深圳大学期末考试试卷

开/闭卷 开卷 A/B 卷 _____

课程编号 1504660001 课序号 01 课程名称 自然语言处理 学分 2.5

命题人(签字)_____ 审题人(签字)_____ _____年____月____日

题号	一	二	三	四	五	六	七	八	九	十	基本题总分	附加题
得分												
评卷人												

座号_____ 学号_____ 姓名_____ 专业_____ 学院_____

（密封线内不答题）

密 封 线

一、随堂测试 3: 基于图网络建模短文本数据信息的计算案例 (共 7 题，共 100 分)

（姓名： 学号： ）

假设文档内容：

文档 1： "我喜欢吃苹果和香蕉，因为它们很健康。"

文档 2： "香蕉和樱桃是我最喜欢的水果之一。"

A. 使用 python 语言，建立单词字典，每个单词用 id 表示

```
# 定义两个短文本的单词列表
words_doc1 = ["我", "喜欢", "吃", "苹果", "和", "香蕉", "因为", "它们", "很", "健康"]
words_doc2 = ["香蕉", "和", "樱桃", "是", "我", "最", "喜欢", "的", "水果", "之一"]

# 合并所有单词并去重
all_words = list(set(words_doc1 + words_doc2))
# 按 Unicode 排序（中文排序规则）并分配唯一 ID
word2id = {word: idx for idx, word in enumerate(sorted(all_words))}
id2word = {idx: word for word, idx in word2id.items()}  # 反向映射（方便后续查看）

print("单词-ID 映射表：")
for word, idx in sorted(word2id.items(), key=lambda x: x[1]):
    print(f"ID {idx}: {word}")
```

```
单词-ID 映射表：
ID 0: 之一
ID 1: 健康
ID 2: 吃
ID 3: 和
ID 4: 喜欢
ID 5: 因为
ID 6: 它们
ID 7: 很
ID 8: 我
ID 9: 是
ID 10: 最
```

```
ID 11: 樱桃
ID 12: 水果
ID 13: 的
ID 14: 苹果
ID 15: 香蕉
```

B. 写出上图单词 id 的邻接矩阵

```
import numpy as np
# 总单词数（节点数）
N = len(word2id)
# 初始化邻接矩阵（N×N，0 表示不共现，1 表示共现）
A = np.zeros((N, N), dtype=int)

# 转换文档单词为 ID 列表
doc1_ids = [word2id[word] for word in words_doc1]
doc2_ids = [word2id[word] for word in words_doc2]

# 填充共现关系（同一文档中任意两个不同单词共现）
for doc_ids in [doc1_ids, doc2_ids]:
    for i in range(len(doc_ids)):
        u = doc_ids[i]
        for j in range(len(doc_ids)):
            v = doc_ids[j]
            if u != v:  # 排除自环
                A[u][v] = 1  # 共现则设为 1

print(f"\n 邻接矩阵形状：（{N}×{N}）")
for i in range(16):
    print(f"ID {i:2d}: {A[i, :16]}")  # 行数据
```

```
邻接矩阵形状：（16×16）
ID  0: [0 0 0 1 1 0 0 0 1 1 1 1 1 1 0 1]
ID  1: [0 0 1 1 1 1 1 1 0 0 0 0 0 1 1]
ID  2: [0 1 0 1 1 1 1 1 0 0 0 0 0 1 1]
ID  3: [1 1 1 0 1 1 1 1 1 1 1 1 1 1 1]
ID  4: [1 1 1 1 0 1 1 1 1 1 1 1 1 1 1]
ID  5: [0 1 1 1 1 0 1 1 1 0 0 0 0 1 1]
ID  6: [0 1 1 1 1 1 0 1 1 0 0 0 0 1 1]
ID  7: [0 1 1 1 1 1 1 0 1 0 0 0 0 1 1]
ID  8: [1 1 1 1 1 1 1 1 0 1 1 1 1 1 1]
ID  9: [1 0 0 1 1 0 0 0 1 0 1 1 1 0 1]
ID 10: [1 0 0 1 1 0 0 0 1 1 0 1 1 1 0 1]
ID 11: [1 0 0 1 1 0 0 0 1 1 1 0 1 1 0 1]
ID 12: [1 0 0 1 1 0 0 0 1 1 1 1 0 1 0 1]
ID 13: [1 0 0 1 1 0 0 0 1 1 1 1 1 0 0 1]
ID 14: [0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
ID 15: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
```

C. 使用 python 语言，为每个单词 id 随机初始化一个向量，构造基于邻接矩阵的相似性方法的损失函数，并写出计算结果。

参考：$$\mathcal{L} = \sum_{(u,v)\in V\times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}\|^2$$

```
# 向量维度（嵌入维度）
d = 2
# 随机初始化单词向量（均值 0，方差 1）
np.random.seed(42)  # 固定随机种子，确保结果可复现
z = np.random.randn(N, d)

# 计算损失函数 L = Σ(||z_u • z_v - A_uv||² )
loss = 0.0
for u in range(N):
    for v in range(N):
        dot_product = np.dot(z[u], z[v])  # 向量内积
        loss += (dot_product - A[u, v]) **2  # 累计平方误差

print(f"\n 单词向量：")
for i in range(16):
    print(f"ID {i}（{id2word[i]}）: {z[i]}")
print(f"损失函数值 L = {loss:.2f}")
```

```
单词向量：
ID 0（之一）: [ 0.49671415 -0.1382643 ]
ID 1（健康）: [0.64768854 1.52302986]
ID 2（吃）: [-0.23415337 -0.23413696]
ID 3（和）: [1.57921282 0.76743473]
ID 4（喜欢）: [-0.46947439  0.54256004]
ID 5（因为）: [-0.46341769 -0.46572975]
ID 6（它们）: [ 0.24196227 -1.91328024]
ID 7（很）: [-1.72491783 -0.56228753]
ID 8（我）: [-1.01283112  0.31424733]
ID 9（是）: [-0.90802408 -1.4123037 ]
ID 10（最）: [ 1.46564877 -0.2257763 ]
ID 11（樱桃）: [ 0.0675282  -1.42474819]
ID 12（水果）: [-0.54438272  0.11092259]
ID 13（的）: [-1.15099358  0.37569802]
ID 14（苹果）: [-0.60063869 -0.29169375]
ID 15（香蕉）: [-0.60170661  1.85227818]
损失函数值 L = 637.78
```

D. 计算获得 2-hop 的邻接矩阵

```
import numpy as np

# 2-hop 邻接矩阵 = 邻接矩阵与自身的矩阵乘法（表示两步路径数）
A_2hop = np.dot(A, A)

print(f"2-hop 矩阵形状：{A_2hop.shape}（{A_2hop.shape[0]}×{A_2hop.shape[1]}）")
# 选择"喜欢"（ID=5）展示其 2-hop 路径数（与其他单词的两步可达路径）
target_id = 5
print(f"\n 以单词「{id2word[target_id]}」（ID={target_id}）为例：")
print("目标单词到其他单词的 2-hop 路径数：")
for v in [4,5,9,15]:  # 挑选几个关键单词 ID
    print(f"→ 到「{id2word[v]}」（ID={v}）：{A_2hop[target_id, v]} 条路径")
# 展示部分矩阵（前 6 行 6 列）
print("\n2-hop 矩阵：")
print(A_2hop[:16, :16])
```

```
2-hop 矩阵形状：(16, 16)（16×16）

以单词「因为」（ID=5）为例：
目标单词到其他单词的 2-hop 路径数：
→ 到「喜欢」（ID=4）：8 条路径
→ 到「因为」（ID=5）：9 条路径
→ 到「是」（ID=9）：4 条路径
→ 到「香蕉」（ID=15）：8 条路径

2-hop 矩阵：
[[ 9  4  4  8  8  4  4  4  8  8  8  8  8  8  4  8]
 [ 4  9  8  8  8  8  8  8  8  4  4  4  4  4  8  8]
 [ 4  8  9  8  8  8  8  8  8  4  4  4  4  4  8  8]
 [ 8  8  8 15 14  8  8  8 14  8  8  8  8  8  8 14]
 [ 8  8  8 14 15  8  8  8 14  8  8  8  8  8  8 14]
 [ 4  8  8  8  8  9  8  8  8  4  4  4  4  4  8  8]
 [ 4  8  8  8  8  8  9  8  8  4  4  4  4  4  8  8]
 [ 4  8  8  8  8  8  8  9  8  4  4  4  4  4  8  8]
 [ 8  8  8 14 14  8  8  8 15  8  8  8  8  8  8 14]
 [ 8  4  4  8  8  4  4  4  8  9  8  8  8  8  4  8]
 [ 8  4  4  8  8  4  4  4  8  8  9  8  8  8  4  8]
 [ 8  4  4  8  8  4  4  4  8  8  8  9  8  8  4  8]
 [ 8  4  4  8  8  4  4  4  8  8  8  8  9  8  4  8]
 [ 8  4  4  8  8  4  4  4  8  8  8  8  8  9  4  8]
 [ 4  8  8  8  8  8  8  8  8  4  4  4  4  4  9  8]
 [ 8  8  8 14 14  8  8  8 14  8  8  8  8  8  8 15]]
```

E. 从节点 1 出发，假设步长长度为 1（即每个方向只走一步），写出基于宽度优先游走可能获得的邻居节点：

```
print("\n=== E. 宽度优先游走（BFS） ===")
start_id = 1  # 起点为"因为"（ID=1）
print(f"起点：「{id2word[start_id]}」（ID={start_id}），步长=1（直接邻居）")

# 步长 1：所有与起点直接相连的节点（A[start_id][v] == 1）
bfs_neighbors = [v for v in range(A.shape[0]) if A[start_id, v] == 1]
# 转换为单词
bfs_words = [id2word[v] for v in bfs_neighbors]

print(f"直接邻居 ID：{bfs_neighbors}")
print(f"对应单词：{bfs_words}")
```

```
=== E. 宽度优先游走（BFS） ===
起点：「健康」（ID=1），步长=1（直接邻居）
直接邻居 ID：[2, 3, 4, 5, 6, 7, 8, 14, 15]
对应单词：['吃', '和', '喜欢', '因为', '它们', '很', '我', '苹果', '香蕉']
```

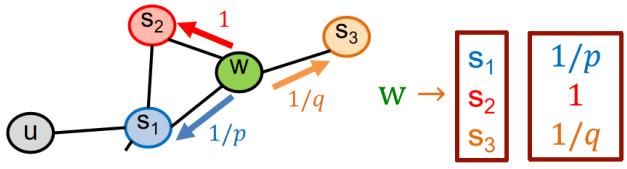
F. 从节点 1 出发，假设步长长度为 2，写出基于深度优先游走可能获得的邻居节点：

```
print("\n=== F. 深度优先游走（DFS） ===")
start_id = 1  # 起点仍为"因为"（ID=1）
print(f"起点：「{id2word[start_id]}」（ID={start_id}），步长=2（两步可达节点）")

# 第一步：获取步长 1 的邻居（u）
step1_neighbors = [v for v in range(A.shape[0]) if A[start_id, v] == 1]

# 第二步：从每个 u 出发，找其邻居 v（排除起点）
step2_neighbors = []
for u in step1_neighbors:
    # u 的邻居中，排除起点 start_id
    u_neighbors = [v for v in range(A.shape[0]) if A[u, v] == 1 and v != start_id]
    step2_neighbors.extend(u_neighbors)
# 去重
step2_neighbors = list(set(step2_neighbors))
# 转换为单词
step2_words = [id2word[v] for v in step2_neighbors]

print(f"两步可达节点 ID：{step2_neighbors}")
print(f"对应单词：{step2_words}")
```

```
=== F. 深度优先游走（DFS） ===
起点：「健康」（ID=1），步长=2（两步可达节点）
两步可达节点 ID：[0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
对应单词：['之一', '吃', '和', '喜欢', '因为', '它们', '很', '我', '是', '最', '樱桃
', '水果', '的', '苹果', '香蕉']
```

G. 在有偏游走算法中，当前节点是 2，上一步所在的节点是 1，给定 p=0.2，q=0.3，则下一步可能游走到的邻接节点可以有哪些，对应的游走的权重和概率是多少。

参考：




```
print("\n=== G. 有偏游走（Node2Vec） ===")
current_id = 2      # 当前节点："健康"（ID=2）
prev_id = 1         # 上一步节点："因为"（ID=1）
p = 0.2             # 回到上一步的惩罚参数
q = 0.3             # 远离上一步的奖励参数

print(f"当前节点：「{id2word[current_id]}」（ID={current_id}）")
print(f"上一步节点：「{id2word[prev_id]}」（ID={prev_id}），参数 p={p}, q={q}")

# 1. 获取当前节点的所有邻居
current_neighbors = [v for v in range(A.shape[0]) if A[current_id, v] == 1]
print(f"\n 当前节点的邻居：{current_neighbors} → 对应单词：{[id2word[v] for v in current_neighbors]}")

# 2. 计算每个邻居的权重（Node2Vec 规则）
weights = []
for v in current_neighbors:
    if v == prev_id:
        # 回到上一步：权重=1/p
        weight = 1 / p
    else:
        # 检查 v 是否是上一步节点的邻居（1-hop 内）
        if A[prev_id, v] == 1:
            # 局部探索（与上一步共享邻居）：权重=1
            weight = 1.0
        else:
            # 深度探索（远离上一步）：权重=1/q
            weight = 1 / q
    weights.append(weight)

# 3. 计算概率（权重归一化）
total_weight = sum(weights)
probabilities = [w / total_weight for w in weights]

# 4. 输出结果
print("\n 邻居节点 | 单词 | 权重 | 跳转概率")
print("-" * 40)
for v, w, prob in zip(current_neighbors, weights, probabilities):
    print(f"{v:8d} | {id2word[v]:4s} | {w:5.2f} | {prob:.4f}")
```

=== G. 有偏游走（Node2Vec） ===
当前节点：「吃」（ID=2）
上一步节点：「健康」（ID=1），参数 p=0.2, q=0.3

当前节点的邻居：[1, 3, 4, 5, 6, 7, 8, 14, 15] → 对应单词：['健康', '和', '喜欢', '因为', '它们', '很', '我', '苹果', '香蕉']

邻居节点	单词	权重	跳转概率
1	健康	5.00	0.3846
3	和	1.00	0.0769
4	喜欢	1.00	0.0769
5	因为	1.00	0.0769
6	它们	1.00	0.0769
7	很	1.00	0.0769
8	我	1.00	0.0769
14	苹果	1.00	0.0769
15	香蕉	1.00	0.0769