Session 24: Shortest Path Problem

Requirement for the assignment per Dr. Waxman

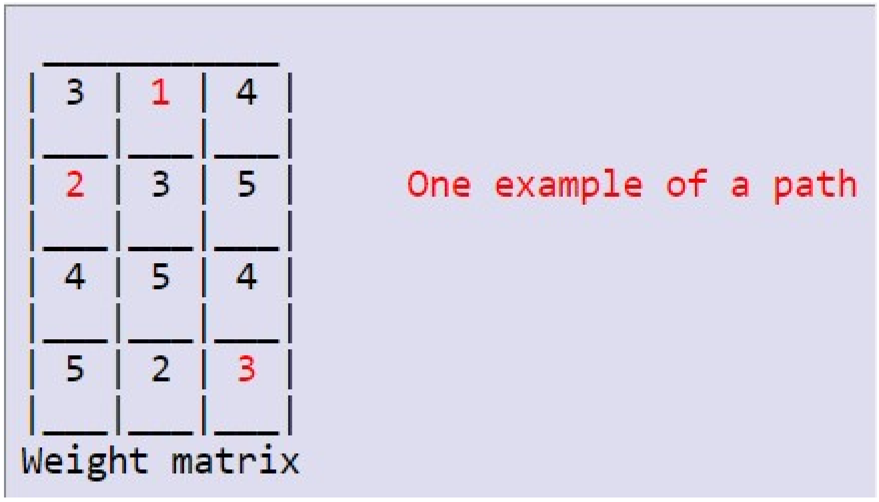
Note: Please submit two separate cpp plain text source code in gradescope.com. Submit the solution with memoization for solving it recursively to hw14a. Submit the one with the bottom up approach to hw14b.

Videos on Shortest Path Problem by Dr. Waxman

Recursive approach: https://www.youtube.com/watch?v=LxulXRatlxk
Non-recursive approach: https://www.youtube.com/watch?v=Sm-pdEAE4as

Shortest Path

The goal is to find the shortest path through a 2D grid, from left to right.



A path must visit exactly one square in each column, and each square in the path must be adjacent to the previous square (horizontally or diagonally). The grid is cylindrical, which means that the top row is touching the bottom row.

The number in each square (the weight) tells you the amount of time it takes to travel through the square. The shortest path is the path with the shortest possible total time. The path in the picture above has a total time of 2 + 1 + 3 = 6.

Data representation

Use 3 matrices of the same size:

- Weight matrix (see above)
- Cost matrix: cost[i][j] tells you total cost of the shortest path from the left side of the grid to the square in row i, column j.
- Path matrix or trace matrix to use for printing (pick one):
    - path[i][j] is a string of the row numbers of the shortest path that leads to that square. path[3][2] would be "103" in this example, because the shortest path that leads to that square travels through rows 1, 0, and 3.
    - trace[i][j] is the row number of the previous square in the shortest path that leads to that square. trace[3][2] would be 0 in this example, because the shortest path that leads to that square travels through row 0 of the previous column.

```
 ___________
| 3 | 3 | 7 |     The highlighted square has a cost of 3 because
|___|___|___|     the shortest path to that square is 2 + 1 = 3.
| 2 | 5 | 8 |
|___|___|___|
| 4 | 7 | 9 |     Note that the leftmost column of the weight and cost matrices are the same
|___|___|___|     because the path to that square only travels through one square.
| 5 | 5 | 6 |
|___|___|___|
 Cost matrix

 ___________
| 0 | 10|100|     The highlighted square in the lower right corner is "103" because
|___|___|___|     the shortest path to that square travels through rows 1, 0, and 3.
| 1 | 11|101|     The "103" comes from concatenating "10"
|___|___|___|     (from the other highlighted square) with "3" (the current row number).
| 2 | 12|112|
|___|___|___|     Note that these are strings and not integers (so they may begin with 0).
| 3 | 03|103|
|___|___|___|
 Path matrix
```

Algorithm

There are two ways to solve the problem, recursive (top-down) and non-recursive (bottom-up). Both approaches use a similar strategy: Fill up the cost matrix, then check the rightmost column to see which square has the lowest cost.

Recursive

Create a calculateCost function that takes 2 parameters, i and j, and returns the cost of the shortest path to the square in row i, column j.

- First, check for the base case. If you are in the leftmost column, the square's cost is the same as its weight.
- If the cost has already been calculated, return it (note that the cost matrix is being used as a memo).
- Calculate the costs of the 3 adjacent squares in the column to the left by calling the function recursively, and store these costs (up, left, down). When you calculate the row numbers of the adjacent squares, don't forget that the grid is cylindrical.
- Calculate the minimum of up, left, and right (use the min function in the algorithm library).
- Update the path matrix. Concatenate the minimum of the 3 paths with the current row number. You can use + to concatenate, and to_string(i) to convert row i to a string. to_string only works in C++11. See this page for instructions on how to use C++11.
- The cost of the current square is the minimum cost of the 3 adjacent squares + the weight of the current square. Store that number in the cost matrix and then return

it.

In the main function, call the calculateCost function once for each square in the rightmost column of the grid. Check which one has the lowest cost and print it. Then print the path from left to right, using either the path matrix or the trace matrix (pick one):

- Path matrix: Get the full path from the element of the path matrix that corresponds with the last square of the shortest path.
- Trace matrix: Starting with the last square of the shortest path, check the corresponding element in the trace matrix to get the row number for the previous square of the path. Repeat until you have the entire path, and store the elements in an array. Then print the array. Alternatively you can use a recursive print function which does not require an array.

Non-recursive

Fill in the cost matrix one column at a time, starting from the left.

- First, copy the leftmost column of the weight matrix to the cost matrix.
- Loop through the rest of the columns, calculating the cost of each square in each column, and update the path matrix.
- Calculate the cost the same way you did in the recursive version, except instead of doing recursive function calls, look up the costs of the 3 adjacent squares in the cost matrix (since you already calculated them).

After you fill in the cost matrix, check which square in the rightmost column has the lowest cost, and print the solution the same way you did in the recursive version.

Recursive template

```
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;

const int rows = 5, cols = 6;
string path[rows][cols];

// Returns the cost of the shortest path from the left to the squa
int calculateCost(int i, int j) {
   static int weight[rows][cols] = {{3,4,1,2,8,6},
                                    {6,1,8,2,7,4},
                                    {5,9,3,9,9,5},
                                    {8,4,1,3,2,6},
                                    {3,7,2,8,6,4}};
   // Declare the cost matrix.
   _____
   // If the cost has already been calculated, return it.
   _____
   // Check for the base case.
   if (j == 0) {
      path[i][j] = to_string(i);
      return weight[i][j];
   }

   // Calculate the costs of the 3 adjacent squares by calling the
   int up = _____
   int left = _____
   int down = _____
   // Find the minimum of the 3 costs.
   int minCost = _____
   // Update the path matrix (store the path to the current square
   // If up is the minimum, get the shortest path to the up-left s
      path[i][j] = _____ + to_string(i);
   // If left is the minimum...
      _____
   // If right is the minimum...
      _____
   // Calculate the cost of the current square, store the correct
   _____
}

int main() {
   int minRow = 0;
   // Call the calculateCost function once for each square in the
   // Check which one has the lowest cost and store the row number
   _____
   cout << "The length of the shortest path is " << calculateCost(
   cout << ".\nThe rows of the path from left to right are " << pa

   return 0;
}
```

Non-recursive template

```
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;

const int rows = 5, cols = 6;
int minRow, minLength;
string path[rows][cols];

// Calculates the cost of every square and fills in the cost matri
void calculateCosts() {
   static int weight[rows][cols] = {{3,4,1,2,8,6},
                                    {6,1,8,2,7,4},
                                    {5,9,3,9,9,5},
                                    {8,4,1,3,2,6},
                                    {3,7,2,8,6,4}};
   // Declare the cost matrix.
   _____
   // Copy the leftmost column of the weight matrix to the cost ma
   _____
   // For each remaining column,
      // Calculate the cost of each square in the column (non-recu
      // and store the correct number in the cost matrix and the c
   _____

   // Check which square in the rightmost column has the lowest co
   _____
}
```

```
int main() {
   calculateCosts();
   cout << "The length of the shortest path is " << minLength;
   cout << ".\nThe rows of the path from left to right are " << pa

   return 0;
}
```