Deque Applications

Ulises Tirado Zatarain [1]
(ulises.tirado@cimat.mx)

[1] Algorists Group

July, 2016

Outline

1. Introduction
   - The challenge
   - Definitions
2. Solving the problem and applications

The ocean currents challenge

Statement

For a boat on a large body of water, strong currents can be dangerous, but with careful planning, they can be harnessed to help the boat reach its destination. Your job is to help in that planning. At each location, the current flows in some direction. The captain can choose to either go with the flow of the current, using no energy, or to move one square in any other direction, at the cost of one energy unit. The boat always moves in one of the following eight directions: north, south, east, west, northeast, northwest, southeast, southwest. The boat cannot leave the boundary of the lake.

You are to help him devise a strategy to reach the destination with the minimum energy consumption.

The ocean currents challenge

Input

The lake is represented as a rectangular grid. The first line of input contains two integers r and c, the number of rows and columns in the grid. The grid has no more than 1000 rows and no more than 1000 columns. Each of the following r lines contains exactly c characters, each a digit from 0 to 7 inclusive. The character '0' means the current flows north (i.e. up in the grid, in the direction of decreasing row number), '1' means it flows northeast, '2' means east (i.e. in the direction of increasing column number), '3' means southeast, and so on in a clockwise mannere.

The ocean currents challenge

Input

$$\begin{array}{ccccc} 7 & & 0 & & 1 \\ & \ddots & \vdots & \iddots & \\ 6 & \cdots & * & \cdots & 2 \\ & \iddots & \vdots & \ddots & \\ 5 & & 4 & & 3 \end{array}$$

The line after the grid contains a single integer n, the number of trips to be made, which is at most 50. Each of the following n lines describes a trip using four integers r_s, c_s, r_d, c_d, giving the row and column of the starting point and the destination of the trip. Rows and columns are numbered starting with 1.

The ocean currents challenge

Output

For each trip, output a line containing a single integer, the minimum number of energy units needed to get from the starting point to the destination.

Example

Sample input   Sample output

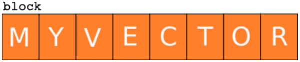
```
5 5                                                       0
0 4 1 2 5                                                 2
0 3 3 5 5                                                 1
6 4 7 3 4
7 2 3 7 7
0 2 0 6 2
3
4 2 4 2
4 5 1 4
5 3 3 4
```

Big question!

How can we solve it?
Some ideas...?

Definitions

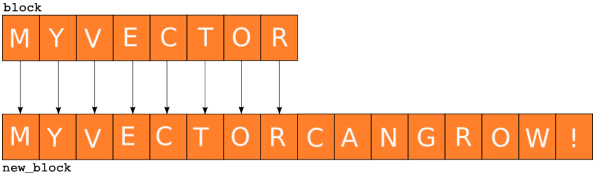
- With an array or vector we have a fixed contiguos block of memory and we access to k-th element in $\mathcal{O}(1)$time.



block

M	Y	V	E	C	T	O	R

- If we need to insert a new element then we have to copy this block to a new location with enough space. This is $\mathcal{O}(n)$ time.



block

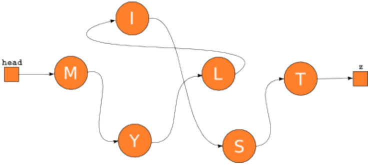
M	Y	V	E	C	T	O	R

M	Y	V	E	C	T	O	R	C	A	N	G	R	O	W	!

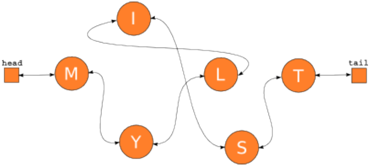
new_block

Definitions

- With a linked list we can insert elements in $\mathcal{O}(1)$ time, but we access to k-th element in $\mathcal{O}(n)$ time.

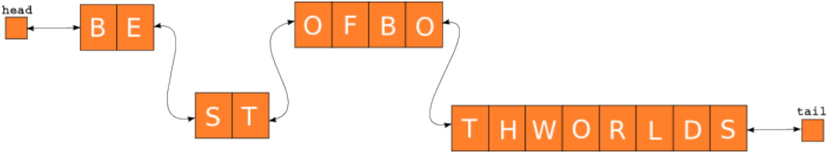


- We can improve our linked list adding another direction, but we have the same inconveniences:



Definitions

- But, deques (in particular `std::deque`) are quasi-magical:



- We can insert elements in both ends of the structure in $\mathcal{O}(1)$ time and we access to k-th element in $\mathcal{O}(1+\alpha)$ time, where α is called load factor and $\alpha \ll n$.

BFS 0/1

```
function BFS-01(struct node *source, struct node *target) begin
    dq.push-front(source);
    while !dq.is-empty() && current != target do
        current←dq.front();
        dq.pop-front();
        for each $v \in$ neighborhood(current) do
            if is-unvisited(v) || can-improve(v) then:
                dq.push-front(v); visit-or-improve(v);
            else if is-unvisited(v) then:
                dq.push-back(v); visit(v);
            end
        end
    end
end
```

References I

- Robert Sedgewick - Algorithms C++
- Wikipedia