Numbers next to nodes indicate the relevant "score" used by the algorithm for those nodes.

For each tree, indicate whether it was generated with

1. Depth first search
2. Breadth first search
3. Uniform cost search
4. A* search
5. Best-first (greedy) search

In all cases a strict expanded list was used. Furthermore, if you choose an algorithm that uses a heuristic function, say whether we used

H1: heuristic 1 = $\{h(A) = 3, h(B) = 6, h(C) = 4, h(D) = 3\}$

H2: heuristic 2 = $\{h(A) = 3, h(B) = 3, h(C) = 0, h(D) = 2\}$

Also, for all algorithms, say whether the result was an optimal path (measured by sum of link costs), and if not, why not. Be specific.

Write your answers in the space provided below (not on the figure).

G1: 1. Algorithm: **Breadth First Search**

2. Heuristic (if any): **None**

3. Did it find least-cost path? If not, why? **No. Breadth first search is only guaranteed to find a path with the shortest number of links; it does not consider link cost at all.**

G2: 1. Algorithm: **Best First Search**

2. Heuristic (if any): **H1**

3. Did it find least-cost path? If not, why?

No. Best first search is not guaranteed to find an optimal path. It takes the first path to goal it finds.

G3: 1. Algorithm: **A***

2. Heuristic (if any): **H1**

3. Did it find least-cost path? If not, why? **No. A* is only guaranteed to find an optimal path when the heuristic is admissible (or consistent with a strict expanded list). H1 is neither: the heuristic value for C is not an underestimate of the optimal cost to goal.**

G4: 1. Algorithm: **Best First Search**

2. Heuristic (if any): **H2**

3. Did it find least-cost path? If not, why? **Yes. Though best first search is not guaranteed to find an optimal path, in this case it did.**

G5: 1. Algorithm: **Depth First Search**

2. Heuristic (if any): **None**

3. Did it find least-cost path? If not, why? **No. Depth first search is an any-path search; it does not consider link cost at all.**

G6: 1. Algorithm: **A***

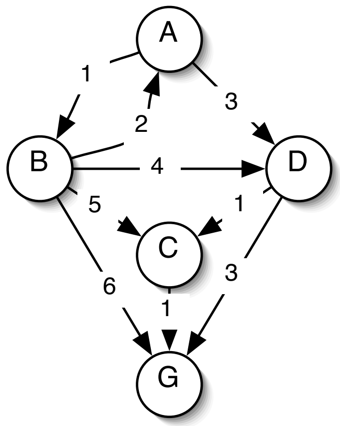
2. Heuristic (if any): **H2**

3. Did it find least-cost path? If not, why? **Yes. A* is guaranteed to find an optimal path when the heuristic is admissible (or consistent with a strict expanded list). H2 is admissible but not consistent, since the link from D to C decreases the heuristic cost by 2, which is greater than the link cost of 1. Still, the optimal path was found.**

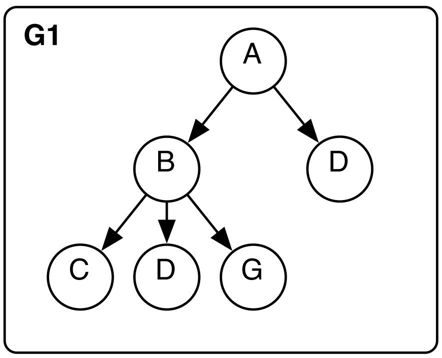
G7: 1. Algorithm: **Uniform Cost Search**

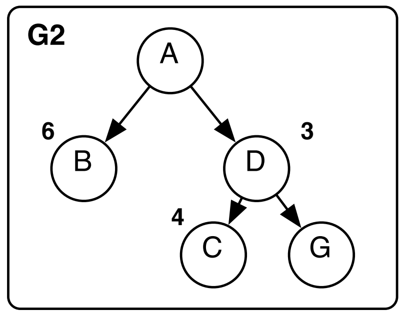
2. Heuristic (if any): **None**

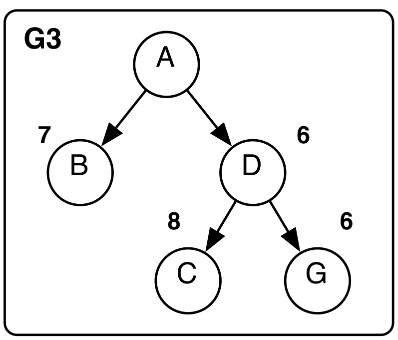
3. Did it find least-cost path? If not, why? **Yes. Uniform Cost is guaranteed to find a shortest path.**

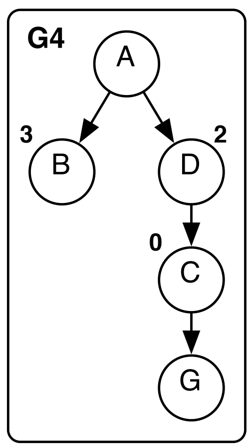


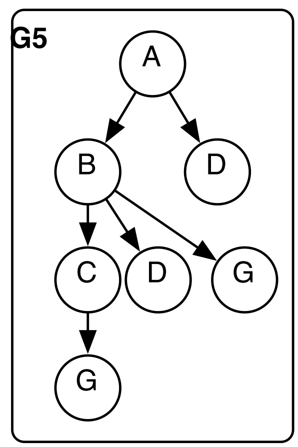
	H1	H2
A	3	3
B	6	3
C	4	0
D	3	2

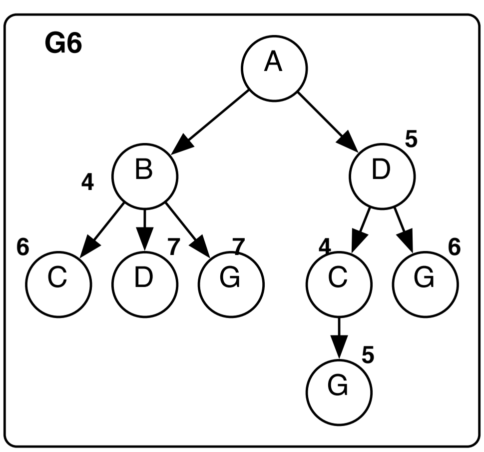


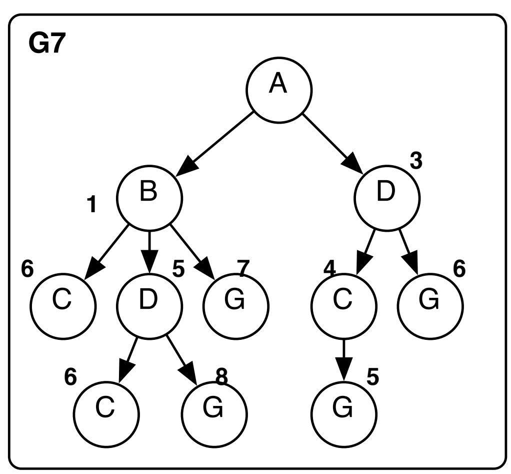






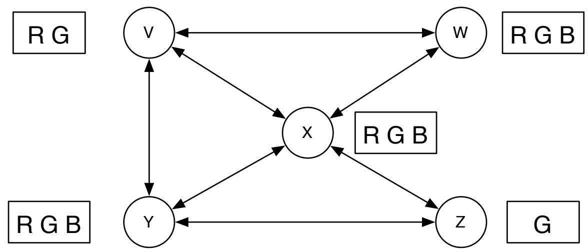






4 Constraint Satisfaction (20 points)

Consider the following constraint graph for a graph coloring problem (the constraints indicate that connected nodes cannot have the same color). The domains are shown in the boxes next to each variable node.



1. What are the variable domains after a full constraint propagation?

Variable	Domain
V	R G
W	R G B
X	R B
Y	R B
Z	G

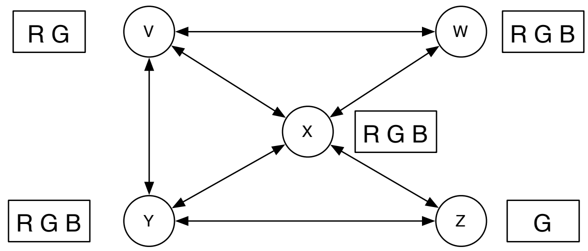
2. What can you conclude about the existence of a solution based on this result? Explain.

   Not much. Since there are multiple values in most of the domains, there could be zero, one or many solutions.

3. Show the sequence of variable assignments during a pure backtracking search (do not assume that the propagation above has been done), assume that the variables are examined in alphabetical order and the values are assigned in the order shown next to each node. Show assignments by writing the variable name and the value in the table below. Don't write more than 12 assignments, even if it would take more to find a consistent answer.

   Show only assignments that satisfy all the constraints up to and including that level of the backtracking search.

Step	Variable	Value		Step	Variable	Value
1	V	R	———	7	Y	B
2	W	G	———	8	V	G
3	X	B	———	9	W	R
4	Y	G	———	10	X	B
5	W	B	———	11	Y	R
6	X	G	———	12	Z	G



4. Show the sequence of variable assignments during backtracking with forward checking, assume that the variables are examined in alphabetical order and the values are assigned in the order shown next to each node. Show assignments by writing the variable name and the value in the table below. Don't write more than 12 assignments, even if it would take more to find a consistent answer.

Show assignment before forward checking is performed.

Step	Variable	Value		Step	Variable	Value
1	V	R	——	7	V	G
2	W	G	——	8	W	R
3	X	B	——	9	X	B
4	Y	G	——	10	Y	R
5	W	B	——	11	Z	G
6	X	G	——	12		

5 Non-Binary Constraints I (20 pts)

We have limited our investigation of constraint satisfaction problems to those involving binary constraints. However, many constraints in the real world are not naturally binary. A simple example is finding satisfying assignments for a propositional formula in conjunctive normal form (CNF). An example of such a formula is:

$$(A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C) \wedge (A \vee B \vee C)$$

1. Assuming that we pick the propositional variables in a CNF formula as the variables of a CSP, in general, what are the constraints?

   Each disjunction in the CNF formula is a constraint. It says that at least one of the variables in the disjunction must be false if the variable appears negated and true otherwise.

2. Describe how this formulation can be transformed into a binary CSP formulation.

   Each disjunction becomes a variable, whose domain is the set of assignments to the propositional variables that would make that disjunction true. The constraints would require that the propositional variables have the same value in each such assignment.

3. Illustrate your binary formulation of CNF satisfiability on the example sentence above. Make sure that you identify the variables, domains and constraints.

   *The variables and domains would be as follows. Each tuple in the domain is a single value for the new variables. We've used a shorthand introducing a "don't care" value * to stand for T or F, so each one of these tuples with 2 don't cares stands for 4 actual values.*
   $V_1 = (A \vee \neg B \vee C), D_1 = < T, *, * >, < *, F, * >, < *, *, T >$
   $V_2 = (A \vee B \vee \neg C), D_2 = < T, *, * >, < *, T, * >, < *, *, F >$
   $V_3 = (A \vee B \vee C), D_3 = < T, *, * >, < *, T, * >, < *, *, T >$
   Then, we have a constraint between each pair of V_1, V_2 and V_3 that requires the bindings for A, B and C be compatible, for example, between V_1 and V_2, the assignment to A must be the same in both, the assignment to B must be different in both and the assignment to C must also be different.

4. The transformation on a non-binary CSP problem into a binary one is an example of a general method. Outline briefly the key idea of this general method?

   Let the constraints be the variables and require shared variables to agree. We also saw this approach in the line-labeling problem.

6 Non-Binary Constraints II (10 pts)

This problem is not limited to CNF satisfiability.

An alternative approach to dealing with non-binary constraints is to adapt our programs for solving CSP so as to be able to deal with non-binary constraints. For each of the following three methods for solving CSPs, outline what would need to be changed to handle constraints involving 3 variables. Think carefully, in some cases the changes are very minor. There may be more than one way to extend the methods, write down one.

1. Min-Conflict Hill Climbing

   We always have a full assignment in hand, so all we need to do is be able to check whether the constraints (binary or not) are satisfied and so no real extension is necessary.

2. Backtracking

   The change is minor. At depth d in the tree we have d tentative assignments to variables, we have to check that all the constraints involving only those d variables are satisfied. Note that we cannot start checking constraints until we have enough assignments, which for 3 variable constraints means that we cannot start checking until level 3 in the tree.

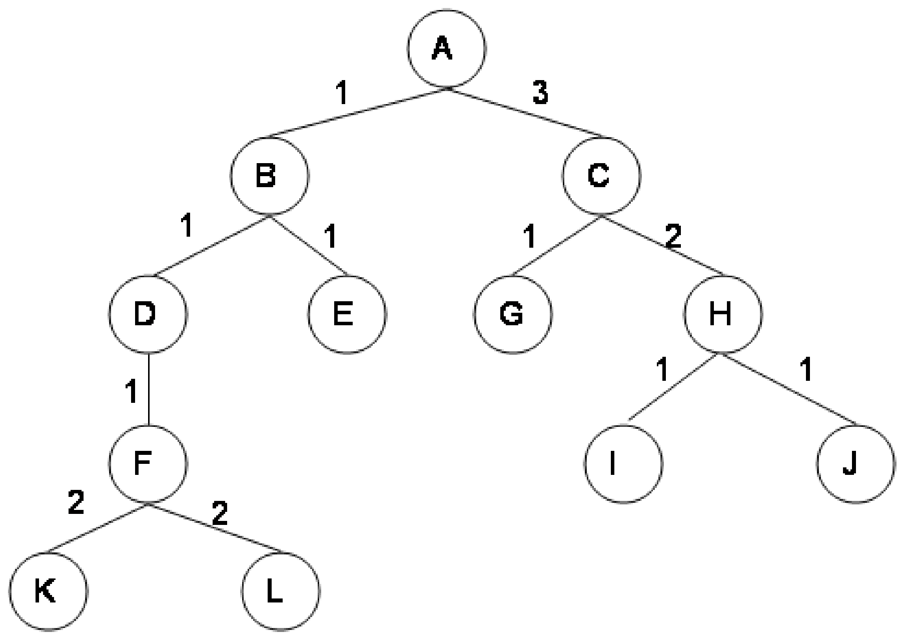
3. Backtracking with Forward Checking

   This is the slightly tricky one. At level d in the tree we have a set of tentative assignments, we want to eliminate from the domains of any future (not yet assigned) variables any values that would conflict with the assigned values. Usually, this would be for constraints involving two assigned variables and one unassigned variable.

6.034 Quiz 1, Spring 2004 — Solutions
Open Book, Open Notes

1 Tree Search (12 points)

Consider the tree shown below. The numbers on the arcs are the arc lengths.

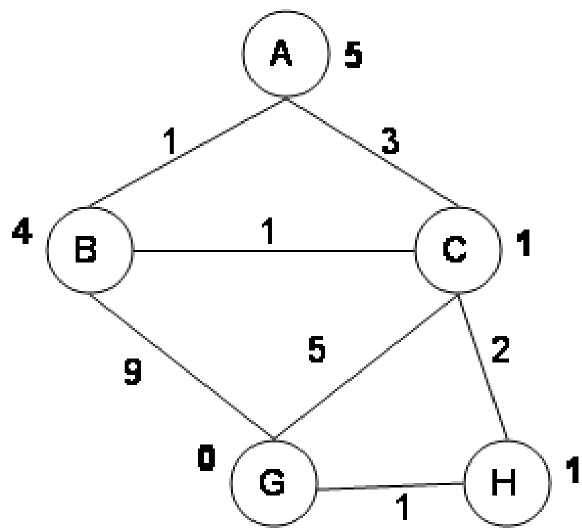


Assume that the nodes are expanded in alphabetical order when no other order is specified by the search, and that the goal is state G. No visited or expanded lists are used. What order would the states be expanded by each type of search? Stop when you expand G. Write only the sequence of states expanded by each search.

Search Type	List of states
Breadth First	A B C D E G
Depth First	A B D F K L E C G
Progressive Deepening Search	A A B C A B D E C G
Uniform Cost Search	A B D E C F G

2 Graph Search (10 points)

Consider the graph shown below where the numbers on the links are link costs and the numbers next to the states are heuristic estimates. Note that the arcs are undirected. Let A be the start state and G be the goal state.



Simulate A* search with a strict expanded list on this graph. At each step, show the path to the state of the node that's being expanded, the length of that path, the total estimated cost of the path (actual + heuristic), and the current value of the expanded list (as a list of states). You are welcome to use scratch paper or the back of the exam pages to simulate the search. However, please transcribe (only) the information requested into the table given below.

Path to State Expanded	Length of Path	Total Estimated Cost	Expanded List
A	0	5	(A)
C-A	3	4	(C A)
B-A	1	5	(B C A)
H-C-A	5	6	(H B C A)
G-H-C-A	6	6	(G H B C A)

3 Heuristics and A* (8 points)

1. Is the heuristic given in Problem 2 admissible? Explain.

   Yes. The heuristic is admissible because it is less than or equal to the actual shortest distance to the goal.

2. Is the heuristic given in Problem 2 consistent? Explain.

   No, the heurstic is not consistent. There are two places in the graph where consistency fails. One is between A and C where the drop in heuristic is 4, but the path length is only 3. The other is between B and C where the drop in heuristic is 3 but the path length is only 1.

3. Did the A* algorithm with strict expanded list find the optimal path in the previous example? If it did find the optimal path, explain why you would expect that. If it didn't find the optimal path, explain why you would expect that and give a simple (specific) change of state values of the heuristic that would be sufficient to get the correct behavior.

   A with a strict expanded list will not find the shortest path (which is ABCHG with cost 5). This is because the heuristic is not consistent. We can make the heuristic consistent by changing its value at C to be 3. There are other valid ways to make the graph consistent (change h(B) to 2 and h(A) to 3, for example) and those were right as well.*

4 Search problem formulation (10 points)

A Mars rover has to leave the lander, collect rock samples from three places (in any order) and return to the lander.

Assume that it has a navigation module that can take it directly from any place of interest to any other place of interest. So it has primitive actions *go-to-lander*, *go-to-rock-1*, *go-to-rock-2*, and *go-to-rock-3*.

We know the time it takes to traverse between each pair of special locations. Our goal is to find a sequence of actions that will perform this task in the shortest amount of time.

1. Formulate this problem as a search problem by specifying the state space, initial state, path-cost function, and goal test. Try to be sure that the state space is detailed enough to support solving the problem, but not redundant.

   - States: ⟨ *current-location, have-rock1?, have-rock2?, have-rock3?* ⟩

     These are state *variables*. The variable *current-location* ranges over the set {*lander, rock1, rock2, rock3* }. The other variables are binary.
   - Initial state: ⟨ *lander, no, no, no* ⟩
   - Path cost: sum of arc costs; arc cost = distance between locations
   - Goal test: ⟨ *lander, yes, yes, yes* ⟩

2. Say what search technique would be most appropriate, and why.

   We want a shortest path, so we need UCS or A. We might as well use A*, since it will probably be faster and there's a reasonable heuristic available.*

3. One possible heuristic evaluation function for a state would be the distance back to the lander from the location of the state; this is clearly admissible. What would be a more powerful, but still admissible, heuristic for this problem? (Don't worry about whether it's consistent or not.)

   This should have read "One possible heuristic evaluation function for a state would be the **amount of time** *required for the robot to go back to the lander from the location of the state..."*

   So, because of the typo, we gave everyone a free two points on this problem.

   The answer we had in mind was the maximum, over uncollected rocks r, of the time to get from the current location to r, and the time to get from r to the lander.

5 CSP (17 points)

Let's look at the problem of scheduling programs on a set of computers as a constraint satisfaction problem.

We have a set of programs (jobs) J_i to schedule on a set of computers (machines) M_j. Each job has a maximum running time R_i. We will assume that jobs (on any machines) can only be started at some pre-specified times T_k. Also, there's a T_{max} time by which all the jobs must be finished running; that is, start time + running time is less than or equal to max time. For now, we assume that any machine can execute any job.

Let's assume that we attack the problem by using the jobs as variables and using values that are each a pair (M_j, T_k). Here is a simple example.

- Running time of J_1 is $R_1 = 2$
- Running time of J_2 is $R_2 = 4$
- Running time of J_3 is $R_2 = 3$
- Running time of J_4 is $R_4 = 3$
- Starting times $T_k = \{1, 2, 3, 4, 5\}$
- Two available machines M_1 and M_2.
- The max time is $T_{max} = 7$.
- An assignment would look like $J_1 = (M_2, 2)$, that is, run job J_1 on machine M_2 starting at time 2.

1. What are the constraints for this type of CSP problem? Write a boolean expression (using logical connectives and arithmetic operations) that must be satisfied by the assignments to each pair of variables. In particular:

   - J_i with value (M_j, T_k)
   - J_m with value (M_n, T_p)

   There is a unary constraint on legal values for a single variable: $T_k + R_i \leq T_{max}$. *This is not a binary constraint on pairs of values.*

   The binary constraint is the one that says that jobs on the same machines must not overlap in time. It can be expressed as:

   $$M_j = M_n \rightarrow T_k + R_i \leq T_p \vee T_p + R_m \leq T_k$$

   So, either the machines are different or the times don't overlap.

2. Write down a complete valid solution to the example problem above.

   - $J_1 = (M_1, 1)$
   - $J_2 = (M_1, 3)$
   - $J_3 = (M_2, 1)$
   - $J_4 = (M_2, 4)$

   Several other answers are also legal.

3. Which variable would be chosen first if we did BT-FC with dynamic ordering of variables (most constrained)? Why?

   J_2 would be chosen since it has the smallest domain of legal values. That job since it takes 4 time steps can only be started at times less than or equal to 3 so that it will finish before $T_{max} = 7$.

4. If we do constraint propagation in the initial state of the example problem, what domain values (if any) are eliminated? Explain.

   If one assumes that domain values inconsistent with the unary (T_{max}) constraint have been eliminated from the domains before constraint propagation, then no further domain values are eliminated. We can always run a pair of jobs on different machines and so the binary constraints do not reduce the domain further. Many people assumed that the unary constraints were checked durning propagation and we allowed that.

5. If we set $J_2 = (M_1, 1)$, what domain values are still legal after forward checking?

   - $J_1 \in (M_1, 5), (M_2, t) t \in \{1, ..., 5\}$
   - $J_2 \in (M_1, 1)$
   - $J_3 \in (M_2, t) t \in \{1, ..., 4\}$
   - $J_4 \in (M_2, t) t \in \{1, ..., 4\}$

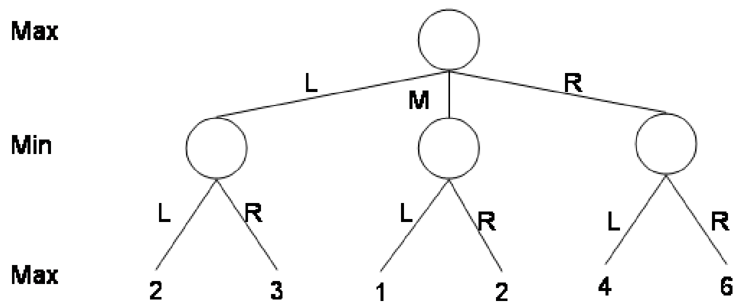
6. We could have formulated this problem using the machines M_j as the variables. What would the values be in this formulation, assuming you have N machines and have K jobs to schedule?

   A value would be a complete schedule for each machine, that is, a list of all the jobs to run on the machine. One could also specify the starting times of each job but that's redundant, since the running time could be used.

7. What are some disadvantages of this formulation (using machines as variables)? *There would be an very large number of possible values in the domain of each variable (every way of splitting K jobs among M machines so that the sum of the running times is less than T_{max}).*

6 Game Search (10 points)

Consider the game tree shown below. The top node is a max node. The labels on the arcs are the moves. The numbers in the bottom layer are the values of the different outcomes of the game to the max player.



1. What is the value of the game to the max player?

   4

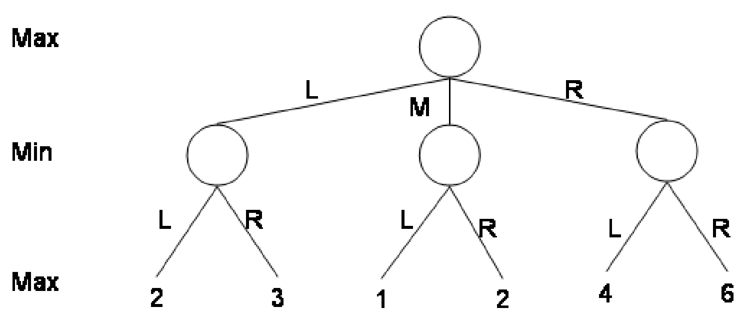
2. What first move should the max player make?

   R

3. Assuming the max player makes that move, what is the best next move for the min player, assuming that this is the entire game tree?

   L

4. Using alpha-beta pruning, consider the nodes from **right to left**, which nodes are cut off? Circle the nodes that are not examined.

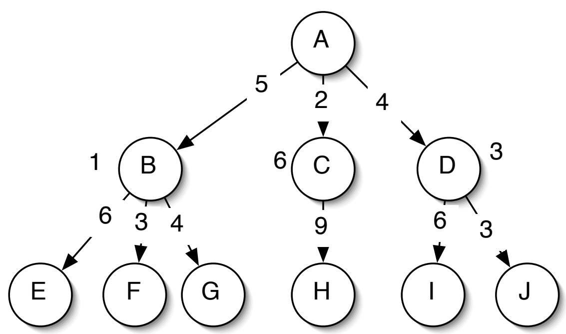


The nodes that are not examined are the left-most node labeled "2" and the node labeled "1."

6.034 Quiz 1, Spring 2003: Solutions v. 1.1
Open Book, Open Notes

1 Tree Search (10 points)

Consider the tree shown below. The numbers on the arcs are the arc lengths; the numbers near states B, C, and D are the heuristic estimates; all other states have a heuristic estimate of 0.

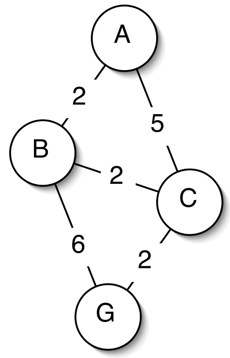


Assume that the children of a node are expanded in alphabetical order when no other order is specified by the search, and that the goal is state J. No visited or expanded lists are used. What order would the states be expanded by each type of search. Write only the sequence of states expanded by each search.

Search Type	List of states
Breadth First	A B C D E F G H I J
Depth First	A B E F G C H D I J
Progressive Deepening Search	A A B C D A B E F G C H D I J
Best-First Search	A B E F G D I J
A* Search	A B D J

2 Graph Search (8 points)

Consider the graph shown below. Note that the arcs are undirected. Let A be the start state and G be the goal state.



Simulate uniform cost search with a strict expanded list on this graph. At each step, show the state of the node that's being expanded, the length of that path, and the current value of the expanded list (as a list of states).

State Expanded	Length of Path	Expanded List
A	0	(A)
B	2	(B A)
C	4	(C B A)
G	6	(G C B A)

3 $A*$ Algorithm (12 points)

1. Let's consider three elements in the design of the $A*$ algorithm:

   - The heuristic, where the choices are:
     - **arbitrary** heuristic
     - **admissible** heuristic
     - **consistent** heuristic
   - History:
     - **none**
     - **visited** list
     - **strict** expanded list
     - **non-strict** expanded list
   - Pathmax
     - **Use** pathmax
     - **Don't use** pathmax

   In the table below, indicate all the combinations that *guarantee* that $A*$ will find an optimal path. Not all rows have to be filled. If multiple values works for any of Heuristic, History and Pathmax, independent of the other choices, you can write the multiple values in one row. So

Heuristic	History	Pathmax
A,B	C	D,E

   can be used to represent all of: A,C,D; A,C,E; B,C,D; and B,C,E.

Heuristic	History	Pathmax
Admissible	None, Non-Strict	Use, Don't Use
Consistent	None, Non-Strict, Strict	Use, Don't Use

2. In the network of problem 2, assume you are given the following heuristic values:

$$A = 5; B = 4; C = 0; G = 0$$

   Is this heuristic:

   - Admissible? $\boxed{\textbf{Yes}}$ No

   - Consistent? Yes $\boxed{\textbf{No}}$

   Justify your answer very briefly.

   It is admissible because it is always less than the length of the shortest path. It is not consistent because the difference between the heuristic values at B and C is 4, which is greater than the arc-length of 2.

3. With the heuristic above will A* using a strict expanded list find the optimal path?

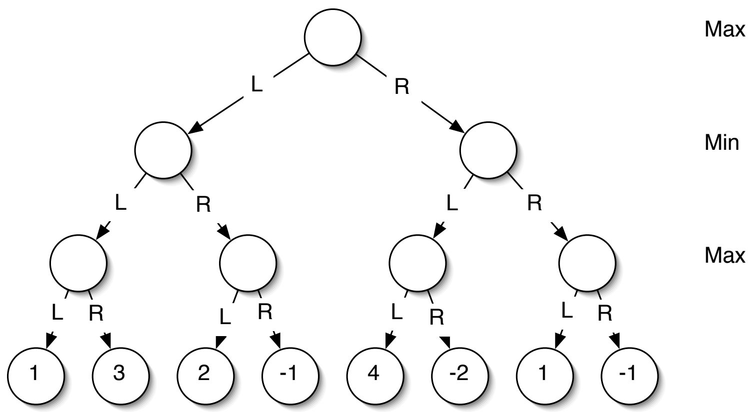
   Yes $\boxed{\textbf{No}}$

   Justify your answer very briefly.

   We will visit C first from A with estimated cost of 5, and because it's on the expanded list, even when we later find a path to C with estimated cost of 4, we won't expand it again.

4 Game Search (5 points)

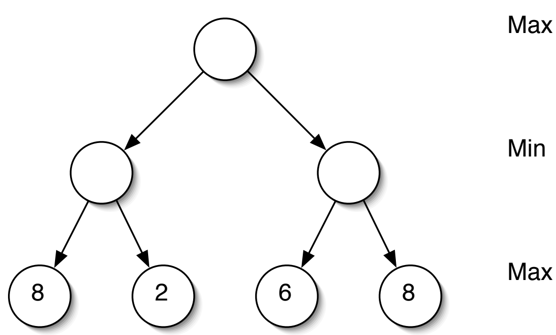
Consider the game tree shown below. Assume the top node is a max node. The labels on the arcs are the moves. The numbers in the bottom layer are the values of the different outcomes of the game to the max player.



1. What is the value of the game to the max player? **2**
2. What first move should the max player make? **L**
3. Assuming the max player makes that move, what is the best next move for the min player, assuming that this is the entire game tree? **R**

5 Alpha-Beta Pruning (5 points)

In the following game tree, are there any alpha-beta cutoffs?



- Consider the nodes from left to right, which nodes are cutoff? Circle the nodes that are not examined and label them with L. **None**.

- Consider the nodes from right to left, which nodes are cutoff? Circle the nodes that are not examined and label them with R. **The leftmost 8 node**.

6 CSP Methods (15 points)

Let's consider some combinations of CSP methods. For each of the combinations described below say very briefly whether:

1. It would be **well-defined** to combine them, in the sense that none of the implementation assumptions of the methods as we defined them are violated in the combination.
2. It could be **useful**, that is, one would expect improved performance (over using only the first method mentioned), at least in some problems. Improved performance could be either from being better able to solve problems or improved efficiency (indicate which).

In each case, circle Yes or No for each of Well-Defined? and Useful? and give a very brief explanation of your answers.

Warning: Please pay careful attention to the definition of the methods being combined, we are refering to the original definition of the methods – in isolation. Almost any idea can be made to work with any other idea with sufficient creativity - but that's not what we are looking for in this problem.

- Full constraint propagation (CP) followed by pure backtracking (BT).
  1. Well-Defined? $\boxed{\textbf{Yes}}$ No
  2. Useful? $\boxed{\textbf{Yes}}$ No

     After full CP, there may still be multiple solutions, and BT will choose one.

- Full constraint propagation (CP) combined with forward checking (FC).
  1. Well-Defined? Yes $\boxed{\textbf{No}}$
  2. Useful? Yes No

     This doesn't make sense; you still need to do some kind of search. Having done CP, FC won't rule out any more options, and you're may be left with multiple possible solutions.

- Pure backtracking (BT) combined with dynamic variable (most constrained) and value ordering (least constraining).

  1. Well-Defined? Yes $\boxed{\textbf{No}}$
  2. Useful? Yes No

     Dynamic variable and value ordering only make sense if you're doing FC to discover changes in legal variable domains.

- Min-conflict-hill-climb (MC) combined with dynamic variable (most constrained) and value ordering (least constraining).

  1. Well-Defined? Yes $\boxed{\textbf{No}}$
  2. Useful? Yes No

     MC always works with a complete assignment of values to variables.

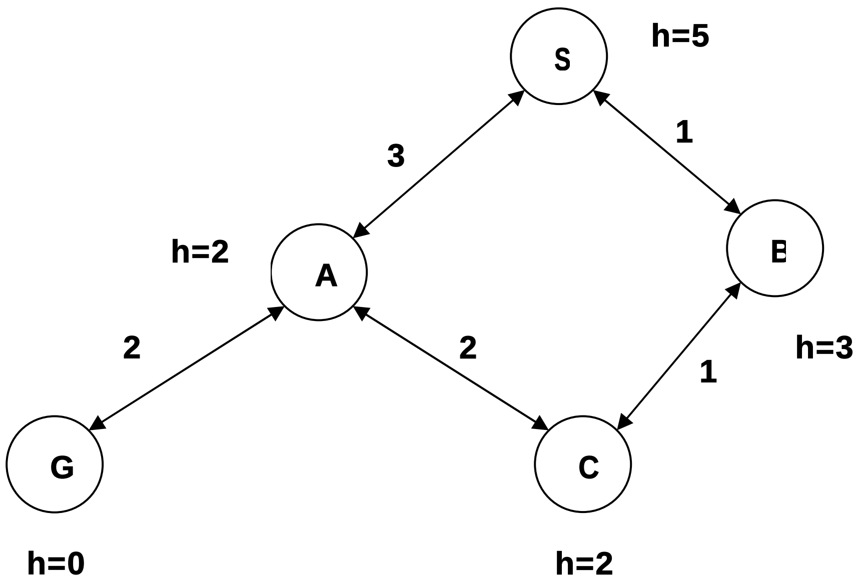
- Pure backtracking (BT) combined with full constraint propagation (CP) after each tentative assignment.

  1. Well-Defined? $\boxed{\textbf{Yes}}$ No
  2. Useful? $\boxed{\textbf{Yes}}$ No

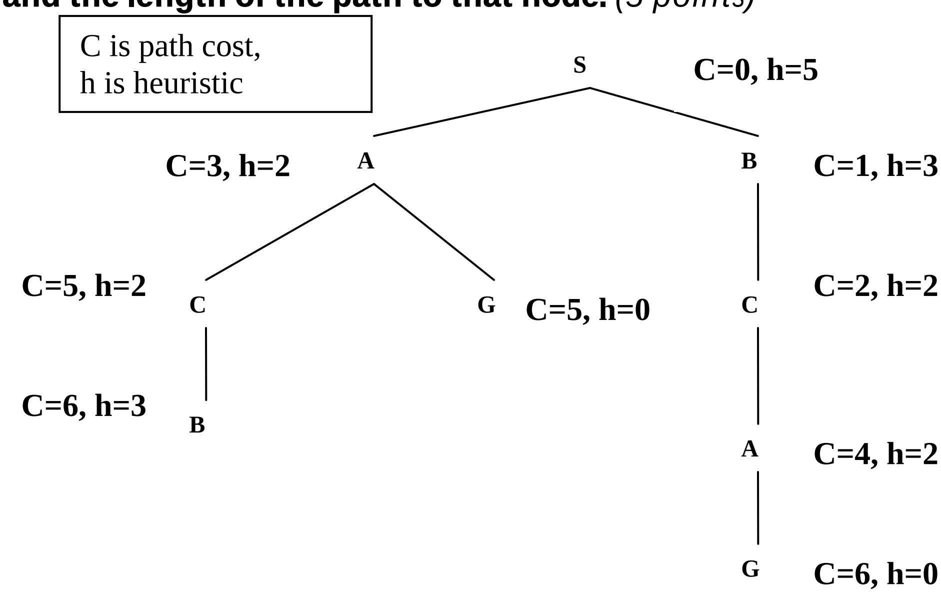
     Although full CP is expensive, uninformed BT can be even worse; so, in some cases, this is an improvement.

Problem 1 – Search *(30 points)*

Below is a graph to be searched (starting at S and ending at G). Link/edge costs are shown as well as heuristic estimates at the states. You may not need all the information for every search.



Draw the complete search tree for this graph. Label each node in the tree with the cost of the path to that node and the heuristic cost at that node. When you need to refer to a node, use the name of the corresponding state and the length of the path to that node. *(5 points)*



S

For each of the searches below, just give a list of node names (state name, length of path) drawn from the tree above. Break ties using alphabetical order. *(2 points each)*

1. **Perform a depth-first search using a visited list. Assume children of a state are ordered in alphabetical order. Show the sequence of nodes that are expanded by the search.**

   S0, A3, C5, G5 note that B6 is not expanded because B is on visited list (placed there when S0 was expanded).

2. **Perform a best-first (greedy search) without a visited or expanded list. Show the sequence of nodes that are expanded by the search.**

   S0 (h=5), A3(h=2), G5(h=0)

3. **Perform a Uniform Cost Search without a visited or expanded list. Show the sequence of nodes that are expanded by the search.**

   S0, B1, C2, A3, A4, C5, G5 note that nodes are ordered first by cost then alphabetically when tied for cost.

4. **Perform an A* search (no pathmax) without an expanded list. Show the sequence of nodes that are expanded by the search.**

   S0(0+5), B1(1+3), C2(2+2), A3(3+2), G5(5+0)

Is the heuristic in this example

1. **admissible?** *Yes*

2. **consistent?** *No*

Justify your answer, briefly. *(3 points)*

All the h values are less than or equal to actual path cost to the goal and so the heuristic is admissible.
The heuristic drops from 5 at S to 3 at B while the path cost between S and B is only 1, and so the heuristic is not consistent.

For each of the following situations, pick the search that is most appropriate (be specific about visited and expanded list). Give a one sentence reason why you picked it. If you write a paragraph, we will not read it.

1. **We have a very large search space with a large branching factor and with possibly infinite paths. We have no heuristic. We want to find paths to the goal with minimum numbers of state.**

   Iterative deepening is the best choice, it uses little memory (like DFS) but guarantees finding the path with minimum number of states (like BFS).

2. **We have a space with a manageable number of states but lots of cycles in the state graph. We have links of varying costs but no heuristic and we want to find shortest paths.**

   Uniform Cost Search with a strict expanded list is the best choice, it guarantees finding shortest paths and the expanded list limits the cost to a function of the number of states, which is reasonable in this case. Recall that a visited list will interfere with the correct operation of UCS.

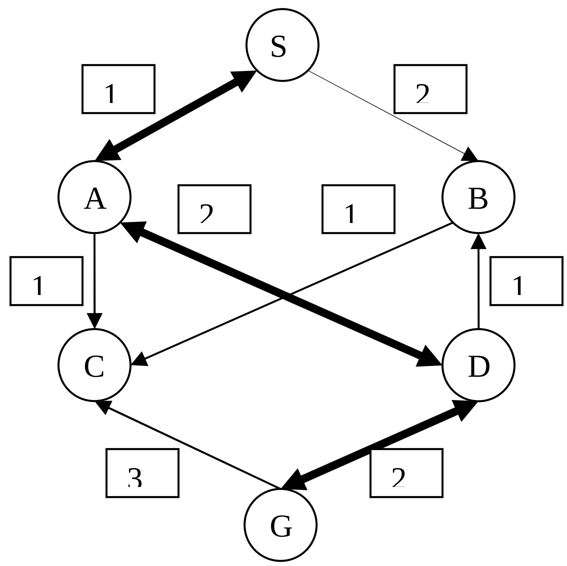
3. **Our search space is a tree of fixed depth and all the goals are the leaves of the tree. We have a heuristic and we want to find any goal as quickly as possible.**

   This has a typo which makes it ambiguous. If you read it as "all the leaves are goals", then depth-first search is the best choice (gets to the leaves fastest). If you read it as "all the goals are at the leaves", then the best choice is a greedy search (best first), which uses the heuristic to guide you to the part of the tree with the goals. In neither case is a visited or expanded list advisable since we are searching a tree (no loops).

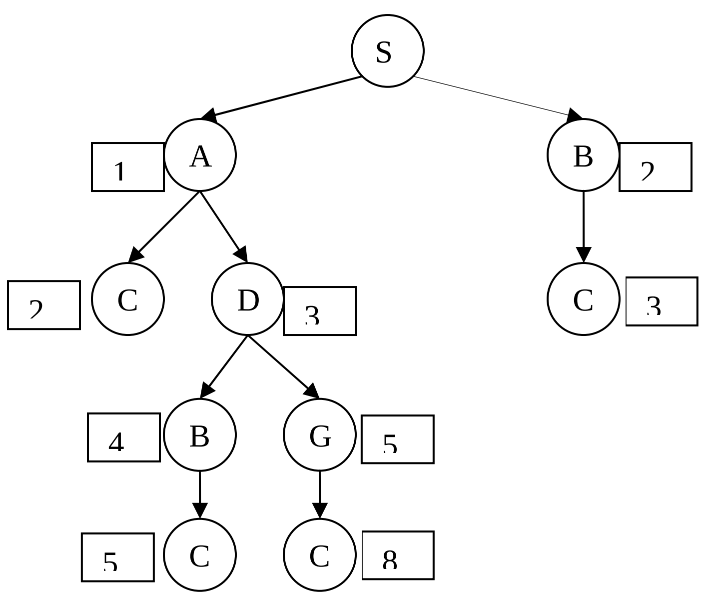
4. **We have a space with a manageable number of states but lots of cycles in the state graph. We have links of varying costs and an admissible heuristic and we want to find shortest paths.**

   This calls for A and a non-strict expanded list and, since we don't know that the heuristic is consistent, using pathmax. This allows us to use all the information we have and to avoid the extra cost due to cycles.*

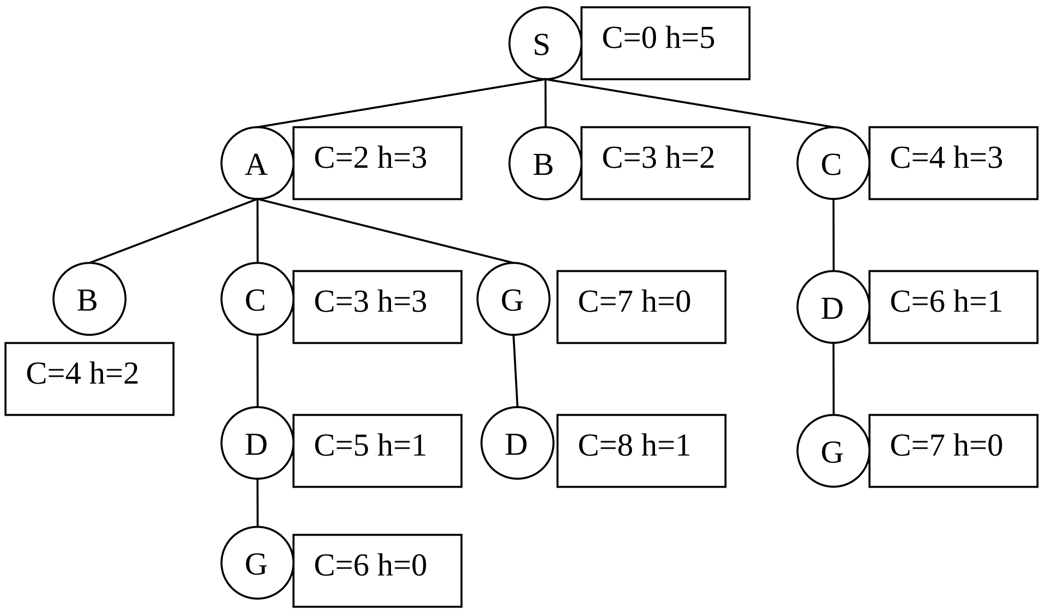
Problem 1: Search (25 points)



A. Construct the search tree for the graph above, indicate the path length to each node. The numbers shown above are link lengths. Pay careful attention to the arrows; some are bi-directional (shown thick) while some are uni-directional.



B. Using the following search tree, perform the searches indicated below (always from S to G). Each node shows both the total path cost to the node as well as the heuristic value for the corresponding state.



For each of the searches below, write the sequence of nodes **expanded** by the search. Specify a node by writing the name of the state and the length of the path (C above), e.g. S0, B3, etc. Break ties using alphabetical order.

1. Depth First Search (no visited list)

   S0, A2, B4, C3, D5, G6

2. Breadth First Search (with visited list)

   S0, A2, B3, C4, G7

3. Uniform Cost Search (with strict expanded list)

   S0, A2, B3, C3, D5, G6

4. A* (without expanded list)

   S0(+5), A2(+3), B3(+2), B4(+2), C3(+3), D5(+1), G6(+0)

C. Choose the most efficient search method that meets the criteria indicated below. **Explain your choice.**

1. You are given a state graph with link costs. The running time of the algorithm should be a function of the number of states in the graph and the algorithm should guarantee that the path with shortest path cost is found.

   UCS + expanded list
   UCS guarantees shortest paths, expanded list makes sure that the running time depends only on the number of states not the number of paths.

2. You are given a state graph with link costs and consistent heuristic values on the states. The running time of the algorithm should be a function of the number of states in the graph and the algorithm should guarantee that the path with shortest path cost is found.

   A* + expanded list
   A* with consistent heuristic guarantees shortest paths, expanded list keeps the running time a function of number of states.

3. You are given a state graph with no link costs or heuristic values. The algorithm should find paths to a goal with the least number of states and the space requirements should depend on the depth of the first goal found.

   Iterative deepening

Guarantees minimum number of states on path to goal and the memory requirements are determined by the last depth-first search (at the level of the first goal found).

Problem 5 – CSP *(12 points)*

Assume we have four variables (A, B, C, D) and two values (1, 2). We write variable/value assignments as A1, B2, etc. Assume the only legal values are as listed below:

- **A-B: A1-B1, A2-B1, A2-B2**
- **A-C: A1-C2, A2-C1**
- **A-D: A2-D2**
- **B-C: B1-C2, B2-C1**
- **B-D: B2-D2**
- **C-D: C1-D1, C1-D2**

An entry in the matrix below indicates a consistent assignment. This is simply another way of presenting the same information in the list above.

	A1	A2	B1	B2	C1	C2	D1	D2
A1			X			X		
A2			X	X	X			X
B1	X	X				X		
B2		X			X			X
C1		X		X			X	X
C2	X		X					
D1					X			
D2		X		X	X			

Assume you do full constraint propagation in this problem. Show the legal values for each variable after propagation:

- **A :** *A2*
- **B :** *B2*
- **C :** *C1*
- **D :** *D2*

Here's the search tree (as in the PS):

```
                   ----------------()---------------
                   |                                |
           -------A1-------                 -------A2------
           |               |                |              |
       ---B1---        ---B2---         ---B1---       ---B2---
       |       |       |       |        |       |      |       |
     -C1-    -C2-    -C1-    -C2-     -C1-    -C2-   -C1-    -C2-
     |   |   |   |   |    |  |   |    |   |   |   |  |   |   |   |
    D1  D2  D1  D2  D1   D2 D1  D2   D1  D2  D1  D2 D1  D2  D1  D2
```

Assume that you do the backtracking with forward checking. Show the assignments in order as they are generated during the search.

A1 (FC reduces domain of D to empty, so fail)
A2 (FC reduces domain of C to C1 and domain of D to D2)
B1 (FC reduces domain of D to empty, so fail)
B2 (FC has no further effect)
C1 (FC has no further effect)
D2 (done)

What is the first solution found in the search?

A=2, B=2, C=1, D=2

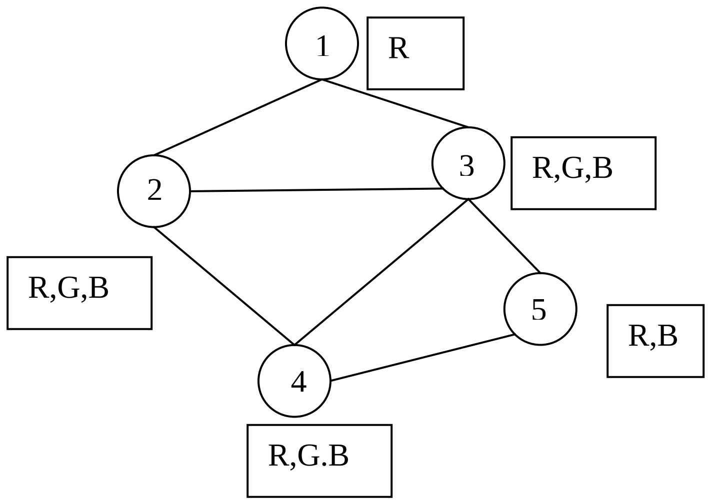
The constraints – repeated for easy reference:

- **A-B: A1-B1, A2-B1, A2-B2**
- **A-C: A1-C2, A2-C1**
- **A-D: A2-D2**
- **B-C: B1-C2, B2-C1**
- **B-D: B2-D2**
- **C-D: C1-D1, C1-D2**

	A1	A2	B1	B2	C1	C2	D1	D2
A1			X			X		
A2			X	X	X			X
B1	X	X				X		
B2		X			X			X
C1		X		X			X	X
C2	X		X					
D1					X			
D2		X		X	X			

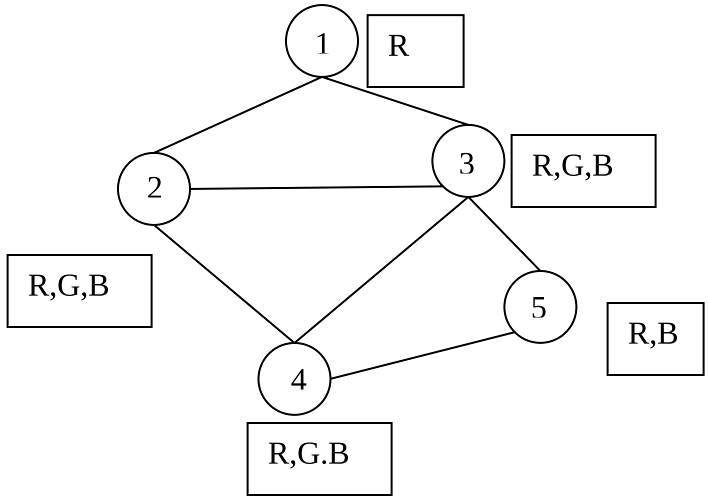
Problem 5: CSP (15 points)

Consider the following constraint graph for a graph coloring problem (the constraints indicate that connected nodes cannot have the same color). The domains are shown in the boxes next to each variable node.



1. What are the variable domains after a full constraint propagation?

   1 = {R}
   2 = {G, B}
   3 = {G, B}
   4 = {R, G, B}
   5 = {R, B}



2. Show the sequence of variable assignments during a pure backtracking search (do not assume that the propagation above has been done), assume that the variables are examined in numerical order and the values are assigned in the order shown next to each node. Show assignments by writing the variable number and the value, e.g. 1R. **Don't write more than 10 assignments, even if it would take more to find a consistent answer.**

   1R 2R 2G 3R 3G 3B 4R 5R 5B 4G [4B 2B 3R 3G 4R 5R 5B]

3. Show the sequence of variable assignments during backtracking with forward checking, assume that the variables are examined in numerical order and the values are assigned in the order shown next to each node. Show assignments by writing the variable number and the value, e.g. 1R.

   1R 2G 3B 4R 2B 3G 4R 5B