令和４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）ゲーム木探索のミニマックス法を題材として、アルゴリズムを考え、再帰的なプログラムで実現する能力を問う。

問1

コンピュータ 1 回目	人間 1 回目	コンピュータ 2 回目	コンピュータの勝敗
1	1	3	勝ち
	2	2	勝ち
	3	1	勝ち
2	1	2	勝ち
	2	1	勝ち
	3		負け
3	1	1	勝ち
	2		負け

問 2

k	c(k)の戻り値	k	h(k)の戻り値
N-1	-1	N-1	1
N-2	1	N-2	-1
N-3	1	N-3	-1
N-4	1	N-4	-1

問 3

```
int c(int k)
{
  int i;

  if ((k >= N) || (k < 0)) {
    return 0;
  }

  if (k >= N-4) {
    if (k == N-1) {
      return -1;
    } else {
      return 1;
    }
  }

  for (i = 1; i <= 3; i++) {
    if (h(k+i) == 1) {
      return 1;
    }
  }
  return -1;
}
```

```
int h(int k)
{
  int i;

  if ((k >= N) || (k < 0)) {
    return 0;
  }

  if (k >= N-4) {
    if (k == N-1) {
      return 1;
    } else {
      return -1;
    }
  }

  for (i = 1; i <= 3; i++) {
    if (h(k+i) == 1) {
      return -1;
    }
  }
  return 1;
}
```

2

問 1　再帰関数呼び出しと漸化式の関係を問う課題である. 位置(i,j)のマス目までの経路数を route[i][j]とする。まず S である位置(0,0)までの経路は 1 つしかない。すなわち, route[0][0]=1 である。次に, 最上行の位置(0,j),j=1,2,…,W-1 については, 左隣(0,j-1)からの経路しかないため, 経路数は route[0][j]= route[0][j-1]と再帰的に与えられる。最左列の位置(i,0),i=1,2,…,H-1 も同様に, 経路数は route[i][0]=route[i-1][0]と与えられる。上記以外の経路数は, route[i][j]= route [i][j-1]+ route[i-1][j]と再帰的に与えられる。なぜならば, 位置(i,j)のマス目に到達するには, 左隣(i,j-1)あるいは上隣(i-1,j)のマス目からの経路しかなく, 位置(i,j)までの経路の総数は, それらの和になるからである。

問 2 繰り返し文を用いて, 経路数 route[i][j]を, (i,j)=(0,0),(0,1),…,(1,0),(1,1),…,(H-1,W-1)の順に計算していく。効率的な計算のための代表的なアルゴリズムの一つである, 動的計画法の例になっている。

```
#include <stdio.h>

#define H 4
#define W 7

int main(void) {

  int route[H][W]={0};

  int i, j;
  route[0][0] = 1;
  for(i=1; i<H; i++)
    route[i][0] = route[i-1][0];
  for(j=1; j<W; j++)
    route[0][j] = route[0][j-1];

  for(i=1; i<H; i++)
    for(j=1; j<W; j++)
        route[i][j] = route[i][j-1] + route[i-1][j];

  printf("経路の数:%d¥n", route[H-1][W-1]);
  return 0;
}
```

問3 問 1 の再帰関数呼び出しを用いたプログラムでは, 同じ引数に対する関数計算を何度も重複して実行しているため遅い。一方, 動的計画法を用いたプログラムでは, 経路数 route[i][j]を順に一度だけ計算するだけであるから, 重複がなく, 高速になる。

問４　あるマス目の通行止めが左隣にあるとき，そのマス目に到達するまでの経路数は，上隣のマス目までの経路数と同じである。なぜならば，左隣が通行止めであるから，上隣から来るしかないからである。すなわち，位置(i,j)のマス目までの経路の数 route[i][j]は，route[i][j]=route[i-1][j]と再帰的に求めることができる。同様に，上隣が通行止めであれば，route[i][j]=route[i][j-1]と求めることができる。左隣や上隣に通行止めがなければ，問１や問２で考えたように，経路の数は，route[i][j]=route[i-1][j]+route[i][j-1]と求めることができる。通行止めのマス目の route[i][j]の値を-1 とすると，上の３つの再帰式はまとめて次のように書くことができる。ただし，max(x,y)は引数x,yのうち，大きい方を返す関数とする。

route[i][j]=max(route[i-1][j],0)+max(route[i][j-1],0)

```
#include <stdio.h>

#define H 4
#define W 7

int max(int x, int y){

  if(x>y)
    return x;
  else
    return y;
}

int main(void) {

  int route[H][W]={0};
  int i, j;

  route[1][2] = -1; /* 通行止めを-1 で表現 */
  route[2][4] = -1;

  route[0][0] = 1;
  for(i=1; i<H; i++)
    route[i][0] = route[i-1][0];
  for(j=1; j<W; j++)
    route[0][j] = route[0][j-1];

  for(i=1; i<H; i++)
    for(j=1; j<W; j++)
       if(route[i][j]!=-1)
        route[i][j] = max(route[i][j-1],0) + max(route[i-1][j],0);

  printf("経路の数:%d¥n", route[H-1][W-1]);
  return 0;
```

3

出題の意図

本問は，比較的単純な統計的手法を理解できる数理的読解力，およびそれを実装し計算コストを見積もる実装力を問う問題である。大学で理系分野を学ぶために数理的読解力は言うまでもなく重要な能力であり，それを試すにあたり昨今の情報科学で進展めざましい機械学習に関連するトピックを採用した。また数理的な理解を実際にプログラムとして実現する力（実装力）も重要である。今回は筆記での試験であるので複雑な実装は求めなかった。説明された手法が正確に読解できていれば即座に実装できると思われる。問 3，問 4 は計算問題である。問 3 は考えている確率変数が 1 または-1 しか取らないので，素直に全通り考えることができる。期待値の定義がわかっていれば解ける問題である。問 4 は多少の計算が必要になる。問 3 の結果がうまく活用できるかが鍵となる。

問 1

```
def sim(a, b):
  s = 0
  n = len(a)
  for i in range(n):
    s += a[i]*b[i]

  return s

def nearest_neighbor_id(y, X):
  k, s, m = -1, -1, len(X)
  for i in range(m):
    s_ = sim(y, X[i])
    if i==0 or s_ > s:
      s = s_
      k = i
```

問 2

計算コストは，$2mn$。二つのベクトルの内積を計算するのがコストn（要素同士の積が 1，sへの加算で 1，それをn回繰り返し$2n$）。最近傍データを見つけるために，この内積計算をm回繰り返すので，結局$2mn$。

問 3

$$\mathbb{E}[w_{i,j}] = 0, \mathbb{E}[w_{i,j}^2] = 1, \mathbb{E}[w_{i,j}w_{i,k}] = 0$$

問４

$$\mathbb{E}[\vec{z}_1 \cdot \vec{z}_2] = \frac{1}{d}\sum_{j=1}^{d} \mathbb{E}\left[(\vec{u}_1 \cdot \vec{w}_j)(\vec{u}_2 \cdot \vec{w}_j)\right] = \frac{1}{d}\sum_{j=1}^{d}\sum_{k=1}^{n}\sum_{l=1}^{n} \mathbb{E}\left[u_{1,k} w_{j,k} u_{2,l} w_{j,l}\right]$$

$$= \frac{1}{d}\sum_{j=1}^{d}\sum_{k=1}^{n} \mathbb{E}\left[u_{1,k} u_{2,k} w_{j,k}^2\right] \quad = \frac{1}{d}\sum_{j=1}^{d}\sum_{k=1}^{n} u_{1,k}\, u_{2,k} = \vec{u}_1 \cdot \vec{u}_2$$

問５

```
def reduction(y, W):
  v = []
  d = len(W)
  for i in range(d):
    s = sim(y, W[i])
    s /= d**0.5        # これはなくても良い
    v.append(s)
  return v

def approximate_nearest_neighbor_id(y, X, Z, W):
  v = reduction(y, W)
  k = nearest_neighbor_id(v, Z)
  return k
```

問６

計算コストは，$2d(m+n+1)$。reduction()ではn次元ベクトルに対し sim()をd回繰り返すので，$(2n+2)d$（平方根・除算の計算で+2 される）。問 2 の結果から最近傍探索で，$2md$。合計して整理すると，$2d(m+n+1)$。ただし問５で，`s /= d**0.5` を書かなかった場合は，計算コストは$2d(m+n)$となる。

4

本問の目的は論理演算への理解の確認である。

問 1 関数 d は引数を 2 進数で表したときの各桁の値を下位桁から順に要素として持つリストを返す。関数 e では引数と戻り値を逆にし、リストから数値を返すようにすればいい。

```
def e(b):
	x=0
	while b:
		x = (x << 1) | b.pop()
	return x
```

別解:python 本来の 2 進数を扱う機能を使ってもよい。

```
def e(b):
	return int(''.join((str(_) for _ in reversed(b))), 2)
```

問 2 関数 t は第 1 引数をしきい値とし、第 2 引数に与えられるリストの内、1 である要素の数を数えるしきい値関数である。ここで、第 2 引数のリストに含まれる要素は 0 と 1 に限る。2 入力論理積はしきい値 2 のしきい値関数である。

```
def c(x,y):
	return t(2,[x,y])
```

問 3 半加算器を作ればいい。関数 c は論理積であり、関数 v は否定であることから、これらを組み合わせれば半加算器を作ることができる。論理和 t(1,[x,y])も使える。別解は多い。

```
def a(x,y):
	return e([c(t(1,[x,y]), v(c(x,y))), c(x,y)])
```

問 4 乗算のアルゴリズムを理解していることと、非常に制限された記述法の中でリストの例外対策が行えるか、が問われる。後者は重視しない。別解は多い。本問が解ければ優秀である。

2 進数 2 ビット値 (x_1x_0), (y_1y_0) の積は次式の通り:

$(2x_1+x_0)(2y_1+y_0)=4x_1y_1+2(x_1y_0+y_1x_0)+y_0x_0$

よって、各桁の値は次式で得られる。4 の位、2 の位では桁上がりが発生することに注意。

1 の位: y_0x_0

2 の位: $(x_1y_0$ xor $y_1x_0)$; 4 の位への桁上がり c は $(x_1y_0$ and $y_1x_0)$

4 の位: $(x_1y_1$ xor c); 8 の位への桁上がりは $(x_1y_1$ and c)

8 の位: $(x_1y_1$ and c)

```
def m(x,y):
	lx=d(x)
	lx.extend([0,0])
	x0=lx.pop(0)
	x1=lx.pop(0)
	ly=d(y)
	y0=ly.pop(0) if ly else 0
	y1=ly.pop(0) if ly else 0
	r = [c(x0,y0)]
	lz = d(a(c(x1,y0), c(x0,y1)))
	lz.extend([0,0])
	r.append(lz.pop(0))
	r.extend(d(a(c(x1,y1), lz.pop(0))))
	return e(r)
```