# THE STRATEGY FILE

## Assignment 1&2:

### Problem:

* 21 coins in the form of a triangle on the board
* If an invalid move is taken, negative score of -50.
* Time limit is 30 sec (for 2nd assignment).
* Scoring scheme is 1(5), 2(9), 3(11), 4(10), 5 (8), 6 (5)
* Winner gets a bonus of 80 points.

### Aim and Plan:

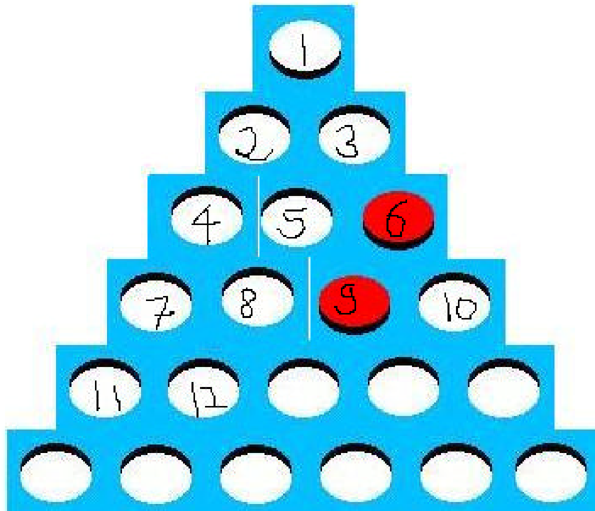
- To maximize wins and score.
- Maximize points if no winning move is found according the scoring scheme.
- Use of Pattern Recognition to recognize patterns and reduce the time for each move.
- See 2 levels ahead and recognize the patterns to find the winning move.

The data structure involved:

The data structures should have been such so that on looking at a particular board configuration,
the decision whether we can put the opponent in a sure loss situation can be done quickly.

- The moves array

On finding out the total number of valid moves that the player could take at the start of the game,
it was found that the number came out to be a number say ‘N’.
The moves array is a ‘N’*6 double dimensional array that contains all the
N possible valid moves at the beginning and the number of the coins present in them .




**For instance, a particular 6 element array of the moves array could be**
**Moves [67] = {6, 9, 999,*,*,*}**
**In all such elements of move 999 is used as the ending element so that while scanning a move you**
**don't have to go till the end.**

- **The status array**

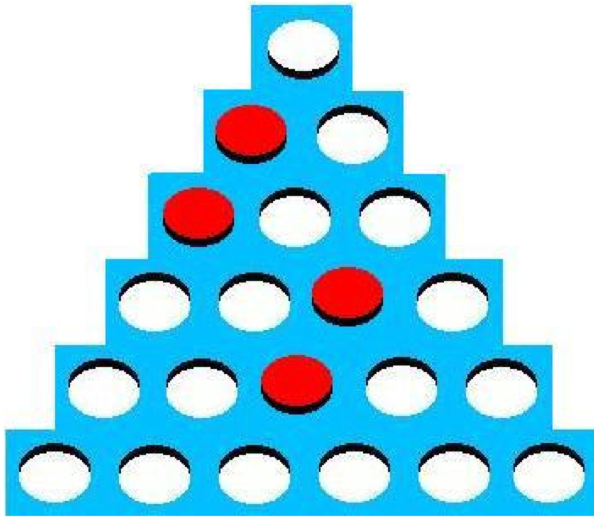
**Status array is a 1 dimensional array of length 'N' where each element can store either a 1 or a 0.Hence at the above shown board position the 67 element of the status array will store a 1 since the moves array element of that index is still a valid move.**

**Strategy:**

**The strategy is based on pattern recognition and graph isomorphism to locate certain patterns on the board configuration and determine whether a given board configuration is surely losing for the opponent and hence winning for the player.**

The Patterns

1



**In the above configuration, all coins have been taken up except the ones that are red .If any player falls in this configuration i.e whoever has to take the coin now is definitely going to lose.**

**<u>Case 1</u>**

**If the player takes two coins in a line, the other player takes one out of the remaining two and he/she has to take the last coin and hence loses.**

**<u>Case 2</u>**

**If the player takes one coin, the other player takes two coins in a line and the player again loses.**

**Hence, this is a sure loss situation for whoever has to take the coin now.**

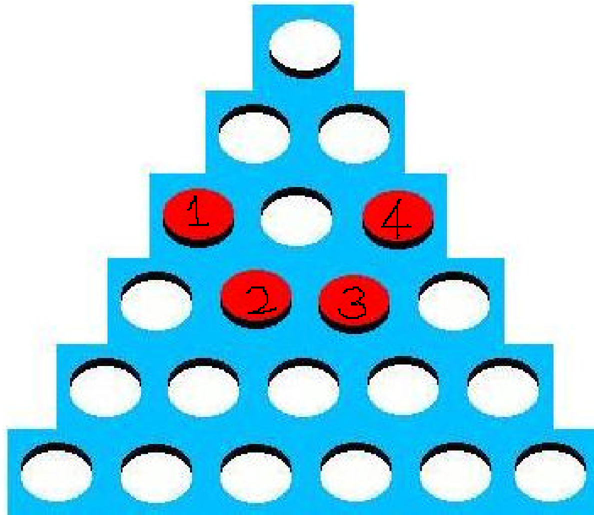
<u>How to recognize the pattern?</u>

**The method used to identify the pattern should be such that the pattern can be recognized anywhere it occurs in the entire board, and in whatever form it appears.**

**For instance, we can quantify the above pattern in a human form as two**
**non-intersecting lines of two coins each.**

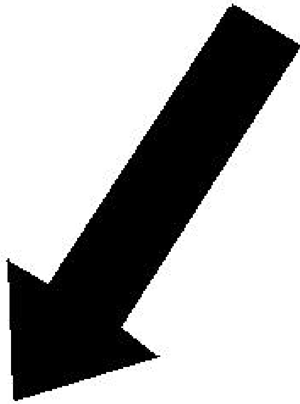
**In order to recognize the above pattern, the program must:**

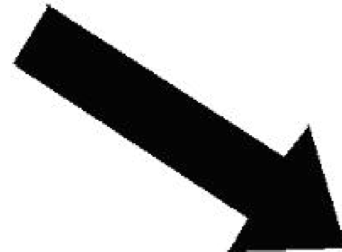
- **Look if only 4 coins are left**
- **No 3 length moves can be made**
- **Only 2 ,2 length moves are there**




**In the above configuration also only the red colored coins remain to be taken. The player landing up in the above board configuration i.e. the player whose it is to pick the coin is also in a sure loss position.**

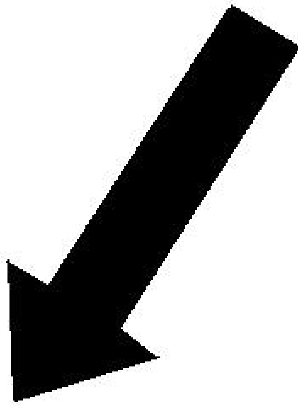
**Case 1** **Player A picks coin 1**





**Player B picks coin 2 or 3 or 4**

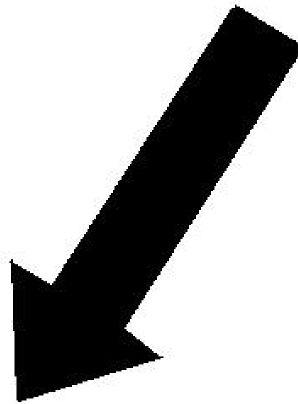
**Player B picks 2and 3 leaving A to pick the last coin**



**Player A picks only 1 coin forcing B to pick the last**

**Case 1**

**Player A picks two coins 2, 3**



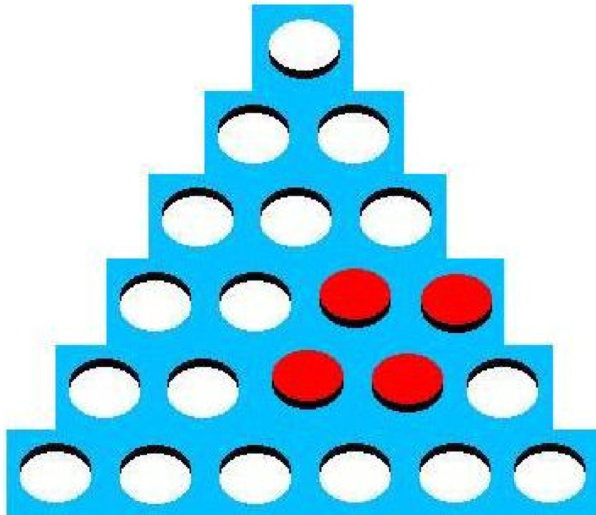
Player B pick only 1 coin leaving a pick the last

Hence, the player A is again in a sure loss position.

How to recognize the pattern?

This time also we see that

- There are 4 coins left on the board
- All possible moves left are only 2 length and 1 length moves



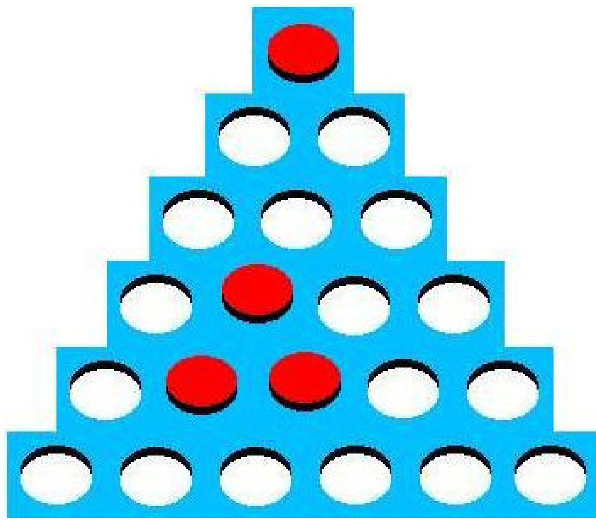
This is also a sure loss position for whoever has to pick the coin.

How to recognize the pattern?

Here again the same two conditions are satisfied

- There are 4 coins left on the board
- All possible moves left are only 2 length and 1 length moves

4



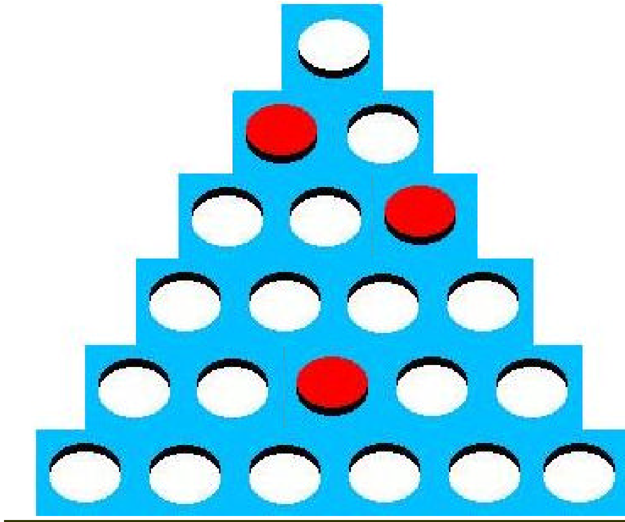
Similarly, this configuration is also a sure loss situation.

How to recognize the pattern?

In this case:

- No of coins are still 4
- No 3 length moves can be made
- 3 ,2 length moves exist

5



Similarly this too is a sure loss situation. A human would identify this board configuration as an odd no of isolated coins.

How to recognize the pattern?

Here we can easily see that

- There are only 1 length moves
- The no of possible moves is odd

Hence it's very easy to identify the patterns using the number of coins left, the number of moves left and the length of the moves that are left.

Implementation:

The pseudo code:

Now let's take a look at how the algorithm works.

```
If the no of coins <18
{
  repeat (for i =0 to length of status array)
  {
    if (status[i]=1)//i.e. is this move a valid
    {
     simulate the board configuration after I take the move 'i'
      {
```

```
       check if a move exists which can result in a sure loss
configuration
       //i.e. check that if I make move 'i' does there exist a move the
       //opponent can take and put me in a sure loss board
configuration
       If there exists such a move mark move 'i' as dangerous move
      }
      {
       if after the simulation the board configuration matches a sure
loss
       pattern then the move 'i' is a winning move take the move
      }
    }
   }
    //if no winning moves are found control comes here
    Now out of all the moves that are not dangerous choose the one
that
    fetches the maximum points
 }
  else
 {
   take the move that fetches the maximum number of coins
 }
```

**Drawbacks:**

- This method of pattern recognition takes a lot of time to recognize a few patterns.
- This program looks only 2 levels a head in determining the winning move.
- We have only recognized patterns that are having size 4.

This drawbacks have been corrected in our next assignment using recursion (MinMax strategy)
and Dynamic programming.

Running times:

- It takes very less time till 10 coins occur since It takes coins greedily till there.

- It takes some time at 10 coins to look ahead and recognize the patterns.
- This doesn’t take more than 30 sec.
- After recognizing the pattern and taking the winning move It takes negligible time.

**Strategy for Assignment 3:**

**Problem:**

*** 21 coins in the form of a triangle on the board**
*** If an invalid move is taken, negative score of -50.**
*** Time limit is 30 sec.**
*** Scoring scheme is 1(5), 2(1), 3(8), 4(-11), 5(11), 6(-30)**
*** Winner gets a bonus of 25 points.**

**Aim and Plan:**

*** To maximize wins and score.**
*** Use a recursive min-max strategy to find the winning move.**
*** Maximize points If no winning move is found according the scoring scheme.**
*** Never take a move which is of length 6.**
*** Take a move of length 4 cautiously.**
*** Use of DYNAMIC PROGRAMMING to reduce the time for each move.**

**Implementation:**

**First we would like to highlight the important features of our strategy:**

*** It uses Min-Max strategy (Backtracking) in finding the best move by recursively going through all the possible**
**moves. (implemented by recursion () function)**

*** It also acquires maximum points when It has found that It doesn't have any winning moves It will take the**
**move GREEDILY which fetches the maximum no. of (implemented by CoolMove() function)**

*** We used Dynamic Programming (Memoization) to store whether a position is winning or not. We used it afterwards**

when this position again comes in the process of recursion instead of computing it again. This strategy greatly reduces the time required for getting the move.(implemented by stored_values []).

* Instead of checking all the moves (every permutation of coins) for validity every time we computed all the 126
possible moves (in the tmp [][] array) and there is another array status[126] which holds whether the corresponding
move is valid or not.

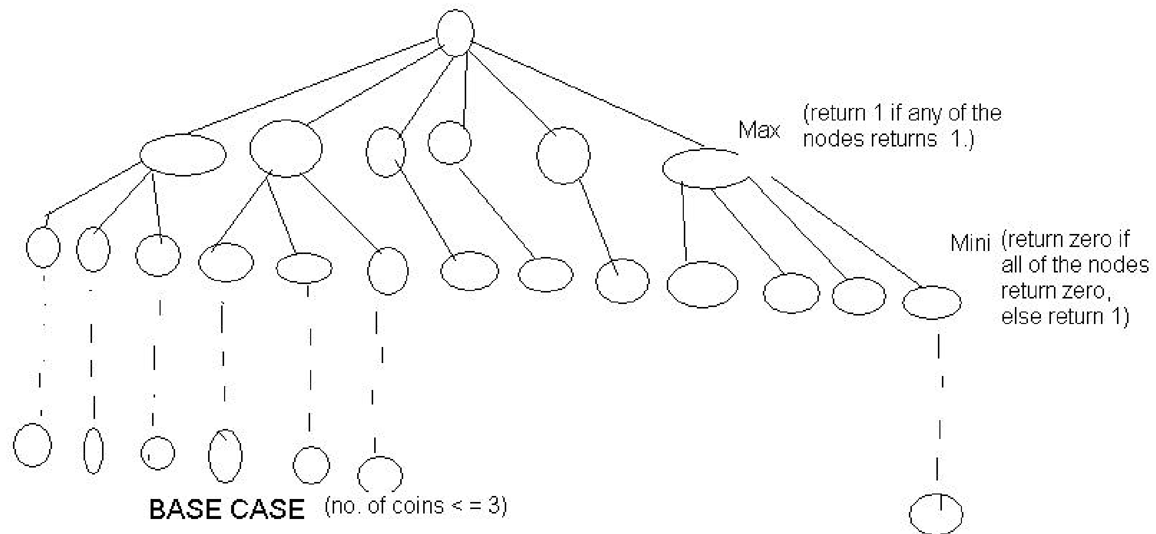
* We used an innovative mapping strategy for mapping each position on the board into the stored_values array.

* In exploring through the nodes of the tree we went through the nodes in the priority order using the priority [] array.

We will explain each feature in detail now.

<u>Min-Max Strategy:</u>

There will be a tree of all the positions possible for each player in the game (as shown in the figure below).There will be alternate levels for each player. And each level contains all the possible moves the player can take as nodes. Each node of the tree represents a position on the board. When 'we' want to find the best move we go through all the possible moves recursively.




Minimax Recursion Tree

**In this process of recursion when the program analyses the opposition player's chances it will return 1(successful) if all the next level positions return a one. i.e The opposite player has no position which leads him to a winning position(since all the next level positions are winning for 'we' returning one).**

**For analyzing the best move at our level we return one if we find at least one child returning one. Since In this case we can take that move and force the opposite player into a losing position.**

**We go on doing this recursively until the no. of coins on The board become less than 3.Then we use the Pattern Recognition strategy in finding the winning move as explained in the previous assignments(1,2)(Implemented using find_pattern () function).**

**This strategy is called min-max strategy since it takes the move which which maximises our score and minimises**

the opposite player's score. This strategy is responsible for us winning many games since It always takes the winning move.

## Greedy Points Scoring Algorithm:

This is the game played by a layman who does it unknowingly. But It is quite useful sometimes in fetching the points. If the opposite player is the first player then He may cleverly force us initially into a losing position.. In this case
It maximises the score by taking the move of higher priority (By the length of the move, using priority [] array).
Since there are not many extra points for winning the game. This may sometimes get us more points than the opposite
player even when losing the game.

## Dynamic Programming:

This is an innovative strategy used by us. It greatly reduces the time taken by each move. Since the tree has 126 branches at each node maximum there are totally N (computed below)

Number of nodes,
N=1+126+126*126+126*126*126+.......
== 1+126^1+126^2+126^3+.....(say m levels)
== {126^(m+1)-1}/125

nodes in the tree. Even if m==10 the no. of nodes is very large that they can't be explored in the prescribed time.

We will calculate the no. of distinct positions possible in the board. Since each coin can be there or not (2 ways) in 21 positions the total no. of distinct positions possible on the board are P= 2^21 positions.

Since The total number of distinct positions P=2^21 is very less than the total nodes
of the order of N=126^m We cash upon this overlapping sub problems property and we can gain a lot of time.

We declared an array of size 2^21 and mapped each position of the board to the array (stored_values []) The first time this position is analyzed the winning move from it is stored in it.

For subsequent accesses of this position this value is used rather than recursively computing it again.

Calculation of Moves:

Since we will always take a move from the possible 126 moves. we never give an invalid move (which fetches -50 points) generally. And the process of iterating through the moves will be speeded up largely. Since we store the validity of a move in status [] array we will only explore the moves which are valid by decreasing the tree size largely.

Mapping Strategy:

This is also an innovative and simple strategy used by us
for mapping all the positions onto the array stored_values which
stores the winning condition of the move. Since the number of
moves is 2^21 an array of it's size is created. Since we store
only a 0(losing) or(winning) 1 corresponding to a position we
need to use only a Boolean array of this size. But Since C doesn't
support a Boolean array, we created an integer array.
This can be implemented by a bit field structure in C.
Since It has an extra overhead of a structure we settled in using an
integer array.
We had a board [] array that contains a 1 if the
coin is present on the board.

In computing the array index of a position we add 2^i to the
Index if the coin is present in the position I (i.e board [i]=1).
Example, if the board array is
{ 0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 } then the
index is 2^1 + 2^3 = 9.
Therefore the index into which this position is mapped is 9.
This greatly reduces the time of computation since It only includes a
couple of left shifts(fast) and doesn't need any exponential operation.

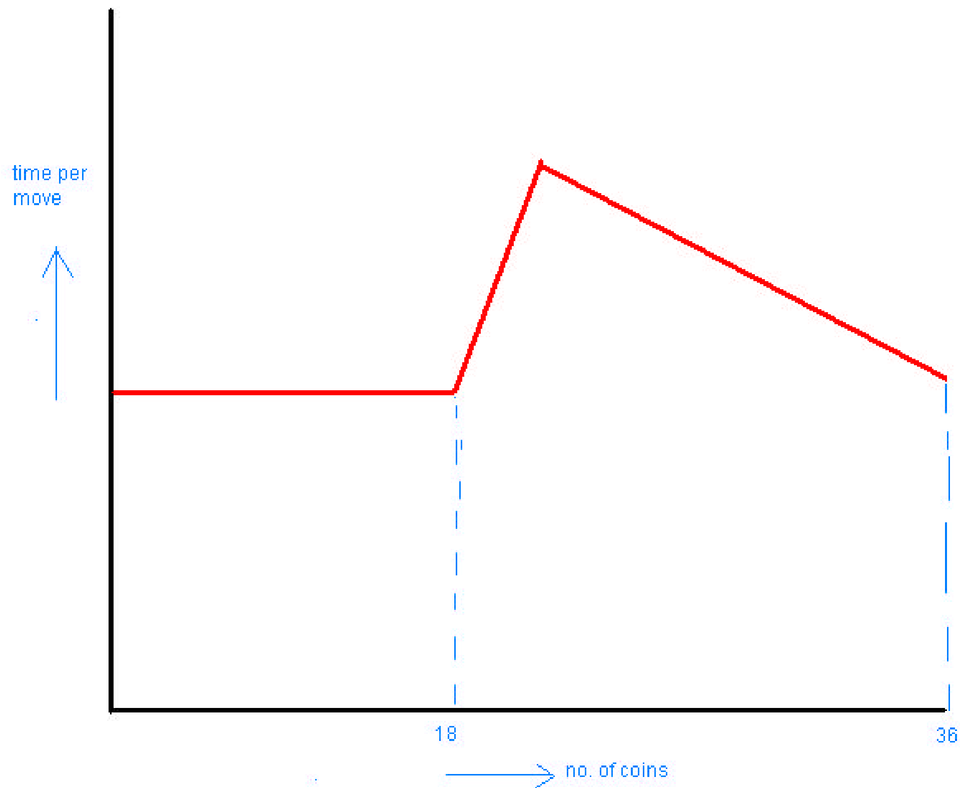
Times taken by the program:

1.Since we are using memoization to store the values of winning positions the time taken to compute the move after the first move is negligible.

**2.The time taken for first move is less than 60 sec when we act as the second player.**

**3.When we are the 1st player it doesn't take any significant time to compute any move.**

**4.The running times is best indicated using the graph shown below. Initially it moves either a predetermined move or a greedy move , which doesn't take any significant time. Afterwards it starts filling**
**the array that takes some time. After the whole array gets filled it takes**
**very less time since It has to just lookup from the array about the winning move.**




---

Strategy for assignment 4:

**Problem:**

* 36 coins in the form of a triangle on the board.
* If an invalid move is taken, negative score of -50(and opponent gets +50).
* Time limit is 90 sec.
* Scoring scheme is 1(3),2(7),3(-5),4(15),5(7),6(-20)
* Winner gets a bonus of 25 points and loser –25 points.

**Aim and Plan:**

- To maximize wins and score.
- Use a recursive min-max strategy to find the winning move.
- Maximize points if no winning move is found according the scoring scheme.
- Never take a move that is of length 6.
- Use of DYNAMIC PROGRAMMING to reduce the time for each move.
- Use of mapping when 18 coins are left and implement dynamic programming.
- Take greedily coins until 18 coins are obtained.

**Implementation:**

First we would like to highlight the important features of our strategy:

- It uses Min-Max strategy (Backtracking) in finding the best move by recursively going through all the possible moves.(implemented by recursion() function)

- It also acquires maximum points when It has found that It doesn't have any winning moves It will take the move GREEDILY which fetches the maximum no. of points.

- We used Dynamic Programming(Memoization) to store whether a position is winning or not.. We used it afterwards when this position again comes in the process of recursion instead of computing it again.. This strategy greatly reduces the time required for getting the move.

- Instead of checking all the moves(every permutation of coins) for validity everytime we computed all the 326

possible moves (in the tmp[][] array) and there is another array
status[326] which holds whether the corresponding
move is valid or not..

(Improvements on assignment 3:)

- ✓ 1.Here for using Memoization it requires a 2^36 array but an array of this

size is not supported in C. So till the no. of coins become 18 we greedily grab
coins that maximize points and then map the remaining positions into the array
stored_values[ ] and store the position's winning move**.**

- ✓ **2.We used an innovative mapping strategy for mapping each position on the board into the stored_values array.**

- **In exploring through the nodes of the tree we went through the nodes in the priority order using the priority[] array.**

We have already explained first 4 points in detail during the 3rd assignment. We explain the new
Feature (Mapping) developed by us:

The first one is a simple way of greedily grabbing the coins of the maximum
points. If the number of coins become less than or equal to 18 we map it in the
following way into the array stored_values [].

Mapping Strategy:

Since the number of ways in which the coins on the board become 18 is 36 C 18(combinatorics).
And the number of distinct positions on the board is 2^21,If we multiply this by 36 C 18 this
amounts to a large array.

If we map the position that we arrived when the number of coins become 18

into a 18 board array (implemented by map [] array)then we can implement the
dynamic programming strategy in a array of 2^18 only.

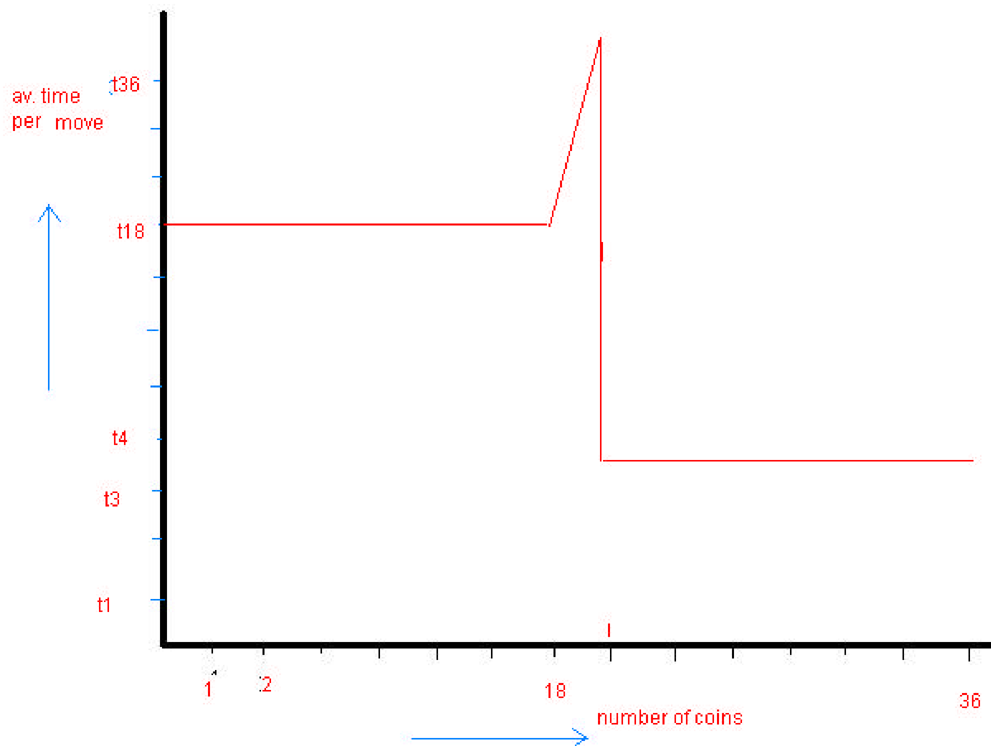
Times taken by the program:

1.When we are the first player or second player, Until the no. of coins become less than
19 It takes very less time since It takes the move according to priority rather than going for recursion

2.When the no. of coins become 18,It will take at the maximum 45 sec.
But On the average It takes less than 20 sec. This is because
Only for those arrangements where the no. of possible moves still left is very high It takes large time(Ex. when all the coins are taken from
a corner).But this happens very rarely

3.After one move that takes long time, It takes very less time
since the array has already been filled and only a lookup into
it does the work.

4.The graph would look like this:




Strategy for Assignment 5:

**Problem:**

* 21 coins in the form of a triangle on the board
* If an invalid move is taken, negative score of -50.
* Time limit is 30 sec.
* Scoring scheme is 1(3),2(7),3(-5),4(5),5(7),6(-20).
* For red coin –5, green +5,yellow 2,white 1 extra points.
* Winner gets a bonus of 25 points.

**Aim and Plan:**

- To maximize wins and score.
- Use a recursive min-max strategy to find the winning move.
- Maximize points If no winning move is found according the scoring scheme.
- Use of the moves array in the order of priority of points.

- **Use of DYNAMIC PROGRAMMING to reduce the time for each move.**
- **Use of mapping when 18 coins are left and implement dynamic programming.**
- **Take greedily coins until 18 coins are obtained**

Implementation:

First we would like to highlight the important features of our strategy:

- It uses Min-Max strategy (Backtracking) in finding the best move by recursively going through all the possible moves.(implemented by recursion() function)

- It also acquires maximum points when It has found that It doesn't have any winning moves It will take the move GREEDILY which fetches the maximum no. of points.

- We used Dynamic Programming(Memoization) to store whether a position is winning or not.. We used it afterwards when this position again comes in the process of recursion instead of computing it again.. This strategy greatly reduces the time required for getting the move.

- Instead of checking all the moves(every permutation of coins) for validity everytime we computed all the 326 possible moves (in the tmp[][] array) and there is another array status[326] which holds whether the corresponding move is valid or not..

(Improvements on assignment 3:)

- Here for using Memoization it requires a 2^36 array but an array of this size is not supported in C. So till the no. of coins become 18 we greedily grab coins that maximize points and then map the remaining positions into the array stored_values[ ] and store the position's winning move.

- We used an innovative mapping strategy for mapping each position on the board into the stored_values array.

- In exploring through the nodes of the tree we went through the nodes in the priority order using the priority[] array.

(Improvements upon assignment 4:)

- ✓ To incorporate the effect of the different colored coins

we first calculated the points associated with each move in the array of moves(tmp[326][6]).We then sorted the array based on the points and always
go through this order while searching through the moves. This helps in giving
the move that gives the maximum points and also speeds up the process of
exploring through the tree.

<u>Times taken by the program</u>:

1.When we are the second or first player, till the no. of coins become 18
we take the move of highest priority taking very less significant time.

2.When the no. of coins become 18 ,For the first move It will take a maximum
of 40 sec some time less than that in the previous assignment
due to the improvement stated above

3.After the first move that takes some time there will be negligible time
taken by the program for subsequent moves due to use of DYNAMIC PROGRAMMING

4.The graph would look similar to the one in the assignment 4, but the time taken by
the move (at 18 coin stage) will be less than that in the previous assignment
due to the improvement stated above.