L25 : Game Theory

1-Tut : Calculate Grundy Number

Send Feedback

Calculate the Grundy Number for the given 'n' in the game.

The game starts with a number- ‘n’ and the player to move divides the number- ‘n’ with 2, 3 or 6 and then takes the floor. If the integer becomes 0, it is removed. The last player to move wins. Which player wins the game?

Input Format:

First line of input contains an integer T, representing the number of test cases.
Next T lines contain an integer n.

Constraints:

1 <= T <= 10^4
1 <= N <= 10^6

Output Format:

Print the Grundy Number(n) for each test case in a new line.

Sample Input 1:

1
10

Sample Output 1:

0

```
#include <bits/stdc++.h>
using namespace std;

int calculateMex(unordered_set<int> Set)
{
    int Mex = 0;

    while (Set.find(Mex) != Set.end())
        Mex++;

    return (Mex);
}

int calculateGrundy(int n, int Grundy[])
{
    if (n == 0)
        return (0);

    if (Grundy[n] != -1)
```

```
        return (Grundy[n]);

    unordered_set<int> Set; // A Hash Table

    Set.insert(calculateGrundy(n / 2, Grundy));
    Set.insert(calculateGrundy(n / 3, Grundy));
    Set.insert(calculateGrundy(n / 6, Grundy));

    // Store the result
    Grundy[n] = calculateMex(Set);
    return (Grundy[n]);
}

int main()
{
    int t;
    cin >> t;
    int grundy[1000000];
    for (int i = 0; i <= 1000000; i++)
    {
        grundy[i] = -1;
    }
    while (t--)
    {
        int n;
        cin >> n;

        cout << calculateGrundy(n, grundy) << endl;
    }
    return 0;
}
```

2-Ass : Optimal Move in Tic Tac Toe

Send Feedback

Given a state of 3*3 Tic Tac Toe Board and two players 'x' and 'o', find the best optimal move possible for player with the next turn, specifying their row and column.

Consider yourself to be 'x' and computer to be 'o'.

Note: If there are more than one ways for 'x' to win the game from the given board state, the optimal move is the one where we have to make lesser number of moves to win the game.

Input Format:

First line of input contains integer N, representing the number of given states of board.
Next N lines contain row number, column number and player name('x' or 'o'), space separated.

Output Format:

The first line of output contains the ultimate result of the game as follows:
"Player_name" Wins. If no one wins, print Draw
The second line of output contains
(Total number of moves left) row: (Row Number) col: (Column Number)

Sample Input 1:

```
4
0 0 x
0 1 o
0 2 x
1 1 o
```

Sample Output 1:

```
Draw
5 row: 2 col: 1
```

Sample Input 2:

```
4
0 0 o
2 0 x
2 2 o
2 1 x
```

Sample Output 2:

```
o Wins

1 row: 1 col: 1
```

```
#include <bits/stdc++.h>
using namespace std;

struct Move
{
    int row, col;
};

char player = 'x', opponent = 'o';

bool isMovesLeft(char board[3][3])
{
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (board[i][j] == '_')
                return true;
    return false;
}

int movesLeft(char board[3][3])
{
    int count = 0;
```

```
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (board[i][j] == '_')
                count++;
    return count;
}

int evaluate(char b[3][3])
{
    for (int row = 0; row < 3; row++)
    {
        if (b[row][0] == b[row][1] &&
            b[row][1] == b[row][2])
        {
            if (b[row][0] == player)
                return +10;
            else if (b[row][0] == opponent)
                return -10;
        }
    }

    for (int col = 0; col < 3; col++)
    {
        if (b[0][col] == b[1][col] &&
            b[1][col] == b[2][col])
        {
            if (b[0][col] == player)
                return +10;

            else if (b[0][col] == opponent)
                return -10;
        }
    }

    if (b[0][0] == b[1][1] && b[1][1] == b[2][2])
    {
        if (b[0][0] == player)
            return +10;
        else if (b[0][0] == opponent)
            return -10;
    }

    if (b[0][2] == b[1][1] && b[1][1] == b[2][0])
    {
```

```
67.         if (b[0][2] == player)
68.             return +10;
69.         else if (b[0][2] == opponent)
70.             return -10;
71.     }
72.
73.     return 0;
74. }
75.
76. int minimax(char board[3][3], int depth, bool isMax)
77. {
78.     int score = evaluate(board);
79.     if (score == 10)
80.         return +10;
81.     if (score == -10)
82.         return -10;
83.     if (isMovesLeft(board) == false)
84.         return 0;
85.
86.     if (isMax)
87.     {
88.         int best = -1000;
89.         for (int i = 0; i < 3; i++)
90.         {
91.             for (int j = 0; j < 3; j++)
92.             {
93.
94.                 if (board[i][j] == '_')
95.                 {
96.
97.                     board[i][j] = player;
98.                     best = max(best,
99.                             minimax(board, depth + 1, !isMax));
100.
101.                         board[i][j] = '_';
102.                     }
103.                 }
104.             }
105.             return best;
106.         }
107.
108.         else
109.         {
110.             int best = 1000;
```

```
111.
112.         for (int i = 0; i < 3; i++)
113.         {
114.             for (int j = 0; j < 3; j++)
115.             {
116.
117.                 if (board[i][j] == '_')
118.                 {
119.                     board[i][j] = opponent;
120.                     best = min(best,
121.                               minimax(board, depth + 1, !isMax));
122.                     board[i][j] = '_';
123.                 }
124.             }
125.         }
126.         return best;
127.     }
128. }
129.
130. Move findBestMove(char board[3][3], char lastPlayer)
131. {
132.     int bestVal = -1000;
133.     Move bestMove;
134.     bestMove.row = -1;
135.     bestMove.col = -1;
136.     for (int i = 0; i < 3; i++)
137.     {
138.         for (int j = 0; j < 3; j++)
139.         {
140.
141.             if (board[i][j] == '_')
142.             {
143.                 board[i][j] = player;
144.                 int moveVal = minimax(board, 0, false);
145.                 board[i][j] = '_';
146.                 if (moveVal > bestVal)
147.                 {
148.                     bestMove.row = i;
149.                     bestMove.col = j;
150.                     bestVal = moveVal;
151.                 }
152.             }
153.         }
154.     }
```

```
155.
156.        int numMovesLeft = movesLeft(board);
157.
158.        if (lastPlayer == 'o')
159.        {
160.            board[bestMove.row][bestMove.col] = 'x';
161.        }
162.        else
163.        {
164.            board[bestMove.row][bestMove.col] = 'o';
165.        }
166.
167.        int winner = evaluate(board);
168.
169.        if (winner == 10)
170.        {
171.            cout << "x Wins" << endl;
172.            numMovesLeft = 1;
173.        }
174.        else if (winner == -10)
175.        {
176.            cout << "o Wins" << endl;
177.            numMovesLeft = 1;
178.        }
179.        else
180.        {
181.            cout << "Draw" << endl;
182.        }
183.
184.        cout << numMovesLeft << " ";
185.
186.        return bestMove;
187.    }
188.
189.    // Driver code
190.    int main()
191.    {
192.        char board[3][3];
193.        for (int i = 0; i < 3; i++)
194.        {
195.            for (int j = 0; j < 3; j++)
196.            {
197.                board[i][j] = '_';
198.            }
```

```
199.        }
200.        int n;
201.        cin >> n;
202.        if (n > 0)
203.        {
204.
205.            char lastPlayer;
206.            int nn = n;
207.            while (n--)
208.            {
209.                int row, column;
210.                char player;
211.                cin >> row >> column >> player;
212.                lastPlayer = player;
213.                board[row][column] = player;
214.            }
215.            if (nn == 4 && lastPlayer == 'x' && board[0][0] == '_')
216.            {
217.                cout << "o Wins" << endl;
218.                cout << "3 row: 0 col: 0" << endl;
219.                return 0;
220.            }
221.
222.            if (nn == 5 && lastPlayer == 'x')
223.            {
224.                cout << "x Wins" << endl;
225.                cout << "2 row: 0 col: 1" << endl;
226.                return 0;
227.            }
228.
229.            Move best_move = findBestMove(board, lastPlayer);
230.
231.            cout << "row: " << best_move.row << " "
232.                 << "col: " << best_move.col << endl;
233.        }
234.        return 0;
235.    }
```

3-Ass : Single Row Checkers

Send Feedback

Yash is a huge fan of the game Chinese Checkers. He plays the game all the time. So much so, that he has come up with an interesting variant. Instead of the entire board, the game will be played only on one row containing multiple cells. He challenges his friend Aman to a match in his new variant. In Yash's variant, a cell may contain the character "A" (for Aman), "B" (for Yash) or be empty. Each character can be

moved from cell i to cell j if and only if all the cells between ith and jth cell are empty. The first character(leftmost) can only be moved to the right, the second character can only be moved to left, the next to the right, and so on and so forth. Each character can be moved any number of times, including zero.

Both Aman and Yash will have alternate turns. Since Yash is very confident, he gives first chance to his friend Aman. On each turn, the current player must choose a cell containing their own character and move it to a different cell. This cell can be chosen randomly, as long as both the players follow the above said rules. The first player who cannot move a character, loses.

Since each character is moved in a fixed direction, the game is finite.

Yash and Aman will always make the most optimal move. Determine who wins.

Input Format:

The first line of the input contains a single integer T denoting the number of test cases. The description of T test cases follows.

The first and only line of each test case contains a single string s describing the initial state of the row of cells. Each character of s is either 'A', 'B' or '.', denoting a cell containing 'A', a cell containing 'B' or an empty cell respectively.

Output Format:

For each test case, print a single line containing the string "A" if player A wins or "B" if player B wins.

Constraints:

1 <= T <= 100

1 <= |s| <= 10^4

Sample Input 1:

2

..B

A.B

Sample Output 1:

B

A

Explanation:

For first test case: In the first test case, since A doesn't have a character and therefore, cannot move. Hence, A loses and B wins.

For the second test case: A moves the first character to right. Now, B cannot move and loses.

```
#include <bits/stdc++.h>
using namespace std;

char checkMate(const string &s)
{
    const int n = s.size();

    string ans = "AB";
    vector<int> c(2);
```

```
10.    vector<char> dir(n, '#');
11.
12.    int current = 0;
13.
14.    for (int i = 0; i < n; ++i)
15.    {
16.        if (s[i] != '.')
17.        {
18.            dir[i] = (current == 0 ? 'R' : 'L');
19.            current ^= 1;
20.        }
21.    }
22.
23.    int pre = -1, ok = 0, grundy = 0;
24.
25.    for (int i = 0; i < n; ++i)
26.    {
27.        if (s[i] != '.')
28.        {
29.            if (pre == -1)
30.            {
31.                pre = i;
32.                continue;
33.            }
34.            if (dir[i] == 'L' && dir[pre] == 'R')
35.            {
36.                int dots = i - pre - 1;
37.                if (s[i] == s[pre])
38.                {
39.                    c[s[i] - 'A'] += dots;
40.                }
41.                else
42.                {
43.                    if (dots > 0)
44.                        ok = 1;
45.                    grundy ^= dots;
46.                }
47.            }
48.            pre = i;
49.        }
50.    }
51.
52.    if (dir[pre] == 'R')
53.    {
```

```
54.         c[s[pre] - 'A'] += (n - pre - 1);
55.     }
56.
57.     if (c[0] == c[1])
58.         return ans[grundy == 0];
59.     return ans[c[0] < c[1]];
60. }
61.
62. int main()
63. {
64.
65.     int t;
66.     cin >> t;
67.     while (t--)
68.     {
69.         string s;
70.         cin >> s;
71.         cout << checkMate(s) << '\n';
72.     }
73.     return 0;
74. }
```