# EL9343 Homework 5

*All problem/exercise numbers are for third edition of CLRS text book*

---

*Reminder: If you have already submitted solutions for problems 1,2, you do not have to re-submit them for this homework.*

**1.** Exercise 22.4-1

**2.** Show how the procedure Strongly-Connected-Components works on the graph in Figure 1. Show the finishing times computed in line 1 and the forest produced in line 3. Assume DFS considers vertices in alphabetical order and and the adjacency lists are also alphabetical order.

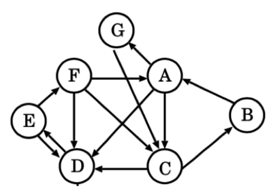



Figure 1: Directed Graph for Question 2

**3.** Given an $M \times N$ matrix $D$ and two coordinates $(a, b)$ and $(c, d)$ which represent top-left and bottom-right coordinates of a sub-matrix of the given matrix, propose a dynamic-programming approach to calculate the sum of all elements in the sub-matrix. What is the complexity of your solution?

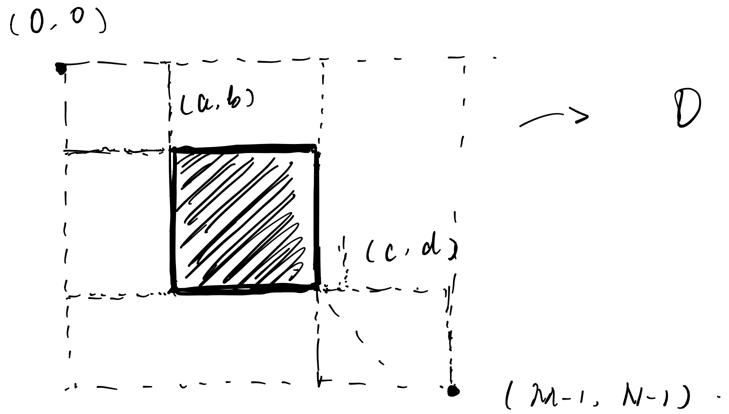
| | | | |
|---|---|---|---|
| 0 | -2 | -7 | 0 |
| 9 | 2 | -6 | 2 |
| -4 | 1 | -4 | 1 |
| -1 | 8 | 0 | -2 |

Figure 2: Example of a sub-matrix where $(a, b) = (1, 0)$ and $(c, d) = (4, 2)$

**4.** Propose a dynamic-programming approach to obtain the minimum number of coins required to get a desired change. Assume that you are given sufficiently many coins of various denominations. For example, consider possible denominations of $(1, 2, 5, 10)$ and desired change of 17. The minimum number of coins is 3 $(2 + 5 + 10)$. What is the complexity of your solution?

**5.** Exercises from CLRS Textbook: 15.1-3, 15.4-3, 15-1, 16.1-1

3. Sol:




Dynamic Programming:

$$\mathrm{sum}((a,b),(c,d)) = \mathrm{sum}((0,0),(c,d)) + \mathrm{Sum}((0,0),(a,b)) - \mathrm{Sum}((0,0),(a,d)) - \mathrm{Sum}((0,0),(c,b))$$

therefore only $\mathrm{Sum}((0,0),(x,y))$ matters, store the results in a matrix $R$ $(M \times N)$:

```
R[0,0] = D[0,0]
for x = 1 ... M-1:
    R[x,0] = R[x-1,0] + D[x,0]        // Compute the marginal R[x,0]
for y = 1 ... N-1:
    R[0,y] = R[0,y-1] + D[0,y]        // Compute the marginal R[0,y]
for x = 1 ... M-1:
    for y = 1 ---- N-1:
        R[x,y] = R[x-1,y] + R[x,y-1] + D[x,y]
                 - R[x-1,y-1]
return R
```

Time complexity: $\mathrm{sum}((a,b),(c,d))$ $\Theta(1)$

matrix $R$ $\Theta(M \times N)$

4. Sol: $(1, 2, 5, 10)$ and desired change $x$.

$DP(x)$: (Up - ~~to~~ - Bottom)

Return one of the following if $x$ satisfies:

$N(0) = 0$; $N(1) = 1$; $N(2) = 1$; $N(5) = 1$; $N(10) = 1$;

else if $x > 10$:

Return $N(x-10) + 1$;

else if $x > 5$:

Return $N(x-5) + 1$;

else if $x > 2$:

Return $N(x-2) + 1$;

else:

Raise ("$x < 0$!").

Time complexity: $O(x)$.

Space complexity: $O(x)$.

$DP'(x)$: (Bottom - UP)

for $n = 0 \cdots x$.

if $n$ satisfies one of the following:

$N(0) = 0$; $N(1) = 1$; $N(2) = 1$; $N(5) = 1$; $N(10) = 1$;

else if $n > 10$:

$N(n) = N(n-10) + 1$;

else if $n > 5$:

$N(n) = N(n-5) + 1$;

else if $n > 2$:

$N(n) = N(n-2) + 1$;.

Time and Space complexity: $O(x)$.

5 . 15.1-3, 15.4-3, 15-1, 16.1-1

MODIFIED - ROD - CUT : ( Bottom - UP )

// $p_i$ is the array of prices, each cut incurs a cost $c$.

$r[0 \ldots n]$, $s[0 \ldots n]$ be new arrays.

$r[0] = 0$.

for $j = 1 \ldots n$ :

    $q = -\infty$.

    for $i = 1 \ldots j$ :

        if $i < j$ :

            if $q < p[i] - c + r[j-i]$ // there's a cutting cost.

                $q = p[i] - c + r[j-i]$

                $s[j] = i$.

        if $i = j$ :

            if $q < p[i]$ // incurs no cost of cutting.

                $q = p[i]$

                $s[j] = i$.

    $r[j] = q$

Return $r[0 \ldots n]$ and $s[0 \ldots n]$

The problem - sub-problem structural transition can be rewritten as:

$$r[n] = \max\{ p[n],\ r[1] + r[n-1] - c,\ \ldots \ldots\ r[n-1] + r[1] - c \}$$

thus we can update the comparison variable $q$

MEMORIZED - LCS - LENGTH ( X, Y ) :

```
m = X.length.
n = Y.length.
Let b[1...m, 1...n] and c[0...m, 0...n] be new tables
for i = 1 ... m
    c[i, 0] = 0
for j = 1 ... n.
    c[0, j] = 0.
for i = 1 ... m
    for j = 1 ... n.
        if X[i] = X[j]
            c[i, j] = c[i-1, j-1] + 1
            b[i, j] = "↖"
        else if c[i-1, j] ≥ c[i, j-1] .
            c[i, j] = c[i-1, j] .
            b[i, j] = "↑" .
        else c[i, j] = c[i, j-1] .
            b[i, j] = "←"
return c and b .
```

The same algorithm .

Longest Simple Path in A Directed Acyclic Graph.

Given $G = (V, E)$ $(s \to t)$.

$n = |V|$ coding nodes $\{0, 1, \dots, n-1\}$.

$r[0 \dots n-1, 0 \dots n-1]$. $p[0 \dots n-1, 0 \dots n-1]$.

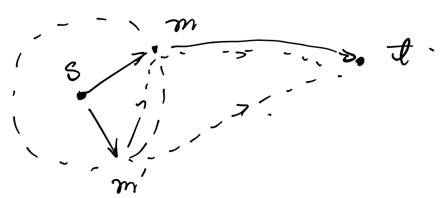
results of path weight sum paths.

$r[s, s] = 0$ and $P[s, s] = \phi$ $\forall s \in \{0, \dots n-1\}$

```
LSP (G, s -> t) :      // not complete.
    if s = t   return 0.
    q = -∞ .
    for m in adj(s).     // list nodes that're reachable.
        if q < LSP(G, s->m) + LSP(G, m->t)
            q = LSP(G, s->m) + LSP(G, m->t)
    Return q
```

Since the graph is acyclic. Once edges going from s to m exist, there will be no path from m to s, therefore sub problem graph is also G since there's no need to exclude s.

this is actually a NP-hard problem.



Dynamic Programming for activity selection :

Sol :

Recurrence : $C[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \phi \\ \max\limits_{a_k \in S_{ij}} \{ c[i,k] + c[k,j] + 1 \} & \text{otherwise.} \end{cases}$

$S_{ij}$ : the set of activities in $\overset{a_i}{\dashv} \underbrace{\{ \qquad \}}_{}$ $\overset{a_j}{\vdash}$ ($|S_{ij}|$ activities)

$A_{ij} \subseteq S_{ij}$ the optimal solution.

$A_{ik} = A_{ij} \cap S_{ik}$ $\quad A_{kj} = A_{ij} \cap S_{kj}$, . $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$

```
DP ( s , f ) :
  n = s.length.
  for j = 1 ...... n :
      for i = 0 --- j :
          q = 0
          S[ij] = ∅ .
          for k = i+1 : j-1 :
              S[ij] = S[ij] ∪ { a_k } if { a_k.s > f.i && a_k.f < s.j }
          // S_ij = { a_k | a_k.s > f.i . a_k.f < s.j } .
          for a_k in . S_ij :
              if q ≤ c[i,k] + c[k,j] + 1 :
                  q = c[i,k] + c[k,j] + 1 .
          c[i,j] = q
  Return C .
```

Running Time

$O(n^3)$.

$> O(n)$

( Greedy ).