National Taiwan University | Handout #20
Department of Electrical Engineering | November 30, 2019
Algorithms, Fall 2019 | TA: Cheng-Yun Hsieh

# Sample Solutions to Homework #3

1. Exercise 16.2-5

   Algorithm:

   (1) Sort all the points to obtain a new array $\{x'_1, x'_2, \dots, x'_n\}$.

   (2) Create an empty solution set *S*.

   (3) Run greedy algorithm that traverses all the points from left to right. If we find a point $x'_i$ which is not contained in the last interval of *S*, we can add $[x'_i, x'_i + 1]$ to the end of solution set *S*. (put the interval as right as possible which still covers the point)

   Proof greedy choice property:

   Let *S* be an optimal solution. Suppose the leftmost interval of *S* is [*x*, *x* + 1]. Our greedy algorithm chooses $x$ to be $x'_1$. Let $S_0$ be the set obtained by S and replacing $[x, x+1]$ by $[x'_1, x'_1 + 1]$. We now argue that all points contained in $[x, x+1]$ are contained in $[x'_1, x'_1 + 1]$ because the region contained in $[x, x+1]$ but not $[x'_1, x'_1 + 1]$ is $[x, x'_1)$. Since $x'_1$ is the leftmost point, there are no points in the region $[x, x'_1)$. Hence $S_0$ is also an optimal solution.

   Proof optimal substructure property:

   Assume the original problem *P* has an optimal solution *S*. After including the interval $[x'_1, x'_1 + 1]$, the subproblem *P*' is to find a solution contains all of the points on the right of $x'_1 + 1$. Assume *S*' is the subset of *S* that solve *P*'. Since, size of (*S*) = size of (*S*') + 1, if *S*' is not an optimal solution, we can replace it by an optimal solution which reduces the size of *S*. This violates our assumption that *S* is an optimal solution. QED.

2. Exercise 16.3-3

   Huffman code for first 8 Fibonacci numbers:

| | |
|---|---|
| a | 1111111 |
| b | 1111110 |
| c | 111110 |
| d | 11110 |
| e | 1110 |
| f | 110 |
| g | 10 |
| h | 0 |

   Huffman code for first *n* Fibonacci numbers:

$$\begin{cases} \overbrace{1 \dots 1}^{n-1}, for\ 1^{st}\ number \\ \overbrace{1 \dots 1}^{n-i} 0, for\ i^{st}\ number\ where\ 2 \le i \le n \end{cases}$$

3. Modified Problem 16-1

(a) Always give the largest coin smaller than the remaining change. Repeat this process until no more remaining change.

(b) Denominations: 5, 4, 1, when we make change for 8 dollars, greedy choose will get {5, 1, 1, 1}, but the best solution is {4, 4}

(c)

```
Make-Change (n, d, k) //d is an array contain k denominations
1  let cnt[0..n] and pre[1..n] be new arrays
2  cnt[0] = 0
3  for i = 1 to n
4      cnt[i] = ∞
5  for j = 1 to k
6      for i = d[j] to n
7          if cnt[i] > cnt[i – d[j]]
8              cnt[i] = cnt[i – d[j]] + 1
9              pre[i] = i – d[j]
10 while n > 0
11     give one coin of size (n – pre[n]) as change
12     n = pre[n]
```

4. Card Game

Algorithm:

(1) Make an array A[1..$m$] to store the score of the card at the center of the lines. $O(m)$

(2) Let $ans$ = 0; $\Theta(1)$

(3) For each line of card L, add the left-most half ($L\left[1..\left\lfloor\frac{L.size}{2}\right\rfloor\right]$) of the line to $ans$.

If size of $i^{th}$ line is even, store 0 into A[$i$], else store $L\left[\left\lceil\frac{L.size}{2}\right\rceil\right]$ into A[$i$]. $\Theta(n)$

(4) Sort A by merge sort. $\Theta(m \log m)$

(5) Add the elements with odd index in A into $ans$. $\Theta(m)$

Justification:

For the line with even number of cards, any player can get the total score $\geq$ the sum of score near his side. Because both players get the optimal score, they can get the score equal to the sum of score near his side.

For any odd line, after one player picks a card, it becomes an even line. The card in the middle becomes the card near the player picking the first card. Hence, the problem becomes choosing the middle cards by turns.

5. Sum

   Algorithm:

   Start to calculate the sum from the last number. If the current sum $\leq 0$, we can remain the number unchanged. If the current sum $> 0$, we can turn the current number negative. At the end of the process, if $a_1$ is negative, we can inverse the sign of every number in the array.

   Complexity: Visit each element at most twice, so time complexity is Θ(n)

   Justification:

   We want to make $\text{abs}(\sum_{i=\text{j}\sim n} a_i) \leq \text{abs}(a_j)$ in every step.

   In the beginning, $\text{abs}(\sum_{i=\text{n}\sim n} a_i) = a_n$ meets the requirement.

   If $\text{abs}(\sum_{i=\text{j+1}\sim n} a_i) \leq \text{abs}(a_{j+1})$, by the algorithm described above, we get $\text{abs}(\sum_{i=\text{j}\sim n} a_i) \leq \text{abs}(\text{abs}(a_{j+1}) - a_j) \leq \text{abs}(a_j)$.

   Thus, the algorithm makes $\text{abs}(\sum_{i=1\sim n} a_i) \leq \text{abs}(a_1)$ until $a_1$.

   The last step of our algorithm makes sure that $a_1 > 0$, so that $\text{abs}(a_1) = a_1$ in a way that keeps the value of $\text{abs}(\sum_{i=1\sim n} a_i)$ the same.

6. Exercise 22.2-7

   Create a graph *G* for this problem. In *G*, every wrestler is represented by a vertex and each rivalry is represented by an edge. Run the DFS algorithm in the textbook on the graph. Randomly assign “babyfaces” and “heels” to the root of each DFS tree root, and assign each vertex as the inverse of its parent in the tree. After that, check each edge in *G*. If it connects two vertices of the same type, the answer does not exist. Or this algorithm gives the correct answer.

7. Exercise 22.3-12

```
DFS(G)
1   for each vertex u ∈ G.V
2   u.color = WHITE
3   u.π = NIL
4   time = 0
5   cc = 1
6   for each vertex u ∈ G.V
7       if u.color == WHITE
8           u.cc = cc
9           DFS-VISIT(G, u)
10          cc = cc + 1
DFS-VISIT(G, u)
1   time = time + 1
2   u.d = time
3   u.color = GRAY
4   for each vertex v ∈ G.Adj[u]
5       if v.color == WHITE
6           v.π = u
7           v.cc = u.cc
8           DFS-VISIT(G, v)
9   u.color = BLACK
10  time = time + 1
11  u.f = time
```

8. Exercise 22.4-4

   Consider the graph G with 4 vertices a, b, c, d and 5 edges *(a, b), (b, c), (b, d), (c, a), (d, a).* Suppose that we start from a, we may get the order: *a, b, c, d*. There are two "bad" edges *(c, a), (d, a)*. But the order: *b, c, d,* a has only one "bad" edge *(a, b).*

9. Simple Cycle in Graph

   Start to run DFS from one vertex. For each node, we record the depth of every node in the DFS tree. When we first reach the vertex, whose neighbors have been visited completely, we can choose a vertex($v$) with the smallest depth. The path in the DFS tree from $v$ to $u$ and the edge from $u$ to $v$ forms a cycle of size at least $n + 1$.

   Complexity: no more than DFS so the time complexity is O($n$)

   Justification:

   Before reaching first tree leaf of DFS, all the vertices are in the same path of the DFS tree. Hence,

each vertex has a different depth. The leaf edge has at least $n$ different back edges so the vertex $v$ with the smallest depth have at least $n$ level away from $u$. Hence, the size of cycle is at least $n + 1$.

10. BFS traversal

From the statement "in graph $G$ there is exactly one simple path between any two vertices" we know that $G$ is a tree, thus, $O(E) = O(V)$. Now, for each vertex, we can store the edges by adjacency list of the neighbor vertices. Sort all vertices in the adjacency lists in the same order as present in sequence $S$. After that, run the BFS algorithm from the first element in $S$ on the new graph, if the traversal order of BFS $S'$ equal to $S$, we can determine $S$ is valid, else $S$ is not valid.

Complexity:

Built the graph $\Theta(V)$, Sort all the lists: $O(V \log V)$, BFS $\Theta(V)$

Total $O(V \log V)$

Justification:

To justify that your algorithm is correct, you have to show that $S$ is a valid traversal sequence of BFS of $G$ if and only if (iff) your algorithm determined it is correct.

Since $S'$ is generated by BFS, it is definitely a vailed BFS traversal order. If $S$ equal to $S'$, the $S$ is also valid.

For the case that they are different. If assuming the first difference is at $k$ element, that means the $k$ element of $S$ can only be discovered by a node later than the vertex which discover the $k$ element of $S'$. This violates the assumption that $S$ is a valid BFS traversal sequence. So, $S$ and $S'$ are the same iff $S$ is a valid BFS traversal sequence of $G$.

Note that this is not an optimal algorithm for this problem. $\Theta(V)$ algorithm exists for this problem.