Fall 2019 Hashing Writeup

Zoe, Andy, Lawrence, Linh, Tommy

Initial Goals

The overarching goal is that when given a string representing a sequence of moves, find a tight hash code which can be used to DP solve from the associated board state.
This hash code must follow several constraints. The first is that children of a specific state (states which lie one or more moves down the tree from the given state) have a larger hash code than their parents. This is required for how we are structuring our DP solver, which Spencer is writing. The second is that the hash code must be deterministic and should be efficient to calculate. Finally, we want the total spread of the hash code to not be much larger than the problem space (total number of canonical paths). This is because our solver will be more efficient on the tightest possible hash space. We do not want to be hashing impossible moves or board states.
As a secondary goal, we are interested in 61C level cache efficiency as we are talking about loading and unloading large amounts (about 4 terabytes) of data. Any cache miss and imperfect stride will necessarily slow down our computation speed.

Past attempts

We started the semester out by picking up where the Sp19 theory team left off. We had shown that any column based hashing approach maxed out at 47 bits, which is a much larger state space than we desired.
We flipped this by starting to think about rows, which we figured would have to be worse. Initially this seemed like a good approach until we realized that all of our schemes relied on (1) using trits rather than bits, as we cannot rely on the gravity of the column, so a delimiter will not work; and (2) the worst case size (and therefore the bounding size of the function) is exactly the same as the column solution, so the entire thing is 60% larger, as designed.
We tried a different approach conceptually related to this which broke down a smaller board into its lines of four in the following way:
Diagonal Down-Right: 3 starting rows * 4 starting columns = 12
Diagonal Up-Right: 3 starting rows * 4 starting columns = 12
Vertical: 3 starting rows * 7 columns = 21
Other: Horizontal: 4 starting columns * 6 rows = 24
Total = 69
The hope was that we could decompose the board into several smaller boards, but we encountered similar issues conceptually as our canonical pathing ideas.

Realization of Problems

After considering our goals for this semester, it occurred to us that we were basically trying to hit a home run. The reality is that goals 1 and 3 are, to some degree, mutually exclusive. Compactness

of a hash function is actually a benefit that comes directly from recursion with memoization, and becomes a tradeoff when you choose to instead use DP. Given the numerous applications of recursion and DP out there, if there was a way to combine both advantages into one, there would already be a framework for doing so. However, as there was no obvious way to do that, we began considering if we could relax one of the two constraints so that we had a more attainable goal.
The idea we came up with was to relax the constraint of compactness to only involve compactness within each tier of game states, split up by the number of total pieces on the board. This seemed like a much more attainable goal, as having the total number of pieces as an extra piece of information also tells you how many red and black pieces there are and it becomes much easier to enumerate the possible positions compactly. Additionally, a hash function for the full search space can be reduced to this tier hash function by simply counting the number of pieces on the board and calling the hash function for that specific tier. Lastly, there might be the possibility to move from one tier to another without having to unhash the numerical values into game states by mapping the values of the parents in one tier to the values of the children in a lower tier.

Tier based approach

We pivoted to a tier based approach to the hash for a dp solver. The hope was that we would then perhaps more easily be able to create compact hashes and integrate multicore optimization down the road.
As an attempt to compromise between a recursive solver and a dynamic programming solver, we attempted to create a tier-based hashing scheme wherein each tier has its own hash function. We split all board states into 43 tiers based on the number of pieces placed (0 to 42). The benefit of such an approach is that it trivially meets the DP requirement of children after parents since for all positions, all of its successors are in the following tier. Moreover, it should be easier to find a tight hash function within each tier rather than across all possible board states.
Tier one is comprised of only one black piece, so there are only seven possible valid board states.
Tier two has 49 possible pairs of locations
We attempted to find a formula for the smaller tiers and an algorithm for the larger tiers, with an emphasis on not over counting in any tier (using our precomputed state space per tier to check).
The biggest challenges then are finding clear systems for going between tiers (finding the parents and children of a specific state programmatically without converting to any other representation of the board) and how to programmatically have different hashes of different tiers feed into each other correctly.

Small tiers

For our small tier-based hashing approach, we tried to see if we could create a formula to calculate the number of possible hashes in each tier. We approached this by trying to find a pattern when manually calculating out the number of possible positions in each tier for the first few tiers.
Note that black goes first and colorings are given top down.
Note: Use nCr if columns are interchangeable (same number of pieces) and multiplying remaining choices for distinguishable columns. You can also rearrange your column choice order. Ex: $7 * \binom{6}{2} = 105 = \binom{7}{2} * 5$.

Tier 0: 1

There's only an empty board.

Tier 1: 7

You can place the first piece in any of the 7 columns.

Tier 2: 49

You can have 2 pieces in one of the 7 columns with exactly 1 coloring (red above black): $\binom{7}{1} * 1$
You can have 1 piece in two of the 7 columns with exactly 2 colorings: $\binom{7}{2} * 2$
So: $[\binom{7}{1} * 1] + [\binom{7}{2} * 2] = 49$

Tier 3: 238

You can have all 3 pieces in one of the 7 columns with exactly 1 coloring (black red black); $\binom{7}{1} * 1$
You can have 1 piece in three of the 7 columns with exactly $\binom{3}{1}$ colorings (choose the red among 3 positions): $\binom{7}{3} * \binom{3}{1}$
You can have two pieces in one column and one piece in a different column ($7 * 6$ column choices because the columns are distinguishable). All three colorings are reachable positions: $(7 * 6) * 3$
So: $[\binom{7}{1} * 1] + [\binom{7}{3} * \binom{3}{1})] + [(7 * 6) * 3] = 238$

Tier 4: 1120

You can have all 4 pieces in one of the 7 columns with exactly 1 coloring (red black red black): $\binom{7}{1} * 1$
You can have 1 piece in four of the 7 columns with exactly $\binom{4}{2}$ colorings (choose the 2 reds among 4 positions): $\binom{7}{4} * \binom{4}{2}$
You can have 2 pieces in 2 columns. The only coloring restriction is that one of the bottom row pieces must be black, giving us $\binom{4}{2} - 1$ colorings. $\binom{7}{2} * (\binom{4}{2} - 1)$
You can have a 2 — 1 — 1 pattern ($7 * \binom{6}{2}$ column choices) where all $\binom{4}{2}$ colorings are possible: $(7 * \binom{6}{2}) * \binom{4}{2}$
You can have a 3 — 1 pattern ($7 * 6$ column choices) with 4 possible colorings (the 1 column piece can be placed in any of the 4 moves, each creating a distinct coloring): $(7 * 6) * 4$
So: $[\binom{7}{1} * 1] + [\binom{7}{4} * \binom{4}{2})] + [\binom{7}{2} * (\binom{4}{2} - 1)] + [(7 * \binom{6}{2}) * \binom{4}{2})] + [(7 * 6) * 4] = 1120$

Tier 5: 4263

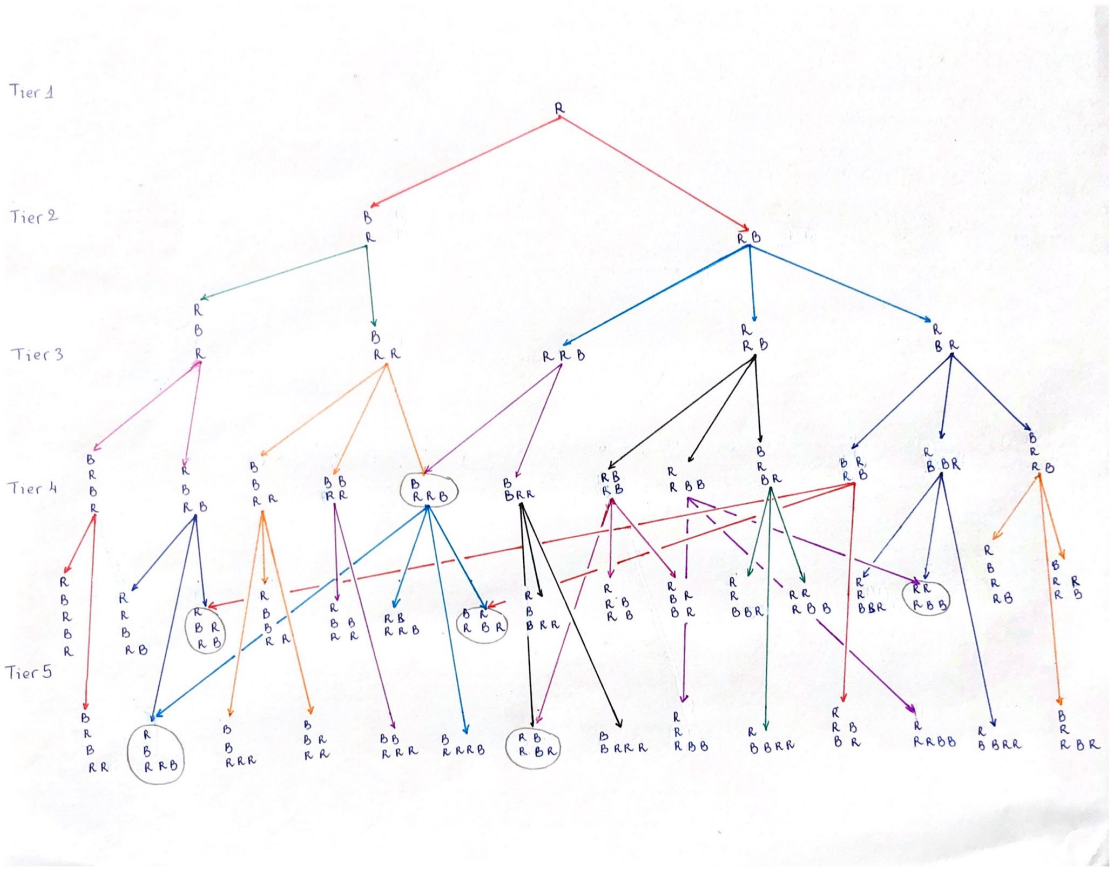
You can have all 5 pieces in one of the 7 columns with exactly 1 coloring (black red black red black): $\binom{7}{1} * 1$
You can have 1 piece in five of the 7 columns with exactly $\binom{5}{2}$ colorings (choose the 2 reds among 5 positions): $\binom{7}{5} * \binom{5}{2}$
You can have a 4 — 1 pattern ($7 * 6$ column choices) with 5 possible colorings (the 1 column piece can be placed in any of the 5 moves, each creating a distinct coloring): $(7 * 6) * 5$
You can have a 3 — 2 pattern ($7 * 6$ column choices) with $\binom{5}{2} - 2$ colorings (all combinations of red are legal except for both red at the bottom since black goes first and both red at the top since black is last to move): $(7 * 6) * (\binom{5}{2} - 2)$
You can have a 3 — 1 — 1 pattern ($7 * \binom{6}{2}$ column choices) where all $\binom{5}{2}$ colorings are possible:

$(7 * \binom{6}{2}) * \binom{5}{2}$
You can have a 2 — 2 — 1 pattern ($7 * \binom{6}{2}$ column choices) where all $\binom{5}{2}$ colorings are possible: $(7 * \binom{6}{2}) * \binom{5}{2}$
You can have a 2 — 1 — 1 — 1 pattern ($7 * \binom{6}{3}$ column choices) where all $\binom{5}{2}$ colorings are possible: $(7 * \binom{6}{3}) * \binom{5}{2}$
So: $[\binom{7}{1} * 1] + [\binom{7}{5} * \binom{5}{2}] + [(7*6)*5] + [(7*6)*(\binom{5}{2} - 2)] + [(7 * \binom{6}{2}) * \binom{5}{2}] + [(7 * \binom{6}{2}) * \binom{5}{2}] + [(7 * \binom{6}{3}) * \binom{5}{2}] = 4263$

Findings

We were unable to find any formula to calculate the number of positions within a tier. We also noticed that any such pattern would likely change starting in tier 8 (since there are only 7 columns), although the fact that $\binom{n}{k} = 0$ if $k > n$ might be able to account for this. Overall, the big challenge was identifying legal colorings. We weren't able to establish an effective way to find legal colorings other than via enumeration, which is only feasible for lower tiers. For the lower tiers listed here (with fewer than 3 red pieces), you can prune illegal colorings by saying there must be a black piece in the bottom row and either a red/black piece at the top of a column (whichever color just went). However, this doesn't hold up for higher tiers. Counterexample: RRBBB — BR.

Visual representation



Understanding Tromp's Hashing Technique

Let's first look at a naive method of hashing a Connect 4 board so that we have a benchmark to compare future methods. There are 42 slots where each piece can go, and a slot can contain either an X, O, or nothing. Thus, this hash captures 3^{42} possibilities which is essentially 67 bits.
A Connect 4 board is comprised of six rows and seven columns. We will add an additional row at the bottom for computation purposes, so we really have a seven by seven grid. Each slot in the seven by seven grid can be a 0 or a 1, so there are 49 bits.
The board is initialized to all 0's except the bottom row which contains 1's.
To ascertain a board from the hash, the algorithm is as follows for each column:
Start at the top of the column and move downwards until the first 1 is reached. This represents your top piece. Anything above this 1 (all 0's) represent empty spaces.
Now, check if the bit at the bottom of the column is a 0 or 1. If it is a 0, then this topmost 1 and all other 1's represent an O. Else, it represents an X and all other 1's represent an X. Any zeros below the 1 and above the top bit represent the opposite value of whatever the 1's represent.
Below this top bit and above the bottom bit are pieces represented by 0's or 1's. The bottom bit

defines exactly what a 0 or 1 means for a given column so that applies here. We can now move from a hash to a board state.
As a basic example, let's look at a 4 x 3 board. Initially, we have:
0 0 0 0
0 0 0 0
0 0 0 0
1 1 1 1

Placing in X in the 2nd column (index 1) we have:
0 0 0 0
0 0 0 0
0 1 0 0
1 1 1 1

Placing an O in the 2nd column we have:
0 0 0 0
0 1 0 0
0 0 0 0
1 0 1 1

Notice how, when the bottom bit was changed as a result of an O being the top piece in a column, the rest of the values in the column below this top bit were flipped. This is because the pieces are not moving places but O?s and X?s are changing the corresponding representations as 0?s or 1?s. To be sure this is clear, let?s place one final X in the 2nd column.
0 1 0 0 _ X _ _
0 0 0 0 → _ O _ _
0 1 0 0 _ X _ _
1 1 1 1

In this way, we can represent any Connect 4 board!