CLRS Exercise

Tongda Xu

November 11, 2018

1 7

1.1 7.3

1.1.1 a

This is certain concerning the *Randomized* procedure, the probability of any index i is chosen from $[0, n-1]$is:
$Pr(pivot = i) = \frac{1}{n}$
$E(X_i) = 1 * Pr(pivot = i) + 0 * Pr(pivot \neq i) = \frac{1}{n}$

1.1.2 b

It is certain that if ith element is chosen as pivot, $Random-Parition$ cost $\Theta(n)$ time, and it will call $QuickSort[1, q-1], QuickSort[q+1, n]$ recursively.
Concerning only the first $Parition$, this would be the result:
$E(T(n)) = \Sigma_{i=1}^{n} Pr(pivot = i)(T(i-1) + T(n-i) + \Theta(n))$
$= \Sigma_{i=1}^{n} X_i(T(i-1) + T(n-i) + \Theta(n))$

1.1.3 c

Concerning $X_i = \frac{1}{n}$
$E(T(n)) = \Sigma_{i=1}^{n} \frac{1}{n}(T(i-1) + T(n-i) + \Theta(n))$
$= \Sigma_{i=1}^{n} \frac{1}{n} T(i-1) + \Sigma_{i=1}^{n} \frac{1}{n} T(n-i) + \Sigma_{i=1}^{n} \frac{1}{n} \Theta(n)$
$= \frac{2}{n} \Sigma_{i=1}^{n-1} T(i) + \Theta(n)$

1.1.4 d

$\Sigma_{k=2}^{n-1} klgk$
$\leq lg\frac{n}{2} \Sigma_{k=2}^{\frac{n}{2}} k + lgn \Sigma_{k=\frac{n}{2}}^{n-1} k$
$= lgn \Sigma_{k=2}^{n-1} k - lg2 \Sigma_{k=2}^{\frac{n}{2}} k$
$= lgn \frac{(n+1)(n-2)}{2} - \frac{(\frac{n}{2}+2)(\frac{n}{2}-1)}{2}$
$\leq lgn \frac{n^2}{2} - \frac{n^2}{8}$
by Calculus, we have:
$(\frac{1}{2}x^2 lgx - \frac{1}{4}x^2)|_1^{n-1} \leq E(T(n)) \leq (\frac{1}{2}x^2 lgx - \frac{1}{4}x^2)|_2^{n}$

1.1.5 e

Proof of $E(T(n)) = O(nlgn)$:
Assume that $\forall k \in [1, n-1], \exists c, E(T(k)) \leq cklgk - \Theta(k)$
For $k = n, E(T(n)) \leq \frac{n}{2}c(lgn\frac{n^2}{2} - \frac{n^2}{4} - \Theta(n^2)) + \Theta(n) \leq cnlgn - \Theta(n)$
Proof of $E(T(n)) = \Omega(nlgn)$:
Assume that $\forall k \in [1, n-1], \exists c, E(T(k)) \geq cklgk + \Theta(k)$
For $k = n, E(T(n)) \geq \frac{n}{2}c(lgn\frac{(n-1)^2}{2} - \frac{(n-1)^2}{4} + \Theta(n^2)) + \Theta(n) \geq cnlgn + \Theta(n)$
$\rightarrow E(T(n)) = \Theta(nlgn)$

1.2 7.5

1.2.1 a

From counting Theorem, it could be noticed that:
$p_i = \frac{(i-1)(n-i)}{C_n^3} = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$

1.2.2 b

$Pr(i = medium)(normal) = \frac{1}{n}$
$Pr(i = medium)(3part) = \frac{6(\frac{1}{2}n-1)(n-\frac{1}{2}n)}{n(n-1)(n-2)} = \frac{3}{2}\frac{1}{n}$
$Pr(3part) - Pr(normal) = \frac{1}{2}\frac{1}{n}$

1.2.3 c

Consider $f_{diff} = \int_{\frac{n}{3}}^{\frac{2}{3}n}(\frac{6(i-1)(n-i)}{n(n-1)(n-2)} - \frac{1}{n})di$
$= \frac{(-2i^3+3(n+1)i^2-6ni-(n-1)(n-2)i)|_{i=\frac{1}{3}n}^{i=\frac{2}{3}n}}{n(n-1)(n-2)}$
$lim_{n\rightarrow\infty} f_{diff} = \frac{4}{27}$

1.2.4 d

Consider we are so lucky that each partition we choose the median:
In the Iteration tree, we have:
$T(n) = \begin{cases} c & n = 1 \\ 2T(\frac{1}{2}n) + n & n > 1 \end{cases}$
The $\Omega(nlgn)$ is kept even in best case.

2 8

2.1 8.1-1

n-1 times, since we need n elements to formulate

2.2 8.1-2

$\Sigma_1^n lgk < \int_1^{n+1} lgkdk = (klgk - k)_1^n = (nlgn - n) - (0 - 1) = nlgn - n + 1$

2.3 8.1-3

$\leftrightarrow$ proof at least half of branch is longer than h
Consider a decision tree with $n!/2$ elements
$\leftrightarrow$ proof at least half of branch is longer than h
Consider a decision tree with $n!/n$ elements
$\leftrightarrow$ proof at least half of branch is longer than h
Consider a decision tree with $n!/2^n$ elements, this is not significant enough and could leave only $\Omega(lg\frac{n!}{2^n}) = \Omega(nlgn - n) = \Omega(nlgn)$ elements

2.4 8.2-4

Consider a trim version of counting sort, build the C map up and query directly:

COUNTING-SORT-TRIM(A, k)

$C[]$
for $i = 0$ **to** k
    $C[i] = 0$
for $j = 1$ **to** $A.length$
    $C[A[j]] + +$
for $m = 1$ **to** k
    $C[m] + = C[m - 1]$
return $C[m]$

DIRECT-QUERT(A, k, a, b)

$C =$ COUNTING-SORT-TRIM(A, k)
if $a < 1$
    return $C[b]$
else return $C[b] - C[a - 1]$

2.5 8.3-2

Heapsort is not stable
The scheme would be very similar to counting sort and takes $\Theta(n)$ time

2.6 8.3-4

First, with $O(n)$ time: convert n numbers k_{10} into k_n which has 3 digits.
Second, with $O(d(n+n))$ time ($Lemma 8.3$): Radix sort n 3-digit numbers with each digits take up to n possible values.

```
digitsConvert(X)
result[]
for i = 2 downto 0
    result[i] = X/n^i
    X = X mod n^i
return result
```

```
sort(A, x)
result[]
for each S in A
    S = digitsConvert(S)
radix-sort(A, x)
```

3 9

3.1 9.2-1

once $p == r$, the function return and recursion end.

3.2 9.2-2

It is because $\forall k, X_k = \frac{1}{n}$, giving information on which k would not effect observation

3.3 9.2-3

```
randomized-select-iter(A, p, r, i)
while 1
    if i == k
        return A[i]
    else
        q = random-partition(A, p, r)
        if i < k
            r = q - 1
        else p = q + 1, i = i - k
```

3.4 9.2-4

The worst case is reverse side:
$pivot = 9, 8, 7, 6, 5, 4, 3, 2, 1, 0$

3.5 9.1

3.5.1 a

Sorting: Merge-sort(A) in worst case $O(nlgn)$
Query: call-by-rank(A, k) i times in worst case $O(i)$, here we assume manip-

ulating $O(n)$ space cost $O(n)$ time.

3.5.2 b

Building: BUILD-MAP-HEAP(A) in worst case $O(n)$
Query: calling EXTRA-MAX(A, k) i times in worst case $O(ilgn)$

3.5.3 c

Selecting: SELECT(A, i) in worst case $O(n)$
Sorting: MERGE-SORT(A') in worst case $O(ilgi)$

3.6 9.2

3.6.1 a

$\Sigma_1^{k-1} w_i = \Sigma_1^{k-1} \frac{1}{n} = \frac{k-1}{n} < \frac{1}{2}$
$\Sigma_{k+1}^{n} = \frac{n-k}{n} \leq \frac{1}{2}$

3.6.2 b

WEIGHT-MEDIAN(A)

w[] = SORT(A).weight
n = w.length
for $i = 1$ **to** n
    $w[i] = w[i] + w[i-1]$
return FIND$(w[], \frac{1}{2})$

3.6.3 c

SUM$(w_1, w_i, lasti, lastsum)$

if $i > lasti$
    return $lastsum +$ NORMAL-SUM($w_{lasti,i}$)
else return $lastsum -$ NORMAL-SUM($w_{i,lasti}$)

WEIGHT-MEDIAN-LINEAR(A)

while 1
    if $sum[w_1, w_i, lasti, lastsum] < \frac{1}{2}, sum[w_1, w_{i+1}, lasti, lastsum] > \frac{1}{2}$
        return i
    else
        $lastsum = sum[w_1, w_i, lasti, lastsum], lasti = i$
        if $sum[w_1, w_i] < \frac{1}{2}$
            $i =$ MEDIAN(A, i, r)
        else $i =$ MEDIAN(A, p, i)

We will experience *logn* literation, but the load is decreasing logarithmically, so the result is linear. Notice the sum is special here, calculating the difference only.

3.7 9.4

3.7.1 a

$k \leq i$ or $k \geq j : 0$
$i < k < j : \frac{2}{j-i+i}$

3.7.2 b

3.7.3 c

3.7.4 d

4 11

4.1 11.1-2

Consider $vector < bool > A, a.size() = m$, just store the bool value of $key = m$ exist or not.

SEARCH(A, key)
if $A(key)$
    return key
else return NIL

INSERT(A, key)
$A(key) = 1$

DELETE(A, key)
$A(key) = 0$

4.2 11.2

4.2.1 a

Consider for a ball i fall into a specific bucket $Pr(i) = \frac{1}{n}$
Then consider Binomial Distribution, $Pr(k) = C_n^k Pr(i)^k (1 - Pr(i))^{n-k}$

4.2.2 b

Consider random picking a slot, the probability of that slot is maximum is $Pr_{max} = \frac{1}{n}$, and it contains k elements Q_k. for conditional probability, we have:
$P_k = Pr_{i=k|max} = \frac{Pr(i=k \cap max)}{Pr_{max}} \leq \frac{Pr(i=k)}{Pr_{max}} = nQ_k$

4.2.3 c

Proof:
$Q_k = (\frac{1}{n})^k (\frac{n-1}{n})^{n-k} C_n^k$
$= \frac{(n-1)^{n-k}}{n^n} \frac{\Pi_0^{k-1} n-k}{k!}$
$\leq \frac{n^n}{n^n} \frac{1}{k!}$
$= \frac{e^k}{k^k} \frac{1}{k^{\frac{1}{2}}(1+\Theta(\frac{1}{n}))}$
$\leq \frac{e^k}{k^k}$

4.2.4 d

Proof for Q_{k_0}:
$Q_{k_0} = \frac{e^{(\frac{clgn}{lglgn})}}{(\frac{clgn}{lglgn})^{\frac{clgn}{lglgn}}}$
$= \frac{n^{\frac{clg\frac{e}{c}}{lglgn}}}{\frac{n^c}{n^{\frac{clglglgn}{lglgn}}}} = n^{\frac{clg\frac{e}{c}+clglglgn}{lglgn}-c}$
It would not take effort to notice that since $lim_{n\to\infty}\frac{clg\frac{e}{c}+clglglgn}{lglgn} = 0$
$\forall c > 3+\epsilon, Q_{k_0} = O(\frac{1}{n^3})$
And $P_k \leq nQ_k \to P_k = O(\frac{1}{n^2})$

4.2.5 e

$E(M) = \Sigma_{M=1}^n MPr(M) < nPr(M > \frac{clgn}{lglgn}) + \frac{clgn}{lglgn}Pr(M \leq \frac{clgn}{lglgn})$
A stronger conclusion to note:
$E(M) = \Sigma_{M=1}^n MPr(M) < MPr(M > \frac{clgn}{lglgn}) + \frac{clgn}{lglgn}Pr(M \leq \frac{clgn}{lglgn})$
$\leq \int_{\frac{clgn}{lglgn}}^{\infty} \frac{1}{n}dn + 1 * \frac{clgn}{lglgn}$
$= lg(\frac{clgn}{lglgn}) + \frac{clgn}{lglgn}$
$= O(\frac{clgn}{lglgn})$

5 15

5.1 15.1-1

$2^n - 1 = \Sigma_{j=0}^{n-1} 2^j$

5.2 15.1-2

Do not know how!

5.3 15.1-3

```
BOTTOM-UP-CUT-ROD(p, n, c)
r[] = c
for j = 1 to n
    for i = 1 to j
        r[i] = max(p[i] + r[j - i] - c)
return r[n]
```

5.4 15.1-4

```
MEMOIZED-CUT-ROD(p, n, m, s)
if m[n] > -1
    return m[n]
else
    for i = 1 to n
        m[n] = max(p[i] + r[n - i])
        s[n] = i
    return m[n]
```

5.5 15.1-5

See Code

5.6 15.2-1

See Code

5.7 15.2-2

See Code

5.8 15.2-3

Assume that $\forall k \leq n-1, T(k) \geq c2^k$
Then $T(n) = \Sigma_{k=1}^{n-1} T(k)T(n-k) = (n-1)c^2 2^n > c2^n$
So $T(n) = \Omega(n), \omega(n)$

5.9 15.2-4

See Figure 1

5.10 15.2-5

For each level $h(i) = i(n-i)$
For tree $T(n) = 2\Sigma_{i=1}^{n-1} i(n-i)$

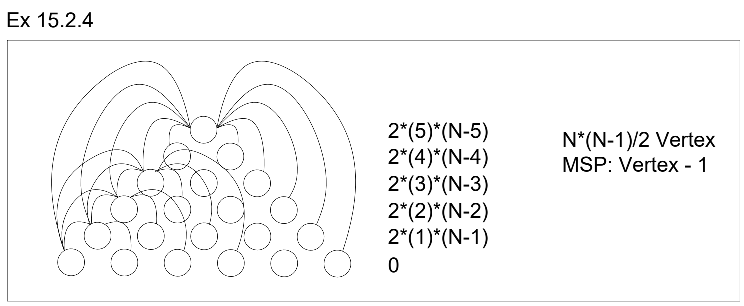

Figure 1: 15.2-4

$= \frac{3n^3+3n^2}{3} - \frac{2n^3+3n^2+n}{3}$
$= \frac{n^3-n}{3}$

5.11 15.2-6

Assume that $\forall k \leq n-1, N(k) = k-1$
Then $N(n) = N(n-1) + 1$
So $N(n) = n - 1$

5.12 15.3-1

running through: $T(n) = n * P_n^n = n * n! > 4^n$
running recursion: $T(n) = 2\Sigma_{i=1}^{n-1} 4^i + n = \frac{8}{3} 4^{n-1} + n \leq 4^n$
running through takes longer

5.13 15.3-2

no overlapping subproblem call

5.14 15.3-3

Yes

5.15 15.3-4

Do not know how!

5.16 15.4-1

See code

5.17 15.4-2

See code

5.18 15.4-3

See code

5.19 15.5-1

A Preorder Traverse of BST

PRE-ORDER-PRINT-AID($i, j, root$)

if $root[i, j] - 1 - i \geq 0$
    k $root[i, root[i, j] - 1]$ is the left child of k $root[i, j]$
    PRE-ORDER-PRINT-AID($i, root - 1, root[i, root[i, j] - 1]$)
else d $i - 1$ is the left child of k $root[i, j]$
if $j - root[i, j] - 1 \geq 0$
    k $root[root[i, j] + 1, j]$ is the right child of k $root$
    PRE-ORDER-PRINT-AID($root + 1, j, root[root[i, j] + 1, j]$)
else d $i - 1$ is the right child of $root$

PRE-ORDER-PRINT($root$)

k $root[1, n]$ is the root
PRE-ORDER-PRINT-AID($1, n, root$)

5.20 15.5-3

Asymptotically there would be no change to the running time, just the constant cn^3 increase
Time spent on w would increase from $\Theta(n^2)$ to $\Theta(n^3)$

5.21 15.1

LSP(s, t, G)

$r = G, size()$
$DPs[r] = 0$
$DPr[r] = path(s, t)$
$max = -\infty$
for $i = 1$ **to** r
    $max(DPs[j] + DPr[r - j] + what)$
return max

5.22 15.1

5.23 22.1-1

for both out-degree and in-degree $\Theta(V+E)$ time
both take $\Theta(V)$ memory

5.24 22.1-2

$1 \to 2 \to 3 \to NIL$
$2 \to 1 \to 4 \to 5 \to NIL$
$3 \to 1 \to 6 \to 7 \to NIL$
$4 \to 2 \to NIL$
$5 \to 2 \to NIL$
$6 \to 3 \to NIL$
$7 \to 3 \to NIL$
$0-1-1-0-0-0-0$
$1-0-0-1-1-0-0$
$1-0-0-0-0-1-1$
$0-1-0-0-0-0-0$
$0-1-0-0-0-0-0$
$0-0-1-0-0-0-0$
$0-0-1-0-0-0-0$

5.25 22.1-3

Transpose(*Adjlist*)

```
new AdjlistPrime
for each node in Adjlist
    for each subnode in Adjlist(node)
        AdjlistPrime(subnode).insert(node)
        Adjlist = AdjlistPrime
```

For adjacent list: just traverse every node and rebuild one
$\Theta(E+V)$ for time and space complexity, hard to do it inplace

Transpose(*Adjmatrix*)

```
for each pair(i, j) in upper left Adjmatrix
    Swap(Adjmatrix[i, j], Adjmatrix[j, i])
```

For adjacent matrix: just transpose the matrix
$\Theta(V^2)$ for time and $\Theta(1)$ for space

5.26 22.1-4

use an adjacent matrix as aid.

5.27 22.1-5

For adjacent list, it is hard. We should regard it as a BREADTH-FIRST-SEARCH(G) end at $d = 2$:

SQUARE(G)

for each u in $G.vertices$
    $G.reset()$
    $list = \emptyset$
    $u.adjlist^{'} =$ BFS-AID$(G, u, list, 0)$

BFS-AID$(G, u, list, dist)$

for each v in $u.adjlist$
    if $v.color = white$ and $dist \leq 2$
        $list.insert(u)$
        BFS-AID$(G, v, list, dist + 1) =$
return $list$

This could cost $\Theta(V^2 + VE)$ time and $\Theta(V + E)$ space (if optimized).

For adjacent matrix, the square process would be simple. for each index m of matrix row, if matrix[m][n] exist, calculate bool union of matrix[m] and matrix[n]:

SQUARE(G)

for each m in $G.adjMatrix$
    for each n $G.adjMatrix[m]$
        if $G.adjMatrix[m][n] == 1$
            $G^{'}.adjMatrix[m] =$ AND$(G.adjMatrix[m], G.adjMatrix[n])$
return $G^{'}$

The SQUARE(G) cost $\Theta(V^3)$ time and $\Theta(V)$ space (if optimize)

5.28 22.2-3

line 2 $\rightarrow u.ifgrey = 0$
line 5 $\rightarrow s.ifgrey = 1$
line 14 $\rightarrow v.ifgrey = 1$

5.29 22.2-4

take $\Theta(V^2)$ time and $\Theta(V^2)$ space, since we need to search every column to find adjacent list.
line 12 $\rightarrow$ **for** : $each \quad v \in M[u]$
line 13 $\rightarrow$ **if** : $v ==$ **true** **and** $v.color == white$

5.30 22.2-5

SQUARE($AdjList$)

1 **for** each u in $vertices$
2   **for** each v in $AdjList(u)$
3         $AdjList(u).append(AdjList(v))$

For adjacent list, for each vertex u, append the adjacent list of each adjacent vertex v to adjacent list of u.
$line3$ would be execute $\Theta(E)$ times in total,

SQUARE($Adjmatrix$)

1 **for** each $pair(i, j)$ in upper left Adjmatrix
2   SWAP($Adjmatrix[i, j], Adjmatrix[j, i]$)

5.31 Edge traverse of undirected graph

According to $Theorem22.10$, all edges are either tree edge or back edge. Modify the DFS-VISIT(G, u), add a PRINT-PATH(G, u) would do it. Assume a $root = u$ is selected:

DFS-VISIT(G, u)

1 $u.color = grey$
2 $dict[(Vertex, Vertex), edgeType] = \emptyset$
3 **for** each v in $u.adjList$
4   **if** $v.color == white$
5         $dict(u, v) = treeEdge$
6   DFS-VISIT(G, v)
7   **else** $dict(u, v) = backEgde$
8 PRINT-PATH(G, u)

PRINT-PATH(G, u)

1 PRINT $("u")$
2 **for** each v in $u.adjList$
3   **if** $(u, v) == treeedge$
4   PRINT $(" \rightarrow ")$
5   PRINT-PATH(G, v)
6   **else** PRINT $(" \rightarrow v")$

line 4,6 cost same level of time as the comparison in line 3, would not change the $\Theta(V + E)$ time complexity of DFS(G)
the print path function as:
This procedure cost $\Theta(V + E)$ as well

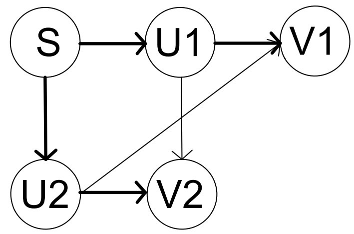

Figure 2: 22.2-6

5.32 22.2-6

Consider the following condition in Figure 2:
$E_\pi =< s, u1 >, < u1, v1 >, < s, u2 >, < u2, v2 >$
In BFS Tree, $\delta(s, v1), \delta(s, v2)$ is either $< s, u1, v1 >, < s, u1, v2 >$ or $< s, u2, v1 >$ $, < s, u2, v2 >$

5.33 22-3.12

Tweak the DFS-VISIT(G, u) and DFS(G) would be enough:

```
DFS(G)
for each u in G.V
    u.color = white
c = 1
for each u in G.V
    if u.color = white
        DFS-Visit(G, u, c)
        c++
```

```
DFS-Visit(G, u, c)
u.color = grey
u.cc = c
for each v in u.adjList
    if v.color == white
        DFS-Visit(G, v)
```

DFS(G) could be tweaked to do it as well

5.34 22.4-1

$p[27 : 28] \to n[21 : 26] \to o[22 : 25] \to s[23 : 24] \to$
$m[1 : 20] \to r[6 : 19] \to y[9 : 18] \to v[10 : 17] \to x[15 : 16] \to$
$w[11 : 14] \to z[12 : 13] \to u[7 : 8] \to q[2 : 5] \to t[3 : 4]$

5.35 22.4-3

A DFS(G)/BFS(G) returns false when a back edge is found, easy to proof it is $\Theta(V)$

5.36 22.1

5.36.1 a-1

Suppose (v, u) is a backedge. u is ancestor elder than parent of v. This means (s, u) + forwardEdge is shorter than (s, v) produced by BFS which is $\delta(s, v)$ by **Theorem 22.5**. Same reason for forward edge.

5.36.2 a-2

By **Theorem 22.5** $\delta(s, u) = u.d = u.level$ and $\delta(s, v) = v.d = vv.level$, so $v.d = u.d + 1$

5.36.3 a-3

Same as a-1, if $v.d > u.d+1$, $\delta(s, v) = (s, u)+cross$ instead of (s, v). if $v.d < u.d$

5.36.4 b-1

Same as a-1, the $(s, u) + backEdge$ would be shorter than (s, v)

5.36.5 b-2

By **Theorem 22.5** $\delta(s, u) = u.d = u.level$ and $\delta(s, v) = v.d = vv.level$, so $v.d = u.d + 1$

5.36.6 b-3

Still consider that BFS always generate shortest path. $\delta(s, u) + 1 < (s, v)$ is not allowed

5.36.7 b-4

By **Corollary 22.4** we know that $parenr.d \leq child.d$, so for backedge $v.d \leq u.d$