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

（9:00－12:00）

注意事項

1. この冊子は，監督者から解答を始めるよう合図があるまで開いてはいけません。
2. 問題冊子に印刷または製本の不具合がある場合は，手を上げて申し出てください。
3. 問題すべてに解答してください。
4. 解答用紙は，課題ごとに解答用紙を分けて使用してください。解答用紙は何枚使用してもかまいません。すべての解答用紙に受験番号を必ず記入してください。
5. 検査室に用意してある資料は自由に使用してかまいません。ただし，諸君が持参した教科書，参考書，ノート，パソコンなどの使用は禁止します。
6. 携帯電話やスマートフォン等の電子機器はすべて電源を切り，カバンにしまってください。
7. その他，監督者の指示に従ってください。

1

1 からＮまでの自然数を小さい順に二人で交互に数え上げ(一度に 1 つから 3 つの自然数を数え上げることができる), 最後に自然数 Ｎ を数え上げた人が負けというゲームについて, 人間とコンピュータの対戦を考える。

例として, N=10 でコンピュータが先に数え上げを開始する場合,

コンピュータ:1,2,3（3 つの自然数を数え上げた）

人間:4,5（2 つの自然数を数え上げた）

コンピュータ:6（1 つの自然数を数え上げた）

人間:7,8,9（3 つの自然数を数え上げた）

コンピュータ:10（自然数 N=10 を数え上げた）

と数え上げた結果、コンピュータの負けとなる。

問1 N=6 でコンピュータが先手の場合, コンピュータが 1 回目に数え上げる自然数の個数 1 から 3, ならびに, その後, 人間が 1 回目に数え上げる自然数の個数 1 から 3 に対して, コンピュータが 2 回目の数え上げの後, コンピュータが必ず勝つことができるようにするためにコンピュータが 2 回目に数え上げるべき自然数の個数を指定された解答用紙の表中の空欄に記述しなさい。なお, 表中の数字はコンピュータ, 人間が一度に数え上げる自然数の個数を表している。また, 人間が自然数 5 を数え上げ, そこで数え上げを終了した場合, コンピュータは 2 回目に自然数 6 を数え上げなくてはならず, コンピュータの負けが確定してしまうため, そのような場合は表中のコンピュータの 2 回目の欄には斜線が引かれ, コンピュータの勝敗は「負け」と記述されている。

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		勝ち
	2		勝ち
	3		勝ち
2	1		勝ち
	2		勝ち
	3		負け
3	1		勝ち
	2		負け

以後の設問では, 人間は可能な限り, 自分自身が勝つための最善の選択を行うものとする。その場合, 上記 N=6 の例でコンピュータが必ず勝てるようにするためには, 1回目の選択で数え上げる自然数の数を 1 個としなければならないことがわかる。

問2　このように人間とコンピュータが可能な限り，自分自身が勝つための最善の選択を行うものと仮定した上で，以下の仕様を満たす関数 c(k)と h(k)を作成する。

関数 c(k) は次がコンピュータの順番の局面において，直前に人間が最後に数え上げた自然数を引数 k(int 型)(0≦k≦N-1)とし，戻り値(int 型)はコンピュータが最終的に勝つ場合に 1，負ける場合に -1 を返すものとする。

関数 h(k) は次が人間の順番の局面において，直前にコンピュータが最後に数え上げた自然数を引数 k(int 型)(0≦k≦N-1)とし，戻り値(int 型)は人間が最終的に勝つ場合に -1，負ける場合に 1 を返すものとする。

なお，両関数において，k=0 をゲーム開始の初期値とし，0≦k≦N-1 以外の k が入力された場合の戻り値は 0 とする。

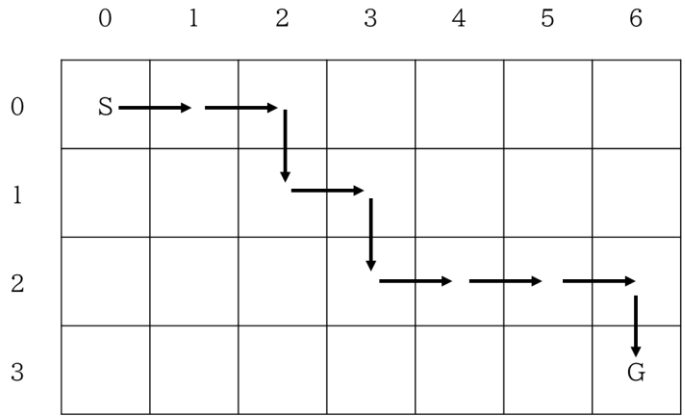
ゲーム終盤における各関数の戻り値を，指定された解答用紙の表中の空欄に記述しなさい。

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

問3　上記以外の状況(0≦k＜N-4)では，関数 c(k) 内では関数 h(k) を呼び出し，関数 h(k) 内では関数 c(k) を呼び出すことにより，相互再帰的にお互いの手順を予想し，戻り値を計算可能である。上記の仕様を満たす C 言語の関数 c(k) と h(k) を解答用紙に記述しなさい。ただし，N は 5 以上とし，int 型の大域変数(グローバル変数)として，すでに値が代入されているものとする。

2

下図のような碁盤の目状の街で，左上の S（スタート地点）から右下の G（ゴール地点）まで移動することを考える。ただし，右あるいは下のマス目に1マスずつしか移動できないとする。



問１ S から G までのすべての経路の数を求めるために，以下の C 言語プログラムを作成した。このプログラム中では，H を縦方向のマス目の数，W を横方向のマス目の数として定義している。なぜ，このプログラムで，H×W の碁盤のすべての経路の数を求めることができるのか答えなさい。

```
#include <stdio.h>
#define H 4
#define W 7

int route(int i, int j){
  int n;

  if(i==0 && j==0)
    n = 1;
  else if(i==0)
    n = route(i, j-1);
  else if(j==0)
    n = route(i-1, j);
  else
    n = route(i, j-1)+route(i-1, j);
  return n;
}

int main(void) {
  printf("経路の数:%d¥n", route(H-1,W-1));
  return 0;
}
```

問２　問１のプログラムでは，再帰関数呼び出しを用いている。再帰関数呼び出しを用いずに，問１のプログラムを，繰り返し文を用いて書き直すことにする。以下のプログラムの[空欄]に入る部分を記述し，プログラムを完成させなさい。ただし，プログラム中の配列 route[i][j]には，位置(i,j)までのすべての経路の数が格納されるようにしなさい。なお，位置(i,j)は，H×W の碁盤の i 行 j 列目を表している。

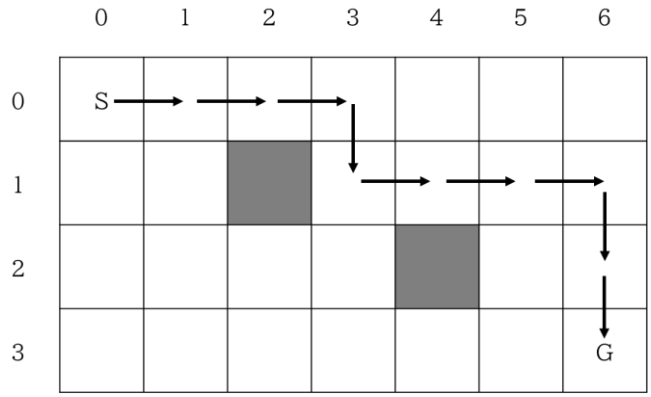
```
#include <stdio.h>

#define H 4
#define W 7

int main(void) {

  int route[H][W]={0}; /*0 で初期化される*/

  [空欄]

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

問３　問１と問２のプログラムについて，どちらがより速く経路の数を求めることができるか，その理由を含めて解答しなさい。

問４　いくつかのマス目が通行止めになってしまった。そのため，例えば，下図のように，通行止めのマス目（灰色）を避けながら，S から G へ行かなければならない。このとき，S から G までのすべての経路の数を求めるプログラムを問 2 のプログラムを修正して記述しなさい。ただし，通行止めのマス目の位置は，配列 route[i][j]に-1 を代入して指定しなさい。下図の例では，通行止めの位置は，(1,2)と(2,4)にあるから，プログラム中では次のように指定することになる。

```
route[1][2] = -1;
route[2][4] = -1;
```

なお，問 1 と同様に，右あるいは下のマス目にしか移動できないとし，最上行と最左列には通行止めはないとする。



3

m個のデータの集合$X = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_m\}$ が与えられたとする。$\vec{x}_i$はi番目のデータに対応し，n次元の実数ベクトル$\vec{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,n})$である（各要素$x_{i,j}$は実数）。

問1　あるn次元の実数ベクトル$\vec{y} = (y_1, y_2, \dots, y_n)$が与えられた時，$X$における$\vec{y}$の最近傍データ，すなわち$X$に含まれるデータのうち$\vec{y}$と最も類似度が高いデータ$\vec{x}_k$を見つけ，そのデータのインデックス$k$を知りたい。そのような関数 nearest_neighbor_id()を記述しなさい。

注意

- このとき，二つのn次元ベクトル$\vec{a} = (a_1, a_2, \dots, a_n), \vec{b} = (b_1, b_2, \dots, b_n)$の類似度$s$は次で定義される内積で計算されることとする（簡単のためベクトルの正規化は考えない）。

$$s = \vec{a} \cdot \vec{b} = \sum_{j=1}^{n} a_j \, b_j$$

  内積が大きいほど，類似度が高いと考える。
- 以下のテンプレートを埋める形で解答しなさい。Python のコードで記述すること。ただし，import と sum()と pow()を使用してはならない。
- ベクトル$\vec{a}$は Python においてリストで実装できる。また，あるリスト a の長さは len(a)で知ることができる。

```
def sim(a, b):  # 内積を計算する関数
  # a, b は同じ長さのリスト
  # --- ここを埋める --------------




  # ----------------------------
  return s

def nearest_neighbor_id(y, X):
  # y は長さ n のリスト
  # X[i] で i 番目のデータベクトル(長さ n のリスト)xi にアクセスできる。
  # --- ここを埋める --------------




  # ----------------------------
  return k
```

問 2　問 1 で$(\vec{y}, X)$に対して探索してきた$\vec{x}_k$を，「(Xにおける)$\vec{y}$の最近傍データ」と呼ぶことにする。問 1 の nearest_neighbor_id()によって$\vec{y}$の最近傍データのインデックスを探索するコストを，m, nを用いて表現しなさい。その理由も説明しなさい。

注意

- ここでコストとは，実数の加減乗除および冪乗，つまり「+」「-」「*」「/」「**」を実行する回数で定義される。例えば 1 + 3**4 のコストは 2 である。

データの次元nが非常に大きい場合に最近傍データを探索するコストを下げたい。その一つの戦略として，次に記述する次元削減を用いた近似的な最近傍探索が知られている。

次元削減を用いた近似的な最近傍探索

n次元ベクトル$\vec{x}$をより低次元のd次元ベクトル$\vec{z}$で近似的に表現することを考える。つまり，n次元データの集合$X = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_m\}$を適当な$d$次元データの集合$Z = \{\vec{z}_1, \vec{z}_2, \dots, \vec{z}_m\}$に置き換えることを考える。$X$における$\vec{y}$の最近傍データ(のインデックス)を探索したい時は，次のような戦略をとる。

1. $\vec{y}$も，あるd次元データ$\vec{v}$で表現する。
2. Zにおける$\vec{v}$の最近傍データのインデックスkを見つけ，それを返す。

さてこのとき，d次元データの集合Zは 2 値ランダムベクトルを用いて計算される。まず，d本のn次元ベクトル$\vec{w}_1, \vec{w}_2, \dots, \vec{w}_d$を準備する。ここで，$\vec{w}_i$の要素$w_{i,1}, w_{i,2}, \dots, w_{i,n}$はランダムに$1, -1$いずれかの値で定める。つまり，$\vec{w}_i = (1,1,-1,1,-1,-1,\cdots)$のようになっている。この時，$\vec{x}_i$に対応する$\vec{z}_i$は次のように定義することとする。

$$\vec{z}_i = \frac{1}{\sqrt{d}}(\vec{x}_i \cdot \vec{w}_1, \vec{x}_i \cdot \vec{w}_2, \dots, \vec{x}_i \cdot \vec{w}_d)$$

問 3　上で登場する各$w_{i,j}$は等確率かつ独立に 1 または-1 の値に設定される。この確率変数に関して，三つの期待値$\mathbb{E}[w_{i,j}], \mathbb{E}[w_{i,j}^2], \mathbb{E}[w_{i,j}w_{i,k}]$をそれぞれ計算しなさい。ただし、$j \neq k$する。

問 4　任意の二つのn次元ベクトル$\vec{u}_1 = (u_{1,1}, u_{1,2}, \dots, u_{1,n}), \vec{u}_2 = (u_{2,1}, u_{2,2}, \dots, u_{2,n})$を考える。この$\vec{u}_1, \vec{u}_2$に対し，上で定義したような$d$次元ベクトル$\vec{z}_1, \vec{z}_2$をそれぞれ定義する。つまり，$\vec{z}_i$の$j$要素目は$z_{i,j} = \vec{u}_i \cdot \vec{w}_j/\sqrt{d}$である。この時，次の関係が成立することを証明しなさい。

$$\vec{u}_1 \cdot \vec{u}_2 = \mathbb{E}[\vec{z}_1 \cdot \vec{z}_2]$$

注意

- ここでは期待値は，問 3 と同様，全ての確率変数$w_{i,j}$に関してとるものである。証明には問 3 の結果を活用すると良い。

問 4 の結果から，任意のn次元ベクトルの$\vec{u}_1, \vec{u}_2$の内積は対応する$\vec{z}_1, \vec{z}_2$の内積の期待値に一致することがわかった。もちろん期待値が一致するからといって， $\vec{z}_1 \cdot \vec{z}_2$が$\vec{u}_1 \cdot \vec{u}_2$の良い近似を常に与えるとは限らない。しかし，ある比較的現実的な条件の下で，非常に高い確率で十分良い近似を与えることが知られている。以降ではこの条件が満たされていることを仮定する．

問 5 問 1 と同様にXにおける$\vec{y}$の最近傍データのインデックスの探索を行いたい。上に述べた次元削減による近似的な最近傍探索を行う関数 approximate_nearest_neighbor_id()を記述しなさい。

注意

- 問 1 で設計した sim()，nearest_neighbor_id()を利用して良い。
- 正の平方根は**0.5 で計算できる。例えば$\sqrt{2}$は 2**0.5 で計算できる。
- リスト v に s を追加する操作は，v.append(s)で行うことができる。
- reduction(y, W)は，$\vec{y}$とd本のn次元ベクトルからなる集合$W = \{\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_d\}$を受け取り，$d$次元ベクトル$\vec{v}$を返す関数である。
- 以下のテンプレートを埋める形で解答しなさい。Python のコードで記述すること。ただし，import と sum()と pow()を使用してはならない。

```
def reduction(y, W):
  v = []
  # --- ここを埋める -------------




  # ----------------------------
  return v


def approximate_nearest_neighbor_id(y, X, Z, W):
  # X[i]で i 番目のデータベクトル(長さ n のリスト)xi にアクセスできる。
  # Z[i]で i 番目のデータベクトル(長さ d のリスト)zi にアクセスできる。
  # W[i]で i 番目の 2 値ランダムベクトル(長さ n のリスト)wi にアクセスできる。
  # --- ここを埋める -------------




  # ----------------------------
  return k
```

問 6 問 5 のプログラムの approximate_nearest_neighbor_id()によって近似的な最近傍探索を行うコストを，m, n, dを用いて表現しなさい。その理由も説明しなさい。

4

CPU や GPU などにおいては，論理演算を行う部品（論理ゲート）を組み合わせることで四則演算等を行う演算回路を構成している。本問では演算回路で行われている論理演算の一部を，Python を利用して再現することを考える。

問 1 任意の正整数値 x に対し e(d(x)) が x と等しくなるような関数 e(b)を記述しなさい。ただし，x が極端に大きい値である場合は考えなくてもよいものとする。

```
def d(x):     # 引数 x は正整数
       b = []
       while x != 0:
              b . append (x % 2)   # '%' は整数除算の余り
              x //= 2              # '//' は整数除算の商
       return b                    # 戻り値はリスト
```

以下の問いにおいては次に示すものの内，必要なものだけを用いてプログラムを記述しなさい。

- def 文
- return 文
- リストを扱うための角括弧（[]）およびコンマ（,）
- リストのメソッド（append, extend, pop など）
- リストのアンパック
- 代入演算子（=）
- 条件演算子（～ if ～ else ～）
- リテラル（0 などの数値，など）
- 前の問いまでに示した関数（例えば問 2 では問 1 に示した d, e, 問 3 では d, e に加え，問 2 で示した関数 c）
- 次に示す関数 t, v

```
def t(x, b):          # 引数 x は整数、b は 0 と 1 を要素に持つリスト
       return int(sum(b) >= x)
def v(x):             # 引数 x は 0 または 1
       return 1 - x
```

指定されていない演算子やメソッド，関数などを使ってはいけません。例えば，以下のものは使用できません。

- import 文
- 累算代入演算子（+= や &= など）
- 算術演算子（+ や * など）
- 比較演算子（== や is, in など）
- ブール演算子（and など）
- ビット演算子（& や ~ など）

問２　引数 x,y が 0 または 1 であるとき，そのビット論理積を返す（すなわち x&y を返す）関数 c(x,y) を記述しなさい。

問３　引数 x,y が 0 または 1 であるとき，その算術和を返す関数 a(x,y)を記述しなさい。

問４　引数 x,y が 0～3 の整数値であるとき，その算術積を返す関数 m(x,y)を記述しなさい。