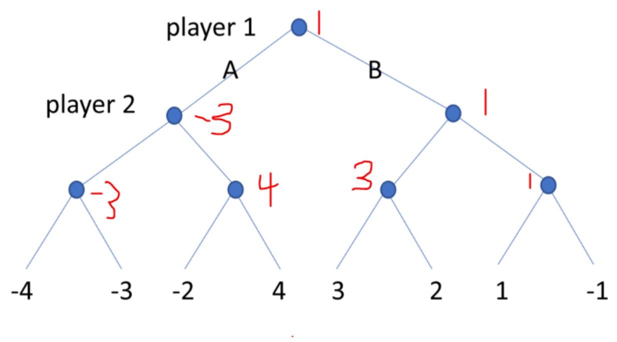
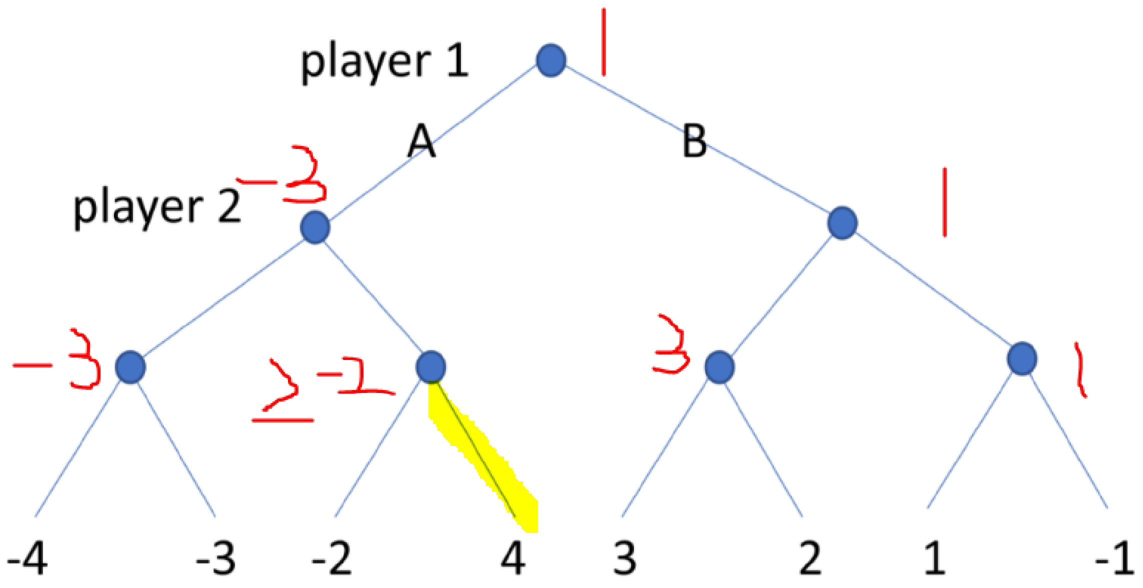
1. Compute the minimax values at the internal nodes (write the values next each node).



Should the player 1 take action A or B at the root?
Player 1 should take action B

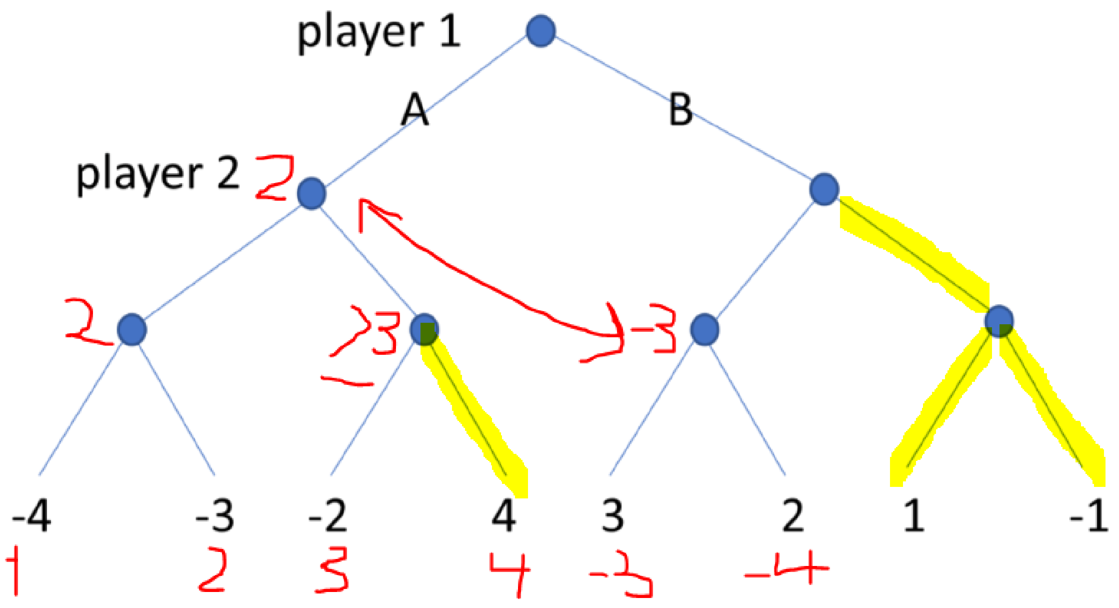
What is the expected outcome (payoff at the end of the game)?
Player one should expect a score of positive one

Which branches would be pruned by alpha-beta pruning? (circle them)



The only branch that would be pruned is the one highlighted in yellow

How could the leaves be relabeled to maximize the number of nodes pruned? (you can move the utilities around arbitrarily to other leaves, but you still have to use -4,-3,-2,-1,+1,+2,+3,+4)

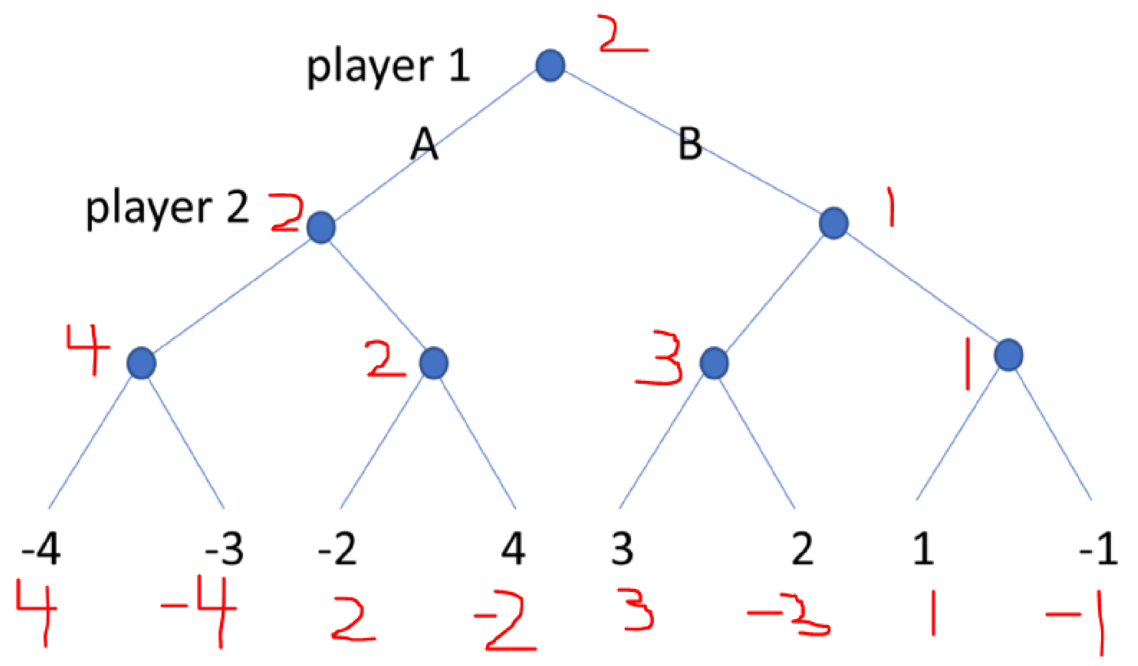


No matter what, the first three leaves on the left have to be evaluated.
So 1, 2, and 3 for the order works find to eliminate the fourth leaf.
The other nodes can be pruned if we go all the way down to the fifth leaf.
This value could literally be anything, so -3 is chosen. We still don't know if the 6th leaf
Has a value that could potentially trump 2, so we have to evaluate it.
It is less than -3, it's negative 4, so negative three goes up.
(The other two nodes can be -1, -2 or the reverse of that, it doesn't matter)

The last four nodes can be anything if they are the remaining -1, -2, -3 or -4, since by the time we evaluate any two of these first that the right side of the tree won't be taken anyways since we already have a max on the left side that player 2 chose of value 2.

Based on this, we know player two will not choose any value that is greater than -3, so it is pointless to even go down the rest of the branches since we know that player 1 will choose 2 if presented with a value less than or equal to -3.

How could the leaves be relabeled to eliminate pruning?



This configuration has no prunining

2. In a simple binary game tree of depth 4 (each player gets 2 moves), suppose all the leaves
have utility 0 except one winning state (+1000) and one loosing state (-1000).

Could the player at the root force a win?
The root player cannot force a win if there is only one leaf node that allows for the winning state. If the oppoenent has knowledge of the game, even if the player at the root chooses the right path towards the win initially, the other player can just choose a node not on the path to the win out of the two given options.

Does it matter where the 2 non-zero states are located in the tree? (e.g. adjacent or far apart)
If the leaf nodes are adjacent and the other player is player 2 (assuming root is player 1), player 2 will always win if player 1 is trying to win. This is because the path to the winning and losing condition are the same until the last chosen node, which is at the discretion of player 2. Player 1 will always lose in a minimax strategy or be at a draw if the nodes are adjacent in this way.

If this question was changed to have a different depth, would it change the answers to the
two questions above? If yes, how do the answers change? If no, explain why no change would happen.
No the answer would not change given that the last node is always at the discretion of the opponent who can choose to always pick a losing node or neutral node for the root player.

3. Consider the task of creating a crossword puzzle (choosing words for a predefined layout).
Suppose you have a finite dictionary of words (see /etc/dictionaries/ on linux distributions for
~50k English words; used for 'ispell' command).
Given a layout with a list of n "across" and m "down" words (see the example in the Figure
below, the goal is to choose words to fill in (that satisfy all the intersections between words).
(Clues for these words can then be defined later by a puzzle expert.)
Show how to model this crossword puzzle design problem as a CSP.
What are the variables, domains, and constraints?
(hint: not all variables have to have the same domain)

Variables:

- The words themselves (the lines of blocks going up or down)

Domains:

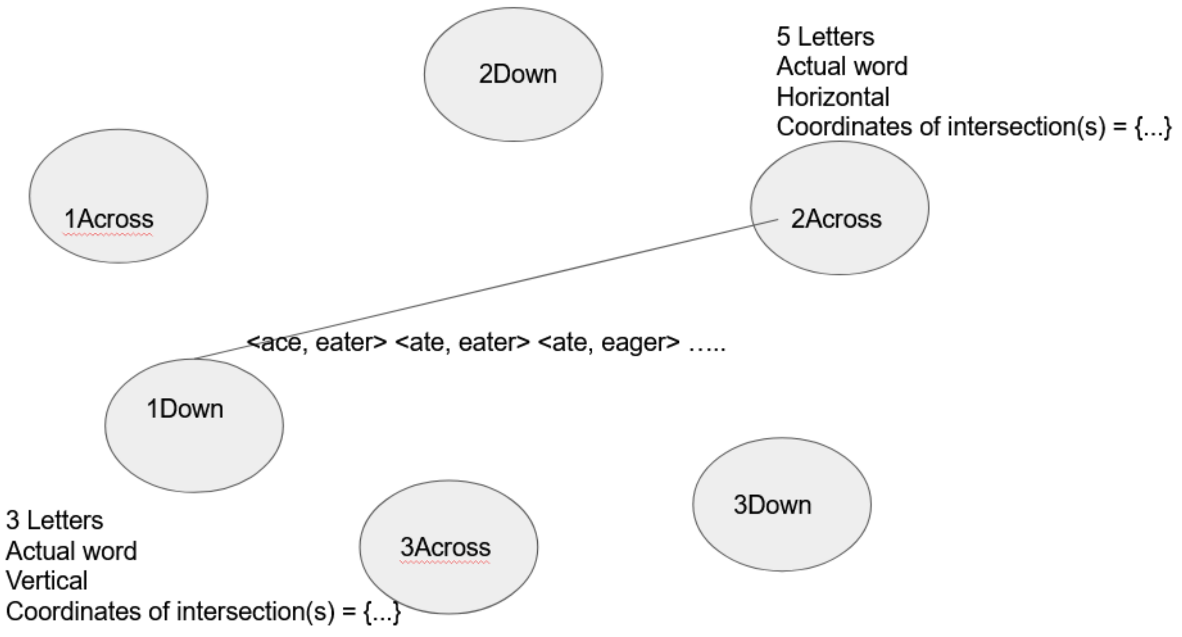
- Horizontal or vertical word
- Number of letters in word
- Coordinate(s) of intersection(s)

*some of the words have no intersections. If that is the case, the coordinates of potential intersections aren't a part of the domain

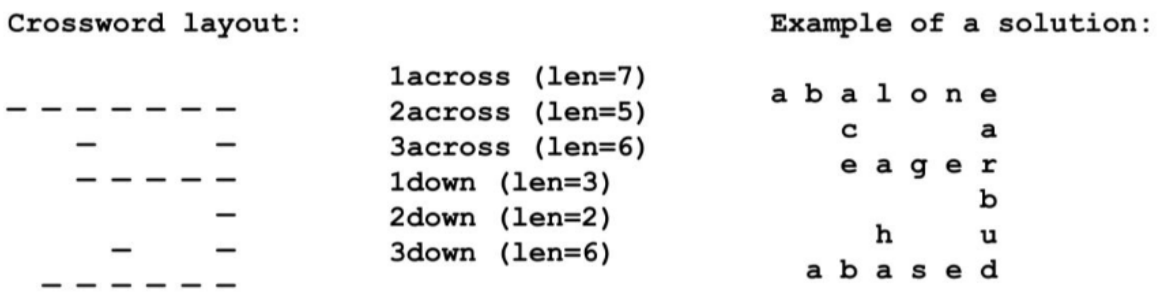
Constraints:

- The words chosen for each variable have to be actual words in addition to fitting in all the boxes of the line of boxes
- If two words share a coordinate of intersection, then the common shared coordinate must contain the same letter for both words

Draw the constraint graph.
Label one of the nodes and one of the edges as examples (the nodes are easy; the edges require
*using the example provided in the HW document:


2Down
5 Letters
Actual word
Horizontal
Coordinates of intersection(s) = {...}
1Across
2Across
<ace, eater> <ate, eater> <ate, eager> …..
1Down
3Down
3 Letters
Actual word
Vertical
Coordinates of intersection(s) = {...}
3Across

some thought). (of course, you don't have to write down the labels completely; just explain what
they would look like and give a couple examples)
Describe the first couple of steps of how standard backtracking search would work (selecting
variables and values in default order), making reasonable choices as you go. (e.g. you could
state: suppose 'ace' is the first 3-letter word starting with 'a'...).



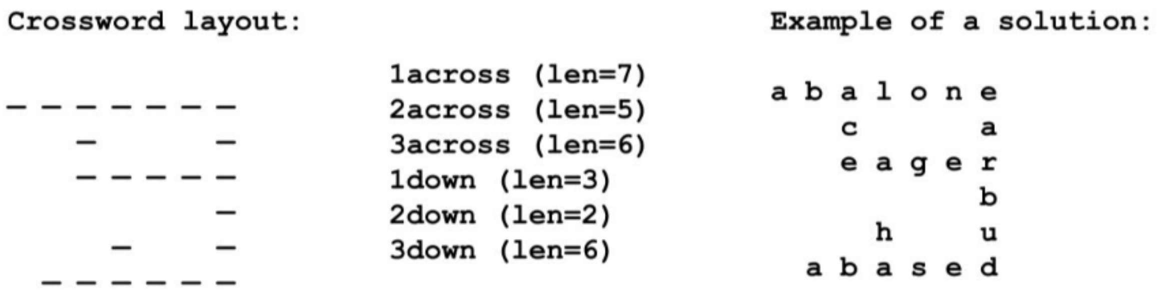
```
Crossword layout:                               Example of a solution:

                      1across (len=7)
_ _ _ _ _ _ _         2across (len=5)           a b a l o n e
    _       _         3across (len=6)               c       a
    _ _ _ _ _         1down  (len=3)                e a g e r
            _         2down  (len=2)                        b
      _     _         3down  (len=6)                  h     u
  _ _ _ _ _ _                                     a b a s e d
```

Assume the words that are used are 1down, 2across, and 3down in the example.

- The first word chosen is ace for 1 down.
- A second word, eaten is chosen for 2across
    - This word is chosen after going through all the words in alphabetical order that are 5 letters in length, so this isn't really "one" step.
- A third word for 3down needs to be chosen. It has to match coordinates such that the letter 'n' matches it's intersection. With 2across

    - All of the words that are 6 letters long are cycled through, and for the sake of argument, assume that there are no letters that are in that intersection between 2across and the third letter of 3down that allow 3down's letter to be 'n'.
- This means that 2across has to change whichever word that is chosen, since it can't possibly be the current word given there is no solution.
- So you cycle back through words that are possible for 2across in order to potentially find a new word that can actually work with 3down.
- Then the search goes on like that, etc.

Describe how using the MRV would change the search; describe the first couple of steps again.
(hint: you might need to make some assumptions about how many words have 3 letters, 4 letters,
etc, or how many 5-letter words start with 'a' or 'y'...)



```
Crossword layout:                                    Example of a solution:

                        1across (len=7)
_ _ _ _ _ _ _           2across (len=5)              a b a l o n e
  _       _             3across (len=6)                c       a
  _ _ _ _ _             1down  (len=3)                 e a g e r
          _             2down  (len=2)                         b
    _     _             3down  (len=6)                   h     u
 _ _ _ _ _ _                                             a b a s e d
```

Again, using the crossword example,
(this is assuming that shorter words have less options allowed to them)

- The MRV variable first would be 2 down. The first word is chosen to be ha
- The second MRV variable would be 1down. We can choose 'the' as the word there.
- After that, the MRV, the one that has the most letters required, is 2 across, with a length of 5. 'Eager' can be chosen, since it matches with the word chosen for 1down, 'the'
- Then you keep choosing words subsequently that can fit the existing configurations or you backtrack in the instance that no options work for a given word and the imposed constraints