

**课 程 设 计 报 告**

**课程名称： 串并行与数据结构及算法**

**专业班级： ACM1601**

**学 号： U201614831**

**姓 名： 苏墨馨**

**指导教师： 陆枫**

**报告日期： 　2018年3月5日**

**计算机科学与技术学院**

## 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. 回答问题

#### 2.0 choose函数思路

|  |
| --- |
| * 算法思路：   先判断如果hist为空或者p超出0到1的范围，输出Range异常。取出所有的hist里的int即表示所有的出现次数，然后对其用scanI求和得到概率密度和次数和的乘积组成的串s3。将p乘以次数和得到要在s3中找的数b。如果b比s3的第i个数小且比s3的第i+1个数大则记录其位置，再回到hist里找对应的词。 |

#### 2.1 choose函数代码实现

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.

|  |
| --- |
| * 代码实现：   fun choose (hist : 'a hist) (p : real) : 'a =  if length hist = 0 then raise Range  else if p > 1.0 andalso p < 0.0 then raise Range  else  let  val s1 = map #2 hist  val s2 = scani op+ 0 s1(\*得到概率密度\*)  val a = nth s2 (length s2 - 1 )(\*次数和\*)  val s3 = map (fn i => real(i)) s2  val s4 = zip (tabulate (fn i=>i) (length s3) ) s3  val b = p\*(Real.fromInt a)  fun fil (0,x) = if b <= x then true else false  |fil (i,x) = if b <= x andalso b >(nth s3 (i-1)) then true else false  val s5 = filter fil s4  in  #1 (nth hist (#1 (nth s5 0)))  end       * 复杂度分析：设n为hist的长度   Work：函数主要的work是map scani zip以及filter的操作，他们的work都是O(n)的，nth length等函数的work和span都是O(1),可以忽略,因此总的work也是O(n)的；  Span：由于map和zip可以并行进行，因此它们的span为O(1)，可以忽略，而zip和filter的span为O(log n )所以总的span为O(log n)的。 |

#### 2.2 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?

|  |
| --- |
| * 测试样例：      * 测试结果：     Tester对于负数的结果处理有误，Exn Range是正确结果 |

#### 3.0 kgramstats ADT类型确定

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 \* (string hist T.table) * 因为需要实现通过kgram查找所有出现在kgram后面的所有tokens的hist，所以用string hist table保存其中key为kgram，value为kgram后面的tokens及次数的串。另外为了快速得到maxk，用一个int存maxk。所以定义kgramstats的类型为int \* (string hist T.table) |

#### 3.1 K\_GRAM 文件中所有代码实现

##### 3.1.1.0 makeStats函数思路

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.

|  |
| --- |
| * 算法思路：   用tokens函数从语料库中找出token的seq。对每个位置i找出以i为开头以len为长度的串，并且和其后第一个单词组成一个二元组即得到(kgram,token)的串，做collect得到(kgram,token seq)seq再用histogram函数得到token串的hist，转换为table完成kgramstats的后半部分。再记录maxk，得到kgramstats的二元组。 |

#### 3.1.1.1 makeStats函数实现及复杂度分析

|  |
| --- |
| * 代码实现：   fun makeStats (corpus : string) (maxK : int) : kgramstats =  let  val tok = tokens (not o Char.isAlphaNum) corpus  fun cmp (a,b) = collate String.compare(a,b) (\*排序比较\*)    fun find len = tabulate (fn i => (subseq tok (i,len), nth tok (len + i))) (length tok - len) (\*长度len\*)  (\* fun find k = tabulate (fn i => (subseq tok (i,(k-i+1)), nth tok (k+1))) k  val start = tabulate (fn i => i) (maxK + 1)  val findk = flatten (map find start)\*)  val findk = flatten (tabulate find (maxK + 1))  (\* val final = filter (fn (i,s) => if (length i) > maxK then false else true) findk \*)  val tohis = collect cmp findk (\*key\*)  val his = map (fn (i,s) => (i,histogram String.compare s)) tohis  in  (maxK,Table.fromSeq his)  end     * 复杂度分析：   函数主要是tokens,flatten,tabulate,collect,map和fromSeq的操作，其中，tokens,flatten,tabulate的work都是O(n)，span都是 O(log n)，map的work为O(n),span为O(1), collect和fromSeq的work是O( nlogn)，span是)。因此总的work为O(n),span为) |

#### 3.1.2.0 lookupExts函数的思路

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.

|  |
| --- |
| * 算法思路：   直接用find函数从kgramstats的第二个元素中找hist |

#### 3.1.2.1 lookupExts函数的实现及复杂度分析

|  |
| --- |
| * 代码实现：   fun lookupExts (stats : kgramstats) (kgram : kgram) : (token \* int) seq =  case Table.find (#2 stats) kgram of  SOME t => t  |NONE => empty()     * 复杂度分析：   总的work和span即Table.find的work和span为O(log n) |

#### 3.1.3.0 maxK函数的代码及复杂度分析

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.

|  |
| --- |
| * 代码实现   fun maxK (stats : kgramstats) : int = #1 stats     * 复杂度分析：只需要取出二元组中的一个元素，复杂度为O(1) |

#### 3.2 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?

|  |
| --- |
| * 测试样例：      * 测试结果： |

#### 3.3.0 randomSentence函数思路

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 shouldhaveO(n(WlookupExts+Wchoose))workandO(n(SlookupExts+Schoose))span, assuming that the words in the corpus have constant length. The output string should end with a period andnothaveanyleadingspaces. Youmightﬁnd String.concatWith : string -> string list -> string to be useful.

|  |
| --- |
| * 算法思路：   每次从句子中找出不超过maxk的最大串作为kgram，利用lookupExts函数和choose函数选择下一个单词，形成新句子。生成一个随机数序列作为下一个单词的p，用iter完成。最后用String.concatWith函数加句号。注意到如果hist为空则去掉kgram的最后一个单词作为新的kgram来形成新句子。 |

#### 3.3.1 randomSentence 函数的实现及其复杂度

|  |
| --- |
| * 代码实现：   fun randomSentence (stats : KS.kgramstats) (n : int) (seed : R.rand) =  let  val maxk = KS.maxK stats  (\*选随机数作为下一个单词的p\*)  fun next (sentence,i) =  let  val kgram = drop (sentence,Int.max(0,(length sentence - maxk)))  val histogram = KS.lookupExts stats kgram  val nword = if (length histogram = 0)  then  let  val kgram\_edge = take (kgram,(length kgram)-1)  val histogram\_edge = KS.lookupExts stats kgram\_edge  in  Util.choose (histogram\_edge) i  end  else Util.choose (histogram) i  (\*val nsentence = append(sentence,singleton nword)\*)  in  append (sentence, singleton nword)  end  val ranseq = R.randomRealSeq seed NONE n  val result = iter next (empty()) ranseq  in  (String.concatWith " " (toList result)) ^ "."(\*加句号\*)  end     * 复杂度分析：   next函数的复杂度主要由append，choose和lookupExts决定，他们的work都是O(m)的，choose和lookupExt的span为O(logm),append的 span 为O(1)，在O意义下忽略append的复杂度得到next函数的work为*W*lookupExts+*W*choose，span为*S*lookupExts+*S*choose。iter是完全串行的，因此它的work为*O*(*n*(*W*lookupExts+*W*choose))，span为*O*(*n*(*S*lookupExts+*S*choose)) |

#### 3.4.0 randomDocument函数的思路

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.

|  |
| --- |
| * 算法思路：   用n生成两个随机数序列，其中一个在5到10之间，作为随机产生的句子的长度，另一个作为random sentence的种子。再用concatWith加空格。 |

#### 3.4.1 randomDocument函数的实现及复杂度分析

|  |
| --- |
| * 代码实现：   fun randomDocument (stats : KS.kgramstats) (n : int) (seed : R.rand) =  let  val senlen = R.randomIntSeq seed (SOME(5,10)) n(\*句子长度\*)  val seedseq = R.randomIntSeq seed NONE n(\*random sentence的种子\*)  fun new i =  let  val len = nth senlen i  val nseed = R.fromInt (nth seedseq i)  in  randomSentence stats len nseed  end  val doc = tabulate new n  in  String.concatWith " " (toList doc)  end     * 复杂度分析：   函数的work主要是用tabulate产生n个random sentence的work，为*O*(*nW*randomSentence)，span也是产生n个random sentence的span，但是由于句子还需要用concatWith加空格，因此总的span是 *O*(*n* + *S*randomSentence)   * 测试结果： |

#### 3.5 skylineLab reload

#### 3.5.1 回答或证明下列问题

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

|  |
| --- |
| Work为O(n) span为O(1) |

Task 7.2(10%). Give pseudocode for a paralle limplementationof 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)

|  |
| --- |
|  |

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：为了解决这个问题，至少需要访问一遍所有的数字，所以work至少为O(n)，每次访问可以并行进行，因此span为 O(1)。 |

## 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. 回答问题

#### 2.1.0 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 = ((vertex seq) table)\*int\*int   以邻接表的形势存储图，后面用两个int分别储存边数和点数便于后面的查找。 |

#### 2.1.1 简述makeGraph函数的思路

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.

|  |
| --- |
| * 算法思路：  1. 边数是edge seq的长度 2. 用Table.collect得到邻接表 3. 考虑到只入不出的点，先将图转换为无向图再求邻接表，这个邻接表的大小即为点的数目 |

#### 2.1.2 makeGraph函数的代码实现及复杂度分析

|  |
| --- |
| * 代码实现：      * 复杂度分析：   因为length，fromList和size的work很小，可以忽略，所以函数主要的work来自于map flatten和Table.collect，map的work为O(|E|),span为O(1), flatten的work为O(|E|),span为O(log|E|),collect的work为O(|E|log|E|)，span为O(log2|E|)。因此总的work为O(|E|log|E|)，span为O(log2|E|) |

#### 2.1.3 numEdges和numVertices函数的实现

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 G 2. fun numVertices (G : graph) : int = #3 G 3. 复杂度分析：   只需要取出元组中的元素，work和span’都是O(1) |

#### 2.1.4 简述outNeighbors函数的思路

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.

|  |
| --- |
| * 算法思路：因为graph是一个邻接表，所以直接查找点v可以得到v的outneighbors |

#### outNeighbors函数的代码实现及复杂度分析

|  |
| --- |
| * 代码实现：   fun outNeighbors (G : graph) (v : vertex) : vertex seq =  case Table.find (#1 G) v of  NONE => empty()  |SOME t => t   * 复杂度分析：   算法主要是调用table.find函数，它的 work和span都为O(log|V|)，然后在记录结果的时候需要O(|Vout|)的work和O(log|Vout|) span，因为|Vout|≤|V|，因此总的work为O(|Vout|+log|V|) work ，span为O(log|V|)。 |

#### 2.1.6 ASP类型的确定及简短说明

Task4.5(2%). InMkAllShortestPaths.sml,deﬁnethetypeaspthatwouldallowyoutoimplement 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 table   vertex seq table中，key是点，value是在BFS时到达key点的上一级结点组成的seq，即为key点的parents点。 |

#### 2.1.7 简述makeASP函数的思路

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), makeASP 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).

|  |
| --- |
| * 算法思路：   从点v开始，对v进行BFS搜索，BFS的第i层的点代表它们到v的最短路距离为i。因为要在后面要实现只知道asp时找到所有的最短路，因此需要记录BFS访问点的parents。BFS记录asp的结果path(所有点目前的parents)，当前访问的点frontier，以及所有已经访问的点visited。每次在搜索时，找frontier的outneighbor里未被访问过的点进行访问，然后找到正在访问的点的父节点加到path里，再用正在访问的点更新frontier和visited，直到frontier为空时path即为asp。 |

#### 2.1.8 makeASP函数的代码实现及复杂度分析

|  |
| --- |
| * 代码实现：   fun BFS (G:graph)(path:asp) (frontier:Set.set) (visited:Set.set) : asp =  if (Set.size frontier = 0) then path  else  let  (\*在frontier里找每个点outneighbor里 未被访问的点作为下一层\*)  fun next1 u = Set.difference(Set.fromSeq(outNeighbors G u),visited)  fun next2 u = map (fn x => (x,u)) (Set.toSeq (next1 u))  val nex = map next2 (Set.toSeq frontier)  val nasp:asp = Table.collect (flatten nex)(\*正在访问的点的父节点\*)  val npath:asp = Table.merge (fn (a,b) => a) (path,nasp)  val nfrontier = Table.domain nasp  val nvisited = Set.union (nfrontier,visited)  in  BFS G npath nfrontier nvisited  end  fun makeASP (G : graph) (v : vertex) : asp =  let  val vpath = Table.singleton (v,singleton v)  val vfrontier = Set.fromSeq (singleton v)  in  BFS G vpath vfrontier (Set.empty())  end     * 复杂度分析：只分析一轮BFS,令n = |V|，m = |E|   对于每一轮BFS来说，work主要由找frontier里未被访问过的点，找到父节点并添加到path里以及更新frontier和visited构成。它们的复杂度和书上的一致如图所示。  在每一轮中，每条边和每个点的cost都为O(logn)，所有的点和边在整个BFS中都只出现一次，所以总的work为O(mlogn+nlogn) = O(mlogn),而span和BFS进行的次数有关，BFS进行的次数为最长的最短路的长度D，因此span为O(Dlog2n)，满足题目要求。 |

#### 2.1.9 简述report函数的思路

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.

|  |
| --- |
| * 算法思路：   在asp中用DFS找终点v的parents点，如果路径为空则返回empty(),如果只有一个点v，那么结果的vertex seq seq只包含点v；如果有多个点，那么继续对剩下的点进行DFS，将点加入路径中。因为asp里包含了所有点的BFS信息，所以可以找到所有的最短路。 |

#### 2.1.10 report函数的实现即复杂度分析

|  |
| --- |
| * 代码实现：   fun report (A : asp) (v : vertex) : vertex seq seq =  let  fun fpath x =  case Table.find A x of  NONE => Seq.empty()  |SOME path => path  fun DFS v : vertex seq seq =  if length (fpath v) = 0 then empty()  else if Table.Key.equal (nth (fpath v) 0, v) then Seq.singleton (Seq.singleton v)  else  let  val nexts\_data = fpath v  val nexts : vertex seq seq = flatten (map DFS nexts\_data)  in  map (fn y => append (y,Seq.singleton v)) nexts  end  in  DFS v  end     * 复杂度分析：   对一条路Table.find的操作work总共是O(|L|log|V|)，append的work都是O(|L|)，又因为有P条最短路，所以全部的work为O(|P||L|log|V|),DFS是串行进行的，因此span也是O(|P||L|log|V|)。 |

#### 2.1.11 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 toyou’re your implementation of functions of ALL\_SHORTEST\_PATHS against your test cases in Tests.sml.

|  |
| --- |
| * 测试样例：      * 测试结果： |

#### 2.1.12 定义thesaurus数据类型并简述理由

|  |
| --- |
| * type thesaurus = ASP.graph   每一个单词相当于一个点，如果两个单词是近义词，则它们两个点之间有边，将thesaurus问题转换为图的问题。 |

#### 2.1.13 实现下列函数并简述算法核心

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.

|  |
| --- |
| * 代码实现：   fun make (S : (string \* string seq) seq) : thesaurus =  let  val edge:(string\*string) seq seq = map (fn (a,b) => map (fn c => (a,c)) b) S  in  ASP.makeGraph (flatten edge)  end   * 算法简述：   将seq中的元素string\*string seq转换为(string \* string) seq，再整体进行flatten从而得到(string \* string) seq，这一个(string \* string) seq 相对于ASP问题中的edge seq，因此直接调用ASP.nakeGraph |

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 = ASP.numVertices T 2. fun synonyms (T : thesaurus) (w : string) : string seq = ASP.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.

|  |
| --- |
| * 代码实现：   fun query (T : thesaurus) (w1 : string) (w2 : string) : string seq seq =  let  val aspp = ASP.makeASP T w1  in  ASP.report aspp w2  end   * 算法思路：求两点之间的所有最短路和ASP中的report问题一样，所以用一个点求它的asp然后再调用report解决 |

#### 2.1.14 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：      * 测试结果： |

## 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不在联通，你可以参考tarjan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. 回答问题

#### 2.1.0定义ugraph的类型保存无向图的信息

|  |
| --- |
| 1. type ugraph = vertex seq seq 2. 按照点的顺序排列好的邻接seq |

#### 2.1.1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.

|  |
| --- |
| * 代码实现：   fun makeGraph (E : edge seq) : ugraph =  let  val dir = map (fn (a,b) => (b,a)) E  val directed = append(E,dir)  val ord = collect Int.compare directed(\*lable\*)  in  map (fn (a,b) => b) ord  End     * 功能描述：先把图转换为有向图，再用collect找到点的所有边并且将点按顺序排好序。 * 复杂度分析：   第一个map和append的work都是O(|E|)，span都是O(1)。第二个map的work是O(|V|),span是O(1)。所以算法的复杂度主要集中在Table.collect的操作上，work为O(|E|log|E|) ，span为O(log2|E|)。又因为|E|≤|V|2，所以符合O(|E|log|V|) work and O(log2|V|) span的要求。 |

#### 2.1.2 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.

|  |
| --- |
| * 算法思路：   利用Tarjan算法记录在DFS中点被访问的顺序(时间戳)dfn以及通过backedge回到的最早访问的节点的顺序low,如果子节点能回到的最早访问的节点的顺序在父节点的dfn之前，则会构成割边。从-1(添加一个点作为开始)点开始对整个图进行DFS，注意在找邻居节点的未访问节点(discover)时避免将无向图转换为有向图之后的自环，最后去掉添加的-1的点形成的割边。 |
| * 代码实现：   fun findBridges (G : ugraph) : edges =  let  val numv = length G  val vertices = tabulate (fn i => i) numv  val start = STSeq.fromSeq (tabulate (fn \_ => NONE) numv)  val maxlow = numv \* numv  fun dfs((X:int option STSeq.stseq, bridges:edge seq, dfn, low, u:vertex), v:vertex)=  case STSeq.nth X v of  SOME stamp =>(X, bridges, dfn, Int.min(low,stamp),u)(\*已访问\*)  |NONE =>  let  val newX = STSeq.update(v,SOME dfn) X  val unvisited = filter (fn x => if x = u then false else true) (nth G v) (\*如果v的邻居里有u 去掉u 避免u-v-u\*)(\*discover\*)  val (X2,nbridge,ndfn,nlow,\_) = iter dfs (newX,bridges,dfn+1,maxlow,v) (unvisited)  val finalb = if (nlow >= dfn) then append(nbridge,singleton(u,v)) else nbridge(\*Tarjan\*)  val finall = Int.min(low,nlow)(\*最早的stamp\*)  in  (X2,finalb,ndfn+1,finall,u)(\*finish\*)  end  val res = iter dfs (start,empty(),0,maxlow,~1) vertices  val resb = #2 res  in  filter (fn (a,b) => a <> ~1) resb (\* (~1,0) \*)  end |
| * 复杂度分析：   找初始值用到函数tabulate和STSeq.fromSeq的work和span分别为O(V),O(1)，函数DFS的实现和书上的DFS with single threaded arrays相同并且DFS是串行的，因此work和span与书上给出的相同，为O(|E|),因此总的work和span为O(|V|+|E|).（也与tarjan算法的复杂度相同） |
| * 测试样例：      * 测试结果： |

#### 2.1.3 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.

|  |
| --- |
| C:\Users\smx\Documents\Tencent Files\392558193\FileRecv\MobileFile\IMG_20180304_024934.jpg(s,0) -> (t,4) -> (u,6) 得到s到t的最短路径长为4  但是s到t的最短路径应该为s->u->t长为1。  Dijkstra算法没有考虑从u到t可能有负边导致的更短的路径的存在，没有走(u,t)的边，得到错误结果。 |

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?

|  |
| --- |
| 改成优先队列的SPFA算法，每次取出Q中的最小节点u并对它能到达的结点v进行松弛，如果v的最短路长度有更新，则将v加入到Q中，直到Q为空。 |

#### 2.1.4 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:  因为Euclidean distance是两点之间的最短距离，和实际距离相等，因此满足a smaller or equal distance than the actual shortest path distance from a vertex to a destination vertex 所以满足admissible的要求。  Consistent：  C:\Users\smx\Documents\Tencent Files\392558193\FileRecv\MobileFile\IMG_20180304_023540.jpg  对于任意一条边(u,v)来说，因为Euclidean distance是实际距离，由三角形的边的性质可以知道h(u) ≤ h(v)+w(e)(当且仅当共线时取等)，所以满足consistent的要求 |

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

|  |
| --- |
| 当h(v) = 0时，相当于只考虑当前点和初始点之间的距离，即为dijkstra算法。 |

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.)

|  |
| --- |
| C:\Users\smx\Documents\Tencent Files\392558193\FileRecv\MobileFile\IMG_20180305_003326_HHT.jpgA\*算法按照s->v->u->t的顺序进行访问，记录的从s到t的最短路为s->v->t的长度，非最优解。因为u和v的heuristic不满足consistent，所以在访问时本来应该先访问u再访问v，出现错误。 |

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.

|  |
| --- |
| 因为hC:\Users\smx\Documents\Tencent Files\392558193\FileRecv\MobileFile\1520130542956.jpg(u)太大，每次选择h+d最小的点，算法按照s->v->t->u的顺序进行访问，结束时记录的最短路长度为s->v->t的长度，不是s到t的最短路。因为h(u)太大，所以本来应该先访问的u在最后被访问。 |

#### 2.1.5 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 |
| * 算法简述： 将点(u,v,w)以(u,(v,w))的形式表示出来，用collect将每一个点作为key，再把value及邻居点和边权重转换为table。 |
| * 代码实现：   fun makeGraph (E : edge Seq.seq) : graph =  let  fun tolable (u,v,w) = (u,(v,w))  val nedge = Table.collect (Seq.map tolable E)  in  Table.map Table.fromSeq nedge  end |
| * 复杂度分析：   将图转换为table的形式复杂度为O(|E|),用Table.collect的操作合并每个点的邻居，work为O(|V|log|V|) ，span为O(log2|V|)。最后用map和fromSeq转换为table的work为O(|E|log|E|)，span为O(log2|E|)。又因为|E|≤|V|2，所以最后的结果满足O(|E|log|V|) work 及O(log2|V|) span |

#### 2.1.6 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.

|  |
| --- |
| * 算法描述：   按照dijkstra的思路，每次取出优先队列Q(即当前可以访问的点)中d+h最小的元素进行访问，如果这个点在target vertices中，以SOME (v,d)的形式记录距离，如果不准target vertices中且已经被访问过了那么对Q剩下的元素继续进行dijkstra，如果没有被访问，将该点加入到已访问的点X中，并且加入启发式函数h对Q的值进行更新。因为要完成多source vertices的最短路，因此假设加入一个节点，将它到source vertices进行dijkstra的结果得到的优先队列Q做为函数的初始值。 |
| * 代码实现：   fun findPath h G (S, T) =  let  fun N(v) =  case Table.find G v  of NONE => Table.empty ()  | SOME nbr => nbr  fun Dijkstra (X,Q) =  case PQ.deleteMin(Q) of  (NONE,\_) => NONE  |(SOME(d,v),Q') =>  if (Set.find T v) then SOME(v,d) else  case Table.find X v of  SOME \_ => Dijkstra (X,Q')  |NONE =>  let  val insert = Table.insert (fn (x,\_) => x)  val X' = insert (v, d) X  (\*change\*)  fun relax (Q,(u,w)) = PQ.insert (d+w-h(v)+h(u),u) Q  val Q'' = Table.iter relax Q' (N v)  in  Dijkstra (X',Q'')  end  (\*初始值\*)  val sources = PQ.fromList (Seq.toList (Seq.map (fn v => (0.0 + h(v),v) ) (Set.toSeq S)))  val result = Dijkstra ((Table.empty()), sources)  in  result  end |
| * 复杂度分析：   因为算法只对dijkstra在队列里记录的数据进行了改变，且初始值相当于一次dijkstra，因此它的复杂度与书上用table实现的dijkstra算法的work和span一样，都是O(|E|log|V|). |
| * 测试样例：      * 测试结果： |

## 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. 回答问题

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.

|  |
| --- |
| * 算法简述：   用到star contract的方法，每次对最小边的集合用随机抛硬币的方法进行选择，如果边(u，v)的两个点分别为H(0)，T(1)，那么缩边，更新剩下的点和最小边集。Booruvka算法的每一轮都进行缩边，记录每一次选择的边，直到最后没有边剩下，这些选择的边构成最小生成树。 |
| * 代码实现：   fun MST (E : edge seq, n : int) : edge seq =  let  fun cmp ((u1,v1,w1),(u2,v2,w2)) = Int.compare (w1,w2)  val sorte = rev (sort cmp E)(\*转换为从大到小\*)  val edge = map (fn (u,v,w) => (u,(v,(u,v,w)))) sorte (\*边作为lable\*)  val vertices = tabulate (fn i => i) n  fun joinerStarContract (V:vertex seq,E:(vertex \* (vertex \* edge)) seq,seed) =  let  val minEw = inject E (tabulate (fn \_ => (~1,(0,0,0))) n)  val minE = filter (fn (\_,(v,\_)) => v >= 0) (enum minEw) (\*得到最小边集\*)  val coins = Rand.flip seed n  fun head u = if (nth coins u) = 0 then true else false  fun contract (u,(v,e)) = if (head u) andalso (not (head v)) then true else false(\*star contract\*)  val P = filter contract minE  val Vcon = map (fn (u,(v,e)) => (u,v)) P  val V' = inject Vcon V(\*更新\*)  in  (V',P)  end  fun Boruvka ((V,E),T:edge seq,i) =  if length E = 0 then T  else  let  val (V',PT) = joinerStarContract (V,E,i)  val P = map (fn (u,(v,l)) => (nth V' u,(nth V' v,l))) E (\*star\*)  val T' = map (fn (u,(v,l)) => l) PT  val E' = filter (fn (u,(v,l)) => u <> v) P  val T'' = append (T,T')  val i' = Rand.next i  in  Boruvka ((V',E'),T'',i')  end  in  Boruvka ((vertices,edge),empty(),Rand.fromInt 1004)  end |
| * 复杂度分析：   书上有证明star contract每一轮移除的点数的期望至少为n/4，Star contract全部移除边到结束的work为O(mlogn + n),span为O(log2n)。在Boruvka中，因为map，append的work都是O(m)，span是O(1),filter的work也是O(m)，span是O(log m)的，在O意义下相对于star contract的work和span可以忽略，因此总的work为O(mlogn+n)，span为O(log2n)。 |
| * 测试样例：      * 测试结果： |

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：   利用Boruvka的算法对图像进行分割操作，每个点都有一个credit值。在每一次contract时，更新H点的credit为它的T和H点中最小的credit减去T-H边的权值。如果对于边(u,v)来说，min(cu, cv)-wuv < 0,会导致缩边时更新的credit<0，所以要删掉这条边。在实现时，前面选择的被删掉的边和Boruvka的star contract一样，后面进行由credit决定的constract时，先用map和collect找出到终点的所有边，再求出这些边的weight的和，再找到点和相邻点中最小的credit求差更新credit seq，如果新的u点或者v点的credit大于边(u,v)的权值，删掉这条边。当剩下的边为0时，结束Boruvka |
| * 代码实现：   fun findSegments (E, n) initial\_credit =  let  fun cmp ((u1,v1,w1),(u2,v2,w2)) = Int.compare (w1,w2)  val sorte = rev (sort cmp E)(\*转换为从大到小\*)  val edge = map (fn (u,v,w) => (u,(v,w))) sorte  val vertices = tabulate (fn i => i) n  val credits = tabulate (fn \_ => initial\_credit) n  fun joinerStarContract (V,E) seed =  let  val minEw = inject E (tabulate (fn \_ => (~1,~1)) n)  val minE = filter (fn (\_,(v,\_)) => v >= 0) (enum minEw)  val coins = Rand.flip seed n  fun contract (u,(v,e)) = if (nth coins u) = 0 andalso (nth coins v) = 1 then true else false  val P = filter contract minE  val Vcon = map (fn (u,(v,e)) => (u,v)) P  val V' = inject Vcon V  in  (V',P)  end  fun Boruvka ((V,E,c),seed) =  if length E = 0 then V  else  let  val (V',PT) = joinerStarContract (V,E) seed  val Vfinal = map (fn v => nth V' v) V'  (\*credit inject then filter\*)  val to\_collect = map (fn (u,(v,w)) => (v,(u,w))) PT  val ngcontract = collect Int.compare to\_collect  fun sum (v:vertex,s:(vertex \* weight) seq) = (v,reduce op+ 0 (map #2 s))  val nsum = map sum ngcontract(\*the sum of the weights of the contracted edges\*)  fun cre (v,s:(vertex \* weight) seq) = reduce Int.min initial\_credit (map (fn (v,w) => nth c v) s)  val minc = map (fn (v,s) => (v,Int.min(nth c v, cre(v,s)))) ngcontract(\*找到endpoints里的最小credit\*)  val nc = inject minc c  val final = map (fn (v,w) => (v,((nth nc v) - w))) nsum  val finalc = inject final nc  val P = map (fn (u,(v,w)) => (nth Vfinal u,(nth Vfinal v,w))) E  fun tofil (u,(v,w)) = (u <> v) andalso (w < (nth finalc u) andalso (w < (nth finalc v)))  val E' = filter tofil P  val seed' = Rand.next seed  in  Boruvka((Vfinal,E',finalc),seed')  end  in  Boruvka((vertices,edge,credits),Rand.fromInt 528)  end |
| * 复杂度分析：   前面的star constract和前面Boruvka一样，全部的work为O(mlogn + n),span为O(log2n)。  后面对credit进行更新的时候，找出到终点的所有边用到的collect的work为O(mlogn),span为O(log2m) 求credit的新的值时用到map reduce的work为O(mlogn)，span为O(logn),更新credit用到inject的work为O(m),span为O(1),选择剩下的边用到filter的work为O(m)，span为O(logn)。对于Boruvka算法来说，因为采用star constract，所以O(logn)轮之内可以结束Boruvka，因此最终的work为O(mlog2 n) ，span为 O(log3n) ，满足题目要求。 |
| * 测试样例：   G:\大二\串并行 新\7-segmentlab-empty-windows\skittles.png   * 测试结果：     G:\大二\串并行 新\7-segmentlab-empty-windows\out.pngG:\大二\串并行 新\7-segmentlab-empty-windows\outpue.png  credit = 500 credit = 1000  G:\大二\串并行 新\7-segmentlab-empty-windows\output5000.pngG:\大二\串并行 新\7-segmentlab-empty-windows\output10000.png  credit = 5000 credit = 10000 |

## 3.书面习题

3.1

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.

|  |
| --- |
| 如果|E| = 2，对于一个简单连通无向图来说，有三个点，所以生成的MST包含最小权值的边和第二小权值的边。  如果|E| > 2，假设最小生成树T中的边为e1,e2…en.假设图中第二小的边e不在最小生成树中。将e加到最小生成树中，那么一定会产生一个含有e的环，因为e是整个图中第二小的边，而一个环至少有三条边，所以一定存在一条边ei比e长，删掉ei之后产生的树的权值比T小，因而T不是最小生成树，产生矛盾。因此第二小权值的边一定在MST中。 |

3.2

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.

|  |
| --- |
| 设一个点的取值为随机变量X，则它旁边的两个点都比它小的概率也分别为X，又因为它们独立，所以P = X2，所以期望为.所以expected number 为 |

#### 3.3

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.

|  |
| --- |
| 由Markov’s Inequality，  所以 (1)  对两边取期望得， (2)  因为f(n)是单调非降的，所以，当时，; 当时， (3)  将(1)(3)代入(2)，得  化简得，  所以由tree method解得，得证。 |

## 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 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：   如果T是空树(order table)，返回NONE，如果T不是空树(order table)，则将根结点表示为{L,key,value,R}的形式，其中L是左子树，R是右子树。由BST的性质知，最小值在L中，最大值在R中。所以first函数递归的搜索L直到L的大小为0后返回SOME(key,value)，last函数递归的搜索R直到R的大小为0后返回SOME(key,value) |
| * 代码实现： |
| * 测试样例：   val ordSet1 = % [5, 7, 2, 8, 9, 1]  val ordSet2 = % [7, 1, 9, 2, 5, 6]  val testsFirst = [  ordSet1,  ordSet2,  % []  ]  val testsLast = [  ordSet1,  ordSet2,  % []  ]   * 测试结果： |

2.2 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：   用spilt分割BST树，得到(L,\_,R)，由BST的性质，查找L中最大的数即为previous的结果，R中最小的数即为next的结果 |
| * 代码实现： |
| * 测试样例：   val ordSet1 = % [5, 7, 2, 8, 9, 1]  val ordSet2 = % [7, 1, 9, 2, 5, 6]  val testsPrev = [  (ordSet1, 8),  (ordSet1, 1),  (ordSet2, 5),  (% [], 8)  ]  val testsNext = [  (ordSet1, 8),  (ordSet1, 9),  (ordSet2, 5),  (% [], 8)  ]   * 测试结果： |

2.3 完成下列函数，做必要说明。

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.

|  |
| --- |
| * 算法思路：   和BST树的join一样，直接调用库函数 |
| * 代码实现： |
| * 测试样例：   val ordSet1 = % [5, 7, 2, 8, 9, 1]  val ordSet2 = % [7, 1, 9, 2, 5, 6]  val testsJoin = [  (ordSet1, % [100]),  (ordSet1, % [3]),  (ordSet2, % [3]),  (% [], % [100])  ]   * 测试结果： |

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.

|  |
| --- |
| * 算法思路：   和BST树的split一样，直接调用库函数 |
| * 代码实现： |
| * 测试样例：   val ordSet1 = % [5, 7, 2, 8, 9, 1]  val ordSet2 = % [7, 1, 9, 2, 5, 6]  val testsSplit = [  (ordSet1, 7),  (ordSet1, 100),  (ordSet2, 9),  (% [], 7)  ]   * 测试结果： |

2.4 完成函数getRange并详述思路

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：   用split分割BST树，取比low大的子树R1，如果low在树里面合并low节点和R1形成新树，对新树进行同样的操作得到low和high范围内且包含low和high的树。 |
| * 代码实现：   fun getRange (T : 'a table) (low : key, high : key) : 'a table =  let  val (L1,x,R1) = Tree.splitAt(T,low)  val T1 = case x of  NONE => R1  |SOME v => Tree.join(singleton(low,v),R1)  val (L2,y,R2) = Tree.splitAt(T1,high)  val T2 = case y of  NONE => L2  |SOME v => Tree.join(L2,singleton(high,v))  in  T2  end |
| * 测试样例：   val ordSet1 = % [5, 7, 2, 8, 9, 1]  val ordSet2 = % [7, 1, 9, 2, 5, 6]  val testsRange = [  (ordSet1, (5,8)),  (ordSet1, (10,12)),  (ordSet2, (2,6)),  (% [], (5,8))  ]   * 测试结果： |

2.5 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 toyou. You should choose the type of countTable such that you can implement count (range queries) in O(logn) work and span. For full credit, your makeCountTable must run within O(nlogn) expected work.

|  |
| --- |
| * 算法思路：   按照x的大小顺序将point seq排列，用Seq.iterh和OrdTable.insert函数然后利用append得到每一个x到0 之间的点(y,point)的point table(类似于scanI，保存中间结果)，再记录x，最终得到一个(x,(y,points))的point table table做为count table |
| * 代码实现：   fun makeCountTable (S : point seq) : countTable =  if Seq.length S = 0 then empty()  else  let  val sorted = Seq.sort (fn (x,y) => compareKey (#1 x,#1 y)) S(\*按横坐标排序\*)  val totablev = Seq.map (fn (x,y) => (y,(x,y)) ) sorted  val totablek = Seq.map (fn (x,y) => x) sorted  fun insertion (a,b) = insert (fn (x,y) => y) b a (\*a sweep line 0~x内的点\*)  val res = Seq.iterh insertion (empty()) totablev(\*类似scanI\*)  val resultv = Seq.append(Seq.drop (#1 res,1),Seq.singleton (#2 res))  val kv = Seq.zip totablek resultv  in  fromSeq kv  end |
| * 复杂度分析：   函数主要是sort，map，iterh insert，append，zip和fromSeq几个函数，其中map，append和zip的work都是O(n)，span都是O(1),work主要是sort和iterh insert两个部分，因为insert的work为O(log n)所以iterh insert的work和span都为O(nlogn)（串行），sort的work是O(nlogn)，所以整体的work为O(nlogn)，span为O(nlogn) |

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.

|  |
| --- |
| iterh是串行的，可以用scan union代替iterh和append这两步得到)的span，这一步的work为 ，相比于原来work变大了。 |

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.

|  |
| --- |
| 空间复杂度为O(n2)，在countTable中，每个点的value最多n个，因此空间复杂度为O(n2). |

2.6 完成函数count并做相关分析

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.

|  |
| --- |
| * 算法思路：   用getRang函数得到T在矩形的x\_1到x\_2范围内的部分xrange。0到x\_1范围内的点为xrange的最小值(x\_1是最小的)，可以用first函数求，再用getRange函数得到y\_1和y\_2范围内的点，点的数目point\_l即为剩下的table大小。同理可以得到0到x\_2内的点的数目point\_r(改用last函数求)则(x\_1,y\_1) (x\_2, y\_2)构成的矩形中点的数目为point\_r - point\_l +1或者如果xrange大小为0则里面点的数目为0。 |
| * 代码实现：   fun count (T : countTable)  ((xLeft, yHi) : point, (xRght, yLo) : point) : int =  let  val xrange = getRange T (xLeft,xRght)  val point\_r =  case last xrange of  NONE => 0  |SOME (x,y) => size (getRange y (yLo,yHi))  val point\_l =  case first xrange of  NONE => 0  |SOME (x,y) => size (getRange y (yLo,yHi))  val num = point\_r - point\_l + 1  in  if (size xrange = 0) then 0 else num  end |
| * 复杂度分析：   函数主要由getRange，last，first构成（size以及加减的work和span为O(1)，忽略）它们的work和span都是O(log n)的，所以总的work和span都是O(log n) |
| * 测试样例：   val points1 = % [(0,0),(1,2),(3,3),(4,4),(5,1)]  val points2 : point seq = % []  val points3 = % [(10000,10000),(0,0)]  val points4 = tabulate (fn i => (i,i)) 1000  val testsCount = [  (points1, ((1,3),(5,1))),  (points1, ((2,4),(4,2))),  (points1, ((100,101),(101,100))),  (points2, ((0,10),(10,0))),  (points3, ((0,10000),(10000,0))),  (points4, ((0,500),(1000,0)))  ]   * 测试结果：   C:\Users\smx\AppData\Roaming\Tencent\Users\392558193\TIM\WinTemp\RichOle\{6@{)39_DS%S6}9VOL6BH6C.png |