More Applications on Queues

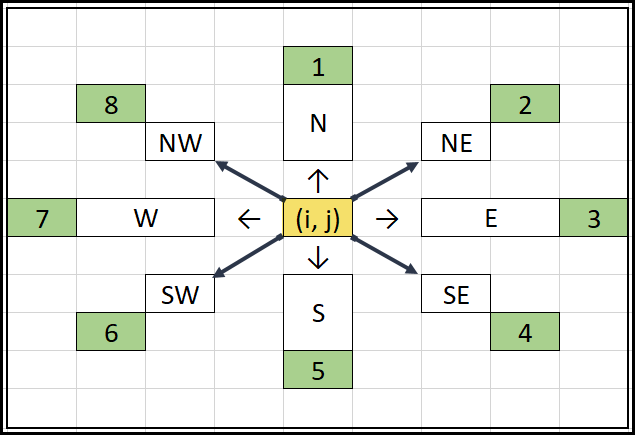
[PROBLEM] Given a matrix and a ‘start’ cell. Fill the matrix starting from ‘start’ cell with initial value 0. Fill its neighbors by incrementing the value.

Sudo code:

|  |
| --- |
| int x8[] = { -1,-1,0,1,1,1,0,-1 };  int y8[] = { 0,1,1,1,0,-1,-1,-1 };  Cell  -int i;  -int j;  fillMatrix(*list*< *list*<int> > m, Cell pos) {  Queue<Cell> q;  *start* = 0;  q.*push*(pos);  m[pos.i][pos.j] = *start*++;  while (!q.*empty*()) {  pos = q.*front*(); q.pop(); // Remove front of the queue  for each VALID and UNVISITED neighbor *npos*  q.*push*(*npos*);  m[*npos*.i][*npos*.j] = *start*++;  MARK *npos* as VISITED  }  } |

## Breadth First Search

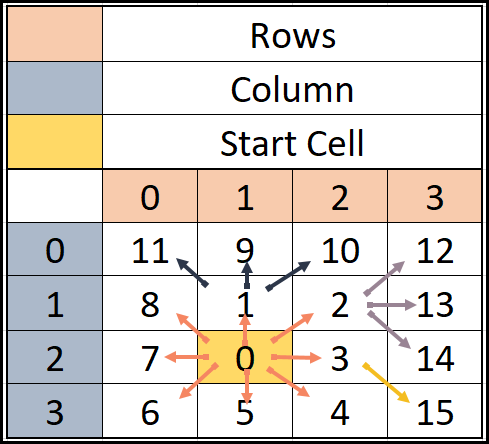
* Given a Cell (i, j) in a matrix. How to reach all its neighbors?

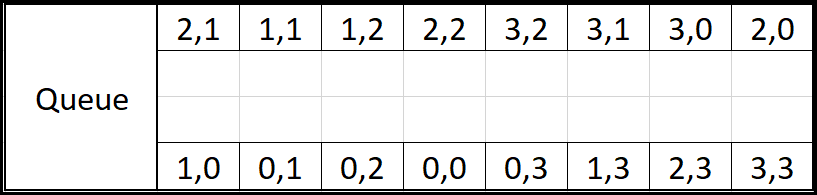


* Prepare an array…

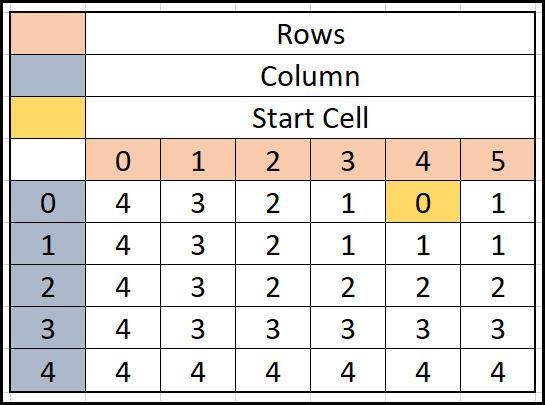
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | N | NE | E | SE | S | SW | W | NW |
| i | -1 | -1 | +0 | +1 | +1 | +1 | +0 | -1 |
| j | +0 | +1 | +1 | +1 | +0 | -1 | -1 | -1 |

* Start filling the matrix. Use queue for the purpose….





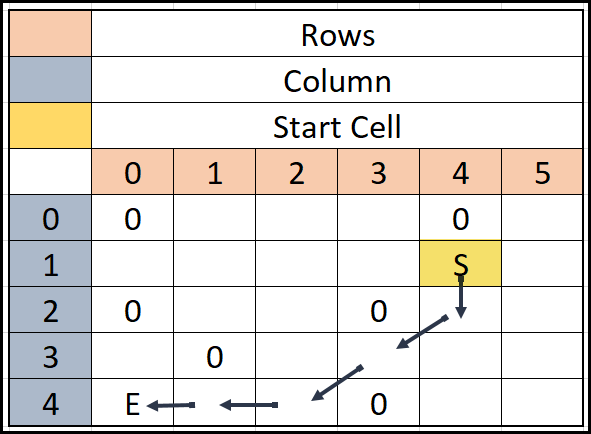
[PROBLEM] Given a ‘start’ cell in a matrix and a value. Fill its neighboring cells with value++.



* Using the same approach we can fill the matrix.

|  |
| --- |
| *vector*<int> x8 = { -1,-1,0,1,1,1,0,-1 };  *vector*<int> y8 = { 0,1,1,1,0,-1,-1,-1 };  void visitCell(*vector*<*vector*<int>>& mat, *pair*<int,int> pos, int value) {  mat[pos.*first*][pos.second] = value;  }  *vector*<*pair*<int,int>> getNextPositions(*pair*<int,int> pos) {  *vector*<*pair*<int,int>> neighbors;  for (int k = 0; k < 8; k++) {  int a = pos.*first* + x8[k];  int b = pos.second + y8[k];  neighbors.*push\_back*({a,b});  }  return neighbors;  }  bool isUnvisited(*vector*<*vector*<int>>& mat, *pair*<int,int> pos) {  return mat[pos.*first*][pos.second] == -1;  }  bool isInvalidPosition(*pair*<int,int> pos, int N, int M) {  return !(pos.*first* >= 0 && pos.*first* < N && pos.second >= 0 && pos.second < M);  }  void bfs(*vector*<*vector*<int>>& mat, *pair*<int,int> startPos)  {  int rows = mat.*size*();  int cols = mat[0].*size*();  *queue*<*pair*<int,int>> q;  q.*push*(startPos);  *set*<*pair*<int,int>> st;  st.insert(startPos);  int value = 0;  visitCell(mat, startPos, value);  while (!q.*empty*())  {  auto pos = q.*front*(); q.*pop*();  *vector*<*pair*<int,int>> neighbors = getNextPositions(pos);  for (auto nb : neighbors) {  if (!isInvalidPosition(nb, rows, cols) && isUnvisited(mat, nb)) {  st.insert(nb);  visitCell(mat, nb, mat[pos.*first*][pos.second]+1);  q.*push*(nb);  }  }  }  }  int main(void) {  *vector*<*vector*<int>> mat(5, *vector*<int>(6, -1));  *cout* << "Initial Matrix: \n";  for (auto i : mat) {  for (auto j : i) {  *cout* << j << " ";  }  *cout* << *endl*;  }  bfs(mat, { 0,4 });  *cout* << "Matrix after filling: \n";  for (auto i : mat) {  for (auto j : i) {  *cout* << j << " ";  }  *cout* << *endl*;  }  return 0;  }  /\*  Initial Matrix:  -1 -1 -1 -1 -1 -1  -1 -1 -1 -1 -1 -1  -1 -1 -1 -1 -1 -1  -1 -1 -1 -1 -1 -1  -1 -1 -1 -1 -1 -1  Matrix after filling:  4 3 2 1 0 1  4 3 2 1 1 1  4 3 2 2 2 2  4 3 3 3 3 3  4 4 4 4 4 4  \*/ |

[PROBLEM] Given a matrix having cells 0s and 1s. 0 means WALL, 1 means PATH. Given (si, sj) - Start Cell and (ei, ej) End Cell. Find the shortest path length from START cell to End Cell. Return -1 if no path exists.



[PROBLEM] Given a matrix having cells -1s and 1s. -1 means WALL, 1 means PATH. Given (si, 3]) - Start Cell and (ei, ej) End Cell. Find the shortest path length from START cell to End Cell. Return -1 if no path exists.