Muskan Arora

102015040

ENC-2

```
import pickle
import numpy as np
```

```
from nltk.tokenize import word_tokenize
from functools import reduce
import nltk
nltk.download('punkt')
def parse_stories(lines):
    '''
    - Parse stories provided in the bAbI tasks format
    - A story starts from line 1 to line 15. Every 3rd line,
      there is a question &amp;amp;amp;amp;amp; answer.
    - Function extracts sub-stories within a story and
      creates tuples
    '''
    data = []
    story = []
    for line in lines:
        line = line.decode('utf-8').strip()
        nid, line = line.split(' ', 1)
        nid = int(nid)
        if nid == 1:
            # reset story when line ID=1 (start of new story)
            story = []
        if '\t' in line:
            # this line is tab separated Q, A &amp;amp;amp;amp;amp; support fact ID
            q, a, supporting = line.split('\t')
            # tokenize the words of question
            q = word_tokenize(q)
            # Provide all the sub-stories till this question
            substory = [x for x in story if x]
            # A story ends and is appended to global story data-set
            data.append((substory, q, a))
            story.append('')
        else:
            # this line is a sentence of story
            sent = word_tokenize(line)
            story.append(sent)
    return data

def get_stories(f):
    '''
    argument: filename
    returns list of all stories in the argument data-set file
    '''
    # read the data file and parse 10k stories
    data = parse_stories(f.readlines())
    # lambda func to flatten the list of sentences into one list
    flatten = lambda data: reduce(lambda x, y: x + y, data)
    # creating list of tuples for each story
    data = [(flatten(story), q, answer) for story, q, answer in data]
    return data
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
from keras.utils.data_utils import get_file
with open('qa1_single-supporting-fact_train.txt', 'rb') as f:
  train_data = get_stories(f)
```

```
with open('qa1_single-supporting-fact_test.txt', 'rb') as f:
  test_data = get_stories(f)
```

```
print(type(train_data))
print(type(test_data))
```

```
<class 'list'>
<class 'list'>
```

```
print("Length of the train data: ", len(train_data))
print("Length of the test data: ", len(test_data))
```

```
Length of the train data:  10000
Length of the test data:  1000
```

```
train_data[:2]
```

```
[(['Mary',
   'moved',
   'to',
   'the',
   'bathroom',
   '.',
   'John',
   'went',
   'to',
   'the',
   'hallway',
   '.'],
  ['Where', 'is', 'Mary', '?'],
  'bathroom'),
 (['Mary',
   'moved',
   'to',
   'the',
   'bathroom',
   '.',
   'John',
   'went',
   'to',
   'the',
   'hallway',
   '.',
   'Daniel',
   'went',
   'back',
   'to',
   'the',
   'hallway',
   '.',
   'Sandra',
   'moved',
   'to',
   'the',
   'garden',
   '.'],
  ['Where', 'is', 'Daniel', '?'],
  'hallway')]
```

```
# Train_data is a list of tuples consist of 3 parts: story, question, answer.
train_data[0]
```

```
(['Mary',
  'moved',
  'to',
  'the',
  'bathroom',
  '.',
  'John',
  'went',
  'to',
  'the',
  'hallway',
  '.'],
 ['Where', 'is', 'Mary', '?'],
 'bathroom')
```

```
' '.join(train_data[0][0])
```

```
'Mary moved to the bathroom . John went to the hallway .'
```

```
' '.join(train_data[0][1])
```

```
'Where is Mary ?'
```

```
train_data[0][2]
```

```
'bathroom'
```

```
all_data = test_data + train_data
```

```
len(all_data)
```

```
11000
```

```
set(train_data[0][0])
```

```
{'.', 'John', 'Mary', 'bathroom', 'hallway', 'moved', 'the', 'to', 'went'}
```

```
# Build vocabulary from all stories and questions
vocab = set()

for story, question, answer in all_data:
    vocab = vocab.union(set(story))
    vocab = vocab.union(set(question))
```

```
vocab.add('no')
```

```
vocab.add('yes')
```

```
vocab
```

```
{'.',
 '?',
 'Daniel',
 'John',
 'Mary',
 'Sandra',
 'Where',
 'back',
 'bathroom',
 'bedroom',
 'garden',
 'hallway',
 'is',
 'journeyed',
 'kitchen',
 'moved',
 'no',
 'office',
 'the',
 'to',
 'travelled',
 'went',
 'yes'}
```

```
# Add one to length of vocabulary: Keras embedding layer requires this.
vocab_len = len(vocab) + 1
print("Actual length of the vocabulary: ", vocab_len-1)
```

```
Actual length of the vocabulary:  23
```

```
# Length of all the stories
all_story_len = [len(data[0]) for data in all_data]
```

```
# Get maximum of the stories
max_story_len = max(all_story_len)
```

```
max_question_len = max([len(data[1]) for data in all_data])
```

```
print("Maximum length of the stories: ", max_story_len)
print("Maximum length of the question: ", max_question_len)
```

```
Maximum length of the stories:  68
Maximum length of the question:  4
```

```
from keras.utils import pad_sequences
from keras.preprocessing.text import Tokenizer
```

```
tokenizer = Tokenizer(filters=[])
tokenizer.fit_on_texts(vocab)  # create a dictionary for the entire corpus
```

```
tokenizer.word_index
```

```
{'the': 1,
 'no': 2,
 'bathroom': 3,
 'garden': 4,
```

```
 'john': 5,
 'moved': 6,
 'where': 7,
 'went': 8,
 'mary': 9,
 'kitchen': 10,
 'sandra': 11,
 'is': 12,
 'hallway': 13,
 'travelled': 14,
 '?': 15,
 'daniel': 16,
 'back': 17,
 'journeyed': 18,
 '.': 19,
 'bedroom': 20,
 'to': 21,
 'office': 22,
 'yes': 23}
```

```
train_story_text = []
train_question_text = []
train_answers = []
```

```
for story, question, answer in train_data:
    train_story_text.append(story)
    train_question_text.append(question)
    train_answers.append(answer)
```

```
# Train_story_text is a list of lists of words
train_story_text[:2]
```

```
[['Mary',
  'moved',
  'to',
  'the',
  'bathroom',
  '.',
  'John',
  'went',
  'to',
  'the',
  'hallway',
  '.'],
 ['Mary',
  'moved',
  'to',
  'the',
  'bathroom',
  '.',
  'John',
  'went',
  'to',
  'the',
  'hallway',
  '.',
  'Daniel',
  'went',
  'back',
  'to',
  'the',
  'hallway',
  '.',
  'Sandra',
  'moved',
  'to',
  'the',
  'garden',
  '.']]
```

```
# transforms each text into a sequence of integers (word embedding)
# Vectorization in machine learning refers to the process of converting data into arrays of numerical values, known as vectors. to make c

train_story_seq = tokenizer.texts_to_sequences(train_story_text)
```

```
print(len(train_story_seq))
print(len(train_story_text))
```

```
10000
10000
```

```
train_story_text[:2]
```

```
[['Mary',
  'moved',
  'to',
  'the',
  'bathroom',
  '.',
  'John',
  'went',
  'to',
  'the',
  'hallway',
  '.'],
 ['Mary',
  'moved',
  'to',
  'the',
  'bathroom',
  '.',
  'John',
  'went',
  'to',
  'the',
  'hallway',
  '.',
  'Daniel',
  'went',
  'back',
  'to',
  'the',
  'hallway',
  '.',
  'Sandra',
  'moved',
  'to',
  'the',
  'garden',
  '.']]
```

```
# Word embedding result
train_story_seq[:2]
```

```
[[9, 6, 21, 1, 3, 19, 5, 8, 21, 1, 13, 19],
 [9,
  6,
  21,
  1,
  3,
  19,
  5,
  8,
  21,
  1,
  13,
  19,
  16,
  8,
  17,
  21,
  1,
  13,
  19,
  11,
  6,
  21,
  1,
  4,
  19]]
```

```
# Create our own list of list of word indicies with padding.
# any sequence that doesn't have the same length, you will pad your sequence to max_len, it will add 0's to it.
# in order to make all sequences in a batch fit a given standard length is to be done.
def vectorize_stories(data, word_index=tokenizer.word_index, max_story_len=max_story_len, max_question_len=max_question_len):
    # Stories = X
    X = []

    # Questions = Xq
    Xq = []

    # Y Correct Answer ['yes', 'no']
    Y = []
    for story, query, answer in data:

        # for each story
        # [23, 14, 15]
        x = [word_index[word.lower()] for word in story] #word.lower() lowers a string
        xq = [word_index[word.lower()] for word in query]
```

```
        y = np.zeros(len(word_index)+1)
        y[word_index[answer]] = 1

        X.append(x)   # X holds list of lists of word indices for stories.
        Xq.append(xq) # Xq holds list of lists for word indices for questions.
        Y.append(y) # Y holds lists of lists of (38) biniary numbers, only 1 of them is 1.

    return (pad_sequences(X, maxlen=max_story_len), pad_sequences(Xq, maxlen=max_question_len), np.array(Y))
```

```
inputs_train, queries_train, answers_train = vectorize_stories(train_data)
```

```
inputs_test, queries_test, answers_test = vectorize_stories(test_data)
```

```
inputs_test
```

```
array([[ 0,  0,  0, ...,  1,  3, 19],
       [ 0,  0,  0, ...,  1, 20, 19],
       [ 0,  0,  0, ...,  1, 10, 19],
       ...,
       [ 0,  0,  0, ...,  1, 20, 19],
       [ 0,  0,  0, ...,  1,  3, 19],
       [ 0,  0,  0, ...,  1,  4, 19]], dtype=int32)
```

```
answers_test
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```
tokenizer.word_index['yes']
```

```
23
```

```
tokenizer.word_index['no']
```

```
2
```

```
sum(answers_test)
```

```
array([  0.,   0.,   0., 149., 187.,   0.,   0.,   0.,   0.,   0., 157.,
         0.,   0., 154.,   0.,   0.,   0.,   0.,   0.,   0., 171.,   0.,
       182.,   0.])
```

```
#Architecture of Model
from keras.models import Sequential, Model
from keras.layers import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM
```

```
# placeholders for inputs
# Recall we technically have two inputs, stories and questions. So we need to use placeholders. Input() is used to instantiate a Keras te
input_sequence = Input((max_story_len,))
question = Input((max_question_len,))
question
```

```
<KerasTensor: shape=(None, 4) dtype=float32 (created by layer 'input_2')>
```

```
# vocab_len
vocab_size = len(vocab) + 1
```

```
# INPUT ENCODER M
input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim=vocab_size, output_dim=64))
input_encoder_m.add(Dropout(0.3))

# OUTPUT
# (samples, story_maxlen, embedding_dim)
```

```
# INPUT ENCODER C
input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim=vocab_size, output_dim=max_question_len))
input_encoder_c.add(Dropout(0.3))
```

```
# OUTPUT
# (samples, story_maxlen, max_question_len)
```

```
# question encoder
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim=vocab_size, output_dim=64, input_length=max_question_len))
question_encoder.add(Dropout(0.3))

# OUTPUT
# (samples, query_maxlen, embedding_dim)
```

```
# ENCODED <---- ENCODER(INPUT)
# # encode input sequence and questions (which are indices) to sequences of dense vectors
input_encoded_m = input_encoder_m(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)

# input_encoded_m: (batch_size, story_maxlen, embedding_dim)
# input_encoded_c: (batch_size, story_maxlen, query_maxlen)
# question_encoded: (batch_size, query_maxlen, embedding_dim)
```

```
print(input_encoded_m.shape)
print(question_encoded.shape)
```

```
(None, 68, 64)
(None, 4, 64)
```

```
match = dot([input_encoded_m, question_encoded], axes=(2,2)) # why axes is (2,2) ==> dot product along the embedding dim (64 numbers dot
match = Activation('softmax')(match)

# NOTE: match after dot: (batch_size, story_maxlen, query_maxlen)
# match after Activation: (batch_size, story_maxlen, query_maxlen)
```

```
response = add([match, input_encoded_c]) # (samples, story_maxlen, query_maxlen)
response = Permute((2,1))(response) # (samples, query_maxlen, story_maxlen)
# Permutes the dimensions of the input according to a given pattern.
# Same as the input shape, but with the dimensions re-ordered according to the specified pattern

# response after add: (batch_size, story_maxlen, query_maxlen)
# response after Permute: (batch_size, query_maxlen, story_maxlen)
```

```
answer = concatenate([response, question_encoded])

# Note: answer: (batch_size, query_maxlen, story_maxlen+embedding_dim)
```

```
answer
# Note: answer: (batch_size, query_maxlen, story_maxlen+embedding_dim)
```

```
<KerasTensor: shape=(None, 4, 132) dtype=float32 (created by layer 'concatenate')>
```

```
# Reduce with RNN (LSTM) : does its work
answer = LSTM(32)(answer) #(samples, 32)
```

```
print(answer.shape)
```

```
(None, 32)
```

```
# Regularization with Dropout
answer = Dropout(0.5)(answer)
```

```
answer = Dense(vocab_size)(answer)
```

```
# we output a probability distribution over the vocabulary
answer = Activation('softmax')(answer)
```

```
answer
```

```
<KerasTensor: shape=(None, 24) dtype=float32 (created by layer 'activation_1')>
```

```
model = Model([input_sequence, question], answer)
#creating deep models
```

```
#model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 68)]         0           []

 input_2 (InputLayer)           [(None, 4)]          0           []

 sequential (Sequential)        (None, None, 64)     1536        ['input_1[0][0]']

 sequential_2 (Sequential)      (None, 4, 64)        1536        ['input_2[0][0]']

 dot (Dot)                      (None, 68, 4)        0           ['sequential[0][0]',
                                                                  'sequential_2[0][0]']

 activation (Activation)        (None, 68, 4)        0           ['dot[0][0]']

 sequential_1 (Sequential)      (None, None, 4)      96          ['input_1[0][0]']

 add (Add)                      (None, 68, 4)        0           ['activation[0][0]',
                                                                  'sequential_1[0][0]']

 permute (Permute)              (None, 4, 68)        0           ['add[0][0]']

 concatenate (Concatenate)      (None, 4, 132)       0           ['permute[0][0]',
                                                                  'sequential_2[0][0]']

 lstm (LSTM)                    (None, 32)           21120       ['concatenate[0][0]']

 dropout_3 (Dropout)            (None, 32)           0           ['lstm[0][0]']

 dense (Dense)                  (None, 24)           792         ['dropout_3[0][0]']

 activation_1 (Activation)      (None, 24)           0           ['dense[0][0]']

==================================================================================================
Total params: 25,080
Trainable params: 25,080
Non-trainable params: 0
__________________________________________________________________________________________________
```

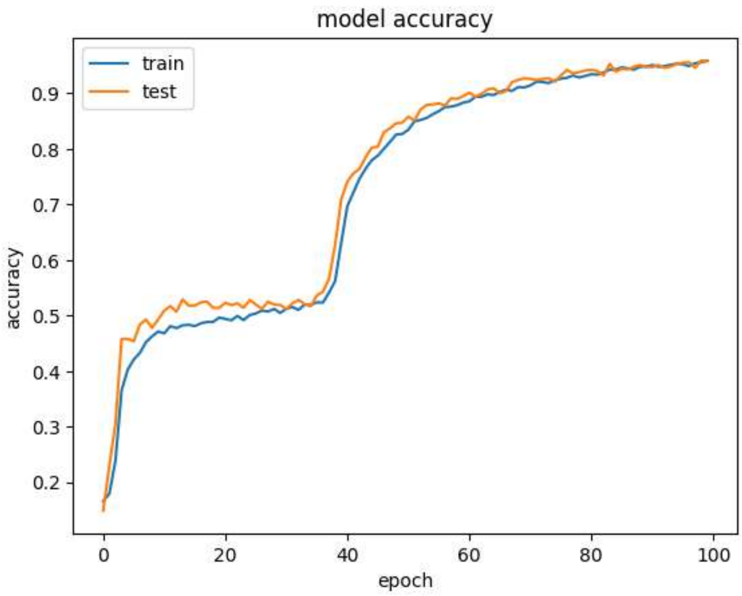
```
history = model.fit([inputs_train, queries_train], answers_train, batch_size=32, epochs=100, validation_data=([inputs_test, queries_test]
```

```
Epoch 1/100
313/313 [==============================] - 8s 13ms/step - loss: 2.0943 - accuracy: 0.1660 - val_loss: 1.8140 - val_accuracy: 0.14
Epoch 2/100
313/313 [==============================] - 4s 11ms/step - loss: 1.8529 - accuracy: 0.1795 - val_loss: 1.7733 - val_accuracy: 0.2
Epoch 3/100
313/313 [==============================] - 5s 15ms/step - loss: 1.7556 - accuracy: 0.2389 - val_loss: 1.6437 - val_accuracy: 0.30
Epoch 4/100
313/313 [==============================] - 3s 11ms/step - loss: 1.6134 - accuracy: 0.3650 - val_loss: 1.5060 - val_accuracy: 0.4
Epoch 5/100
313/313 [==============================] - 3s 11ms/step - loss: 1.5600 - accuracy: 0.4026 - val_loss: 1.4725 - val_accuracy: 0.4
Epoch 6/100
313/313 [==============================] - 5s 15ms/step - loss: 1.5360 - accuracy: 0.4210 - val_loss: 1.4595 - val_accuracy: 0.4
Epoch 7/100
313/313 [==============================] - 3s 11ms/step - loss: 1.5081 - accuracy: 0.4328 - val_loss: 1.4378 - val_accuracy: 0.4
Epoch 8/100
313/313 [==============================] - 5s 15ms/step - loss: 1.4814 - accuracy: 0.4522 - val_loss: 1.4046 - val_accuracy: 0.49
Epoch 9/100
313/313 [==============================] - 5s 16ms/step - loss: 1.4475 - accuracy: 0.4628 - val_loss: 1.4077 - val_accuracy: 0.4
Epoch 10/100
313/313 [==============================] - 3s 11ms/step - loss: 1.4222 - accuracy: 0.4713 - val_loss: 1.3539 - val_accuracy: 0.49
Epoch 11/100
313/313 [==============================] - 3s 11ms/step - loss: 1.4049 - accuracy: 0.4681 - val_loss: 1.3399 - val_accuracy: 0.50
Epoch 12/100
313/313 [==============================] - 4s 12ms/step - loss: 1.3902 - accuracy: 0.4811 - val_loss: 1.3140 - val_accuracy: 0.5
Epoch 13/100
313/313 [==============================] - 6s 18ms/step - loss: 1.3896 - accuracy: 0.4774 - val_loss: 1.3150 - val_accuracy: 0.50
Epoch 14/100
313/313 [==============================] - 3s 11ms/step - loss: 1.3781 - accuracy: 0.4825 - val_loss: 1.2964 - val_accuracy: 0.5
Epoch 15/100
313/313 [==============================] - 3s 11ms/step - loss: 1.3633 - accuracy: 0.4835 - val_loss: 1.3063 - val_accuracy: 0.5
Epoch 16/100
313/313 [==============================] - 4s 14ms/step - loss: 1.3604 - accuracy: 0.4810 - val_loss: 1.2775 - val_accuracy: 0.5
Epoch 17/100
313/313 [==============================] - 4s 12ms/step - loss: 1.3467 - accuracy: 0.4862 - val_loss: 1.2730 - val_accuracy: 0.5
Epoch 18/100
313/313 [==============================] - 4s 11ms/step - loss: 1.3349 - accuracy: 0.4885 - val_loss: 1.2597 - val_accuracy: 0.5
Epoch 19/100
313/313 [==============================] - 4s 12ms/step - loss: 1.3257 - accuracy: 0.4885 - val_loss: 1.2565 - val_accuracy: 0.5
Epoch 20/100
313/313 [==============================] - 4s 14ms/step - loss: 1.3275 - accuracy: 0.4964 - val_loss: 1.2452 - val_accuracy: 0.5
```

```
Epoch 21/100
313/313 [==============================] - 4s 11ms/step - loss: 1.3144 - accuracy: 0.4940 - val_loss: 1.2509 - val_accuracy: 0.5
Epoch 22/100
313/313 [==============================] - 4s 11ms/step - loss: 1.3175 - accuracy: 0.4915 - val_loss: 1.2390 - val_accuracy: 0.5
Epoch 23/100
313/313 [==============================] - 5s 15ms/step - loss: 1.3018 - accuracy: 0.4994 - val_loss: 1.2375 - val_accuracy: 0.5
Epoch 24/100
313/313 [==============================] - 3s 11ms/step - loss: 1.3095 - accuracy: 0.4918 - val_loss: 1.2387 - val_accuracy: 0.5
Epoch 25/100
313/313 [==============================] - 3s 11ms/step - loss: 1.2926 - accuracy: 0.5010 - val_loss: 1.2313 - val_accuracy: 0.5
Epoch 26/100
313/313 [==============================] - 4s 13ms/step - loss: 1.2907 - accuracy: 0.5038 - val_loss: 1.2384 - val_accuracy: 0.5
Epoch 27/100
313/313 [==============================] - 4s 12ms/step - loss: 1.2793 - accuracy: 0.5089 - val_loss: 1.2285 - val_accuracy: 0.5
Epoch 28/100
313/313 [==============================] - 3s 11ms/step - loss: 1.2827 - accuracy: 0.5073 - val_loss: 1.2202 - val_accuracy: 0.5
```

```
import matplotlib.pyplot as plt
%matplotlib inline
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
model.save('mybrandnewmodel.h5')
```

```
pred_result = model.predict(([inputs_test, queries_test]))
```

```
32/32 [==============================] - 1s 3ms/step
```

```
pred_result.shape
```

```
(1000, 24)
```

```
pred_result[0]
```

```
array([3.1504215e-15, 3.5272553e-15, 3.8099169e-15, 3.4767931e-05,
       8.1328642e-07, 2.9902442e-15, 2.5731832e-15, 2.8998366e-15,
       2.7468196e-15, 4.3525063e-15, 7.8504842e-10, 3.6281610e-15,
       2.7918642e-15, 9.9996096e-01, 2.9918186e-15, 4.1651850e-15,
       3.1772267e-15, 2.9632641e-15, 3.3961161e-15, 2.9229148e-15,
       3.3056333e-06, 2.9486276e-15, 6.5802460e-08, 3.1249868e-15],
      dtype=float32)
```

```
index_word = {index: word for word, index in tokenizer.word_index.items()}
```

```
predictions = np.argmax(pred_result, axis=1)
pred_answers = [index_word[pred] for pred in predictions]
pred_answers
```

```
 'kitchen',
 'bedroom',
 'bathroom',
 'garden',
 'hallway',
 'hallway',
 'garden',
 'kitchen',
 'hallway',
 'bathroom',
 'office',
 'bedroom',
 'bedroom',
 'office',
 'garden',
 'garden',
 'garden',
 'bedroom',
 'kitchen',
 'hallway',
 'bedroom',
 'hallway',
 'hallway',
 'office',
 'bathroom',
 'garden',
 'garden',
 'kitchen',
 'kitchen',
 'bathroom',
 'office',
 'office',
 'garden',
 'hallway',
 'garden',
 'bedroom',
 'bathroom',
 'kitchen',
 'kitchen',
 'kitchen',
 'hallway',
 'office',
 'garden',
 'bathroom',
 'bedroom',
 'kitchen',
 'office',
 'garden',
 'garden',
 'hallway',
 'kitchen',
 'kitchen',
 'bathroom',
 'bathroom',
 'hallway',
 'bathroom',
 'bathroom',
 'kitchen',
 'kitchen',
```

0s completed at 11:41 PM