Report 5

Algorithm explanation

This program contains 3 parts: input, dfs function part and main part.

- input:

  `1.` first get in a char, store it in the R1.

  If the next char is '\r', store the first char to R3.

  else store the second char plus 10 to R3.

  `2.` then loop R3 times to input numbers, one row contains 2 numbers.

- main part: init the necessary register and JSR the dfs function.

  ```
  dfs(1,0)
  ```

- dfs function:

  - push R1(the row), R2(now state: using total number to specify if each number is included), R7(the location)

  -
    ```
    void dfs(int R1, int R2){
        if(R1==R3+1){
            if(R2==(1<<R3))PRINT;
            RETURN
        }
        dfs(R1+1,R2+(1<<(A1[2*(R1-1)]-1)));
        dfs(R1+1,R2+(1<<(A1[2*(R1-1)+1]-1)));
    }
    ```
  - pop R1,R2,R7

Essential parts

input the first number

```
;input
            GETC
            OUT
            ADD R3,R0,#0
            LD R4,N48
            ADD R3,R3,R4
            GETC
            OUT
            ADD R4,R0,#-10
            BRz inover
            AND R3,R3,#0
            LD R4,N48
            ADD R0,R0,R4
```

```
            LD R4,P10
            ADD R3,R0,R4
            GETC
            OUT
    inover  ADD R4,R3,#0        ;cnt
            LD R6,A1
```

push and pop

```
;push
            ADD R6,R6,#-1
            STR R1,R6,#0        ;m
            ADD R6,R6,#-1
            STR R2,R6,#0        ;now
            ADD R6,R6,#-1
            STR R7,R6,#0
;pop
    NEXT    LDR R7,R6,#0
            ADD R6,R6,#1
            LDR R2,R6,#0
            ADD R6,R6,#1
            LDR R1,R6,#0
            ADD R6,R6,#1
            RET
```

judge if the dfs can be over

```
;judge
            NOT R0,R1
            ADD R0,R0,R3
            ADD R0,R0,#1
            BRnp CON
            AND R0,R0,#0
            ADD R0,R0,#1
            ADD R4,R3,#0
    zuoyi   ADD R0,R0,R0
            ADD R4,R4,#-1
            BRnp zuoyi
            NOT R0,R0
            ADD R0,R0,R2
            ADD R0,R0,#2
            BRnp NEXT
            BRz  PRINT
    CON     LD R5,A1
            ADD R0,R1,R1
            ADD R5,R5,R0
            LDR R4,R5,#0
            LD R0,ANSWER
            ADD R0,R0,R1
            STR R4,R0,#0
            AND R0,R0,#0
            ADD R0,R0,#1
            ADD R4,R4,#-1
            BRz ok
    zuoyi2  ADD R0,R0,R0
            ADD R4,R4,#-1
            BRnp zuoyi2
    ok      ADD R2,R2,R0
            ADD R1,R1,#1
```

Q&A

1. How to implement recursion

   Answer:

   - push R1(the row), R2(now state: using total number to specify if each number is included), R7(the location)

   -
     ```
     void dfs(int R1, int R2){
         if(R1==R3+1){
             if(R2==(1<<R3))PRINT;
             RETURN
         }
         dfs(R1+1,R2+(1<<(A1[2*(R1-1)]-1)));
         dfs(R1+1,R2+(1<<(A1[2*(R1-1)+1]-1)));
     }
     ```
   - pop R1,R2,R7

2. Where is the part of load A1[2*(R1-1)]

   Answer:

```
    CON     LD R5,A1
            ADD R0,R1,R1
            ADD R5,R5,R0
            LDR R4,R5,#0
```

3. How to ensure that the answer sequence is the same as the previous one

   Answer: The ANSWER array contains R1 rows, every row indicates to a row in the input array.

   In the output part output the ANSWER array in sequence.