# ALGORITHM

The program uses a recursion to traverse all available path starting from a particular place in the map. It`s very similar to the *Exit the Maze* paradigm in textbook P.282. So let`s be brief here.

Furthermore, there`s a loop outside the recursion to traverse all available starting place. When you return to the starting place in a recursion and have nowhere to go, which means you`ve gone through all possible path, change a new starter and just continue the recursion.

Every time you step forward, compare the path length now with the biggest length before and decide whether to update your answer.

# MAIN CODE

1. NEWSTART    LD  R6, USP\_BOTTOM; reset R6 as USP
2. LD  R0, START
3. ADD R0, R0, #1
4. LD  R1, MAP\_TAIL
5. NOT R1, R1
6. ADD R1, R1, #1
7. ADD R1, R0, R1
8. BRp ENDSKI; have tried all place to start
9. ST  R0, START; update current start point
11. ADD R6, R6, #-4
12. AND R7, R7, #0
13. STR R7, R6, #3; PUSH PC, but not avaliable for the first step
14. STR R0, R6, #2; PUSH start point addr
15. AND R1, R1, #0
16. STR R1, R6, #1; PUSH breadcrumb, unavaliable for the first step
17. ADD R2, R1, #1
18. STR R2, R6, #0; PUSH path length, initialize as 1
19. BRnzp SKI\_N; start recursion

Code above shows how to change a new place as starter and how to initialize user stack.

1. SKI\_LOOP    ADD R6, R6, #-4
2. STR R7, R6, #3; PUSH PC for previous step
3. STR R0, R6, #2; PUSH addr now
4. STR R1, R6, #1; PUSH breadcrumb
5. STR R2, R6, #0; PUSH path length now
7. LD  R3, RESULT
8. NOT R3, R3
9. ADD R3, R3, #1
10. ADD R3, R3, R2
11. BRnz SKI\_N; length now is no bigger than the result before
12. ST  R2, RESULT; get a new longest path, update RESULT

Code above shows what to do at the very beginning of a new recursion.

1. ; try to ski toward north
2. SKI\_N       LD  R0, MASK\_S
3. LDR R1, R6, #1
4. AND R0, R0, R1
5. BRp SKI\_E; came here form north in last step, don`t go back
6. LDR R0, R6, #2; addr now
7. LDI R1, COLUMN
8. NOT R1, R1
9. ADD R1, R1, #1
10. ADD R0, R0, R1
11. LD  R1, MAP\_HEAD
12. NOT R1, R1
13. ADD R1, R1, #1
14. ADD R0, R0, R1
15. BRn SKI\_E; cannot go north, out of map
16. LDR R0, R6, #2; addr now
17. LDI R1, COLUMN
18. NOT R1, R1
19. ADD R1, R1, #1
20. ADD R2, R0, R1; addr of north
21. LDR R0, R0, #0; altitude here
22. LDR R2, R2, #0; altitude of north
23. NOT R2, R2
24. ADD R2, R2, #1
25. ADD R2, R2, R0
26. BRnz SKI\_E; cannot go north for it`s higher
27. LDR R0, R6, #2; addr now
28. LDI R1, COLUMN
29. NOT R1, R1
30. ADD R1, R1, #1
31. ADD R0, R0, R1; new addr
32. LD  R1, MASK\_N; breadcrumb
33. LDR R2, R6, #0
34. ADD R2, R2, #1; new path length
35. JSR SKI\_LOOP; north is avaliable

Code above shows three different limitations to tell whether Patt can ski to north or not now.

# TA`s CHECK

Q1: Can you briefly tell me your algorithm?

A1: ……( The same as ALGORITHM part )

Q2: What`s the time complexity of your program?

A2: Apparently the time complexity of outside loop is the number of points in map. I`m not sure the answer to the inside recursion. N\*3^N maybe.

Q3: How you transmit parameters you need in recursion and loop, respectively?

A3: I store the parameters of loop like temporary result and current starter in memory. Every time I need to change a new starter, I fetch them in memory and update them if needed. Parameters of recursion such as previous PC, length now and breadcrumb are pushed into user stack.