МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» им.В.И.УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

Отчет по курсовой работе
по дисциплине
«Алгоритмы и структуры данных»

Тема: «ГРАФЫ»

Студент гр. 9305 Николаенко К.Н.

Преподаватель Манираген В.

Санкт-Петербур
г 2020

Содержание

Цель работы

Исследование алгоритмов для работы с графами

Задание

Получение множества двудольных компонент неориентированного графа.

Математическая формулировка задачи в терминах теории множеств

Задание задачи гласит, что нам требуется получить множество двудольных компонент неориентированного графа. Неориентированный граф – это упорядоченная пара (V,E) где, V – непустое множество вершин и узлов, а E – множество пар неупорядоченных вершин – рёбер. В контексте же определения двудольных компонент мы использовали понятие цвета вершины.

То есть компонента графа будет двудольной в том случае, когда любые две вершины, соединенные ребром, будут иметь разную раскраску.

Выбор и обоснование способа представления данных

Как способ представления данных мы выбрали классы. Как мы уже замечали в предыдущих выводах, классы очень удобны в работе с абстрактными структурами данных и их определенной логикой обработки. В классе собираются данные, относящиеся к графу в целом и функции-члены для работы с ним. И в главной функции main просто по порядку вызываются нужные функции для исполнения алгоритма. Как способ хранения графа в памяти мы использовали массив, каждый элемент которого множество рёбер в форме вектора битов. Единичные биты соответствуют
ребрам, инцидентным данной вершине. Заполняется такой массив рандомно.

Описание алгоритма и оценка временной сложности

С самого начала пользователь задает только количество ребер в графе. Далее генерируется граф с заданным количеством ребер (связями рандомных вершин в случайном порядке). Сам граф представлен в виде массива. И чтобы вывести граф, нам нужно вывести сгенерированный массив, состоящий из нулей и единиц. Единичные биты массива и представляют собой инцидентность вершин. Как мы уже писали в математической формулировке задачи, в контексте определения двудольности мы использовали понятие цвета вершины. Граф будет двудольным в том случае, когда ребра соединяют вершины разного цвета.

Покраску графа мы реализовали с помощью одномерного массива, ячейки в котором принимают значения 1 и -1, которые в свою очередь обозначают цвета: красный и синий соответственно. Покраска графа начинается с вершины А, которая “красится в цвет 1”. Далее рекурсивный алгоритм делает проверку 3 условий. Не окрашена ли вершина уже, не вернется ли алгоритм в вершину из которой пришел и есть ли вообще путь в проверяемую нами вершину. Если все условия выполнелись,
алгоритм перейдет на другую вершину, цвет которой уже будет -1 и начнет дальнейшую проверку уже для этой вершины и так далее. Затем мы выводим получившиеся компоненты. Мы также проходим по всем вершинам, проверяя при этом цвет вершины, из которой мы пришли. Если цвет одинаков, значит компонента не является двудольной и мы ее не выводим.

Получившийся алгоритм имеет сложность O (n^2).

Дополнительная информация

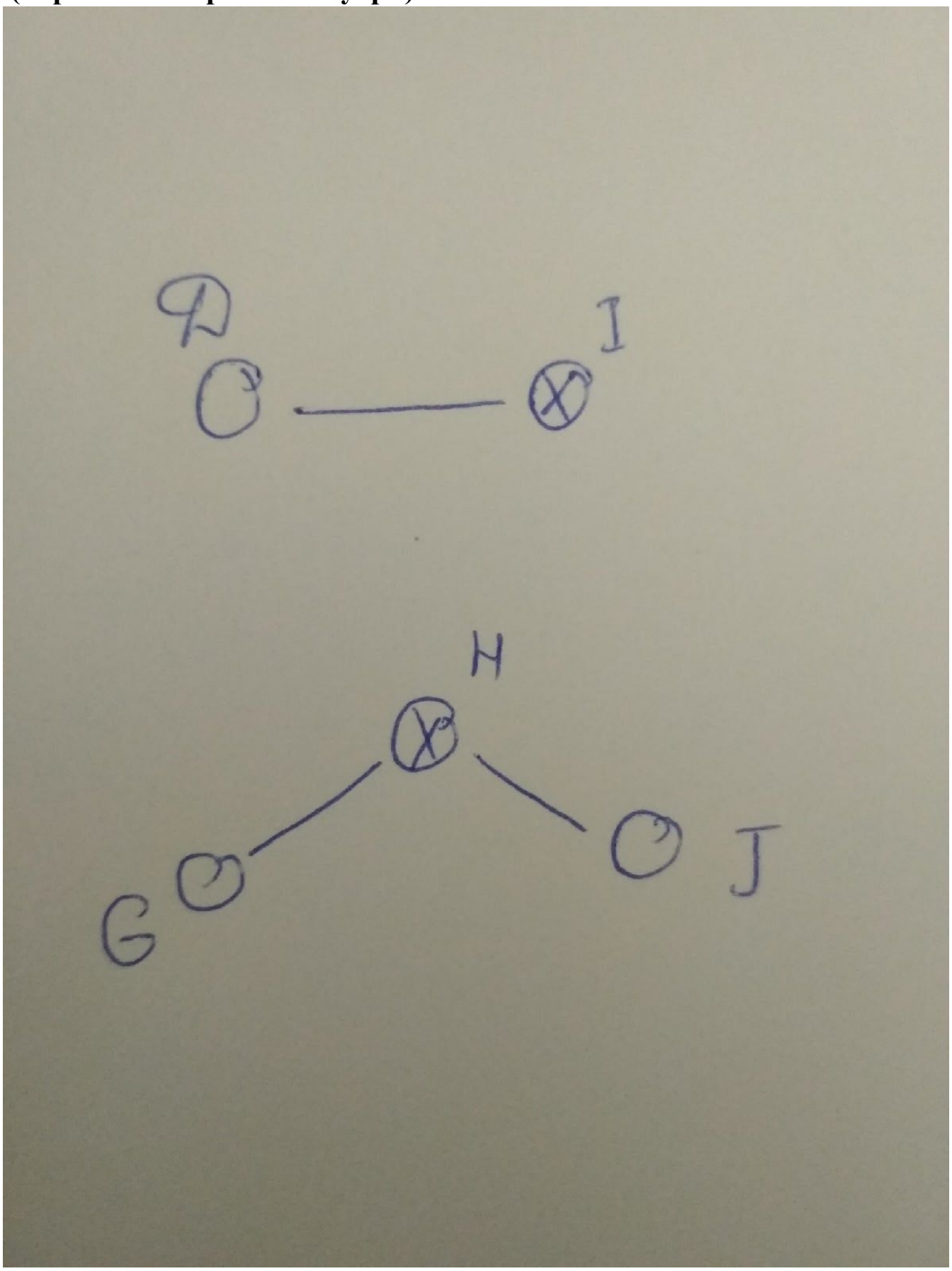
Двудольные графы часто используют в нахождении максимального паросочетания (раздел курса Комбинаторика и Теория Графов), то есть там граф разделяют на две доли, и после этого происходит поиск пар, так чтобы их было максимальное количество.

Тесты и результаты проверки алгоритмов

1. 2 двудольные компоненты

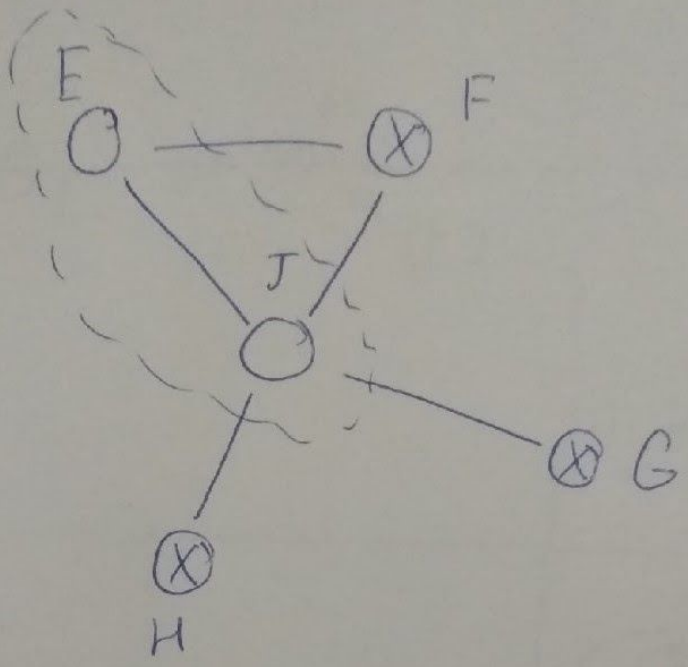
```
Generated graph:

      A  B  C  D  E  F  G  H  I  J
     ------------------------------
 A  |  0  0  0  0  0  0  0  0  0  0
 B  |  0  0  0  0  0  0  0  0  0  0
 C  |  0  0  0  0  0  0  0  0  0  0
 D  |  0  0  0  0  0  0  0  0  1  0
 E  |  0  0  0  0  0  0  0  0  0  0
 F  |  0  0  0  0  0  0  0  0  0  0
 G  |  0  0  0  0  0  0  0  1  0  0
 H  |  0  0  0  0  0  0  1  0  0  1
 I  |  0  0  0  1  0  0  0  0  0  0
 J  |  0  0  0  0  0  0  0  1  0  0

Graph coloring:

Color [A] - RED
Color [B] - RED
Color [C] - RED
Color [D] - RED
Color [E] - RED
Color [F] - RED
Color [G] - RED
Color [H] - BLACK
Color [I] - BLACK
Color [J] - RED

Bipartite components of the graph:

1.[ID]
2.[JHG]

Process returned 0 (0x0)   execution time : 3.819 s
Press any key to continue.
```

(окрашена = крестик внутри)



2. 0 Двудольных компонент

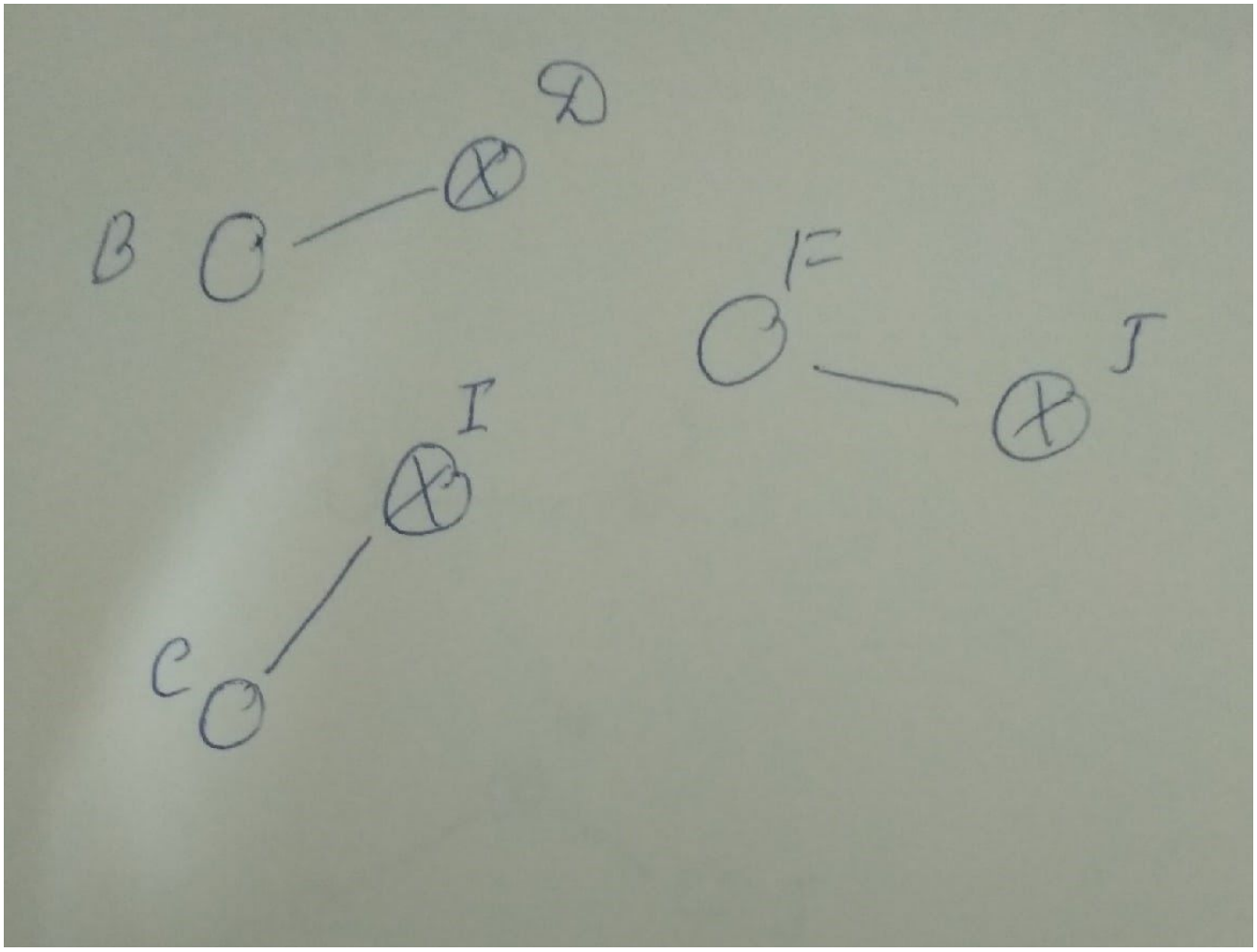
```
Generated graph:

      A  B  C  D  E  F  G  H  I  J
     ------------------------------
 A  |  0  0  0  0  0  0  0  0  0  0
 B  |  0  0  0  0  0  0  0  0  0  0
 C  |  0  0  0  0  0  0  0  0  0  0
 D  |  0  0  0  0  0  0  0  0  0  0
 E  |  0  0  0  0  0  1  0  0  0  1
 F  |  0  0  0  0  1  0  0  0  0  1
 G  |  0  0  0  0  0  0  0  0  0  1
 H  |  0  0  0  0  0  0  0  0  0  1
 I  |  0  0  0  0  0  0  0  0  0  0
 J  |  0  0  0  0  1  1  1  1  0  0

Graph coloring:

Color [A] - RED
Color [B] - RED
Color [C] - RED
Color [D] - RED
Color [E] - RED
Color [F] - BLACK
Color [G] - BLACK
Color [H] - BLACK
Color [I] - RED
Color [J] - RED

Bipartite components of the graph:

NO!

Process returned 0 (0x0)   execution time : 0.048 s
Press any key to continue.
```



E и J одного цвета, что не позволительно

3. 3 Двудольные компоненты

```
Generated graph:

      A  B  C  D  E  F  G  H  I  J
     ------------------------------
 A  |  0  0  0  0  0  0  0  0  0  0
 B  |  0  0  0  1  0  0  0  0  0  0
 C  |  0  0  0  0  0  0  0  0  1  0
 D  |  0  1  0  0  0  0  0  0  0  0
 E  |  0  0  0  0  0  0  0  0  0  0
 F  |  0  0  0  0  0  0  0  0  0  1
 G  |  0  0  0  0  0  0  0  0  0  0
 H  |  0  0  0  0  0  0  0  0  0  0
 I  |  0  0  1  0  0  0  0  0  0  0
 J  |  0  0  0  0  0  1  0  0  0  0

Graph coloring:

Color [A] - RED
Color [B] - RED
Color [C] - RED
Color [D] - BLACK
Color [E] - RED
Color [F] - RED
Color [G] - RED
Color [H] - RED
Color [I] - BLACK
Color [J] - BLACK

Bipartite components of the graph:

1.[DB]
2.[IC]
3.[JF]

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```


B
D
I
C
F
J

Выводы

В процессе выполнения данной лабораторной работы мы использовали 2 рекурсивных алгоритма для раскраски графа и поиска двудольных компонент. Оба алгоритма имеют сложность $O(n^2)$. Также мы поняли, что для тестирования алгоритмов нужно использовать графы с большим количеством вершин (в нашем случае это 10), чтобы проверить правильность работы при всевозможных случаях их соединения.

Источники

Колинько П.Г. Пользовательские структуры данных: Методические указания по дисциплине “Алгоритмы и структуры данных, часть 1” - СПб.: СПБГЭТУ “ЛЭТИ”, 2020. -64 с.

Двудольные графы https://brestprog.by/topics/bipartite/

Двудольность графа C++ https://www.cyberforum.ru/cpp- builder/thread1036647.html

Поиск в глубину https://kvodo.ru/dfs.html

Приложение А

main.cpp

```
#include <iostream>
#include <ctime>
#include "graf.h"

using namespace std;

int main()
{
   srand(time(nullptr));
   int a = rand() % 10 + 1;

   graf gr(a);

   cout << "\n Generated graph: \n\n";

   gr.PrintGraph();

   cout << "\n Graph coloring:\n\n";

   gr.Set1();

   cout << "\n Bipartite components of the graph: \n\n";

   gr.Search();

   return 0;
}
```

graf.h

```
#ifndef GRAF_H
#define GRAF_H
#include <iostream>

class graf
{
   static const int gSize = 10;
   bool g[gSize][gSize]{};
   int color[gSize]{};

public:
   graf(int a);

   void PrintGraph();

   void SetColor(int s, int col, int t);

   void Set1();

   void Print(int& count, int color2[gSize], int* result, int s, int col, int t);

   void Search();
};

#endif // GRAF_H
```

graf.cpp

```
#include "graf.h"

using namespace std;

graf :: graf(int a)
{
   for (int t = 0; t < a; t++)
   {
      int x = rand() % gSize;
      int y = rand() % gSize;

      if (x == y) y++;

      if (g[x][y] != 0) t--;

      g[x][y] = 1;
      g[y][x] = g[x][y];
   }
}


void graf :: PrintGraph()
{
   cout << "     ";

   for (int i = 0; i < gSize; i++) printf("%3c", 'A' + i);

   cout << "\n      ";
   cout << "-----------------------------\n";

   for (int i = 0; i < gSize; i++)
   {
      printf("  %c  ", 'A' + i);

      cout << "|";

      for (int j = 0; j < gSize; j++) printf("%3d", g[i][j]);

      cout << "\n";
   }
}
```

```
void graf :: SetColor(int s, int col, int t)
{
   color[s] = col;

   for (int j = 0; j < gSize; j++)
      if ((j != t) && (color[j] == 0) && (g[s][j] != 0))
         SetColor(j, col * -1, s);
}


void graf :: Set1()
{
   int col = 1;
   for (int i = 0; i < gSize; i++)
      if (color[i] == 0)
         SetColor(i, col, i);

   for (int i = 0; i < gSize; i++)
   {
      printf(" Color [%c] - ", 'A' + i);

      if (color[i] == 1) cout << "RED\n";

      if (color[i] == -1) cout << "BLACK\n";
   }
}


void graf :: Print(int& count, int color2[gSize], int* result, int s, int col, int t)
{
   bool flag = true;
   color2[s] = 0;

   for (int j = 0; j < gSize; j++)
      if ((j != t) && (g[s][j] != 0))
         if (color[j] != col * -1)
            flag = false;

   if (flag)
      for (int j = 0; j < gSize; j++)
         if ((j != t) && (g[s][j] != 0))
            if (color2[j] != 0)
               Print(count, color2, result, j, col * -1, s);
```

```
    if (flag == false) count = -10;

    if (count >= 0)
    {
        result[count] = s;
        count++;
    }
}


void graf :: Search()
{
    int col = 1;
    int count = 0;
    int count2 = 1;
    int color2[gSize];
    int kod = 0;

    for (int i = 0; i < gSize; i++) color2[i] = 2;

    int result[gSize];

    for (int i = 0; i < gSize; i++)
    {
        if (color2[i] != 0)
            Print(count, color2, result, i, col, i);

        if (count > 1)
        {
            kod = 1;
            printf(" %d.", count2);
            cout << "[";

            for (int t = 0; t < count; t++)
                printf("%1c", 'A' + result[t]);

            cout << "]";
            cout << '\n';

            count2++;
        }
        count = 0;
    }
```

```
    if(!kod) cout << " NO!" << endl;
}
```