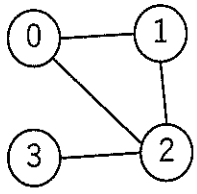
問題 4　情報基礎 2

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;
}
```
