|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| package bottricks;  import static java.lang.reflect.Array.set;  import java.util.\*;  class Pair  {      int first;      int second;      public static Pair make\_pair(int f,int s)      {          return new Pair(f,s);      }        public Pair(int t,int u)      {          first=t;          second=u;      }  }  class pPair  {      double first;      Pair second;        public static pPair make\_pair(double a,Pair b)      {          return new pPair(a,b);      }        public pPair(double a,Pair b)      {          first=a;          second=b;      }        public void display()      {          System.out.println("fValue: "+first+"--> "+second.first+","+second.second);      }  }  class cell  {      int parent\_i,parent\_j;      double f,g,h;  }  public class BotTricks  {      static List<String> sendAllStrinsg=new LinkedList<>();      public BotTricks() {      }      static String finStr;      static int ROW=5,COL=6;      static int finalPath[][]=new int[ROW][COL];        public static boolean isValid(int row,int col)      {          return (row >= 0) && (row < ROW) && (col >= 0) && (col < COL);      }        public static boolean isUnBlocked(int grid[][], int row, int col)      {              // Returns true if the cell is not blocked else false              if (grid[row][col] == 1)                  return (true);              else                  return (false);      }        public static boolean isDestination(int row, int col, Pair dest)      {              if (row == (int)dest.first && col == (int)dest.second)                      return (true);              else                      return (false);      }        public static double calculateHValue(int row, int col, Pair dest)      {              // Return using the distance formula              return ((double)Math.sqrt(Math.pow((row-(int)dest.first),2)+ Math.pow((col-(int)dest.second),2)));      }        public static int tracePath(cell cellDetails[][],Pair dest,boolean isDestDecided)      {          int row = (int)dest.first;  int col = (int)dest.second;          int totalFValue=0;            Stack<Pair> path=new Stack<>();          while (!(cellDetails[row][col].parent\_i == row && cellDetails[row][col].parent\_j == col ))  {  path.push (Pair.make\_pair(row, col));  int temp\_row = cellDetails[row][col].parent\_i;  int temp\_col = cellDetails[row][col].parent\_j;  row = temp\_row;  col = temp\_col;  totalFValue++;  }            if(isDestDecided==true)          {              int lx=row,ly=col,nx,ny;              System.out.println("\nThe Path is ");              finStr="";              path.push (Pair.make\_pair (row, col));              while (!path.empty())              {                  Pair p = path.peek();                  path.pop();                  nx=p.first;                  ny=p.second;                    if(nx!=lx || ny!=ly)                  {                      if(nx-lx==1 && ny-ly==1)                          finStr+="M";                      else if(nx-lx==1 && ny==ly)                          finStr+="S";                      else if(nx-lx==1 && ny-ly==-1)                          finStr+="Z";                      else if(nx==lx && ny-ly==-1)                          finStr+="W";                      else if(nx-lx==-1 && ny-ly==-1)                          finStr+="Q";                      else if(nx-lx==-1 && ny==ly)                          finStr+="N";                      else if(nx-lx==-1 && ny-ly==1)                          finStr+="P";                      else if(nx-lx==0 && ny-ly==1)                          finStr+="E";                        lx=nx;  ly=ny;                  }                    System.out.print("-> "+p.first+","+p.second);                  finalPath[(int)p.first][(int)p.second]=1;              }              System.out.println("\nString is: "+finStr);  //-------------MOST IMP STRING IN CODE THAT WE NEED TO PASS TO ARDUINO-----------------              sendAllStrinsg.add(finStr);          }          return totalFValue;      }            public static int aStarSearch(int grid[][], Pair src, Pair dest,boolean isDestDecided)      {          int totalFValue=0;              // If the source is out of range              if (isValid ((int)src.first, (int)src.second) == false)              {                      System.out.print("Source is invalid\n");                      return 0;              }              // If the destination is out of range              if (isValid ((int)dest.first, (int)dest.second) == false)              {                      System.out.printf ("Destination is invalid\n");                      return 0;              }              // Either the source or the destination is blocked                if (isUnBlocked(grid, src.first, src.second) == false ||                              isUnBlocked(grid, dest.first, dest.second) == false)              {                      System.out.println("Source or the destination is blocked\n");                      return 0;              }                // If the destination cell is the same as source cell              if (isDestination((int)src.first, (int)src.second, dest) == true)              {                      System.out.printf ("We are already at the destination\n");                      return 0;              }              // Create a closed list and initialise it to false which means              // that no cell has been included yet              // This closed list is implemented as a boolean 2D array              boolean closedList[][]=new boolean[ROW][COL];                for (boolean[] row: closedList)                  Arrays.fill(row, false);                // Declare a 2D array of structure to hold the details              //of that cell              cell cellDetails[][]=new cell[ROW][COL];              int i, j;              for (i=0; i<ROW; i++)              {                      cellDetails[i]=new cell[COL];                      for (j=0; j<COL; j++)                      {                          cellDetails[i][j]=new cell();                          cellDetails[i][j].f = Integer.MIN\_VALUE;                          cellDetails[i][j].g = Integer.MIN\_VALUE;                          cellDetails[i][j].h = Integer.MIN\_VALUE;                          cellDetails[i][j].parent\_i = -1;                          cellDetails[i][j].parent\_j = -1;                      }              }              // Initialising the parameters of the starting node              i = (int)src.first;              j = (int)src.second;              cellDetails[i][j].f = 0.0;              cellDetails[i][j].g = 0.0;              cellDetails[i][j].h = 0.0;              cellDetails[i][j].parent\_i = i;              cellDetails[i][j].parent\_j = j;              /\*              Create an open list having information as-              <f, <i, j>>              where f = g + h,              and i, j are the row and column index of that cell              Note that 0 <= i <= ROW-1 & 0 <= j <= COL-1              This open list is implenented as a set of pair of pair.\*/      //            Set<Map> openList=new HashSet();                List openList=new LinkedList();    //            set<pPair> openList;              // Put the starting cell on the open list and set its              // 'f' as 0              openList.add(pPair.make\_pair (0.0, Pair.make\_pair (i, j)));              // We set this boolean value as false as initially              // the destination is not reached.              boolean foundDest = false;              while (!openList.isEmpty())              {                      pPair p = (pPair)openList.get(0);                        openList.remove(0);                      // Add this vertex to the closed list                      i = p.second.first;                      j = p.second.second;                      closedList[i][j] = true;              /\*                      Generating all the 8 successor of this cell                              N.W N N.E                              \ | /                              \ | /                              W----Cell----E                                      / | \                              / | \                              S.W S S.E                      Cell-->Popped Cell (i, j)                      N --> North (i-1, j)                      S --> South (i+1, j)                      E --> East (i, j+1)                      W --> West (i, j-1)                      N.E--> North-East (i-1, j+1)                      N.W--> North-West (i-1, j-1)                      S.E--> South-East (i+1, j+1)                      S.W--> South-West (i+1, j-1)\*/                      // To store the 'g', 'h' and 'f' of the 8 successors                      double gNew, hNew, fNew;                      //----------- 1st Successor (North) ------------                      // Only process this cell if this is a valid one                      if (isValid(i-1, j) == true)                      {                              // If the destination cell is the same as the                              // current successor                              if (isDestination(i-1, j, dest) == true)                              {                                      // Set the Parent of the destination cell                                      cellDetails[i-1][j].parent\_i = i;                                      cellDetails[i-1][j].parent\_j = j;                      // printf ("The destination cell is found\n");                                      totalFValue=tracePath (cellDetails, dest,isDestDecided);                                      foundDest = true;                                      return totalFValue;                              }                              // If the successor is already on the closed                              // list or if it is blocked, then ignore it.                              // Else do the following                              else if (closedList[i-1][j] == false &&                                              isUnBlocked(grid, i-1, j) == true)                              {                                      gNew = cellDetails[i][j].g + 1.0;                                      hNew = calculateHValue (i-1, j, dest);                                      fNew = gNew + hNew;                                        if (cellDetails[i-1][j].f == Integer.MIN\_VALUE ||cellDetails[i-1][j].f > fNew)                                      {                                              openList.add( pPair.make\_pair(fNew,Pair.make\_pair(i-1, j)));                                              // Update the details of this cell                                              cellDetails[i-1][j].f = fNew;                                              cellDetails[i-1][j].g = gNew;                                              cellDetails[i-1][j].h = hNew;                                              cellDetails[i-1][j].parent\_i = i;                                              cellDetails[i-1][j].parent\_j = j;                                      }                              }                      }                      //----------- 2nd Successor (South) ------------                      // Only process this cell if this is a valid one                      if (isValid(i+1, j) == true)                      {                              // If the destination cell is the same as the                              // current successor                              if (isDestination(i+1, j, dest) == true)                              {                                      // Set the Parent of the destination cell                                      cellDetails[i+1][j].parent\_i = i;                                      cellDetails[i+1][j].parent\_j = j;                      // printf("The destination cell is found\n");                                      totalFValue= tracePath(cellDetails, dest,isDestDecided);                                      foundDest = true;                                      return totalFValue;                              }                              // If the successor is already on the closed                              // list or if it is blocked, then ignore it.                              // Else do the following                              else if (closedList[i+1][j] == false &&                                              isUnBlocked(grid, i+1, j) == true)                              {                                      gNew = cellDetails[i][j].g + 1.0;                                      hNew = calculateHValue(i+1, j, dest);                                      fNew = gNew + hNew;                                      // If it isn’t on the open list, add it to                                      // the open list. Make the current square                                      // the parent of this square. Record the                                      // f, g, and h costs of the square cell                                      // OR                                      // If it is on the open list already, check                                      // to see if this path to that square is better,                                      // using 'f' cost as the measure.                                      if (cellDetails[i+1][j].f == Integer.MIN\_VALUE ||                                                      cellDetails[i+1][j].f > fNew)                                      {                                              openList.add( pPair.make\_pair (fNew, Pair.make\_pair (i+1, j)));                                              // Update the details of this cell                                              cellDetails[i+1][j].f = fNew;                                              cellDetails[i+1][j].g = gNew;                                              cellDetails[i+1][j].h = hNew;                                              cellDetails[i+1][j].parent\_i = i;                                              cellDetails[i+1][j].parent\_j = j;                                      }                              }                      }                      //----------- 3rd Successor (East) ------------                      // Only process this cell if this is a valid one                      if (isValid (i, j+1) == true)                      {                              // If the destination cell is the same as the                              // current successor                              if (isDestination(i, j+1, dest) == true)                              {                                      // Set the Parent of the destination cell                                      cellDetails[i][j+1].parent\_i = i;                                      cellDetails[i][j+1].parent\_j = j;                      // printf("The destination cell is found\n");                                      totalFValue=tracePath(cellDetails, dest,isDestDecided);                                      foundDest = true;                                      return totalFValue;                              }                              // If the successor is already on the closed                              // list or if it is blocked, then ignore it.                              // Else do the following                              else if (closedList[i][j+1] == false &&                                              isUnBlocked (grid, i, j+1) == true)                              {                                      gNew = cellDetails[i][j].g + 1.0;                                      hNew = calculateHValue (i, j+1, dest);                                      fNew = gNew + hNew;                                      // If it isn’t on the open list, add it to                                      // the open list. Make the current square                                      // the parent of this square. Record the                                      // f, g, and h costs of the square cell                                      // OR                                      // If it is on the open list already, check                                      // to see if this path to that square is better,                                      // using 'f' cost as the measure.                                      if (cellDetails[i][j+1].f == Integer.MIN\_VALUE ||                                                      cellDetails[i][j+1].f > fNew)                                      {                                              openList.add( pPair.make\_pair(fNew,Pair.make\_pair (i, j+1)));                                              // Update the details of this cell                                              cellDetails[i][j+1].f = fNew;                                              cellDetails[i][j+1].g = gNew;                                              cellDetails[i][j+1].h = hNew;                                              cellDetails[i][j+1].parent\_i = i;                                              cellDetails[i][j+1].parent\_j = j;                                      }                              }                      }                      //----------- 4th Successor (West) ------------                      // Only process this cell if this is a valid one                      if (isValid(i, j-1) == true)                      {                              // If the destination cell is the same as the                              // current successor                              if (isDestination(i, j-1, dest) == true)                              {                                      // Set the Parent of the destination cell                                      cellDetails[i][j-1].parent\_i = i;                                      cellDetails[i][j-1].parent\_j = j;                      // printf("The destination cell is found\n");                                      totalFValue=tracePath(cellDetails, dest,isDestDecided);                                      foundDest = true;                                      return totalFValue;                              }                              // If the successor is already on the closed                              // list or if it is blocked, then ignore it.                              // Else do the following                              else if (closedList[i][j-1] == false &&                                              isUnBlocked(grid, i, j-1) == true)                              {                                      gNew = cellDetails[i][j].g + 1.0;                                      hNew = calculateHValue(i, j-1, dest);                                      fNew = gNew + hNew;                                      // If it isn’t on the open list, add it to                                      // the open list. Make the current square                                      // the parent of this square. Record the                                      // f, g, and h costs of the square cell                                      // OR                                      // If it is on the open list already, check                                      // to see if this path to that square is better,                                      // using 'f' cost as the measure.                                      if (cellDetails[i][j-1].f == Integer.MIN\_VALUE ||                                                      cellDetails[i][j-1].f > fNew)                                      {                                              openList.add( pPair.make\_pair (fNew,Pair.make\_pair (i, j-1)));                                              // Update the details of this cell                                              cellDetails[i][j-1].f = fNew;                                              cellDetails[i][j-1].g = gNew;                                              cellDetails[i][j-1].h = hNew;                                              cellDetails[i][j-1].parent\_i = i;                                              cellDetails[i][j-1].parent\_j = j;                                      }                              }                      }                      //----------- 5th Successor (North-East) ------------                      // Only process this cell if this is a valid one                      if (isValid(i-1, j+1) == true)                      {                              // If the destination cell is the same as the                              // current successor                              if (isDestination(i-1, j+1, dest) == true)                              {                                      // Set the Parent of the destination cell                                      cellDetails[i-1][j+1].parent\_i = i;                                      cellDetails[i-1][j+1].parent\_j = j;                      ///// printf ("The destination cell is found\n");                                      totalFValue=tracePath (cellDetails, dest,isDestDecided);                                      foundDest = true;                                      return totalFValue;                              }                              // If the successor is already on the closed                              // list or if it is blocked, then ignore it.                              // Else do the following                              else if (closedList[i-1][j+1] == false &&                                              isUnBlocked(grid, i-1, j+1) == true)                              {                                      gNew = cellDetails[i][j].g + 1.414;                                      hNew = calculateHValue(i-1, j+1, dest);                                      fNew = gNew + hNew;                                      // If it isn’t on the open list, add it to                                      // the open list. Make the current square                                      // the parent of this square. Record the                                      // f, g, and h costs of the square cell                                      // OR                                      // If it is on the open list already, check                                      // to see if this path to that square is better,                                      // using 'f' cost as the measure.                                      if (cellDetails[i-1][j+1].f == Integer.MIN\_VALUE ||cellDetails[i-1][j+1].f > fNew)                                      {                                              openList.add( pPair.make\_pair (fNew,Pair.make\_pair(i-1, j+1)));                                              // Update the details of this cell                                              cellDetails[i-1][j+1].f = fNew;                                              cellDetails[i-1][j+1].g = gNew;                                              cellDetails[i-1][j+1].h = hNew;                                              cellDetails[i-1][j+1].parent\_i = i;                                              cellDetails[i-1][j+1].parent\_j = j;                                      }                              }                      }                      //----------- 6th Successor (North-West) ------------                      // Only process this cell if this is a valid one                      if (isValid (i-1, j-1) == true)                      {                              // If the destination cell is the same as the                              // current successor                              if (isDestination (i-1, j-1, dest) == true)                              {                                      // Set the Parent of the destination cell                                      cellDetails[i-1][j-1].parent\_i = i;                                      cellDetails[i-1][j-1].parent\_j = j;                      // printf ("The destination cell is found\n");                                      totalFValue=tracePath (cellDetails, dest,isDestDecided);                                      foundDest = true;                                      return totalFValue;                              }                              // If the successor is already on the closed                              // list or if it is blocked, then ignore it.                              // Else do the following                              else if (closedList[i-1][j-1] == false && isUnBlocked(grid, i-1, j-1) == true)                              {                                      gNew = cellDetails[i][j].g + 1.414;                                      hNew = calculateHValue(i-1, j-1, dest);                                      fNew = gNew + hNew;                                      // If it isn’t on the open list, add it to                                      // the open list. Make the current square                                      // the parent of this square. Record the                                      // f, g, and h costs of the square cell                                      // OR                                      // If it is on the open list already, check                                      // to see if this path to that square is better,                                      // using 'f' cost as the measure.                                      if (cellDetails[i-1][j-1].f == Integer.MIN\_VALUE ||cellDetails[i-1][j-1].f > fNew)                                      {                                              openList.add( pPair.make\_pair (fNew, Pair.make\_pair (i-1, j-1)));                                              // Update the details of this cell                                              cellDetails[i-1][j-1].f = fNew;                                              cellDetails[i-1][j-1].g = gNew;                                              cellDetails[i-1][j-1].h = hNew;                                              cellDetails[i-1][j-1].parent\_i = i;                                              cellDetails[i-1][j-1].parent\_j = j;                                      }                              }                      }                      //----------- 7th Successor (South-East) ------------                      // Only process this cell if this is a valid one                      if (isValid(i+1, j+1) == true)                      {                              // If the destination cell is the same as the                              // current successor                              if (isDestination(i+1, j+1, dest) == true)                              {                                      // Set the Parent of the destination cell                                      cellDetails[i+1][j+1].parent\_i = i;                                      cellDetails[i+1][j+1].parent\_j = j;                      // printf ("The destination cell is found\n");                                      totalFValue=tracePath (cellDetails, dest,isDestDecided);                                      foundDest = true;                                      return totalFValue;                              }                              // If the successor is already on the closed                              // list or if it is blocked, then ignore it.                              // Else do the following                              else if (closedList[i+1][j+1] == false &&                                              isUnBlocked(grid, i+1, j+1) == true)                              {                                      gNew = cellDetails[i][j].g + 1.414;                                      hNew = calculateHValue(i+1, j+1, dest);                                      fNew = gNew + hNew;                                      // If it isn’t on the open list, add it to                                      // the open list. Make the current square                                      // the parent of this square. Record the                                      // f, g, and h costs of the square cell                                      // OR                                      // If it is on the open list already, check                                      // to see if this path to that square is better,                                      // using 'f' cost as the measure.                                      if (cellDetails[i+1][j+1].f == Integer.MIN\_VALUE ||cellDetails[i+1][j+1].f > fNew)                                      {                                              openList.add(pPair.make\_pair(fNew,Pair. make\_pair (i+1, j+1)));                                              // Update the details of this cell                                              cellDetails[i+1][j+1].f = fNew;                                              cellDetails[i+1][j+1].g = gNew;                                              cellDetails[i+1][j+1].h = hNew;                                              cellDetails[i+1][j+1].parent\_i = i;                                              cellDetails[i+1][j+1].parent\_j = j;                                      }                              }                      }                      //----------- 8th Successor (South-West) ------------                      // Only process this cell if this is a valid one                      if (isValid (i+1, j-1) == true)                      {                              // If the destination cell is the same as the                              // current successor                              if (isDestination(i+1, j-1, dest) == true)                              {                                      // Set the Parent of the destination cell                                      cellDetails[i+1][j-1].parent\_i = i;                                      cellDetails[i+1][j-1].parent\_j = j;                      // printf("The destination cell is found\n");                                      totalFValue=tracePath(cellDetails, dest,isDestDecided);                                      foundDest = true;                                      return totalFValue;                              }                              // If the successor is already on the closed                              // list or if it is blocked, then ignore it.                              // Else do the following                              else if (closedList[i+1][j-1] == false &&                                              isUnBlocked(grid, i+1, j-1) == true)                              {                                      gNew = cellDetails[i][j].g + 1.414;                                      hNew = calculateHValue(i+1, j-1, dest);                                      fNew = gNew + hNew;                                      // If it isn’t on the open list, add it to                                      // the open list. Make the current square                                      // the parent of this square. Record the                                      // f, g, and h costs of the square cell                                      // OR                                      // If it is on the open list already, check                                      // to see if this path to that square is better,                                      // using 'f' cost as the measure.                                      if (cellDetails[i+1][j-1].f == Integer.MIN\_VALUE ||cellDetails[i+1][j-1].f > fNew)                                      {                                              openList.add(pPair.make\_pair(fNew,Pair.make\_pair(i+1, j-1)));                                              // Update the details of this cell                                              cellDetails[i+1][j-1].f = fNew;                                              cellDetails[i+1][j-1].g = gNew;                                              cellDetails[i+1][j-1].h = hNew;                                              cellDetails[i+1][j-1].parent\_i = i;                                              cellDetails[i+1][j-1].parent\_j = j;                                      }                              }                      }              }              return totalFValue;      }          public static void main(String[] args)      {          int grid[][] =  {  { 1, 1, 1, 1, 1 ,1},  { 1, 0, 1, 1, 1 ,1},  { 1, 1, 0, 1, 1 ,1},  { 0, 1, 1, 0, 0 ,1},  { 1, 1, 1, 1, 0 ,1}  };            Pair src = new Pair(0, 0);  //        Pair dest = new Pair(4, 5);          Pair dest;            List<pPair> l=new LinkedList<>();    //        Vector<pPair> vect=new Vector<>();          int destIndex[][]={                          { 2,1},                          { 3,2},                          { 1,5}                      };            int numberofrows = destIndex.length;          int numberofcols= 2;            int d1,d2,fValue;            for(int i=0;i<numberofrows;i++)          {              d1=destIndex[i][0];              d2=destIndex[i][1];              l.add(pPair.make\_pair(0, Pair.make\_pair(d1,d2 )));          }            d1=0;   d2=0;            for(int i=0;i<numberofrows;i++)          {              for (int[] row: finalPath)                  Arrays.fill(row, 0);              src=new Pair(d1,d2);              for(int j=0;j<l.size();j++)              {                  d1=l.get(j).second.first;                  d2=l.get(j).second.second;                  dest=new Pair(d1,d2);                  fValue=aStarSearch(grid, src, dest,false);  //                System.out.println(fValue+"of "+src.first+","+src.second+" to "+dest.first+","+dest.second);                  l.set(j, new pPair(fValue, new Pair(d1,d2 )));              }              Collections.sort(l, new SortpPair());  //            System.out.println("Least fvalue is "+l.get(0).first);                  d1=l.get(0).second.first;              d2=l.get(0).second.second;              dest=Pair.make\_pair(d1,d2);              aStarSearch(grid, src, dest,true);              l.remove(0);              printPath(src,dest);          }  //        System.out.println("final d1,d2 are: "+d1+","+d2);          System.out.println("Back to counter: ");          for (int[] row: finalPath)              Arrays.fill(row, 0);          src=new Pair(d1,d2);          dest=new Pair(0,0);          aStarSearch(grid, src, dest, true);          printPath(src, dest);            System.out.println("========================================================================");          for(int i=0;i<sendAllStrinsg.size();i++)              System.out.println(sendAllStrinsg.get(i));      }        public static void printPath(Pair src,Pair dest)      {          System.out.println("\n----------------------------------------------------");          System.out.println("Path from source: "+src.first+","+src.second+" to "+dest.first+","+dest.second);          for(int i=0;i<ROW;i++)          {              for(int j=0;j<COL;j++)              {                  System.out.print(finalPath[i][j]+" ");              }              System.out.println("");          }      }  }  class SortpPair implements Comparator<pPair>  {      @Override      public int compare(pPair a, pPair b)      {          return (int)a.first-(int)b.first;      }  } |