TAREFA TEG#2

Aluno: Alex Halatiki Vicente

Para iniciar a parte 2 do trabalho, foram necessárias algumas alterações no código referente à parte 1. A principal foi a necessidade de criação de duas structs para representar as flores e os grupos de flores como mostrado abaixo.

```
typedef struct {
  char nome[50];
  double valores[N];
} Flor;

typedef struct {
  int *vertices;
  int quantVertices;
  int ordem;
  double centro;
  char nome[50];
} Grupo;
```

A partir disso, a forma como lemos os valores e o nome das flores foi alterado, armazenando os dados como ponteiro de Flor, ou seja, um array de flores, como mostra o código abaixo.

```c
Flor *leEntrada(int *linhas)
{
    FILE *arquivo;
    Flor * matrizVertices = NULL;
    char linha[50];

    arquivo = fopen("IrisDataset.csv", "r");

    if (arquivo == NULL)
        return NULL;

    while (fgets(linha, 50, arquivo) != NULL)
    {
        matrizVertices = realloc(matrizVertices, sizeof(Flor) * (linhas[0]+1));

        Flor flor;

        char *token = strtok(linha, ",");
        int aux = 0;

        while (token != NULL)
        {
            if(aux == 4)
            {
              strcpy(flor.nome, token);
            }
            else
            {
              flor.valores[aux] = atof(token);
              aux++;
            }
            token = strtok(NULL, ",");
        }
        matrizVertices[linhas[0]] = flor;
        linhas[0]++;
    }

    fclose(arquivo);

    return matrizVertices;
}
```

Assim podemos dar continuidade no trabalho utilizando boa parte do código existente. Então, a partir da matriz de adjacências encontrada na parte 1, realizamos uma busca em profundidade sobre ela (grafo), para separar os grupos existentes como mostra o código abaixo. Vale ressaltar que o limiar utilizado foi de 0.08, limiar esse que nos gera 6 grupos no total, sendo 3 grupos principais que possuem a maior quantidade de vértices, como desejado na descrição do trabalho.

```c
void dfs(double **matriz, int *visitados, int vertice, int numVertices, Grupo *grupos, int quantGrupos, Flor *vertices)
{
    visitados[vertice] = 1;

    for (int i = 0; i < numVertices; i++)
      if(matriz[vertice][i] && visitados[i] == 0)
      {
        //printf("\n%d com %d\n", vertice, i);
        for(int j=0;j<N;j++)
            grupos[quantGrupos-1].centro += vertices[i].valores[j];

        grupos[quantGrupos-1].quantVertices++;
        int aux = grupos[quantGrupos-1].quantVertices;
        grupos[quantGrupos-1].vertices = realloc(grupos[quantGrupos-1].vertices, sizeof(int) * aux);
        grupos[quantGrupos-1].vertices[aux-1] = i;
        dfs(matriz, visitados, i, numVertices, grupos, quantGrupos, vertices);
      }
}

Grupo * dfsInicia(double **matriz, int numVertices, Grupo *grupos, int *quantGrupos, Flor *vertices)
{
    int *verticesVisitados = malloc(sizeof(int) * numVertices);

    for (int i = 0; i < numVertices; i++)
    {
        verticesVisitados[i] = 0;
    }

    int grupoAux = 0;

    for (int i = 0; i < numVertices; i++)
    {
        if (verticesVisitados[i] == 0)
        {
            //printf("\ngrupo %d\n", grupoAux);
            Grupo grupo;
            grupo.ordem = grupoAux++;
            grupo.quantVertices = 1;
            grupo.vertices = malloc(sizeof(int));
            grupo.vertices[0] = i;

            double soma = 0;
            for(int j=0;j<N;j++)
                soma += vertices[i].valores[j];
            grupo.centro = soma;

            quantGrupos[0]++;
            grupos = realloc(grupos, sizeof(Grupo) * quantGrupos[0]);
            grupos[quantGrupos[0] - 1] = grupo;

            dfs(matriz, verticesVisitados, i, numVertices, grupos, quantGrupos[0], vertices);
            grupos[quantGrupos[0] - 1].centro = grupos[quantGrupos[0] - 1].centro / (grupos[quantGrupos[0] - 1].quantVertices * N);
        }
    }

    free(verticesVisitados);

    return grupos;
}
```

Com os vértices separados em 6 grupos desconexos, ordenamos o array de grupos para que possamos calcular o centro de cada grupo e alocar os demais grupos menores nos 3 grupos principais, levando em conta a menor distância do centro de um grupo para outro, como mostra o código abaixo.

```c
void ordenaGruposDecescente(Grupo *grupos, int quantGrupos)
{
    for (int i = 0; i < quantGrupos - 1; i++)
    {
        int indiceMaior = i;

        for (int j = i + 1; j < quantGrupos; j++)
        {
            if (grupos[j].quantVertices > grupos[indiceMaior].quantVertices)
            {
                indiceMaior = j;
            }
        }

        Grupo temp = grupos[i];
        grupos[i] = grupos[indiceMaior];
        grupos[indiceMaior] = temp;
    }
}

Grupo *ajustaEm3Grupos(Grupo *grupos, int quantGrupos)
{
    Grupo *gruposAux = malloc(sizeof(Grupo) * 3);

    for (int i = 0; i < 3; i++)
    {
        gruposAux[i] = grupos[i];

        gruposAux[i].vertices = malloc(sizeof(int) * grupos[i].quantVertices);
        for (int j = 0; j < grupos[i].quantVertices; j++)
        {
            gruposAux[i].vertices[j] = grupos[i].vertices[j];
        }
    }

    for (int i = 3; i < quantGrupos; i++)
    {
        double menor = abs(grupos[i].centro - gruposAux[0].centro);
        int indice = 0;
        for (int j = 1; j < 3; j++)
        {
            if (abs(grupos[i].centro - gruposAux[j].centro) < menor)
            {
                menor = abs(grupos[i].centro - gruposAux[j].centro);
                indice = j;
            }
        }

        gruposAux[indice].vertices = realloc(gruposAux[indice].vertices, sizeof(int) * (gruposAux[indice].quantVertices + grupos[i].quantVertices));

        for (int c = 0; c < grupos[i].quantVertices; c++)
        {
            gruposAux[indice].vertices[gruposAux[indice].quantVertices + c] = grupos[i].vertices[c];
        }
        gruposAux[indice].quantVertices += grupos[i].quantVertices;
    }

    return gruposAux;
}
```

Assim, com os todos os vértices alocados em apenas 3 grupos, analisamos a predominância do tipo de flor em cada grupo, e então, definimos o nome do grupo de acordo com o tipo de flor predominante nele, como mostra o código abaixo.

```c
void defineNomeGrupos(Grupo *grupos, int quantGrupos, Flor *vertices)
{
    for(int i=0;i<quantGrupos;i++)
    {
        int setosa = 0;
        int virginica = 0;
        int versicolor = 0;

        for(int j=0;j<grupos[i].quantVertices;j++)
        {
            if(strstr(vertices[grupos[i].vertices[j]].nome, "Virginica") != NULL)
            virginica++;
            if(strstr(vertices[grupos[i].vertices[j]].nome, "Setosa") != NULL)
            setosa++;
            if(strstr(vertices[grupos[i].vertices[j]].nome, "Versicolor") != NULL)
            versicolor++;
        }

        if(setosa > virginica && setosa > versicolor)
        strcpy(grupos[i].nome, "Setosa");
        else
        {
        if(virginica > setosa && virginica > versicolor)
            strcpy(grupos[i].nome, "Virginica");
        else
            strcpy(grupos[i].nome, "Versicolor");
        }
    }
}
```

Com isso, e possível calcularmos os casos TP (true positive), FP (false positive), TN (true negative) e FN (false negative) e realizar a extração das métricas de qualidade da classificação, como mostra o código abaixo.

```c
int truePositivos(Grupo grupo, Flor *vertices)
{
    int soma = 0;
    for(int i=0;i<grupo.quantVertices;i++)
        if(strstr(vertices[grupo.vertices[i]].nome, grupo.nome) != NULL)
            soma++;

    return soma;
}

int falsePositivos(Grupo grupo, Flor *vertices)
{
    int soma = 0;
    for(int i=0;i<grupo.quantVertices;i++)
        if(strstr(vertices[grupo.vertices[i]].nome, grupo.nome) == NULL)
            soma++;

    return soma;
}

int falseNegativos(Grupo *grupos, int quantGrupos, Flor*vertice, Grupo grupo)
{
    int soma = 0;

    for(int i=0;i<quantGrupos;i++)
    {
        if(grupos[i].ordem == grupo.ordem)
            continue;

        for(int j=0;j<grupos[i].quantVertices;j++)
            if(strstr(vertice[grupos[i].vertices[j]].nome, grupo.nome) != NULL)
                soma++;
    }

    return soma;
}

int trueNegativos(Grupo *grupos, int quantGrupos, Flor*vertice, Grupo grupo)
{
    int soma = 0;

    for(int i=0;i<quantGrupos;i++)
    {
        if(grupos[i].ordem == grupo.ordem)
            continue;

        for(int j=0;j<grupos[i].quantVertices;j++)
            if(strstr(vertice[grupos[i].vertices[j]].nome, grupo.nome) == NULL)
                soma++;
    }

    return soma;
}
```

Concluindo, obtemos as seguintes métricas em relação a cada grupo:

```
Grupo Virginica:

True Positivos: 50
False Positivos: 46
True Negativos: 54
False Negativos: 0

Acuracia: 0.693333
Recall: 1.000000
Precision: 0.520833
F1 Score: 0.684932

Grupo Setosa:

True Positivos: 50
False Positivos: 0
True Negativos: 100
False Negativos: 0

Acuracia: 1.000000
Recall: 1.000000
Precision: 1.000000
F1 Score: 1.000000

Grupo Versicolor:

True Positivos: 4
False Positivos: 0
True Negativos: 100
False Negativos: 46

Acuracia: 0.693333
Recall: 0.080000
Precision: 1.000000
F1 Score: 0.148148
```