Lab 5 Report

1 Introduction

In this lab, we will write a program to tell the longest distance we can slide in a given $X \times Y$ map.

2 Solution

We will use recursion to solve this problem. In the function we check whether we can slide to the north, south, west and east in turn. If we can go one way, we move to that point and run the function at this point. If we can't, we return to the next address of the function.

2.1 Algorithm

We here write a function to store the longest distance in memory starting at arbitrary point as followed.

```
procedure Find_Way(*P, *L, m, n)
*L1 := *L + 1
l := *P// l contains the altitude of the point
// we first test whether we can go north
N := P - n// pointer to the target point
if N < x4002 then
    goto South
nor := *N// altitude of the point
if nor < l then
    procedure Find_Way(*N, *L1, m, n)
// next we test whether we can go south
South:
S := P + n// pointer to the target point
En := x4001 + m * n// pointer to the last point
if En < S then
    goto West
sou := *S// altitude of the point
if sou < l then
    procedure Find_Way(*S, *L1, m, n)
// next we test whether we can go west
West:
T := P
while T >= x4002
    if T = x4002 then
        goto East// if the point is at the far left
    T := T - n
W := P - 1// pointer to the target point
wes := *W// altitude of the point
if wes < l then
    procedure Find_Way(*W, *L1, m, n)
// next we test whether we can go east
East:
T := P
while T <= En
    if T = En then
        goto ending// if the point is at the far right
```

```
    T := T + n
E := P + 1// pointer to the target point
eas := *E// altitude of the point
if eas < l then
    procedure Find_Way(*E, *L1, m, n)
ending:
if *L1 > *L then
    *L = *L1
return
```

2.2 Essential part of the code

First I'll show how we check the west point and go to that point.

```
; R0 points to the current point, R1 contains the altitude of it
WEST    LDI R4, INI1; R4 gets the value of N
        NOT R4, R4
        ADD R4, R4, #1; inverse
        LD R3, CHE; R3 points to the first point
        NOT R3, R3
        ADD R3, R3, #1; inverse
        ADD R2, R0, #0
        ADD R2, R2, R3
MIN     BRz RIGHT; if the point is on the far left
        BRn DONEL
        ADD R2, R2, R4
        BR MIN
DONEL   ADD R3, R0, #-1; R3 points to the point left
        LDR R4, R3, #0; R4 contains the altitude of R3
        NOT R4, R4
        ADD R4, R4, #1
        ADD R4, R4, R1
        BRnz RIGHT
        LDR R2, R6, #0; POP R2
        STR R0, R6, #0; PUSH R0
        ADD R0, R3, #0
        JSR FIND
        LDR R0, R6, #0; POP R0
        STR R2, R6, #0; PUSH R2
```

Secondly, the ending part of the function is as followed. It's worth noting that we need to pop all the elements we push in the function to ensure the stack works well.

```
OVER    LDR R2, R6, #0
        ADD R6, R6, #1; POP R2
        NOT R3, R5
        ADD R3, R3, #1
        ADD R3, R3, R2
        BRzp OVE; not larger than original R2
        ADD R2, R5, #0
OVE
        LDR R5, R6, #0
        ADD R6, R6, #1; POP R5
        LDR R1, R6, #0
```

```
        ADD R6, R6, #1; POP R1
        LDR R4, R6, #0
        ADD R6, R6, #1; POP R4
        LDR R7, R6, #0
        ADD R6, R6, #1; POP R7
        LDR R3, R6, #0
        ADD R6, R6, #1; POP R3
        RET
```

Lastly, the part outside of the function is as followed. In the loop, we will continue the loop until the pointer R0 points to the last point in the area.

```
        .ORIG x3000
        LD R6, STACK; R6 is the stack pointer
        AND R2, R2, #0; initialize R0 to 0
        LD R0, CHE; R0 points to each point in the skiing field
        LDI R4, INI1; R4 contains n
        LDI R5, INI0; R5 contains m
        AND R3, R3, #0
TOT     ADD R3, R3, R4
        ADD R5, R5, #-1
        BRp TOT; R3 contains the quantity of all the points
        LD R4, INI1
        ADD R3, R3, R4; R3 points to the last point
        NOT R3, R3
        ADD R3, R3, #1; inverse
LOOP    AND R5, R5, #0; initialize R5 to 0
        JSR FIND
        ADD R0, R0, #1
        ADD R4, R3, R0
        BRnz LOOP
        HALT
```

3 Q & A

> Q: How to optimize your algorithm?

A: Every time we start at a new point, we check whether we have been to this point. If we have been to here, we need to skip this point and move to the next.

> Q: What's the maximum recursion depth of a $m \times n$ area?

A: When we can go through every point in one path, which the recursion depth is $m \times n$.