

华中科技大学

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 scanl 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 $0 \leq r \leq 1$, choose hist r should evaluate to the value at r from the cumulative distribution that corresponds to the histogram hist. If r is not in the range [0,1] or hist is empty, you should raise an exception. For full credit, choose should have $O(|\text{hist}|)$ work and $O(\log|\text{hist}|)$ 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 = scanl op+ 0 s1 (*得到概率密度*)
      val a = nth s2 (length s2 - 1) (*次数和*)
    in
      ...
    end
```

```

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

```

```

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 = scanl 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
  end

```

- 复杂度分析：设 n 为 hist 的长度

Work: 函数主要的 work 是 map scanl 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：

```

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),
  ([], 0.2), (*空的情况*)
  ([("test", 10)], 1.5), (*p大于1*)
  ([("test", 10)], ~1.5), (*p为负数*)
  ([("yup", 4), ("test", 10)], ~0.1),
  ([("yay", 1), ("woah", 2), ("oh", 3), ("yup", 4), ("hello", 3), ("bye", 4)], 0.36)(*longer more complex*)
]

```

● 测试结果:

```

- Tester.testChoose ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test failed on input (<("test",10)>,<~1.5); got Exn Range; expected "test".
Test failed on input (<("yup",4),("test",10)>,<~0.1); got Exn Range; expected "yup".
Test passed.

```

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 $T : \text{TABLE}$ where type $\text{Key.t} = \text{string Util.Seq.seq}$. 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(described above) in the functor MkTableKGramStats in MkTableKGramStats.sml. For full credit, makeStats corpus maxK should have $O(n \log n)$ work and $O(\log^2 n)$ 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
```

```
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

fun lookupExts (stats : kgramstats) (kgram : kgram) : (token * int) seq =
  case Table.find (#2 stats) kgram of
    SOME t => t
  | NONE => empty()

fun maxK (stats : kgramstats) : int =
  #1 stats
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 O(log² n)。因此总的 work 为 O(n),span 为 O(log² n)

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 $O(\log n)$ work and span, where n is 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()

```
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

```
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：

```

* corporuses will not be submitted. *)
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",
      "",
      "notaword"
    ])
]

```

- 测试结果:

```

- Tester.testKGramStats ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
-

```

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` should have $O(n(W_{\text{lookupExts}} + W_{\text{choose}}))$ work and $O(n(S_{\text{lookupExts}} + S_{\text{choose}}))$ 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.

- 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

```

```

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
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 $O(\log m)$, 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 $W_{\text{lookupExts}} + W_{\text{choose}}$, span 为 $S_{\text{lookupExts}} + S_{\text{choose}}$. iter 是完全串行的, 因此它的 work 为 $O(n(W_{\text{lookupExts}} + W_{\text{choose}}))$, span 为 $O(n(S_{\text{lookupExts}} + S_{\text{choose}}))$

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 $O(nW_{\text{randomSentence}})$ work and $O(n + S_{\text{randomSentence}})$ 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
```

```

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 $O(n \times \text{randomSentence})$, 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 $O(n + \text{SrandomSentence})$
- 测试结果:

```

- Tester.testBabble "data/shakespeare.txt";
her train oberon well go thy way. is thisby a wandering knight. met together t-
o rehearse a. on some true love s sight of thy. separation as may well be said-
becomes a. the lion s part and i hope here. to her close and consecrated bowe-
r. yield you up my part and. will do it in action as. thou bring her here i ll-
charm his eyes.

```

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 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)

```

fun extrema (s:int seq) : int option seq =
  let
    fun compare (a,b) = if a > b then true else false
    fun cmp compare (i,j) =
      (j < 0) orelse (j > length s - 1) (* first/last true *)
      orelse compare (nth s i, nth s j)
    fun choose i =
      if cmp compare (i,i-1) then
        if not (cmp compare (i,i+1) then SOME i else NONE
        else if cmp compare (i,i+1) then SOME i else NONE
  in
    tabulate choose [length s]
  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: 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 $O(\log^2|E|)$ 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：

```
fun makeGraph (E : edge seq) : graph =  
  let  
    val gra = Table.collect E  
    val m = length E  
    val undir = map (fn (a,b) => fromList [(a,b),(b,a)]) E  
    val n = size (Table.collect (flatten undir))  
    (*注意只入不出的点*)  
  in  
    (gra,m,n)  
  end
```

- 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 $O(\log^2|E|)$ 。因此总的 work 为 $O(|E|\log|E|)$, span 为 $O(\log^2|E|)$

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` contains all w s.t. $(v,w) \in E$. 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`

2.1.5 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 $|Vout| \leq |V|$, 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

Task 4.5 (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 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 $O(D\log^2|V|)$ 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
```

```

end

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
  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
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。

| | Work | Span |
|-----------------|-------------------|---------------|
| $X \cup F$ | $O(F \log n)$ | $O(\log n)$ |
| $N_G^+(F)$ | $O(\ F\ \log n)$ | $O(\log^2 n)$ |
| $N \setminus X$ | $O(\ F\ \log n)$ | $O(\log n)$. |

在每一轮中, 每条边和每个点的 cost 都为 $O(\log n)$, 所有的点和边在整个 BFS 中都只出现一次, 所以总的 work 为 $O(m \log n + n \log n) = O(m \log n)$, 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 $O(D \log^2 n)$, 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  
  end
```

```
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  
  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 to you're 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 testfile = "input/thesaurus.txt"
val testfile2 = "input/simpletest.txt"

(* The following are required *)
val testsNum = [edgeseq, edgeseq2, test3];

val testsOutNeighbors =
  [(edgeseq, 1),
   (edgeseq, 2),
   (test3, 1),
   (edgeseq2, 5),
   (edgeseq2, 7),
   (test3, 9)]

val testsReport =
  [((edgeseq, 1), 2),
   ((edgeseq2, 1), 4),
   ((edgeseq2, 1), 7),
   ((test3, 4), 2),
   ((test3, 1), 3),
   ((test3, 6), 2),
   ((test3, 1), 6)]

val testsNumWords = [testfile, testfile2]
```

- 测试结果:

```

- Tester.testReport ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
- Tester.testNumEdges ();
Test passed.
Test passed.
Test passed.
val it = () : unit
- Tester.testNumVertices ();
Test passed.
Test passed.
Test passed.
val it = () : unit
- Tester.testOutNeighbors ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
- Tester.testReport ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit

```

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

```

end

```
Task 5.2 (6%)  
fun make (S : (string * string) 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:

```

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"))]]

```

● 测试结果:

```

- Tester.testNumWords ();
Test passed.
Test passed.
val it = () : unit
- Tester.testSynonyms ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
- Tester.testQuery ();
HANDSOME -> VINCENT -> YOLO
GOOD -> EXCELLENT -> SUPERB -> IMPRESSIVE -> AWESOME -> TERRIBLE -> WORST -> BAD
GOOD -> WORTH -> MERIT -> EARN -> WIN -> CONQUER -> WORST -> BAD
CLEAR -> EVACUATE -> EMPTY -> EXHAUST -> WEAKEN -> FAINT -> INAUDIBLE -> INDISTINCT -> UNCLEAR -> VAGUE
LOGICAL -> VALID -> ACCURATE -> EXQUISITE -> DAINITY -> DELICATE -> UNHEALTHY -> UNSOUND -> IMPRACTICAL -> ILLOGICAL
HAPPY -> GAY -> LIVELY -> VOLATIBLE -> CHANGEABLE -> VARIABLE -> MOODY -> SAD
LIBERAL -> LAVISH -> SQUANDER -> WASTE -> EBB -> WITHDRAW -> RETREAT -> SHELTER -> PROTECTION -> CONSERVATION
LIBERAL -> LAVISH -> SQUANDER -> WASTE -> LEAK -> ESCAPE -> RETREAT -> SHELTER -> PROTECTION -> CONSERVATION
LIBERAL -> LAVISH -> SQUANDER -> DISSIPATE -> SCATTER -> SPREAD -> COVER -> SHELTER -> PROTECTION -> CONSERVATION
LIBERAL -> LAVISH -> SQUANDER -> DISSIPATE -> DIFFUSE -> SPREAD -> COVER -> SHELTER -> PROTECTION -> CONSERVATION
LIBERAL -> LAVISH -> SQUANDER -> WASTE -> LEAK -> OOZE -> FILTER -> SCREEN -> PROTECTION -> CONSERVATION
LIBERAL -> LAVISH -> SQUANDER -> DISSIPATE -> SCATTER -> SPREAD -> COVER -> GUARD -> PROTECTION -> CONSERVATION
LIBERAL -> LAVISH -> SQUANDER -> DISSIPATE -> DIFFUSE -> SPREAD -> COVER -> GUARD -> PROTECTION -> CONSERVATION
EARTHLY -> SENSUAL -> EROTIC -> OBSCENE -> FOUL -> POLLUTE -> TAINT -> 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 $O(\log^2|V|)$ 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
```

```
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 $O(\log^2|E|)$ 。又因为 $|E|\leq|V|^2$, 所以符合 $O(|E|\log|V|)$ work and $O(\log^2|V|)$ 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
      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
```

```

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) (*discover*) (*如果v的邻居里有u 去掉u*)
        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
    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:

```

val testBridge = List.map ArraySequence.% [
  [(0,1),(1,2),(0,2),(2,3)],
  [(0,1),(1,2),(2,0),(2,3)],
  [(3,1),(4,2),(0,1),(1,2),(2,6),(6,0),(3,4),(4,5),(6,8),(8,7),(8,9),(9,10),(10,11),(9,11)],
  []
]

```

- 测试结果:

```

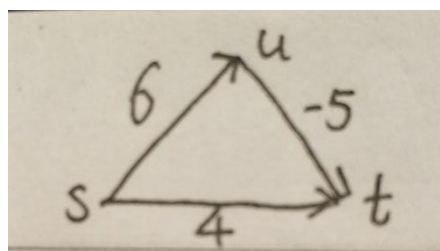
- Tester.testBridges ();
Test passed.
Test passed.
Test passed.
Test passed.

*** Retest with another implementation ***
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
-

```

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.



$(s,0) \rightarrow (t,4) \rightarrow (u,6)$ 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 $s \rightarrow u \rightarrow t$ 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。

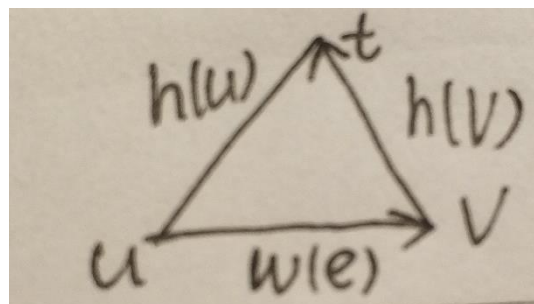
2. 1. 4 回答关于启发式函数的一些问题

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:

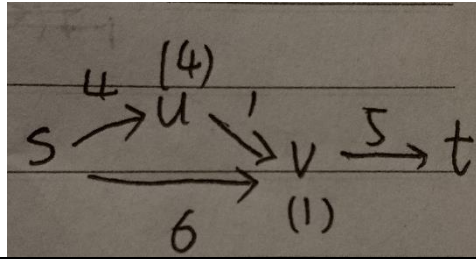


对于任意一条边 (u,v) 来说，因为 Euclidean distance 是实际距离，由三角形的边的性质可以知道 $h(u) \leq h(v) + w(e)$ (当且仅当共线时取等)，所以满足 consistent 的要求

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

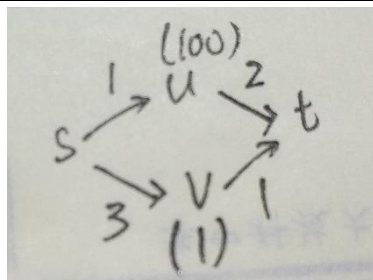
当 $h(v) = 0$ 时，相当于只考虑当前点和初始点之间的距离，即为 dijkstra 算法。

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.)



A*算法按照 $s \rightarrow v \rightarrow u \rightarrow t$ 的顺序进行访问，记录的从 s 到 t 的最短路为 $s \rightarrow v \rightarrow t$ 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 $h(u)$ 太大，每次选择 $h+d$ 最小的点，算法按照 $s \rightarrow v \rightarrow t \rightarrow u$ 的顺序进行访问，结束时记录的最短路长度为 $s \rightarrow v \rightarrow t$ 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 $O(\log^2|V|)$ 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 totable (u,v,w) = (u,(v,w))
    val nedge = Table.collect (Seq.map totable 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 $O(\log^2|V|)$ 。最后用 map 和 fromSeq 转换为 table 的 work 为 $O(|E|\log|E|)$ ，span 为 $O(\log^2|E|)$ 。又因为 $|E| \leq |V|^2$ ，所以最后的结果满足 $O(|E|\log|V|)$ work 及 $O(\log^2|V|)$ 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, $S \subseteq V$.
3. Multiple target vertices, $T \subseteq V$. If multiple sources and destinations are given, your algorithm should return the shortest path distance between any $s \in S$ and any $t \in T$ (a shortest S-T path). Specifically, $\text{findPath } h \ G \ (S,T)$ should evaluate to SOME (v,d) if the shortest S-T path in G ends at vertex $v \in T$ 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，如果不在 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

```

```

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
  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:

```

val heauris = Real.fromInt o (fn x => 4*(x div 10))
fun doubleSide edges = ArraySequence.append((ArraySequence.map (fn (s,d,w)=>(d,s,w)) edges),edges)
(*val k = doubleSide (ArraySequence.% [(1,2,0.2)])*)
val testAStar = List.map (fn (edges, (S, T),h) => (doubleSide(ArraySequence.% edges), (ArraySequence.% S, ArraySequence.% T),h),
[
  ([[(1,2,0.2)],[(1],[2]),fn 1=>0.1 | 2=>0.0 | _=> 10000.0),
  ([[(1,2,0.2)],[(1],[3]),fn 1=>0.1 | 2=>0.0 | _=> 10000.0),
  ([[(1,2,0.2)],[(0],[2]),fn 1=>0.1 | 2=>0.0 | _=> 10000.0),
  ( [
    ([
      (43,44,3.0),
      (44,31,4.0),(43,31,5.0),(43,30,5.0),
      (30,31,6.0),
      (31,20,5.0),(30,20,5.0),
      (20,12,5.0),(20,11,5.0),
      (12,01,4.0),(12,02,5.0),(11,02,5.0)
    ]),([20],[1,2]),heauris),
    ( [
      (43,44,3.0),
      (44,31,4.0),(43,31,5.0),(43,30,5.0),
      (30,31,6.0),
      (31,20,5.0),(30,20,5.0),
      (20,12,5.0),(20,11,5.0),
      (12,01,4.0),(12,02,5.0),(11,02,5.0)
    ]),([43,44],[01,02]),heauris),
    ( [
      (43,44,3.0),
      (44,31,4.0),(43,31,5.0),(43,30,5.0),
      (30,31,106.0),
      (31,20,105.0),(30,20,5.0),
      (20,12,5.0),(20,11,5.0),
      (12,01,4.0),(12,02,5.0),(11,02,5.0)
    ]),([31],[01,02]),(fn x=>1000.0*x) o heauris)
  ]
]

```

- 测试结果:

```

- Tester.testAStar ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.

*** Retest with another implementation ***
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
-

```

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 `MkKruskalMST.sml` where `MST (E, n)` computes the minimum spanning tree of the graph represented by the input edge sequence `E` using Borůvka'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 $O(m \log n + n)$ work and expected $O(\log^k n)$ 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
    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
        end
    in
        Boruvka ((vertices,edge),empty(),Rand.fromInt 1004)
    end
end

```

```

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
        end
    in
        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
            end
        in
            Boruvka ((vertices,edge),empty(),Rand.fromInt 1004)
        end
    end
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 $O(m \log n + n)$, span 为 $O(\log^2 n)$ 。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 $O(m \log n + n)$, span 为 $O(\log^2 n)$ 。

- 测试样例：

```
val testsMST =
[
  ([ (0,2,10), (1,0,5)], 3),
  ([ (0,1,10), (0,2,2), (0,3,5), (1,4,10), (2,1,10), (2,5,2), (4,5,2)], 6),
  ([ (0,1,3), (0,2,6), (1,3,2), (1,4,9),
    (2,3,2), (2,6,9), (3,5,8), (4,5,8),
    (4,9,18), (5,6,7), (5,8,9), (6,7,4),
    (6,8,5), (7,8,1), (7,9,4), (8,9,3)], 10),
  ([ (1,2,101), (1,3,100), (2,3,2), (2,4,2), (3,4,1), (0,1,0)], 5),
  ([ (1,2,50), (1,3,80), (2,3,60), (2,4,20), (3,5,40),
    (2,5,30), (4,5,10), (4,6,10), (5,6,50), (0,4,0)], 7),
  ([ (6,0,166), (8,2,309), (11,3,570), (1,4,158), (8,5,816), (8,6,660),
    (11,7,7), (4,8,635), (11,9,292), (5,10,242), (14,11,34), (9,12,697),
    (6,13,544), (1,14,356), (8,10,353), (5,11,717), (11,10,851), (5,1,290),
    (13,14,908), (5,12,341), (2,0,716), (6,11,500), (6,14,698), (13,12,629),
    (13,10,335)], 15),
  ([ (2,1,91), (3,2,63), (0,3,57), (3,4,83), (4,0,57),
    (3,1,56), (4,1,10), (4,2,88), (1,0,17), (2,0,77)], 5),
  ([ (0,1,1000), (1,2,1000), (2,3,1000), (1,4,1000), (4,5,1000)], 6),
  ([ (6,0,1), (6,1,3), (7,2,3), (6,3,10), (7,5,7),
    (7,6,5), (4,7,1), (2,8,0), (6,9,1), (3,0,0),
    (5,6,3), (8,6,7), (8,9,3), (1,9,4), (3,1,7),
    (0,2,8), (5,8,10), (2,3,4), (0,8,7), (6,4,8),
    (9,0,4), (0,7,6), (8,4,0), (5,0,6), (3,4,7),
    (9,4,0), (9,5,7), (7,3,7), (0,4,0), (2,1,5)], 10)
]
```

- 测试结果：

```
- Tester.testMST ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
*** We will now use Mr. Gou's solution to test yours again. ***
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
-
```

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 $O(m \log^2 n)$ work and expected $O(\log^k n)$ 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, $\min(c_u, c_v) - w_{uv} < 0$, 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 contract 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'
        end
      end
  end
```



```

(*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

```

```

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
  end

```

- 复杂度分析:

前面的 star contract 和前面 Boruvka 一样,全部的 work 为 $O(m\log n + n)$,span 为 $O(\log^2 n)$ 。


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 $O(m\log n)$,span 为 $O(\log^2 m)$ 求 credit 的新的值时用到 map reduce 的 work 为 $O(m\log n)$,span 为 $O(\log n)$,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 $O(\log n)$ 。对于 Boruvka 算法来说,因为采用 star contract,所以 $O(\log n)$ 轮之内可以结束 Boruvka,因此最终的 work 为 $O(m\log^2 n)$, span 为 $O(\log^3 n)$, 满足题目要求。

- 测试样例:




- 测试结果:


| | | | |
|-------------|-----------------|--------|-------|
| out | 2018/2/28 19:08 | PNG 文件 | 35 KB |
| outpue | 2018/3/5 17:17 | PNG 文件 | 49 KB |
| output5000 | 2018/3/5 17:39 | PNG 文件 | 13 KB |
| output10000 | 2018/3/5 17:18 | PNG 文件 | 6 KB |
| output50000 | 2018/3/5 17:37 | PNG 文件 | 1 KB |



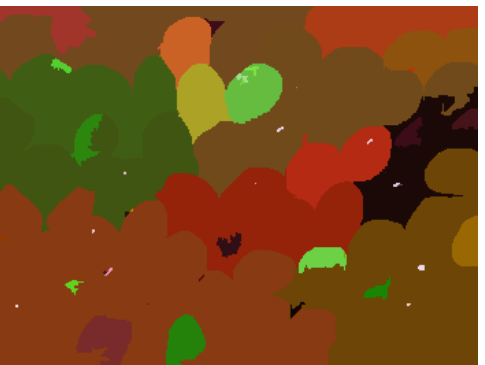
credit = 500



credit = 1000



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 $|E| \geq 2$ 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 e_1, e_2, \dots, e_n 。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 e_i 比 e 长，删掉 e_i 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 $n \geq 3$ 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 $r \in \mathbb{R} [0,1]$ (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 $P = X^2$, 所以期望为 $E(X^2) = \int_0^1 x^2 * f(x) dx = \int_0^1 x^2 dx = \frac{1}{3}$. 所以 expected number 为 $\frac{n}{3}$

3. 3

Task 6.3 (15%). It's Probably Linear. For all $n > 1$, let X_n be a random variable with $X_n \leq n$. Let f be a non-decreasing function satisfying $f(n) \leq f(X_n) + \theta(n)$, where $f(1)=1$. Prove that if $E[X_n] = n/3$, then $\bar{f}(n) = E[f(n)] \in O(n)$.

Hint: what is $\Pr[X_n \geq 2n/3]$? Use Markov's Inequality, covered in recitation 9.

由 Markov's Inequality, $\Pr\left[X_n \geq \frac{2n}{3}\right] \leq \frac{\frac{n}{3}}{\frac{2n}{3}} = \frac{1}{2}$

所以 $E[f(n)] = \Pr\left[X_n \geq \frac{2n}{3}\right] * E[f(n)] + \Pr\left[X_n < \frac{2n}{3}\right] * E[f(n)]$ (1)

对 $f(n) \leq f(X_n) + \theta(n)$ 两边取期望得, $E[f(n)] \leq E[f(X_n)] + \theta(n)$ (2)

因为 $f(n)$ 是单调非降的, 所以, 当 $X_n \geq \frac{2n}{3}$ 时, $E[f(X_n)] \leq E[f(n)]$; 当 $X_n < \frac{2n}{3}$ 时, $E[f(X_n)] \leq E\left[f\left(\frac{2n}{3}\right)\right]$ (3)

将(1)(3)代入(2), 得 $E[f(n)] \leq E[f(n)] * \frac{1}{2} + E\left[f\left(\frac{2n}{3}\right)\right] * \frac{1}{2} + \theta(n)$

化简得, $E[f(n)] \leq E\left[f\left(\frac{2n}{3}\right)\right] + \theta(n)$

所以由 tree method 解得 $E[f(n)] \in O(n)$, 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 $(k,v) \in T$ and k is the minimum key in T. Analogously, last T should evaluate to `SOME (k,v)` iff $(k,v) \in T$ 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 $\{L, \text{key}, \text{value}, R\}$ 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：

```
fun first (T : 'a table) : (key * 'a) option =  
  case Tree.expose T of  
  | NONE => NONE  
  | SOME {left,key,value,right} => if size left = 0 then SOME(key,value) else first left  
  
fun last (T : 'a table) : (key * 'a) option =  
  case Tree.expose T of  
  | NONE => NONE  
  | SOME {left,key,value,right} => if size right = 0 then SOME(key,value) else last right
```

- 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,
```

| |
|---|
| <pre> ordSet2, % []] </pre> |
| <ul style="list-style-type: none"> 测试结果: <pre> - Tester.testFirst (); Test passed. Test passed. Test passed. val it = () : unit - Tester.testLast (); Test passed. Test passed. Test passed. val it = () : unit </pre> |

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 $(k_0,v) \in T$ 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`.

| |
|--|
| <ul style="list-style-type: none"> 算法思路: 用 <code>split</code> 分割 BST 树, 得到 <code>(L,_,R)</code>, 由 BST 的性质, 查找 <code>L</code> 中最大的数即为 <code>previous</code> 的结果, <code>R</code> 中最小的数即为 <code>next</code> 的结果 |
| <ul style="list-style-type: none"> 代码实现: <pre> fun previous (T : 'a table) (k : key) : (key * 'a) option = let val (L,_,_) = Tree.splitAt(T,k) in last L end fun next (T : 'a table) (k : key) : (key * 'a) option = let val (_,_,R) = Tree.splitAt(T,k) in first R end </pre> |
| <ul style="list-style-type: none"> 测试样例: <pre> 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), </pre> |

```

(ordSet1, 9),
(ordSet2, 5),
(% [], 8)
]

```

- 测试结果:

```

- Tester.testPrev ();
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
- Tester.testNext ();
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit

```

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:

```

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

```
- 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:

```

- Tester.testJoin ();
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit

```

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 $(k',v) \in T$ such that $k' < k$,
2. SOME v if $(k,v) \in T$ and NONE otherwise, and
3. an ordered table containing every $(k',v) \in T$ 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:

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

- 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:

```
- Tester.testSplit ();
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
```

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 $(k,v) \in T$ such that $l \leq k \leq h$.

- 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
end

```

```

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
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:

```

- Tester.testRange ();
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit

```

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 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 $O(n \log n)$ 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
end

```

```

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内点的个数*)
      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
  end
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 $O(\log^2 n)$ 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 $O(\log^2 n)$ 的 span，这一步的 work 为 $\sum n \log \frac{n}{i} = O(n \log^2 n)$ ，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 $O(n^2)$ ，在 countTable 中，每个点的 value 最多 n 个，因此空间复杂度为 $O(n^2)$ 。

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

As described earlier, count T ((x₁, y₁), (x₂, y₂)) will report the number of points within the 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(log n) work and span.

- 算法思路:

用 getRange 函数得到 T 在矩形的 x₁ 到 x₂ 范围内的部分 xrange。0 到 x₁ 范围内的点为 xrange 的最小值(x₁ 是最小的), 可以用 first 函数求, 再用 getRange 函数得到 y₁ 和 y₂ 范围内的点, 点的数目 point₁ 即为剩下的 table 大小。同理可以得到 0 到 x₂ 内的点的数目 point_r(改用 last 函数求)则(x₁, y₁) (x₂, y₂)构成的矩形中点的数目为 point_r - point₁ + 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, (xRight, yLo) : point) : int =
    let
        val xrange = getRange T (xLeft, xRight)
        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
```

```
fun count (T : countTable)
    ((xLeft, yHi) : point, (xRight, yLo) : point) : int =
    let
        val xrange = getRange T (xLeft, xRight)
        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:

```
- Tester.testCount ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
-
```