算法设计与分析第三次作业 - 贪心策略

201628013229058 洪鑫

1 Undirected graph existence problem

1

问题描述

> Given a list of n natural numbers $d_1, d_2, \ldots, d_n$, show how to decide in polynomial time whether there exists an undirected graph $G = (V, E)$ whose node degrees are precisely the numbers $d_1, d_2, \ldots, d_n$. G should not contain multiple edges between the same pair of nodes, or “loop” edges with both endpoints equal to the same node.

基本思路

不失一般性地，可以假设$d_1, d_2, \ldots, d_n$是从大到小排好序的。那么，以$\{d_1, d_2, \ldots, d_n\}$为度数的点可以构成一个无向图是以$\{d_2 - 1, \ldots, d_{k+1} - 1, d_{k+2} \ldots, d_n\}$为度数的点可以构成无向图的充要条件，其中$k = d_1$。在正确性证明里，将给出其证明。

通过这种策略，可以不断减少集合的数。在缩减过程中，如果数出现负值，或者剩余长度小于缩减的那个数时，即$n < d_1 + 1$，则不存在以这些自然数为度数的无向图。缩减过程在所有剩下的数都为零时停止。

伪代码

```
GraphExist(d):
    sort d in decreasing order;
    i = 0;
    n = size(d);
    while (d[i] != 0)
        if n < d[i] + 1
            return false;
        else
            for j = i + 1 to d[i] + i
                d[j] = d[j] - 1;
                if d[j] < 0
                    return false
            i++;
            /* make sure the decreasing order */
```

```
            merge(d[i..j], d[j+1..n]);  /* merge process of merge sort */
    return true;
```

正确性证明（贪心策略证明 + 循环不变式）

首先证明**“以$\{d_1, d_2, \dots, d_n\}$为度数的点可以构成一个无向图是以$\{d_2-1, \dots, d_{k+1}-1, d_{k+2} \dots, d_n\}$为度数的点可以构成无向图的充要条件”**这句话的正确性，其中$k = d_1$。

<u>充分性</u>：给定一个以$\{d_2-1, \dots, d_{k+1}-1, d_{k+2} \dots, d_n\}$为度数的无向图，再来一个点，只要将这个点与集合里前d_1个点都连上一条边，就可以得到以$\{d_1, d_2, \dots, d_n\}$为度数的点构成的一个无向图。

<u>必要性</u>：假设$\{d_1, d_2, \dots, d_n\}$对应的边为$\{v_1, v_2, \dots, v_n\}$。如果v_1与$v_i(2 \le i \le d_1+1)$都有一条边，那么直接从图上消除d_1即可；如果不，那么$\exists\, j > d_1+1$, v_1与v_j之间存在一条边。如果$d_i = d_j$，只需要将两个点的序号交换一下就可以执行缩减过程，否则$d_i > d_j$，也因此$\exists k \notin \{1, i, j\}$使得$v_i$与$v_k$间有一条边，而$v_j$与$v_k$间不存在边。对现有的图做如下变换：去掉边$(v_1, v_j)$和$(v_i, v_k)$，在点$(v_1, v_i)$和$(v_k, v_j)$之间建立新的边，显然这种变换对所有点的度数没有发生改变。可以通过这种变换达到第一种情况，然后进行缩减过程。

接下来通过证明的定律来证明算法的正确性，我们把$d_{i+1} \dots d_n$始终保持非负递减的性质定义为一个<u>循环不变式</u>。

<u>初始化</u>：循环开始前，数组经过排序，且各元素均为自然数，循环不变式显然为真。

<u>保持</u>：考虑我们的算法，在第k次循环，假定开始前循环不变式为真。循环中，如果出现“数组中的数不够减”的情况，或者某个度数减为负值的情况，则可判定无向图不存在，程序退出。否则，在缩减执行后，对缩减的部分和未缩减的部分进行一次归并排序的合并过程，保持了序列的递减性质。循环不变式在循环结束后为真。

<u>终止</u>：循环在判定图不存在或发现某度数为零时终止，或者，遇到0时终止。数组的大小是有限的，每次循环i加1，持续缩减，必然程序能达到终止条件。因为遇到零达到终止条件时，我们显然可以构建一个图，这个图包含点的个数为集合剩下的数的个数，不包含边。通过上面证明的定律，我们可以得出该图既可以推出原始条件下也可以构造一个无线图。所以，算法正确。

复杂度分析

该算法的时间复杂度为$O(n^2)$。第一次排序的时间复杂度为$O(nlogn)$。while最多循环n次，每次for循环最多$d[i] < n$次。每次for循环结束后，为保证序列的降序排列，使用归并排序中的合并过程合并，时间复杂度为$O(n)$。

2 Time schedule

2

问题描述

There are n distinct jobs, labeled $J_1, J_2, \ldots, J_n$, which can be performed com- pletely independently of one another. Each jop consists of two stages: first it needs to be *preprocessed* on the supercomputer, and then it needs to be *finished* on one of the PCs. Let's say that job J_i needs p_i seconds of time on the su- percomputer, followed by f_i seconds of time on a PC. Since there are at least n PCs available on the premises, the finishing of the jobs can be performed on PCs at the same time. However, the supercomputer can only work on a single job a time without any interruption. For every job, as soon as the preprocessing is done on the supercomputer, it can be handed off to a PC for finishing.

Let's say that a *schedule* is an ordering of the jobs for the supercomputer, and the *completion time* of the schedule is the earliest time at which all jobs have finished processing on the PCs. Give a polynomial-time algorithm that finds a schedule with as small a completion time as possible.

基本思路

超级电脑处理的总时间是不变的，所以为了使总的时间最短，只要先执行在PC上花费时间比较长的任务即可使完成时间最短。所以只要按照J_i的f_i进行降序排序，即可得到拥有最短完成时间的序列。

伪代码

```
GreaterThan(J1, J2):
    return J1->f > J2->f;
```

```
Schedule(J):
    /* using GreaterThan() to compare jobs */
    merge_sort(J) in decreasing order;
    return J;
```

正确性证明 （反证法）

假设存在一个时间表$S' \neq S$，且执行S'的完成时间小于S。那么在S'中必然存在J_m, J_n，J_n紧接着J_m执行，并且有$f_m < f_n$。交换J_m, J_n，因为两个任务的总超级电脑处理时间和保持不变，对其他所有任务的完成时间没有任何影响。而因为$f_n > f_m$，交换后，J_m的完成时间要早于交换前J_n的完成时间。所以这种交换不会增加总完成时间。对S'所有逆序对进行上述变换，最终可以不增加总完成时间而得到S。这与假设相矛盾，所以不存在比S完成时间更短的时间表了。算法正确。

复杂度分析

该算法的时间复杂度为$O(nlogn)$。因为排序的复杂度为$O(nlogn)$。

3 Huffman code

问题描述

Write a program in your favorate language to compress a file using Huffman code and then decompress it. Code information may be contained in the com- pressed file if you can. Use your program to compress the two files (*graph.txt* and *AesopFables.txt*) and compare the results (Huffman code and compression ratio).

代码实现（Python）

```
#!/usr/bin/python
# Huffman-code Encoder
# Author: HongXin
# 2016.10.28

import json
import sys


class Encoder():
    def __init__(self):
        self.huff = Huffman()

    def encode(self, source, target):
        print 'Encoding ...'
        fs = open(source, 'rb')
        ft = open(target, 'wb')
        plain = fs.read()
        (bits_len, fq, cypher) = self.huff.encode(plain)
        ft.write(self.link(bits_len, json.dumps(fq), cypher))
        print 'Rate: {0:.2f}%'.format(float(len(cypher)) / len(plain) * 100)
        fs.close()
        ft.close()

    def decode(self, source, target):
        print 'Decoding ...'
        fs = open(source, 'rb')
        ft = open(target, 'wb')
        (bits_len_str, fq_str, cypher) = self.split(fs.read())
        plain = self.huff.decode(int(bits_len_str), json.loads(fq_str), cypher)
        ft.write(plain)
        print 'Done'
```

```python
        fs.close()
        ft.close()

    @staticmethod
    def link(bits_len, fq, cypher):
        return str(bits_len) + ' ' + fq + '\n' + cypher

    @staticmethod
    def split(mix):  # split encoded message into three part
        x = mix.find(' ')
        y = mix.find('\n')
        return mix[0:x], mix[x + 1:y], mix[y + 1:]


class Node:
    def __init__(self, l, fq=0, left=None, right=None):
        self.l = l
        self.fq = fq
        self.right = right
        self.left = left


class Huffman:
    tree = None
    __nodes = {}
    code = {}

    def __init__(self):
        pass

    # Encode Part
    def encode(self, plain):
        fq = self.__count(plain)
        self.__fq2nodes(fq)
        self.__gen_tree()
        self.__gen_code(self.tree, '')
        print json.dumps(self.code, indent=4, sort_keys=True)
        bits_len, cypher = self.__compress(plain)
        return bits_len, fq, cypher

    @staticmethod
    def __count(plain):
        fq = {}
        for l in plain:
            fq[l] = fq.get(l, 0) + 1
        return fq

    def __fq2nodes(self, fq):
```

```python
        for key, value in fq.iteritems():
            self.__nodes[key] = Node(key, value)

    def __pop_min(self):
        return self.__nodes.pop(min(self.__nodes.values(),
                                    key=lambda x: x.fq).l)

    def __push_link(self, left, right):
        root = Node(left.l + right.l, left.fq + right.fq, left, right)
        self.__nodes[root.l] = root

    def __gen_tree(self):
        if len(self.__nodes) == 1:  # for one node tree
            self.tree = Node('', 0, self.__nodes.values()[0])
        else:
            while len(self.__nodes) > 1:
                self.__push_link(self.__pop_min(), self.__pop_min())
            self.tree = self.__nodes.values()[0]

    def __gen_code(self, root, c):
        if root.left is None:
            self.code[root.l] = c
        else:
            self.__gen_code(root.left, c + '0')
            self.__gen_code(root.right, c + '1')

    def __compress(self, plain):
        bits_tmp, cypher_tmp = [], []
        # compress according to huffman-code
        for byte in plain:
            bits_tmp.append(self.code[byte])
        bits = ''.join(bits_tmp)
        bits_len = len(bits)
        bits += (8 - bits_len % 8) * '0'    # fill '0' to the end
        # bits to bytes
        for x in range(len(bits))[0::8]:
            cypher_tmp.append(chr(int(bits[x:x + 8], 2)))
        cypher = ''.join(cypher_tmp)
        return bits_len, cypher

    # Decode part
    def decode(self, bits_len, fq, cypher):
        bits = self.__get_bits(cypher, bits_len)
        self.__fq2nodes(fq)
        self.__gen_tree()
        plain = self.__decompress(bits)
        return plain

```

```python
    @staticmethod
    def __get_bits(cypher, bits_len):
        bits = []
        for l in cypher:
            bits.append(('0' * 8 + bin(ord(l))[2:])[-8:])
        return ''.join(bits)[0:bits_len]

    def __decompress(self, bits):
        plain = []
        current = self.tree
        for bit in bits:
            if bit == '0':
                current = current.left
            elif bit == '1':
                current = current.right
            if current.left is None:
                plain.append(current.l)
                current = self.tree
        return ''.join(plain)

if __name__ == '__main__':
    args = sys.argv
    n = len(args)
    if n != 4:
        print 'Argument Error!'
    elif args[1] == 'encode':
        Encoder().encode(args[2], args[3])
    elif args[1] == 'decode':
        Encoder().decode(args[2], args[3])
```

结果比较

graph.txt

```
>20:29:56< bash3.2 hugh@3 ~> python huffman_simple.py encode graph.txt a.txt
Encoding ...
{
    "\n": "1101",
    " ": "111",
    "#": "0100001000",
    ",": "0100011011010",
    ".": "01000011010",
    "0": "01001",
    "1": "00",
    "2": "1100",
    "3": "1010",
```

```
    "4": "1001",
    "5": "0110",
    "6": "0111",
    "7": "0101",
    "8": "1011",
    "9": "1000",
    ":": "01000011011",
    ";": "010001101100",
    "E": "0100011011011",
    "F": "0100001100010",
    "S": "0100001100011",
    "T": "010001011010",
    "a": "01000110111",
    "c": "0100010111",
    "d": "010001010",
    "e": "01000111",
    "f": "0100001001",
    "g": "010000010",
    "h": "010000101",
    "i": "01000000",
    "l": "0100001110",
    "m": "0100000110",
    "n": "010001100",
    "o": "010001000",
    "p": "01000011001",
    "r": "0100001111",
    "s": "0100011010",
    "t": "010001001",
    "u": "0100000111",
    "v": "010001011011",
    "w": "01000101100",
    "y": "010000110000"
}
Rate: 44.41%
```

Aesop_Fables.txt

```
>20:30:03< bash3.2 hugh@3 ~> python huffman_simple.py encode Aesop_Fables.txt a
Encoding ...
{
    "\n": "101100",
    "\r": "101101",
    " ": "111",
    "!": "110100101111",
    "\"": "10011111",
    "'": "11010010101",
    "(": "110100101110111",
```

```
    ")": "110100101110110",
    ",": "011111",
    "-": "11010010100",
    ".": "0111101",
    "0": "100111010101101",
    "1": "0100010110010",
    "2": "0100010110000",
    "3": "0100010110011",
    "4": "1001110101010",
    "5": "0100010110001",
    "6": "11010010111010",
    "7": "10011101010111",
    "8": "110100101110001",
    "9": "110100101110000",
    ":": "11010010011",
    ";": "11010010010",
    "?": "10011101011",
    "A": "01111001",
    "B": "1001111010",
    "C": "1001110110",
    "D": "0100010111",
    "E": "011110001",
    "F": "010000110",
    "G": "11010010110",
    "H": "100111100",
    "I": "01000010",
    "J": "01000100000",
    "K": "100111101111",
    "L": "010000111",
    "M": "010001001",
    "N": "1101001000",
    "O": "011110000",
    "P": "1001110111",
    "Q": "1001110101011000",
    "R": "1001110100",
    "S": "100111000",
    "T": "0100000",
    "U": "100111101110",
    "V": "100111010100",
    "W": "100111001",
    "X": "11010010111001",
    "Y": "01000101101",
    "Z": "1001110101011001",
    "a": "1000",
    "b": "1101000",
    "c": "100100",
    "d": "11000",
    "e": "001",
```

```
    "f": "101111",
    "g": "100101",
    "h": "0101",
    "i": "0000",
    "j": "10011110110",
    "k": "11010011",
    "l": "01110",
    "m": "101110",
    "n": "0001",
    "o": "0110",
    "p": "010010",
    "q": "0100010001",
    "r": "11001",
    "s": "11011",
    "t": "1010",
    "u": "110101",
    "v": "0100011",
    "w": "100110",
    "x": "010001010",
    "y": "010011",
    "z": "01000100001"
}
Rate: 57.02%
```

*graph.txt*的压缩率为44.41%，其中数字较多，从编码上可以观察到数字的编码明显较短。

*Aesop_Fables.txt*的压缩率为57.02%，其中字母较多，从编码上可以观察到字母的编码明显较短。