Hoje eu tenho um algoritmo   
  
<Modelo algoritmo>  
  
que lê uma entrada de 2 informações, produto e quantidade de uma base em sql,  
  
<Modelo sql>  
  
 e utiliza de um algoritmo de distribuição que permite consultar essa base, lê a ficha técnica do produto e calcula a melhor distribuição de peças levando em consideração que só podemos agrupar peças de tubos iguais e que tenha limite de 6000 mm. Dentre outras tarefas, como mostrar percentual de perda, exportar para pdf e etc.  
  
Foi constatado que precisamos ajustar algumas coisas para ele funcionar de uma forma melhor.   
  
Primeira melhoria:  
  
Precisamos que ele pesquise uma tabela de sql com mais um parâmetro (cl1), então precisamos que seja ajustado.   
  
Segunda Melhoria:  
  
Precisamos que ao invés de observar qual tubo é a peça e fazer a melhor distribuição do conjunto ele primeiramente veja se o parâmetro cl1 está 0 ou 1.   
  
peças com parâmetro 1 não podem ser agrupadas com peças de parâmetro 0, mesmo que tenham a mesma descrição de tubo.   
  
e o limite de peças com o parâmetro 1 é de 5700 mm.   
as peças com parâmetro 0 continuam com limite de 6000 mm.   
  
Me passe um algoritmo que atualize essas necessidades.  
  
  
alg:  
  
import mysql.connector  
from fpdf import FPDF  
import tkinter as tk  
from tkinter import messagebox  
from tkinter import ttk  
  
def get\_product\_data(codigo\_produto):  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="madmax00st1",  
 database="FICHAS\_TECNICAS"  
 )  
 cursor = conn.cursor()  
 query = ("SELECT tipo\_tubo, quantidade, tamanho, cl1 FROM produtos\_v2 WHERE codigo = %s")  
 cursor.execute(query, (codigo\_produto,))  
  
 tubos = {0: {}, 1: {}}  
 for (tipo\_tubo, quantidade, tamanho, cl1) in cursor:  
 if tipo\_tubo not in tubos[cl1]:  
 tubos[cl1][tipo\_tubo] = []  
 tubos[cl1][tipo\_tubo].extend([(tamanho, tamanho + 2)] \* quantidade) # Adicionar 2mm a cada tamanho  
  
 cursor.close()  
 conn.close()  
 return tubos  
  
def cortar\_barras(tubos):  
 barras = []  
 tipo\_contador = {0: {}, 1: {}}  
  
 for cl1, tipos\_tubos in tubos.items():  
 limite\_barra = 5700 if cl1 == 1 else 6000  
  
 for tubo\_tipo, tamanhos in tipos\_tubos.items():  
 tamanhos.sort(key=lambda x: x[1], reverse=True) # Ordenar tamanhos em ordem decrescente pelo tamanho com 2mm  
  
 tipo\_contador[cl1][tubo\_tipo] = tipo\_contador[cl1].get(tubo\_tipo, 0) + 1  
  
 while tamanhos:  
 barra\_atual = []  
 comprimento\_restante = limite\_barra  
  
 for tamanho\_real, tamanho\_com\_folga in tamanhos[:]:  
 if tamanho\_com\_folga <= comprimento\_restante:  
 barra\_atual.append(tamanho\_real)  
 comprimento\_restante -= tamanho\_com\_folga  
 tamanhos.remove((tamanho\_real, tamanho\_com\_folga))  
  
 barras.append((tubo\_tipo, barra\_atual, comprimento\_restante, tipo\_contador[cl1][tubo\_tipo], cl1))  
 tipo\_contador[cl1][tubo\_tipo] += 1  
  
 return barras  
  
def calcular\_percentual\_sobra(comprimento\_barra, sobra):  
 return (sobra / comprimento\_barra) \* 100  
  
def gerar\_pdf(barras\_cortadas, codigo\_produto, quantidade, numero\_pedido, item):  
 pdf = FPDF()  
 pdf.add\_page()  
 pdf.set\_font("Arial", size=12)  
  
 # Título  
 pdf.cell(200, 10, f"Relatório de Cortes para o Produto {codigo\_produto} (Quantidade: {quantidade})", ln=True, align='C')  
 pdf.ln(10)  
  
 desperdicio\_por\_tipo = {}  
 cortes\_por\_tipo = {}  
  
 for idx, (tipo, cortes, sobra, barra\_id, cl1) in enumerate(barras\_cortadas):  
 comprimento\_barra = 5700 if cl1 == 1 else 6000  
 percentual\_sobra = calcular\_percentual\_sobra(comprimento\_barra, sobra)  
 # Detalhes da barra  
 detalhes\_barra = f"Barra {barra\_id} ({tipo}, CL1={cl1}): Cortes = {cortes}, Sobra = {sobra} mm ({percentual\_sobra:.2f}%)"  
 pdf.multi\_cell(0, 10, detalhes\_barra, border=0)  
  
 # Acumular dados para cálculo da média de desperdício  
 if tipo not in desperdicio\_por\_tipo:  
 desperdicio\_por\_tipo[tipo] = 0  
 cortes\_por\_tipo[tipo] = 0  
 desperdicio\_por\_tipo[tipo] += sobra  
 cortes\_por\_tipo[tipo] += 1  
  
 pdf.ln(10)  
  
 # Calcular e imprimir a média de desperdício por tipo de tubo  
 for tipo, desperdicio\_total in desperdicio\_por\_tipo.items():  
 media\_desperdicio = desperdicio\_total / cortes\_por\_tipo[tipo]  
 comprimento\_barra = 5700 if cl1 == 1 else 6000  
 percentual\_media\_desperdicio = calcular\_percentual\_sobra(comprimento\_barra, media\_desperdicio)  
 detalhes\_media = f"Média de desperdício para {tipo}: {media\_desperdicio:.2f} mm ({percentual\_media\_desperdicio:.2f}%)"  
 pdf.multi\_cell(0, 10, detalhes\_media, border=0)  
  
 # Nome do arquivo PDF  
 nome\_arquivo = f"{numero\_pedido}\_{item}.pdf"  
 pdf.output(nome\_arquivo)  
 return nome\_arquivo  
  
def main():  
 # Função para processar os dados de entrada  
 def processar\_dados():  
 codigo\_produto = codigo\_entry.get()  
 quantidade = int(quantidade\_entry.get())  
 numero\_pedido = pedido\_entry.get()  
 item = item\_entry.get()  
  
 # Buscar dados do produto no banco de dados  
 produto\_data = get\_product\_data(codigo\_produto)  
  
 # Multiplicar tamanhos pelo número de produtos a serem produzidos  
 tubos = {0: {}, 1: {}}  
 for cl1, tipos in produto\_data.items():  
 for tipo, tamanhos in tipos.items():  
 if tipo not in tubos[cl1]:  
 tubos[cl1][tipo] = []  
 tubos[cl1][tipo].extend(tamanhos \* quantidade)  
  
 # Calcular a melhor combinação de cortes  
 barras\_cortadas = cortar\_barras(tubos)  
  
 # Gerar PDF com o resultado  
 nome\_arquivo = gerar\_pdf(barras\_cortadas, codigo\_produto, quantidade, numero\_pedido, item)  
 messagebox.showinfo("Sucesso", f"PDF gerado com sucesso: {nome\_arquivo}")  
  
 # Criar a janela principal  
 root = tk.Tk()  
 root.title("Amplio Móveis Exteriores")  
 root.geometry("400x300")  
  
 # Configurar o grid  
 root.columnconfigure(0, weight=1)  
 root.columnconfigure(1, weight=3)  
  
 # Título  
 title\_label = tk.Label(root, text="Novo cálculo produtivo", font=("Helvetica", 16))  
 title\_label.grid(row=0, column=0, columnspan=2, pady=10)  
  
 # Campos de entrada  
 tk.Label(root, text="Código do Produto:").grid(row=1, column=0, sticky=tk.E, padx=10, pady=5)  
 codigo\_entry = tk.Entry(root)  
 codigo\_entry.grid(row=1, column=1, pady=5, sticky=tk.W+tk.E, padx=(0, 20))  
  
 tk.Label(root, text="Quantidade:").grid(row=2, column=0, sticky=tk.E, padx=10, pady=5)  
 quantidade\_entry = tk.Entry(root)  
 quantidade\_entry.grid(row=2, column=1, pady=5, sticky=tk.W+tk.E, padx=(0, 20))  
  
 tk.Label(root, text="Número do Pedido:").grid(row=3, column=0, sticky=tk.E, padx=10, pady=5)  
 pedido\_entry = tk.Entry(root)  
 pedido\_entry.grid(row=3, column=1, pady=5, sticky=tk.W+tk.E, padx=(0, 20))  
  
 tk.Label(root, text="Item:").grid(row=4, column=0, sticky=tk.E, padx=10, pady=5)  
 item\_entry = tk.Entry(root)  
 item\_entry.grid(row=4, column=1, pady=5, sticky=tk.W+tk.E, padx=(0, 20))  
  
 # Botões  
 button\_frame = tk.Frame(root)  
 button\_frame.grid(row=5, column=0, columnspan=2, pady=10)  
  
 tk.Button(button\_frame, text="Cancelar", command=root.quit).pack(side=tk.LEFT, padx=5)  
 tk.Button(button\_frame, text="Confirmar", command=processar\_dados).pack(side=tk.LEFT, padx=5)  
  
 # Iniciar o loop principal da interface gráfica  
 root.mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()