

**课 程 实 验 报 告**

**课程名称： 串并行课程设计实验**

**专业班级： ACM1601**

**学 号： U201614787**

**姓 名： 宁嘉**

**指导教师： 陆枫**

**报告日期： 　 2018/3/1**

**计算机科学与技术学院**

[Lab4 图灵测试实验 5](#_Toc507765330)

[1. 实验背景 5](#_Toc507765331)

[2. 完成代码 5](#_Toc507765332)

[2.1 choose函数 5](#_Toc507765333)

[2.1.1 choose函数思路 5](#_Toc507765334)

[2.1.2 choose函数代码实现及复杂度分析 5](#_Toc507765335)

[**2.1.3 关于choose函数测试** 6](#_Toc507765336)

[2.2 MkTableKGramStats的实现 7](#_Toc507765337)

[2.2.1 定义kgramstats类型 7](#_Toc507765338)

[2.2.2 makeStats函数 7](#_Toc507765339)

[2.2.2.1 makeStats函数思路 7](#_Toc507765340)

[2.2.2.2 makeStats函数实现及复杂度分析 8](#_Toc507765341)

[2.2.3 lookupExts函数 8](#_Toc507765342)

[2.2.4 maxK函数 8](#_Toc507765343)

[2.2.5 关于代码测试 9](#_Toc507765344)

[2.3 MkBabble的实现 10](#_Toc507765345)

[2.3.1 randomSentence 函数 10](#_Toc507765346)

[**2.3.2** **randomDocument函数** 11](#_Toc507765347)

[2.3.2.1 randomDocument函数思路 11](#_Toc507765348)

[2.3.2.2 randomDocument函数的实现及复杂度分析 11](#_Toc507765349)

[3 回答问题 12](#_Toc507765350)

[Lab5 同义词实验 13](#_Toc507765351)

[1. 实验要求 13](#_Toc507765352)

[2.完成代码 13](#_Toc507765353)

[2.1 MkAllShortestPaths的实现 13](#_Toc507765354)

[2.1.1 Graph Construction定义graph的类型 13](#_Toc507765355)

[2.1.2 makeGraph函数 13](#_Toc507765356)

[2.1.3 numEdges和numVertices函数 14](#_Toc507765357)

[**2.1.4 outNeighbors函数** 14](#_Toc507765358)

[2.1.5 ASP类型的确定及简短说明 15](#_Toc507765359)

[2.1.6 makeASP函数 15](#_Toc507765360)

[2.1.6.1简述makeASP函数的思路 15](#_Toc507765361)

[2.1.6.2 makeASP函数的代码实现及复杂度分析 15](#_Toc507765362)

[2.1.7 reprot函数 16](#_Toc507765363)

[2.1.7.1 简述report函数的思路 16](#_Toc507765364)

[2.1.7.2 report函数的实现即复杂度分析 17](#_Toc507765365)

[2.1.8 关于测试 17](#_Toc507765366)

[2.2 MkThesaurusASP的实现 18](#_Toc507765367)

[2.2.1 定义thesaurus数据类型 18](#_Toc507765368)

[2.2.2 实现下列函数 18](#_Toc507765369)

[**2.2.3 关于测试** 19](#_Toc507765370)

[Lab6割边实验 19](#_Toc507765371)

[1. 实验要求 19](#_Toc507765372)

[2. 完成代码并回答问题 20](#_Toc507765373)

[2.1 MkBridges的实现 20](#_Toc507765374)

[2.1.1定义ugraph的类型保存无向图的信息 20](#_Toc507765375)

[2.1.2完成makeGraph函数，简述其功能与复杂度 20](#_Toc507765376)

[2.1.3 完成findBridge函数，简述思路与复杂度 20](#_Toc507765377)

[2.1.4 回答关于Dijkstra’s algorithm的相关问题 22](#_Toc507765378)

[2.1.5 回答关于启发式函数的一些问题 22](#_Toc507765379)

[2.2 MkAStarCore的实现 23](#_Toc507765380)

[2.2.1 完成makeGraph函数 23](#_Toc507765381)

[2.2.2 完成findPath函数 23](#_Toc507765382)

[Lab7 图像分割实验 24](#_Toc507765383)

[1. 实验要求 24](#_Toc507765384)

[2. 完成代码 25](#_Toc507765385)

[2.1 完成函数MST 25](#_Toc507765386)

[2.2 完成函数findSegment并简述思路与复杂度 26](#_Toc507765387)

[3.书面习题 28](#_Toc507765388)

[Lab8 范围搜索实验 29](#_Toc507765389)

[1. 实验要求 29](#_Toc507765390)

[2. 回答问题 29](#_Toc507765391)

[2.1 MkBSTOrderedTable的实现 29](#_Toc507765392)

[2.1.1完成函数first，last简述思路。 29](#_Toc507765393)

[2.1.2 完成函数previous和next并简述思路。 30](#_Toc507765394)

[2.1.3 完成函数join, split 30](#_Toc507765395)

[2.2.4完成函数getRange并简述思路 31](#_Toc507765396)

[2.2 MkRangeCount的实现 31](#_Toc507765397)

[2.2.1 完成函数makeCountTable并回答相关问题 31](#_Toc507765398)

[2.2.2 完成函数count 33](#_Toc507765399)

# Lab4 图灵测试实验

## 1. 实验背景

“图灵测试”是指测试者在与被测试者（一个人和一台机器）隔开的情况下，通过一些装置（如键盘）向被测试者随意提问。进行多次测试后，如果有超过30%的测试者不能确定出被测试者是人还是机器，那么这台机器就通过了测试，并被认为具有人类智能。

本实验设计了一个算法，使得机器能够通过图灵测试。即通过对所提供的文本进行分析，根据每个词之后的一定词数范围内的词出现的频率决定机器所给出的一句话之中下一个词是什么。本实验中提供了莎士比亚的作品，以达到机器能够“写诗”的目的。

## 2. 完成代码

### 2.1 choose函数

Task 4.1 (10%). Implement the function

val choose : 'a hist -> real -> 'a in the functor MkSeqUtil in MkSeqUtil.sml. If 0 r 1, choose hist r should evaluate to the valueat r from the cumulativedistribution that corresponds to the histogramhist. If r is not inthe range [0,1] or hist is empty, you should raise an exception. For full credit, choose should have O(|hist|) work and O(log|hist|) span.

#### 2.1.1 choose函数思路

|  |
| --- |
| Choose 函数按照累加频率分布和给定的随机数（位于0，1之间）去估计得到的是哪个字母，利用此方式实现得到每个字母的概率对应于相应的频率。 |

#### 2.1.2 choose函数代码实现及复杂度分析

|  |
| --- |
| 1. fun choose (hist : 'a hist) (p : real) : 'a = 2. if length hist = 0 orelse p < 0.0 orelse p > 1.0 then raise Range (\*The p < 0.0 case has some bug in tester\*) 3. else 4. let 5. val range = scani op+ 0 (map (fn(\_,i)=>i) hist); 6. val enrange = enum range; 7. val value = Real.ceil (p \* Real.fromInt(nth range (length range - 1))) 8. (\*进行二分查找\*) 9. fun search(x,s) = 10. if length s = 1 then (fn(i,\_)=>i)(nth s 0) else 11. let val halfl = length s div 2 12. in if x <= (fn(\_,a)=>a)(nth s (halfl-1)) then search(x,take(s,halfl) ) 13. else search(x,drop(s,halfl)) 14. end 15. in (fn(a,\_)=>a)(nth hist (search(value,enrange))) 16. end   **复杂度分析**  **记|hist| = n**   1. line 5 复杂度花费在map 与 scani，其中Wmap = O(n) ，Smap = O(1) Wscani = O(n)，Sscani = O(log n) 2. line 6 Wenum = O(n)，Senum = O(1) 3. line 9 对于二分查找work和span皆是O(log n)   综上知，Choose函数花费O(n)的work和O(log n)的span。 |

**2.1.3 关于choose函数测试**

Task 4.2 (5%). Add test cases for choose in Tests.sml. Be sure to consider edge cases, as well as to include some longer, more complex tests. Using comments , brieﬂy explain the motivation behind each test – why is the test useful?

|  |
| --- |
| val testsChoose : (((string \* int) list) \* real) list = [  ([("test", 10)], 0.5),  ([("test", 2), ("awesome", 2)], 0.5),  ([("yay", 1), ("woah", 2), ("oh", 3), ("yup", 4)], 0.47),  ([("hello",1),("goodbye",999)],0.0),  ([("awesome",4),("wow",888)],1.0),  (\* a long tests\*)  ([("a",34),("b",42),("c",23),("d",12),("e",43),("f",73),("g",14),("h",9),("i",23),("j",3),("k",63),  ("l",4),("m",46),("n",34),("o",32),("p",24),("q",15),("r",25),("s",54),("t",42),("u",24),("v",52),  ("w",34),("x",25),("y",36),("z",23)],0.55),  (\* an empty sequence\*)  ([], 0.1),  (\* p < 0\*)(\* need to rase Exn Range, but there are some bugs in reference\*)  ([("neg",13),("ative",24)],~0.1),  (\* p > 1 \*)  ([("ex",5),("out",2)],1.1)  ] |

### 2.2 MkTableKGramStats的实现

#### 2.2.1 定义kgramstats类型

Task 5.1 (2%). Deﬁne the abstract kgramstats type and explain in a comment why you chose the type you did. Hint: You should make use of the functor argument structure T : TABLE where type Key.t = string Util.Seq.seq. Speciﬁcally, 'a T.table deﬁnes a table with keys of type string seq and values of type 'a.

|  |
| --- |
| type kgramstats = int \* (token hist Table.table)  理由如下：  Int是因为之后maxK函数要用，而至于用token hist类型的table是因为要实现的目的就是对每个长度为k的token sequence (k-gram)，找到其后面的那个token，并统计出其后面这样的token出现的次数。 |

#### 2.2.2 makeStats函数

Task5.2(22%).

Implement the function makeStats(describedabove) in the functor MkTableKGramStats in MkTableKGramStats.sml. For full credit, makeStats corpus maxK should have O(nlogn) work and O(log2 n) span, where n is the number of tokens in corpus, and assuming the constant maxK is small.

##### 2.2.2.1 makeStats函数思路

|  |
| --- |
| 算法分以下几个步骤实现：   1. 将document转化成tokens 2. 对每个k (0≤k**≤**maxK)，将对应的k-gram与其后一个元素配对。将所有的这样的pair拼接起来，按gram为键值做一个collect操作，得到gram -> 紧跟其后的token 为元素的table 3. 利用histogram进行Table.map，将token seq map成为(token,frequency)pair sequence. |

##### 2.2.2.2 makeStats函数实现及复杂度分析

|  |
| --- |
| 1. fun makeStats (corpus : string) (maxK : int) : kgramstats = 2. let 3. val tok = tokens (not o Char.isAlphaNum) corpus 4. fun lengthkgram k = tabulate (fn i => (subseq tok (i,k),nth tok (i+k))) ((length tok) - k) 5. val last = flatten (tabulate (fn i => lengthkgram i) (maxK+1)) 6. val res = Table.map (fn x => histogram String.compare x) (Table.collect last) 7. in (maxK,res) 8. end   **复杂度分析：**  **记 n 代表corpus中token的个数**   * line 3 Wtokens = O(n)，Stokens = O(log n) * line 4 Wlengthkgram = O(n)，Slengthkgram = O(1) * line 5 Wtabulate = O(n)，Stabulate = O(1) Wflatten = O(n)，Sflatten = O(log n) * line 6 Wcollect = O(nlogn)，Scollect = O(log2n) Wmap = O(Whistogram) = O(nlog n)，Smap = O(Shistogram) = O(log2 n) |

#### 2.2.3 lookupExts函数

Task5.3(4%).

Implement the function lookupExts(describedabove) in the functor MkTableKGramStats in MkTableKGramStats.sml. For full credit, lookupExts should have O(logn) work and span, where n is the the size of the kgramstats type.

|  |
| --- |
| 1. fun lookupExts (stats : kgramstats) (kgram : kgram) : (token \* int) seq = 2. case Table.find (#2 stats) kgram of 3. NONE => empty() 4. |SOME th => th |

#### 2.2.4 maxK函数

Task 5.4 (2%). Implement the function maxK (described above) in the functor MkTableKGramStats in MkTableKGramStats.sml. For full credit, maxK should have O(1) work and span.

|  |
| --- |
| 1. fun maxK (stats : kgramstats) : int = 2. #1 stats |

#### 2.2.5 关于代码测试

Task 5.5 (5%). Write test cases for MkTableKGramStats in Tests.sml. You should see some existing tests which use the corpus in the ﬁle corpus.txt. corpus.txt will get handed in, so feel free to change it, but any tests dealing with any other corpus will not be graded. You may test with a corpus in a different test ﬁle, but we will just ignore those test cases. Be sure to consider edge cases, as well as to include some longer, more complex tests. Using comments, brieﬂy explain the motivation behind each test – why is the test useful?

|  |
| --- |
| val testsKGramStats : ((string \* int) \* (string list)) list = [  ((corpus, 50),  ["direction",  "time",  "direction of time",  "would write",  "What Eddington says about",  "British Museum",  "write all the books in the British Museum",  (\* more tests cases\*)  "all the books in the British" ,  "What Arthur Eddington says about the infinite monkey theorem and the direction",  "then the world may be rendered",  (\*empty case\*)  ""  ])  ] |

### 2.3 MkBabble的实现

#### 2.3.1 randomSentence 函数

Task 6.1 (10%). Implement the function

val randomSentence : kgramstats -> int -> Rand.rand -> string

in the functor MkBabble in MkBabble.sml. randomSentence stats n seed should generate a sentence with n > 0 words given the stats of some corpus and a random seed. For full credit, randomSentence should have O(n(WlookupExts+Wchoose))work and O(n(SlookupExts+Schoose))span, assuming that the words in the corpus have constant length. The output string should end with a period and not have any leading spaces. You might ﬁnd String.concatWith : string -> string list -> string to be useful.

|  |
| --- |
| 1. fun randomSentence (stats : KS.kgramstats) (n : int) (seed : R.rand) = 2. let 3. val rrseq = Rand.randomRealSeq seed NONE n 4. fun next l len = (\*len is the length of l\*) 5. let 6. fun helper k = (\*k is the length of gram\*) 7. let val temp = Stats.lookupExts stats (fromList(List.rev(subseq l (0, Int.min(len, k))))) 8. in if length temp = 0 then helper (k-1) 9. else temp 10. end 11. in helper (Stats.maxK stats) 12. end 13. fun output L i = (\*i represent the length of L\*) 14. if i = n then L 15. else output ((Util.choose (next L i) (nth rrseq i))::L) (i+1) 16. val res = List.rev(output nil 0) 17. in (String.concatWith " " res) ^ "." 18. end   **复杂度分析：(Assuming the maxK is a constant)**   * line 3 WrandomRealSeq = O(n)，SrandomRealSeq = O(n) * line 4 Wnext = O(WStats.lookupExts), Snext = O(SStats.lookupExts) * line 13 考虑output函数的递归的某一轮有W = O(WUtil.choose+Wnext) = O(WUtil.choose+WStats.lookupExts), S = O(SUtil.choose+Snext) = O(SUtil.choose+SStats.lookupExts)，一共执行了n轮，所以有Woutput = O(n(WUtil.choose+WStats.lookupExts)), Soutput = O(n(SUtil.choose+SStats.lookupExts)) * line 16调用output函数，复杂度如上 * line 16 WList.rev = SList.rev = O(n) ，line 17 WString.concatWith = SString.concatWith = O(n)   综上知，总的W = O(n(WlookupExts+Wchoose))  S = O(n(SlookupExts+Schoose)) |

* + 1. **randomDocument函数**

Task 6.2 (5%). Implement the function

val randomDocument : kgramstats -> int -> Rand.rand -> string

in the functor MkBabble in MkBabble.sml. randomDocument stats n seed should generate a document of n > 0 sentences given the stats of some corpus and a random seed. Each sentence should have a random length between 5 and 10 words. For full credit, randomDocument should have O(nWrandomSentence) work and O(n+SrandomSentence) span, assuming that each sentence has constant length. Each sentence should be separated with a space, and there should be no leading or trailing spaces. Again, you should use String.concatWith which you may assume to have work and span linear in the length of the input string list.

##### 2.3.2.1 randomDocument函数思路

|  |
| --- |
| 产生两个随机序列：随机长度序列与随机种子序列，利用这两个序列对应的元素产生随机句子，一一对应产生n个随机句子。 |

##### 2.3.2.2 randomDocument函数的实现及复杂度分析

|  |
| --- |
| 1. fun randomDocument (stats : KS.kgramstats) (n : int) (seed : R.rand) = 2. let 3. val lenseq = Rand.randomIntSeq seed (SOME(5,11)) n 4. val seedseq = map R.fromInt (Rand.randomIntSeq seed NONE n) 5. fun combine (length,seed) = randomSentence stats length seed 6. val result = map2 combine lenseq seedseq 7. in String.concatWith " " (toList rIesult) 8. end   **复杂度分析：**   * line 3 WrandomIntSeq = O(n)，SrandomIntSeq = O(n) * line 4 WrandomIntSeq = O(n)，SrandomIntSeq = O(n)  Wmap = O(n)，Smap = O(1) * line 6 Wresult = O(nWrandom­­­­­­­­Sentence)，Sresult = O(SrandomSentece) * line 7 WconcatWith = O(n)，SconcatWith = O(n)   综上知：randomDocument have O(nWrandomSentence) work and O(n+SrandomSentence) span. |

## 3 回答问题

Task 7.1 (5%). Estimate the work and span (in big-Θ notation) of an optimal parallel implementation of the function extrema.

|  |
| --- |
| Work = Θ(n), Span = Θ(1)其中n是序列长度 |

Task 7.2(10%). Give pseudocode for a parallel implementation of extrema that meets the cost bounds you gave in 7.1. You need not prove that the implementation meets the cost bounds. You may assume that you have access to the function cmp, given below.

fun cmp f (i,j) = (j < 0) orelse (j > length s - 1) (\* first/last true \*) orelse f (nth s i, nth s j)

|  |
| --- |
| 1. fun extrema s = 2. let 3. val f1 = op< 4. val f2 = op> 5. fun cmp f (i,j) = (j < 0) orelse (j > length s - 1) (\* first/last true \*) orelse f (nth s i, nth s j) 6. fun choose i = if ((cmp f1 (i, i-1)) andalso (cmp f1 (i, i+1))) orelse ((cmp f2 (i, i-1)) andalso (cmp f2 (i, i+1))) then SOME i else NONE 7. in tabulate choose (length s) 8. end |

Task 7.3 (5%). Informally justify why you believe no more efﬁcient algorithm exists. Proving this formally can be quite difﬁcult, but you can often give an informal but convincing argument by appealing to intuition about the problem. For example, a justiﬁcation might take the form of “If a better solution existed, it would let us do X, which is known to be impossible” or “Any algorithm that solves this problem must at least do Y, which takes O(Z).”

|  |
| --- |
| Prove：Span O(1)最优结论平凡，下面说明Work是O(n)的时候是最优的。要判断一个数i是否被选中为SOME i或者NONE，其至少和两边的数都进行比较过，所以可知对任意一个判断算法，我们都必须得知道序列中相邻两个数谁大谁小。而这用**两两比较**的算法至少要比较n-1次，所以Θ(n)的bound已是最优。 |

# Lab5 同义词实验

## 1. 实验要求

本次实验中，你将完成一个寻找无权图中最短路径的ADT，最短路算法被广泛应用于很多地方，你将应用你的解决方案用于处理同义词问题。即给定任意两个单词，在给定的同义词库中找出它们之间的最短路，并且任意两个单词有边相连表示他们是同义词关系。

## 2.完成代码

### 2.1 MkAllShortestPaths的实现

#### 2.1.1 Graph Construction定义graph的类型

Task 4.1 (2%). In MkAllShortestPaths.sml, deﬁne the type graph that would allow you to implement the following functions within the required cost bounds. **Leave a brief comment explaining why you chose the representation that you did.**

|  |
| --- |
| type graph = (set table) \* int \* int (\*number of edges and vertices\*)  (\*remain the number of edges and vertices for they will be needed later.  To find the outNeighbors quickly, specify it using the table structure\*) |

#### 2.1.2 makeGraph函数

Task 4.2 (8%). Implement the function

makeGraph : edge seq -> graph

which generates a graph based on an input sequence E of directed edges. The number of vertices in the the resulting graph is equal to the number of vertex labels in the edge sequence. For full credit, makeGraph must have O(|E|log|E|) work and O(log2|E|) span.

|  |  |  |  |
| --- | --- | --- | --- |
| 1. fun makeGraph (E : edge seq) : graph = 2. let 3. val edgenum = length E 4. val adjs = Table.collect E 5. val adjt = Table.map Set.fromSeq adjs 6. val rangeset = Set.fromSeq (map #2 E) 7. val verticesnum = Set.size(Set.union (domain adjs,rangeset)) 8. in (adjt,edgenum,verticesnum) 9. end   **复杂度分析：**   * line 4 Wcollect = O(|E|log |E|), Scollect = O(log2|E|) * line 5 由Table.map的cost specification：map f T  |  |  |  | | --- | --- | --- | |  | Work = ∑(k↦v)∈TW(f(v))  Span = O(log |T|) + max(k↦v)∈TS(f(v))  以及Set.fromSeq的cost specification : fromSeq S  Work = O(|S|log|S|)  Span = O(log2|S|)  由于alog a + blog b < (a+b)log(a+b) for a>0 andalso b>0  所以有Wmap = O(|E|log |E|)  Smap = O(log|E|) + O(log2|E|) = O(log2|E|) | O(log|T|)+max(k↦v)∈TS(f(v)) |  * line 6 Wmap = O(|E|), Smap = O(1)  WSet.fromSeq = O(|E|log |E|), SSet.fromSeq = O(log2|E|) * line 7 Wdomain = O(|E|), Sdomain = O(log |E|)  WSet.union = O(|E|), SSet.union = O(log |E|)   综上知：makeGraph的Work = O(|E|log|E|), Span = O(log2|E|) |

#### 2.1.3 numEdges和numVertices函数

Task 4.3 (6%). Implement the functions

numEdges : graph -> int numVertices : graph -> int

which return the number of directed edges and the number of unique vertices in the graph, respectively.

|  |
| --- |
| 1. fun numEdges (G : graph) : int = 2. #2 G 3. fun numVertices (G : graph) : int = 4. #3 G |

**2.1.4 outNeighbors函数**

Task 4.4 (6%). Implement the function

outNeighbors : graph -> vertex -> vertex seq

which returns a sequence Vout containing all out neighbors of the input vertex. In other words, given a graph G =(V,E), outNeighbors G v containsall w s.t. (v,w)∈E. If the input vertex is not in the graph, outNeighbors returns an empty sequence. For full credit, outNeighbors must have O(|Vout|+log|V|) work and O(log|V|) span, where V is the set of vertices in the graph.

|  |
| --- |
| 1. fun outNeighbors (G : graph) (v : vertex) : vertex seq = 2. case find (#1 G) v of 3. NONE => Seq.empty() 4. |SOME(ot) => Set.toSeq ot   复杂度分析：Wfind = Sfind = O(log |G|) = O(log |V|)  If find the vertex then locate to its out-neighbors and do Set.toSeq  WSet.toSeq = O(|Vout|), SSet.toSeq = O(log|Vout|) note that |Vout| doesn’t exceed |V|  We have Work = O(|Vout| + log|V|), Span = O(log|V|) |

#### 2.1.5 ASP类型的确定及简短说明

Task4.5(2%). In MkAllShortestPaths.sml, deﬁne the type asp that would allow you to implement the following functions within the required cost bounds. Leave a brief comment explaining why you chose the representation that you did.

|  |
| --- |
| type asp = vertex Seq.seq table  (\*to track back the path of all shortest path between two vertices via asp type, we remain the parents of the discovered vertices with them in bfs, there is the vertex matching to all of their parents in bfs process spcified as a sequence table\*) |

#### 2.1.6 makeASP函数

Task 4.6 (23%). Implement the function

makeASP : graph -> vertex -> asp

to generate an asp which contains information about all of the shortest paths from the input vertex v to all other reachable vertices. If v is not in the graph, the resulting asp will be empty. Given a graph G = (V,E), make ASP G v must have O(|E|log|V|) work and O(Dlog2|V|) span, where D is the longest shortest path (i.e., the shortest distance to the vertex that is the farthest from v).

##### 2.1.6.1简述makeASP函数的思路

|  |
| --- |
| 由于要找最短路径，采用bfs去进行图搜，将每个距离为i的点map到所有距离为i-1的点，这只需小小地修改一下bfs的过程即可，即在bfs的过程中每一步都将这样的点collect为一个sequence即可。 |

##### 2.1.6.2 makeASP函数的代码实现及复杂度分析

|  |
| --- |
| 1. fun makeASP (G : graph) (v : vertex) : asp = 2. case find (#1 G) v of 3. NONE => Table.empty() 4. | SOME(\_) => 5. let 6. fun spwithprt(X,F) =(\*shortest path with parents\*) 7. if size F = 0 then X else 8. let 9. fun vpp s = Seq.map (fn v => (v,s)) (outNeighbors G s) (\*vetex with parents pair\*) 10. (\* Work = O(|NGout of s| + log|V|); Span = O(log|V|)\*) 11. val frontout = flatten(map vpp (Set.toSeq (domain F))) (\*merge together\*) 12. (\*work = O(||F|| + |F|log|V|); Span = O(log|V|)\*) 13. val vwp = Table.collect frontout (\*vetex(key) with parents(value)\*) 14. (\*work = O(||F||log|V|); Span = O(log|V| ^2\*) 15. val newX = Table.merge (fn (x,\_) => x) (X,F) 16. val newF = Table.erase (vwp,domain newX) 17. in spwithprt(newX,newF) 18. end 19. in spwithprt(Table.empty(),Table.singleton(v,Seq.empty())) 20. end   **Cost Analysis:**  只需考虑与最基本的bfs中不同的部分，证明这些部分的cost不会超过bfs所用的cost即可，已将这部分的cost作为了注释插入在了相应的代码处。其中||F||代表domain F中所有顶点的出度之和，|F|表示F的size。所以每一轮额外的cost为Work = O(||F||log|V|) (\*F represents the frontier in bfs\*), Span = O(log2|V|).  接着考虑bfs循环至结束的总的Work和Span.  由于每一轮的frontier都不一样，知每一轮的frontier出边也不一样，所以有如下等式∑||F|| ≤ |E|，且当bfs遍历所有的边的时候等号成立。于是所有轮的总Work = O(|E|log|V|)，总Span = O(Dlog2|V|)。这与最基本的bfs span的asymtotic cost相同，于是总的Work = O(|E|log|V|)，总的Span = O(Dlog2|V|)。 |

#### 2.1.7 reprot函数

Task 4.7 (15%). Implement the function

report : asp -> vertex -> vertex seq seq

which, given an asp for a source vertex u, returns all shortest paths from u to the input vertex v as a sequence of paths (each path is a sequence of vertices). If no such path exists, report asp v evaluates to the empty sequence. For full credit, report must have O(|P||L|log|V|) work and span, where V is the set of vertices in the graph, P is the number of shortest paths from u to v, and L is the length of the shortest path from u to v.

##### 2.1.7.1 简述report函数的思路

|  |
| --- |
| 从u到v的某路径是最短路径，由子路径性质知（可视边权为1），其中从u出发的子路径也是从u开始到该子路径终点的最短路径，即恰对应bfs中的每一层(level)，所以要找子路径可以从v开始往回走，利用存下的asp不难实现该功能。事实上，将问题化归为了在vertex seq table represent的图中从一点出发去search到另外一点的所有路径，这可以通过稍微的改动dfs即可实现该效果。 |

##### 2.1.7.2 report函数的实现即复杂度分析

|  |
| --- |
| 1. fun report (A : asp) (v : vertex) : vertex seq seq = 2. let 3. fun bottomdfs l (s:vertex list list,v) =(\*different from dfs for searching to the bottom every time\*) 4. case find A v of 5. NONE => nil 6. | SOME(ng) => if length ng = 0 then (v::l)::s 7. else Seq.iter (bottomdfs (v::l)) s ng 8. val pathlist = bottomdfs nil (nil,v) 9. in Seq.map Seq.fromList (Seq.fromList pathlist) 10. end   **Cost Analysis:**  Iterate dfs中cost主要花费在find这一步，而Wfind = Sfind = O(log n)，只需要计算在图搜中find了多少次，容易知道find A v中的v一定属于某path，且每个这样的v只会被find恰好一次，又在所有的shortest path上的点的个数不超过|P|(|L|+1)，又|L|≥1有|P|(|L|+1)≤2|P||L|，所以find的总cost为W = S = O(|P||L|log|V|)  又fromList的work和span花费为线性，所以line 9 work = span = O(n)  综上所述有Work = Span = O(|P||L|log|V|) |

#### 2.1.8 关于测试

Task 4.8 (6%). Test your ALL\_SHORTEST\_PATHS implementation in the ﬁle Tests.sml by adding test cases to the appropriate lists (see the ﬁle for reference – there are existing test cases to guide you). The following functions are deﬁned to run your implementation of functions of ALL\_SHORTEST\_PATHS against your test cases in Tests.sml.

|  |
| --- |
| 测试样例：  val edgeseq = [(1,2)]  val edgeseq2 = [(1,2),(2,3),(3,4),(2,4),(1,5),(5,4),(5,6),(6,7)]  val test3 = [(2,1),(4,2),(4,3),(4,5),(1,3),(3,5),(3,2),(1,4),(2,5),(5,1)]  val test4 = [(1,2),(1,12),(2,11),(2,3),(3,4),(11,4),(2,4),(4,5),(3,12),(4,6),(4,12),(11,7),(5,7),(6,7),(1,8),(4,9),(9,7),(8,4)]  val testfile = "input/thesaurus.txt"  val testfile2 = "input/simpletest.txt"  (\* The following are required \*)  val testsNum = [edgeseq, edgeseq2, test3, test4];  val testsOutNeighbors = [(edgeseq, 1), (edgeseq, 2),(test3,1),(edgeseq2,5),(edgeseq2,7),(test3,9),(test4,11)]  val testsReport = [((edgeseq, 1), 2), ((edgeseq2, 1), 4), ((edgeseq2, 1), 7),((test3,4),2),((test3,1),3),((test3,6),2),((test3,1),6),((test4,11),3)] |

### 2.2 MkThesaurusASP的实现

#### 2.2.1 定义thesaurus数据类型

|  |
| --- |
| 1. type thesaurus = graph |

#### 2.2.2 实现下列函数

Task 5.1 (10%). Implement the function

make : (string \* string seq) seq -> thesaurus which generates a thesaurus given an input sequence of pairs (w,S) such that each word w is paired with its sequence of synonyms S. You must deﬁne the type thesaurus yourself.

|  |
| --- |
| 1. fun make (S : (string \* string seq) seq) : thesaurus = 2. let 3. fun toedge (str,strseq) = map (fn x => (str,x)) strseq 4. val edgeseq = flatten (map toedge S) 5. in makeGraph edgeseq 6. end |

Task 5.2 (6%). Implement the functions

numWords : thesaurus -> int

synonyms : thesaurus -> string -> string seq

where numWords counts the number of distinct words in the thesaurus while synonyms returns a sequence containing the synonyms of the input word in the thesaurus. synonyms returns an empty sequence if the input word is not in the thesaurus.

|  |
| --- |
| 1. fun numWords (T : thesaurus) : int = 2. numVertices T 3. fun synonyms (T : thesaurus) (w : string) : string seq = 4. outNeighbors T w |

Task 5.3 (10%). Implement the function

query : thesaurus -> string -> string -> string seq seq such that query th w1 w2 returns all shortest path from w1 to w2 as a sequence of strings with w1 ﬁrst and w2 last. If no such path exists, query returns the empty sequence. For full credit, your function query must be staged.

|  |
| --- |
| 1. fun query (T : thesaurus) (w1 : string) (w2 : string) : string seq seq = 2. report (makeASP T w1) w2 |

**2.2.3 关于测试**

Task 4.8 (6%). Test your ALL\_SHORTEST\_PATHS implementation in the ﬁle Tests.sml by adding test cases to the appropriate lists (see the ﬁle for reference – there are existing test cases to guide you).

|  |
| --- |
| 测试样例：  val testsNumWords = [testfile, testfile2]  val testsSynonyms =  [(testfile2, "HANDSOME"),  (testfile2, "VINCENT"),  (testfile2, "PRETTY"),  (testfile2,"BADASS"),  (testfile, "GOOD")]  val testsQuery =  [(testfile2, ("HANDSOME", "YOLO")),(testfile2,("BADASS","STUPID")),(testfile2,("PRETTY","CHRIS")),(testfile, ("GOOD", "BAD")),(testfile, ("CLEAR", "VAGUE")),(testfile, ("LOGICAL", "ILLOGICAL")),(testfile,("HAPPY","SAD")),(testfile,("LIBERAL","CONSERVATION")),(testfile, ("EARTHLY", "POISON"))] |

# Lab6割边实验

## 1. 实验要求

本次实验中，你需要在一个图中找出所有割边，所谓割边即删除此边之后图的极大联通子图增加，你可以参考targan算法。此外你还将完成最短路的Astar算法，你只需要稍微改变一下迪克斯特拉算法就可以达成要求。

## 2. 完成代码并回答问题

### 2.1 MkBridges的实现

#### 2.1.1定义ugraph的类型保存无向图的信息

|  |
| --- |
| type ugraph = vertex seq seq  只有用邻接sequence去存图才能实现之后findbridge函数的线性复杂度。 |

#### 2.1.2完成makeGraph函数，简述其功能与复杂度

Task 4.1 (5%). Deﬁne the type ugraph representing an undirected graph and write the function

val makeGraph : edge seq -> ugraph

in MkBridges.sml which takes in a sequence S representing the edges of a graph G as described above and returns that same graph under your ugraph representation. For full credit, makeGraph must have have O(|E|log|V|) work and O(log2|V|) span.

|  |
| --- |
| 1. fun makeGraph (E : edge seq) : ugraph = 2. let 3. val edgeseq = append(map (fn(u,v) => (v,u)) E, E) 4. val res = collect Int.compare edgeseq 5. in map (fn (x,y) => y) res 6. end  * 功能描述： 将edge seq转换为adjacent seq 的graph representation * 复杂度分析：由于append与map都是线性的work与常数的span，所以复杂度主要花费在collect，有Wcollect = O(|E|log|E|) = O(|E|log|V|) (\*for |E|<|V|2 in simple graph\*)  Scollect = O(log2|E|) = O(log2|V|) |

#### 2.1.3 完成findBridge函数，简述思路与复杂度

Task 4.2 (30%). Implement the function

val findBridges : ugraph -> edge seq

in MkBridges.sml which takes an undirected graph and returns a sequence containing exactly the edges which are bridges of G. For full credit, findBridges must have O(|V|+|E|) work and span. The edges need not be ordered in any way, but for any edge{u,v}, at most one of{(u,v),(v,u)}should appear in the output sequence.

|  |
| --- |
| 一. 思路：  先只考虑图连通时的情况，对每个点赋予两个标签，一个是发现它时的时间邮戳，一个是minValue。minValue在dfs的过程中先初始化为对应的discovered time stamp，然后在dfs回溯的过程中进行维护，即将其在dfs回溯过程中将该点的minValue更新为前一个点的minValue与自己本身的minValue中更小的那个。如果某条边(a,b)是割边，则b点能探测到的minValue一定会大于a的discovered timestamp；若(a,b)不是割边，则(a,b)在一个cycle中，b点能探测到的minValule一定会小于等于a的discovered timestamp，且割边一定是树边，借用该特征便可以很方便的判断一条边是不是割边。若图不连通，用dfs进行检测，为了能遍历到所有的顶点，补充一个新的点，并添上该点指向其他所有点的边，从该点出发去dfs，最后再删掉一些从这个点出发的被判为割边的边即可。 |
| 1. fun findBridges (G : ugraph) : edges = 2. let 3. val newG = append(G,singleton (tabulate (fn i => i) (length G)))(\*in order to run all vertices\*) 4. val numV = length newG 5. val initX = STSeq.fromSeq (tabulate (fn i => false) numV) (\*discoverd vertices\*) 6. val initminV = STSeq.fromSeq (tabulate (fn i => 0) numV) (\*updated min value\*) 7. fun updatemin p v minV= if STSeq.nth minV p > STSeq.nth minV v then STSeq.update (p,STSeq.nth minV v) minV 8. else minV 9. fun bridge p v i l minV= if STSeq.nth minV v > i-1 then (p,v)::l else l(\*i-1 is the lable of p\*) 10. fun dfs p ((X, minV, L, i),v) = 11. case STSeq.nth X v of 12. true => (X, updatemin p v minV, L, i) (\*revisit v\*) 13. |false => 14. let 15. val X' = STSeq.update (v, true) X 16. val minV' = STSeq.update (v,i) minV (\*discover v\*) 17. val (X'',minV'', L',i') = iter (dfs v) (X', minV',L,i+1) (filter (fn x => x<>p) (nth newG v)) 18. val minV''' = updatemin p v minV'' 19. val L'' = bridge p v i L' minV'''(\*finish v\*) 20. in (X'', minV''',L'', i') 21. end 22. val res = (fn(\_,\_,l,\_) =>l)(dfs (numV-1) ((initX,initminV,nil,0),numV-1)) 23. in filter(fn(p,\_)=>p<>numV-1)(fromList res) 24. end |
| 三. 复杂度分析：   * line 3 work = O(|V|), span = O(1) * line 4 work = span = O(1) * line 5 work = O(|V|), span = O(1) * line 6 work = O(|V|), span = O(1) * line 10 每个点都会被恰好discover一次，所以单独考虑每个点v被discover的那一轮dfs的work，除了递归操作外，对邻接点集Gv去filter一次的work是O(|Gv|)，而∑Gv = O(|E|)，从而，所有轮中filter总的work是O(|E|)，而对v点dfs时候的其他操作都是线性的，从而所有轮中其他操作的总的work是O(|V|)。从而dfs总的work是O(|E|+|V|)。 * line 23 Wfilter = O(|V|), WfromList = O(|E|)   综上知：总的work = O(|E|+|V|)，也可知总的span = O(|E|+|V|) |

#### 2.1.4 回答关于Dijkstra’s algorithm的相关问题

Task 5.1 (5%). Give an example of a graph on≤4 vertices with negative edge weights where Dijkstra’s algorithm fails to ﬁnd the shortest paths. List the priority queue operations (i.e. insertions and updates of shortest path lengths) up to the point of failure. Clearly point out where the algorithm has failed and what it should have done.

|  |
| --- |
| Input: {0->{1->1.0, 2->0.0}, 1->{2->-2.0}} and to find the shortest path from 0 to 2 by Dijkstra’s algorithm.   1. initial PQ : Q = {(0.0, 0)} ; D = {} 2. PQ.deleteMin: Q = {} ; return SOME(0.0, 0) ; D = {(0, 0.0)} 3. PQ.insert : Q = {(0.0, 2), (1.0, 1)}; 4. PQ.deleteMin: Q = {(1.0, 1)}; return SOME(0.0, 2) So the work of get the path to 2 has completed! However, the shortest path from 0 to 2 is 0->1->2 and its weight is -1.0 less than the 0.0 . |

Task 5.2 (5%). Assuming there are no negative-weight cycles in G, how would you modify Dijkstra’s to accomodate negative edge weights and return the correct solution?

|  |
| --- |
| Using Bellman Ford algorithm. Denote dk(v) is the shortest path whose length is not exceeding k, and use the recursion dk(v) = min(dk(v), minx∈N(v)(dk(v)+w(x,v))) |

#### 2.1.5 回答关于启发式函数的一些问题

Task 5.3 (5%). Brieﬂy argue why the Euclidean distance heuristic is both admissible and consistent for edge weights that represent distances between vertices in Euclidean space.

|  |
| --- |
| Admissible:  由于两点之间线段距离最短，所以Euclidean distance heuristic 是admissible的。  Consistent：  由三角不等式易知Euclidean distance heuristic is consistent. |

Task 5.4 (5%). Give a heuristic that causes A∗ to perform exactly as Dijkstra’s algorithm would.

|  |
| --- |
| h(v) = 0 for any v |

Task 5.5 (5%). Give an example of a weighted graph on≤4 vertices with a heuristic that is admissible but inconsistent, where the Dijkstra-based A\* algorithm fails to ﬁnd the shortest path from a single source s to a single target t. Label each vertex with its heuristic value, and clearly mark the vertices s and t. In 2-3 clear sentences, explain why the shortest path is not found (e.g., when does the algorithm fail, what exactly does it do wrong.)

|  |
| --- |
| Counterexample：  G = {0->{1->3.0, 2->1.0}, 1->{3->2.0}, 2->{1->1.0}}  s = 0, t = 3  h(0) = 0.0, h(3) = 0.0, h(2) = 3.0, h(1) = 0.0  不难验证，该heuristic function 满足admissible but inconsistent用Dijkstra-based A\* algorithm去找出的最短路径为0->1->3，而事实上最短路径是0->2->1->3，返回的并不是正确结果。 |

Task 5.6 (5%). Give an example of a weighted graph on≤4 vertices with heuristic values that are inadmissible, where the A∗ algorithm fails to ﬁnd the shortest path from a single source to a single target. Again, clearly label your vertices with heuristic values and explain why the shortest path is not found.

|  |
| --- |
| G = {0->{1->1.0, 2->3.0}, 1->{2->1.0}}  h(0) = 0.0, h(2) = 0.0, h(1) = 3.0  不难验证，该heuristic function不满足admissible，返回的路径是0->2，而事实上最短路径是0->1->2，并未返回正确结果。 |

### 2.2 MkAStarCore的实现

#### 2.2.1 完成makeGraph函数

Task 5.7 (5%). Write the function

val makeGraph : edge seq -> graph

which takes a sequence of edges representing a graph and returns the same graph conforming to the provided graph type. Each edge is represented as a triple (u,v,w) representing a directed edge from u to v with weight w. You may assume that all weights are non-negative. For full credit, makeGraph should have O(|E|log|V|) work and O(log2|V|) span.

|  |
| --- |
| Graph类型的定义：type graph = weight table table |
| 代码实现：   1. fun makeGraph (E : edge Seq.seq) : graph = 2. Table.map Table.fromSeq (Table.collect (Seq.map (fn (u,v,w) => (u,(v,w))) E)) |
| 复杂度分析：   1. WSeq.map = O(|E|), SSeq.map = O(1) 2. WTable.collect = O(|E|log|E|), STable.collect = O(log2|E|) 3. WTable.map = O(|E|log|E|), STable.map = O(log2|E|)   综上知：makeGraph函数的Work = O(|E|log|V|), Span = O(log2|V|) |

#### 2.2.2 完成findPath函数

Task 5.8 (30%). You will implement the function

val findPath : heuristic -> graph -> (set \* set) -> (vertex \* real) option

which augments Dijkstra’s Algorithm to accept the following arguments:

1. An A∗ heuristic, h, assuming that h is both admissible and consistent.

2. Multiple source vertices, S⊆V.

3. Multiple target vertices, T ⊆ V. If multiple sources and destinations are given, your algorithm should return the shortest path distance between any s∈S and any t∈T (a shortest S−T path). Speciﬁcally, findPath h G (S,T) should evaluate to SOME (v,d) if the shortest S−T path in G ends at vertex v∈T with distance d, or NONE if no such path exists. If there are multiple shortest paths, you may return any one. The asymptotic complexity of your algorithm should not exceed that of Dijkstra’s algorithm as discussed in lecture.

|  |
| --- |
| 1. fun findPath h G (S, T) = 2. let 3. fun N(v) = 4. case Table.find G v 5. of NONE => Table.empty() 6. |SOME nbr => nbr 7. fun astar D Q = 8. case PQ.deleteMin Q 9. of (NONE,\_) => NONE 10. | (SOME (d,v),Q') =>(\*d is the actual distance add h v\*) 11. if Set.find T v then SOME(v,d - h v) else 12. case Table.find D v 13. of SOME \_ =>astar D Q' 14. |NONE => 15. let 16. val insert = Table.insert (fn \_ => raise SegmentFault) 17. val D' = insert (v,d - h v) D 18. fun relax (q,(u,w)) = PQ.insert ((d - h v + w + h u),u) q 19. val Q'' = Table.iter relax Q' (N v) 20. in astar D' Q'' 21. end 22. val initpq = PQ.fromList (Seq.toList (Seq.map (fn p => (h p, p)) (Set.toSeq S))) 23. in astar (Table.empty()) initpq 24. end |
| 复杂度分析： 该算法只是将Dijkstra’s algorithm 中压入压出优先队列的内容稍加改变，其他地方做细微的处理，很容易可以看出findPath的asymtotic complexity不会超过Dijkstra’s algorithrm，即有work仍为|E|log |V|。 |

# Lab7 图像分割实验

## 1. 实验要求

本次实验你将完成寻找最小生成树以及基于MST的图像分割算法。

## 2. 完成代码

### 2.1 完成函数MST

Task 4.1 (40%). Implement the function

MST : edge seq \* int -> edge seq

in MkBoruvkaMST.sml where MST (E, n) computes the minimum spanning tree of the graph represented by the input edge sequence E using Bor˚uvka’s Algorithm. There will be n vertices in the graph, labeled from 0 to n−1. Recall that you do not need to manually reverse edges – assume that if the edge (x, y,w) appears in the input, then so does (y,x,w) . For full credit, your solution must have expected O(mlogn+n) work and expected O(logk n) span for some k, where n is the number of vertices and m is the number of edges.

|  |
| --- |
| * 算法思路   类似于书上的Boruvka算法伪代码，其中利用inject排好序的边对每个点去找最小边权。 |
| * 代码实现：  1. fun MST (E : edge seq, n : int) : edge seq = 2. let 3. fun cmp ((\_,\_,w1),(\_,\_,w2)) = 4. if w1 > w2 then LESS else if w1 < w2 then GREATER else EQUAL 5. val edge = sort cmp E (\*sort edges E from largest to smallest by weight\*) 6. val initedge = enum (map (fn (v1,v2,\_) => (v1,v2)) edge) (\*map with lable\*) 7. val initseed = Rand.fromInt 23333 8. fun helper(orderE,seed,T) = 9. if length orderE = 0 then flatten (%T) 10. else 11. let 12. val coins = Rand.flip seed n 13. val ET = map (fn (l,(v1,v2)) => (v1,SOME(l,(v1,v2)))) orderE 14. val minE = inject ET (tabulate (fn v => NONE) n) 15. val P = map (fn x => case x of NONE => NONE 16. | SOME(l,(v1,v2))=> if nth coins v1 = 0 andalso nth coins v2 = 1 then SOME(l,v2) else NONE) minE 17. val selectE = filter (fn x => x<>NONE) P 18. val T' = map (fn(SOME(l,\_))=>l) selectE 19. val self = map (fn SOME(l,v) => (v,SOME(l,v))) selectE 20. val P' = inject self P 21. fun produce (l,(v1,v2)) = case (nth P' v1,nth P' v2) of 22. (SOME(\_,v3),SOME(\_,v4)) => if v3 = v4 then empty() else singleton(l,(v3,v4))(\*drop the edges in the same partition\*) 23. |(NONE,SOME(\_,v)) => singleton(l,(v1,v)) 24. |(SOME(\_,v),NONE) => singleton(l,(v,v2)) 25. |(NONE,NONE) => singleton(l,(v1,v2)) 26. val orderE' = flatten (map produce orderE) 27. in helper(orderE',Rand.next seed,T'::T) 28. end 29. in map (fn i => nth edge i) (helper(initedge,initseed,nil)) 30. end |
| * 复杂度分析：   + line 5 Wsort = O(mlog m) Ssort = O(log2m)   + line 6 W = O(n) S = O(1)   以下考虑一轮helper函数的复杂度   * + line 26 WRand.flip = O(n) SRand.flip = O(log n)   + line 27 Wmap = O(m) Smap = O(1)   + line 28 Winject = O(n+m) Sinject = O(1)   + line 29 Wmap = O(m) Smap = O(1)   + line 31 Wfilter = O(n) S­filter = O(log n)   + line 32 Wmap = O(n) Smap = O(1)   + line 34 Winject = O(n+m) Sinject = O(1)   + line 40 W = O(m) S = O(log m)   所以可以看出每一轮helper函数总的work = O(n+m)，span = O(log m)  轮数的期望是O(log n)，类似书上的证明，算总的work和span的期望的时候，可以认为轮数是O(log n)，又在连通简单图中，所以总的期望Work = O(mlog n),Span = O(log2n) |

### 2.2 完成函数findSegment并简述思路与复杂度

Task 5.1 (25%). Implement the function

findSegment: (edge Seq.seq \* int) -> int -> vertex Seq.seq

where findSegment (E, n) c computes the image segmentation of a picture (that is represented as graph). E is the input edge sequence, n is the number of vertices in the graph, and c is the initial credit for each vertex. An edge Seq.seq is deﬁned as a sequence of vertex \* vertex \* weight, where the type of vertex and the type of weight are both int. These types can also be found in support/SEGMENTER.sig. Once again, you may assume that the input edge sequence already contains edges in both directions (as we are dealing with an undirected graph). Your solution must have expected O(mlog2 n) work and expected O(logk n) span for some k, where n is the number of vertices and m is the number of edges.

|  |
| --- |
| * 算法简述：总体是利用基于Bruvka算法的MST，其中收缩的时候记录下收缩的star和satelite，然后进行collect与map，进而得到新的点及其流量，其余部分皆类似于前面的MST算法，最后生成了新的边之后，对边做filter处理之后进入下一轮递归即可。 |
| * 代码实现：  1. fun findSegments (E, n) initial\_credit = 2. let 3. fun cmp ((\_,\_,w1),(\_,\_,w2)) = 4. if w1 > w2 then LESS else if w1 < w2 then GREATER else EQUAL 5. val initedge = sort cmp E (\*sort edges E from largest to smallest\*) 6. val initseed = Rand.fromInt 233 7. val initV = tabulate (fn \_ => initial\_credit) n 8. fun helper(E,V,seed) = 9. if length E = 0 then inject (mapIdx (fn (i,\_) => (i,i)) V) (tabulate (fn i => ~1) n) 10. else 11. let 12. val coins = Rand.flip seed n 13. val ET = map (fn (v1,v2,w) => (v1,SOME (v1,v2,w))) E 14. val minE = inject ET (tabulate (fn v => NONE) n) 15. val P = map (fn x => case x of NONE => NONE 16. |SOME(v1,v2,w) => if nth coins v1 = 0 andalso nth coins v2 = 1 then SOME (v1,v2,w) else NONE) minE 17. val P' = filter (fn x => x<> NONE) P 18. fun cmp(a,b) = if a>b then GREATER else if a<b then LESS else EQUAL 19. val revP = collect cmp (map (fn SOME(v1,v2,w) => (v2,(v1,w,nth V v1))) P') 20. fun newV (v,s) = reduce (fn ((\_,w1,crdt1),(\_,w2,crdt2)) => (v, w1 + w2,Int.min(crdt1,crdt2))) (v,0,nth V v) s 21. val head = map (fn (v,w,crdt) => (v, crdt - w)) (map newV revP) 22. val V' = inject head V (\*finish producing new vertices\*) 23. val P'' = inject (map (fn (v,\_) => (v,SOME(v,v,0))) head) P(\*add v to self\*) 24. fun produce (v1,v2,w) = case (nth P'' v1,nth P'' v2) of 25. (SOME(\_,v3,\_),SOME(\_,v4,\_)) => if v3 = v4 then empty() else singleton(v3,v4,w) 26. |(NONE,SOME(\_,v,\_)) => singleton(v1,v,w) 27. |(SOME(\_,v,\_),NONE) => singleton(v,v2,w) 28. |(NONE,NONE) => singleton(v1,v2,w) 29. val E' = flatten (map produce E) (\*finish produce edges\*) 30. val selectE = map (fn SOME(x,y,\_) => (x,y)) P' 31. fun dealedge (v1,v2,w) = if ((nth V v1) + (nth V v2) - w) >=0 then true else false 32. val sv = helper(filter dealedge E',V',Rand.next seed) 33. val upd = map (fn (v1,v2) => (v1, nth sv v2)) selectE 34. in inject upd sv 35. end 36. in helper(initedge,initV,initseed) 37. end |
| * 复杂度分析：   + line 5 Wsort = O(mlog m) Ssort = O(log2m)   + line 7 Wtabulate = O(n) Stabulate = O(1)   以下考虑一轮helper函数的复杂度   * + line 9 W = O(n) S = O(1)   + line 12 WRand.flip = O(n) SRand.flip = O(log n)   + line 13 Wmap = O(m) Smap = O(1)   + line 14 W= O(m) S = O(1)   + line 15 Wmap = O(m) Smap = O(1)   + line 16 Wmap = O(m) Smap = O(1)   + line 17 Wfilter = O(n) Sfilter = O(log n)   + line 19 W= O(nlog n) S = O(log2n)   + line 21 W = O(m+n) S = O(log n)   + line 22 Winject = O(n) Sinject = O(1)   + line 23 W = O(n) S = O(1)   + line 29 W =O(m) S = O(log m)   + line 30 W = O(n) S = O(1)   + line 32 Wfilter = O(m) Sfilter = O(log m)   + line 33 Wmap = O(m) Smap = O(1)   + line 34 Winject = O(n) Sinject = O(1)   从而期望O(log n)轮后，总的work期望为O(mlog n)，span期望为O(log3n) |

## 3.书面习题

**Task 6.1** (10%). Second-best is good enough for my MST. Let G = (V,E) be a simple, connected, undirected graph G =(V,E) with|E|≥2 and distinct edge weights. We know for a fact that the smallest (i.e., least heavy) edge of G must be in the minimum spanning tree (MST) of G. Prove that the 2nd smallest edge of G must also be in the minimum spanning tree of G.

|  |
| --- |
| 证：  首先知道边权不同的的图中最小生成树的一个性质，最小的跨割的边一定属于最小生成树。于是考虑图G中边权第二小的边e，设其连接的两个顶点为a,b，这两点中至少有一点与最小权值的边不相邻，不妨设为a点，于是e为跨{a}与G/{a}的割的最小权值边，所以e一定在图G的最小生成树中。 |

**Task 6.2** (10%). I Prefer Chalk. There is a very unusual street in your neighborhood. This street forms a perfect circle, and there are n≥3 houses on this street. As the unusual mayor of this unusual neighborhood, you decide to hold an unusual lottery. Each house is assigned a random number r ∈R [0,1] (drawn uniformly at random). Any house that receives a larger number than both of its two neighbors wins a prize package consisting of a whiteboard marker and two pieces of chalk in celebration of education. What is the expected number of prize packages given? Justify your answer.

|  |
| --- |
| 解：由于n个实数中出现相等的数的概率为0，所以只需考虑n个实数都不想等的情况，设这n个实数按从小往大的顺序排列为a1,a2,…,an。为了方便，称这n个房子为点1， 点2，点3，……点n。下面求其中点j处的实数x比周围两点都要大的事件为Aj。  P(x = ai) = 1/n, 从而P(Aj | x = ai) = C(i-1,2) / C(n-1,2)，于是有  P(Aj) = = 1/3  Define Xj = IAj , X为prize packages总数量­ （I是示性函数）  E(X) = ∑E(Xj) = ∑P(Aj) = n/3 |

**Task 6.3** (15%). It’s Probably Linear. For all n > 1, let Xn be a random variable with Xn≤n. Let f be a non-decreasing function satisfying f(n)≤ f(Xn)+Θ(n), where f(1)=1.Prove that if E[Xn]= n/3, then (n) =E[f(n)]∈O(n).

Hint: what is Pr[Xn≥2n/3]? Use Markov’s Inequality, covered in recitation 9.

|  |
| --- |
| 证明：  由Markov’s Inequality得：P(Xn≥2n/3) ≤ (n/3)/(2n/3) = 1/2  由f(n)≤ f(Xn)+Θ(n)，两边取期望，得：  E[f(n)] ≤ E[f(Xn)] +Θ(n)  = E[f(Xn)|Xn≥2n/3] \* P(Xn≥2n/3) + E[f(Xn)|Xn<2n/3] \* P(Xn<2n/3) +Θ(n)  (\*全期望公式\*)  = (E[f(Xn)|Xn≥2n/3] - E[f(Xn)|Xn<2n/3]) \* P(Xn≥2n/3) + E[f(Xn)|Xn<2n/3] +Θ(n)  ≤ (E[f(Xn)|Xn≥2n/3] - E[f(Xn)|Xn<2n/3]) \* 1/2 + E[f(Xn)|Xn<2n/3] +Θ(n)  = E[f(Xn)|Xn≥2n/3] \* 1/2 + E[f(Xn)|Xn<2n/3] \* 1/2 + Θ(n)  ≤ E[f(n)] \* 1/2 + E(⌊2n/3⌋) \*1/2 + Θ(n)  得到：E[f(n)] ≤ E(⌊2n/3⌋) +Θ(n)  由主定理易得 E[f(n)] ∈O(n). |

# Lab8 范围搜索实验

## 1. 实验要求

本次实验你将基于BST扩展order table 的ADT接口，完成一些基本函数，你可以从一般的库函数出发扩展此库。此外，你将完成一个范围搜索实验，即给定一个二维点集，以及一个矩形（用左上和右下坐标表示）范围，找出在此范围内点的个数，你需要自定义数据结构以满足复杂度需求。

## 2. 回答问题

### 2.1 MkBSTOrderedTable的实现

#### 2.1.1完成函数first，last简述思路。

Task 4.1 (6%). Implement the functions

fun first (T : 'a table) : (key \* 'a) option fun last (T : 'a table) : (key \* 'a) option Given an ordered table T, first T should evaluate to SOME (k,v) iff (k,v)∈T and k is the minimum key in T. Analagously, last T should evaluate to SOME (k,v) iff (k,v)∈T and k is the maximum key in T. Otherwise, they evaluate to NONE.

|  |
| --- |
| * 算法思路：   利用递归的思想，first T当T的左子树l非空时，就是first l，左子树为空时就是本身的SOME(KEY,VAL)。对于last是类似的道理。 |
| * 代码实现：  1. fun first (T : 'a table) : (key \* 'a) option = 2. case Tree.expose T of 3. NONE => NONE 4. |SOME{left = l, key = k, value = v, ...} => 5. case first l of NONE => SOME(k,v) 6. |x => x 7. fun last (T : 'a table) : (key \* 'a) option = 8. case Tree.expose T of 9. NONE => NONE (\*base case\*) 10. |SOME{right = r, key = k, value = v, ...}=> 11. case last r of NONE => SOME(k,v) 12. |x => x |

#### 2.1.2 完成函数previous和next并简述思路。

Task 4.2 (8%). Implement the functions

fun previous (T : 'a table) (k : key) : (key \* 'a) option fun next (T : 'a table) (k : key) : (key \* 'a) option Given an ordered table T and a key k, previous T k should evaluate to SOME (k',v) if (k0,v)∈ T and k0 is the greatest key in T strictly less than k. Otherwise, it evaluates to NONE. Similarly, next T k should evaluate to SOME (k',v) iff k0 is the least key in T strictly greater than k.

|  |
| --- |
| * 算法思路：   Previous就是左子树的last，next就是右子树的first |
| * 代码实现：  1. fun previous (T : 'a table) (k : key) : (key \* 'a) option = 2. last (#1 (Tree.splitAt (T,k))) 3. fun next (T : 'a table) (k : key) : (key \* 'a) option = 4. first (#3 (Tree.splitAt (T,k))) |

#### 2.1.3 完成函数join, split

Task 4.3 (2%). Implement the function

fun join (L : 'a table, R : 'a table) : 'a table

Given ordered tables L and R, where all the keys in L are strictly less than those in R, join (L, R) should evaluate to an ordered table containing all the keys from both L and R.

|  |
| --- |
| fun join (L : 'a table, R : 'a table) : 'a table =  Tree.join (L,R) |

Task 4.4 (2%). Implement the function

fun split (L : 'a table, k : key) : 'a table \* 'a option \* 'a table

Given an ordered table T and a key k, split should evaluate to a triple consisting of

1. an ordered table containing every (k',v)∈T such that k' < k,

2. SOME v if (k,v)∈T and NONE otherwise, and

3. an ordered table containing every (k',v)∈T such that k'> k.

|  |
| --- |
| fun split (T : 'a table, k : key) : 'a table \* 'a option \* 'a table =  Tree.splitAt(T,k) |

#### 2.2.4完成函数getRange并简述思路

Task 4.5 (7%). Implement the function

fun getRange (T : 'a table) (low : key, high : key) : 'a table

Given an ordered table T and keys l and h, getRange T (l, h) should evaluate to an ordered table containing every (k,v)∈T such that l≤k≤h.

|  |
| --- |
| * 算法思路：   先用下界low去split，得到大于等于low的table部分，再用上界去split即可；处理下边界。 |
| * 代码实现：  1. fun getRange (T : 'a table) (low : key, high : key) : 'a table = 2. let 3. val lt = case split (T, low) of 4. (\_, NONE, r) => r 5. |(\_,SOME v,r) => join(Tree.singleton(low,v),r) 6. val hlt = case split (lt, high) of 7. (l,NONE,\_) => l 8. |(l,SOME v, \_) => join(l, Tree.singleton(high, v)) 9. in hlt 10. end |

### 2.2 MkRangeCount的实现

#### 2.2.1 完成函数makeCountTable并回答相关问题

Task 5.1 (25%). In the MkRangeQuery functor, deﬁne the countTable type and implement the function

fun makeCountTable: point seq -> countTable

The type point is deﬁned to be OrdTable.Key.t \* OrdTable.Key.t where OrdTable is an ordered table structure provided to you. You should choose the type of countTable such that you can implement count (range queries) in O(log n) work and span. For full credit, your makeCountTable must run within O(nlogn) expected work.

|  |
| --- |
| 1. 代码实现： 2. type countTable = unit table table (\*set table\*) 3. exception FindTheSameYCoordinate 4. fun makeCountTable (S : point seq) : countTable = 5. let 6. val orderX = fromSeq S 7. fun inserty(t,(\_,y)) = insert (fn \_ => raise FindTheSameYCoordinate) (y,()) t 8. in iterih inserty (empty()) orderX 9. end |
| 1. 复杂度分析：   WfromSeq = O(nlog n)  Winserty = O(log n)  Witerih = ∑Witerty = O(nlog n) |

Task 5.2 (10%). Brieﬂy describe how you would parallelize your code so that it runs in O(log2 n) span. Does the work remain the same? You don’t need to formally prove the bounds, just brieﬂy justify them.

|  |
| --- |
| 并行代码使其span为O(log2n)，可以这样做：   1. 先对point按x坐标进行排序 2. 然后将 (x,y) map成(x,table.singleton(y,()))，得到sequence S. 3. 进行scani (fn ((\_,y1),(x,y2)) => (x, merge(y1,y2))) S 4. 最后Table.fromSeq即可实现相同的效果。   Cost Analysis:  每一步对应的cost   1. Wsort = O(nlog n), Ssort = O(log2n) 2. Wmap = O(n), Smap = O(1) 3. 这时，scani f b S中f的work不能再当作常数，所以需要考虑scani的具体实现。   不失一般性，为简化思考，可以设n是2的幂。易知递归的第t层，每个元素作为table的size是2t，即共有n/2t个这样的元素进行scani，易得到递推公式：  W(n/2t) = W(n/2t+1) + O(2t \* n/2t+1)  S(n/2t) = S(n/2t+1) + O(t)  亦即W(m) = W(m/2) + O(n)  S(m) = S(m/2) + O(log n – log m)  从而W(n) = O(nlog n), S(n) = O(log2n)   1. WTable.fromSeq = O(nlog n), STable.fromSeq = O(log2n)   由以上分析知，该算法的span达到了O(log2n)，且work的渐进复杂度保持不变，仍为O(nlog n). |

Task 5.3 (5%). What is the expected space complexity of your countTable in terms of n the number of input points? That is, how many nodes in the underlying binary search tree(s) does your countTable use in expectation? Explain in a few short sentences.

|  |
| --- |
| 由于空间复杂度不会超过时间复杂度（串行），所以空间复杂度为O(nlog n)，之所以不是简单的把所有树的size相加，是因为在tree的insert操作中，存在着很大程度的引用，不是创建一棵新的树就要创建所有结点。 |

#### 2.2.2 完成函数count

Task 5.4 (25%). Implement the function

count: countTable -> point \* point -> int

Asdescribedearlier, count T ((x\_1,y\_1), (x\_2, y\_2))willreportthenumberofpointswithinthe rectangle with the top-left corner (x1, y1) and bottom-right corner (x2, y2). Your function should return the number of points within and on the boundary of the rectangle. You may ﬁnd the OrdTable.size function useful here. Your implementation should have O(logn) work and span.

|  |
| --- |
| 1. fun count (T : countTable) 2. ((xLeft, yHi) : point, (xRght, yLo) : point) : int = 3. let 4. val lbound = #1(split (T, xLeft)) 5. val rbound = case split (T, xRght) of 6. (t, NONE, \_) => t 7. | (t, SOME e, \_) => insert (fn(x,\_)=>x) (xRght, e) t 8. fun ysize t = size (getRange t (yLo, yHi)) 9. fun opttot(SOME (\_, v)) = v 10. | opttot NONE = empty() 11. in ysize(opttot(last rbound)) - ysize(opttot(last lbound)) 12. end   由于split, insert, getrange的work都为O(log n)知  总的work = span = O(log n) |