hw1

PB21051012 刘祥辉

1.1

length(tangerine)+length(trees) = 363,465

length(marmalede)+length(skies) = 379,571

length(kaleidoscope) +length(eyes) = 300,321

推荐的处理次序：

kaleidoscope OR eyes

tangerine OR trees

marmalade OR skies

1.2

1）发生了一次跳转

2） <3,3> <5,5> <9,89> <15,89> <24,89><75,89><92,89><81,89> <84,89><89,89>

<92,95><115,95><96,95><96,97><97,97><100,99><100,100> <115,101>

共比较了18次

3） <3,3><5,5><9,89><15,89><24,89><39,89><60,89><60,89><75,89> <81,89><84,89>

<89,89><92,95><96,95><96,97><97,97><100,99><100,100> <115,101>

共比较了19次

1.3

文档ID编码：

777

原始字节：1100001001

可变字节编码：00000110 10001001

17743

原始字节：10001010 1001111

可变字节编码：00000001 00001010 11001111

294068

原始字节： 10001 1111001 0110100

可变字节编码：00010001 01111001 10110100

31251336

原始字节：1110 1110011 0110111 0001000

可变字节编码： 00001110 01110011 00110111 10001000

间距编码：

777

原始字节：1100001001

可变字节编码：00000110 10001001

16966

原始字节：1 0000100 1000110

可变字节编码：00000001 00000100 11000110

276325

原始字节：10000 1101110 1100101

可变字节编码：00010000 01101110 11100101

30957268

原始字节：1110 1100001 0111101 1010100

可变字节编码： 00001110 01100001 00111101 11010100

1.4

第一天（晴天）

概率：

城市1：0.2*0.5=0.1

城市2：0.4*0.4=0.16

城市3：0.4*0.7=0.28

第二天（雨天）

概率：

城市1->城市1：0.1 * 0.5 * 0.5 = 0.025

城市1->城市2：0.1 * 0.2 * 0.6 = 0.012

城市1->城市3：0.1 * 0.3 * 0.3 = 0.009

城市2->城市1：0.16 * 0.3 * 0.5 = 0.024

城市2->城市2：0.16 * 0.5 * 0.6 = 0.048

城市2->城市3: 0.16 * 0.2 * 0.3 = 0.0096

城市3->城市1: 0.28 * 0.2 * 0.5 = 0.028

城市3->城市2: 0.28 * 0.3 * 0.6 = 0.0504

城市3->城市3: 0.28 * 0.5 * 0.3 = 0.042

第三天(晴天)

概率： 城市1->城市1->城市1：0.025 * 0.5 * 0.5 = 0.00625

城市1->城市1->城市2：0.025 * 0.2 * 0.4 = 0.002

城市1->城市1->城市3：0.025 * 0.3 * 0.7 = 0.00525

城市1->城市2->城市1：0.012 * 0.3 * 0.5 = 0.0018

城市1->城市2->城市2：0.012 * 0.5 * 0.4 = 0.0024

城市1->城市2->城市3：0.012 * 0.2 * 0.7 = 0.00168

城市1->城市3->城市1：0.009 * 0.2 * 0.5 = 0.0009

城市1->城市3->城市2：0.009 * 0.3 * 0.4 =0.00108

城市1->城市3->城市3：0.009 * 0.5 * 0.7 =0.00315

城市2->城市1->城市1：0.024 * 0.5 *0.5 = 0.006

城市2->城市1->城市2：0.024 * 0.2 * 0.4 =0.00192

城市2->城市1->城市3：0.024 * 0.3 * 0.7 = 0.004032

城市2->城市2->城市1：0.048 * 0.3 * 0.5 = 0.0072

城市2->城市2->城市2：0.048 * 0.5 * 0.4 = 0.0096

城市2->城市2->城市3：0.048 * 0.2 * 0.7 = 0.00672

城市2->城市3->城市1: 0.0096 * 0.2 * 0.5 = 0.00096

城市2->城市3->城市2: 0.0096 * 0.3 * 0.4 = 0.001152

城市2->城市3->城市3: 0.0096 * 0.5 * 0.7 = 0.00336

城市3->城市1->城市1: 0.028 * 0.5 * 0.5 = 0.007

城市3->城市1->城市2: 0.028 * 0.2 * 0.4 = 0.00224

城市3->城市1->城市3: 0.028 * 0.3 * 0.7 = 0.00588

城市3->城市2->城市1: 0.0504 * 0.3 * 0.5 = 0.00756

城市3->城市2->城市2: 0.0504 * 0.5 * 0.4 = 0.01008

城市3->城市2->城市3: 0.0504 * 0.2 * 0.7 = 0.007056

城市3->城市3->城市1: 0.042 * 0.2 * 0.5 = 0.0042

城市3->城市3->城市2: 0.042 * 0.3 * 0.4 = 0.00504

城市3->城市3->城市3: 0.042 * 0.5 * 0.7 = 0.0147

所以最有可能的旅行轨迹是:

城市3->城市3->城市3

2.1

1. **一致性哈希算法**：使用一致性哈希算法来分配URL任务给不同的节点。这种算法能够在节点动态增加或减少时，最小化任务重新分配的数量。一致性哈希会根据URL的哈希结果将任务分布到虚拟节点，而不是物理节点，这降低了任务重新分配的成本。

2. **节点监控和自动伸缩**：实施节点监控机制，以监测节点的健康状况。如果节点崩溃或不可用，自动将任务分配给其他可用节点。反之，如果节点动态增加，也可以自动将任务分配给新的节点。云服务提供商通常提供这种自动伸缩的功能。

3. **任务队列**：使用任务队列作为中间层，而不是直接将URL分配给节点。任务队列可以存储待爬取的URL，然后工作节点从队列中获取任务。这种方法允许动态添加或移除节点，而不会中断任务的进行。

4. **负载均衡器**：引入负载均衡器来均衡不同节点之间的任务分配。负载均衡器可以监控各节点的负载情况，并根据节点的性能和可用性来调整任务分发策略。流行的负载均衡器软件包括Nginx、HAProxy和硬件负载均衡器。

5. **动态调整哈希环**：当节点数量发生变化时，动态地重新构建一致性哈希环。这可以确保任务在新的节点上得到均匀分配。重新构建哈希环的频率可以根据节点变更的频率来调整。

6. **分布式存储**：将已爬取的数据存储在分布式存储系统中，例如Hadoop HDFS或云存储服务。这可以减轻单一节点的负载，同时确保数据的持久性。

7. **监控和警报**：实施全面的监控系统，以便能够及时检测到节点故障或负载不平衡的情况。在问题发生时，及时发出警报，以便进行干预和修复。

2.2

- 考虑查询词项的分布：如果查询词项的分布是均匀的，可以选择较小的指针步长和较少的层数，以减少跳表的空间占用和维护成本。
- 考虑查询频率：如果某些查询词项比其他词项更频繁地出现，可以考虑增加其在跳表中的重复节点，以加速查询。
- 考虑数据规模：数据集的大小也会影响跳表的性能。大规模数据集可能需要更多的层数和更大的指针步长。
-

2.3

同时使用位置索引和停用词表：

考虑引入一个停用词列表，以便在创建位置索引时，只记录非停用词的位置信息。这可以有效减少索引的大小和复杂度。潜在问题即某些停用词可能具有特殊的语义含义，无法被正确检索。例如，在查询短语时，像 "to be or not to be" 中的 "be" 可能被视为停用词，因此无法根据相对位置信息来确定短语的含义。为了解决这个问题，可以考虑使用一个通用的符号来代替所有的停用词，这样就可以进行更加模糊的停用词位置判断，以确保不会丢失重要的语义信息。

2.4

1. **建立词汇概率模型**：首先，需要建立一个词汇概率模型，该模型可以为每个词汇分配一个概率值，反映该词汇在语料库中出现的频率。这可以使用大规模文本语料库来训练。
2. **正向最大匹配分词**：在正向最大匹配分词中，从左到右扫描文本，每次尝试匹配最长的词汇。在这个过程中，根据词汇概率模型为每个匹配到的词汇分配一个概率值。
3. **反向最大匹配分词**：在反向最大匹配分词中，从右到左扫描文本，同样每次尝试匹配最长的词汇。同样，根据词汇概率模型为每个匹配到的词汇分配一个概率值。
4. **结合词汇概率**：在正向和反向最大匹配分词的过程中，将每个匹配到的词汇的概率值累加，得到整个分词结果的概率。可以使用一个简单的概率累积方法，如将各个词汇的概率相乘或相加。

5. **选择概率最高的分词结果**：比较正向和反向最大匹配分词的概率累积结果，选择概率最高的分词结果作为最终的分词结果。

反向最大匹配分词的效果是否仍优于正向最大匹配分词?

不一定，分情况讨论

1. **正向最大匹配分词优于反向：** 如果文本中的大多数词汇在正向序列中更容易匹配，并且词汇概率模型更强调正向匹配的信息，那么正向最大匹配分词可能会优于反向。这可能发生在某些语言或文本类型中，其中正向序列更常见。

2. **反向最大匹配分词优于正向：** 如果文本中的词汇更倾向于反向序列匹配，并且词汇概率模型更强调反向匹配的信息，那么反向最大匹配分词可能会优于正向。这在其他语言或文本类型中可能更为常见。

3. **双向匹配效果接近：** 在某些情况下，正向和反向最大匹配的效果可能接近，因为文本中的词汇分布较均匀，或者词汇概率模型的权重平衡。在这种情况下，双向匹配可能是一个更稳健的选择，可以综合两种方向的信息。

2.5

对于英文：

1. 压缩节点：在 Trie 树中，可以采用压缩节点的方式，将相邻的节点合并，只保留一个节点，以节省存储空间。这可以通过将子节点的字符连接到父节点上来实现。这种方法可以有效地减小 Trie 树的规模。

2. 前缀压缩：对于相同的前缀，可以将它们合并为一个节点，只保留一个前缀。例如，对于英文单词 "apple" 和 "appetite"，可以共享前缀 "app"，从而节省存储空间。

3. 使用压缩编码：将每个节点的子节点信息编码成一个整数，以减小存储开销。这可以通过使用一种字典或映射来实现，将字符映射到整数，并将整数与子节点关联。

对于中文：

1. 利用汉字的拼音：将中文字符转化为拼音，然后构建 Trie 树。这样可以大幅减小中文 Trie 树的规模，因为拼音字母较少，且有限。

2. 合并同音字：对于具有相同拼音的字符，可以合并为一个节点，只保留一个拼音，从而减小 Trie 树的大小。

这些优化方法可以有效地减小 Trie 树的存储空间，但这种压缩方法会牺牲一部分查询效率。因为在查找时需要进行更多的节点遍历和比较，可能会稍微增加查询时间。因此，在实际应用中，需要权衡存储空间和查询效率，选择最适合特定需求的 Trie 树结构。