CRACK A HACK

Ways To Give A Check

Course: ALGORITHMIC PROBLEM SOLVING

Course code: 17ECSE309

by:

Tejas Arlimatti

USN: 01FE15BEC211

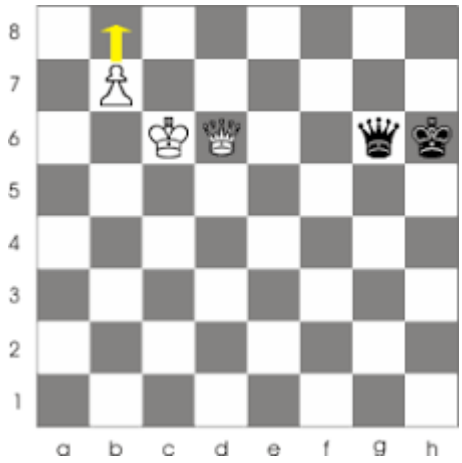
1. Introduction

A chess engine consists of several components. In this coding problem, the task is to implement a simple pawn promotion component of a chess engine that can be incorporated into a chess engine. The concept of pawn promotion is described in detail at this page. Therefore, long story short, Promotion is a chess rule that requires a pawn that reaches its eighth rank to be immediately replaced by the player's choice of a queen, knight, rook, or bishop of the same color.

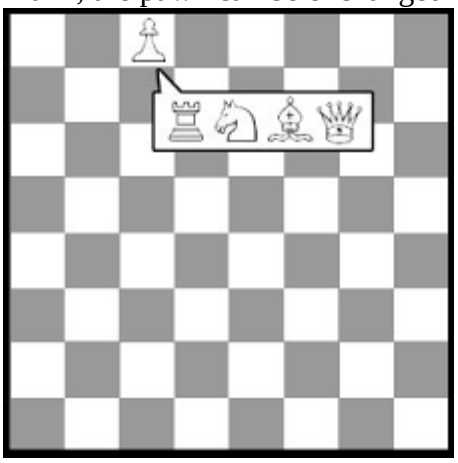
Here are a couple of illustrations to help with the understanding :

A pawn-promotion is possible if and only if→

- The eight rank (the rank higher than that of the pawn in question) is empty or contains no chess piece.
- The King of the pawn under question is not under a check.



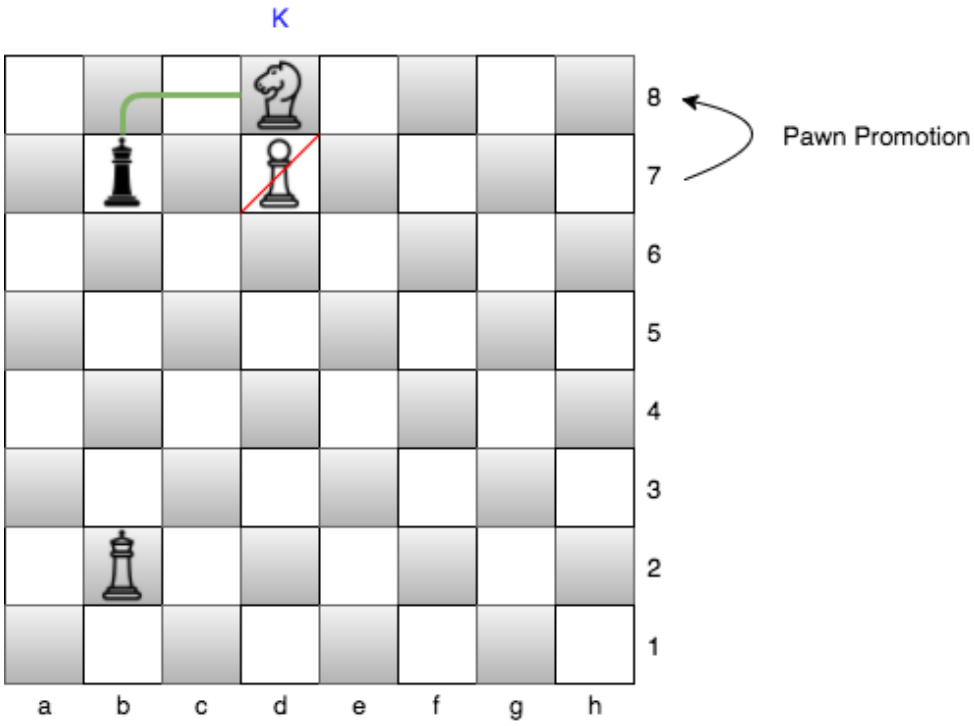
Once the pawn is promoted to the 8th rank, the pawn can be exchanged with the following pieces :



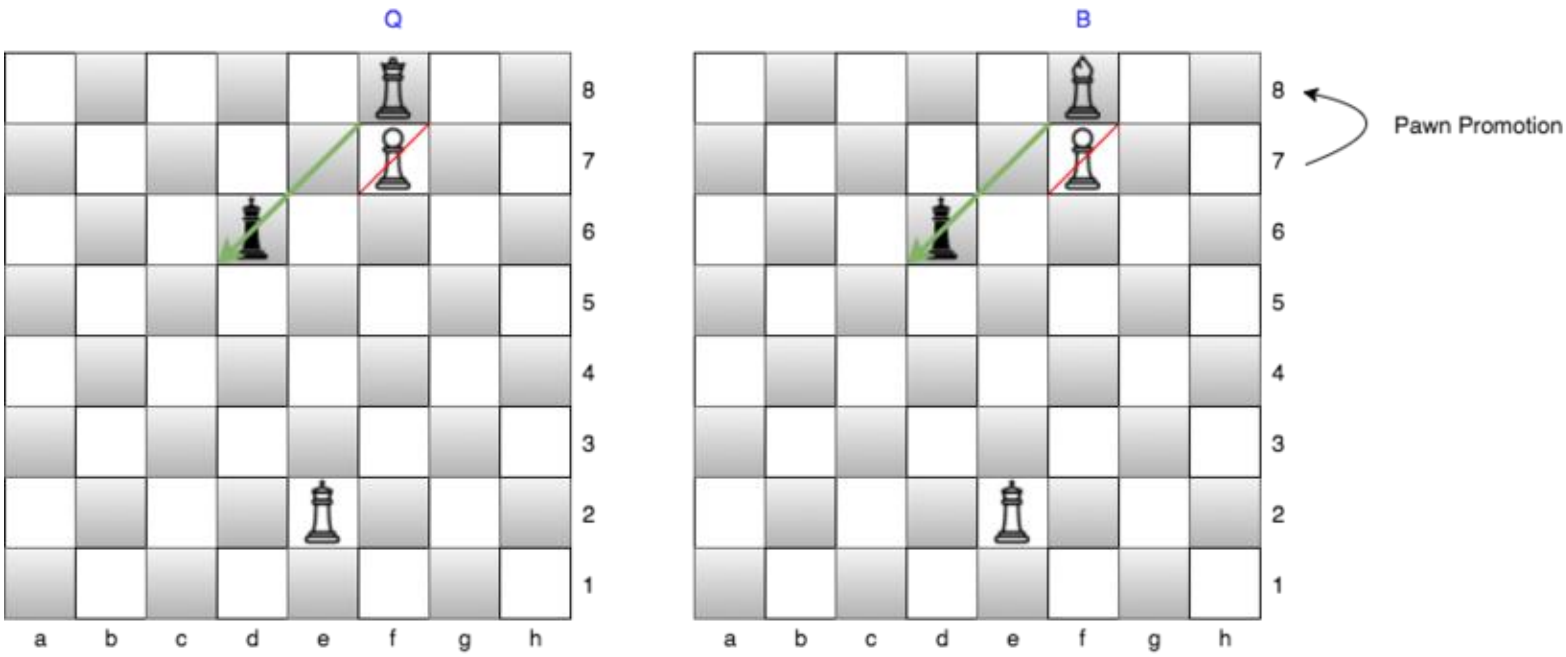
Hence, the pawn is promoted to a Queen, a Knight, a Rook, or a Bishop.
As far as this problem is concerned, we have been asked to determine in how many ways a pawn in the 7th rank can be promoted to the 8th rank that result in a check to the opponent's King.

2. Example

Knight giving check :



Queen and Bishop (respectively) giving check :



3. Algorithm

A concept needed to understand the forthcoming code:

Discovered checks :

If the movement of any of the chess pieces of a player introduces an opportunity for the player to check the opponent, it is called a discovered check. It is called so because the check opportunity only arises after the movement of the aforementioned chess piece.

These kind of checks are to be considered in the solution of this problem to satisfy all test cases.

Also, one needs to realize that checks here can arise from any chess piece on the board, and not necessarily from the promoted piece alone. This, and the ***discovered checks*** concepts' inclusion in the problem weren't explicitly mentioned in the problems description.

1) The pawns in the 7^{th} rank, which do not have a chess piece in the immediate higher rank (8^{th} rank) are marked.
2) Now, for each of these pawns, the immediate higher position (8^{th} rank position) is replaced, one at a time, with the Queen, a Knight, a Bishop, and a Rook (will be referenced as the 'QKBR' henceforth for convenience). The initial position of the pawn is made vacant on the board.
3) Each of the above mentioned scenarios is tested for Checks on the opposing king in the following manner :
   - Every chess piece on the board (be it a pawn, or the 'QKBR') is scrutinized for moves that could result in a check on the opposing King. If any check is found, a counter that is keeping track of the checks is incremented.

   This method is especially convenient because it also takes care of ***discovered checks.***

4. Code

```
#include <bits/stdc++.h>

using namespace std;

bool check(const vector < vector<char> > &board) {
   for (int i = 0; i < 8; ++i) {
      for (int j = 0; j < 8; ++j) {
         if (board[i][j] == 'N') {
            for (int dx : {-2, -1, 1, 2}) for (int dy : {-2, -1, 1, 2}) if (abs(dx) != abs(dy)) {
```

```
                int x = i + dx, y = j + dy;
                if (x < 0 || x >= 8 || y < 0 || y >= 8) continue;
                if (board[x][y] == 'k') return true;
            }
        } else if (board[i][j] == 'P') {
            int x = i - 1;
            for (int y : {j-1, j + 1}) if (x >= 0 && y >= 0 && y < 8) {
                if (board[x][y] == 'k') return true;
            }
        }
        if (board[i][j] == 'R' || board[i][j] == 'Q') {
            for (int dx : {-1, 0, 1}) for (int dy : {-1, 0, 1}) if (abs(dx) != abs(dy)) {
                for (int x = i + dx, y = j + dy; x >= 0 && x < 8 && y >= 0 && y < 8; x += dx, y += dy) {
                    if (board[x][y] == 'k') return true;
                    if (board[x][y] != '#') break;
                }
            }
        }
        if (board[i][j] == 'B' || board[i][j] == 'Q') {
            for (int dx : {-1, 1}) for (int dy : {-1, 1}) {
                for (int x = i + dx, y = j + dy; x >= 0 && x < 8 && y >= 0 && y < 8; x += dx, y += dy) {
                    if (board[x][y] == 'k') return true;
                    if (board[x][y] != '#') break;
                }
            }
        }
    }
  }
  return false;
}

int waysToGiveACheck(const vector < vector<char> > &board) {
  int res = 0
  string v = "QRBN";
  for (int i = 0; i < 8; ++i) if (board[1][i] == 'P' && board[0][i] == '#') {
    auto b = board;
    b[1][i] = '#';
    for (char c : v) {
      b[0][i] = c;
      res += check(b);
    }
  }
  return res;
}



int main() {
  int t;
  cin >> t;
  for(int a0 = 0; a0 < t; a0++){
    vector< vector<char> > board(8,vector<char>(8));
    for(int board_i = 0;board_i < 8;board_i++){
      for(int board_j = 0;board_j < 8;board_j++){
```

```
            cin >> board[board_i][board_j];
          }
        }
        int result = waysToGiveACheck(board);
        cout << result << endl;
    }
    return 0;
}
```

5. Time complexity

The time complexity of the above code is O (|P| + |A|), where ‘P’ denotes the number of pawns, and ‘A’ denotes all instances of ‘QKBR’ on the board.

6. Applications

- It is used in several chess engines like Stockfish, Komodo to enable the AI to play against an user.
- Such components can be integral to chess engines that can help calculate the best moves from any position.
- The engines formed by such components could provide assistance to the user and to help a user identify potential fallacies.

7. References

1) https://stockfishchess.org/
2) https://komodochess.com/
3) https://chess.com/

4. Code (in C++)

```
#include <map>
#include <set>
#include <list>
#include <cmath>
#include <ctime>
#include <deque>
#include <queue>
#include <stack>
#include <string>
#include <bitset>
#include <cstdio>
#include <limits>
#include <vector>
#include <climits>
#include <cstring>
#include <cstdlib>
#include <fstream>
#include <numeric>
#include <sstream>
#include <iostream>
#include <algorithm>
#include <unordered_map>
using namespace std;

#define INF INT_MAX/2

int mini(int a, int b){
    if(a<b)
        return a;
    return b;
}

void calculatedistance(int nodes, vector<vector<int>> mat, int
    start){ int i,j,k,temp,u,min;
    int dist[nodes+1];
    for(i=1;i<nodes+1;i++){ //distance array intialization
        dist[i]=mat[start][i];

    }
    int path[nodes+1];
    for(i=1;i<nodes+1;i++){     //path array intialization
        path[i]=start;
    }
    set<int> R;
    set<int>::iterator itr;
    for(i=1;i<nodes+1;i++){
```

```
        R.insert(i);
    }
    itr=R.find(start);
    R.erase(itr);
    while(!R.empty()){
        min=INF+1;
        for(itr=R.begin();itr!=R.end();itr++){
            if(dist[*itr]<min){
                min=dist[*itr];
                u=*itr;
            }
        }
        itr=R.find(u);
        R.erase(itr);

        min=INF;
        for(itr=R.begin();itr!=R.end();itr++){
            temp=mini(dist[*itr],dist[u]+mat[u][*itr]);
            if(temp<dist[*itr]){
                dist[*itr]=temp;
                path[*itr]=u;
            }
        }
    }

    for(i=1;i<nodes+1;i++){
        if(i!=start && dist[i]!=INF)
            cout<<dist[i]<<" ";
        else if(dist[i]==INF)
            cout<<"-1 ";
    }
    cout<<endl;
}

int main(){
    int t;
    cin>>t;
    int i,j,k,a,b,c;
    for(i = 0; i < t; i++){
        int nodes;
        int edges;
        cin >>nodes>>edges;
        vector<vector<int>> mat(nodes+1,vector<int>(nodes+1,INF));
        for(j=1;j<nodes+1;j++){
            mat[j][j]=0;
        }
        for(j=0;j<edges;j++){
            cin>>a>>b>>c;
            if(mat[a][b]==INF || mat[a][b]>c){ //there might be multiple edges between same 2 nodes
                mat[a][b]=c;
```

```
            mat[b][a]=c;
        }

    }
    int s;
    cin>>s;
    calculatedistance(nodes,mat,s);
    mat.clear();
  }
  return 0;
}
```

5. Time Complexity

The time complexity of the above code is O(ElogV), Where ‘E’ is the number of edges and ‘V’ is the number of vertices.

6. Applications

It is used in geographical Maps.

- To find locations of Map which refers to vertices of graph.
- Distance between the location refers to edges.

It is used in IP routing to find Open shortest Path First.

It is used in the telephone network.

7. References

1) https://www.researchgate.net
2) https://www.cs.umd.edu