```
In [1]:
    from torch.utils.data import DataLoader
    import torch
    from torch.utils.data.dataset import Dataset
    import torch.nn as nn
    import numpy as np
    import torch.optim as optim
    import operator
    import time
    from torch.optim.lr_scheduler import StepLR
    from itertools import chain
    from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence,pad_packed_sequence
```

Reading all the data

```
In [2]: ## Defining Hyperparameters
        batch_size=16
        dimension_embedding = 100
        hidden_dim = 256
        lstm_layers = 1
        lstm_dropout = 0.33
        linear_output_dim = 128
        batch_size = 16
        learning_rate = 0.1
        num_epochs = 20
        ## Reading the train data
        train_data=[]
        trainPath="train"
        dev_data=[]
        devPath="dev"
        test_data = []
        testPath="test"
        with open(trainPath, "r") as trainFile:
            for x in trainFile:
                x=x.rstrip()
                train_data.append(x.split(" "))
        with open(devPath, "r") as devFile:
            for x in devFile:
                x=x.rstrip()
                dev_data.append(x.split(" "))
        with open(testPath, "r") as testFile:
            for x in testFile:
                x=x.rstrip()
                test_data.append(x.split(" "))
```

```
In [3]: # create list of train tagged words
        train_word_tag_list=[]
        for each_line in train_data:
             if len(each_line)<2:</pre>
                 continue
            else:
                 train_word_tag_list.append([each_line[1],each_line[2]])
        # create list of dev tagged words
        dev__word_tag_list=[]
        for x in dev_data:
            if len(x) < 2:
                continue
            else:
                 dev\_word\_tag\_list.append((x[1],x[2]))
        # create list of test tagged words
        test_word_tag_list=[]
        for x in test_data:
            if len(x)>1:
                 test_word_tag_list.append((x[0],x[1]))
        # creating the each sentences (word and tag) pair from the train data
        train_sentence_tag_list=[]
        check=0
        tstl=[]
        for x in train_data:
            if len(x)>1:
                     if x[0]=='1':
                         check+=1
                         if check == 1:
                             tstl=[]
                             tstl.append((x[1],x[2]))
                         elif check==14987:
                             train_sentence_tag_list.append(tstl)
                             tstl=[]
                             tstl.append((x[1],x[2]))
                             train_sentence_tag_list.append(tstl)
```

3/27/23, 11:45 PM Task2

```
else:
                    train_sentence_tag_list.append(tstl)
                    tstl=[]
                    tstl.append((x[1],x[2]))
            else:
                tstl.append((x[1],x[2]))
# creating the each sentences (word and tag) pair from the train data
dev_sentence_tag_list=[]
check_dev=0
tstl_dev=[]
for x in dev_data:
    if len(x)>1:
            if x[0]=='1':
                check_dev+=1
                if check_dev == 1:
                    tstl_dev=[]
                    tstl_dev.append((x[1],x[2]))
                elif check_dev==3466:
                    dev_sentence_tag_list.append(tstl_dev)
                    tstl_dev=[]
                    tstl_dev.append((x[1],x[2]))
                    dev_sentence_tag_list.append(tstl_dev)
                    break
                else:
                    dev_sentence_tag_list.append(tstl_dev)
                    tstl_dev=[]
                    tstl_dev.append((x[1],x[2]))
            else:
                tstl_{dev.append((x[1],x[2]))}
# creating the each sentences (word and tag) pair from the test data
test_sentence_tag_list=[]
check=0
tstl_test=[]
for x in test_data:
    if len(x)>1:
            if x[0]=='1':
                check+=1
                if check == 1:
                    tstl_test=[]
                    tstl_test.append((x[1]))
                elif check==3684:
                    test_sentence_tag_list.append(tstl_test)
                    tstl_test=[]
                    tstl_test.append((x[1]) )
                    test_sentence_tag_list.append(tstl_test)
                    break
                else:
                    test_sentence_tag_list.append(tstl_test)
                    tstl_test=[]
                    tstl_test.append((x[1]))
            else:
                tstl_test.append((x[1]))
```

Defining parameters

```
In [4]: batch_size=16
  dimension_embedding = 100
```

Vocab creation

```
In [5]: vocab_data={}
        counter=0
         for x in train_data:
            if len(x)>1:
                 if x[1] in vocab_data:
                     temp=vocab_data[x[1]]
                     vocab_data[x[1]]=temp+1
                else:
                     vocab_data[x[1]]=1
         unk=0
         list_of_unk=[]
         for key in vocab_data:
            if vocab_data[key]<=2:</pre>
                unk+=vocab_data[key]
                 list_of_unk.append(key)
            else:
                continue
         sorted_vd = sorted(vocab_data.items(), key=operator.itemgetter(1),reverse=True)
```

3/27/23, 11:45 PM Task2

Word2ldx Mapping

```
In [6]: vocab_file_map={}
starter=1
for i in sorted_vd:
    starter+=1
    x= i[0]
    y= i[1]
    vocab_file_map[starter]= (x,y)

# create a mapping from words to integers
word2idx= {word[0] : idx for idx, word in vocab_file_map.items()}
word2idx['PAD>']= 0
word2idx['CUNK>']= 1
```

Tag2Idx Mapping

```
In [7]: tag_data_map={}
        for x in train_data:
            if len(x)>1:
                 if x[2] in tag_data_map:
                     temp=tag_data_map[x[2]]
                     tag_data_map[x[2]]=temp+1
                else:
                     tag_data_map[x[2]]=1
        # create a mapping from tags to integers
In [8]:
        tag2idx ={}
        tag_map=0
        tag2idx['<PAD>']= tag_map
        for key, value in tag_data_map.items():
                tag_map+=1
                 tag2idx[key]=tag_map
In [9]: | idx2tag={}
        for key, value in tag2idx.items():
             idx2tag[value]=key
```

GloVe Word Embeddings

```
In [10]: g_index_embeddings = {}
         with open("glove", 'r', encoding='utf-8') as f:
             for x in f:
                 data = x.split()
                 word = data[0]
                 coefs = np.asarray(data[1:],dtype='float32')
                 g_index_embeddings[word] = coefs
In [11]: # Create a list of numpy arrays using the values of `g_index_embeddings` dictionary
         arrays_list = [np.array(v) for v in g_index_embeddings.values()]
         # Stack the numpy arrays vertically to create a 2D array
         z = np.vstack(arrays_list)
         \# Calculate the mean of the 2D array column-wise along axis 0
         mean_array = np.mean(z, axis=0)
In [12]: word2idx['<PAD>']
         g_index_embeddings['<UNK>']=mean_array
         g_index_embeddings['<PAD>']=np.zeros(100)
In [13]: embedding_data = torch.zeros(len(word2idx), dimension_embedding)
         for word, i in word2idx.items():
             gl_arr = g_index_embeddings.get(word.lower(), mean_array)
             embedding vector = torch.tensor([float(val) for val in gl arr])
             if embedding_vector is not None:
                 embedding data[i] = embedding vector
```

Data set creation

```
In [14]: # sentences and labels
         trainSentences = [[t[0] for t in sublst] for sublst in train_sentence_tag_list]
         trainTags = [[t[1] for t in sublst] for sublst in train_sentence_tag_list]
         devSentences = [[t[0] for t in sublst] for sublst in dev_sentence_tag list]
         devTags = [[t[1] for t in sublst] for sublst in dev_sentence_tag_list]
         testSentences = [[t[0] for t in sublst] for sublst in test_sentence_tag_list]
In [15]: class creating_iterator(torch.utils.data.Dataset):
             def __init__(self, sentences, labels, word2idx, tag2idx):
                 self.sentences = sentences
                 self.labels = labels
                 self.word2idx = word2idx
                 self.tag2idx = tag2idx
             def __len__(self):
                 return len(self.sentences)
             def __getitem__(self, idx):
                 sentence = self.sentences[idx]
                 class_label = self.labels[idx]
                 # Create a list of boolean flags where 1 corresponds to lowercase and 0 corresponds to uppercase
                 sentence_flags = [int(word.lower() == word) for word in sentence]
                 # Convert the words and labels to their corresponding indices using word2idx and tag2idx
                 converted sentence = [self.word2idx.get(word, self.word2idx['<UNK>']) for word in sentence]
                 converted_labels = [self.tag2idx.get(tag, 0) for tag in class_label]
                 return converted_sentence, converted_labels, sentence_flags
         train dataset fnn = creating iterator(trainSentences,trainTags,word2idx,tag2idx)
         test dataset fnn = creating iterator(devSentences,devTags,word2idx,tag2idx)
In [18]: def collate fn(batch):
             # Pad the sentences, labels, and flags with zeros using pad_sequence
             padded_sentences = pad_sequence([torch.LongTensor(sentence) for sentence, _, _ in batch], batch_first=True)
             padded_labels = pad_sequence([torch.LongTensor(label) for _, label, _ in batch], batch_first=True)
             sent_flags = pad_sequence([torch.LongTensor(flag) for _, _, flag in batch], batch_first=True)
             # Calculate the sentence lengths
             sentence_lengths = torch.LongTensor([len(sentence) for sentence, _, _ in batch])
             return padded_sentences, padded_labels, sentence_lengths, sent_flags
In [20]: # create PyTorch DataLoader objects for batching the data
         train_loader = DataLoader(train_dataset_fnn, batch_size=batch_size, shuffle=True, collate_fn=collate_fn)
         dev_loader = DataLoader(test_dataset_fnn, batch_size=batch_size, shuffle=False, collate_fn=collate_fn)
In [21]: class TestDataset(torch.utils.data.Dataset):
             def __init__(self, sentences, word2idx):
                 self.sentences = sentences
                 self.word2idx = word2idx
             def __len__(self):
                 return len(self.sentences)
             def __getitem__(self, idx):
                 sentence = self.sentences[idx]
                 # Create a list of boolean flags where 1 corresponds to lowercase and 0 corresponds to uppercase
                 sentence_flags = [int(word.lower() == word) for word in sentence]
                 # Convert the words to their corresponding indices using word2idx
                 converted_sentence = [self.word2idx.get(word, self.word2idx['<UNK>']) for word in sentence]
                 return converted_sentence, sentence_flags
In [23]: def test_collate(batch):
             # Separate the sentences and flags in the batch
             sentences, flags = zip(*batch)
             # Pad the sentences with zeros using pad_sequence
             padded_sentences = pad_sequence([torch.LongTensor(sentence) for sentence in sentences], batch_first=True)
             # Calculate the sentence lengths
             sentence lengths = torch.LongTensor([len(s) for s in sentences])
             # Pad the flags with zeros using pad_sequence
             padded flags = pad sequence([torch.LongTensor(flag) for flag in flags], batch first=True)
             return padded_sentences, sentence_lengths, padded_flags
```

```
In [25]: test_dataset_fnn=TestDataset(testSentences,word2idx)
   test_loader = DataLoader(test_dataset_fnn, batch_size=batch_size, collate_fn=test_collate)
```

Model

```
In [26]: tag_pad_idx=tag2idx['<PAD>']
         word pad idx=word2idx['<PAD>']
In [27]: class BiLSTM(nn.Module):
             def __init__(self, input_dim, embedding_dim, hidden_dim, num_labels, lstm_layers, output_dim,
                         emb_dropout, lstm_dropout, fc_dropout, word_pad_idx, pretrained_embed):
                 super().__init__()
                 self.embedding_dim = embedding_dim
                 # LAYER 1: Embedding
                 self.embedding = nn.Embedding.from_pretrained(pretrained_embed, freeze=False)
                 self.emb_dropout = nn.Dropout(emb_dropout)
                 # LAYER 2: BiLSTM
                 self.lstm = nn.LSTM(
                     input_size=101,
                     hidden_size=hidden_dim,
                     num_layers=lstm_layers,
                     bidirectional=True,
                     dropout=lstm_dropout if lstm_layers > 1 else 0
                 # LAYER 3: Fully-connected
                 self.dropout3 = nn.Dropout(lstm_dropout)
                 self.elu = nn.ELU()
                 self.fc = nn.Linear(hidden_dim * 2, output_dim)
                 self.linear2 = nn.Linear(output_dim, num_labels)
             def forward(self, sentence, sentence_lengths, sentence_flags):
                 embedded = self.embedding(sentence)
                 concatenated_tensor = torch.cat((embedded, sentence_flags.unsqueeze(-1)), dim=-1)
                 packed_embedded = pack_padded_sequence(concatenated_tensor, sentence_lengths, batch_first=True, enforce_sorted
                 packed_output, (hidden, cell) = self.lstm(packed_embedded)
                 output, output_lengths = pad_packed_sequence(packed_output, batch_first=True)
                 ner out = self.fc(self.elu(output))
                 out = self.linear2(ner out)
                 return out
             def init_weights(self):
                 # to initialize all parameters from normal distribution
                 # helps with converging during training
                 for name, param in self.named_parameters():
                     nn.init.normal_(param.data, mean=0, std=0.1)
             def count_parameters(self):
                 return sum(p.numel() for p in self.parameters() if p.requires_grad)
In [29]: | bilstm = BiLSTM(
             input_dim=len(word2idx),
             embedding_dim=100,
             hidden_dim=256,
             num_labels = len(tag2idx),
             output_dim=128,
             lstm_layers=1,
             lstm_dropout=0.33,
             fc_dropout=0.25,
             emb dropout=0.5,
             word pad idx=word pad idx,
             pretrained_embed=embedding_data
         bilstm.init_weights()
In [30]: dev lengths=[]
         for a in dev_sentence_tag_list:
             dev_lengths.append(len(a))
         test_lengths=[]
         for b in test_sentence_tag_list:
             test lengths.append(len(b))
```

To Run the model

```
In [31]:
    def __init__(self, model, train_loader,test_loader,dev_loader,test_sentence_tag_list,dev_sentence_tag_list,optimiz
        self.model = model
        self.data_train = train_loader
        self.data_dev=dev_loader
        self.data_test=test_loader
        self.optimizer = optimizer_cls(model.parameters(),lr=0.5)
        self.loss_fn = nn.CrossEntropyLoss(ignore_index=0)
        self.scheduler = StepLR(self.optimizer, step_size=3, gamma=0.1)
        print(self.scheduler)

    @staticmethod
    def epoch_time(start_time, end_time):
```

```
elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
def accuracy(self, preds, y):
    max_preds = preds.argmax(dim=1, keepdim=True) # get the index of the max probability
    non_pad_elements = (y != tag_pad_idx).nonzero() # prepare masking for paddings
    # print("non_pad_elements", non_pad_elements)
    correct = max_preds[non_pad_elements].squeeze(1).eq(y[non_pad_elements])
    return correct.sum() / torch.FloatTensor([y[non_pad_elements].shape[0]])
def epoch(self):
   epoch_loss = 0
    epoch acc = 0
    self.model.train()
    for text, true_tags,sentence_lengths,sentence_flags in self.data_train:
        self.optimizer.zero_grad()
        pred_tags = self.model(text,sentence_lengths,sentence_flags)
        pred_tags = pred_tags.view(-1, pred_tags.shape[-1])
        true_tags = true_tags.view(-1)
        batch_loss = self.loss_fn(pred_tags, true_tags)
        batch_acc = self.accuracy(pred_tags, true_tags)
        batch_loss.backward()
        self.optimizer.step()
        epoch_loss += batch_loss.item()
        epoch_acc += batch_acc.item()
        self.scheduler.step( epoch_loss / len(self.data_train))
    return epoch_loss / len(self.data_train), epoch_acc / len(self.data_train)
def evaluate(self):
    epoch_loss = 0
    epoch_acc = 0
    self.model.eval()
   with torch.no_grad():
        for text, true_tags,sentence_lengths,sentence_flags in self.data_dev:
            pred tags = self.model(text,sentence lengths,sentence flags)
            pred_tags = pred_tags.view(-1, pred_tags.shape[-1])
            true_tags = true_tags.view(-1)
            batch_loss = self.loss_fn(pred_tags, true_tags)
            batch_acc = self.accuracy(pred_tags, true_tags)
            epoch_loss += batch_loss.item()
            epoch_acc += batch_acc.item()
            self.scheduler.step(epoch_loss / len(self.data_dev))
    return epoch_loss / len( self.data_dev), epoch_acc / len( self.data_dev)
def train(self, n_epochs):
    valid_loss_min2 = np.Inf
    for epoch in range(n_epochs):
        start_time = time.time()
        train_loss, train_acc = self.epoch()
        end_time = time.time()
        epoch_mins, epoch_secs = NER.epoch_time(start_time, end_time)
        print(f"Epoch: {epoch + 1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s")
        print(f"\tTrn Loss: {train_loss:.3f} | Trn Acc: {train_acc * 100:.2f}%")
        val_loss, val_acc = self.evaluate()
        print(f"\tVal Loss: {val_loss:.3f} | Val Acc: {val_acc * 100:.2f}%")
        if val_loss<valid_loss_min2:</pre>
            torch.save(self.model, 'GloveEmbed.pt')
            valid_loss_min2=val_loss
    self.model.eval()
    predictions = []
    true_labels = []
   with torch.no_grad():
        for inputs, targets,sent_len,sent_fl in self.data_dev:
            outputs = self.model(inputs,sent_len,sent_fl)
            _, preds = torch.max(outputs, dim=2)
            predictions.extend(preds.tolist())
            true_labels.extend(targets.tolist())
    # Convert the predicted tag sequences to string representations
    predictions dev = []
    for sentence_tags in predictions:
        predicted_tags_list = [idx2tag[idx] for idx in sentence_tags]
        predictions_dev.append(predicted_tags_list)
    # Save the predictions to a file
   with open('GloveEmbed_dev_pred.txt', 'w') as f:
        for predicted_tags in predictions_dev:
            f.write(' '.join(predicted_tags) + '\n')
    self.model.eval()
    predictions_test = []
   with torch.no_grad():
        for inputs, sent_len,sent_fl in self.data_test:
            outputs = self.model(inputs,sent len,sent fl)
            , preds = torch.max(outputs, dim=2)
            predictions_test.extend(preds.tolist())
```

```
# Convert the predicted tag sequences to string representations
test_preds = []
for sentence_tags in predictions_test:
    predicted_tags_list = [idx2tag[idx] for idx in sentence_tags]
    test_preds.append(predicted_tags_list)

# Save the predictions to a file
with open('GloveEmbed_test_pred.txt', 'w') as f:
    for predicted_tags in test_preds:
        f.write(' '.join(predicted_tags) + '\n')
```

<torch.optim.lr_scheduler.StepLR object at 0x7ff6908ebb50>

/Users/apurvagupta/opt/anaconda3/lib/python3.9/site-packages/torch/optim/lr_scheduler.py:163: UserWarning: The epoch p arameter in `scheduler.step()` was not necessary and is being deprecated where possible. Please use `scheduler.step()` to step the scheduler. During the deprecation, if epoch is different from None, the closed form is used instead of the new chainable form, where available. Please open an issue if you are unable to replicate your use case: https://github.com/pytorch/issues/new/choose.

warnings.warn(EPOCH_DEPRECATION_WARNING, UserWarning)

```
Epoch: 01 | Epoch Time: 2m 35s
  Trn Loss: 0.344 | Trn Acc: 88.41%
  Val Loss: 0.262 | Val Acc: 91.46%
Epoch: 02 | Epoch Time: 2m 33s
  Trn Loss: 0.183 | Trn Acc: 94.00%
  Val Loss: 0.177 | Val Acc: 94.85%
Epoch: 03 | Epoch Time: 2m 34s
  Trn Loss: 0.120 | Trn Acc: 96.23%
  Val Loss: 0.174 | Val Acc: 95.53%
Epoch: 04 | Epoch Time: 2m 47s
  Trn Loss: 0.082 | Trn Acc: 97.48%
  Val Loss: 0.155 | Val Acc: 96.17%
Epoch: 05 | Epoch Time: 2m 41s
  Trn Loss: 0.055 | Trn Acc: 98.34%
  Val Loss: 0.145 | Val Acc: 96.56%
Epoch: 06 | Epoch Time: 2m 41s
  Trn Loss: 0.038 | Trn Acc: 98.89%
  Val Loss: 0.147 | Val Acc: 96.67%
Epoch: 07 | Epoch Time: 2m 43s
  Trn Loss: 0.025 | Trn Acc: 99.31%
  Val Loss: 0.173 | Val Acc: 96.69%
Epoch: 08 | Epoch Time: 2m 40s
  Trn Loss: 0.017 | Trn Acc: 99.56%
  Val Loss: 0.180 | Val Acc: 96.79%
Epoch: 09 | Epoch Time: 2m 42s
  Trn Loss: 0.012 | Trn Acc: 99.71%
  Val Loss: 0.268 | Val Acc: 94.85%
Epoch: 10 | Epoch Time: 2m 40s
  Trn Loss: 0.007 | Trn Acc: 99.82%
  Val Loss: 0.202 | Val Acc: 96.92%
2, 2, 2, 2, 2, 2, 6, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 4, 5, 2,
                               2, 2, 2, 2, 2, 2, 2, 1,
6, 9, 2, 2, 1, 2, 2, 1, 2, 2, 4, 5, 2, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2], [6, 2, 1, 2, 2, 4,
5, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2, 1, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 1, 2, 2, 2, 2, 4, 5, 2,
2, 4, 5, 2, 2, 4, 5, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 1, 2, 2, 2, 2, 3, 2, 2, 4, 5, 2, 2,
4, 2, 2, 2, 2, 4, 5, 2, 2, 4, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 1, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 1, 2, 2, 4,
2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2,
2, 2, 2], [3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 6, 2, 2, 2, 2, 2]
2, 2, 2, 2, 2], [2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 1, 2, 4, 2, 5, 2, 2, 6, 2, 2], [2, 2, 2, 1, 2, 2,
2, 2, 2, 2, 2, 2], [2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2]
6, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 6, 9, 2, 2, 2], [6, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2], [2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 6,
2, 2|, |2, 2, 2,
     _ 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 0,
                                    9, 2,
                                      ۷,
                                       ۷,
[2, 2, 2, 2, 2, 2, 6, 9, 2, 6, 2, 2], [2, 2, 2, 1, 4, 5, 2, 2, 2, 2, 2, 2], [2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2], [2,
2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 4, 2, 2, 2, 6, 2, 2, 2, 2], [2, 2,
1, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 4, 2, 2, 2, 6, 9, 2, 6, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 4, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2,
```

```
2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2]
2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1,
2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1,
2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2]
4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 6, 2, 2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 6,
2, 2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 6, 2, 2, 4, 5, 2, 6, 9, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
2, 2, 2, 2], [4, 5, 2, 6, 2, 2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 6, 2, 2, 4, 5, 2, 6, 9, 2, 2,
2, 2, 2, 2, 2, 2, 2,
        ۷,
        2, 2,
```

2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2], [2, 2, 2, 2, 2], [1, 7, 2, 6, 2, 2], [1, 2, 6, 2, 2], [1, 2, 6, 2, 2, 2], [1, 2, 6, 2, 2, 2], [1, 7, 2, 6, 2, 2], [1, 2, 6, 2, 2, 2], [1, 2, 6, 2, 2, 2], [3, 8, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2] 2, 2], [2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2], [1, 2, 6, 2, 2, 2], [1, 2, 6, 2, 2, 2, 2], [1, 7, 2, 6, 2, 2, 2], [1, 7, 2, 6, 2, 2, 2], [1, 2, 6, 2, 2, 2, 2], [1, 7, 2, 6, 9, 2, 2, 2, 2, 2, 2, 2], [1, 2, 6, 9, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 3, 8, 2, 2, 2, 2, 2, 2, 2, 2], [6, 9, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 3, 8, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 7, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 7, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 1, 7, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2], [4, 5, 2, 2, 2, 2, 2, 3, 2, 4, 5, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 8, 2, 2, 2, 2, 2, 2, 2, 2], [2, [4, 5, 2, 2, 3, 7, [4, 2,

```
2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2,
4, 5, 2, 6, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2,
2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2], [2, 4, 5, 2,
6, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2],
[2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2,
6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2],
[2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2,
6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2,
2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4,
2], [2, 4, 5, 2, 1, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6,
2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2,
4, 5, 2, 6, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6,
2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2,
4, 5, 2, 6, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6,
2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2,
2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5,
2, 6, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2],
[2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2]
2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5,
2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2]
2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5,
2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2,
2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6,
2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2,
2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6, 2,
2], [2, 4, 5, 2, 6, 2, 2], [2, 4, 5, 2, 6, 2, 2], [2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2,
2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4,
5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2,
2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4,
5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2],
[2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2,
2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 2, 2, 2]
2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2],
```

```
[2, 4, 5, 2, 6, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4,
2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4,
5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4,
5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 4, 5, 2,
6, 2, 2, 4, 2], [4, 2, 6, 2, 2, 4, 5, 2, 6, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 4, 5, 2, 6, 2, 2, 4, 5], [2, 6,
2, 4, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2, 2, 2, 2, 2, 2], [4, 5, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2, 4, 2, 2,
2, 2, 2], [4, 5, 2, 4, 2, 4, 2, 2, 2, 2], [4, 2, 2, 4, 2, 4, 2, 2, 2, 2], [4, 5, 2, 4, 2, 4, 2, 2, 2, 2],
[4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2]
2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 4, 2, 2, 4, 5, 2, 2, 4, 2], [2, 2, 4, 2, 2, 4, 2, 2,
4, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 9, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2, 4, 2, 2, 2, 2, 2],
[4, 5, 2, 4, 5, 2, 2, 2, 2, 2, 2], [4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4,
5, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2, 2, 2, 2, 2, 2, 2], [4, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 4, 2, 4, 2, 4, 2, 2, 2], [2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 4, 2, 2], [6, 2, 2, 4,
2, 2, 4, 5, 2, 2, 4, 2], [2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 1, 7, 5, 2, 2, 2, 2], [2, 2, 2, 2, 6,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
```

2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 2, 2, 2, 2, 2] 2, 2], [3, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 3, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2] 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 7, 2, 2, 2, 2, 2, 2, 3, 8, 7, 7, 2], [2, 2,

Task2 |3, 2,

```
2, 2, 2, 2, 2,
Ι,
۷,
2, 2,
```

2, 2,

2, 2, 2, 2, 2, 2,

Task2 2, 2, 2], [2, 1, 7, 2, 4, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 4, 2,

2, 2, 2, 2, 2, 2, 2, 2, ۷, 2, 2, 2, 2, 3, 2, 2, 2, 3, 2, 6, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 7, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2,

5, 5, 2, 4], [2, 2, 1, 7, 7, 7, 2, 1, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2] 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2]

```
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2,
```

```
۷,
۷,
```

```
2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2], [2, 2, 2, 1, 2, 4, 5, 2, 2, 2, 2, 2, 2, 1, 2, 2, 3, 2, 4, 5, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2], [2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 6, 9, 9, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 3,
9, 9, 2, 6, 2, 2, 2, 2, 2, 2, 2, 3, 8, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 4, 5, 2, 2, 2, 2, 4, 5, 2,
2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 4, 5,
2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 3, 8, 8, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 6, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 3, 8, 8, 8, 2], [2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2],
[2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2]
2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 7, 2,
2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2]
2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 1, 2, 2, 2, 2, 2, 2], [2, 4,
1, 2, 2, 2, 2, 2, 2], [2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2,
2, 2], [2, 2, 2, 2, 2, 2, 2, 6, 2, 6, 2, 6, 2, 2], [6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 5, 2, 4,
5, 2, 2, 2], [4, 5, 2, 4, 5, 2, 2, 2, 2, 2, 2], [4, 5, 2, 2, 4, 5, 2, 2, 2, 2, 2], [4, 5, 2, 2, 4, 2, 2, 2, 2, 2],
[4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 2, 2, 2]
2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 4, 5, 2, 2, 2, 2],
[2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2], [4, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2], [2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2]
2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2, 4, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2, 4, 2, 2, 2, 2, 2], [4, 5, 2, 2, 2, 2]
4, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2, 2, 2, 2, 2, 2, 2], [4, 2, 4, 2, 4, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2,
4, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2, 4, 2, 2, 2, 2, 2], [4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 5, 2, 4, 2, 2,
2, 2, 2, 2, 2, 2], [4, 5, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2], [4, 2, 4, 2, 4, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2]
2, 2, 2], [2, 2, 4, 2, 2, 4, 2, 2, 4, 2, 2], [2, 2, 4, 2, 2, 4, 2, 2, 4, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2]
2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2]
2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2],
```

2, 2, 2], [6, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 2], [6, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 2, 2, 6, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 2, 2, 2], [6, 2, 2, 2, 2, 2, 2, 2], [6, 2, 4, 5, 2, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2], [6, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2], [4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 7, 7, 2, 2, 2, 2, 2, 2, 2, 1, 7, 2], [4, 2, 4, 5, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 4, 5, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 4, 5, 2, 2, 2, 2, 3, 2, 4, 5, 2, 2,

2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 2, 2, 2, 2], [2, 4, 5, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 9, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 4, 5, 2, 6, 9, 2, 2], [2, 4, 5, 5, 2, 6, 2, 2], [2, 4, 5, 2, 2, 2, 2, 2], [2, 4, 5, 5, 2, 6, 2, 2], [2, 4, 5, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2], [3, 8, 8, 2, 2, 2, 2, 2, 2], [2, 2, 4, 2, 2, 2, 2, 2, 2], [1, 7, 2, 1, 2, 2, 2, 2, 2], [1, 2, 2, 9, 2, 2, 2] 2, 2, 2, 2], [2, 2, 1, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 6, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2] 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2], [1, 7, 7, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2] 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2] 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2] 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2] 2], [1, 7, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2] 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2] 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2] 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 3, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2] 2, 2, 2, 2, 2, 2, 2], [2, 2, 3, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 1, 2, 2, 2, 2, 2, 2], [1, 2, 1, 7, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 1, 7, 2, 1, 2, 1, 7, 2], [2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2], [1, 2, 1, 7, 7, 2], [1, 2, 1, 2, 2, 2], [1, 2, 1, 2, 2, 2], [1, 2, 1, 2, 2, 2], [2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2], [1, 2, 1, 7, 2, 2], [1, 7, 2, 1, 2, 2], [1, 2, 1, 7, 2, 2], [1, 7, 7, 2, 1, 2], [2, 2, 2, 2, 2, 2], [2, 2, 3, 2, 2, 2], [6, 2, 2, 2, 2, 2], [2, 2, 3, 2, 2, 2, 2, 2]

Task2 $2,\ 2,\ 2],\ [2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2,\ 2],\ [1,\ 2,\ 2],\$ 2, 2, 2, 2, 2, 2], [1, 2, 1, 7, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2], [1, 7, 2, 1, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2], [2, 2, 1, 2, 1, 2, 1, 7, 2, 1], [2, 2, 2, 1, 7, 7, 2, 1, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 1, 2], [1, 2, 1, 7, 2], [1, 2, 1, 2, 2], [1, 2, 1, 2, 2], [1, 2, 1, 2, 2], [1, 2, 1, 2, 2], [1, 7, 2, 1, 2], [1, 2, 2, 2, 2], [2, 2, 1, 2, 1], [2, 2, 2, 2, 2], [1, 2, 1, 2, 2], [1, 7, 2, 1, 2], [1, 2, 1, 2, 2],2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2], [1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 6, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 4, 5, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 6, 2, 1, 2, 2, 1, 2, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 6, 9, 2, 1, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 1, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 4, 5, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 5, 2, 6, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 4, 5, 2, 6, 2, 2, 2], [2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 4, 2, 2, 2, 2, 3, 2, 4, 5, 2, 4, 5, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 4, 5, 2, 4, 5, 2, 2, 2, 2, 2, 2, 4, 5, 5, 2, 2, 2, 4, 5, 2, 2, 2, 4, 5, 2, 2, 2, 4, 5, 5, 5, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 7, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2, 2, 2, 2], [1, 2, 1, 7, 7, 7, 2, 2, 2, 2, 2, 2], [1, 7, 7, 7, 2, 1, 7, 2, 2, 2, 2, 2, 2], [1, 2, 1, 7, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 7, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2] 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2, 2], 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 7, 7, 2, 2, 2, 2, 2, 2] 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 6, 2, 2, 2, 2, 2, 2, 2], 2, ۷, 6, 2, 2, 2, 2, 2, 2, 2, 3, 8, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 4, 5, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],

2, 2], [2, 2, 2, 2, 3, 8, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 6, 2, 2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 3, 2, 4, 2, 4, 5, 4, 2, 2, 2], 2, 2, 2, [2, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 6, 2, 2, 4, 5, 2, 6, 2, 2, 2, 2, 2, 2, 4, 5, 2, 6, 2, 2, 4, 5, 2, 6, 2, 2, 4, 5, 2,

Task2 [2, 2, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2], [1, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2], [1, 7, 2, 6, 2, 2], [1, 2, 6, 2, 2, 2], [1, 2, 6, 2, 2, 2], [1, 2, 6, 2, 2, 2], [1, 2, 6, 2, 2, 2], [1, 2, 6, 2, 2, 2], [1, 7, 2, 6, 2, 2], [3, 8, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2] 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2], [1, 7, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2], [1, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 6, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 6, 9, 2, 2, 2, 2, 2] 2, 2, 2], [1, 2, 1, 7, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2] 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 1, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2], [3, 8, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 1, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 1, 2, 2, 2, 2, 2, 2] 2, 2], [1, 2, 1, 2, 2, 2, 2, 2, 2, 2], [1, 7, 2, 1, 7, 2, 2, 2, 2, 2], [1, 7, 2, 1, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 3, 2, 4, 2, 2, 2, 2], [2, 6, 9, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 7, 7, 2, 2, 1, 2, 2, 5, 2, 2, 2, 2, 2, 3, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2,

4, 5, 2, 4, 5, 2], [6, 2, 4, 5, 4, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 2, 2, 2, 4, 7, 2, 2, 2, 4, 5, 2, 4, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 4, 5, 5, 5, 2, 2, 2, 2, 4, 5, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2, 4, 5, 2, 2, 2, 4, 5, 5, 5, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2, 4, 5, 5, 5, 2, 2, 2, 2], [6, 2, 2, 2, 4, 5, 2, 4, 5, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 4, 5, 2, 2, 2, 4, 5, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2], [3, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 3, 2, 2, 2] 2, 2, 2, 2, 3, 8, 2, 2, 2, 2], [6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 3, 8, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2, 2, 2, [2, 2, 3, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2], [2, 2, 2, 1, 2, 4, 2, 2, 2, 2, 3, 3, 2, 4, 5, 2, 6, 2, 2, 3, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [4, 2, 4, 2, 1, 2, 6, 2, 2, 2, 4, 2], [2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 4, 5, 2], [2, 2, 2, 4, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 6, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

```
2, 2,
2, 2, 2, 2], [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 6, 2, 6, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 1, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1,
```

2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 3, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2], 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 8, 8, 2, 2, 2, 2, 2, 2, 8, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 3, 8, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2], [2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ۷, ۷,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ۷, ۷, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 6, 9, 2, 2, 2, 2, 2, 2, 2, 3, 8, 7, 7, 2, 4, 5, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [3, 2, 2, 2, 2, 3, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 3, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [6, 2, 4, 5, 2, 2, 2, 2, 6, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 6, 2, 6, 2, 6, 2, 6, 9, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 4, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 1, 7, 7, 7, 2, 2, 4, 5, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 7, 7, 2, 1, 2, 6, 2, 2, 2, 2, 2, 3, 8, 2, 6, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 4, 2], [2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 8, 2, 2, 2, 4, 2, 2, 2, 3, 9, 2, 2, 2, 2, 2, 2, 2, 6, 2, 4, 2, 2, 6, 2], [2, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 6, 2, 2, 3, 2, 2, 2, 2, 6, 2, 6, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 5, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 4, 5, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 6, 2, 2, 2], [6, 2, 2, 6, 2, 2, 4, 2, 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 2, 6, 2, 4, 5, 2, 2, 1, 2, 4, 5, 2, 2, 3, 3, 8, 2, 2, 2, 2, 2, 6, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 4, 5, 2, 4, 5, 2, 2, 2, 2, 2, 4, 5, 2, 2, 3, 3, 7, 2, 2, 7, 2,

```
2, 2, 2, 2, 2], [2, 2, 2, 2,
              2, 2, 2, 2, 2,
                    2, 2, 2, 2, 2,
                          2, 2,
                             2, 2,
                               2, 2,
                                 2, 2, 2, 2, 2,
                                       2, 2,
                          2, 2,
                             2, 2,
                               2, 2, 4, 2, 3, 2, 2,
   2, 2, 2, 2, 2], [4, 2, 2, 1,
              5, 2, 2, 2, 2,
                    2, 2, 4, 2, 2,
                                       2, 2,
   2, 2, 2, 2, 2], [4, 2, 2, 4,
              5, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                             2, 4,
                               5, 2, 2, 2, 4, 2, 2,
                                       2, 2, 2, 2, 2, 2, 2,
   2, 2, 2, 2, 2, 2, 2, 2, 2,
   2, 2, 2, 2, 2, 2, 2, 2, 2,
   2, 2, 2, 2, 2, 2, 2, 2, 2,
   2, 2, 2, 2, 2, 2, 2, 2, 2,
   CPU times: user 38min 14s, sys: 32min 19s, total: 1h 10min 34s
   Wall time: 28min 21s
In [34]: pred_dev=[]
   with open('GloveEmbed_dev_pred.txt', 'r') as file:
      for inputs in file:
        pred_dev.append(inputs.split(' '))
In [35]: dev_res_list=[]
   dev_st=[]
   dev_flag=0
   for x in dev_data:
     if len(x)>1:
        if x[0]=='1':
          dev_flag=dev_flag+1
          if dev_flag == 1:
           dev_st=[]
           dev_st.append((x[0],x[1],x[2]))
          elif dev_flag==3466:
           dev_res_list.append(dev_st)
           dev_st=[]
           dev_st.append((x[0],x[1],x[2]))
           dev_res_list.append(dev_st)
           break
         else:
           dev_res_list.append(dev_st)
           dev_st=[]
           dev_st.append((x[0],x[1],x[2]))
        else:
          dev_st.append((x[0],x[1],x[2]))
In [36]: result_dict = {}
   for i in range(len(dev_res_list)):
     for j in range(len(dev_res_list[i])):
      result_dict[idx] = (dev_res_list[i][j][0], dev_res_list[i][j][1],dev_res_list[i][j][2], pred_dev[i][j])
      idx += 1
In [37]: start_i=0
   with open("GloveEmbed_dev_pred_out_rerun.txt", 'w') as f:
     for key,i in result_dict.items() :
      if i[0] == '1' and start_i!=0:
        f.write('\n')
        f.write('%s %s %s %s\n' % (i[0], i[1], i[2], i[3]))
      else:
        f.write('%s %s %s %s\n' % (i[0], i[1], i[2], i[3]))
        start_i=start_i+1
In [38]: start_i=0
   with open("dev2.out", 'w') as f:
     for key,i in result_dict.items() :
      if i[0] == '1' and start_i!=0:
        f.write('\n')
        f.write('%s %s %s\n' % (i[0], i[1], i[3]))
      else:
        f.write('%s %s %s\n' % (i[0], i[1], i[3]))
        start_i=start_i+1
In [41]: !perl conll03eval < {'GloveEmbed_dev_pred_out_rerun.txt'}</pre>
```

```
processed 51578 tokens with 5942 phrases; found: 5751 phrases; correct: 4785.
         accuracy: 96.91%; precision: 83.20%; recall: 80.53%; FB1: 81.84
                       LOC: precision: 93.24%; recall: 81.06%; FB1: 86.72 1597
                      MISC: precision: 84.89%; recall: 76.79%; FB1: 80.64 834
                       ORG: precision: 79.98%; recall: 71.51%; FB1: 75.51 1199
                       PER: precision: 76.80%; recall: 88.44%; FB1: 82.21 2121
In [42]: test_res_list=[]
         st_test=[]
         flag_test=0
         for x in test_data:
             if len(x)>1:
                     if x[0]=='1':
                         flag_test=flag_test+1
                         if flag_test == 1:
                             st_test=[]
                             st_test.append((x[0],x[1]))
                         elif flag_test==3684:
                             print(flag_test)
                             test_res_list.append(st_test)
                             st_test=[]
                             st_test_append((x[0],x[1]))
                             test_res_list.append(st_test)
                         else:
                             test_res_list.append(st_test)
                             st_test=[]
                             st_test_append((x[0],x[1]))
                     else:
                         st_test.append((x[0],x[1]))
         3684
In [44]: pred_test=[]
         with open('GloveEmbed_test_pred.txt', 'r') as readFile:
                 for inputs in readFile:
                     pred_test.append(inputs.split(' '))
In [47]: test_dict = {}
         test_idx = 0
         for i in range(len(test_res_list)):
             for j in range(len(test_res_list[i])):
                 test_dict[test_idx] = (test_res_list[i][j][0], test_res_list[i][j][1], pred_test[i][j])
                 test_idx += 1
In [49]: start_ie=0
         with open("test2.out", 'w') as f:
             for key,i in test_dict.items() :
                 if i[0] == '1' and start_ie!=0:
                     f.write('\n')
                     f.write('%s %s %s\n' % (i[0], i[1], i[2]))
                 else:
                     f.write('%s %s %s\n' % (i[0], i[1], i[2]))
                     start_ie=start_ie+1
In [51]: torch.save(ner.model,'blstm2.pt')
 In [ ]:
```