CS 6363: Computer Algorithms – Fall 2019
Homework #3 Solutions

Problem #1 (#15.5 – 1)

```
CONSTRUCT_OPTIMAL_BST(root[1 ... m])
    print k[root[1, m]] is the root.
    CONSTRUCT_OPT_SUBTREE(1, root[1, m] - 1, left, k[root[1, m]], root)
    CONSTRUCT_OPT_SUBTREE(root[1, m] + 1, m, right, k[root[1, m]], root)

CONSTRUCT_OPT_SUBTREE(i, j, childtype, parent, root)
    if j < i
        print d[j] is the childtype child of parent
    else
        print k[root[i, j]] is the childtype child of parent
        CONSTRUCT_OPT_SUBTREE(i, root[i, j] - 1, left, k[root[i, j]], root)
        CONSTRUCT_OPT_SUBTREE(root[i, j] + 1, j, right, k[root[i, j]], root)
```

Problem #2

Algorithm: ternary Hoffman

Input: a set of symbols C using ternary codewords $\{0,1,2\}$ with probability distribution

$$f : C \rightarrow \mathbb{R}^{+} \text{ s.t. } \sum_{c \in C} f(c) = 1$$

Output: Optimal ternary codes for C

Method

$Q \leftarrow C$

for $i \leftarrow 1$ **to** $\left\lfloor \frac{|C|}{2} \right\rfloor$ **do**

Create a new node w

Let x, y and z be 3 roots of 3 trees in Q with least probabilities

Assign x, y and z to the children of w

$f(w) \leftarrow f(x) + f(y) + f(z)$

if there are only two trees left at the last iteration

then combine them

Correctness

1. Let a, b and c be three sibling nodes in max depth of T and let x, y and z be the nodes with least probabilities. Without loss of generality, we assume $f(a) \leq f(b) \leq f(c)$ and $f(x) \leq f(y) \leq f(z)$. Then we have $f(x) \leq f(a)$, $f(y) \leq f(b)$ and $f(z) \leq f(c)$.
   We swap x and a and produce T'.
   We swap y and b and produce T''.
   We swap z and c and produce T'''.
   Then, $B(T) - B(T') = \sum_{c \in C} f(c) \cdot d_T(c) - \sum_{c \in C} f(c) \cdot d_{T'}(c)$

$$= f(a)d_T(a) + f(x)d_T(x) - f(a)d_{T'}(a) - f(x)d_{T'}$$
$$= f(a)d_T(a) + f(x)d_T(x) - f(a)d_T(x) - f(x)d_T(a)$$
$$= \big(f(a) - f(x)\big) \cdot \big(d_T(a) - d_T(x)\big) \geq 0$$

   Similarly, $B(T) - B(T'') \geq 0$ and $B(T'') - B(T''') \geq 0$.
   Thus, we have $B(T) \geq B(T''')$.
   But since T is optimal, we have $B(T) \leq B(T''')$. Therefore, $B(T) = B(T''')$.

2. Let T be the tree representing optimal prefix code for ternary codewords. We use same x, y and z in 1 and let u be the parent node of them. Then we have $f(u) = f(x) + f(y) + f(z)$ and $T' = T - \{x, y, z\}$. We claim that T' is the optimal prefix code for $C' = C - \{x, y, z\} \cup \{u\}$.
   proof: For every $c \in C - \{x, y, z\}$, we have $d_T(c) = d_{T'}$ and $f(c)d_T(c) = f(c)d_{T'}(c)$. Since $d_T(x) = d_T(y) = d_T(z) = d_{T'}(u) + 1$, $f(x)d_T(x) + f(y)d_T(y) + f(z)d_T(z) = \big(f(x) + f(y) + f(z)\big)(d_{T'}(u) + 1) = f(u)d_{T'}(u) + f(x) + f(y) + f(z)$.
   Thus, $B(T) = B(T') + f(x) + f(y) + f(z)$.
   If T' is not an optimal prefix code for C' then there exists another tree T'' s.t. $B(T'') \leq B(T')$ because $u \in C'$ and is a leaf of T''. If we add x, y and z as children of u in T'', then we obtain a prefix code for C with $B(T'') + f(x) + f(y) + f(z) < B(T)$. This contradicts the claim T is optimal. Thus, T' should be an optimal prefix code for C'.

From 1 and 2 above, the algorithm is correct.

Problem #3 – 1

Algorithm Coin_changes
Input n
Output the least number of coins
Method

$n_1 \leftarrow n$ div 25
$r_1 \leftarrow n$ mod 25
$n_2 \leftarrow r_1$ div 10
$r_2 \leftarrow r_1$ mod 10
$n_3 \leftarrow r_2$ div 5
$n_4 \leftarrow r_2$ mod 5
return $(n_1 + n_2 + n_3 + n_4)$

Correctness

Claim: There exists another set $\{n_1', n_2', n_3', n_4'\}$ to satisfy $25n_1' + 10n_2' + 5n_3' + n_4' = n$, $n_1' + n_2' + n_3' + n_4' < n_1 + n_2 + n_3 + n_4$. Here $n_1 \geq 0, 0 \leq n_2 \leq 2, 0 \leq n_3 \leq 1, 0 \leq n_4 \leq 4$. Assume that $n_1' \neq n_1$.

Case1: when $n_1' > n_1$ $(n_1' \geq n_1 + 1)$,
$25n_1' + 10n_2' + 5n_3' + n_4' \geq 25(n_1 + 1) + 10n_2' + 5n_3' + n_4 > 25n_1 + (10n_2 + 5n_3 + n_4) + 10n_2' + 5n_3' + n_4' > n$ ➔ Contradiction occurs.

Case2: when $n_1' < n_1$ $(n_1' \leq n_1 - 1$ and $10n_2' + 5n_3' + n_4' > 10n_2 + 5n_3 + n_4 + 25)$
$25n_1' + 10n_2' + 5n_3' + n_4 > 25(n_1 - 1) + (10n_2 + 5n_3 + n_4 + 25) = n$ ➔ Contradiction occurs.

By the same way, $n_1' = n_1, n_2' = n_2, n_3' = n_3, n_4' = n_4$.
Thus, the greedy algorithm is optimal.

Problem #3 – 2

Procedure Change_Generic(X)
begin

for $i \leftarrow k$ **downto** 0 **do begin**
$x_i \leftarrow X$ div c^i
$X \leftarrow X$ mod c^i
end

The result is optimal, because of the fact that if x_k is decreased by 1, then x_{k-1} needs to be increased by c or x_{k-2} by c^2, and so on. It would make $\sum x_i$ greater than it was.

Problem #3 – 3

Consider a set of coins $\{10, 6, 1\}$, and assume that we want to change of 12 cents. By the greedy algorithm, the change will be one 10¢ and two 1¢'s. But the optimal is two 6¢'s.

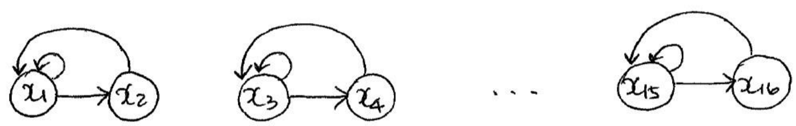
Problem #4 (#16.4 – 5)

Given a weight function $w(A) = \sum_{x \in A} w(x)$, GREEDY$\big((S, I), w\big)$ returns A in I of maximal weight. If we want to find a set A with minimum-weight maximal independent subset, we use GREEDY$\big((S, I), w'\big)$ with a modified weight function $w'(A) = m - w(x)$, where m is a real number such that $m > \max_{s \in S} w(s)$.

Problem #5 (#21.2 – 2)

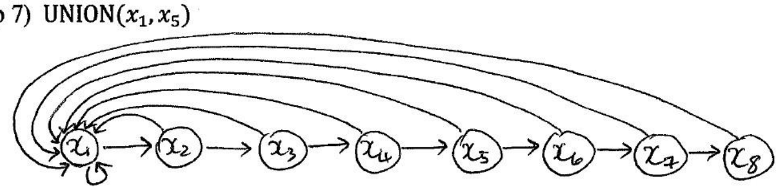
(Step 1 – 2) MAKE_SET(x_i)

(Step 3 – 4) UNION(x_i, x_{i+1})

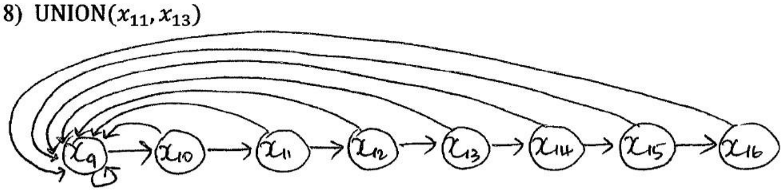


(Step 5 – 6) UNION(x_i, x_{i+2})

(Step 7) UNION(x_1, x_5)



(Step 8) UNION(x_{11}, x_{13})



(Step 9) UNION(x_1, x_{10})

(Step 10) FIND_SET(x_2)

Print x_1

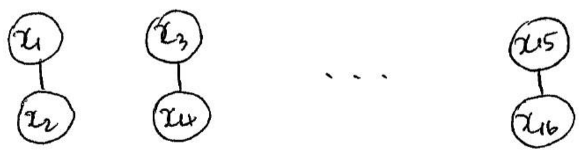
(Step 11) FIND_SET(x_9)

Print x_1

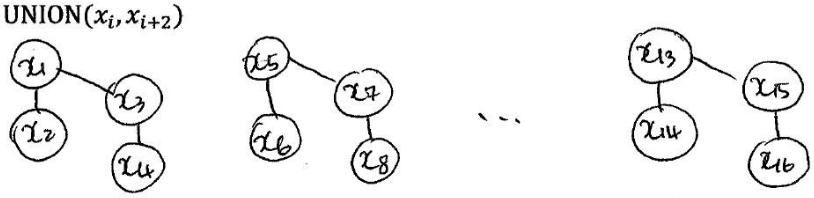
Problem #5 (#21.3 – 1)

(Step 1 – 2) MAKE_SET(x_i)

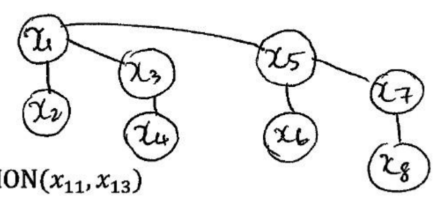
(Step 3 – 4) UNION(x_i, x_{i+1})



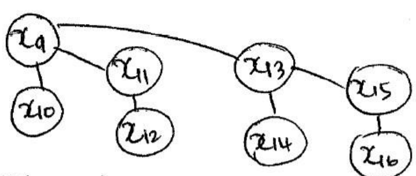
(Step 5 – 6) UNION(x_i, x_{i+2})



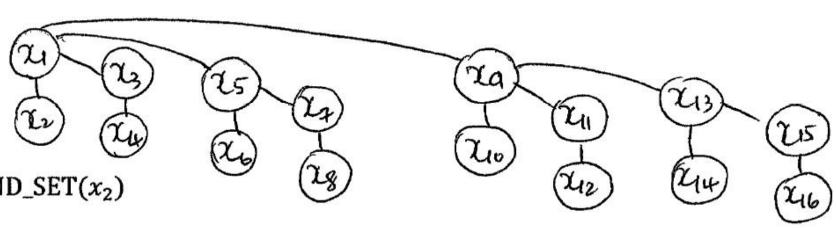
(Step 7) UNION(x_1, x_5)



(Step 8) UNION(x_{11}, x_{13})



(Step 9) UNION(x_1, x_{10})



(Step 10) FIND_SET(x_2)

Print x_1

(Step 11) FIND_SET(x_9)

Print x_1

Problem #6

Claim:

$$\delta(q,a) = \begin{cases} \delta(\pi[q], a) & \text{if } P[q+1] \neq a \text{ or } q = m \\ q+1 & \text{otherwise} \end{cases}$$

From the claim, we have:

procedure $\mathbf{M}(\boldsymbol{P}[\mathbf{1} \dots \boldsymbol{m}], \boldsymbol{\Sigma})$
$\pi =$ COMPUTE_PREFIX_FUNCTION(P, m)
for $q \leftarrow 0$ to m **do**
  for each $a \in \Sigma$ **do**
    if $P[q+1] \neq a$ or $q = m$ **then**
      $\delta(q,a) \leftarrow \delta(\pi[q], a)$
    else $\delta(q,a) \leftarrow q+1$

The complexity of above procedure is $O(m|\Sigma|)$

Proof of the claim:

If $P[q+1] = a$ then it is obvious that $\delta(q,a) = q+1$,
otherwise $\delta(q,a) = \sigma(P[1 \dots q]a) = \text{Max}\{k | P[1 \dots k] \supseteq P[1 \dots q]a\}$.
$\pi[q]$ is the longest prefix of q that matches the suffix of q.
$\sigma(P[1 \dots q]a) = \sigma(P[1 \dots \pi(q)]a) = \delta(\pi(q), a)$.

Problem #7 (#32.4 – 7)

Let T'' be the concatenation of the string T and T, itself.
Use the algorithm **KMP_matcher** (in the lecture note page 4.23).

$$\boxed{\text{KMP\_matcher}(T'', T''.\text{length}, T', T'.\text{length})}$$

If the algorithm returns "valid shift", then T is a cycle rotation of T'.