Project UTS KB

Judul : N-Queen Problem With Backtrack

Kelompok :

* Kevin William Wibisono (c14190081)
* Alexander Dean Pandreou (c14190109)
* Rico Ronaldi (c14190123)
* Leonardo Christofer (c14190125)
* Albertus Farrel Juanda (c14190134)

Isi:algoritma singkat(dinlanjutkan di cpp),kendala yang dialami,contoh output hasil

Goal state:menemukan solusi berupa queen sebanyak n dari n-queen problem

Initial state:sebuah board/papan kosong

Algoritma BFS

|  |
| --- |
| int n = 5;  int arr[100][100];  int queenCounter = 0, col = 0, mode;  //initialize board  initBoard(arr, n);  //animation  mode = 0;  run(arr, n, col, queenCounter, mode);  //display solution  mode = 1;  run(arr, n, col, queenCounter, mode);  if (solutionCounter == 0)  cout << "No solution found for " << n << "x" << n << " size board" << endl;  else  cout << "Solutions found : " << solutionCounter << endl; |

1. Int n berfungsi untuk menentukan ukuran dari n-problem
2. Int arr[100][100] adalah array yang berfungsi untuk menampung board
3. Int queenCounter untuk menentukan jumlah queen yang berada pada board
4. Int col, declare kolom
5. Int mode sebagai pembeda output untuk animasi saat mencari solusi dan output solusi dari solusi N-Queen itu sendiri
6. Run adalah function yang akan menyelesaikan N-Queen Problem
7. SolutionCounter adalah jumlah seluruh solusi yang ditemukan pada nxn board

Penjelasan issafe

|  |
| --- |
| bool isSafe(int row, int col, int arr[100][100], int n) //mengecek collision dengan queen lain  { // 0 -> kosong  //diagonal kanan bawah  // 2 -> ada queen  if (col != n - 1 && row != n - 1)  for (int i = row, j = col; i < n; i++, j++)  if (arr[i + 1][j + 1] == 2)  return false;  //diagonal kiri bawah  if (col != 0 && row != n)  for (int i = row, j = col; i < n; i++, j--)  if (arr[i + 1][j - 1] == 2)  return false;  //diagonal kiri atas  if (row != 0 && col != 0)  for (int i = row, j = col; i > 0; i--, j--)  if (arr[i - 1][j - 1] == 2)  return false;  //diagonal kanan atas  if (row != 0 && col != n - 1)  for (int i = row, j = col; i > 0; i--, j++)  if (arr[i - 1][j + 1] == 2)  return false;  //kanan  if (col != n - 1)  for (int j = col, i = row; j < n; j++)  if (arr[i][j + 1] == 2)  return false;  //kiri  if (col != 0)  for (int j = col, i = row; j > 0; j--)  if (arr[i][j - 1] == 2)  return false;  //atas  if (row != 0)  for (int j = col, i = row; i > 0; i--)  if (arr[i - 1][j] == 2)  return false;  //bawah  if (row != n - 1)  for (int j = col, i = row; i < n; i++)  if (arr[i + 1][j] == 2)  return false;  return true;  } |

1.digunakan untuk mengecek apakah seorang queen menyerang atau tidak

2.menggunakan return,karena diperlukan backtrack bila ternyata tidak memenuhi jumlah n-queen

3.ada pengecekan secara diagonal atas kanan,atas kiri,bawah kanan,bawah kiri,kanan,kiri,atas dan bawah

4.jika ditemukan angka 2 ,artinya daerah tersebut ada queen,sedangkan 0 artinya daerah masih kosong

Penjelasan lebih lanjut tertera pada cpp dan sudah diberi komen ditiap fungsinya.

Kendala

1. Untuk membedakan mana area yang masih kosong, area yang merupakan daerah serang queen.Tidak bisa hanya menggunakan string kosong /- saja,karena perlu dibedakan mana yang benar-benar kosong dan mana yang merupakan daerah serang.

Oleh karena itu, digunakan angka-angka untuk membedakan.

0 = Daerah yang masih kosong

2 = Letak dari queen

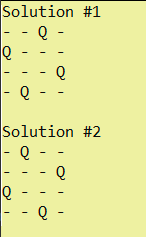
1. Awalnya menggunakan sistem markqueen untuk membedakan tempat yang terkena attack.Namun mempersulit proses backtrack,karena saat remove queen,lokasi dari area serang queen yang seharusnya tidak ikut ter remove,malah ikut ikut ter remove.
2. Awalnya ingin menggunakan nested loop untuk proses peletakan queen. Namun pada akhirnya kita menggunakan rekursi karena jika menggunakan nested loop tidak bisa melakukan backtrack.

Output

N=3



N = 4



N=5

Solution #1

Q - - - -

- - - Q -

- Q - - -

- - - - Q

- - Q - -

Solution #2

Q - - - -

- - Q - -

- - - - Q

- Q - - -

- - - Q -

Solution #3

- - Q - -

Q - - - -

- - - Q -

- Q - - -

- - - - Q

Solution #4

- - - Q -

Q - - - -

- - Q - -

- - - - Q

- Q - - -

Solution #5

- Q - - -

- - - Q -

Q - - - -

- - Q - -

- - - - Q

Solution #6

- - - - Q

- - Q - -

Q - - - -

- - - Q -

- Q - - -

Solution #7

- Q - - -

- - - - Q

- - Q - -

Q - - - -

- - - Q -

Solution #8

- - - - Q

- Q - - -

- - - Q -

Q - - - -

- - Q - -

Solution #9

- - - Q -

- Q - - -

- - - - Q

- - Q - -

Q - - - -

Solution #10

- - Q - -

- - - - Q

- Q - - -

- - - Q -

Q - - - -

Solutions found : 10