令和５年度

千葉大学先進科学プログラム入学者選考課題

課題論述

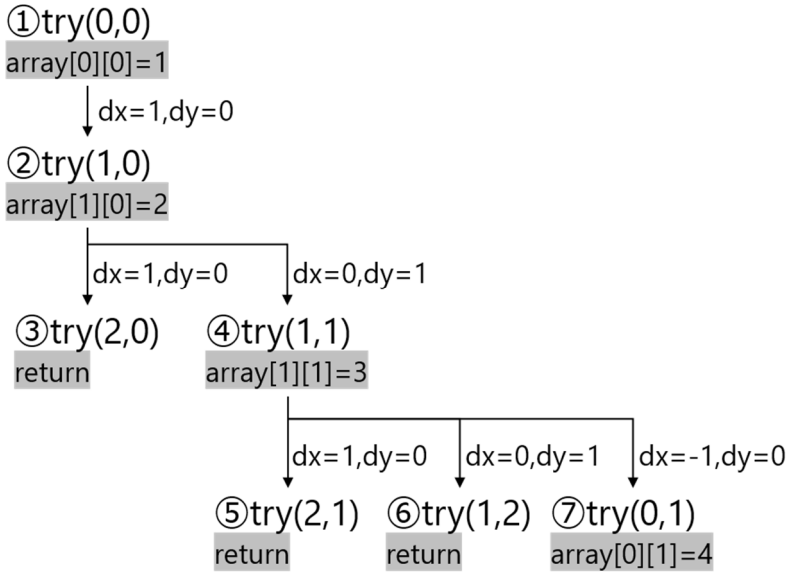
情報

解答例

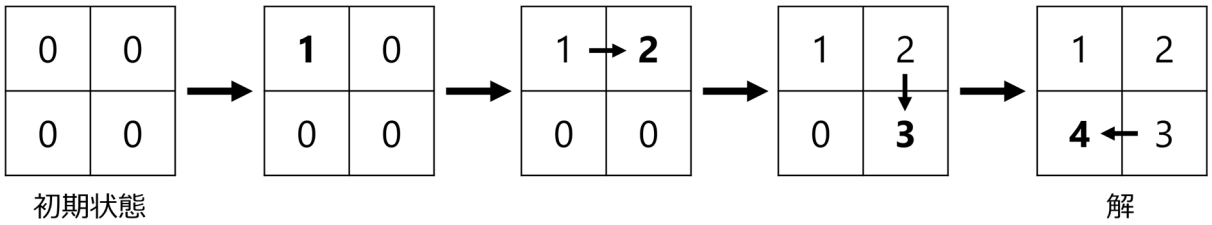
1

（出題意図）問題のアルゴリズムは，再帰関数呼び出しで実装したバックトラック法になっている．バックトラック法の動作や探索方法についての理解を問うている。

問 1 `try` 関数のなかで `try` 関数が呼び出しており，再帰関数呼び出しを使っている。`main` 関数で実行される `try(x=0,y=0)` を起点に，この再帰関数呼び出しの流れを追うと次のようになる。



すなわち，2次元配列 `array` の要素には，次のように `count` 値が順次代入されていく。



最終的に，2 次元配列 `array` の各要素には次の値が代入される。

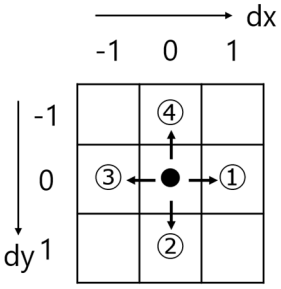
```
array[0][0] = 1
array[1][0] = 2
array[1][1] = 3
array[0][1] = 4
```

したがって，標準出力は次のようになる。

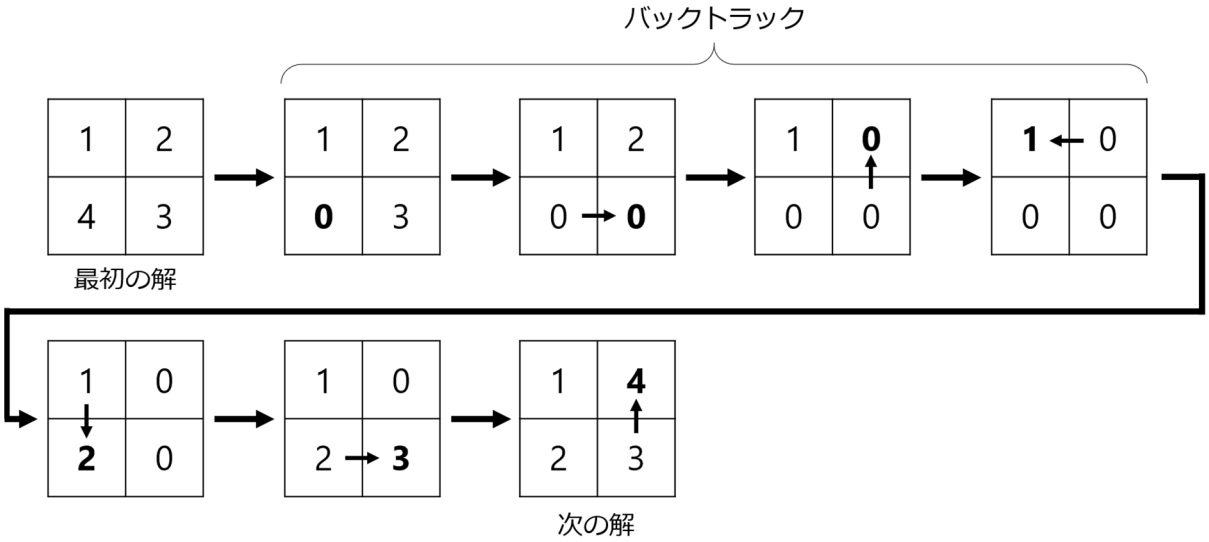
```
1  2
4  3
```

問2 問題で与えた2行を追加することで，このプログラムはバックトラック法になる。一般に，バックトラック法は，与えられた制約のなかで解を探索するアルゴリズムで，探索がそれ以上できない，あるいは解を見つけたときに，これまでの探索経路をバックし，別の解の探索を始める。このように探索経路を戻りながら探索することで，系統的にすべての解を見つけることができる。

このプログラムは，下図で示すように上下左右の4方向を①，②，③，④の順に探索しながら，2次元配列の0でない要素に `count` の値を代入していく。



問1の解を見つけたあとは，`array[y][x]=0;  count--;` で探索経路をバックしながら，別の解を探索する。探索過程は次の図のようになる。



したがって，これら2つの解が標準出力される。

```
1  2
4  3
1  4
2  3
```

問３ 壁は2次元配列 wall で対応する位置に１を代入することで表現してある。したがって，壁があるところへ移動できないようにするためには，wall[y][x]== 1 のとき，探索をやめて return すればよい。さらに，ゴール(G)への到達したとき x==gx && y==gy，解が見つかったと判定してするように変更すればよい。

```
...(省略)
void try(int x, int y)
{
    int i;

    if (array[y][x]!=0 || wall[y][x] == 1)
        return;
    if (x<0 || x>=N || y<0 || y>=N)
        return;

    count++;
    array[y][x] = count;

    if (x == gx && y == gy){
        print_array();
    }
    else{
        for(i = 0; i < D; i++)
            try(x+dx[i], y+dy[i]);
    }

    array[y][x] = 0;
    count--;
}
int main(void)
{
    init_array();
    try(sx, sy);   /* 変更:スタート(s)位置 */

    return 0;
}
```

問４ 変数 count の値は，スタート(S)から現在地までの経路長を表している。したがって，ゴール(G)に到達したときの count 値は，ゴール(G)までの経路長になっている。ゴール(G)までの最小経路長(min_count)を保持しておき，現在地までの経路長が最小経路長より短いときに限り，探索を続け，そうでないときは探索を打ち切るようにする。このようにすることで，無駄な探索(最短経路長よりも長くなる経路の探索)をしなくなる。print_array 関数の最後の出力が最短経路になっている。

```
...(省略)

int min_count = 999; /* 変更:経路長の最小値(グローバル変数で宣言) */

...(省略)
void try(int x, int y)
{
    int i;

    if (array[y][x]!= 0 || wall[y][x] == 1)
        return;
    if (x<0 || x>=N || y<0 || y>=N)
        return;

    count++;
    array[y][x] = count;

    if (x == gx && y == gy){
        min_count = count; /* 変更:最短経路長の更新 */
        print_array();
    }
    else if (count < min_count){ /* 変更:最短経路長より短いときに探索を継続*/
        for(i = 0; i < D; i++)
            try(x+dx[i], y+dy[i]);
    }

    array[y][x] = 0;
    count--;

}
```

2

（出題意図）解析的に求めることが難しい数（円周率，積分，組み合わせ数）を，モンテカルロ法の初歩的な考え方を用いて求めるプログラムの理解を問う。

問 1－1　提示したプログラムのなかの（14）行目と（17）行目で，半径 1 の円の内部の点を数えていることを理解していれば，近づく値は，円周率であることがわかる。

問1－2　初等数学の範囲では，解析的に求めることができない積分の値をモンテカルロ法の考え方を使えば，近似値を求めることができることを問う。問 1－1の考え方が理解できていれば，（14）行目の条件として，灰色部分を規定する条件に書き換えればよいことに気づくはずである。

```
(x>=0.2) && (x<=0.8) && (y>=0) && (y<=-2*log(x)*sin(x))
```

問2－1　円順列をランダムに作成する方法として，最初に，整列化した円順列を与え（配列の初期化），その後，ランダムに，数字の入れ替えを行う。その入れ替えの方法を問うている。

```
int x=rand()%N, y=rand()%N; /* 入れ替え候補 2 つを乱数で選ぶ */
while(x==y) y=rand()%N; /* 同じ数字が選ばれたら，一方を選び直し */
int tmp=c[x]; c[x]=c[y]; c[y]=tmp; /*数字の入れ替え */
```

問2－2　プログラムの全体像を示した後，隣り合っているときは，条件に合っている数を表す変数 count_in の値を減らす。隣り合っているかどうかの判定の条件を尋ねている。

```
(c[k]==a[j] && c[(k+1)%N]==b[j]) || (c[k]==b[j]&&c[(k+1)%N]==a[j])
```

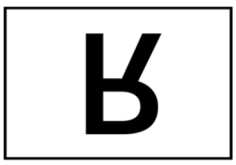
3

(出題意図)画像処理を題材として，アルゴリズムを実現する能力と順列・組合せについての知識を問う。

問 1 256階調は8ビット，すなわち，1バイトで表現される。したがって，1画素あたり1バイトの情報量であり， 画素数をかけることで求められる。

1 バイト/画素×64 画素×32 画素 ＝ 2,048 バイト

問2 上下反転すると生成される画像は以下の通りである。



したがって，(22)行目は次のように書き換えられる。

```
image[p2][x][y] = image[p1][x][height[p1]-1-y];
```

問3 階調反転すると生成される画像は以下の通りである。

したがって，(22)行目は次のように書き換えられる。

```
image[p2][x][y] = 255 - image[p1][x][y];
```

問4 右回りに90度回転すると生成される画像は問題文と同様，以下の通りである。

したがって，(18)，(19)，(22)行目はそれぞれ次のように書き換えられる。

(18) `width[p2]= height[p1];`

(19) `height[p2]= width[p1];`

(22) `image[p2][width[p2]-1-y][x]=image[p1][x][y];`

問5 上下/左右反転は4パターン，階調反転は2パターン，回転は4パターンである。ただし，上下/左右反転と回転の組合せにおいて，同じパターンが二つずつ生成される。したがって，

4×2×4÷2= 16 種類

問6 ブロック数は 8 個となる。ブロックの位置入替えのパターンは，8 ブロックの順列として求めることができる。したがって，

${}_8P_8$ ＝ 40,320 種類

4

（出題意図）本問の目的は 2 次元図形を描画するための幾何学に関する理解の確認である。

問1

```
canvas2 = [
    [1, 0, 1, 1],
    [0, 1, 0, 0],
    [1, 1, 1, 0],
    [1, 1, 0, 0],
]
```

問2 縦方向と横方向との画素位置の和が偶数か奇数かで白・黒を判断する。

```
def checker(width, height):

    # 幅 width 画素，高さ height 画素でキャンバスを初期化
    canvas = [[0] * width for i in [0] * height]

    for h in range(height):
        for w in range(width):
            if (w+h)%2 == 1: canvas[h][w] = 1

    return canvas
```

問3 注目する画素と円の中心までの距離を指標に白・黒を判断する

```
def circle(width, height, cx, cy, r):

    # 幅 width 画素，高さ height 画素でキャンバスを初期化
    canvas = [[0] * width for i in [0] * height]

    for h in range(height):
        for w in range(width):
            if (w-cx)*(w-cx)+(h-cy)*(h-cy) <= r*r:
                canvas[h][w] = 1

    return canvas
```

問4 注目する画素から円の中心までの距離を指標にして，距離が内円の半径より遠く，かつ外円の半径より近ければ白色にする。

```
def ring(width, height, cx, cy, r, t):

    # 幅 width 画素，高さ height 画素でキャンバスを初期化
    canvas = [[0] * width for i in [0] * height]

    for h in range(height):
        for w in range(width):
            d_sq = (w-cx)*(w-cx)+(h-cy)*(h-cy)
            r_inner_sq = (r-0.5*t)*(r-0.5*t)
            r_outer_sq = (r+0.5*t)*(r+0.5*t)
            if (d_sq>=r_inner_sq) and (d_sq<=r_outer_sq):
                canvas[h][w] = 1

    return canvas
```

問5 各画素から線分に下ろした垂線が線分の中にあるかどうかと，各画素と線分との距離が線の太さの範囲内になっているかどうかで判断する。ベクトルを使って計算すると比較的簡単に求まる。

```
def line(width, height, xa, ya, xb, yb, t):

    # キャンバス（２次元配列の初期化）
    canvas = [[0] * width for i in [0] * height]

    for h in range(height):
        for w in range(width):
            # 点 A = (xa, ya)，点 B = (xb, yb)，点 Q = (w, h) とする
            A = [xa, ya]
            B = [xb, yb]
            Q = [w, h]

            # ベクトル AB，AQ を次の様に定める
            # ベクトル AB = B - A
            # ベクトル AQ = Q - A
            AB = [B[0] - A[0], B[1] - A[1]]
            AQ = [Q[0] - A[0], Q[1] - A[1]]

            # 点 Q から直線に下ろした垂線との交点 H における媒介変数 p は
            # ベクトル AB とベクトル AQ の内積を
            # ベクトル AB の長さの 2 乗で割り算して求める
            p = (AB[0]*AQ[0]+ AB[1]*AQ[1])/(AB[0]*AB[0]+AB[1]*AB[1])
            # このとき点 A から点 H までのベクトル AH はベクトル AB の p 倍である
            AH = [p * AB[0], p * AB[1]]
            # さらに点 H から点 Q までのベクトル HQ は
            # HQ = -AH + AQ ゆえ
            HQ = [-AH[0] + AQ[0], -AH[1] + AQ[1]]

            # 従って点 H から点 Q までの距離 d はベクトル HQ の長さでありその 2 乗は
            d_sq = HQ[0]*HQ[0] + HQ[1]*HQ[1]

            # 媒介変数が[0, 1]の範囲内（線分からはみ出さない）で，
            # かつ線分との距離の 2 乗が線幅の半分の 2 乗以下の場合，白くする
            if (0 <= p) and (p <= 1) and (d_sq <= (0.5*t)*(0.5*t)):
                canvas[h][w] = 1

    return canvas
```