Instituto Tecnológico de Aeronáutica – ITA

Graduação

CTC-17: Inteligencia Artificial

Bruno De Souza Neves , Rahyan Azin

12 de setembro de 2018

1 Menor caminho com Algoritmos A* e Greedy

Basicamente , leu-se os dados das cidades e os mesmos foram processados e armazenados numa lista de adjacência . No A* criou-se um struct ”node” para armazenar informações de x,y,f,g,id . Em todos os casos usou-se uma priority queue para determinar quais nós devem ser processados em cada passo . Essa estrutura é boa principalmente por recuperar em O(lgn) com menor f . Os resultados foram os seguintes :

```
Tempo de execucao: 833
Caminho A* :
Alice Springs -- Andamooka -- Armidale -- Ayr -- Ballarat -- Bairnsdale East --
    Ballina -- Batemans Bay -- Bathurst -- Bendigo -- Bicheno -- Birdsville --
    Bordertown -- Bourke -- Brisbane -- Broome -- Bundaberg -- Burnie -- Byron Bay --
    Cairns -- Caboolture -- Caloundra -- Canberra -- Ceduna -- Charleville -- Clare --
    Cobram -- Colac -- Cowell -- Cranbourne -- Dalby -- Deniliquin -- Dubbo -- East
    Maitland -- Eidsvold -- Esperance -- Forbes -- Gawler -- Georgetown -- Gingin --
    Geraldton -- Gladstone -- Goondiwindi -- Griffith -- Gympie South -- Hamilton --
    Hobart -- Hughenden -- Inverell -- Kalbarri -- Karratha -- Katanning -- Katoomba
    -- Kiama -- Kimba -- Kingoonya -- Kingston South East -- Kwinana -- Laverton --
    Leonora -- Longreach -- Manjimup -- Maryborough -- Meekatharra -- Melton --
    Melbourne -- Meningie -- Mildura -- Morawa -- Mount Barker -- Mount Isa -- Mudgee
    -- Muswellbrook -- Narrabri West -- Newcastle -- Norseman -- North Mackay -- North
    Lismore -- North Scottsdale -- Nowra -- Oatlands -- Orange -- Pambula -- Parkes --
    Perth -- Penola -- Peterborough -- Port Augusta West -- Port Douglas -- Port
    Lincoln -- Port Pirie -- Portland -- Queanbeyan -- Quilpie -- Richmond --
    Rockhampton -- Roma -- Scone -- Shepparton -- Seymour -- Singleton -- South
    Grafton -- South Melbourne -- Stawell -- Sunbury -- Sydney -- Thargomindah --
    Three Springs -- Toowoomba -- Traralgon -- Tumut -- Ulladulla -- Wagga Wagga --
    Wallaroo -- Wangaratta -- Warwick -- West Tamworth -- Wilcannia -- Winton --
    Windorah -- Wollongong -- Woomera -- Yeppoon -- Yulara
Numero de nos percorridos: 124

Custo: 1497.58
```

No Greedy, como esperado , não foi calculado o melhor caminho , porém seu custo computacional foi menor que o A* , pois este gasta mais tempo para computar a heurística.

```
Tempo de execucao: 269
Alice Springs -- Ararat -- Atherton -- Ayr -- Ballarat -- Barcaldine -- Bathurst --
    Bendigo -- Bedourie -- Berri -- Biloela -- Birdsville -- Bordertown -- Bourke --
    Boulia -- Bowen -- Broken Hill -- Bunbury -- Burketown -- Busselton -- Caboolture
    -- Cairns -- Camooweal -- Carnarvon -- Canberra -- Ceduna -- Charleville -- Clare
    -- Cobram -- Colac -- Cowell -- Cranbourne -- Dalby -- Deniliquin -- Dubbo --
    Echuca -- Emerald -- Esperance -- Forbes -- Gawler -- Georgetown -- Gingin --
    Geraldton -- Gladstone -- Goondiwindi -- Griffith -- Gunnedah -- Halls Creek --
    Hervey Bay -- Horsham -- Innisfail -- Ivanhoe -- Kalgoorlie -- Karumba --
    Katherine -- Katanning -- Katoomba -- Kiama -- Kimba -- Kingoonya -- Kingston
    South East -- Kwinana -- Laverton -- Leonora -- Longreach -- Manjimup --
    Maryborough -- Meekatharra -- Melton -- Merredin -- Meningie -- Mildura -- Morawa
    -- Mount Barker -- Mount Isa -- Mount Magnet -- Murray Bridge -- Narrabri West --
    Muswellbrook -- Narrogin -- Newman -- Norseman -- North Mackay -- Northam -- North
    Scottsdale -- Nowra -- Onslow -- Ouyen -- Pannawonica -- Penola -- Peterborough --
    Port Augusta West -- Port Douglas -- Port Lincoln -- Port Pirie -- Portland --
```

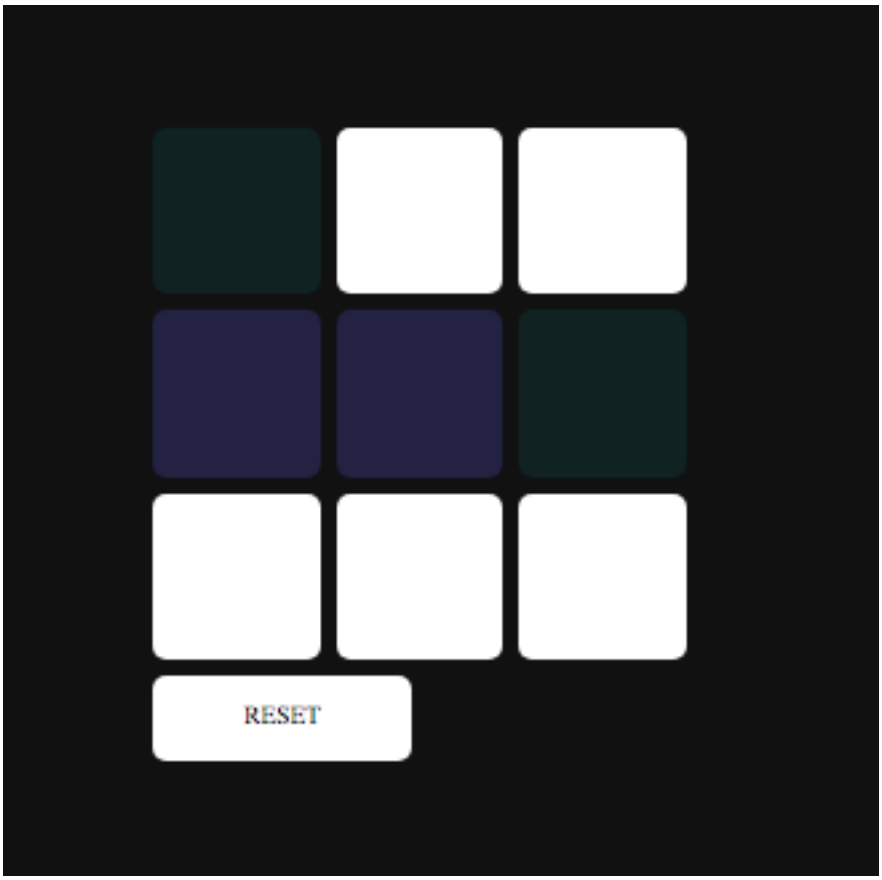
```
    Queanbeyan -- Quilpie -- Richmond -- Rockhampton -- Roma -- Roebourne -- Sale --
    Seymour -- Shepparton -- Smithton -- South Ingham -- Southern Cross -- Streaky Bay
    -- Swan Hill -- Sunbury -- Sydney -- Thargomindah -- Three Springs -- Tom Price --
    Townsville -- Tumby Bay -- Traralgon -- Tumut -- Ulladulla -- Victor Harbor --
    Wagin -- Wangaratta -- Wallaroo -- Warrnambool -- Weipa -- Whyalla -- Windorah --
    Wollongong -- Woomera -- Yeppoon -- Yulara
Numero de nos percorridos: 132

Custo: 2092.73
```

Em ambos os casos o custo foi calculado ao armazenar a rota em um map(int,int) father , tal que se father[u] = v , v é o nó que precede u , e soma-se as distâncias euclidianas até chegar-se à origem.

2 Jogo da Velha

O algoritmo minimax é um dos mais antigos algoritmos de inteligência artificial de todos os tempos. Ele usa uma regra simples de soma zero para descobrir qual jogador ganhará dada uma posição atual. Para a interface gráfica usou-se HTML e CSS , manipulados pelo JavaScript. Para iniciar o jogo basta abrir o arquivo index.html por meio de algum browser. No arquivo index.js encontra-se toda a lógica do jogo , onde foi implementada uma versão recursiva do minMax e funcões auxiliares para avaliar se houve ganhador, para reiniciar o jogo ,etc. A funcão recursiva analisa , percorrendo uma árvore, se sob a perspectiva do jogador ele ganha ou não , de forma a minimizar esse resultado. Com a implementação do minMax o jogador humano ou perde ou empata.



index.html

```
<!DOCTYPE html>
```

```
<html>
<head>
   <meta charset="UTF-8">
   <title>Velha[U+FFFD]s Game</title>
   <link rel="stylesheet" type="text/css" href="style.css">
   <script type="text/javascript" src = "logic.js"></script>
</head>
<body>
   <div id = "main">
     <div id = "board">
        <div class = "quad" onclick="req(this)"></div>
        <div class = "quad" onclick="req(this)"></div>
        <div class = "quad" onclick="req(this)"></div>
        <div class = "quad" onclick="req(this)"></div>
        <div class = "quad" onclick="req(this)"></div>
        <div class = "quad" onclick="req(this)"></div>
        <div class = "quad" onclick="req(this)"></div>
        <div class = "quad" onclick="req(this)"></div>
        <div class = "quad" onclick="req(this)"></div>
        <div class = "button" onclick="resetar()">Reset</div>
     </div>
   </div>
</body>
</html>
```

logic.js

```
var quads = document.getElementsByClassName("quad")
var buttons = document.getElementsByClassName("button")
var estado = [0,0,0,0,0,0,0,0,0]
var game = true
var humano = false
var comp = true
var humanoVal = -1
var compVal = 1
//todas as jogadas vencedoras
var winMat = [
   3
];
[0,1,2],
[3,4,5],
[6,7,8],
[0,3,6],
[1,4,7],
[0,4,8],
[2,4,6],
[2,5,8]
function resetar(){
   for(var x = 0;x<9;x++){
     quads[x].style.background = "#fff"
     estado[x] = 0
   }
game = true }
function req(clicked){
   if(!game) return;
```

```
   for(var x = 0;x<9;x++){
     if(quads[x]==clicked && estado[x]==0){
        set(x,humano)
chamaIA() }
} }
function set(ind,player){
   if(!game) return
    if(estado[ind]==0){
     if(player==humano){
        quads[ind].style.background = "#224"
        estado[ind] = humanoVal
} else{
        quads[ind].style.background = "#122"
        estado[ind] = compVal
     }
     if(checkWin(estado,player)){
        game = false;
} }
}
function checkWin(board,player){
   var value = player == humano ? humanoVal : compVal;

for(var x = 0;x<8;x++){
     var win = true;
     for(var y = 0;y<3;y++){
        if(board[winMat[x][y]] != value){
           win = false;
           break;
} }
     if(win) {
        return true;
} }
   return false;
}
function checkFull(board){
   for(var x = 0;x<9;x++){
     if(board[x]==0) return false;
   }
   return true;
}
function chamaIA(){
   iaTurn(estado,0,comp)
}
//versao recursiva do minMax , minimiza score do jogador humano
function iaTurn(board,depth,player){
   if(checkWin(board,!player)) return -10+depth;
   if(checkFull(board)) return 0;
   var value = player == humano ? humanoVal : compVal;
   var max = -Infinity;
   var index;
   for(var x=0;x<9;x++){
     if(board[x]==0){ //can i place here?
        var newBoard = board.slice()
        newBoard[x] = value
```

```
        var moveval = -iaTurn(newBoard,depth+1,!player)
        if(moveval>max){
max = moveval
index = x; }
} }
   if(depth==0) set(index,comp)
return max; }
```

style.css

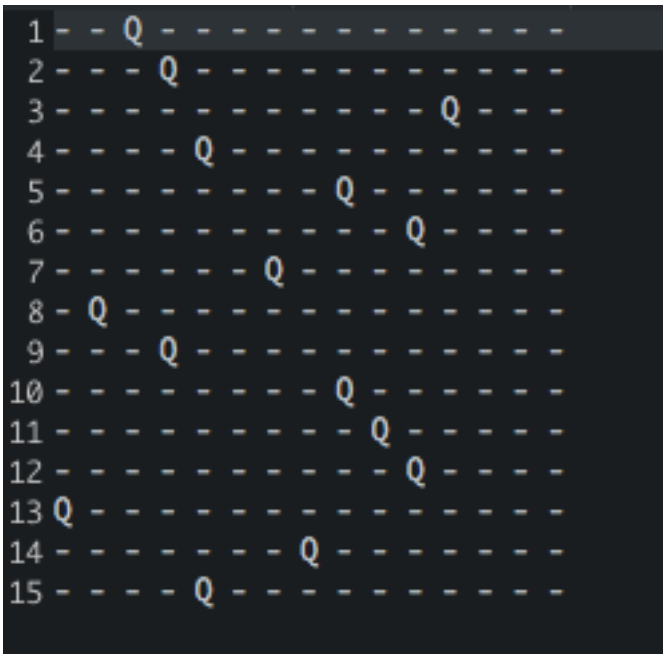
```
html,body{
padding: 0;
margin: 0;
background: #111;
text-transform: uppercase;
color:#111;
font-size: 1.5vh;
}
#main{
   margin:0;
   padding:0;
   padding-top:25vh;
   height:auto;
   width: 100%;
}
#board{
   padding:0;
   margin: auto;
   width: 33vh;
   height: 33vh;
}
.quad{
   padding:0;
   margin:0.5vh;
   height:10vh;
   width:10vh;
   border-radius:5px;
   background: #fff;
   cursor: pointer;
   float: left;
}
.button{
   margin:0.5vh;
   padding:0;
   height: 5vh;
   width: 15.5vh;
   line-height: 5vh;
   vertical-align: middle;
   background: #fff;
   text-align: center;
   border-radius: 5px;
   cursor: pointer;
   float: left;
}
```

3 N queens

Dado que as Redes Neurais são um modelo quantitativo, não é surpreendente que o cálculo apareça. Em particular, o processo de otimização de pesos e vieses é um problema em um espaço contínuo (já que podem ser quaisquer números reais), portanto as derivadas (e especialmente as derivadas parciais) desempenham um papel fundamental na construção de Redes Neurais de bom desempenho. O Gradient Descent , ou Hill clmibing , é um método heurístico que começa em um ponto aleatório e se move iterativamente na direção (daí "gradiente") que diminui (daí "descida") a função que queremos minimizar. Com um número suficiente desses passos na direção decrescente, um mínimo local pode, teoricamente, ser alcançado. Coloquialmente, pense nisso como um jogo de "quente"e "frio", até que a melhora se torne desprezível. O algoritmo de hill climbing recebe o nome dada a metáfora de subir uma colina. Cada iteração é um passo mais alto que o outro.Se ficar preso no máximo local, randomiza o estado.

Abaixo foi implementado uma versão recursiva do HillClimbing . Inicialmente é lido um arquivo .txt ou .csv e as posições das linhas e colunas das rainhas é fornecido à função HillClimbing. Um arquivo .txt ou .csv é gerado como saída , fornecendo uma solução . Para mudar N , basta modificá-lo na linha de cógido define N 50 .

Entrada N = 15:



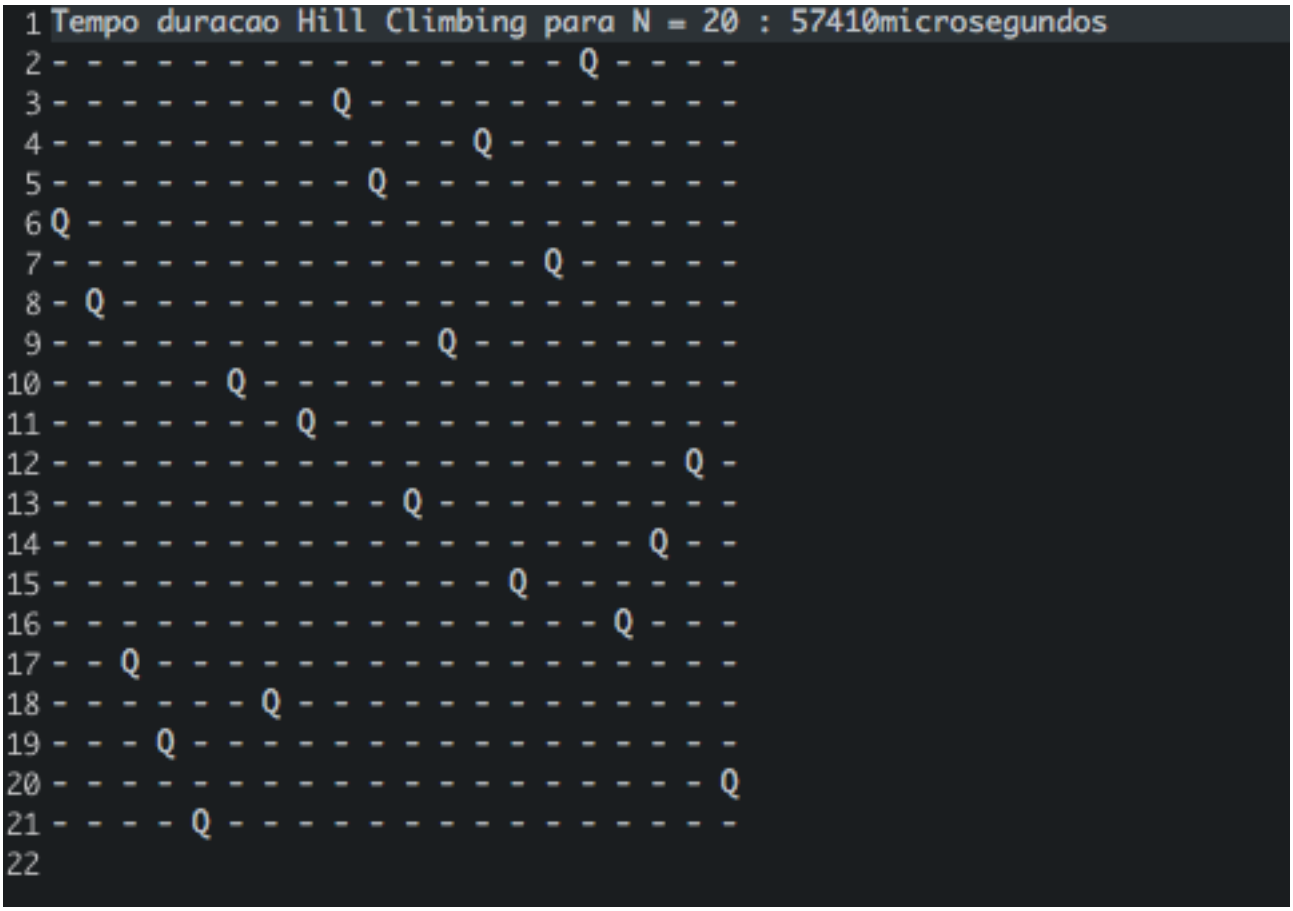
Saída N = 15:

```
Tempo duracao Hill Climbing para N = 15 : 25830microsegundos
- - - - - - - Q - - - - - - -
- - - - Q - - - - - - - - - -
- - - - - - - - - - - Q - - -
- Q - - - - - - - - - - - - -
- - - - - - Q - - - - - - - -
- - - - - - - - - - Q - - - -
- - - - - - - - - - - - Q - -
- - Q - - - - - - - - - - - -
- - - - - - - - - Q - - - - -
- - - Q - - - - - - - - - - -
- - - - - - - - - - - - - - Q
Q - - - - - - - - - - - - - -
- - - - - - - - Q - - - - - -
- - - - - - - - - - - - - Q -
- - - - - Q - - - - - - - - -
```

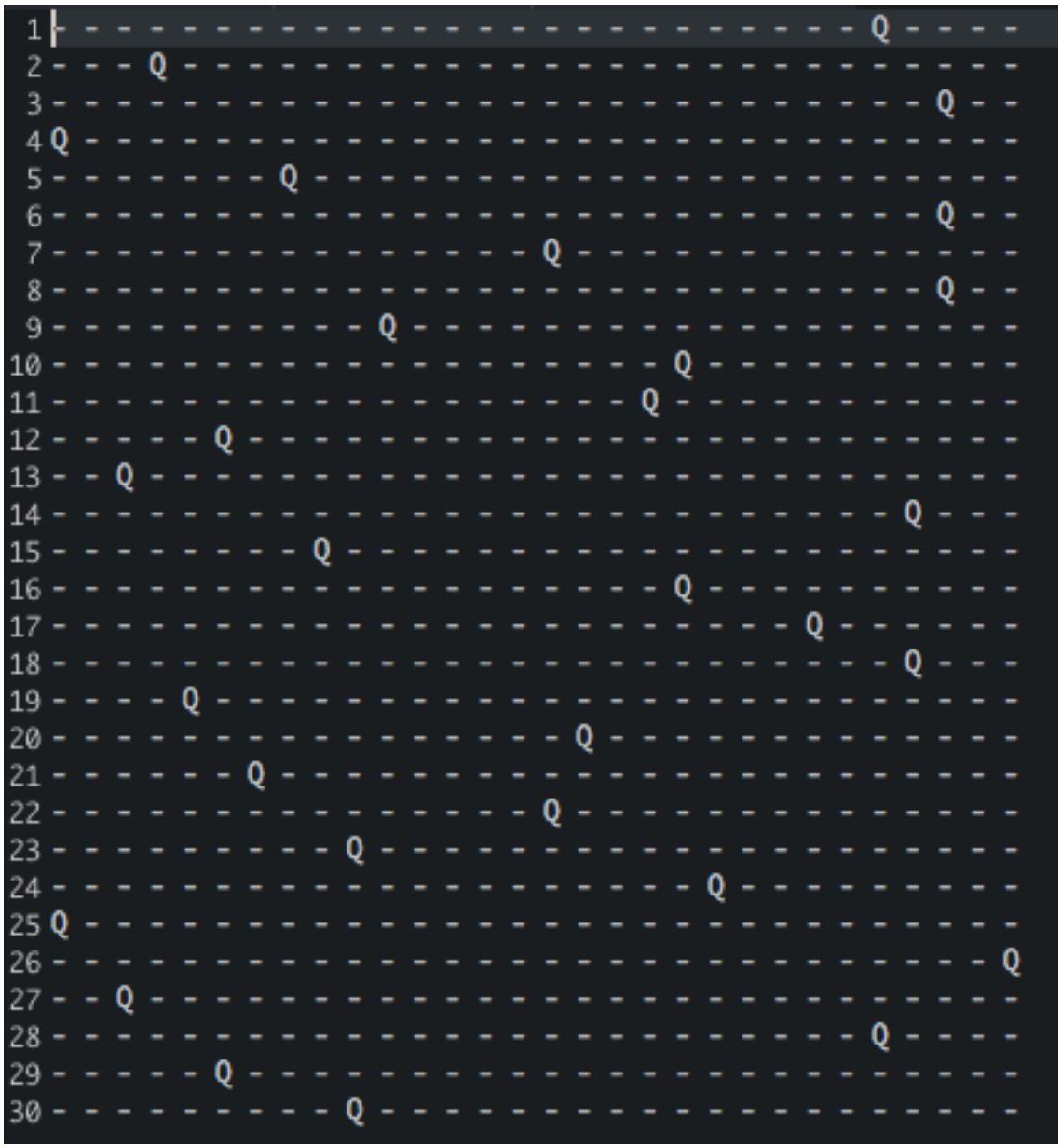
Entrada N = 20:

```
- - - - - - - - - - - - - - - - - - Q - -
- - - - Q - - - - - - - - - - - - - - - -
- - - - Q - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - Q - -
- - - - - - - - - - - - - - - - - - Q - -
- - - - - - - Q - - - - - - - - - - - - -
- Q - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - Q - - - - - - - - - -
- - - - - - - - - - - - - - - - - Q - - -
- - - - - - - - - - Q - - - - - - - - - -
- - - - - - - - - - - - - - - - - Q - - -
- - - - - - - - - - - - - - - - - - Q - -
- - - Q - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - Q
- - - - - - - - - - - - - - - - - - Q - -
- - - - - Q - - - - - - - - - - - - - - -
- - - - - - Q - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - Q - - -
- - - - - - - - - - - - - - - - - - - - Q
- - - - - - - - - - Q - - - - - - - - - -
```

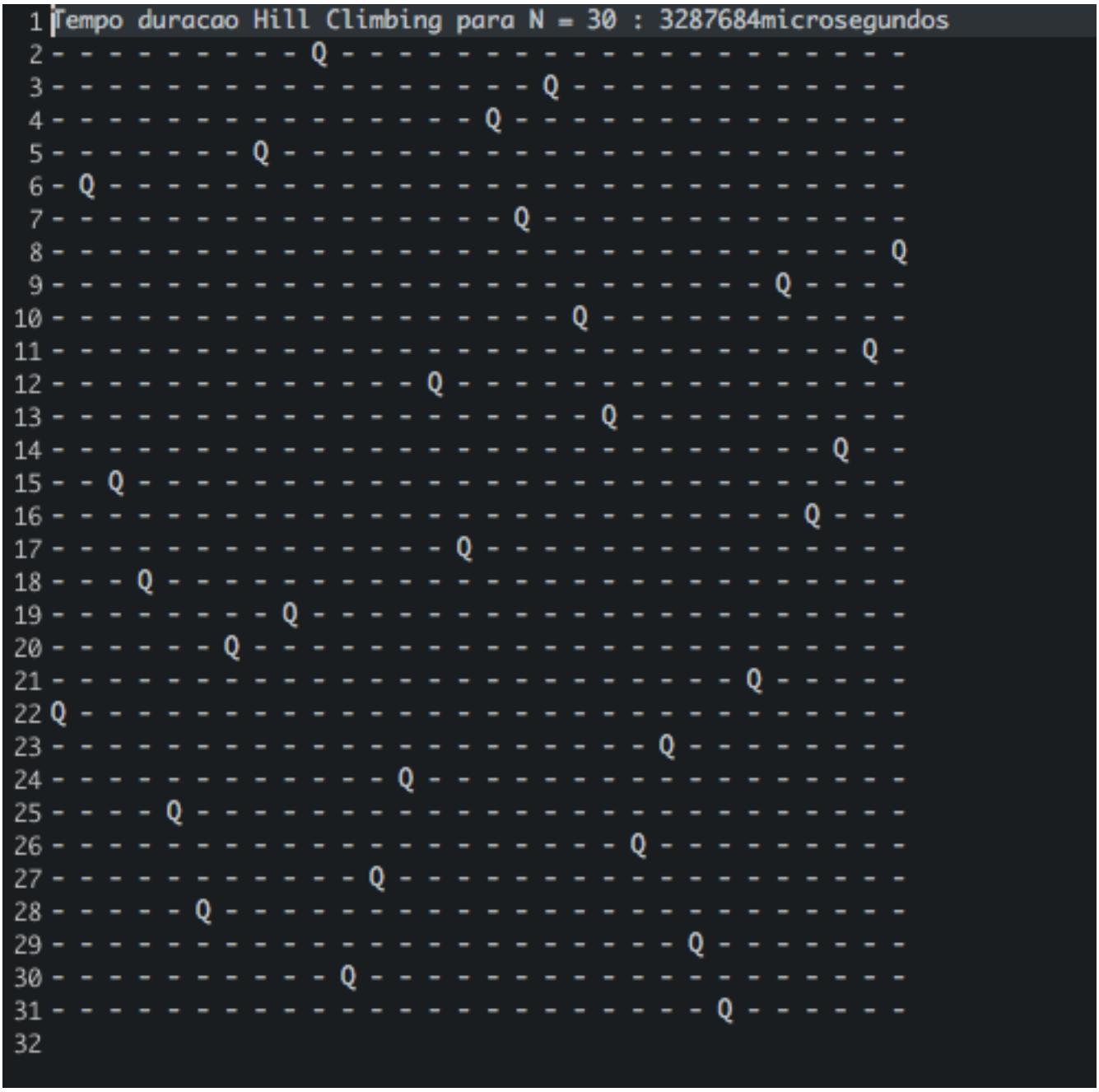
Saída N = 20:



Entrada N = 30:


1 - Q - - - -
2 - - - Q -
3 - Q - -
4 Q -
5 - - - - - - - Q -
6 - Q - -
7 - - - - - - - - - - - - - - - - Q - - - - - - - - - - - - - - -
8 - Q - -
9 - - - - - - - - - - Q -
10 - Q - - - - - - - - - - -
11 - Q - - - - - - - - - -
12 - - - - - Q -
13 - - Q -
14 - Q - -
15 - - - - - - - - Q -
16 - Q - - - - - - - - - -
17 - Q - - - - - -
18 - Q - -
19 - - - - Q -
20 - - - - - - - - - - - - - - - - - - - Q - - - - - - - - - - -
21 - - - - - - Q -
22 - - - - - - - - - - - - - - - - - - - Q - - - - - - - - - - -
23 - - - - - - - - - - Q -
24 - Q - - - - - - - -
25 Q -
26 - Q
27 - - Q -
28 - Q - - -
29 - - - - - Q -
30 - - - - - - - - - - Q -

Saída N = 30:


Tempo duracao Hill Climbing para N = 30 : 3287684microsegundos

Entrada N = 50:

Saída N = 50:

nqueens.cpp

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <map>
#include <vector>
#include <string>
#include <algorithm>
#include <fstream>
#include <set>
#include <stack>
#include <queue>
#include <math.h>
#include <unordered_set>
using namespace std;
using namespace std::chrono;


#define _CRT_SECURE_NO_DEPRECATE
typedef vector<int> vi;
typedef pair<int, int> ii;

typedef vector<ii> vii;
typedef set<int> si;
typedef map<string, int> msi;

high_resolution_clock::time_point t1 ;
high_resolution_clock::time_point t2 ;
#define REP(i, a, b) \
for (int i = int(a); i < int(b); i++)
#define N 50

int Queens[N] = {};
char tab[N][N] ;

int len(int vec[]){
    int i=0;
    while(vec[i] != -1) i++;
    return i;
}

int getRand(int mod){
    return random() % mod;
}

void generateTab(){
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            tab[i][j] = '-';
        }
    }
    for(int i=0;i<N;i++) tab[i][Queens[i]] = 'Q';
}
```

```
int isValid(int vec[], int curlinha, int curcoluna){
    int i;
    if(len(vec)){
        for(i=0;i<len(vec);i++) if(i==curlinha || vec[i]==curcoluna ||
            abs(i-curlinha)==abs(vec[i]-curcoluna)) return 0;
        return 1;
    } else return 1;
    return 0;
}

int getpeso(int vec[]){
    int peso = 0;
    int queen;
    for(queen=0;queen<N;queen++){
        int nextqueen;
        for(nextqueen=queen+1;nextqueen<N;nextqueen++){
            if(vec[queen] == vec[nextqueen] ||
                abs(queen-nextqueen)==abs(vec[queen]-vec[nextqueen])){ //se tem colisao
                peso++;
            }
        }
    }
    return peso;
}

void hillClimbing(int vec[]){
    int peso = getpeso(vec);
    if (peso == 0){
            t2 = high_resolution_clock::now();
            auto duration = duration_cast<microseconds>( t2 - t1 ).count();
                string time = "Tempo duracao Hill Climbing para N = " + to_string(N) +
                    " : " + to_string(duration) + "microsegundos\n";
                string s2 = "solution" + to_string(N) +".txt";
                ofstream sol;
                sol.open(s2);
                sol << time;
                generateTab();

                for (int i = 0;i<N;i++){
                    string s3;
                        for(int j=0;j<N;j++){
                     if (j==N-1) s3 += tab[i][j];
                           else {
                              s3 += tab[i][j];
                              s3 += ' ';
                    }
                              }
                              s3 += '\n';

                              sol << s3;
                             }
                             sol.close();
```

```
            exit(0);
} else {
    int nextlinha[] = {-1};
    int nextcoluna[] = {-1};
    int nextpeso = peso;
    int queen;
    for(queen=0;queen<N;queen++){                       //for each linha
        int origcoluna = vec[queen];                        //save the original
            colunaumn
        int validcoluna;
        for(validcoluna = 0; validcoluna<N;validcoluna++){      //para cada coluna
            valida
            if(validcoluna != origcoluna){
                vec[queen] = validcoluna;                   //coloca rainha na
                    proxima coluna
                int newpeso = getpeso(vec);           //pega o peso do tabuleiro
                if(newpeso < nextpeso){                //se eh um movimento melhor
                    int i;
                    for(i=0;i<len(nextlinha);i++){        //clear tabuleiro
                        nextlinha[i] = (int)NULL;
                        nextcoluna[i] = (int)NULL;
                    }
                    nextlinha[0] = queen;
                    nextlinha[1] = -1;
                    nextcoluna[0] = validcoluna;
                    nextcoluna[1] = -1;
                    nextpeso = newpeso;

                    hillClimbing(vec);
                } else if (newpeso == nextpeso){
                    int leng = len(nextlinha);
                    nextlinha[leng] = queen;
                    nextlinha[leng+1] = -1;
                    nextcoluna[leng] = validcoluna;
                    nextcoluna[leng+1] = -1;
                }

            }
        }
        vec[queen] = origcoluna;
    }
    //quando acabarmos de procurar no tabuleiro
    if (nextcoluna[0] != -1 && nextlinha[0] != -1){    //se encontramos um
        movimento melhor
        int i;
        for(i=0;i<len(nextlinha);i++){            //para cada movimento que de um
            melhor peso
            vec[nextlinha[i]] = nextcoluna[i];        //do it
            hillClimbing(vec);                    //recursao
        }
    } else {

        int i;
        for(i=0;i<N;i++) Queens[i] = getRand(N);
```

```
            hillClimbing(Queens);
        }
    }
}

int main(int argc, const char * argv[]){
    //Leitura arquivo
    string s = "queens" + to_string(N) + ".txt",line;
    ifstream file;
    file.open(s);
    int linha = 0;
    while(getline(file,line)){
        int spaces = 0;
        REP(col,0,line.size()){
            if (line[col] == 'Q'){
                Queens[linha++] = col-spaces;
            }
            if(line[col] == ' ') spaces++;
        }
    }
    file.close();
    // Queens eh um vetor tal que a posicao i representa a linha e o conteudo
    // e Queens[i] representa coluna de uma rainha


    ///comeca a contar tempo
    t1 = high_resolution_clock::now();


    hillClimbing(Queens);
    return 0;
}
```