Assignment 3 of Algorithm

Zuyao Chen 201728008629002

1. Problem 1

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

1. initialize $d[n] = \{d_1, d_2, \cdots, d_n\}$, $flag = True$
2. for $i = 1$ to n,
   - sort $d[n]$ in descending order, choose $d[0]$ to add edges, if $d[0] = 0$ break
   - $d[k] = d[k] - 1$, $k = 1, 2, \cdots, d[0]$, if $d[k] < 0$ then $flag = False$, stop
   - $d[0] = 0$
3. if $flag = True$, there exists such a graph;otherwise , no graph can meet the needs.

Correctness:
Time complexity:
using quick-sort costs $O(n \log n)$, thus the time complexity is $O(n^2 \log n)$.

2. Problem 2

sort f_i in descending order, the maximum has high priority on supercomputer.

3. Problem 3–subsequence

a). **Description**
Goal: given two strings s and t, check whether s is subsequence of t.
Solution: let m, n be the length of s, t respectively and $j = 0$, then scan the string t from start to end using index i, $j = j + 1$ when $t[i] = s[j]$.Finally , if j is equivalent to the length of m,that means s is the subsequence of t;otherwise, s is not the subsequence of t.
pseudo-code:

Algorithm 1 Is subsequence

Input: two string s, t

Output: whether s is subsequence of t

```
function isSubsequence(s, t)
    initialize m = len(s), n = len(t), j = 0
    for i = 0 to n - 1 do
        if s[j] = t[i] then
            j = j + 1
        end if
    end for
    if j = m then
        return true
    else
        return false
    end if
end function
```

b). **Greedy choice property and optimal structure**

If s is the subsequence of t, then the substring of s should be the subsequence of t. In words, given $s = \{s_0s_1...s_{m-1}\}$, let $s' = \{s_0s_1...s_{j-1}\} \subset s$ be the subsequence of t, $t' = t - \{t_0t_1...t_k\}$, s' is the subsequence of substring $\{t_0t_1...t_k\}$ which has the minimum length. If s_j can be found in the remainder substring t', we add s_j to the subset s'. Finally, s is the subsequence of t if and only if $s' = s$.

the optimal structure can be written as

$$s' = \begin{cases} s' \cup \{s_j\}, \text{if } s_j \in t' \\ s', \text{otherwise} \end{cases}$$

c). **correctness**

suppose that we obtain a optimal substring $s' = \{s_0s_1...s_k\} \subset s$ using the method mentioned above, but there exists an another optimal substring $s'' \subset s$.

- If $\text{len}(s'') \leq \text{len}(s')$, s' is the final optimal substring.
- If $\text{len}(s'') > \text{len}(s')$, suppose that $s'' = \{s_0s_1...s_ks_{k+1}s_{k+2}...\} \subset s$.However, s' is the subsequence of $t - t'$ and $s_{k+1} \in t'$, according to the rule above, s_{k+1} should be added to s', the same condition is also applied to $s_{k+2}, s_{k+3}, ...$, which is contrary to the hypothesis. Thus s' should be equivalent to s''.

In summary, s' is the final optimal subsequence of t,Hence the algorithm is correct.

d). **complexity**

it costs $O(n)$ time.

4. Problem 4

a). **Description**

Goal: given two sets A and B, maximize $\prod_{i=1}^{n} {a_i}^{b_i}$.

Solution: We rearrange the order of A and B in descending order, then $\prod_{i=1}^{n} {a_i}^{b_i}$ is the maximum payoff.
pseudo-code:

Algorithm 2 Maximize payoff

```
1: function MAX-PAYOFF(A = {a_1, a_2, ..., a_n} , B = {b_1, b_2, ..., b_n})
2:     sort A, B in descending order
3:     return ∏_{i=1}^{n} a_i^{b_i}
4: end function
```

b). **Greedy choice property and optimal structure**
For simplicity , let $P_i = \prod_{j=1}^{i} {a_j}^{b_j}$, $A_i, B_i (i = 1, 2, ..., n)$ are the subsets of A, B respectively with the length of i.
If P_n^* is the optimal payoff of A_n and B_n, then P_{n-1}^* should be the optimal payoff of A_{n-1} and B_{n-1}, hence $P_1^* = a_1'^{b_1'}$ should be the optimal payoff of A_1 and B_1, P_1^* is given by $a_1' = \max(A_n), b_1' = \max(B_n)$, thus $P_2^* = P_1^* a_2'^{b_2'}$ is given by $a_2' = \max(A_n - \{a_1'\}), b_2' = \max(B_n - \{b_1'\})$,etc. So we just need to rearrange the two sets in descending order. the optimal structure is give by

$$P_i^* = \begin{cases} a_1'^{b_1'}, \text{if } i = 1 \\ P_{i-1}^* a_i'^{b_i'}, \text{otherwise} \end{cases}$$

c). **correctness**
Let $P = \prod_{i=1}^{n} {a_i}^{b_i}$ be the optimal payoff($a_1 > a_2 > ... > a_n, b_1 > b_2 > ... > b_n$). Suppose that there exists an another optimal solution P' in which a_1 is paired with b_q and a_q is paired with b_1,then

$$\begin{aligned} \frac{P'}{P} &= \frac{\prod_{i=1}^{n} a_i'^{b_i'}}{\prod_{i=1}^{n} {a_i}^{b_i}} \\ &= \frac{a_1^{b_q} a_q^{b_1}}{a_1^{b_1} a_q^{b_q}} \\ &= (\frac{a_1}{a_q})^{b_q - b_1} \end{aligned}$$

note that $a_1 > a_q, b_1 > b_q$, thus $P'/P < 1$; to change other pairs gives the same answer. Thus P is the optimal payoff.

d). **complexity**
If the two sets are already sorted, the time complexity is $O(n)$; otherwise, sort the sets first and the time complexity is $O(n \log n)$.

5. Problem 5 Huffman code

the result is showed below, the compression ratio is near 0.5.

```
joseph_chen@JosephChen-Desktop:~/Documents/algorithms_course/c_plus_plus/huffman_coding$ g++ -g main.cpp -o test -std=c++11
joseph_chen@JosephChen-Desktop:~/Documents/algorithms_course/c_plus_plus/huffman_coding$ ./test
compressing file...
read 2094720 bytes from file:graph.txt
write 930543 bytes to file:compress_graph.bin
compression ratio:0.444233
compressing file...
read 190066 bytes from file:Aesop_Fables.txt
write 108761 bytes to file:compress_Aesop_Fables.bin
compression ratio:0.572228
decompressing file ...
read 930543 bytes from file:compress_graph.bin
extract 2094720 bytes from :compress_graph.bin output file:recover_graph.txt
decompressing file ...
read 108761 bytes from file:compress_Aesop_Fables.bin
extract 190066 bytes from :compress_Aesop_Fables.bin output file:recover_Aesop_Fables.txt
```