CS 180 Discussion 1A

Outline

- Traveling salesperson related questions
- Interview questions

Polynomial-time reductions

- Suppose $Y \leq_p X$. If Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.

Traveling salesperson

- Definition: Consider a salesman who must visit n cities labeled v_1, v_2, …, v_n. The salesman starts in city v_1, his home, and wants to find a tour - an order in which to visit all the other cities and return home.
- Goal: Find a tour that causes him to travel as little total distance as possible

Hamiltonian cycle

- Definition: Given a directed graph G = (V, E), we say that a cycle C in G is a Hamiltonian cycle if it visits each vertex exactly once.

Prove that TSP is NP-complete

- Reducing Hamiltonian cycle to TSP.
- Given a directed graph G = (V, E), we define the instances of TSP.
  - We have a city v_i' for each node vi of the graph G.
  - We define d(v_i', v_j') to be 1 if there is an edge (v_i, v_j) in G, and we define it to be 2 otherwise.
  - G has a hamiltonian cycle iff there is a tour of length at most n in TSP.

Euclidean traveling salesperson

A very natural restriction of the TSP is to require that the distances between cities form a metric to satisfy the triangle inequality; that is the direct connection from *A* to *B* is never farther than the route via intermediate *C*: $D_{AB} <= D_{AC} + D_{CB}$

Prove that Euclidean TSP is NPC.

Relaxed traveling salesperson

We relax the condition on the non-Euclidean TSP that each city is to be visited only once.

Prove that Relaxed TSP is NPC.

Interview questions

- Valid Parentheses [leetcode 20]
- Longest Valid Parentheses [leetcode 32]

Valid Parentheses

Given a string containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Example 1:

```
Input: "()"
Output: true
```

Example 2:

```
Input: "()[]{}"
Output: true
```

Example 3:

```
Input: "(]"
Output: false
```

Example 4:

```
Input: "([)]"
Output: false
```

Example 5:

```
Input: "{[]}"
Output: true
```

Valid Parentheses

Given a string containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Key Idea:

Use a stack

Extension: What if the sequences are composed of either ‘(’ or ‘)’

Is that possible to reduce the space complexity

Longest Valid Parentheses

Given a string containing just the characters `'('` and `')'`, find the length of the longest valid (well-formed) parentheses substring.

Example 1:

```
Input: "(()"
Output: 2
Explanation: The longest valid parentheses substring is "()"
```

Example 2:

```
Input: ")()())"
Output: 4
Explanation: The longest valid parentheses substring is "()()"
```

Longest Valid Parentheses

Given a string containing just the characters `'('` and `')'`, find the length of the longest valid (well-formed) parentheses substring.

Key idea:

Use two counters, one for ‘(’ and other one for ‘)’

Step 1: scan from left to right

Increase ‘(’ counter by 1 if it is ‘(’. Or Increase ‘)’ counter by 1 if it is ‘)’

If ‘(’ counter == ‘)’ counter:

  Update the longest valid parentheses

If ‘)’ counter > ‘(’ counter:

  The current parentheses is invalid. Reset these two counters to 0

Step 2: scan from right to left and do the same thing.

Example: ‘((( )) ’ result from the left scan = 0; result from the right scan = 2