2021 年 6 月 2 日

スーパーコン２１予選・認定問題：マス目と到達可能性判定問題

1. 問題説明

N 行 N 列のマス目状の盤があり、y（$1 \leq y \leq N$）行 x（$1 \leq x \leq N$）列目のマスをマス (y, x) と呼ぶ。マス $(1, 1)$ を除く $N^2 - 1$ 個のマスのうち N 個のマスには障害物があり、i 番目（$1 \leq i \leq N$）の障害物はマス (y_i, x_i) にある。

マス $(1, 1)$ には警備用のロボットがいて、このロボットは以下の４つの基本命令（U、D、L、R）からなる M 個の命令（各命令は長さ 1 以上の基本命令の列）のうちいくつかを搭載することができる。j 番目（$1 \leq j \leq M$）の命令は U、D、L、R の４種類の文字からなる文字列 S_j によって表され、S_j の左から基本命令を順々に実行することを表す。

- U: マス (y, x) からマス $(y - 1, x)$ に移動する。
- D: マス (y, x) からマス $(y + 1, x)$ に移動する。
- L: マス (y, x) からマス $(y, x - 1)$ に移動する。
- R: マス (y, x) からマス $(y, x + 1)$ に移動する。

各基本命令において、移動先が盤上に存在しない場合や移動先に障害物がある場合には、移動せず壊れて停止する（その後の命令は実行されない）。

ここで、ロボットは次の仕様（†）を満たして欲しい：「４つの基本命令の 0 回以上の繰り返しで到達可能な任意のマスに、搭載された命令の 0 回以上の繰り返しで到達可能」。ただし、命令の実行途中に経由できるだけでは到達可能とは呼ばない。また、ロボットが壊れるような命令を実行してはならない。

さて、あなたの仕事はロボットが仕様（†）を満たすかどうか検査することである。加えて、（予選問題では）搭載する命令の個数を少なくできるとさらに良い。

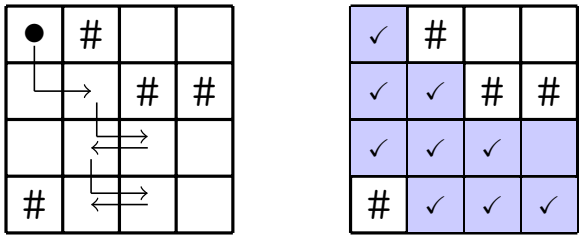

図 1: 問題例

図 1 に問題例を示す（$N = 4$）。● はロボットのいるマス（つまり、マス $(1, 1)$）、#は障害物のあるマスを表す。ロボットには 2 つの命令、DR と L が搭載されているとする。この時たとえばマス $(4, 2)$ には、DR、DR、L、DR、L を順に実行することで到達可能である（図 1 左の矢印）。ここで、最後の L を実行する前の DR の実行でマス $(4, 2)$ を経由しているが、経由できるだけでは到達可能とは呼ばないことに注意すること。図 1 右の、青く塗られたマスは 4 つの基本命令で到達可能なマス、✓ がついたマスは搭載された命令で到達可能なマスをあらわす。したがって、この問題例において、仕様（†）は満たされない（マス $(3, 4)$ には 4 つの基本命令の 0 回以上の繰り返しで到達可能だが、搭載された命令の 0 回以上の繰り返しで到達できない）。

入力

入力は以下の形式で与えられる：

N
$y_1\ x_1$
$\vdots$
$y_N\ x_N$
M
S_1
$\vdots$
S_M

- 1 行目には整数 N が与えられる。
- 2 行目から $1 + N$ 行目のうちの i 行目には、整数 y_i と整数 x_i が空白区切りで与えられる。
- $2 + N$ 行目には整数 M が与えられる。
- $3 + N$ 行目から $2 + N + M$ 行目のうちの j 行目には、文字列 S_j が与えられる。

制約

入力データは以下の制約を満たすと仮定してよい：

- $2 \leq N \leq 250$。
- $1 \leq y_i \leq N$、$1 \leq x_i \leq N$、$(y_i, x_i) \neq (1, 1)$、かつ $i \neq j$ のとき $(y_i, x_i) \neq (y_j, x_j)$。

- $1 \leq M \leq 2,000$。
- S_j は U、D、L、R の文字からなる、$1 \leq |S_j|$、かつ $\sum_{j=1}^{M} |S_j| = N \times M$。ただし $|S_j|$ は、文字列 S_j の長さを表す。

出力

出力は、標準出力におこない、各行の末尾には改行を入れること。
ロボットが仕様 (†) を満たすことができる場合、以下の出力形式で出力すること。

```
YES
k
v_1 v_2 ... v_k
```

- 1 行目には YES を出力する。
- 2 行目には整数 k （$1 \leq k \leq M$） を出力する。
- 3 行目には k 個の整数 $v_1, v_2, \ldots, v_k$ を空白区切りで出力する。各 $i (1 \leq i \leq k)$ について $1 \leq v_i \leq M$ を満たすようにすること。この時、v_1 から v_k の k 個の命令を搭載したロボットについて仕様（†）を満たすことをあらわす。
- 上記の条件を満たさない出力については不正解として扱われる。

ロボットが仕様（†）を満たすことができない場合、以下の出力形式で出力すること。

```
NO
```

- 1 行目に NO を出力する。

なお、（予選問題、級認定問題ともに）解の出力は 10 回までおこなってよい。（時間内に最後まで出力された解のうち）最後に出力された解のみを解答として使用する。毎回完全な解を出力し、解の間に空行などの余計な文字を入れないこと。解を 11 回以上出力した場合には不正解と見なされる。出力方法の例については、後述の出力例を参考にするとよい。出力する解は「搭載する命令の個数（k）」が少ないほど有利である。

入出力の例

例 1　以下の入力を考える。

```
4
1 2
2 3
2 4
4 1
3
DR
L
RRRRRRRRR
```

この入力は問題文中の問題例（図 1）に対応する。この時、仕様（†）を満たさないので以下を出力する。

```
NO
```

例２　以下の入力を考える。

```
4
1 2
2 3
3 4
4 1
3
DR
L
RRRRRRRRR
```

以下の図 2 は上記の入力に対応する図である。青く塗られたマスは 4 つの基本命令の 0 回以上の繰り返しで到達可能なマス、✓ がついたマスは、3 つ全ての命令を搭載した場合に、搭載された命令の 0 回以上の繰り返しで到達可能なマスをあらわす。

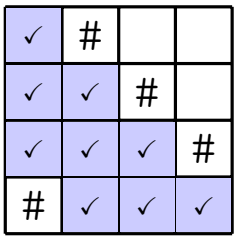

図 2: 例 2 の入力に対応する図

この時、仕様（†）を満たすので、たとえば以下のように出力する。

```
YES
3
1 2 3
```

なお、以下の出力の方がよりよい解である。

```
YES
2
1 2
```

この時、「搭載する命令の個数（k）」は 2 である。なお解の出力は 10 回までおこなってよい。以下は 3 回の解の出力がおこなわれている例である。

```
YES
3
1 2 3
NO
YES
2
1 2
```

この時最後に出力された解（3 番目の解）のみが解答として使用される。出力方法に関しては、添付プログラム template.c の関数 output を参考にするとよい。

例３　以下の入力を考える。

```
2
2 1
2 2
2
RRL
R
```

以下の図 3 は上記の入力に対応する図である。青く塗られたマスは 4 つの基本命令の 0 回以上の繰り返しで到達可能なマス、✓ がついたマスは、2 つ全ての命令を搭載した場合に、搭載された命令の 0 回以上の繰り返しで到達可能なマスをあらわす。この時、たとえば以下のように出力する。

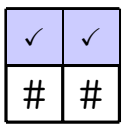

図 3: 例 3 の入力に対応する図

```
YES
1
2
```

なお命令 1 だけを搭載した場合では、仕様（†）を満たさないことに注意すること（ロボットが壊れる移動は実行できないことに注意すること）。

2.　予選問題

前節で述べた問題を解くプログラムを作成する。

審査方法

- 審査は、各チームの応募プログラムを、10 個の問題例に対し、各 10 秒間実行し、(i) 正解数が多い順、もし同数の場合には、(ii) 「搭載する命令の個数 (k)」の総和が小さい順、もし同点の場合には、(iii) 総実行時間が短い順で順位を決める。
- 10 個の問題例は、入力生成器プログラム `generator.c` に従って生成されたものを使用する（参照：付録 A.)。
- 各問題例毎に合計 10 回まで解を出力してもよいが、最後に出力された解のみを解答として使用する。
- 最終的な予選通過者の選抜には、上記の順位の他、応募時に提出するプログラムの解説 (*1) も参考にする。(*1) 解答プログラムの基本方針やアルゴリズムをＡ４で 2 ページ程度で説明した文書。**チーム名**`-algo.txt` もしくは、**チーム名**`-algo.docx` というファイルで提出すること。

実行環境

- OS: macOS Big Sur、CPU: プロセッサ 3.6GHz 第 8 世代 Intel Core i3 を搭載したコンピュータ上でおこなう。
- コンパイラは Homebrew GCC 10.2.0_3 を使用する。各言語について次のようにコンパイルする。[1]
  `gcc-10 -O2 -std=gnu17 -lm` **チーム名**`.c`　(C 言語の場合)
  `g++-10 -O2 -std=gnu++20` **チーム名**`.cpp`　(C++言語の場合)
- メモリ使用量上限を 1 ギガバイト、スタック使用量上限を 65532 キロバイトとする。具体的には、次のコマンドを実行した上で審査をおこなう：
  `ulimit -d 1000000 -m 1000000 -v 1000000 -s 65532`
- 各問題例に対して、コンパイルした実行可能プログラムを 10 秒間実行し、10 秒間のうち、最後に出力された解を使用する。プログラムは 10 秒以内に停止しなくてもよい。つまり、時間切れ強制終了となってもよい。
- 出力関数の作成にあたって付録 B. を参考にしてもよい。独自に作成してもよいが、正しい出力形式で出力すること。また、時間切れで強制終了した場合でも、時間内に出力したものが確実に出力されるように注意すること（出力の flush を忘れずにおこなうこと)。

[1] コンパイル時に Warning などが出た場合でも、実行ファイルが作られ、実行可能な場合には（特段ペナルティを課することなく）審査を行う。

3. 級認定問題

下記の各問題を解くプログラムを作成する。

スーパーコン３級問題　第 1. 節「問題説明」の問題を解くプログラムを作成する。入力は先述の制約に加えて以下の制約を満たす：

- $N = 2$

スーパーコン２級問題　第 1. 節の問題を解くプログラムを作成する。入力は先述の制約に加えて以下の制約を満たす：

- $N \leq 50$

スーパーコン１級問題　第 1. 節の問題を解くプログラムを作成する。入力に関する追加の制約はない。

審査方法

各級毎に用意されたいくつかの問題例に対して各 10 秒間実行し、すべてで正解を出せば合格とする。予選問題と同じ形式で出力するが、級認定審査では「搭載する命令の個数（k）」は関係ない。

実行環境

予選問題と同じ実行環境で審査する。

付録 A. 予選問題の入力生成器

予選では、以下の入力生成器プログラム（generator.c）から 4 行目の疑似乱数のシード値（SEED）を「ある値」に変更したプログラムによって生成された 10 個の問題例（qual_random_01.in, ..., qual_random_10.in）を用いて審査する。

```
#include <stdio.h>
#include <stdlib.h>
// 予選審査ではSEED の値を別の「ある値」に変更する。
#define SEED 0x0000000000000003ull

#define N_MIN 2
#define N_MAX 250
#define M_MIN 1
#define M_MAX 2000

unsigned long long xor_shift() {
```

```
  static unsigned long long x = SEED;
  x                           = x ^ (x << 13);
  x                           = x ^ (x >> 7);
  x                           = x ^ (x << 17);
  return x;
}

// [0, x)
unsigned long long rnd(unsigned long long x) { return xor_shift() % x; }

// [l, r]
unsigned long long range_rnd(unsigned long long l, unsigned long long r)
    { return l + rnd(r - l + 1); }

void swap(int *x, int *y) {
  int t;
  t  = *x;
  *x = *y;
  *y = t;
}

int main() {
  for (int t = 0; t < 10; t++) {
    char file_name[64];
    sprintf(file_name, "qual_random_%02d.in", t + 1);

    int N = range_rnd(N_MIN, N_MAX);

    int C   = N * N - 1;
    int *ys = (int *)malloc(C * sizeof(int));
    int *xs = (int *)malloc(C * sizeof(int));
    {
      int i = 0;
      for (int y = 1; y <= N; y++) {
        for (int x = 1; x <= N; x++) {
          if (y == 1 && x == 1) continue;
          ys[i] = y;
          xs[i] = x;
          i++;
        }
      }
    }
    // The Fisher − Yates shuffle
    for (int i = 0; i < C - 1; i++) {
      int j = range_rnd(i, C - 1);
      swap(&ys[i], &ys[j]);
      swap(&xs[i], &xs[j]);
    }

    int M   = range_rnd(M_MIN, M_MAX);
    int L   = M * N;
    int *ls = (int *)calloc(M, sizeof(int));
    for (int i = 0; i < L; i++) {
      int pos = i < M ? i : range_rnd(0, M - 1);
      ls[pos]++;
    }
    char **ss = (char **)malloc(M * sizeof(char *));
    for (int i = 0; i < M; i++) {
      ss[i] = (char *)malloc((ls[i] + 1) * sizeof(char));
```

```
      int l = 0;
      while (l < ls[i]) {
        int r = range_rnd(0, 3);
        char c;
        if (r == 0) c = 'R';
        if (r == 1) c = 'D';
        if (r == 2) c = 'L';
        if (r == 3) c = 'U';
        int len = range_rnd(1, ls[i] - l);
        for (int j = l; j < l + len; j++) ss[i][j] = c;
        l = l + len;
      }
      ss[i][ls[i]] = '\0';
    }

    {
      FILE *fp = fopen(file_name, "w");
      if (fp == NULL) return 1;
      fprintf(fp, "%d\n", N);
      for (int i = 0; i < N; i++) fprintf(fp, "%d %d\n", ys[i], xs[i]);
      fprintf(fp, "%d\n", M);
      for (int i = 0; i < M; i++) fprintf(fp, "%s\n", ss[i]);
      fclose(fp);
    }

    free(ls);
    for (int i = 0; i < M; i++) free(ss[i]);
    free(ss);
    free(xs);
    free(ys);
  }
  return 0;
}
```

付録 B. 出力の方法、時間計測の方法

プログラムの作成にあたって、以下のプログラム（template.c）を用いてもよい。関数 output は解（の候補）を出力するための関数、関数 get_elapsed_time は実行経過時間を計測するための関数である。

```
#include <stdio.h>
#include <sys/time.h>

/**
 * 答えが YES のとき yn = 1、NO のとき yn = 0 で呼び出す。
 * 整数 k は搭載する命令の個数を表す。
 * vs は搭載する命令の情報を持った長さ k のint 型配列。
 */
void output(int yn, int k, int *vs) {
  if (!yn) {
    printf("NO\n");
  } else {
    printf("YES\n");
    printf("%d\n", k);
    for (int j = 0; j < k; j++) {
```

```
      if (j != 0) printf(" ");
      printf("%d", vs[j]);
    }
    printf("\n");
  }
  fflush(stdout);
}

double get_elapsed_time(struct timeval *begin, struct timeval *end) {
  return (end->tv_sec - begin->tv_sec) * 1000 + (end->tv_usec - begin->
    tv_usec) / 1000.0;
}

int main() {
  struct timeval t1, t2;
  gettimeofday(&t1, NULL);

  /**
  * ここにプログラム (*)を書く
  */

  gettimeofday(&t2, NULL);
  double t = get_elapsed_time(&t1, &t2);
  // t にプログラム (*) の部分の実行時間(ミリ秒)が代入される
  return 0;
}
```