P.PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Trabalho Prático de

Matemática Discreta

2024/2025

Trabalho Elaborado por:

Grupo W

8240696 Diogo Fernado Águia Costa (8240696@estg.ipp.pt)

8240352 Guilherme Ribeiro Gomes (8240352@estg.ipp.pt)

8240779 Vasco Meireles Leal (8240779@estg.ipp.pt)

8240308 João Filipe Oliveira (8240308@estg.ipp.pt)

Curso de:

LSIRC - Licenciatura em Segurança informatica em redes de computadores.

Docentes:

Eliana Costa e Silva (eos@estg.ipp.pt)

Isabel Cristina Duarte (icd@estg.ipp.pt)

Felgueiras, 5 de maio de 2025

Conteúdo

Lista de Figuras

Lista de Tabelas

Problema 1 — Operações com Conjuntos

1.1 Introdução

Este capítulo tem como objetivo resolver e explicar, de forma clara e didática, o **Problema 1**, que envolve operações fundamentais e compostas com conjuntos. Utilizou-se o software *SciLab* para validar os resultados obtidos manualmente, promovendo a consolidação da teoria através da prática computacional.

1.2 Definição dos Conjuntos

O **conjunto universo** U é definido como o conjunto dos 20 primeiros números naturais positivos consecutivos:

$$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$$

Definimos dois subconjuntos A e B com as seguintes restrições:

- $5 \leq |A| < 10$
- $|B| > 15$
- $A \neq B$
- $B \neq U$

Assim, escolhemos:

$$A = \{1, 3, 5, 7, 9, 11, 13\} \quad (|A| = 7)$$

$$B = \{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\} \quad (|B| = 16)$$

1.3 Operações Realizadas

Nesta seção, são descritas as operações efetuadas entre os conjuntos A, B e U, acompanhadas dos respectivos resultados e explicações.

a) **Cardinalidade:** Refere-se ao número de elementos de um conjunto.

$$|A| = 7 \quad \text{(A tem 7 elementos)}, \quad |B| = 16 \quad \text{(B tem 16 elementos)}$$

b) **Complemento de B em relação a U:** São todos os elementos do conjunto universo U que não pertencem ao conjunto B. Denotamos como $\overline{B}$ ou $U - B$.

$$\overline{B} = U \setminus B = \{1, 2, 3, 4\}$$

c) **União ($A \cup B$):** É o conjunto formado por todos os elementos que pertencem a A, a B, ou a ambos.

$$A \cup B = \{1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$$

d) **Interseção ($A \cap B$):** É o conjunto dos elementos que pertencem simultaneamente a A e a B.

$$A \cap B = \{5, 7, 9, 11, 13\}$$

e) **Diferença ($B - A$):** São os elementos que pertencem a B, mas não pertencem a A.

$$B - A = \{6, 8, 10, 12, 14, 15, 16, 17, 18, 19, 20\}$$

f) **Diferença Simétrica ($A \oplus B$):** São os elementos que pertencem a apenas um dos conjuntos, mas não a ambos. Ou seja:

$$A \oplus B = (A \cup B) - (A \cap B) = \{1, 3, 6, 8, 10, 12, 14, 15, 16, 17, 18, 19, 20\}$$

g) **Operação composta:** $\overline{A \oplus B} \cup (A - B)$:

- $A \oplus B$: já foi calculado no item anterior.
- $\overline{A \oplus B}$: são os elementos do universo que não estão em $A \oplus B$.
- $A - B$: são os elementos que estão em A, mas não estão em B.
- A união final junta todos os elementos que pertencem a pelo menos um dos dois conjuntos anteriores.

Resultado:

$$\overline{A \oplus B} = \{2, 4, 5, 7, 9, 11, 13\}, \quad A - B = \{1, 3\}$$

$$\overline{A \oplus B} \cup (A - B) = \{1, 2, 3, 4, 5, 7, 9, 11, 13\}$$

h) **Produto Cartesiano $B \times A$:**

O produto cartesiano $B \times A$ consiste em formar pares ordenados (b, a), onde $b \in B$ e $a \in A$. Com $|B| = 16$ e $|A| = 7$, o total de pares é:

$$|B \times A| = 16 \times 7 = 112 \text{ pares}$$

A matriz resultante pode ser representada como uma tabela com 112 linhas (cada par) e 2 colunas:

$$\begin{bmatrix} (5,1) \\ (5,3) \\ (5,5) \\ \vdots \\ (6,1) \\ (6,3) \\ \vdots \\ (20,13) \end{bmatrix}$$

Os primeiros pares da matriz são:

b	a
5	1
5	3
5	5
5	7
5	9
5	11
5	13
6	1
6	3
⋮	⋮
20	13

Cada linha representa um par ordenado de $B \times A$. A matriz pode ser facilmente gerada em SciLab ou outro software para visualização completa.

i) **Produto Cartesiano Triplo A^3:**

O produto A^3 é o conjunto de todos os trios ordenados (a_1, a_2, a_3), onde cada elemento pertence ao conjunto A. Com $|A| = 7$, o total de trios é:

$$|A^3| = 7^3 = 343 \text{ trios}$$

A matriz resultante terá 343 linhas e 3 colunas, uma para cada posição do trio:

$$\begin{bmatrix} (1,1,1) \\ (1,1,3) \\ (1,1,5) \\ \vdots \\ (13,13,13) \end{bmatrix}$$

Exemplos das primeiras linhas:

a_1	a_2	a_3
1	1	1
1	1	3
1	1	5
1	1	7
1	1	9
1	1	11
1	1	13
1	3	1
⋮	⋮	⋮
13	13	13

A matriz segue o mesmo princípio que uma contagem com três dígitos, variando de forma sistemática. Cada combinação representa um caminho no espaço tridimensional de $A \times A \times A$.

1.4 Resolução Manual

Esta anexado com este relatorio a resolução manual do Problema 1, com o nome do ficheiro **ResoluçãoPergunta1**.

1.5 Validação Computacional com SciLab

Todas as operações foram validadas por meio de um script no software *SciLab*, nomeado **ScripEx01.sci**, que encontra-se anexo a este relatório. O SciLab foi utilizado para confirmar automaticamente os resultados obtidos manualmente, garantindo precisão e reforçando a aplicação prática da teoria.

1.6 Conclusão

A resolução do Problema 1 permitiu aplicar e revisar conceitos fundamentais da Teoria dos Conjuntos, como união, interseção, complemento, diferença, diferença simétrica e produto cartesiano. O uso do *SciLab* fortaleceu o vínculo entre teoria e prática computacional, tornando o aprendizado mais concreto e visual. A análise manual, aliada à computacional, assegurou uma compreensão completa das operações realizadas.

Problema 2 — Somatórios e Produtórios

2.1 Introdução

Considere β como sendo o último algarismo do número de estudante de um dos elementos do grupo. Aluno: 8240308, ou seja, $\beta = 8$. Escolha um número natural $n \in \mathbb{N}$ tal que:

$$50 + \beta < 2n < 100 - \beta$$

Neste capítulo, resolve-se computacionalmente, com auxílio do *Scilab*, o valor de cada uma das expressões seguintes:

a) Somatório com base racional:

$$\sum_{j=\beta+2}^{n} \left(\frac{-2\beta - 1}{5} \right)^{j}$$

b) Produtório com conjunto indexado:

$$\prod_{i \in C} \left(\frac{\beta + 1}{i - 1} \right)^{4}, \quad \text{com } C = \{5m \in \mathbb{Z} : m = 1, \ldots, M\}, \quad M = \min\left(5 + \beta, \left\lceil \frac{100}{\beta + 1} \right\rceil \right)$$

c) Produtório com somatório interno:

$$\prod_{k=1}^{n-15} \left(3 \times \sum_{j=n-5}^{n} \left(\left\lfloor 1 + \frac{j+k}{200} \right\rfloor - \left\lfloor \frac{6!}{\beta + 1} \right\rfloor \right) \right)$$

2.2 Resolução Detalhada

Parâmetros Considerados

Para a resolução das expressões propostas, foram fixados os seguintes valores:

- **Último algarismo do número de estudante:** $\beta = 8$
- **Valor escolhido para** n: $n = 32$, pois satisfaz a desigualdade imposta:

$$50 + \beta = 58 < 2n = 64 < 100 - \beta = 92$$

2.2.1 Alínea (a): Somatório Geométrico

A expressão fornecida corresponde a uma progressão geométrica:

$$S = \sum_{j=\beta+2}^{n} \left(\frac{-2\beta - 1}{5}\right)^j = \sum_{j=10}^{32} \left(\frac{-17}{5}\right)^j$$

- **Razão geométrica:** $r = \dfrac{-17}{5}$
- **Número de termos:** $n - (\beta + 2) + 1 = 32 - 10 + 1 = 23$

Apesar de existir uma fórmula fechada para somatórios de progressões geométricas, a elevada magnitude da razão inviabiliza o uso direto devido à instabilidade numérica. Por isso, optou-se pelo uso de *Scilab* para computação precisa.

Código *Scilab*:

```
base = (-2*beta - 1)/5;
S = 0;
for j = beta + 2 : n
    S = S + base^j;
end
```

Resultado:

$$\boxed{S \approx 7{,}859 \times 10^{16}}$$

2.2.2 Alínea (b): Produtório sobre um Conjunto Indexado

O produtório é definido sobre um conjunto C de múltiplos de 5, até um limite calculado com base em β:

$$M = \min\left(5 + \beta, \left\lceil \frac{100}{\beta + 1} \right\rceil\right) = \min(13, \lceil 11{,}11 \rceil) = 13$$

$$C = \{5m \in \mathbb{Z} : m = 1, \ldots, 13\} = \{5, 10, 15, \ldots, 65\}$$

Cada termo do produtório tem a forma:

$$\left(\frac{\beta + 1}{i - 1}\right)^4 = \left(\frac{9}{i - 1}\right)^4$$

Código *Scilab*:

```
M = min(5 + beta, ceil(100 / (beta + 1)));
C = 5 * (1:M);
P = 1;
for i = 1:length(C)
    termo = ((beta + 1) / (C(i) - 1))^4;
    P = P * termo;
end
```

Resultado:

$$\boxed{P \approx 2{,}008 \times 10^{-11}}$$

Este resultado evidencia a rápida diminuição do produtório devido à presença de potências elevadas com denominadores crescentes.

2.2.3 Alínea (c): Produtório com Somatório Interno

Esta expressão envolve um somatório interno dependente de dois índices j e k, aninhado dentro de um produtório. O termo interno depende da subtração entre duas partes:

$$\left\lfloor 1 + \frac{j+k}{200} \right\rfloor - \left\lfloor \frac{6!}{\beta + 1} \right\rfloor$$

- **Valor fixo de referência:**

$$\left\lfloor \frac{6!}{\beta + 1} \right\rfloor = \left\lfloor \frac{720}{9} \right\rfloor = 80$$

- **Intervalos:**

$$k \in [1, n-15] = [1, 17], \quad j \in [n-5, n] = [27, 32]$$

- Para cada k, o somatório interno é:

$$\sum_{j=27}^{32} \left(\left\lfloor 1 + \frac{j+k}{200} \right\rfloor - 80 \right)$$

  Como $\frac{j+k}{200} < 1$ para todos os j, k considerados, temos:

$$\left\lfloor 1 + \frac{j+k}{200} \right\rfloor = 1 \Rightarrow \text{cada termo é } 1 - 80 = -79 \Rightarrow \sum = 6 \times (-79) = -474$$

O termo do produtório para cada k é então:

$$3 \cdot (-474) = -1422 \quad \Rightarrow \quad \prod_{k=1}^{17} (-1422) = (-1422)^{17}$$

Código *Scilab*:

```
divF = floor(720 / (beta + 1));
PROD = 1;
for k = 1 : n - 15
    somaInterna = 0;
    for j = n - 5 : n
        termo = floor(1 + (j + k)/200) - divF;
        somaInterna = somaInterna + termo;
    end
    PROD = PROD * (3 * somaInterna);
end
```

Resultado:

$$\boxed{PROD \approx -3{,}975 \times 10^{53}}$$

O sinal negativo e a elevada ordem de grandeza do resultado ilustram o impacto da repetição de termos altamente negativos com potência ímpar.

2.3 Validação dos Resultados

Com o objetivo de validar os resultados obtidos nas alíneas anteriores, foi desenvolvido um ficheiro de script com o nome **ScriptEx02**. Este ficheiro encontra-se anexado ao presente trabalho e contém o código necessário para a verificação e validação dos resultados apresentados.

O script implementa os métodos e procedimentos discutidos ao longo do exercício, permitindo uma análise automatizada e rigorosa da correção das soluções obtidas. A execução deste ficheiro permite ao utilizador confirmar a exatidão dos cálculos e a consistência dos resultados, contribuindo para a fiabilidade do trabalho desenvolvido.

2.4 Conclusão

A análise detalhada dos somatórios e produtórios definidos neste problema permitiu explorar diferentes conceitos fundamentais da matemática discreta, com apoio computacional para garantir a precisão dos resultados. A seguir, são destacadas as principais observações:

- **Alínea (a):** A progressão geométrica com razão negativa e módulo elevado ($r = -\frac{17}{5}$) produziu um somatório de grande magnitude. Isso ocorre porque os termos crescem exponencialmente à medida que o expoente aumenta. A alternância de sinais imposta pela razão negativa não foi suficiente para cancelar esse crescimento, resultando em um valor final da ordem de 10^{16}.
- **Alínea (b):** O produtório envolveu razões fracionárias elevadas à quarta potência, com denominadores crescentes. Essa estrutura levou a uma rápida diminuição dos termos multiplicativos, causando uma redução exponencial do resultado final. O valor extremamente pequeno ($\approx 2{,}008 \times 10^{-11}$) ilustra o impacto de potências altas aplicadas a frações menores que 1.
- **Alínea (c):** Esta alínea combinou somatórios internos constantes e negativos com um produtório externo. Cada termo multiplicado foi igual a -1422, e sua repetição 17 vezes resultou num valor altamente negativo e de grande magnitude absoluta, da ordem de 10^{53}. O uso de uma potência ímpar preservou o sinal negativo, enquanto a estrutura repetitiva amplificou o módulo do resultado.

Além dos aspectos computacionais, esta resolução reforçou a importância da análise simbólica prévia para evitar erros conceituais e simplificar os cálculos antes da implementação em código. O *Scilab* provou ser uma ferramenta eficaz na avaliação precisa de expressões complexas envolvendo somatórios e produtórios com condições personalizadas.

Em suma, este exercício promoveu uma abordagem completa e integrada entre teoria matemática, lógica de programação e interpretação numérica, aspectos essenciais na formação analítica de um estudante de ciências exatas.

Problema 3 - Aplicação do Algoritmo de Dijkstra em Grafos

3.1 Introdução

A integração de robôs na indústria tem transformado profundamente o sector produtivo, promovendo ganhos substanciais em eficiência, inovação e sustentabilidade.

No âmbito da Indústria 4.0, os robôs desempenham um papel central nas fábricas inteligentes, aliando automação, Internet das Coisas (IoT) e Inteligência Artificial para a execução de operações autónomas e precisas. Com a evolução para a Indústria 5.0, o foco desloca-se para a colaboração sinérgica entre humanos e robôs, explorando o potencial criativo e estratégico dos primeiros e a consistência operacional dos segundos.

Neste contexto, propõe-se o planeamento da trajectória de um braço robótico, cujo *end-effector* deve deslocar-se de um ponto inicial para um ponto final, evitando colisões e minimizando a distância total percorrida.

O problema a resolver contempla os seguintes objectivos principais:

- Gerar 20 pontos tridimensionais (coordenadas x, y, z), assegurando que todos se encontram a uma distância máxima de 27 cm do ponto de origem $(0, 0, 0)$.
- Construir um grafo não orientado, ligando pares de pontos cuja distância euclidiana seja inferior ou igual a 5,0 cm, limitando o total de arestas a um máximo de 100.
- Aplicar o algoritmo de Dijkstra para determinar o caminho de menor custo (menor distância total) entre dois pontos, designados por β e σ, definidos com base nos números de estudante do grupo.
- Garantir que a distância total do percurso obtido seja, no mínimo, de 50 cm.

3.2 Metodologia

A resolução do problema foi desenvolvida em linguagem de programação C, estando o código-fonte anexo a este relatório sob o nome `CodigoProblema3`. O processo seguiu uma sequência lógica e estruturada, que incluiu:

1. Geração manual de 20 pontos tridimensionais, com base nos critérios geométricos estabelecidos.
2. Cálculo da matriz de adjacência, conectando apenas os pares de pontos cuja distância seja inferior ou igual a 5,0 cm.
3. Construção do grafo, respeitando o limite máximo de 100 arestas.
4. Implementação do algoritmo de Dijkstra para determinar o caminho de menor custo entre os pontos β e σ, conforme definidos pelas regras do enunciado.

3.2.1 Introdução ao Código

Este documento descreve um algoritmo em C para encontrar caminhos entre pontos no espaço tridimensional. O programa é baseado num grafo de pontos 3D conectados por arestas de comprimento limitado. O objectivo é encontrar caminhos entre pontos, com a condição de que o comprimento do caminho seja maior do que um valor mínimo definido.

3.2.2 Estruturas de Dados

O código utiliza várias estruturas para representar os dados do problema.

Estrutura de Ponto 3D

A estrutura `Ponto3D` é usada para representar pontos no espaço 3D. Cada ponto possui três coordenadas: x, y e z.

```
typedef struct {
    float x, y, z;
} Ponto3D;
```

Estrutura de Aresta

A estrutura `Aresta` representa uma aresta entre dois pontos. Cada aresta contém a origem e o destino dos pontos conectados, além de um peso que indica o comprimento da aresta.

```
typedef struct {
    int origem, destino;
    float peso;
} Aresta;
```

Estrutura de Caminho Encontrado

A estrutura `CaminhoEncontrado` armazena um caminho válido que foi encontrado, ou seja, uma sequência de pontos conectados por arestas cujo comprimento total é maior do que um limite definido.

```
typedef struct {
    int pontos[NUM_PONTOS];
    int tamanho;
    double comprimento_total;
} CaminhoEncontrado;
```

Estrutura de Lista de Arestas

A estrutura `ListaAdj` representa um grafo utilizando listas de adjacência. Cada ponto tem uma lista de arestas conectadas a ele, onde cada aresta contém o destino e o peso.

```
typedef struct {
    ArestaGrafo arestas[NUM_PONTOS];
    int num_arestas;
} ListaAdj;
```

3.2.3 Funções

Agora vamos detalhar as principais funções do programa.

Função para Calcular Distância

A função `distancia` calcula a distância euclidiana entre dois pontos no espaço 3D. A fórmula utilizada é:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

onde x_1, y_1, z_1 e x_2, y_2, z_2 são as coordenadas dos dois pontos.

```
float distancia(Ponto3D a, Ponto3D b) {
    return sqrtf((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y) + (a.z - b.z)*(a.z - b.z));
}
```

Função para Adicionar Aresta

A função `adicionar_aresta` adiciona uma aresta entre dois pontos. A aresta é bidireccional, ou seja, se o ponto A está conectado ao ponto B, o ponto B também está conectado ao ponto A.

```
void adicionar_aresta(int origem, int destino, float peso) {
    if (grafo[origem].num_arestas < NUM_PONTOS)
        grafo[origem].arestas[grafo[origem].num_arestas++] = (ArestaGrafo){destino, peso};
    if (grafo[destino].num_arestas < NUM_PONTOS)
        grafo[destino].arestas[grafo[destino].num_arestas++] = (ArestaGrafo){origem, peso};
}
```

Função para Guardar Caminhos

A função `salvar_caminho` armazena um caminho válido no array `caminhos` se o comprimento total do caminho for maior ou igual ao valor mínimo $LIMITE$. O caminho é guardado juntamente com o seu comprimento total e os pontos envolvidos.

```
void salvar_caminho(int caminho[], int profundidade, double soma) {
    if (soma >= LIMITE && total_caminhos < MAX_CAMINHOS) {
        CaminhoEncontrado *c = &caminhos[total_caminhos++];
        c->tamanho = profundidade + 1;
        c->comprimento_total = soma;
        for (int i = 0; i <= profundidade; i++) {
            c->pontos[i] = caminho[i];
        }
    }
}
```

Função de Busca Recursiva

A função `buscar_caminhos` utiliza uma busca em profundidade (DFS) para explorar todos os caminhos possíveis de um ponto inicial até um ponto de destino. Ela mantém o controlo de quais pontos já foram visitados, para evitar ciclos no caminho.

```
void buscar_caminhos(int atual, int destino, int caminho[],
int profundidade, double soma, int visitados[]) {
    caminho[profundidade] = atual;

    // Se o ponto atual for o destino, guardamos o caminho encontrado
    if (atual == destino) {
        salvar_caminho(caminho, profundidade, soma);
        return;
    }
```

```
    visitados[atual] = 1;

    // Explorar vizinhos do ponto atual
    for (int i = 0; i < grafo[atual].num_arestas; i++) {
        int vizinho = grafo[atual].arestas[i].destino;
        double peso = grafo[atual].arestas[i].peso;

        // Se o vizinho não foi visitado, fazemos a recursão
        if (!visitados[vizinho]) {
            buscar_caminhos(vizinho, destino, caminho, profundidade + 1, soma + peso, visitados);
        }
    }

    visitados[atual] = 0; // Backtracking
}
```

Funções de Comparação

As funções `comparar_caminhos` e `comparar_arestas` são usadas para ordenar os caminhos e as arestas, respectivamente. Elas são passadas como argumentos para a função `qsort`, que realiza a ordenação.

```
int comparar_caminhos(const void *a, const void *b) {
    CaminhoEncontrado *c1 = (CaminhoEncontrado *)a;
    CaminhoEncontrado *c2 = (CaminhoEncontrado *)b;
    if (c1->comprimento_total < c2->comprimento_total) return -1;
    else if (c1->comprimento_total > c2->comprimento_total) return 1;
    else return 0;
}

int comparar_arestas(const void *a, const void *b) {
    Aresta *ar1 = (Aresta *)a;
    Aresta *ar2 = (Aresta *)b;
    if (ar1->peso < ar2->peso) return -1;
    else if (ar1->peso > ar2->peso) return 1;
    else return 0;
}
```

3.2.4 Fluxo do Programa

O programa inicia calculando as distâncias entre todos os pares de pontos e construindo um grafo com arestas conectando pontos próximos (distância menor que $LIMIAR$). Em seguida, o programa busca caminhos entre dois pontos definidos (origem e destino), e os caminhos válidos são guardados e ordenados por comprimento total.

3.2.5 Em Resumo

O algoritmo é eficiente para encontrar todos os caminhos possíveis entre dois pontos num grafo tridimensional, respeitando um limite de comprimento. A utilização de busca em profundidade e backtracking garante que todas as possibilidades sejam exploradas.

3.2.6 Geração dos Pontos Tridimensionais

Foram definidos manualmente 20 pontos tridimensionais, garantindo que a distância entre cada ponto e a origem fosse inferior ou igual a 27 cm. A Tabela 3.1 apresenta as coordenadas e respectivas distâncias à origem, calculadas pela fórmula:

$$d = \sqrt{x^2 + y^2 + z^2}$$

Tabela 3.1: Coordenadas dos pontos e distância à origem

Ponto	**Coordenadas** (x, y, z)	**Distância à origem (cm)**
1	(0, 4, 0)	4,00
2	(1, 4, 0)	4,12
3	(2, 4, 0)	4,47
4	(4, 2, 0)	4,47
5	(1, 2, 1)	2,45
6	(2, 1, 0)	2,24
7	(3, 0, 0)	3,00
8	(0, 3, 1)	3,16
9	(1, 1, 1)	1,73
10	(2, 2, 0)	2,83
11	(3, 1, 1)	3,32
12	(4, 3, 0)	5,00
13	(2, 2, 2)	3,46
14	(3, 3, 1)	4,36
15	(0, 0, 3)	3,00
16	(1, 1, 3)	3,32
17	(2, 0, 4)	4,47
18	(0, 4, 3)	5,00
19	(3, 0, 3)	4,24
20	(3, 3, 3)	5,20

3.3 Resultados

3.3.1 Construção da Matriz de Adjacência

Foi implementado um algoritmo para calcular a distância euclidiana entre cada par de pontos e construir uma matriz de adjacência representando as conexões válidas (com distância $\leq$ 5,0 cm). O número de arestas foi limitado a 100.

A distância entre dois pontos $P_i = (x_i, y_i, z_i)$ e $P_j = (x_j, y_j, z_j)$ é dada por:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

3.3.2 Matriz de Adjacência

Foi construída a matriz de adjacência (Figura: 3.1) representando as ligações entre os pontos segundo o critério de distância, limitado a 100 arestas, retirada após a execução do código.

Matriz de Adjacência:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1
2	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1
3	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1
4	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1
5	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	0
8	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
12	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
15	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
16	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
20	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 3.1: Output da matriz de adjacência dos pontos.

3.3.3 Arestas do Grafo

As arestas criadas, ordenadas por comprimento, estão apresentadas nas tabelas seguintes,Tabela 3.2 e Tabela 3.3

Tabela 3.2: Arestas criadas (parte 1).

Ponto A	Ponto B	Distância (cm)	Ponto A	Ponto B	Distância (cm)
1	2	1,00	6	9	1,41
6	10	1,00	4	14	1,73
5	9	1,00	3	14	1,73
4	12	1,00	5	6	1,73
2	3	1,00	4	11	1,73
6	7	1,41	2	8	1,73
6	11	1,41	4	10	2,00
5	13	1,41	1	3	2,00
7	11	1,41	3	10	2,00
5	10	1,41	2	10	2,24
1	8	1,41	2	5	2,24
5	8	1,41	3	12	2,24
4	6	2,24	7	14	3,16
4	7	2,24	3	11	3,32
5	11	2,24	3	9	3,32
5	14	2,24	3	20	3,32
5	16	2,24	4	9	3,32
6	13	2,24	4	20	3,32
7	10	2,24	1	14	3,32
3	8	2,45	5	12	3,32
2	14	2,45	1	9	3,32
3	5	2,45	6	19	3,32
7	9	2,45	1	13	3,46
6	14	2,45	5	19	3,46
1	5	2,45	3	18	3,61
4	13	2,83	2	4	3,61
3	4	2,83	1	6	3,61
6	12	2,83	6	20	3,74
1	10	2,83	5	17	3,74
3	13	2,83	4	19	3,74
7	13	3,00	7	16	3,74
2	13	3,00	2	20	3,74
7	19	3,00	2	11	3,74
6	8	3,00	6	15	3,74
3	6	3,00	7	17	4,12
1	18	3,00	3	7	4,12
5	20	3,00	1	12	4,12

Tabela 3.3: Arestas criadas (parte 2).

Ponto A	Ponto B	Distância (cm)	Ponto A	Ponto B	Distância (cm)
5	7	3,00	6	17	4,12
5	18	3,00	7	15	4,24
5	15	3,00	2	16	4,24
6	16	3,16	4	8	4,24
2	18	3,16	4	16	4,36
4	5	3,16	7	8	4,36
7	12	3,16	1	20	4,36
2	9	3,16	1	16	4,36
2	6	3,16	3	16	4,36
2	12	3,16	1	11	4,36
7	14	3,16	1	4	4,47
2	7	4,47	6	18	4,69
4	17	4,90	1	7	5,00
1	15	5,00			

3.3.4 Caminho Encontrado

Aplicando o algoritmo de Dijkstra, determinou-se o caminho de menor custo entre os pontos $\beta = 6$ e $\sigma = 10$, atribuídos com base nos Números de Aluno 8240696 e 8240352, respetivamente. A escolha seguiu as regras estabelecidas, sendo 52 superior a 20.

Foram encontrados dois percursos com o mesmo custo total, ambos satisfazendo a distância mínima exigida de 50 cm:

- **Opção 1:** $6 \to 8 \to 4 \to 16 \to 3 \to 18 \to 5 \to 17 \to 7 \to 15 \to 1 \to 11 \to 2 \to 10$ (50,01 cm)
- **Opção 2:** $6 \to 8 \to 4 \to 16 \to 3 \to 18 \to 5 \to 17 \to 7 \to 15 \to 1 \to 20 \to 2 \to 10$ (50,01 cm)

```
Total de caminhos encontrados: 20

Caminho 1 (50.01 cm): 6 -> 8 -> 4 -> 16 -> 3 -> 18 -> 5 -> 17 -> 7 -> 15 -> 1 -> 11 -> 2 -> 10
Caminho 2 (50.01 cm): 6 -> 8 -> 4 -> 16 -> 3 -> 18 -> 5 -> 17 -> 7 -> 15 -> 1 -> 20 -> 2 -> 10
Caminho 3 (50.07 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 20 -> 2 -> 11 -> 1 -> 15 -> 7 -> 16 -> 3 -> 10
Caminho 4 (50.10 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 19 -> 7 -> 15 -> 1 -> 20 -> 3 -> 16 -> 2 -> 10
Caminho 5 (50.10 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 19 -> 7 -> 15 -> 1 -> 11 -> 3 -> 16 -> 2 -> 10
Caminho 6 (50.12 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 12 -> 2 -> 16 -> 3 -> 20 -> 1 -> 15 -> 7 -> 10
Caminho 7 (50.12 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 12 -> 2 -> 16 -> 3 -> 11 -> 1 -> 15 -> 7 -> 10
Caminho 8 (50.12 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 12 -> 7 -> 15 -> 1 -> 11 -> 3 -> 16 -> 2 -> 10
Caminho 9 (50.12 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 12 -> 7 -> 15 -> 1 -> 20 -> 3 -> 16 -> 2 -> 10
Caminho 10 (50.17 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 18 -> 3 -> 16 -> 7 -> 15 -> 1 -> 11 -> 2 -> 10
Caminho 11 (50.17 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 18 -> 3 -> 16 -> 7 -> 15 -> 1 -> 20 -> 2 -> 10
Caminho 12 (50.23 cm): 6 -> 8 -> 4 -> 17 -> 7 -> 15 -> 1 -> 11 -> 2 -> 16 -> 3 -> 18 -> 5 -> 10
Caminho 13 (50.29 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 19 -> 7 -> 15 -> 1 -> 20 -> 2 -> 16 -> 3 -> 10
Caminho 14 (50.29 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 19 -> 7 -> 15 -> 1 -> 11 -> 2 -> 16 -> 3 -> 10
Caminho 15 (50.31 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 12 -> 7 -> 15 -> 1 -> 20 -> 2 -> 16 -> 3 -> 10
Caminho 16 (50.31 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 12 -> 7 -> 15 -> 1 -> 11 -> 2 -> 16 -> 3 -> 10
Caminho 17 (50.38 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 20 -> 2 -> 16 -> 3 -> 11 -> 1 -> 15 -> 7 -> 10
Caminho 18 (50.38 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 20 -> 3 -> 16 -> 2 -> 11 -> 1 -> 15 -> 7 -> 10
Caminho 19 (50.67 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 18 -> 3 -> 16 -> 2 -> 20 -> 1 -> 15 -> 7 -> 10
Caminho 20 (50.67 cm): 6 -> 8 -> 4 -> 17 -> 5 -> 18 -> 3 -> 16 -> 2 -> 11 -> 1 -> 15 -> 7 -> 10

Busca finalizada.
```

Figura 3.2: Representação gráfica do caminho encontrado.

3.4 Conclusão

Este trabalho permitiu consolidar conhecimentos teóricos e práticos no âmbito da Teoria dos Grafos, com especial ênfase na aplicação do algoritmo de Dijkstra para o planeamento de trajetórias em ambientes industriais, alinhados com os princípios das Indústrias 4.0 e 5.0.

A simulação de cenários reais evidenciou a importância da distribuição espacial adequada dos nós e da conectividade do grafo. As restrições impostas nomeadamente uma distância máxima de 27 cm entre pontos, um mínimo de 20 pontos (Tabela: 3.1) e a existência de arestas com comprimento igual ou superior a 5 cm exigiram um planeamento criterioso durante a fase de configuração. Verificou-se que, quando os pontos estavam demasiado afastados, o limite de 100 arestas não era suficiente para garantir conexões válidas, o que inviabilizava a obtenção de trajetos mínimos com comprimento superior a 50 cm.

Apesar desses desafios, o algoritmo de Dijkstra demonstrou-se eficaz sempre que as restrições foram cumpridas. Para os pontos definidos como $\beta = 6$ e $\sigma = 10$, foram identificados dois percursos distintos com custo total idêntico de 50,01 cm, Figura 3.2.

A existência de múltiplas soluções com custo equivalente reforça a robustez do algoritmo e a sua aplicabilidade prática em sistemas reais de navegação robótica. Em suma, o trabalho proporcionou uma experiência rica na modelação e análise de redes de nós, contribuindo para o desenvolvimento de competências essenciais na área da otimização de trajetórias.