CS533 — Written Homework 1 — Solutions

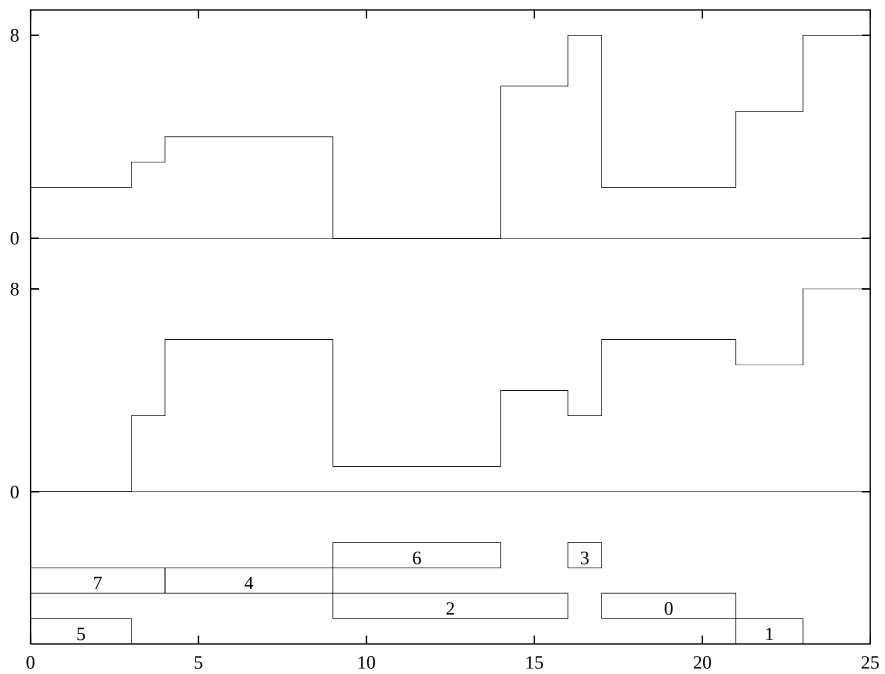
1. [10] Construct the best schedule you can for the following scheduling problem.

		Resource Type		
Task	Duration	R1	R2	Prerequisites
0	4	2	6	
1	2	3	3	
2	7	4	2	0
3	1	5	0	0 1
4	5	2	4	2
5	3	3	1	1 2
6	5	3	6	3
7	4	5	5	4 6

Here is the optimal solution computed in the iterative repair problem space:

task	start time
0	17
1	21
2	9
3	16
4	4
5	0
6	9
7	0

It has length 23



2. Consider the game of tic-tac-toe (also known as naughts and crosses). The rules of the game provide a natural problem space.

(a) [10] Give a precise definition of the problem space including the set of all states, the starting state, the operators, and the feasible (or terminal states).

- **state space**. A state is a possible configuration of the tic-tac-toe board. The board has nine squares, which we can treat as a two-dimensional 3x3 array of nine values drawn from the set $\{O, X, blank\}$. We will assume that the cross ("X") player plays first. Hence, a state has $0 \le x \le 5$ positions with "X" in them and $x - 1 \le y \le x$ positions with "O" in them. If $x = y$, then "X" must play next. If $x = y + 1$, then "O" must play next.
- **starting state**. The starting state is the 3x3 array $\langle\langle blank, blank, blank\rangle, \langle blank, blank, blank\rangle, \langle blank, blank, blank\rangle\rangle$.
- **operators**. There is a single operator. If $x = y$, the operator chooses any blank square and replaces it with an "X". If $x = y + 1$, the operator chooses any blank square and replaces it with an "O".
- **feasible (or terminal) states**. A final state is any state where there are three "X"'s or three "O"s in a horizontal, vertical, or diagonal configuration or any state with no blank positions.

(b) [5] What is the branching factor of this space? Express the branching factor as a function of the number of moves already made. (We will consider each move by a player as a single move.)

If n moves have already been made, the branching factor is $9 - n$.

(c) [5] What is the minimum and maximum solution depth? Show move sequences that produce these.

The minimum solution depth is 5.

Move number	Side	Row	Column
1	X	1	1
2	O	2	1
3	X	1	2
4	O	2	2
5	X	1	3

The maximum is 9.

Move number	Side	Row	Column
1	X	1	1
2	O	1	2
3	X	1	3
4	O	2	1
5	X	2	2
6	O	2	3
7	X	3	2
8	O	3	1
9	X	3	3

(d) [10] Estimate the total number of states in the state space? What fraction of these are terminal states? Is this state space redundant? If so, then on the average how many distinct sequences of operators generate a given state?

A simple estimate is based on the realization that after making d moves, the total number of leaves is $9 \cdot 8 \cdot 7 \cdots 10 - d$. So the total number of possible nodes in the search tree is

$$\sum_{d=0}^{9} \frac{9!}{(9-d)!} = 986,410.$$

If we assume that only the leaves of this search tree are terminal states, then we get $9!$ terminal states = 362880, or 36.8%.

I implemented a depth-first search of the space and computed the answer to be 422,074 states of which 255,168 are terminal states. A terminal state was defined as a state in which either X had won, O had won, or the board was full. Search was terminated as soon as any player won. However, this computation does not consider the fact that the state space is highly redundant: the same board position can be reached by many different operator sequences.

We can count the number of distinct board positions as follows. Suppose we want to count the number of distinct boards having x X's and y O's. We can do this by computing

$$\binom{9}{x} \cdot \binom{9-x}{y}.$$

This counts the number of possible ways of choosing x X positions out of the 9 squares and then choosing y O positions from the remaining $9-x$ squares. Call this quantity $N(x, y)$. Then the total number of distinct board positions is

$$N(0,0) + N(1,0) + N(1,1) + N(2,1) + N(2,2) + N(3,2) + N(3,3) + N(4,3) + N(4,4) + N(5,4)$$

This evaluates to 6,046. However, this does not consider the fact that some of these positions could not be created without already causing one side to win before the state was completely constructed.

These two computations provide a bound on the redundancy of the search space. On the average, each of the 6,046 states must be generated in 69.8 different ways. Because not all 6,046 states can actually be generated during a real game, the degree of redundancy is even higher.

Furthermore, there is another source of redundancy: Symmetry. Any board position can be rotated 90, 180, and 240 degrees to yield 4-fold symmetry. Any position can also be reflected horizontally and vertically. Not all of these rotations and reflections will create distinct states, but these kinds of symmetry redundancies permit further reductions in the size of the state space. These kinds of symmetry considerations are very important in computational chemistry.

(e) [5] Estimate the proportion of terminal states in which the first player wins the game.

Using my depth-first search, the X player has 49,392 winning positions or 19.36% of the terminal states. However, this includes the redundancy mentioned above.

To get an idea of the number of distinct winning states of X, we can start by considering that there are 8 possible 3-in-a-row configurations for X. Using these as starting points, we have 6 remaining squares on the board of which at least 2 must be O's. Let us define $M(a, b)$ to be the number of ways of choosing a additional X's and b total O's from these remaining 6 squares. Then the total number of winning positions for X is no more than

$$8 \cdot [M(0,2) + M(0,3) + M(1,3) + M(1,4) + M(2,4)] = 1120.$$

This is an overestimate because it will include some positions where we get three O's in a row or more than one set of three X's in a row.

2. [20] House Floorplanning.

Suppose we wish to design a 1-story rectangular house with dimensions 20 by 40 feet. The house must contain exactly 5 rooms: living room, dining room, kitchen, bath room, and closet. There must be doors connecting the following rooms:

From Room	To Rooms
Living Room	Dining room, bath room, and closet
Dining Room	Kitchen, Living room

In addition, there must be a door connecting the living room to the front of the house (one of the 40-foot long sides). Each door is 2 feet wide.

The living room, dining room, and kitchen must have windows to the outside.

Each room must have the following minimum dimensions:

Room	Dimensions
Living room	10x20
Dining room	10x15
Kitchen	10x20
Bath room	4x8
Closet	3x3

All interior walls must be parallel to the exterior walls of the house and positioned on a 1-foot-square grid (i.e., the length of each wall must be a multiple of 1 foot). All rooms must be rectangular. All space within the house must be part of one of the five rooms.

Design a problem space for this problem (and for problems with the same kinds of constraints). The goal should be to find any feasible solution. What search algorithm do you recommend for your problem space? Why?

I have tried to design a problem space based on partitioning the house by placing walls one after another, rather than by placing rooms.

- **state**. A state consists of a binary tree where each node represents
  - **a rectangular region**
  - **the set of rooms that need to be contained inside this rectangular region**
  - **the location of a wall that subdivides the region**
  - **the location of any doors in the wall**

  A leaf node in the binary tree lacks the wall and door locations.

  In addition to this binary tree, the state contains the location of any doors in external walls.
- **operators**. There are three operators.
  - **Partition:** This operator chooses one leaf node of the tree that contains more than one room. It chooses one way of subdividing the rooms in the node into two subsets and one axis (X or Y) along which to subdivide the rectangle. It then computes the maximum required size of each child node along the chosen dimension, and places the wall in a way that splits the rectangle proportionately.

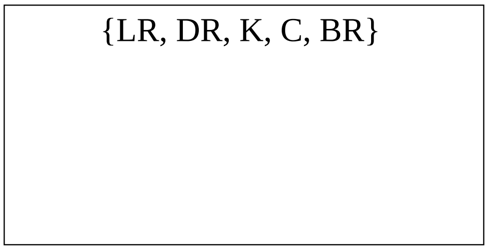
- **Install doors:** This operator takes a node whose wall separates two rooms that are to be connected by a door. It adds a door in that wall.
- **Install exterior doors:** This operator takes an exterior wall that separates a single room from the outside and adds a door in that wall if it is needed.
- **Test room area:** This operator takes a leaf node and checks whether the minimum required area of the rooms in that node is less than the area of the rectangle in that node. If not, it causes backtracking to make another choice.

- **Starting State:** A root node of the tree. The rectangle of this node is the entire house; the set of rooms is the set of all rooms to be placed in the house.
- **Feasible Solution:** A tree where every leaf node contains exactly one room; where the room's minimum size is less than or equal to the size of the rectangle in the leaf node; and where all door and exterior wall constraints are satisfied.

My plan is to use depth-first search for this space because there are no infinite descending paths and we can accept *any* feasible solution. It will probably be necessary to backtrack during the search.

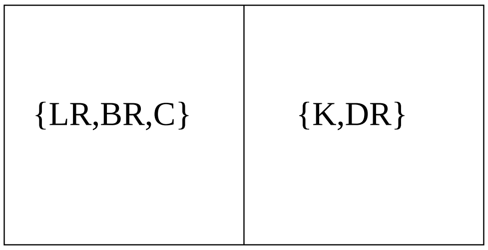
Solution to the example problem.

1. Starting state

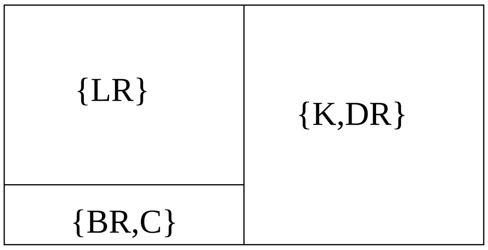


2. Test room area. Maximum x dimension is 20, actual x dimension is 40. Maximum y dimension is 10, actual y dimension is 20. Total area of rooms is 591 ft^2. Actual area is 800 ft^2.
3. Partition. Choose to partition the rectangle into the sets $\{LR, BR, C\}$ and $\{K, DR\}$ splitting along the x dimension. Obviously, we could use some heuristics here such as splitting along the longest dimension and trying to partition the rooms into two equal-sized sets. Also we could try to minimize the number of doors that will need to pass through the wall we are adding.

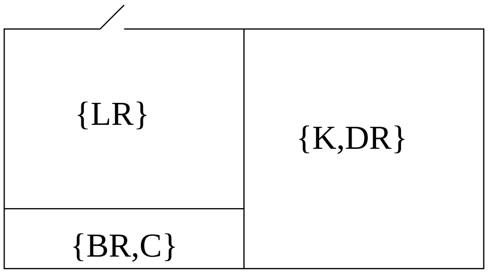
   We measure the maximum x dimension of $\{LR, BR, C\}$ to be 20 and the maximum x dimension of $\{K, DR\}$ to be 20, so we position the wall at $x = 20$. This gives the following state:



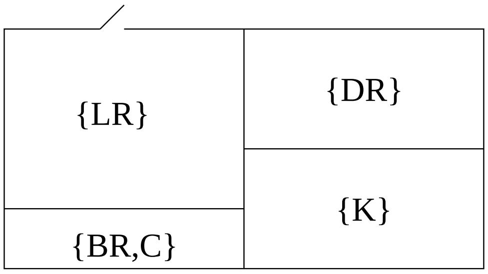
4. Test room area for left room. OK.
5. Test room area for right room. OK.
6. Partition. We partition the left rectangle along the y axis by splitting it into the two subsets $\{LR\}$ and $\{BR, C\}$. The maximum y dimension of the first set is 10, the maximum y dimension of the second set is 4, so we split the rectangle at approximately 4/14ths of the 20 feet, which rounds to $y = 6$ feet. This gives the state:



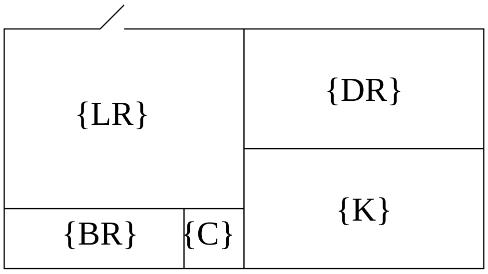
7. Test room area for LR. OK.
8. Test room area for $\{BR, C\}$. OK.
9. Add Exterior Door. Now that the living room is a single room, we add an exterior door in the top wall.



10. Partition the right rectangle along the y axis by splitting it into the two subsets $\{DR\}$ and $\{K\}$. Note that there are only two possible partitions here depending on whether the Kitchen is above or below the dining room. We will choose the one with the dining room higher, since this will make it easier to connect the dining room to the living room. The maximum y dimension of the DR is 10, the maximum y dimension of the kitchen is 10, so we get an even split at $y = 10$.



11. Test room area for DR. OK.
12. Test room area for K. OK.
13. Partition the $\{BR, C\}$ into two subsets, $\{BR\}$ and $\{C\}$ along the x dimension. The maximum x dimension of BR is 8, the maximum x dimension of C is 3, so we split the 20 feet at $x = 15$.



14. Test room area for BR. OK.
15. Test room area for C. OK.

16. Add door between LR and DR.
17. Add door between DR and K.
18. Add door between LR and C.
19. Add door between LR and BR.

Final design is