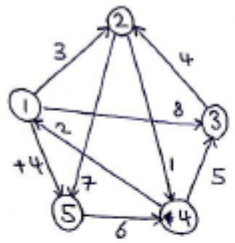
Algorithm Design 3992
April 2021
Instructor : Dr.Shafiei

1.



W_2

	1	2	3	4	5
1	0	3	8	∞	4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	5	0	∞
5	∞	∞	∞	6	0

1 edge :

	1	2	3	4	5
1	0	3	8	∞	4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	5	0	6
5	∞	∞	∞	6	0

k=2

	1	2	3	4	5
1	0	3	8	4	4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	5	0	6
5	∞	0 ∞	∞	6	0

k=3

	1	2	3	4	5
1	0	3	8	4	4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	5	0	6
5	∞	∞	∞	6	0

k=4

	1	2	3	4	5
1	0	3	8	4	4
2	3	0	6	1	7
3	7	4	0	5	11
4	2	5	5	0	6
5	8	11	11	6	0

k=5

	1	2	3	4	5
1	0	3	8	4	4
2	3	0	6	1	7
3	7	4	0	5	11
4	2	5	5	0	6
5	8	11	11	6	0

P

	1	2	3	4	5
1	-	-	-	2	-
2	4	-	4	-	-
3	4	-	-	2	2
4	-	1	-	-	1
5	4	4	4	-	-

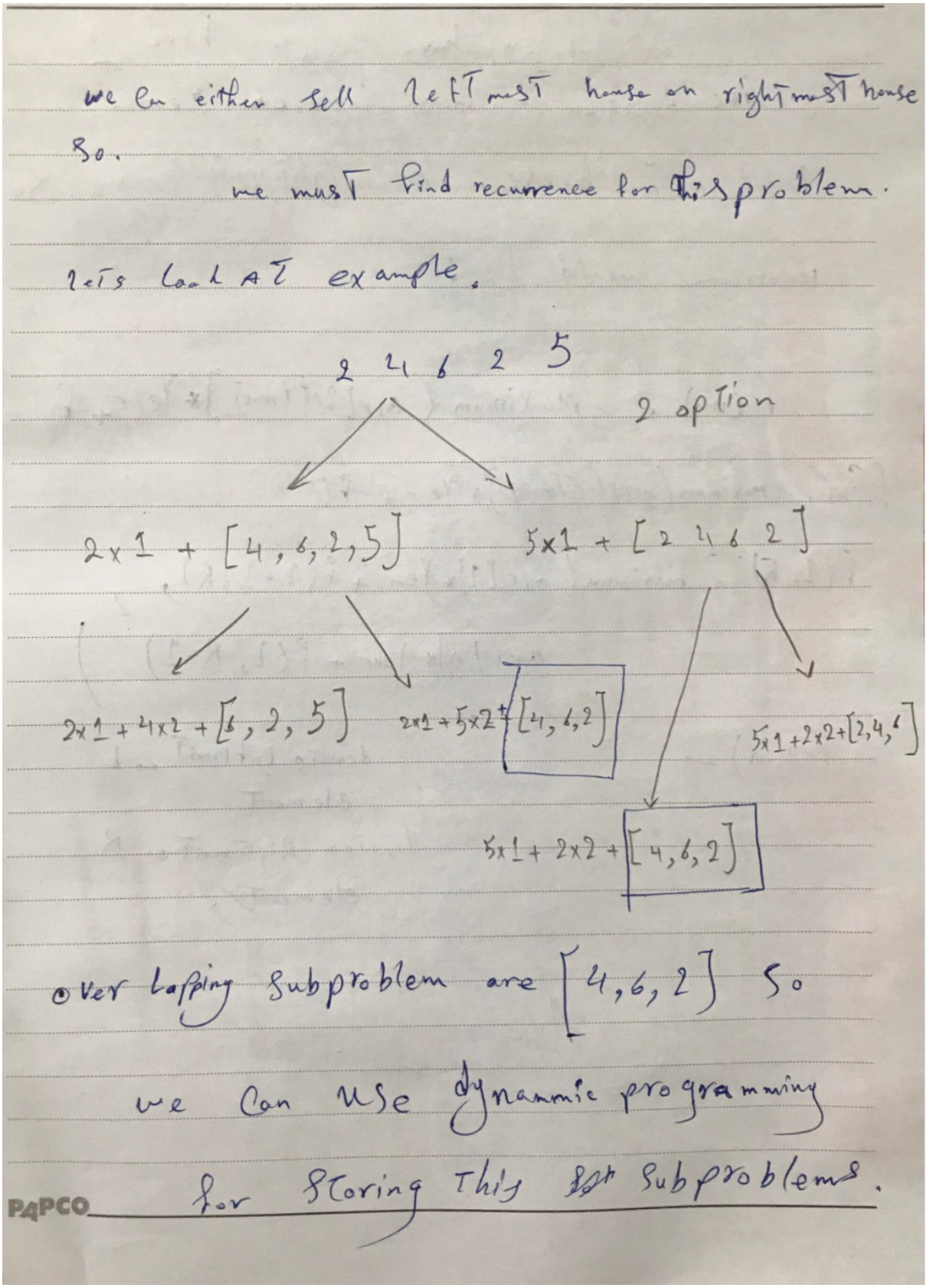
2.

	∅	{2}	{3}	{4}	{2 , 3}	{3 , 4}	{2 , 4}
2	2	2	INF	10	INF	19	10
3	INF	8	INF	13	8	13	13
4	6	6	INF	6	INF	INF	6

W(1 , 2) + D(2 , {3 , 4}) OR W(1 , 3) + D(3 , {2 , 4}) OR W(1 , 4) + D(4 , {2 , 3}) = (22 , 21,inf)

Algorithm Design 3992
April 2021
Instructor : Dr.Shafiei

3.



Algorithm Design 3992
April 2021
Instructor : Dr.Shafiei

So likeyou have seen we have 2 choice ~~decision~~ to make in each year. left most house or rightmost house. So recurrence would be:

~~f(n) = Maximum(arr[leftmost] * Year +~~

~~f(n) = Maximum(arr[leftmost] * Year + f(~~

$f(L, R) = \text{maximum}(arr[L] * Year + f(L+1, R),$
$arr[R] * Year + f(L, R-1))$

$f(L, R)$ → denote leftmost ← L element
denotes Rightmost ← R element.

Ashkan

Algorithm Design 3992
April 2021
Instructor : Dr.Shafiei

---

ex > house.go > findMaxProfit

```go
package main

import "fmt"

func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}

func maxProfit(profit []int) int {
    n := len(profit)
    dp := make([][]int, n)
    for i := 0; i < n; i++ {
        dp[i] = make([]int, n)
        for j := 0; j < n; j++ {
            dp[i][j] = -1
        }
    }
    return findMaxProfit(profit, dp, 0, n-1, 1)
}

func findMaxProfit(profit []int, dp [][]int, left, right, year int) int { // main function
    if left == right {
        return profit[left] * year
    }
    if dp[left][right] != -1 { // using memoization
        return dp[left][right]
    }

    dp[left][right] = max(
        profit[left]*year+findMaxProfit(profit, dp, left+1, right, year+1),
        profit[right]*year+findMaxProfit(profit, dp, left, right-1, year+1)) // two choice either pick left house or right one

    return dp[left][right]
}

func main() {
    fmt.Println(maxProfit([]int{2, 4, 6, 2, 5}))
}
```

TERMINAL SQL CONSOLE: MESSAGES DEBUG CONSOLE PROBLEMS OUTPUT 1: zsh

```
(env) ➜  ex go run house.go
64
(env) ➜  ex
```

Algorithm Design 3992
April 2021
Instructor : Dr.Shafiei

4.

ex > alpha.go > ...

```go
func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}

const (
    ASCENDING  = 0
    DESCENDING = 1
)

func aplhaFinder(arr []int) int {
    n := len(arr)
    dp := make([][]int, n) // denote 0 for ascending order and 1 for descending Order
    for i := 0; i < n; i++ {
        dp[i] = make([]int, 2)
        dp[i][ASCENDING], dp[i][DESCENDING] = 1, 1 // every single element has the property of alpha length 1
    }

    for i := 0; i < n; i++ {
        for j := 0; j < i; j++ {
            if arr[i] > arr[j] {
                // ASCENDING when element is higher and last element was DESCENDING one.
                dp[i][ASCENDING] = max(dp[i][ASCENDING], dp[j][DESCENDING]+1)
            } else {
                // DESCENDING when element is lower and last element was ASCENDING one.
                dp[i][DESCENDING] = max(dp[i][DESCENDING], dp[j][ASCENDING]+1)
            }
        }
    }
    maximum := -int(1e3)
    for i := 0; i < n; i++ {
        maximum = max(maximum, max(dp[i][ASCENDING], dp[i][DESCENDING]))
    }
    return maximum
}

func main() {
    fmt.Println(aplhaFinder([]int{10, 22, 9, 33, 49, 50, 31, 60}))
}
```

TERMINAL SQL CONSOLE: MESSAGES DEBUG CONSOLE PROBLEMS 1 OUTPUT

```
(env) ➜  ex go run alpha.go
6
(env) ➜  ex
```

Algorithm Design 3992
April 2021
Instructor : Dr.Shafiei

Let's find problem recurrence and then code it up! Aforin

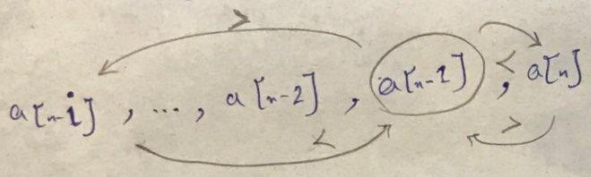
we can see if our last element is like a[n]

so we can make two choice either last element before this element was larger or was lower than this element. so what can we say?

a[n-i] ... a[n-1], a[n]

this was larger than a[n] → a[n-1] < a[n]
or lower than a[n] → a[n-1] > a[n]

so our graph for a[n-i], .., a[n-1], a[n] would be:



so each element has 2 property so recurrence will be:

dp [i][0] → when we Assume ith element is larger than previous elements.

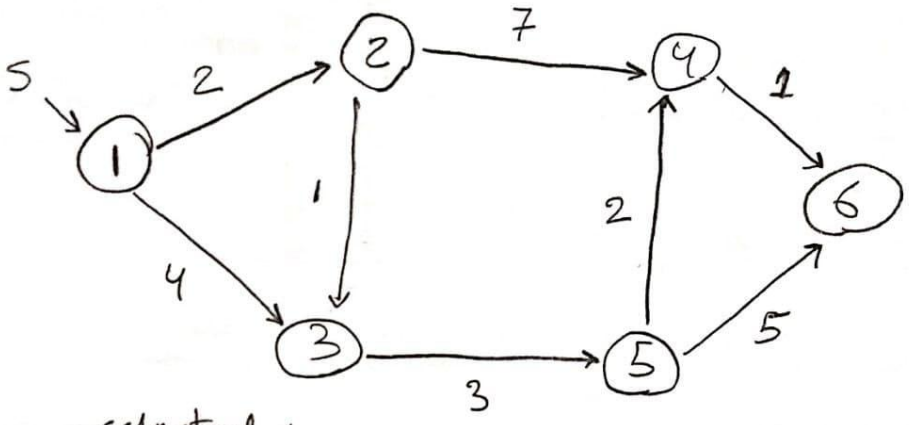
dp [i] [1] → when we Assume ith element is lower than previous elements.

$$dp[i][0] = \max\left(\underbrace{dp[i][0]}_{\text{final element}}, \max_{j=0}^{i} \text{if } a[i] > a[j]\ dp[j][1] + 1\right)$$

اگر آخرین عنصر را بزرگ فرض کنیم پس تمام عناصر کوچکتر از این عنصر می توانند در شرط صدق کنند و همین شکل برای زمانی که فرض کنیم این عنصر از بقیه عناصر کوچک است.

Algorithm Design 3992
April 2021
Instructor : Dr.Shafiei

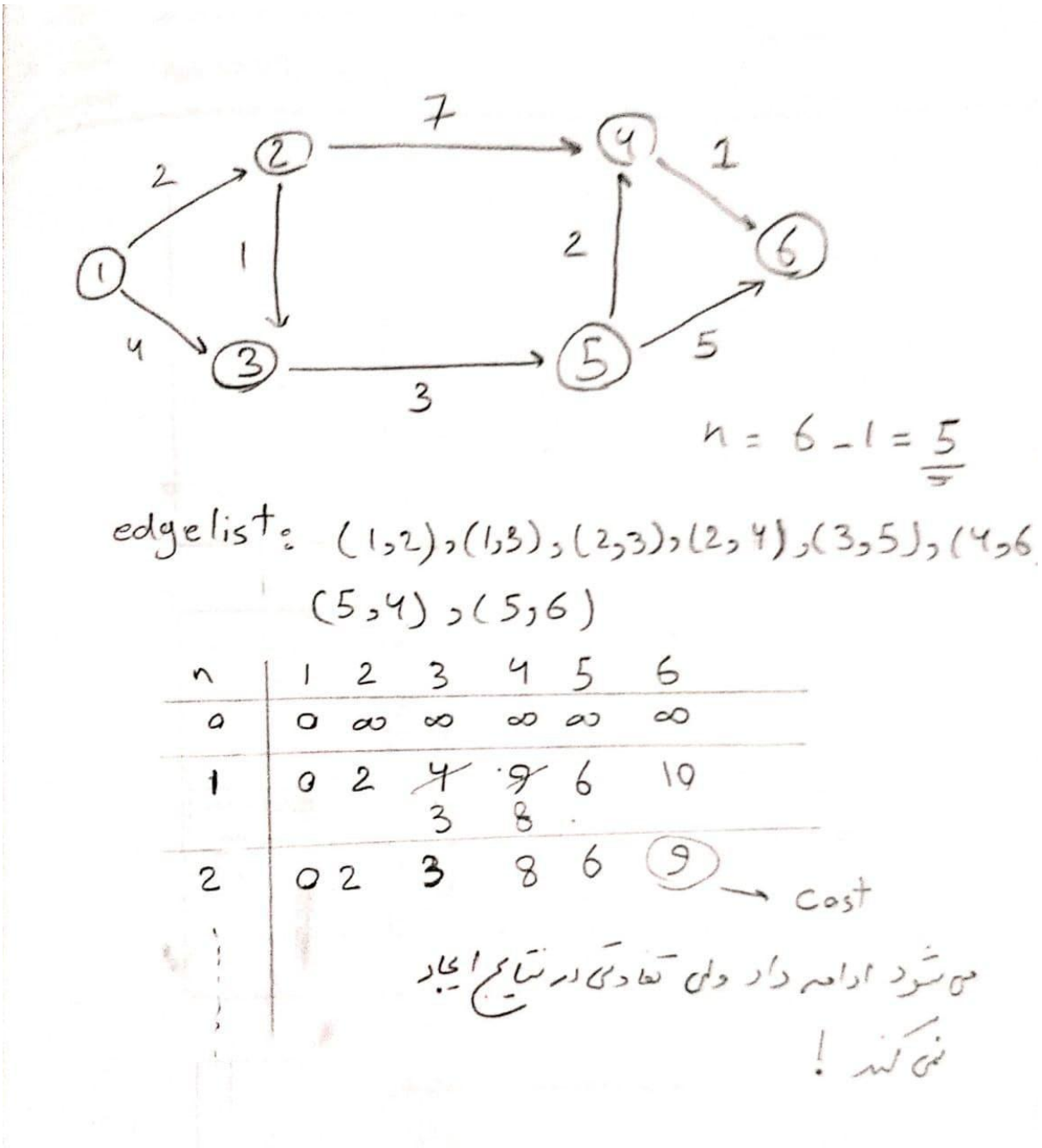
5.
Dijkstra :



selected vertex	2	3	4	5	6
2	(2)	4	∞	∞	∞
3	2	(3)	9	∞	∞
5	2	3	9	(6)	∞
4	2	3	(8)	6	11
6	2	3	8	6	(9) → cost

Algorithm Design 3992
April 2021
Instructor : Dr.Shafiei

Bellman-Ford :



b) The only difference between the two is that Bellman-Ford is also capable of handling negative weights whereas Dijkstra Algorithm can only handle positives.

Note :
Dijkstra is however generally considered better in the absence of negative weight edges, as a typical binary heap priority queue implementation has O((|E|+|V|)log|V|) time complexity [A Fibonacci heap priority queue gives O(|V|log|V| + |E|)], while the Bellman-Ford algorithm has O(|V||E|) complexity.