FIREFLY ALGORITHM

```
import numpy as np
import math
import random

class FireflyAlgorithm:
    def __init__(self, n_fireflies, n_documents, n_sentences, alpha, beta, gamma, max_iter):
        self.n_fireflies = n_fireflies
        self.n_documents = n_documents
        self.n_sentences = n_sentences
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.max_iter = max_iter

    def initialize_population(self):
        population = np.zeros((self.n_fireflies, self.n_documents, self.n_sentences))
        for i in range(self.n_fireflies):
            for j in range(self.n_documents):
                selected_sentences = random.sample(range(self.n_sentences), self.n_sentences
// 2)
                population[i][j][selected_sentences] = 1
        return population

    def calculate_fitness(self, population, documents):
        fitness = np.zeros((self.n_fireflies,))
        for i in range(self.n_fireflies):
            summary = np.zeros((self.n_documents, self.n_sentences))
            for j in range(self.n_documents):
                for k in range(self.n_sentences):
                    if population[i][j][k] == 1:
                        summary[j][k] = 1
            fitness[i] = self.objective_function(summary, documents)
        return fitness

    def objective_function(self, summary, documents):
        rouge_scores = []
        for i in range(self.n_documents):
            rouge_scores.append(self.calculate_rouge_score(summary[i], documents[i]))
        return np.mean(rouge_scores)

    def calculate_rouge_score(self, summary, document):
        # code for computing ROUGE score goes here
        pass

    def move_fireflies(self, population, fitness):
        for i in range(self.n_fireflies):
            for j in range(self.n_fireflies):
```

```
            if fitness[i] < fitness[j]:
                r = math.sqrt(np.sum((population[i] - population[j]) ** 2))
                beta_i = self.beta * math.exp(-self.gamma * r ** 2)
                population[i] += beta_i * (population[j] - population[i]) + self.alpha *
(random.random() - 0.5)
                population[i] = np.clip(population[i], 0, 1)
        return population

    def optimize(self, documents):
        population = self.initialize_population()
        fitness = self.calculate_fitness(population, documents)
        best_fitness = np.max(fitness)
        best_summary = population[np.argmax(fitness)]

        for i in range(self.max_iter):
            population = self.move_fireflies(population, fitness)
            fitness = self.calculate_fitness(population, documents)
            if np.max(fitness) > best_fitness:
                best_fitness = np.max(fitness)
                best_summary = population[np.argmax(fitness)]
        return best_summary
```

BACKEND

```
import os
from flask import Flask, flash, request, redirect, url_for, session, jsonify
from werkzeug.utils import secure_filename
from flask_cors import CORS, cross_origin
import logging
import PyPDF2
import fitz
import os
import openai
import torch
import json
from transformers import T5Tokenizer, T5ForConditionalGeneration, T5Config
from rouge import Rouge
import bert_score
from evaluate import load

openai.api_key = "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"
def summarize(text):
    response = openai.Completion.create(
    model="text-davinci-003",
    prompt="Summarize this for a second-grade student:\n\n" + text,
    temperature=0.7,
    max_tokens=1000,
    top_p=1.0,
```

```
    frequency_penalty=0.0,
    presence_penalty=0.0
    )
    return response

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger('HELLO WORLD')



UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'])

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def readPDF(path):
    doc = fitz.open(path)
    text = ""
    for page in doc:
        text+=page.get_text()
    # print(text)
    return text

def summarizer(text):
    model = T5ForConditionalGeneration.from_pretrained('t5-base')
    tokenizer = T5Tokenizer.from_pretrained('t5-base')
    t5_prepared_Text = "summarize: "+ text
    # print ("Original text preprocessed: \n", text)

    tokenized_text = tokenizer.encode(t5_prepared_Text, return_tensors="pt")


    # summmarize
    summary_ids = model.generate(tokenized_text,
                                 num_beams=4,
                                 no_repeat_ngram_size=2,
                                 min_length=75,
                                 max_length=700,
                                 early_stopping=True)

    output = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    print ("\n\nSummarized text: \n",output)
    return output


@app.route('/upload', methods=['POST'])
```

```
def fileUpload():
    print('hello')
    if 'files' not in request.files:
        print('no files')
    file_obj = request.files
    # for f in file_obj:
    #     file = request.files.get(f)
    #     print(file)
    # print(request.files)
    uploaded_files = request.files.getlist('files')
    print(uploaded_files)
    target=os.path.join(UPLOAD_FOLDER,'test_docs')
    if not os.path.isdir(target):
        os.mkdir(target)
    # logger.info("welcome to upload`")
    file = request.files['file']
    print(file)
    filename = secure_filename(file.filename)
    destination="/".join([target, filename])
    file.save(destination)
    session['uploadFilePath']=destination
    text = readPDF(destination)
    # print(text)

    summary = summarizer(text)

    print(summary)
    rouge = Rouge()

    scores = rouge.get_scores(text, summary)
    print(scores)
    bertscore = load("bertscore")
    predictions = summary.split('.')
    references = text.split('.')
    b_score = bertscore.compute(predictions=predictions,
references=references[:len(predictions)], lang="en")
    print(b_score)
    data = {'message': {'summary': summary, 'rouge': scores, 'bert': b_score}}
    response = jsonify(data)
    response.headers.add('Access-Control-Allow-Origin', '*')
    response.headers.add('Access-Control-Allow-Headers', 'Content-Type')
    response.headers.add('Access-Control-Allow-Methods', 'GET, POST, OPTIONS')
    print(response)
    return response

if __name__ == "__main__":
    app.secret_key = os.urandom(24)
    app.run(debug=True,host="0.0.0.0",use_reloader=False)
```

```
    CORS(app, expose_headers='Authorization')
```

FRONTEND

```
import React, { useState } from 'react';
import "../style/FileUpload.css"
import axios from 'axios'

function FileUpload() {
  const [selectedFiles, setSelectedFiles] = useState(null);
  const [ficheros, setFicheros] = useState(null);
  const [filename, setFilename] = useState([]);

  const handleFileSelection = (event) => {
    setFicheros(event.target.files);
    const temp = [];
    for (const iterator of event.target.files) {
      temp.push(iterator.filename);
    }
    setFilename(temp);
    setSelectedFiles(event.target.files);
  };

  // const handleSubmit = async (event) => {
  //   event.preventDefault();

  //   const fileCount = selectedFiles.length;
  //   let fileSize = 0;

  //   for (let i = 0; i < fileCount; i++) {
  //     fileSize += selectedFiles[i].size;
  //   }

  //   alert(`You have selected ${fileCount} files with a total size of ${fileSize} bytes.`);

  // };

  const handleUpload = async (evt) => {
    evt.preventDefault();
    // let formData = new FormData();
    // console.log(selectedFiles)
    // // formData.append('file', selectedFiles);
    // // console.log(selectedFiles.length)
    // for (let i = 0; i < selectedFiles.length; i++) {
    //   formData.append(`file[${i}]`, selectedFiles[i]);
    //   console.log(formData)
    // }
    const formdata = new FormData();
```

```
    filename.forEach((filename,index) => {
      formdata.append("files", ficheros[index], filename);
    });

    // const headers ={
    //   'Content-Type': 'multipart/form-data'
    // }
    const response  = await axios
      .post("http://127.0.0.1:5000/upload", formdata)
      .then((response) => {
        console.log(response.data);
      })
      .catch((error) => {
        console.log(error);
      });
    console.log(response)
  };

  // const handleUpload = (ev) => {
  //   ev.preventDefault();

  //   const data = new FormData();
  //   for (let i = 0; i < this.uploadInput.files.length; i++) {
  //     data.append("file", this.uploadInput.files[i]);
  //   }

  //   fetch("http://localhost:5000/upload", {
  //     method: "POST",
  //     body: data,
  //   }).then((response) => {
  //     response.json().then((res) => {
  //       console.log(res);
  //     });
  //   });
  // }

  return (
    <div style={{}}>
      <form>
        <label for="images" class="drop-container">
          <span class="drop-title">Drop files here</span>
          <input type="file" id="images" accept="pdf/*" multiple required
onChange={handleFileSelection} />
        </label>
        <br/> <br/>
        <button type="submit" class="uploadbutton" onClick={handleUpload}><h3>Upload
Files</h3></button>
      </form>
```

```
    </div>
  );
}

export default FileUpload;
```