ADS Practice Sheet Solution

HT, IK, KS, FD

April 2020

1 STL Containers

1.1 Statement

Select the corrrect answer for the following question:
Which list is the longest list with containers in which you can you use the insert method?

(1) vector, set, map

(2) vector, list, deque

(3) set, multiset, map, multimap

(4) vector, list, deque, set, multiset, map, multimap

(5) None of the above

1.2 Solution

(4) vector, list, deque, set, multiset, map, multimap

Check http://www.cplusplus.com/reference/stl/ for more information on different types of containers (click on each container to find the respective member functions).

2 Templates I

2.1 Solution

The output will be:
2
1

3 Templates II

```
#include <iostream>
template<class T>
void multiples(T &sum, T x, int n){
        sum = 1.0;
        for (int i=1; i <= n; i++){
                sum += x*i;
        }
}

int main(){
        int su = 1; int ex = 5; int m = 2;
        multiples(su, ex, m);
        double su1 = 0; double ex1 = 5.5; int m1 = 3;
        multiples(su, ex, m);
        multiples(su1, ex1, m1);
        std::cout<<su<<std::endl;
        std::cout<<su1<<std::endl;
return 0;
}
```

4 Search in Data Structures

Search Times:

- heap (meaning it is either max or min) $O(n)$
- max heap $O(n)$
- queue $O(n)$
- BST $O(\log n)$
- RBT $O(\log n)$

Therefore, all data structures can perform search in better than $O(n \log n)$.

You may find it useful to try to check complexities here: `bigocheatsheet.com/`

5 Asymptotic Time Complexity

Using Master Theorem. $T(n) = 3T(n/2) + \Theta(n)$. Hence, $a = 3, b = 2, f(n) = \Theta(n)$. Hence, $log_b a = log_2 3 > 1$. Hence, $f(n) = n^{\log_2 3-\epsilon} = n = \Theta(n)$. Hence, Case 1 in Master Theorem. Hence, $T(n) = \Theta(n^{\log_2 3})$. Hence, all options are false.

You may find it useful to try to solve problems from here: `csd.uwo.ca/`

6 Fast Multiplication

6.1 Statement

Recall the divide and conquer formula of fast exponentiation of a number a $\in \mathbb{R}$ to a power n $\in \mathbb{N}$:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{n is even} \\ a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor} \cdot a & \text{n is odd} \end{cases}$$

1. Give a similar formula for a divide and conquer approach to the fast multiplication problem of a $\in \mathbb{R}$ and n $\in \mathbb{N}$.
2. Write a pseudocode implementation of a function that uses the derived formula to multiply a and n.
3. Derive and prove the asymptotic time complexity (upper and lower bound) of your algorithm.

6.2 Solution

$$a \cdot n = \begin{cases} a \cdot \frac{n}{2} + a \cdot \frac{n}{2} & \text{n is even} \\ a \cdot \lfloor \frac{n}{2} \rfloor + a \cdot \lfloor \frac{n}{2} \rfloor + a & \text{n is odd} \end{cases}$$

```
/* Assume $0^0 = 1$                                          */

function Fast-Multiplication($a$, $n$):
    if $n = 0$ then
        return 1
    else
        temp ← Fast-Multiplication($a$, $n$ div 2)
        if $n$ mod 2 = 0 then
            return temp + temp
        else
            return temp + temp + a
        end
    end
```

The recurrence formula of this algorithm is:

$$T(n) = \begin{cases} T(n/2) + \Theta(1) & n > 0 \\ \Theta(1) & n = 0 \end{cases}$$

Trying to apply master method one can see that $a = 1$, $b = 2$ and $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$. Since $f(n) = \Theta(1) = \Theta(n^{\log_b a})$ the second case of the master theorem is applicable $\Rightarrow T(n) = \Theta(log(n))$.

7 Binary search tree

7.1 Statement

Study the given code. The goal of the transform function is to convert a binary search tree into a new one that meets the following requirements:

- It is still a binary search tree
- Every vertex has only one child

- If a vertex is the left child of it's parent it can't have a left child. If it is a right child it can't have a right child.
- It contains the exact same nodes as the initial tree

Implement the transform function using the other given functions. The root of the resulting tree should be returned through the given parameter.

```
struct node{
    int data;

    node *left;
    node *right;
    node *parent;
};

void replace(node **root, node *x, node *y){
    if(x == *root)
        *root = y;
    else {
        if(y != NULL)
            y->parent = x->parent;

        if(x->parent->left == x)
            x->parent->left = y;
        else
            x->parent->right = y;
    }
}

node* find_min(node **root){
    node *min    = NULL;
    node *tmp    = *root;

    while(tmp != NULL)
        min = tmp,
        tmp = tmp->left;

    return min;
}

node* extract_min(node **root){
    node *min = find_min(root);

    if(min == NULL)
        return NULL;

    replace(root, min, min->right);
    min->right = NULL;
    return min;
}

node* find_max(node **root){
    node *max   = NULL;
    node *tmp   = *root;

    while(tmp != NULL)
        max = tmp,
        tmp = tmp->right;

    return max;
}

node* extract_max(node **root){
    node *max = find_max(root);

    if(max == NULL)
        return NULL;

    replace(root, max, max->left);
    max->left = NULL;

    return max;
}

void transform(node **root);
```

7.2 Solution

```
void transform(node **root){
    node *new_root = extract_min(root);
    node *tmp      = new_root;

    for(bool left = false; *root != NULL; left = !left){
        if(left){
            tmp->left              = extract_min(root),
            tmp->left->parent      = tmp,
            tmp                    = tmp->left;
        }
        else
            tmp->right             = extract_max(root),
            tmp->right->parent     = tmp,
            tmp                    = tmp->right;
    }
    *root = new_root;
}
```

8 Quicksort

8.1 Statement

Write down the complete pseudocode for randomized Quicksort including functions which are required for partitioning.

8.2 Solution

```
function Partition($A$, $p$, $q$):
    $x = A[p]$;
    $i = p$;
    for $j = p + 1$ to $q$ do
        if $A[j] <= x$ then
            $i = i + 1$;
            exchange $A[i]$ and $A[j]$;
        end
    end
    exchange $A[p]$ and $A[i]$;
    return $i$;

function Randomized-Partition($A$, $p$, $r$):
    $i = Random(p, r)$;
    exchange $A[p]$ with $A[i]$;
    return Partition($A$, $p$, $r$);

function Randomized-Quicksort($A$, $p$, $r$):
    if $p < r$ then
        $q$ = Randomized-Partition($A$, $p$, $r$);
        Randomized-Quicksort($A$, $p$, $q - 1$);
        Randomized-Quicksort($A$, $q + 1$, $r$);
    end
```

9 Red Black Trees

LEVEL 0: 27 BLACK (Root)

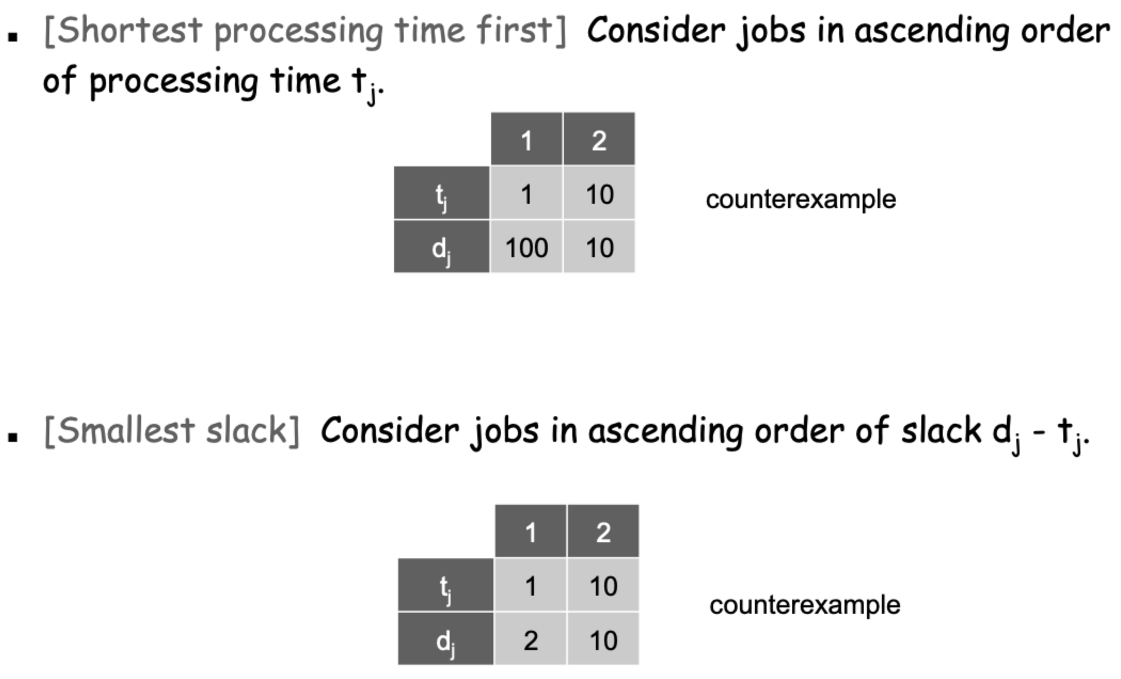
LEVEL 1: 20 RED (Parent: 27) (Children: 18,21) — 30 BLACK (Parent: 27) (Children: 33)

LEVEL 2: 18 BLACK (Parent: 20) — 21 BLACK (Parent: 20) (Children: 25) — 33 RED (Parent: 30)

LEVEL 3: 25 RED (Parent: 21)

10 Greedy Algorithm

a)



- [Shortest processing time first] **Consider jobs in ascending order of processing time t_j.**

	1	2
t_j	1	10
d_j	100	10

counterexample

- [Smallest slack] **Consider jobs in ascending order of slack $d_j - t_j$.**

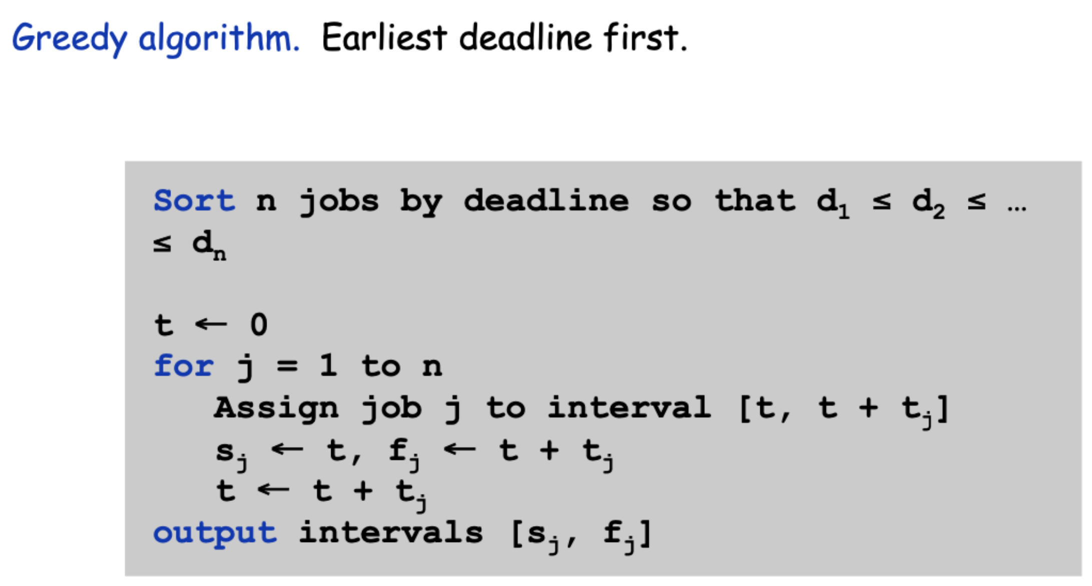
	1	2
t_j	1	10
d_j	2	10

counterexample

Figure 1: 3a solution

- Shortest processing time first: the generated solution [1,2] gives a max lateness of 1; whereas the optimal solution [2,1] gives a max lateness of 0.
- Smallest slack: the generated solution [2,1] gives a max lateness of 9; whereas the optimal solution [1,2] gives a max lateness of 1.

b)



Greedy algorithm. Earliest deadline first.

```
Sort n jobs by deadline so that $d_1 \le d_2 \le \dots$
$\le d_n$

t ← 0
for j = 1 to n
   Assign job j to interval [t, t + $t_j$]
   $s_j$ ← t, $f_j$ ← t + $t_j$
   t ← t + $t_j$
output intervals [$s_j$, $f_j$]
```

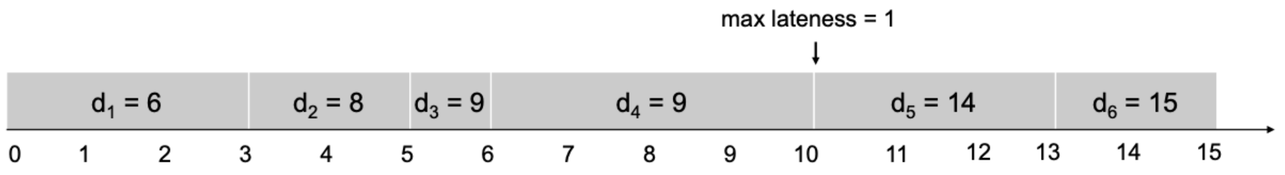


Figure 2: 3b solution

11 Graph Algorithms

STEP 0:
The color of all nodes is white.
dist(node 0) = 0
QUEUE $\{0\}$
STEP 1:
QUEUE $\{\}$
Node 0 becomes gray.
The neighbouring nodes of node 0 are node 1 and node 2. Both are white.
dist(node 1) = dist(node 0)+1 = 1
dist(node 2) = dist(node 0)+1 = 1
parent(node 1) = node 0
parent(node 2) = node 0
QUEUE $\{1, 2\}$
Node 0 becomes black.
STEP 2:
QUEUE $\{2\}$
Node 1 becomes gray.

The neighbouring nodes of node 1 are node 2 and node 3. Only node 3 is white.
dist(node 3) = dist(node 1)+1 = 2
parent(node 3) = node 1
QUEUE $\{2, 3\}$
Node 1 becomes black
STEP 3:
QUEUE $\{3\}$
Node 2 becomes gray.
The neighbouring node of node 2 is node 3. It is not white.
QUEUE $\{3\}$
Node 2 becomes black
STEP 4:
QUEUE $\{\}$
Node 3 becomes gray.
The neighbouring node of node 3 is node 4. It is white.
dist(node 4) = dist(node 3)+1 = 3
parent(node 4) = node 3
QUEUE $\{4\}$
Node 3 becomes black
STEP 5:
QUEUE $\{\}$
Node 4 becomes gray.
The neighbouring nodes of node 4 are node 0, node 1 and node 5. Only node 5 is white.
dist(node 5) = dist(node 4)+1 = 4
parent(node 5) = node 4
QUEUE $\{5\}$
Node 4 becomes black
STEP 6:
5 becomes black automatically since it has no neighbours.