```
import shell
import util
import wordsegUtil

############################################################
# Problem 1b: Solve the segmentation problem under a unigram model

class SegmentationProblem(util.SearchProblem):
    def __init__(self, query, unigramCost):
        self.query = query
        self.unigramCost = unigramCost

    def startState(self):
        # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if
you deviate from this)
        return self.query
        # END_YOUR_CODE

    def isEnd(self, state):
        # BEGIN_YOUR_CODE (our solution is 2 lines of code, but don't worry if
you deviate from this)
        return len(state) == 0
        # END_YOUR_CODE

    def succAndCost(self, state):
        # BEGIN_YOUR_CODE (our solution is 7 lines of code, but don't worry if
you deviate from this)
        result = []
        if not self.isEnd(state):
            for i in range(len(state), 0, -1):
                action = state[:i]
                result.append((action, state[len(action):],
self.unigramCost(action)))
        return result
        # END_YOUR_CODE

def segmentWords(query, unigramCost):
    if len(query) == 0:
        return ''

    ucs = util.UniformCostSearch(verbose = 0)
    ucs.solve(SegmentationProblem(query, unigramCost))

    # BEGIN_YOUR_CODE (our solution is 3 lines of code, but don't worry if you
deviate from this)
    if len(ucs.actions) == 0:
        return ''
    return ' '.join(ucs.actions)
    # END_YOUR_CODE

############################################################
# Problem 2b: Solve the vowel insertion problem under a bigram cost

class VowelInsertionProblem(util.SearchProblem):
    def __init__(self, queryWords, bigramCost, possibleFills):
        self.queryWords = queryWords
        self.bigramCost = bigramCost
        self.possibleFills = possibleFills

    def startState(self):
        # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if
you deviate from this)
        return (self.queryWords[0], 0)
        # END_YOUR_CODE
```

```
    def isEnd(self, state):
        # BEGIN_YOUR_CODE (our solution is 2 lines of code, but don't worry if
you deviate from this)
        return state[1] == len(self.queryWords) - 1
        # END_YOUR_CODE

    def succAndCost(self, state):
        # BEGIN_YOUR_CODE (our solution is 8 lines of code, but don't worry if
you deviate from this)
        result = []
        index = state[1] + 1
        word = self.queryWords[index]
        choices = self.possibleFills(word).copy()
        if len(choices) == 0:
            choices.add(word)
        for action in choices:
            cost = self.bigramCost(state[0], action)
            result.append((action, (action, index), cost))
        return result
        # END_YOUR_CODE

def insertVowels(queryWords, bigramCost, possibleFills):
    # BEGIN_YOUR_CODE (our solution is 3 lines of code, but don't worry if you
deviate from this)
    if (len(queryWords) == 0):
        return ''

    queryWords.insert(0, wordsegUtil.SENTENCE_BEGIN)
    ucs = util.UniformCostSearch(verbose = 0)
    ucs.solve(VowelInsertionProblem(queryWords, bigramCost, possibleFills))

    if len(ucs.actions) == 0:
        return ''
    return ' '.join(ucs.actions)
    # END_YOUR_CODE

############################################################
# Problem 3b: Solve the joint segmentation-and-insertion problem

class JointSegmentationInsertionProblem(util.SearchProblem):
    def __init__(self, query, bigramCost, possibleFills):
        self.query = query
        self.bigramCost = bigramCost
        self.possibleFills = possibleFills

    def startState(self):
        # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if
you deviate from this)
        return (self.query, wordsegUtil.SENTENCE_BEGIN)
        # END_YOUR_CODE

    def isEnd(self, state):
        # BEGIN_YOUR_CODE (our solution is 2 lines of code, but don't worry if
you deviate from this)
        return len(state[0]) == 0
        # END_YOUR_CODE

    def succAndCost(self, state):
        # BEGIN_YOUR_CODE (our solution is 14 lines of code, but don't worry if
you deviate from this)
        result = []
        for i in range(1, len(state[0]) + 1):
            segWord = state[0][:i]
```

```
            restOfInput = state[0][i:]
            choices = self.possibleFills(segWord).copy()
            for word in choices:
                cost = self.bigramCost(state[1], word)
                result.append((word, (restOfInput, word), cost))
        return result
        # END_YOUR_CODE

def segmentAndInsert(query, bigramCost, possibleFills):
    if len(query) == 0:
        return ''

    # BEGIN_YOUR_CODE (our solution is 4 lines of code, but don't worry if you
deviate from this)
    ucs = util.UniformCostSearch(verbose = 1)
    ucs.solve(JointSegmentationInsertionProblem(query, bigramCost,
possibleFills))

    if len(ucs.actions) == 0:
        return ''
    return ' '.join(ucs.actions)
    # END_YOUR_CODE

############################################################

if __name__ == '__main__':
    shell.main()
```