Lab03-Greedy Strategy

Exercises for Algorithms by Xiaofeng Gao, 2018 Spring Semester

* If there is any problem, please contact TA Mingding Liao.

*

1. There are n field stacks and the mass of i-th field stack is m_i ($m_i > 0$). Bob wants to merge the n field stacks into only 1 stack. Every time Bob can merge one field stack with another and the merging cost is the mass of two stacks. For example, there are 3 field stacks, and their masses are $m_1 = 1$, $m_2 = 2$ and $m_3 = 4$. You can merge m_1 and m_2 with the cost of $1 + 2 = 3$, and then merge the new stack with m_3, where the cost is $3 + 4 = 7$, and the total cost is $3 + 7 = 10$.

   Given the number of field stacks n and their masses m_i, you should:

   (a) Design a greedy algorithm to merge all field stacks with the lowest cost;

   (b) Analyze the time complexity of your greedy algorithm;

   (c) Prove the correctness of your algorithm;

   (d) Complete the implementation in the provided C/C++ source code (The source code *Code-Greedy.c* and the input data file *input.txt* are compressed into *lab03-code.zip*, which is attached on the course webpage). You should show the executing result of your program by screenshot and upload your code along with the assignment. The executing time of your program should be not longer than 10 seconds.

Solution:
(a)

Algorithm 1: Greedy Algorithm For Merging Via Huffman Coding Technique

```
1  $INPUT : Array[1..n]$
2  minimum_Merging_Cost ← 0
3  minimum_Heap ← CREATE_Minimum_HEAP (Array[1..n]);
4  for ( $i \leftarrow 0$ ; minimum_Heap.SIZE()¿0 ; $i++$ ) do
5      $w_1$ ← minimum_Heap.POP();
6      $w_2$ ← minimum_Heap.POP();
7      minimum_Heap.PUSH($w_1 + w_2$);
8      minimum_Merging_Cost ← $w_1 + w_2$;
9  end
10 return minimum_Merging_Cost;
```

(b) Complexity is: Intial heap: O(n)Cost: O(log n)Time Complexity: (nlogn)

(c) Correctness of the Algorithm

This problem can be regarded as a Huffman tree as we need to sort and merge all the matching number values that we find, and then the weight of those values is regarded as frequencies. this scheme is the same as Huffman coding scheme. This Algorithm HUF(A,f) computes an optimal tree for weights f and numbers A.

*Note: For case of simplicity we can also consider f as frequency.

Proof : The proof is by induction on the size of alphabet. The induction hypothesis is that

for all A with |A|=n and for all frequencies f, HUF(A,f) computes the optimal tree.
In the base case (n=1), the tree is only one vertex and the cost is zero, which is the smallest possible.
For the general case, assume that the induction hypothesis holds for n-1. That is, $T^{'}$ is optimal for A' and f'.
ABL(T) = ($\sum_{\frac{x \in A}{w,y}}$ f(x)depth(x,T)) + f(w)depth(w,T)+ f(y)depth(y,T)
= ($\sum_{\frac{x \in A}{w,y}}$ f(x) depth(x,T))+ (f(w)+f(y)) (depth(z,$T^{'}$)+1)
= ($\sum_{\frac{x \in A}{w,y}}$ f(x) depth(x,T)) + $f^{'}$(z) depth(z,$T^{'}$) + f(w) + f(y)
=($\sum_{x \in A^{'}}$ $f^{'}(x)$ depth (x, $T^{'}$)) + f(w) +f(y)
= ABL($T^{'}$ + f(w)+ f(y))
Now, assume for the sake of contradiction that T is not optimal, and let Z be an optimal tree that has w and y as nodes (this exists by the above lemma). Let $Z^{'}$ be the tree obtained from Z by removing w and y. We can view $Z^{'}$ as a tree for the alphabet $A^{'}$ and frequency function $f^{'}$.
We can then repeat the calculation above and get
ABL(Z) = ABL($Z^{'}$) + f(w) + f(y)
So, ABL($T^{'}$) = ABL(T) - f(w) - f(y) > ABL(Z) -f(w) - f(y) = ABL($Z^{'}$).
Since $T^{'}$ is optimal for $A^{'}$ and $f^{'}$, this is a contradiction.

2. G be an undirected graph. A set of cycles $\{c_1, c_2, \ldots, c_k\}$ in G is called redundant if every edge in G appears in an even number of c_i's. A set of cycles is *independent* if it contains no redundant subset. A maximal independent set of cycles is called a *cycle basis* for G.

   (a) Let $\mathbf{C}$ be any cycle basis for G. Prove that for any cycle γ in G, there is a subset $A \subseteq \mathbf{C}$ such that $A \cup \{\gamma\}$ is redundant. In other words, γ is "exclusive" of the cycles in A.

   (b) Prove that the set of independent cycle sets form a matroid.

   (c) Now suppose each edge of G has a weight. Graph G contains n vertices and m circles. Define the weight of a cycle to be the total weight of its edges, and the weight of a set of cycles to be the total weight of all cycles in the set. (Thus, each edge is counted once for every cycle in which it appears.) Describe and analyze an efficient algorithm to compute the minimum-weight cycle basis in G.

Solution: (a)
The claim follows directly from the definitions. A cycle basis is a maximal independent set, so if C is a cycle basis, then for any cycle $\gamma \notin C$, the larger set C ∪ γ cannot be an independent set, so it must contain a redundant subset. On the other hand, if C is a basis, then C is independent, so C contains no redundant subsets. Thus, C ∪ γ must have a redundant subset B that contains γ. Let $A = \frac{B}{\gamma}$

(b) An independent system (C, γ) is a matroid if it satisfies the exchange property:
A, B ∈ C and |A| < |B| $\implies$ ∃ X ∈ $\frac{B}{A}$ such that A ∪ B ∈ C
Thus a matroid should satisfy two requirements: hereditary and exchange property.
(c) A greedy algorithm for finding a minimal cycle basis:
Step 1: Determine the size of the minimal cycle basis, denoted as k = |E| - |V|+1.
Step 2: For any edge e=(a,b) and vertex v find the cycle C_{ev} consisting of edge e and shortest path from v to a and b.
Step 4: Find all of the cycles. Sort all cycles by weights.
Step 5: Add cycles to the cycle basis one by one. Check if the added cycle is a combination of some cycles already existing in the basis. If yes, delete this cycle.

Step 6: Stop if the cycle basis has k cycles.
Analyzing : For v vertices and e edges would be nm cycles as step 4 indicates that adding cycles one by one to cycle basis by making sure that current cycle is not a combination of some cycle that already exists in basis indicates that this could be done by Guassian elmination on a binary matrix of edges and vertices as well. So by this we can analyze that the v vertices and e edges there would be ve cycles and Guassian elmination on a v*e matrix. a cost cycle will be O(v+e)/ This would make run-time complexity to $O((v+e)ve^3)$.