Министерство высшего образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Пермский национальный исследовательский политехнический университет» (ПНИПУ)
Электротехнический факультет
Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ
Дискретная математика
«Построение остова»

Выполнил
Студент группы РИС-22-2б
Прядеин И.А.
Проверил доцент кафедры
ИТАС
Рустамханова Г. И.

Пермь 2024

Постановка задачи:

Дана матрица смежности неориентированного взвешенного графа, состоящего из 10 вершин.

Найти остов исходного графа с помощью алгоритмов Прима и Краскала.

Алгоритм работы:

Считывание файла (рис. 1) реализовано с помощью функции “LoadGraph”

```
void LoadGraph(char *filename)
{
    FILE *file;
    file = fopen(filename, "r");
    int c, num, row, column;

    row = column = 0;
    if (file) {
        while ((c = fgetc(file)) != EOF) {
            num = c - '0';
            if (num >= 0 && num <= 100)
                graph[row][column++] = num;
            if (column == 10) {
                column = 0;
                ++row;
            }
        }
    } else {
        printf("Can't open the file.\n");
    }
}
```

Рис.1 – Считывание матрицы из файла

```
void KruskalAlgorithm(void)
{
    printf("Kruskal's algorithm:\n");
    int cost = 0;
    Edge result[MAX_EDGES] = {0};

    int edgeIndex = 0;
    for (int row = 0; row < MAX_VERTICES; ++row)
        for (int col = row + 1; col < MAX_VERTICES; ++col)
            if (graph[row][col] != 0) {
                edgeList[edgeIndex].u = row;
                edgeList[edgeIndex].v = col;
                edgeList[edgeIndex].w = graph[row][col];
                ++edgeIndex;
                if (edgeIndex >= MAX_EDGES)
                    return;
            }

    for (int vertIndex = 0; vertIndex < MAX_VERTICES; ++vertIndex) {
        MakeSet(vertIndex);
   }

    Sort();

    for (int edgeIndex = 0; edgeIndex < MAX_EDGES; ++edgeIndex)
        if (edgeList[edgeIndex].w)
            if (FindSet(edgeList[edgeIndex].u) != FindSet(edgeList[edgeIndex].v)) {
                cost += edgeList[edgeIndex].w;
                result[edgeIndex] = edgeList[edgeIndex];
                UnionSets(edgeList[edgeIndex].u, edgeList[edgeIndex].v);
            }

    Print(result);
    printf("Total cost: %d\n", cost);
}
```

Рис. 2 – Алгоритм Краскала

Алгоритм Краскала реализован в функции “KruskalAlgorithm” (рис. 2). Изначально строится массив ребер и система непересекающихся множеств. После происходит сортировка ребер и поиск остова.

Алгоритм Прима реализован в функции “PrimAlgorithm” (рис. 3). Сначала берется первая вершина и находиться инцидентное ребро с наименьшим весом. Найденная вершина помечается. После проверяются оставшиеся вершины.

```
void PrimAlgorithm(void)
{
    printf("\nPrim's algorithm:\n");
    int selected[MAX_EDGES] = {0};
    int cost = 0;
    int minRow, minCol;

    selected[0] = 1;
    int edgeIndex = 0;
    for (int vertIndex = 0; vertIndex < MAX_VERTICES - 1; ++vertIndex) {
        int min = 100;
        minRow = minCol = 0;

        for (int row = 0; row < MAX_VERTICES; ++row)
            if (selected[row])
                for (int col = 0; col < MAX_VERTICES; ++col)
                    if (!selected[col] && graph[row][col])
                        if (graph[row][col] < min) {
                            min = graph[row][col];
                            minRow = row;
                            minCol = col;
                        }

        printf("%2d -- %2d: %2d\n", minRow + 1, minCol + 1, graph[minRow][minCol]);
        cost += graph[minRow][minCol];
        selected[minCol] = 1;
    }
    printf("Total cost: %d\n", cost);
}
```

Рис. 3 – Алгоритм Прима

После происходит вывод конечного остова функцией “Print” (рис. 4).

```
void Print(Edge edges[MAX_EDGES])
{
    for (int edgeIndex = 0; edgeIndex < MAX_EDGES; ++edgeIndex)
        if (edges[edgeIndex].w)
            printf("%2d -- %2d: %2d\n",
                   edges[edgeIndex].u + 1, edges[edgeIndex].v + 1, edges[edgeIndex].w);
}
```

Рис. 4 – Вывод остова

Результат тестирования:

```
    1  2  3  4  5  6  7  8  9 10
 1|  0  0  0  7  8  6  3  0  0  0
 2|  0  0  0  4  2  1  0  3  9  1
 3|  0  0  0  5  9  0  6  8  5  0
 4|  7  4  5  0  0  0  6  0  0  9
 5|  8  2  9  0  0  0  0  4  5  0
 6|  6  1  0  0  0  0  3  0  7  0
 7|  3  0  6  6  0  3  0  0  3  0
 8|  0  3  8  0  4  0  0  0  4  5
 9|  0  9  5  0  5  7  3  4  0  8
10|  0  1  0  9  0  0  0  5  8  0

Kruskal's algorithm:
 2 -- 10:  1
 2 --  6:  1
 2 --  5:  2
 7 --  9:  3
 2 --  8:  3
 1 --  7:  3
 6 --  7:  3
 2 --  4:  4
 3 --  9:  5
Total cost: 25

Prim's algorithm:
 1 --  7:  3
 7 --  6:  3
 6 --  2:  1
 2 -- 10:  1
 2 --  5:  2
 2 --  8:  3
 7 --  9:  3
 2 --  4:  4
 4 --  3:  5
Total cost: 25
```

Рис. 5 – Файл “g21”

```
    1  2  3  4  5  6  7  8  9 10
 1|  0  1  0  3  0  0  0  0  0  0
 2|  1  0  5  0  2  0  0  4  0  2
 3|  0  5  0  0  0  1  0  0  0  3
 4|  3  0  0  0  0  0  0  5  3  0
 5|  0  2  0  0  0  4  0  0  0  0
 6|  0  0  1  0  4  0  4  0  0  5
 7|  0  0  0  0  0  4  0  2  0  0
 8|  0  4  0  5  0  0  2  0  1  0
 9|  0  0  0  3  0  0  0  1  0  0
10|  0  2  3  0  0  5  0  0  0  0

Kruskal's algorithm:
 8 --  9:  1
 1 --  2:  1
 3 --  6:  1
 2 -- 10:  2
 2 --  5:  2
 7 --  8:  2
 3 -- 10:  3
 4 --  9:  3
 1 --  4:  3
Total cost: 18

Prim's algorithm:
 1 --  2:  1
 2 --  5:  2
 2 -- 10:  2
 1 --  4:  3
 4 --  9:  3
 9 --  8:  1
 8 --  7:  2
10 --  3:  3
 3 --  6:  1
Total cost: 18
```

Рис. 6 – Файл “g12”

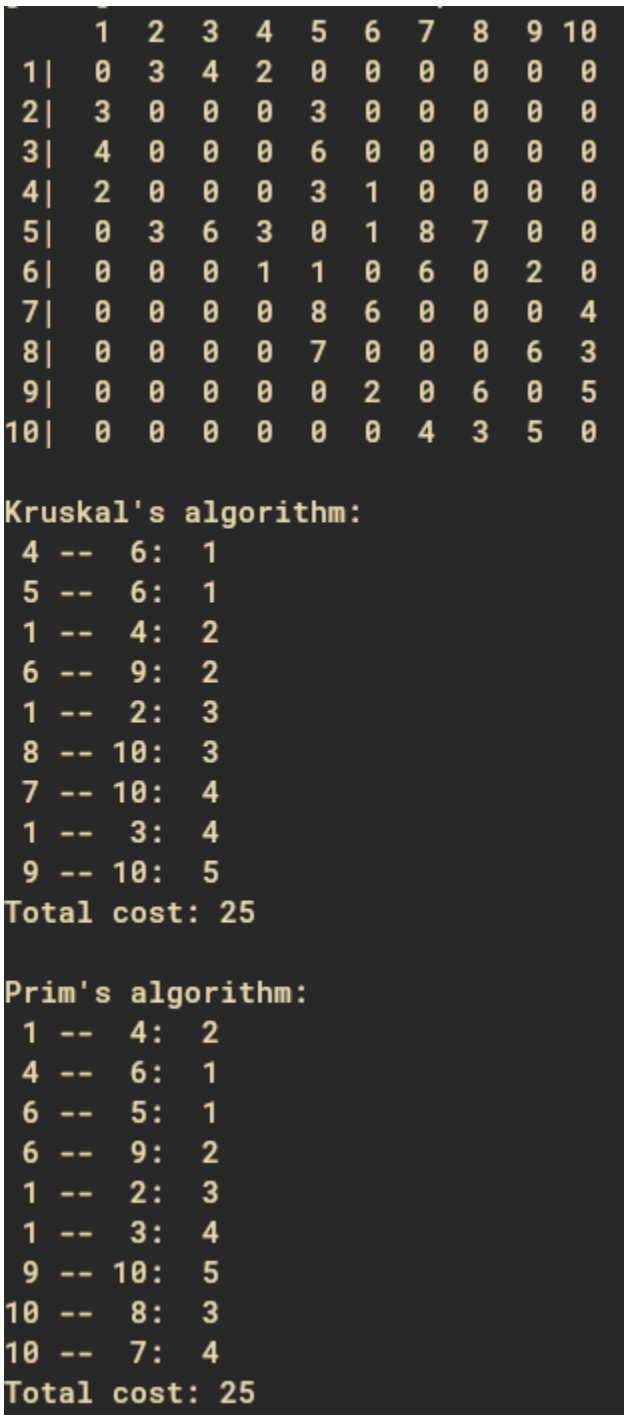

Рис. 7 – Файл “g13”

Исходный код:

```
#include <stdio.h>

#define MAX_VERTICES 10
#define MAX_EDGES 45

typedef struct {
    int u;
    int v;
    int w;
} Edge;

void KruskalAlgorithm(void);
void Sort(void);
void Print(Edge edges[MAX_EDGES]);

Edge edgeList[MAX_EDGES] = {0};
int graph[MAX_VERTICES][MAX_VERTICES] = {0};
int parent[MAX_VERTICES] = {0};
int rank[MAX_VERTICES] = {0};

void LoadGraph(char *filename)
{
    FILE *file;
    file = fopen(filename, "r");
    int c, num, row, column;

    row = column = 0;
    if (file) {
```

```
        while ((c = fgetc(file)) != EOF) {
            num = c - '0';
            if (num >= 0 && num <= 100)
                graph[row][column++] = num;
            if (column == 10) {
                column = 0;
                ++row;
            }
        }
    } else {
        printf("Can't open the file.\n");
    }
}

void MakeSet(int v)
{
    parent[v] = v;
    rank[v] = 0;
}

int FindSet(int v)
{
    if (v == parent[v])
        return v;
    return parent[v] = FindSet(parent[v]);
}

void UnionSets(int a, int b)
{
    a = FindSet(a);
```

```
    b = FindSet(b);
    if (a != b) {
        if (rank[a] < rank[b]) {
            int temp = a;
            a = b;
            b = temp;
        }
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}

void KruskalAlgorithm(void)
{
    printf("Kruskal's algorithm:\n");
    int cost = 0;
    Edge result[MAX_EDGES] = {0};

    int edgeIndex = 0;
    for (int row = 0; row < MAX_VERTICES; ++row)
        for (int col = row + 1; col < MAX_VERTICES; ++col)
            if (graph[row][col] != 0) {
                edgeList[edgeIndex].u = row;
                edgeList[edgeIndex].v = col;
                edgeList[edgeIndex].w = graph[row][col];
                ++edgeIndex;
                if (edgeIndex >= MAX_EDGES)
                    return;
            }
```

```
    for (int vertIndex = 0; vertIndex < MAX_VERTICES; ++vertIndex) {
        MakeSet(vertIndex);
    }

    Sort();

    for (int edgeIndex = 0; edgeIndex < MAX_EDGES; ++edgeIndex)
        if (edgeList[edgeIndex].w)
            if (FindSet(edgeList[edgeIndex].u) != FindSet(edgeList[edgeIndex].v)) {
                cost += edgeList[edgeIndex].w;
                result[edgeIndex] = edgeList[edgeIndex];
                UnionSets(edgeList[edgeIndex].u, edgeList[edgeIndex].v);
            }

    Print(result);
    printf("Total cost: %d\n", cost);
}

void Sort(void)
{
    for (int prev = 0; prev < MAX_EDGES; ++prev)
        for (int next = prev + 1; next < MAX_EDGES - 1; ++next)
            if (edgeList[prev].w > edgeList[next].w) {
                Edge temp = edgeList[prev];
                edgeList[prev] = edgeList[next];
                edgeList[next] = temp;
            }
}
```

```
void PrimAlgorithm(void)
{
   printf("\nPrim's algorithm:\n");
   int selected[MAX_EDGES] = {0};
   int cost = 0;
   int minRow, minCol;

   selected[0] = 1;
   int edgeIndex = 0;
   for (int vertIndex = 0; vertIndex < MAX_VERTICES - 1; ++vertIndex) {
      int min = 100;
      minRow = minCol = 0;

      for (int row = 0; row < MAX_VERTICES; ++row)
         if (selected[row])
            for (int col = 0; col < MAX_VERTICES; ++col)
               if (!selected[col] && graph[row][col])
                  if (graph[row][col] < min) {
                     min = graph[row][col];
                     minRow = row;
                     minCol = col;
                  }

      printf("%2d -- %2d: %2d\n", minRow + 1, minCol + 1, graph[minRow][minCol]);
      cost += graph[minRow][minCol];
      selected[minCol] = 1;
   }
   printf("Total cost: %d\n", cost);
}
```

```
void Print(Edge edges[MAX_EDGES])
{
    for (int edgeIndex = 0; edgeIndex < MAX_EDGES; ++edgeIndex)
        if (edges[edgeIndex].w)
            printf("%2d -- %2d: %2d\n",
                edges[edgeIndex].u + 1, edges[edgeIndex].v + 1, edges[edgeIndex].w);
}

int main(void)
{
    LoadGraph("g23.txt");
    printf("    ");
    for (int col = 0; col < MAX_VERTICES; ++col)
        printf("%2d ", col + 1);
    printf("\n");
    for (int row = 0; row < MAX_VERTICES; ++row) {
        printf("%2d| ", row + 1);
        for (int column = 0; column < MAX_VERTICES; ++column)
            printf("%2d ", graph[row][column]);
        printf("\n");
    }
    printf("\n");

    KruskalAlgorithm();
    PrimAlgorithm();
    return 0;
}
```