Homework 3B

2021315385
이건 / Gun Daniel Lee

Problem explanation

- Find a path that minimizes the sum of weights of all points that the path passes
- Each path must only have at most 1 zero
- Path may not be the shortest path

Problem explanation

- Input:

1. Starting and destination point indices
2. 10-by-10 matrix; weight being an integer between -100 and 100, or -9999 (-9999 means not included)

- Output:

1. The sum of minimum weight
2. The series of point indices from starting to destination point including themselves

Solution explanation

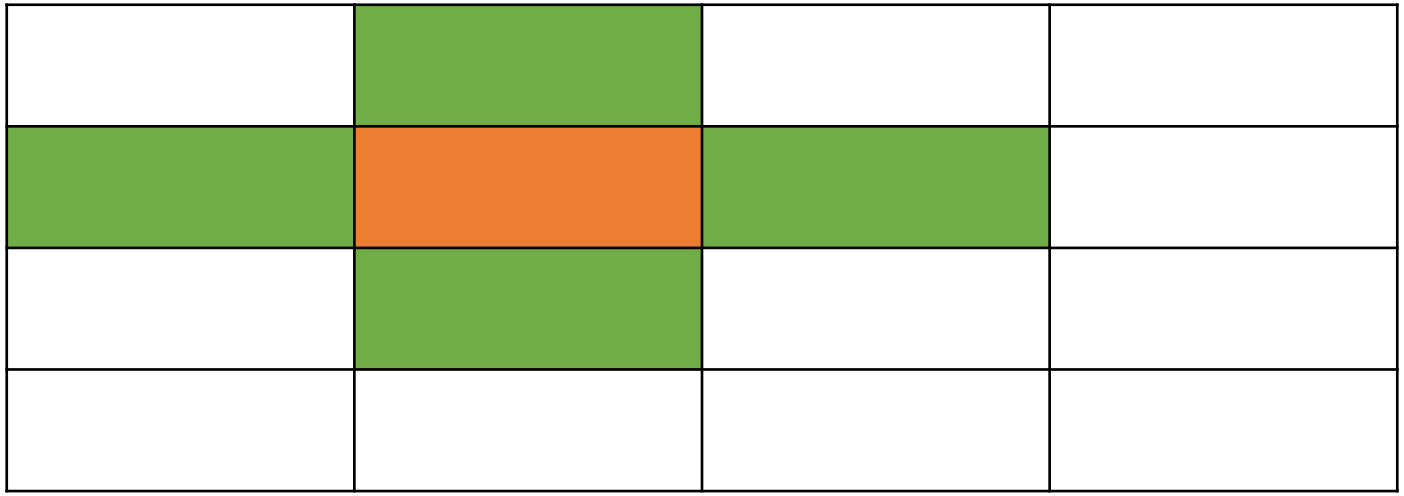
- Use DFS (Depth First Search) with priority to the smaller values

- During the DFS, branch out to the smallest value in reach

Solution explanation

- Each index has at most 4 possible paths to branch out to

- As shown, the orange index has the top, left, bottom, and right index to branch out to



Solution explanation

- For the solution, use DFS on the four possible pathways and find the minimum out of the four

- The minimum out of the four will be the final answer

Solution explanation

Visited_already function:

- Checks whether a certain index has been visited already or not

```
int visited_already(int visited[4][MAX * MAX], int row, int col, int index)
{
  for(int k = 0; k < col; k++)
  {
    if(visited[row][k] == index)
    {
      return 1;
    }
  }

  return 0;
}
```

Solution explanation

- source_row, source_col, dest_row, dest_col: source and destination indices
- matrix: 6 by 6 matrix
- Visited: DFS path for each possible answer
- Sum: each minimum sum
- Zero_count: number of zeroes
- Col_store: number of Indices passed by each DFS

```
int source_row, source_col, dest_row, dest_col;
int matrix[MAX][MAX];
int visited[4][MAX * MAX];
//sum of going to a certain path
//[0 - 3]: up, left, down, right, respectively
int sum[4];
//number of zeroes passed by going to that index
int zero_count = 0;
int col_store[4];
```

Solution explanation

- Scan for variables

```
//scan
scanf("%d %d %d %d", &source_row, &source_col, &dest_row, &dest_col);

//match the usual indices for arrays (0 - N)
source_row -= 1;
source_col -= 1;
dest_row -= 1;
dest_col -= 1;

//scan for the matrix
for(int i = 0; i < MAX; i++)
{
  for(int j = 0; j < MAX; j++)
  {
    scanf("%d", &matrix[i][j]);
  }
}
```

Solution explanation

- For loop for each pathway
- Insert the source as the first value for each path
- Reinitialize variables

```
for(int i = 0; i < 4; i++)
{
  int j = 0;
  zero_count = 0;
  int row = source_row;
  int col = source_col;
  visited[i][j] = source_row * 100 + source_col + 101;
  sum[i] = matrix[source_row][source_col];
  if(matrix[source_row][source_col] == 0)
  {
    zero_count++;
  }
  j++;
```

Solution explanation

- The preparation of each of the four paths

- i = 0: up
- i = 1: left
- i = 2: down
- i = 3: right

```
if(i == 0 && row > 0)
{
  row--;
}
else if(i == 1 && col > 0)
{
  col--;
}
else if(i == 2 && row < 5)
{
  row++;
}
else if(i == 3 && col < 5)
{
  col++;
}
```

Solution explanation

Start of DFS while loop:

- Insert the next value and sum up
- Minimum value was already decided in the later part of the loop

```
while(1)
{
    visited[i][j] = row * 100 + col + 101;
    int cur_index = visited[i][j];
    int dest_index = dest_row * 100 + dest_col + 101;
    sum[i] += matrix[row][col];
    j++;
```

Solution explanation

- If the destination is within reach, approach it directly as the next index

```
if(cur_index - 100 == dest_index)
{
  row--;
  continue;
}
else if(cur_index - 1 == dest_index)
{
  col--;
  continue;
}
else if(cur_index + 100 == dest_index)
{
  row++;
  continue;
}
else if(cur_index + 1 == dest_index)
{
  col++;
  continue;
}
else if(cur_index == dest_index)
{
  break;
}
```

Solution explanation

- Decide the four possible pathways to branch out to every single time:
- Also making sure each index has not been visited yet

```
int candidates[4][2] = {{9999, 0}, {9999, 0}, {9999, 0}, {9999, 0}};

if(row > 0 && visited_already(visited, i, j, (row - 1) * 100 + col + 101) == 0)
{
  candidates[0][0] = matrix[row - 1][col];
  candidates[0][1] = (row - 1) * 100 + col + 101;
}
if(col > 0 && visited_already(visited, i, j, row * 100 + col - 1 + 101) == 0)
{
  candidates[1][0] = matrix[row][col - 1];
  candidates[1][1] = row * 100 + col - 1 + 101;
}
if(row < 6 && visited_already(visited, i, j, (row + 1) * 100 + col + 101) == 0)
{
  candidates[2][0] = matrix[row + 1][col];
  candidates[2][1] = (row + 1) * 100 + col + 101;
}
if(col < 6 && visited_already(visited, i, j, row * 100 + col + 1 + 101) == 0)
{
  candidates[3][0] = matrix[row][col + 1];
  candidates[3][1] = row * 100 + col + 1 + 101;
}
```

Solution explanation

- Find the minimum value among the possible indices

```
int min_index;
int min = 9999;
for(int k = 0; k < 4; k++)
{
  if(candidates[k][0] == -9999)
  {
    continue;
  }
  else if(candidates[k][0] == 0 && zero_count == 1)
  {
    continue;
  }

  if(candidates[k][0] < min)
  {
    min = candidates[k][0];
    min_index = candidates[k][1];
  }
}
```

Solution explanation

- Next row and column values get updated depending on the minimum value and index
- If a zero exists, increase zero_count
- Col_store happens after the while loop and before the next for loop

```
    if(min == 0)
    {
        zero_count++;
    }
    row = min_index / 100 - 1;
    col = min_index % 100 - 1;
}
col_store[i] = j;
```

Solution explanation

- Once for loop for each path is done, compare the sum of each pathway
- Save the data for the minimum sum and print out its data

```
int min = 9999;
int index;

for(int i = 0; i < 4; i++)
{
  if(sum[i] < min)
  {
    index = i;
    min = sum[i];
  }
}

printf("%d\n", min);
for(int i = 0; i < col_store[index]; i++)
{
  printf("%d %d\n", visited[index][i] / 100, visited[index][i] % 100);
}
```

Solution analysis

Pros:

- Consistent
- Efficient due to greedy algorithm application

Cons:

- Quite lacking
- When the destination index is nearby, it will directly approach it
    - This reduces certain possible options that can reduce the sum

Thank you!