Problem 2146: K Highest Ranked Items Within a Price Range

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a

0-indexed

2D integer array

grid

of size

m x n

that represents a map of the items in a shop. The integers in the grid represent the following:

0

represents a wall that you cannot pass through.

1

represents an empty cell that you can freely move to and from.

All other positive integers represent the price of an item in that cell. You may also freely move to and from these item cells.

It takes

1

step to travel between adjacent grid cells.

You are also given integer arrays

pricing

and

start

where

pricing = [low, high]

and

start = [row, col]

indicates that you start at the position

(row, col)

and are interested only in items with a price in the range of

[low, high]

(

inclusive

). You are further given an integer

k

.

You are interested in the

positions

of the

k

highest-ranked

items whose prices are

within

the given price range. The rank is determined by the

first

of these criteria that is different:

Distance, defined as the length of the shortest path from the

start

(

shorter

distance has a higher rank).

Price (

lower

price has a higher rank, but it must be

in the price range

).

The row number (

smaller

row number has a higher rank).

The column number (

smaller

column number has a higher rank).

Return

the

k

highest-ranked items within the price range

sorted

by their rank (highest to lowest)

. If there are fewer than

k

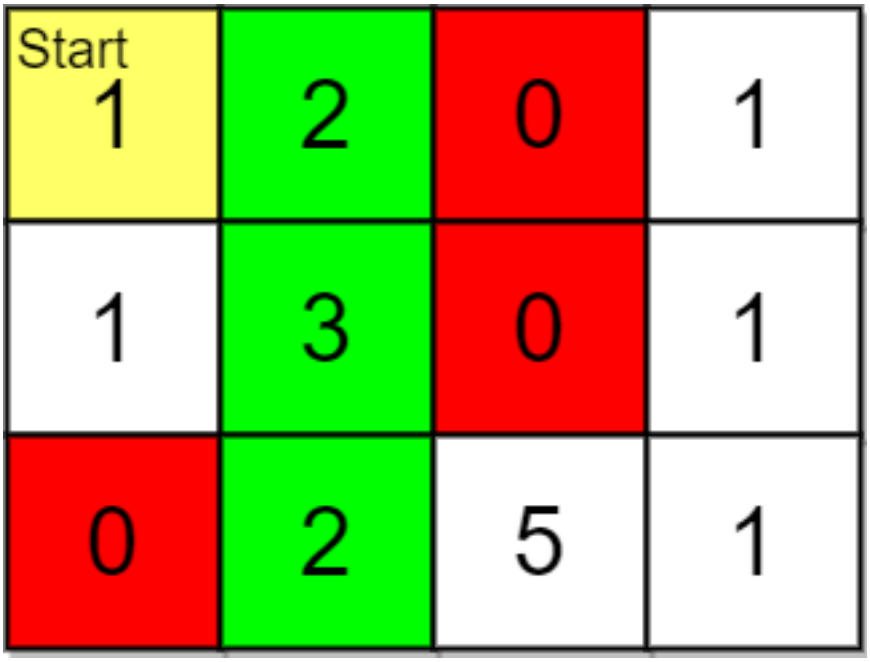
reachable items within the price range, return

all

of them

.

Example 1:



Input:

grid = [[1,2,0,1],[1,3,0,1],[0,2,5,1]], pricing = [2,5], start = [0,0], k = 3

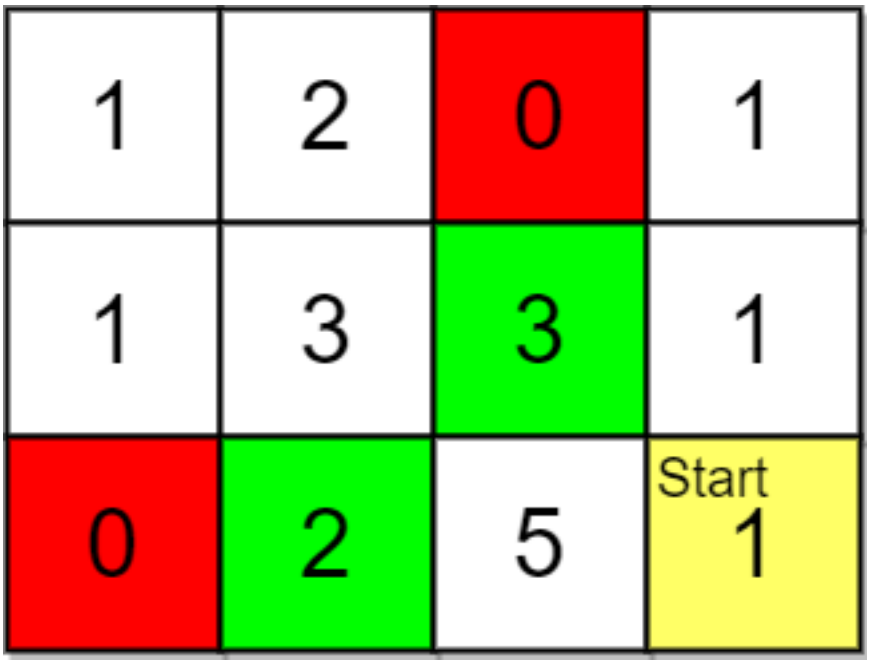
Output:

[[0,1],[1,1],[2,1]]

Explanation:

You start at (0,0). With a price range of [2,5], we can take items from (0,1), (1,1), (2,1) and (2,2). The ranks of these items are: - (0,1) with distance 1 - (1,1) with distance 2 - (2,1) with distance 3 - (2,2) with distance 4 Thus, the 3 highest ranked items in the price range are (0,1), (1,1), and (2,1).

Example 2:



Input:

grid = [[1,2,0,1],[1,3,3,1],[0,2,5,1]], pricing = [2,3], start = [2,3], k = 2

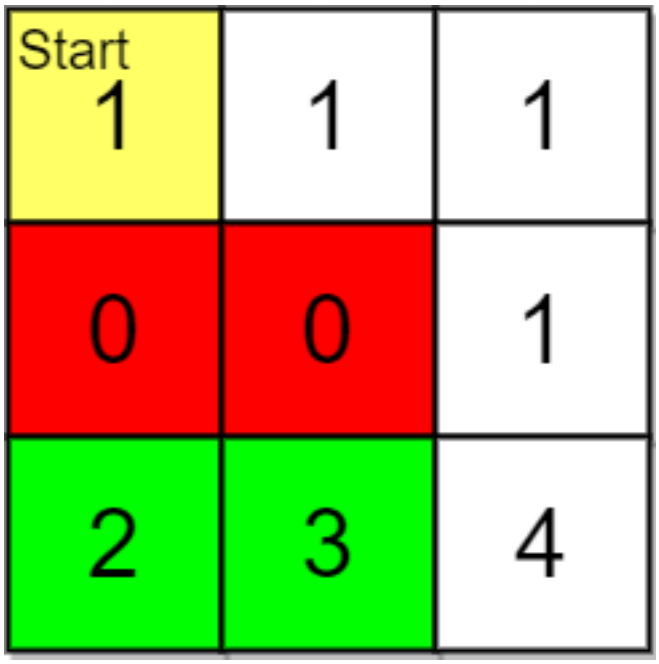
Output:

[[2,1],[1,2]]

Explanation:

You start at (2,3). With a price range of [2,3], we can take items from (0,1), (1,1), (1,2) and (2,1). The ranks of these items are: - (2,1) with distance 2, price 2 - (1,2) with distance 2, price 3 - (1,1) with distance 3 - (0,1) with distance 4 Thus, the 2 highest ranked items in the price range are (2,1) and (1,2).

Example 3:



Input:

grid = [[1,1,1],[0,0,1],[2,3,4]], pricing = [2,3], start = [0,0], k = 3

Output:

[[2,1],[2,0]]

Explanation:

You start at (0,0). With a price range of [2,3], we can take items from (2,0) and (2,1). The ranks of these items are: - (2,1) with distance 5 - (2,0) with distance 6 Thus, the 2 highest ranked items in the price range are (2,1) and (2,0). Note that k = 3 but there are only 2 reachable items within the price range.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 10

1 <= m * n <= 10

5

0 <= grid[i][j] <= 10

5

pricing.length == 2

2 <= low <= high <= 10

5

start.length == 2

0 <= row <= m - 1

0 <= col <= n - 1

grid[row][col] > 0

1 <= k <= m * n

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> highestRankedKItems(vector<vector<int>>& grid,
vector<int>& pricing, vector<int>& start, int k) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> highestRankedKItems(int[][] grid, int[] pricing,
int[] start, int k) {

}
}
```

Python3:

```
class Solution:
def highestRankedKItems(self, grid: List[List[int]], pricing: List[int],
start: List[int], k: int) -> List[List[int]]:
```

Python:

```
class Solution(object):
def highestRankedKItems(self, grid, pricing, start, k):
"""
:type grid: List[List[int]]
:type pricing: List[int]
:type start: List[int]
:type k: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @param {number[]} pricing
* @param {number[]} start
* @param {number} k
* @return {number[][]}
*/
var highestRankedKItems = function(grid, pricing, start, k) {

};
```

TypeScript:

```
function highestRankedKItems(grid: number[][], pricing: number[], start:
number[], k: number): number[][] {
```

```
};
```

C#:

```
public class Solution {
public IList<IList<int>> HighestRankedKItems(int[][] grid, int[] pricing,
int[] start, int k) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** highestRankedKItems(int** grid, int gridSize, int* gridColSize, int*
pricing, int pricingSize, int* start, int startSize, int k, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

```
func highestRankedKItems(grid [][]int, pricing []int, start []int, k int)
[][]int {

}
```

Kotlin:

```
class Solution {
fun highestRankedKItems(grid: Array<IntArray>, pricing: IntArray, start:
IntArray, k: Int): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func highestRankedKItems(_ grid: [[Int]], _ pricing: [Int], _ start: [Int], _
k: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn highest_ranked_k_items(grid: Vec<Vec<i32>>, pricing: Vec<i32>, start:
Vec<i32>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer[]} pricing
# @param {Integer[]} start
# @param {Integer} k
# @return {Integer[][]}
def highest_ranked_k_items(grid, pricing, start, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer[] $pricing
* @param Integer[] $start
* @param Integer $k
* @return Integer[][]
*/
function highestRankedKItems($grid, $pricing, $start, $k) {

}
}
```

Dart:

```
class Solution {
List<List<int>> highestRankedKItems(List<List<int>> grid, List<int> pricing,
List<int> start, int k) {

}
}
```

Scala:

```
object Solution {
def highestRankedKItems(grid: Array[Array[Int]], pricing: Array[Int], start:
Array[Int], k: Int): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec highest_ranked_k_items(grid :: [[integer]], pricing :: [integer], start
:: [integer], k :: integer) :: [[integer]]
def highest_ranked_k_items(grid, pricing, start, k) do

end
end
```

Erlang:

```
-spec highest_ranked_k_items(Grid :: [[integer()]], Pricing :: [integer()],
Start :: [integer()], K :: integer()) -> [[integer()]].
highest_ranked_k_items(Grid, Pricing, Start, K) ->
.
```

Racket:

```
(define/contract (highest-ranked-k-items grid pricing start k)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?) exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: K Highest Ranked Items Within a Price Range
* Difficulty: Medium
* Tags: array, graph, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<int>> highestRankedKItems(vector<vector<int>>& grid,
vector<int>& pricing, vector<int>& start, int k) {

}
};
```

Java Solution:

```
/**
* Problem: K Highest Ranked Items Within a Price Range
* Difficulty: Medium
* Tags: array, graph, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<List<Integer>> highestRankedKItems(int[][] grid, int[] pricing,
int[] start, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: K Highest Ranked Items Within a Price Range
Difficulty: Medium
Tags: array, graph, sort, search, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def highestRankedKItems(self, grid: List[List[int]], pricing: List[int],
start: List[int], k: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def highestRankedKItems(self, grid, pricing, start, k):
"""
:type grid: List[List[int]]
:type pricing: List[int]
:type start: List[int]
:type k: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: K Highest Ranked Items Within a Price Range
* Difficulty: Medium
* Tags: array, graph, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} grid
```

```
* @param {number[]} pricing
* @param {number[]} start
* @param {number} k
* @return {number[][]}
*/
var highestRankedKItems = function(grid, pricing, start, k) {

};
```

TypeScript Solution:

```
/**
* Problem: K Highest Ranked Items Within a Price Range
* Difficulty: Medium
* Tags: array, graph, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function highestRankedKItems(grid: number[][], pricing: number[], start:
number[], k: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: K Highest Ranked Items Within a Price Range
* Difficulty: Medium
* Tags: array, graph, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<IList<int>> HighestRankedKItems(int[][] grid, int[] pricing,
int[] start, int k) {
```

```
}
}
```

C Solution:

```
/*
* Problem: K Highest Ranked Items Within a Price Range
* Difficulty: Medium
* Tags: array, graph, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** highestRankedKItems(int** grid, int gridSize, int* gridColSize, int*
pricing, int pricingSize, int* start, int startSize, int k, int* returnSize,
int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: K Highest Ranked Items Within a Price Range
// Difficulty: Medium
// Tags: array, graph, sort, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func highestRankedKItems(grid [][]int, pricing []int, start []int, k int)
[][]int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun highestRankedKItems(grid: Array<IntArray>, pricing: IntArray, start:
IntArray, k: Int): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func highestRankedKItems(_ grid: [[Int]], _ pricing: [Int], _ start: [Int], _
k: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: K Highest Ranked Items Within a Price Range
// Difficulty: Medium
// Tags: array, graph, sort, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn highest_ranked_k_items(grid: Vec<Vec<i32>>, pricing: Vec<i32>, start:
Vec<i32>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer[]} pricing
```

```
# @param {Integer[]} start
# @param {Integer} k
# @return {Integer[][]}
def highest_ranked_k_items(grid, pricing, start, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer[] $pricing
* @param Integer[] $start
* @param Integer $k
* @return Integer[][]
*/
function highestRankedKItems($grid, $pricing, $start, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> highestRankedKItems(List<List<int>> grid, List<int> pricing,
List<int> start, int k) {

}
}
```

Scala Solution:

```
object Solution {
def highestRankedKItems(grid: Array[Array[Int]], pricing: Array[Int], start:
Array[Int], k: Int): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec highest_ranked_k_items(grid :: [[integer]], pricing :: [integer], start
:: [integer], k :: integer) :: [[integer]]
def highest_ranked_k_items(grid, pricing, start, k) do

end
end
```

Erlang Solution:

```
-spec highest_ranked_k_items(Grid :: [[integer()]], Pricing :: [integer()],
Start :: [integer()], K :: integer()) -> [[integer()]].
highest_ranked_k_items(Grid, Pricing, Start, K) ->
.
```

Racket Solution:

```
(define/contract (highest-ranked-k-items grid pricing start k)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?) exact-integer? (listof (listof exact-integer?)))
)
```