Tian Qiu

0026535063

ECE 608 Assignment Report

**Part I**

Combinatorial Optimization Problem:

Given a string and a dictionary of words, determine how many ways the string can be break into a space-separated sequence of one or more dictionary words.

Informally:

Given s = "catsanddog", dict = ["cat", "cats", "and", "sand", "dog"]. Return True because " catsanddog" can break as "cat san dog".

Given s = "learncode", dict = ["learn", "code"]. Return True because "learncode" can break as "learn code".

Given s = "learncode", dict = ["lear", "code"]. Return False because "learncode" cannot break.

**Part II**

The time of method first come up to the brain is O(2^n) which is because it does not use memorizing method. We can use dynamic programming for this question.

The reason why this question can use dynamic programming is that we can initialize an array to record how many ways to break the string for each combination. In this way, program does not need to recalculate it, which is also called memorization.

Dynamic Programming method:

Step1: Characterize the structure of optimal solution

Set function F[i] which means whether substring 0 to index “i” can be separated perfectly.

Step2: Recursively define the value of optimal solution

Step3: Compute the value by bottom-up way

For I from 1 to N:

For j from 0 to N+1:

If string.substring(j, I) is a word:

Word\_flag = true

Else:

Word\_flag = false

F[i] = F[i] || F[j] && Word\_flag

End

End

Step4: Construct the solution from step 3

F[n] is exactly whether the string can be broken into a sequence of words.

Pseudo code:

Function WordBreak (string)

N = length of string

// initial array

Count

// initial array F to record whether the string can be separated to word

For I = 0 to N + 1:

F [I] = False

F[0] = True

For I from 1 to N:

For j from 0 to N+1:

If string.substring(j, I) is a word:

Word\_flag = true

Else:

Word\_flag = false

F[i] = F[i] || F[j] && Word\_flag

End

End

Return F[n]

Time analyze:

Because it has two loops and inside the loop, the time is constant. Also the initialization time is constant.

T(n) = O(N^2)

Examples:

S = "catsanddog", dict = ["cat", "cats", "and", "sand", "dog"]

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| String | C | A | T | S | A | N | D | D | O | G |
| F(n) | False | False | True | True | False | False | True | True | False | True |

S = "learncode", dict = ["lear", "code"]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| String | L | E | A | R | N | C | O | D | E |
| F(n) | False | False | False | True | False | False | False | False | False |

S = "learncode", dict = ["learn", "code"]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| String | L | E | A | R | N | C | O | D | E |
| F(n) | False | False | False | False | True | False | False | False | True |

**Part III**

|  |  |  |
| --- | --- | --- |
| String | Dictionary | Result |
| alsdkfjlaskdjfw | ["alsdkfjla", "skdjfw"] | True |
| Ilovetoeaticecreamandmango | ["sam","sung","man","mango",  "icecream","and","to","i","love","ice","cream"] | True |
| Learncode | ["lear", "code"] | False |
| catsanddog | ["cat", "cats", "and", "sand", "dog"] | True |
| Learncode | ["learn", "code"] | True |
| youbuymacpro | ["you", "pro",”buy”,”mac”,”macpro”] | True |
| moimpok | [] | False |
| qewoikm | [“qewoikm”] | True |
| rfvj | [“r”,”w”,”q”,”v”,”j”,”k”] | False |
| rfvj | [“r”,”w”,”q”,”v”,”j”,”k”,”f”] | True |