Week 8 Recitation

1. Consider an $n \times n$ grid A whose cells contain either a block "X" or a certain number of coins, which can also be negative. Design an algorithm to collect the maximum number of coins starting from the upper-left corner $(1, 1)$ and finishing in the lower-right corner (n, n) by moving only to the right or down. Find the solution path like shown below.

0	5	2	X	2
1	3	2	-5	3
10	5	X	5	5
10	1	0	1	2
100	X	3	-5	1

0	5	2	X	2
1	3	2	-5	3
10	5	X	5	5
10	1	0	1	2
100	X	3	-5	1

You can assume that at least one path exists from (1,1) to (n,n)

In this example, the maximum number of coins collected is 26 and the path can be described by { D, D, D, R, R, R, R, D}.

$Opt(i,j)$ = max coins collected starting at (1,1) and ending at (i,j)

$$Opt(i,j) = \begin{cases} -\infty, & \text{if } A[i][j] = \text{"X"} \text{ or } i=0 \text{ or } j=0 \\ A[i][j], & \text{if } i=1 \text{ and } j=1 \\ A[i][j] + \max\{Opt(i,j-1), Opt(i-1,j)\}, & \text{otherwise} \end{cases}$$

$Opt(n,n)$ = max coins collected starting at (1,1) and ending at (n,n).

Answer to the problem!

Recursive top-down approach with memoization.

```
Opt(i, j, A, dict):                    // Assume d constant per line
    if i==0 or j==0 or A[i][j] == "X"
        return -∞

    if i==1 and j==1
        return A[i][j]

    if dict[i][j] contains value       -> if (i,j) state has been visited before
        return dict[i][j]

    dict[i][j] = A[i][j] + max{Opt(i, j-1, A, dict),
                               Opt(i-1, j, A, dict)}.

    return dict[i][j]
```

Runtime:

Since each state (i,j) is calculated at most once thanks to memoization using dict[i][j] and that we do constant "d" number of operations during that, the total is

$$8d \cdot n \cdot n \in O(n^2)$$

Iterative bottom-up dynamic programing approach.

```
Opt(n, A, dict):
for i=0 to n                                   //constant d per line
    for j=0 to n
        if i==0 or j==0 or A[i][j]=="X"
            dict[i][j] = -∞
        else if i==1 and j==1
            dict[i][j] = A[i][j]
        else
            dict[i][j] = A[i][j] + max{ dict[i][j-1],
                                        dict[i-1][j] }.
return dict[n][n]
```

Runtime:

$$d + 2dn + n^2 \cdot 7d \leq (d + 2d + 7d)\, n^2 = 10d\, n^2 \in \underline{\Theta(n^2)}$$

This is the same as recursive top-down approach but iterative!

$A =$

0	5	2	X	2
1	3	2	-5	3
10	5	X	5	5
10	1	0	1	2
100	X	3	-5	1

dict =

	0	1	2	3	4	5
0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
1	$-\infty$	0	5	7	$-\infty$	$-\infty$
2	$-\infty$	1	8	10	5	8
3	$-\infty$	11	16	$-\infty$	10	15
4	$-\infty$	21	22	22	23	25
5	$-\infty$	121	$-\infty$	25	20	26

$\max\{-\infty, -\infty\} + 2 = -\infty$

max coins from (1,1) to (n,n)

We still need to find the path {D, D, D, R, R, R, R, D} that leads to the maximum coin collection.

```
SolutionPath(n, A, dict):
Sol ← n×n grid of strings.        // Assume this has d·n^2 cost
                                  // and other lises have d each
for i=0 to n
    for j=0 to n
        if dict[i][j] == -∞
            sol[i][j] = ""        // empty string.
        else
            prevOptimal = dict[i][j] - A[i][j].
            if prevOptimal == dict[i][j-1]
                sol[i][j] = sol[i][j-1] + "R"
            else
                sol[i][j] = sol[i-1][j] + "D"

return sol[n][n]
```

Runtime:

$$d \cdot n^2 + d + 2dn^2 + n^2 \cdot 9d \leq (d + d + 2d + 9d)n^2 = 13dn^2 \in O(n^2)$$

Bonus: Try to find the solution path in $c \cdot (2n-1) \in O(n)$ time!

2. You are in a book shop which sells n different books. Let $p = \{ p_1, p_2, \ldots, p_n \}$ represent the prices of the books and $s = \{ s_1, s_2, \ldots, s_n \}$ represent the number of pages of each book. Buying book i will cost p_i money and will gain you s_i number of pages.

Design an algorithm to determine the maximum number of pages you can buy in total given that you only have X amount of money. You can buy each book at most once.

Example:
$n = 4$ and $X = 10$
$p = \{ 4, 8, 5, 3 \}$
$s = \{ 5, 12, 8, 1 \}$

Explanation: You can buy books indexed 1 and 3. Their price is $4 + 5 = 9$ and the number of pages is $5 + 8 = 13$.

It does not matter whether you go from n to 1 or from 1 to n.

Let's start from the n'th book with X amount of money. We can proceed to (n-1)'th book with either $(X - p_n)$ money if we buy n'th book or with X money without buying.

Let $Opt(i, j)$ be the maximum number of pages we can buy with j amount of money from the first i books.

Then,

$$Opt(i,j) = \begin{cases} 0, & \text{if } i == 0 \quad (1) \\ Opt(i-1, j), & \text{if } i > 0 \text{ and } j < p_i \quad (2) \\ \max\{ s_i + Opt(i-1, j-p_i), Opt(i-1, j) \}, & \text{if } i > 0 \text{ and } j \geq p_i \quad (3) \end{cases}$$

(1) There are 0 number of pages we can buy from the first 0 books

(2) if we can not afford i'th book, then $Opt(i,j) = Opt(i-1, j)$

(3) if we can afford, then it does not necessarily mean we should buy it because skipping might result in better overall number of pages.

Recursive top-down approach:

```
Opt(i, j, dict, n, s, p):                    // Assume d for each line
    if i == 0
        return 0
    if dict[i][j] contains value             // state (i,j) has been seen
        return dict[i][j]                    // before
    if j < p[i]
        dict[i][j] = Opt(i-1, j)
    else j ≥ p[i]
        dict[i][j] = max{s[i] + Opt(i-1, j-p[i]), Opt(i-1, j)}
    return dict[i][j]
```

(dict, n, s, p) are omitted.

Runtime:

There are $n \cdot X$ number of unique (i, j) pairs/states is visited at most once and each such state does at most $9d$ number of operations, hence $\underbrace{9d}_{c=9d} \cdot n \cdot X \in O(nX)$

Iterative bottom-up dynamic programming approach:

```
Opt(n, X, s, p, dict):
for i=0 to n                                   // Constant d per line
    for j=0 to X
        if i==0
            dict[i][j]=0
        else if j < p[i]
            dict[i][j]= dict[i-1][j]
        else   // j ≥ p[i] so j-p[i] ≥ 0
            dict[i][j]= max{ s[i] + dict[i-1][j-p[i]],
                             dict[i-1][j] }
return dict[n][X]
```

Runtime:

$$d + 2dnX + 7d \cdot nX \leq (d + 2d + 7d)nX = \underbrace{10d}_{C=10d}nX \in O(nX)$$