筑波大学　情報学群　情報科学類・情報メディア創成学類

令和６年度　学群編入学試験

学力試験問題（専門科目）

[注意事項]

1. 試験開始の合図があるまで、問題の中を見てはいけません。
2. 解答用紙と下書き用紙の定められた欄に、氏名、受験番号を記入すること。
3. この問題冊子は全部で 11 ページ（表紙、白紙を除く）です。
4. 問題１から問題４（数学、情報基礎）の計４問をすべて答えなさい。
5. 解答用紙は、４問に対して、各問１枚の合計４枚を用いること。
6. 解答用紙上部の［　　］欄に解答する問題番号を記入すること。
7. 解答用紙の裏面を使用する場合には、その旨を解答用紙の表面に示してください。

問題１　数学１

逆三角関数$\tan^{-1}x$について次の問いに答えなさい.

(1) 第 3 次導関数を求めなさい.

(2) $\tan^{-1}\frac{1}{2}$ の近似値として $\frac{11}{24}$ を用いた場合，その誤差は $\frac{1}{32}$ を超えないことを示しなさい.

問題2　数学2

$\boldsymbol{a} = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 4 \end{pmatrix}$, $\boldsymbol{b} = \begin{pmatrix} 4 \\ 2 \\ 3 \\ 1 \end{pmatrix}$, $\boldsymbol{c} = \begin{pmatrix} 5t \\ t+4 \\ 2t+3 \\ -t+6 \end{pmatrix}$ とする．ここでtは実数である．このとき，以下の問いに答えなさい．

(1) $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$が線形独立となるための条件を求めなさい．

(2) $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \perp \boldsymbol{c}$ となる $\boldsymbol{c}$ を $\boldsymbol{d}$ とする．$\boldsymbol{d}$を求めなさい．

(3) 次の連立一次方程式の解$\boldsymbol{x}$を求めなさい．

$$\begin{pmatrix} \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} & \boldsymbol{d} \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

問題3　情報基礎1

自然数 n に対して数列 $\{a_n\}$ を次のように定義する。

$$a_n = \begin{cases} 1 & (n = 1 \text{ または } n = 2 \text{ のとき}) \\ a_{n-1} + a_{n-2} & (n \geq 3 \text{ のとき}) \end{cases}$$

以下、C言語を用いてこの数列を計算することを考える。次の関数fは、自然数の引数nに対し、数列 $\{a_n\}$ の第n項を再帰計算によって求める。

```
long f(int n) {
    if (n < 3)
        return 1;
    else
        return [      (a)      ];
}
```

(1) 空欄 (a) を埋めて、関数fを完成させなさい。

上記の計算を高速化したい。次の関数gは、途中の計算を、サイズが十分に大きい配列に格納して再利用する。

```
#define MAX_SIZE 2000
long g(int n) {
    long a[MAX_SIZE];
    a[1] = a[2] = 1;
    for (int i = 3; i <= n; ++i)
        [      (b)      ];
    return a[n];
}
```

(2) 空欄 (b) を代入文1つで埋めて、関数gを完成させなさい。

(3) 関数gの時間計算量を、nを用いた漸近的計算量 (オーダー) で表しなさい。また、関数gがそのような時間計算量になる理由を述べなさい。

次ページに続く

さらなる高速化を考える。数列 $\{a_n\}$ の漸化式を行列を用いて表現すると

$$\begin{pmatrix} a_{n+2} \\ a_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix}$$

となる。ここで

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

とおくと、

$$\begin{pmatrix} a_{n+2} \\ a_{n+1} \end{pmatrix} = A \begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix} = A^2 \begin{pmatrix} a_n \\ a_{n-1} \end{pmatrix} = \cdots = A^n \begin{pmatrix} a_2 \\ a_1 \end{pmatrix}$$

となる。このとき、行列 A のべき乗 A^n の計算を高速化することを考える。一般に、

- n が奇数のとき $A^n = A \cdot A^{n-1}$
- n が偶数のとき $A^n = (A^{\frac{n}{2}})^2$

という関係が成り立つ。これに基づいて、2×2 行列の掛け算を行う関数 mult を用い、数列の第 n 項を計算する関数 h を実装する。

```
void mult(long A[2][2], long B[2][2], long C[2][2]) {
    long c00 = A[0][0] * B[0][0] + A[0][1] * B[1][0];
    long c10 = A[1][0] * B[0][0] + A[1][1] * B[1][0];
    long c01 = A[0][0] * B[0][1] + A[0][1] * B[1][1];
    long c11 = A[1][0] * B[0][1] + A[1][1] * B[1][1];
    C[0][0] = c00; C[1][0] = c10; C[0][1] = c01; C[1][1] = c11;
}

long h(int n) {
    long X[2][2] = {{1, 1}, {1, 0}};
    long Y[2][2] = {{1, 0}, {0, 1}};
    while (n > 0) {
        if (n % 2 == 1) mult( (c) );
        mult( (d) );
        n /= 2;
    }
    return Y[1][0];
}
```

次ページに続く

(4) 空欄 (c) および (d) を埋めて、関数 h を完成させなさい。

(5) 関数 h の時間計算量を、n を用いた漸近的計算量 (オーダー) で表しなさい。また、関数 h がそのような時間計算量になる理由を述べなさい。

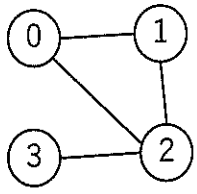
問題 4　情報基礎 2

重みなし無向グラフで幅優先探索を行い，探索の始点となる頂点から訪問できる他の頂点までの最短経路長を求める C 言語プログラム（プログラム 1）について，以下の問いに答えなさい．

(1) 本プログラムでは，グラフにおける各頂点を整数で表現し，頂点間の接続関係を隣接リストと呼ばれるデータ構造で表現している．図 1 は 4 頂点からなる無向グラフとそれに対応する隣接リストの例である．

プログラム中の g[i] は頂点 i に隣接する頂点を格納したリストの先頭を指している．

関数 addEdge は，グラフの隣接リスト g に辺 {u,v} を追加する関数である．空欄（ア）～（カ）を埋めて，関数を完成させなさい．



グラフ

頂点番号	隣接する頂点番号のリスト
0	(1, 2)
1	(0, 2)
2	(0, 1, 3)
3	(2)

左のグラフに対応する隣接リスト

図 1:グラフとそれに対応する隣接リストの例

(2) グラフを表すデータ構造として，辺 {u,v} が存在する場合に a[u][v]=1，存在しない場合に a[u][v]=0 とする二次元配列を用いることもできる．このようなデータ構造は隣接行列と呼ばれる．グラフを表現するデータ構造として，隣接行列ではなく隣接リストを用いることの利点を 120 字程度で説明しなさい．

(3) 関数 graphSearch はグラフ g における頂点 startVertex から辺をたどりながらグラフを探索し，頂点 startVertex から他の頂点への最短経路長を求める関数である．

空欄（キ）～（ケ）を埋めて，関数を完成させなさい．

(4) このプログラムを実行した時の出力（標準出力に表示される文字列）を答えなさい．

(5) このグラフ探索アルゴリズムの時間計算量の漸近計算量（オーダー）を理由とともに説明しなさい．ただしグラフの頂点数を N, 辺数を M としなさい．

次ページに続く

(プログラム1)

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_QUEUE_SIZE 100
// キューを管理する変数
int q[MAX_QUEUE_SIZE];
int tail=0;
int head=0;

// キューが空かどうかを判定する関数
int isEmpty() {
  if (head == tail)
    return 1;
  else
    return 0;
}

// キューが満杯かどうかを判定する関数
int isFull() {
  if (head == (tail+1) %MAX_QUEUE_SIZE)
    return 1;
  else
    return 0;
}
```

次ページに続く

```
// キューに要素を追加する関数
void enqueue(int value) {
  if (isFull()) {
    printf("Error: Queue is full¥n");
  } else {
    q[tail]=value;
    tail++;
    if(tail ==MAX_QUEUE_SIZE){
      tail=0;
    }
  }
}

// キューから要素を取り出す関数
int dequeue() {
  int item;
  if (isEmpty()) {
    printf("Error: Queue is empty¥n");
    return -1;
  } else {
    item = q[head];
    head++;
    if(head ==MAX_QUEUE_SIZE){
      head=0;
    }
    return item;
  }
}
```

次ページに続く

```
// グラフの隣接リストを表す構造体
typedef struct Element {
  int vertex;
  struct Element* nextElement;
} Element;

// グラフに辺を追加する関数
void addEdge(Element* g[], int u, int v) {
    Element* newElement = (Element*)malloc(sizeof(Element));
    newElement->vertex = [  (ア)  ];
    newElement->nextElement = [  (イ)  ];
    g[u] = [  (ウ)  ];

    newElement = (Element*)malloc(sizeof(Element));
    newElement->vertex = [  (エ)  ];
    newElement->nextElement = [  (オ)  ];
    g[v] = [  (カ)  ];
}

// グラフ探索を行う関数
void graphSearch(Element* g[], int startVertex, int numVertices)
{
  int visited[numVertices];
  int dist[numVertices];

  for (int i = 0; i < numVertices; ++i) {
    visited[i] = 0;
    dist[i]=-1;
  }

  visited[startVertex] = 1;
  enqueue(startVertex);
  dist[startVertex]=0;
```

次ページに続く

```
  while (!isEmpty()) {
    int currentVertex = [ (キ) ];
    printf("%d %d¥n", currentVertex,dist[currentVertex]);
    Element* tempElement = g[currentVertex];
    while (tempElement != NULL) {
      int adjVertex = tempElement->vertex;
      if (visited[adjVertex] == 0) {
        visited[adjVertex] = 1;
        dist[adjVertex]= [ (ク) ];
        [ (ケ) ];
      }
      tempElement = tempElement->nextElement;
    }
  }
}

int main() {
  int numVertices = 8;
  Element* g[numVertices];
  for (int i = 0; i < numVertices; ++i) {
    g[i] = NULL;
  }
  addEdge(g, 0, 1);
  addEdge(g, 0, 2);
  addEdge(g, 0, 4);
  addEdge(g, 1, 3);
  addEdge(g, 1, 6);
  addEdge(g, 2, 5);
  addEdge(g, 3, 7);
  addEdge(g, 4, 6);
  addEdge(g, 5, 6);
  addEdge(g, 5, 7);
```

次ページに続く

```
  int startVertex = 0;
  graphSearch(g, startVertex, numVertices);
  return 0;
}
```
