Algorithms Group cs20408

Strategies for various assignments:

Assignment 1:

Objective: Winning was the criteria for this particular assignment

In this assignment we chose an iterative way of dealing with the problem
The base conditions were specified exhaustively and the program analyzed the
Board based on those constraints given by base conditions.
Most of the times it worked out.
Important features:

- Concept of virtual move which obviates recursion
- Simple functions for generating moves (this is also iterative in nature)
- Due to iterative approach time taken was considerably less
- Looks into two-three steps in advance just by a use of simple iterative methods

Time considerations: less than a second execution time for moves
Average time: Again less than a second
The program earned 1st position overall in the individual submission round
Complexity: O (n)

Assignment 2:

This one is further refinement of previous assignment
There is a significant change in overall algorithm
since our last submission.
Working :

- As the game progresses my program identifies different geometrical figures coming up on the board
- Now there are finite no of such figures the ones which are totally segregated decide the game

Winning Block: a block in which whoever takes the first move wins (W)
Losing Block: a block in which whoever takes the first move loses (L)

So every time we try to maintain even number of W blocks with odd number of L block or vice versa before giving opponent to play a move on the board.
It can be shown:

W + L = L
W + W = W
L + L = W

Also these are commutative and associative so using these properties we force the opponent into position of (W + L or L + W).

Advantages:

- Again being a approach inclined towards iteration working speed is quite fast.
- Backtracking is also being used but single stepped it goes to one level of partial solution and chooses or discards it depending on the given constraints defined as memoization.

Again the source code was from the previous game.

Novel concept of using blocks:
Winning and Losing blocks, those make decision-making lot easier.

Time Considerations: The time take again was less then a sec for moves
Disadvantages

- Using hard code for determining geometrical figures

Assignment 3:

Analysis :

→ Triangular table of 21 coins.
→ Negative points for some moves
→ Less score for actually winning the game.
→ Priority depends only on the length of the move.

Observations :

→ One can score more than the opponent though actually losing the game.
→ There are some –ve points scoring winning moves for particular positions.
→ If one plays those –ve scoring winning moves he can win the game but can't score much.

Why a new approach is required:

→ The algorithm for the previous assignment can not correctly detect the winning moves and losing. It works only in some cases . So a better algorithm for deciding the winning move is required.
→ Recursive algorithm can make the purpose survived.

Approach (Strategy):

→ To stay in losing position and make the opponent play those –ve scoring winning moves.
→ If opponent plays those –ve scoring winning moves , he will score very less. The opponent score will be almost equal to us with in 5 points.
→ If opponent avoids to play the –ve scoring winning move , he will be in losing

position, which means i am in the winning position. Now we can play with the winning strategy and win the game scoring a lot. We can score well initially and also win the game .hence score is much greater than the opponent.

→ A recursive algorithm(DFS) is used for deciding the winning move.

→ we can memoize the algorithm by increasing the base cases.
We can say a position with 5 or less coins is winning or losing with just knowing whether the coins or connected or not.

Algorithm :

Handling the moves :

→ A complete move list is made using arrays. Total number of moves are 126.

→ Score of a move depends only on the length. Therefore priority of moves depends on length. An array is used to know whether a particular move is possible or not. Moves are loaded in priority and each picked by me or opponent is updated , ie., all the moves in which the coin removed occurs is made unavailable.

Recursive part :

→ It contains two functions . 1. winner and 2. loser .

→ Virtual moves were played in both the functions. My virtual move is played in the winner and the opponents virtual move is played in the loser function. A copy of the present table is sent to the winner function in the main function and based on that the recursion is done .
Winner plays all the possible moves until it gets a win in return.(I am going to win).so return win from this function .else return lose.(I am losing).
Loser plays all the possible moves until it gets a win in return .(I am going to lose ,since the opponent gets win in return) . so return lose from this function. else return win(the loser has no winning move).
Winner and loser functions are called from one another .
This is as below ……..

winner	loser
winner gets atleast one winning move	loser gets atleast one winning move
return win	return lose
else	else
return lose	return win

* note : both win and lose are with reference to my point of view.
Win – I am going to win.
Lose – I am going to lose.

Selecting the winning move or the best scoring move :

- → If winner function called from main returns win then I am going to win and I Play a winning move . else I am going to lose and I play the best scoring move I have.
- → While recursion , win_move (global variable) contains the number of the winning . So we can load the winning move by passing the content of the variable win_move to the function isMove.
- → In case of losing case win_move is 1. Since the moves were loaded in priority the first possible move is the best scoring move.

Strategy :

- → Best scoring move is played as long as coins left on the board is less than 15. Good scoring is done initially.
- → When coins left fell into my bound, I will start applying my recursion and play the winning moves.

Run time of the algorithm :

- → It is exponential in the worst case.
- → The move list contains only 126 moves. Though we have exponential algorithm the time taken is within given bound. We have a better algorithm to reduce the running time of the recursive part. We didn't implement that because of 2 reasons.
    1. winning is not the primary in the assignment .scoring is the primary thing.
    2. no of coins in the board is less to apply that algorithm. run time won't differ much.

  we applied that algorithm in the next assigment since it is required

Assignment 4 & 5 :

Analysis :

→ Square table containing 36 coins.

4^{th} assignment :

→ Priority depends only on the length of the move.

5^{th} assignment ;

→ Priority depends on both the length and position of the coins in the table.

Observations :

→ Score for picking particular moves with 4 or 5 coins is very high compared to the bonus of actually winning the game..

→ Winning is the primary goal.

About the Algorithm :

We tried to implement the algorithm we mentioned about in the previous assignment report. It makes use of divide and conquer method (only once).

→ First try to break the table into to two disconnected parts.

→ We defined a new game , which is just opposite to the present game. Ie., one who picks the last coin wins the game. Its called reverse game.

→ After breaking is done, we carry out recursion for each part and for each of two games (we defined new game). Now we got four results.

* first part normal game ---- win / lose
* first part reverse game ---- win / lose
* second part normal game ---- win / lose
* second part reverse game ---- win / lose

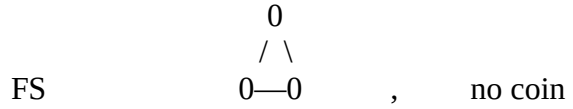
→ Our notation is explained by an example.

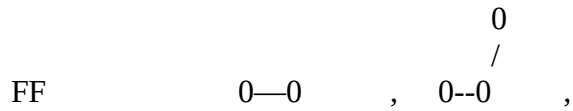
FS

F - means winning position in normal game.

S - means losing position in reverse game.

→ Therefore 4 cases are possible for a single part (particular table) .
FS , FF , SF , SS. Examples for each one is given below.





SF 0 , 0—0

SS 0—0 0—0

➔ Now we have to know which move leads to what position . No coin position is defined FS. The positions after playing particular moves converts to below shown positions.

FS	-----	winning move(normal game)	-----	SF
	-----	any other move	-----	FF
SF	-----	winning move(reverse game)	-----	FS
	-----	any other move	-----	FF
SS	-----	any move	-----	FF
FF	-----	winning move (normal game)	-----	SF
	-----	winning move(reverse game)	-----	FS
	-----	any other move	-----	FF
	-----	winning move in both games	-----	SS

→ We decided to break the table into two parts. In assignment 4 , division is done diagonally . It led to bad results. So in assignment 5 division is done horizontally. We try to divide the table as long as it is not divided. After it is divided we will have one of four cases for upper half and one of four cases for the lower half. Totally 16 cases are possible. We have the proof how to decide winning positions and the winning moves to be played .

→ The proof is as follows :

Statement 1: FS , SF is losing case.
Statement 2: SS , FS is losing case.
Statement 3: SS , SS is losing case.
2N-1 = no of moves required to play.

Proof by induction.

For N=1.

FS is position with no coin and SF with only one coin. This is sure losing case.ie.,

0 losing

SS is position as shown at the start of this page and FS is position With no coin.

0—0 0—0 losing.

SS is position shown in previous page .

0—0 0—0 0—0 0—0 losing

by observation all the above examples are losing cases.

Therefore statements are true for N=1. ----------(1)

Let us assume statements are true for N=n,n-1,….1 ----------(2)

Consider,

FS , SF postion(N=n+1). this can be changed into following 4 positions After Playing any move. Every position consisits of atleast one move which can Convert back into the FS , SF case.

	----- SF , SF	----- winning move of reverse game in 1st part
	----- FF , SF	----- winning move of reverse game in 1st part
FS , SF	----- FS , FS	----- winning move of normal game in 2nd part
	----- FS , FF	----- winning move of normal game in 2nd part (SF , FS)

therefore we got a position FS , SF with N= n , which is losing by(2). Hence FS , SF is losing case.

SS , FS position(N=n+1). Same as above.

	----- FF , FS	------ winning move of normal game in 1st part (SF, FS) (proved above)
SS , FS	----- SS , SF	------ winning move of reverse game in 2nd part (SS , FS)
	----- SS , FF	------ winning move of reverse game in 2nd part (SS , FF)

therefore we got N=n position which is true by (2). Hence SS , FS is losing case.

SS, SS position (N=n+1)

SS , SS ------- any move (SS , FF) ----winning move of reverse game In 2nd part (SS , FS)(proved above)

Hence all the statements are true for N=n+1. ----------(3).

By FiniteMathematical Induction ,and from (1),(2),(3) , we can say that Statements 1,2,3 are true for any position.

-->

winning cases	losing cases
FS , FS	FS , SF
SF , SF	SS , FS
FS , FF	SS , SS
SF , FF	
SS , SF	
SS , FF	

-->The only other position remained is FF , FF. its nature can't be decided basing on this proofs. Some of them are winning and some of them are losing. so we treated it as losing one and we do not try to fall into that.

0—0—0 (FF) 0—0—0 (FF) losing case.

0
/
0—0 (FF) 0—0 (FF) winning case.

→ This decides the classification of winning cases and losing cases .Hence the position can be found out and the corresponding move is played .

In winning case – the winning move.

In losing case - the best scoring move

Implementation :

→The movelist is made depending on the priority given.

4th assignment : arrays are used to store the moves total moves only 82.

5th assignment : file is used to store the moves.

→ After the opponent plays his move the move is encoded and updated in the Corresponding table.

→ Before going for recursion, we have to make sure that the table is divided into two parts.if divided then go for recursion .else we will try to break the table.

4th assignment : we try to divide the table diagonally and the method is serial.

5th assignment : we made the efficient way to divide and check for division. If table is not divided then the max scoring move which helps in dividing the table is played.

Algo used for checking whether the table is divided:

1. consider only two rows 3 and 4 .
2. move through the blank places where no coin is present.
3. if the next place horizontally is blank then move horizontally.
4. else if the next place vertically is blank then move vertically.
5. start from any of the two vertices in first column. .

6. continue this till u reach the last column or no path.
7. if we reach the last column without break in the path, then table is divided.
8. If u reach no path at a place before reaching the last Column, then table is not divided. Take the two coins horizontal and vertical to the place where u break the path. Play the max possible scoring move of the two coins .

→ The recursive algo stated in the previous assignment is used to find out the current position like FS (upper half) , SF (lower half) , before we play a move . depending on the case we are in, we play the corresponding move.

→ Before playing the move we decode the move and after playing the move we encode the move and update.

Advantages :

1. Run time is reduced to $4*(n/2)^{(n/2)}$ from n^n(n to the power of n).
   Run time for normal recursive algo is exponential. If there are N possible moves for the given game, then running time will be n^n(n to the power of n) in the worst case. Using this algorithm run time reduces to $4* (n/2)^{(n/2)}$, which is much much lesser than normal recursive algorithm. This algorithm is highly efficient when n is high. (36 is appropriate rather than 21 .so we didin't use this algo in previous assignment).

2. dynamic storage of moves is reduced to less than half.
   There is no need to store all the possible moves(N say) dynamically. In This we need to store only the possible moves of less than half table. The total moves reduces to $(< N/2)$. This can be proved as follows.
   N = 2* all moves of one part + moves connecting both the parts + moves Lying in division zone (diagonal in assign4).
   We can use functions like encode and decode for mapping the real moves into The listed move list.

3. Can be improved (scope of Divide and Conquer algorithm)
   division is performed only in first step. It can be implemented in further steps to find out the winning move . then run time reduces drastically to log N. implementation is difficult , may be because of dividing the table in to two parts .but have a scope to be implemented by divide and conquer.

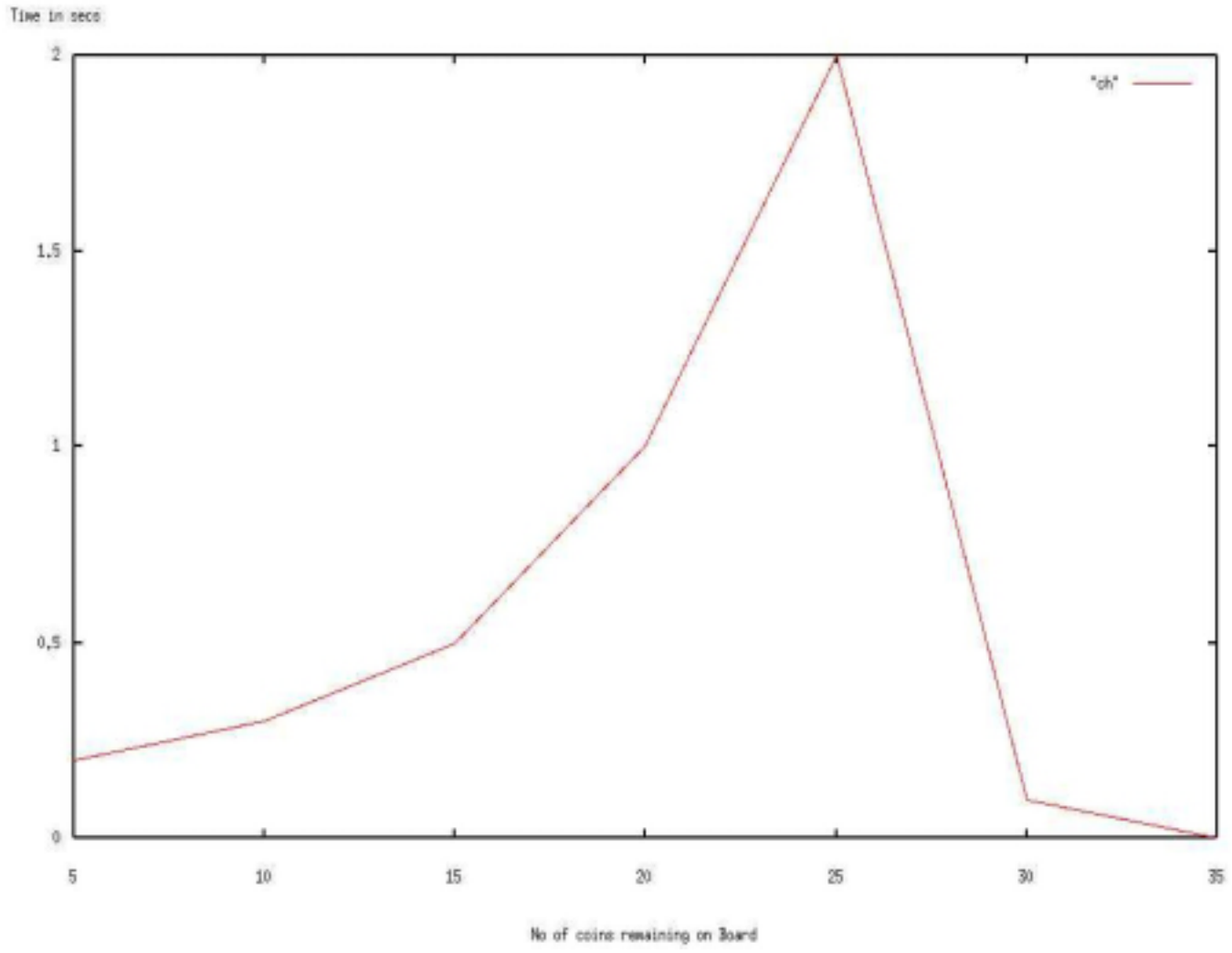
Attempts to further reduce running time :

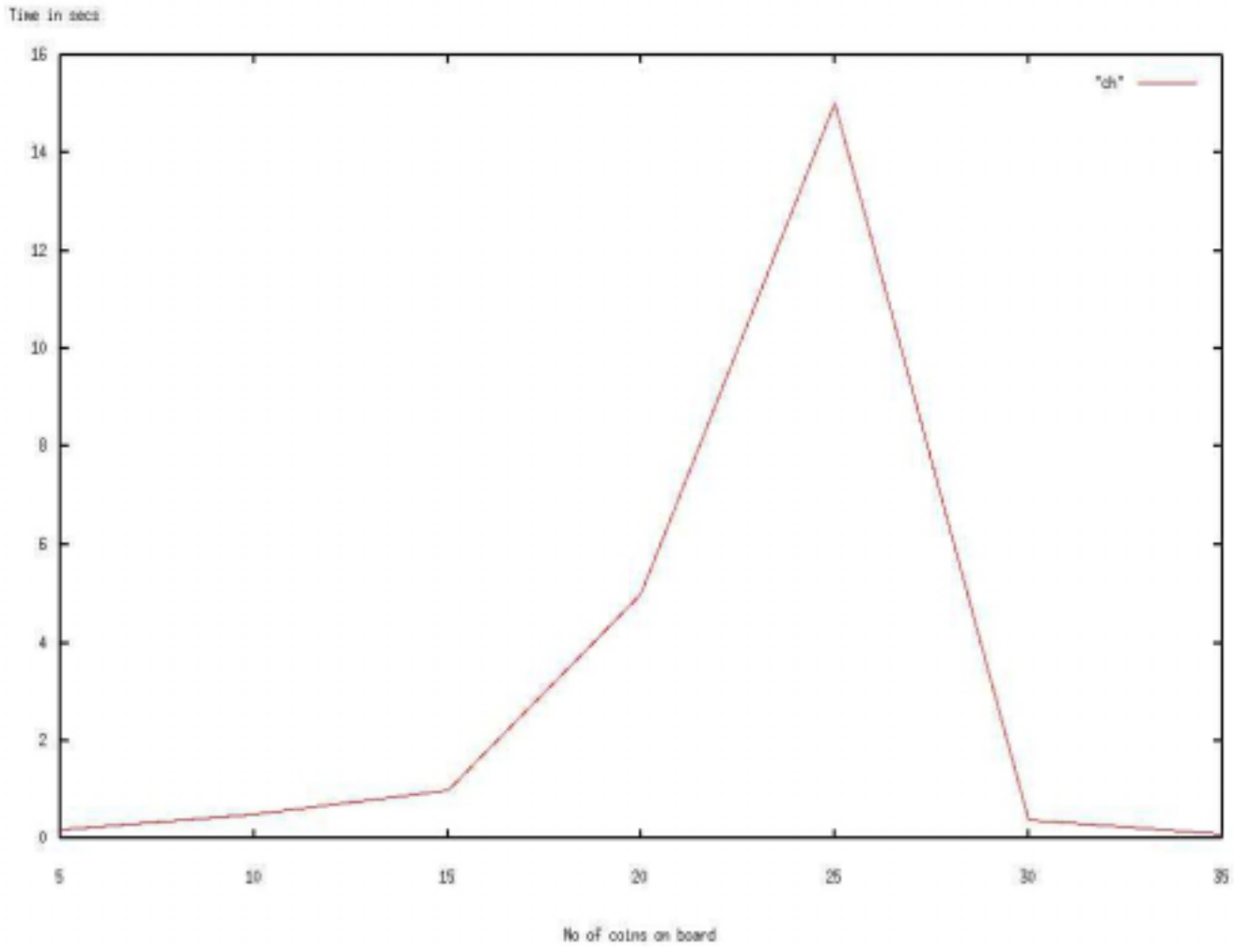
We tried to further reduce the running time in assignment 5. we had a move list. If coins left are around 13 r more , its taking time like(10min) to do the recursion. For coins <10 , its taking max of 10sec.So,we tried to shortlist the winning moves we get when coins greater than 10 for both the games normal and reverse. Since the first 2 rows

and last 2 rows contains red coins(-ve points if picked). So there is high probability for few positions repeating. Therefore we need to store some moves(line nos of the move in the move list) and do recursion for those moves. We can get the winning moves among those shortlisted. This short list of moves is like cache in processor. If we get the winning move within the shortlist then it's a hit (very fast). Else it have to check the whole list, then it's a miss(vey slow).we coded it , but we need some more time to debug it. We could not implement it finally.

Hereby we are also giving graphs showing the performance of the algorithm 4 and 5 assignments. The sharp decline after 25 coins is obvious because for coin count more than that we are not using recursion because of its cost.

Graphs:





Assignment 5