(1) Adam Call 4/12/2023

Homework 3 Final Draft

1)

(1) $V_n = \underset{\{i_1,\dots,i_n\}}{\text{maximize}} \sum_{i=1}^{n} V_i$

V – value, n – number of jobs, s_i – start time
f_i – finish time, t – time
x_i – ith decision made about job

$s_1 \xrightarrow{x_1} s_2 \xrightarrow{x_2} \cdots \xrightarrow{x_{k-1}} s_k$

$M_{i+1} = f_i(M_i, x_i)$

selection function

$V(x, M_1) = g_n(M_n) + \sum_{i=1}^{n-1} g_i(M_i, x_i)$

Bellman Equation

$V[i,t] = \max_{x_i \in \{0,1\}} \{ x_i v_i + V[i-1, t - x_i(f_i - s_i)$

$(x_i s_i \geq x_k f_k \quad k > i) \quad (1 \leq i \leq n, 0 \leq t \leq f(n))$

(2)

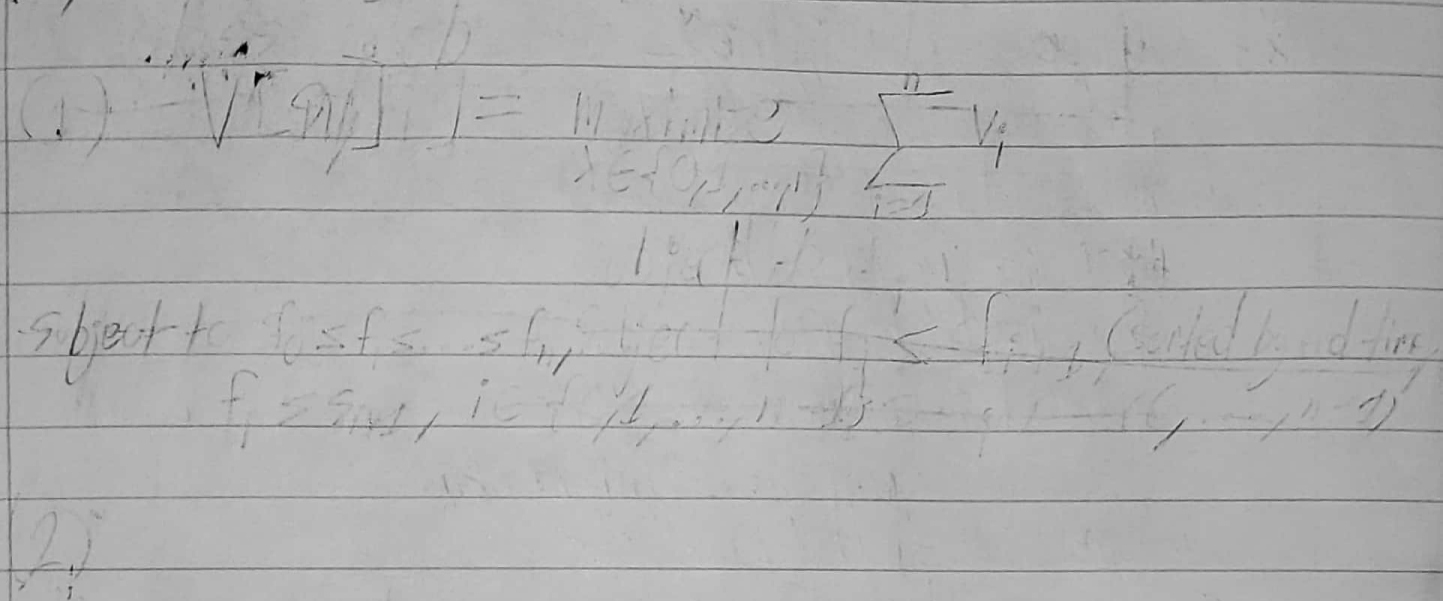
HW3 Final Draft

```
class Job
    init(start, finish, value)
        self.s = start
        self.f = finish
        self.v = value
```

(3)

HW 3 Final Draft

(2)



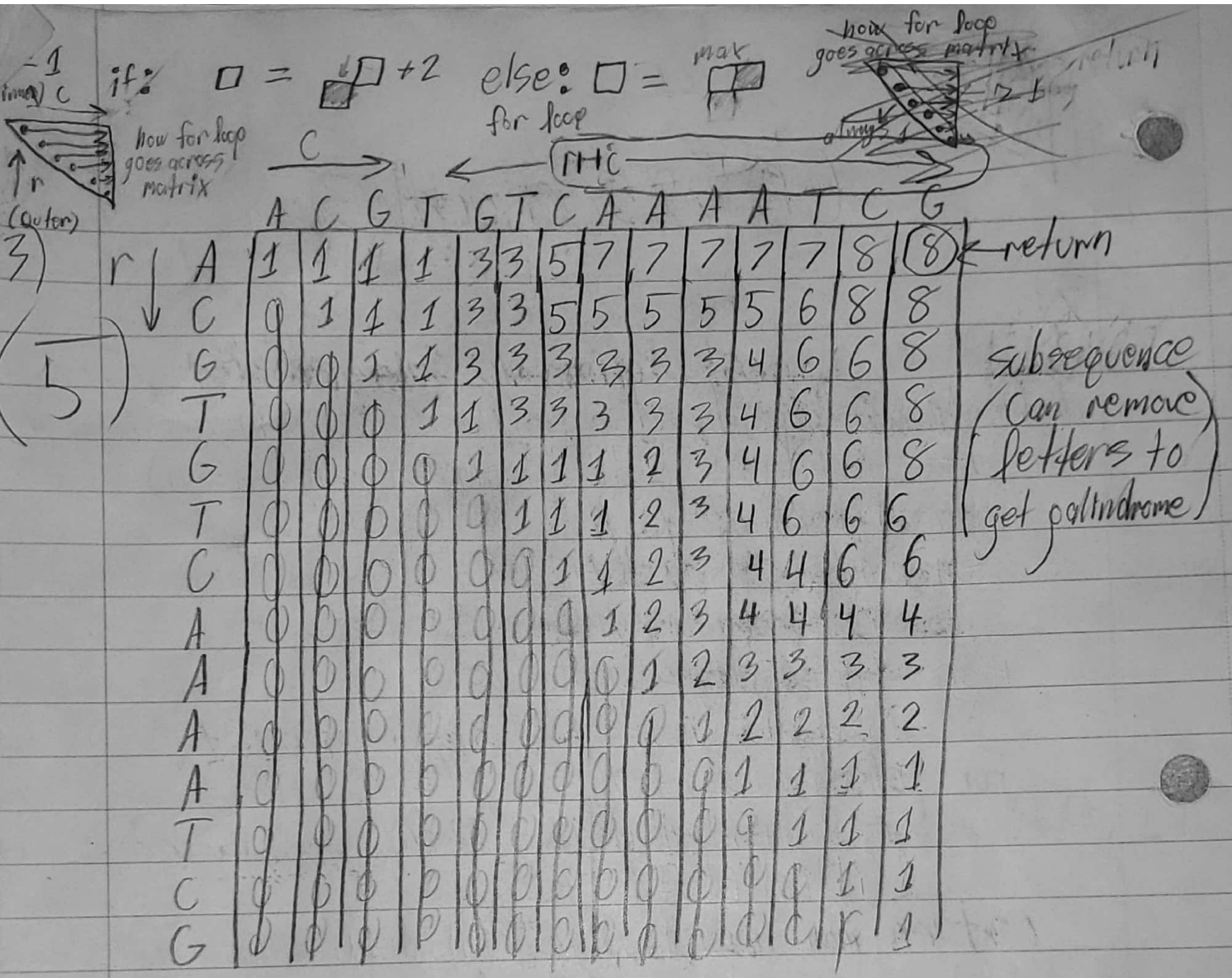
(2)

```
def Weighted_Task_Selection(job_list[], n)
    // sort job list by end time (low to high)
    total_profit[] = arr[n]
    total_profit[n] = job_list[n].V

    for i from n-1 to 1
        compatible_table[] = arr(len=n)
        for j from i+1 to n
            if job_list[i].f ≤ job_list[j].S
                compatible_table[j] = job_list[j].V
        job_list[i].V += max(compatible_table)
        total_profit[i] = job_list[i].V

    return max(total_profit)
```

(4)

(3)

```
// job_list sorted by v_i/(f_i - s_i)
def Greedy_Weighted_Selection(job_list[], n)
    max_value = 0
    memoize[] = arr[2n+1]

    for i = 1 to n
        memoize[i] = 0

    for i = 1 to n
        count = 0
        for jobN = job_list[i].s to job_list[i].f
            count += memoize[jobN]
            if count = 0
                for k = job_list[i].s to job_list[i].f
                    memoize[k] = 1
            max_value += job_list[i].v

    return max_value
```


-1 times
c
r
(outer)
how for loop goes across matrix
if: □ = □ +2
else: □ = max
for loop
C
R
how for loop goes across matrix
A C G T G T C A A A A T C G
A 1 1 1 1 3 3 5 7 7 7 7 7 8 8
C 0 1 1 1 3 3 5 5 5 5 5 6 8 8
G 0 0 1 1 3 3 3 3 3 3 4 6 6 8
T 0 0 0 1 1 3 3 3 3 3 4 6 6 8
G 0 0 0 0 1 1 1 1 2 3 4 6 6 8
T 0 0 0 0 0 1 1 1 2 3 4 6 6 6
C 0 0 0 0 0 0 1 1 2 3 4 4 6 6
A 0 0 0 0 0 0 0 1 2 3 4 4 4 4
A 0 0 0 0 0 0 0 0 1 2 3 3 3 3
A 0 0 0 0 0 0 0 0 0 1 2 2 2 2
A 0 0 0 0 0 0 0 0 0 0 1 1 1 1
T 0 0 0 0 0 0 0 0 0 0 0 1 1 1
C 0 0 0 0 0 0 0 0 0 0 0 0 1 1
G 0 0 0 0 0 0 0 0 0 0 0 0 0 1
return
subsequence (can remove letters to get palindrome)
if □ == □ : □ = □ +2 ; □ = max (only use valid indices)
KEEP

(6)

HW3 Final Draft

2)

(1) $a_1 \xrightarrow{s_1} a_2 \xrightarrow{s_2} \cdots \xrightarrow{s_m} a_m$

$b_1 \xrightarrow{s_1} b_2 \xrightarrow{s_2} \cdots \xrightarrow{s_n} b_n$

State variables: $a_1 - a_m$, $b_1 - b_n$

decision variables: $s_1 - s_i = D_i, I_i, C_{ij}, (s_1 - s_j)$

deletion cost $s_i = D_i > 0$
insertion cost $s_i = I_i > 0$
substitution cost $s_j = c_{ij} \geq 0$

$$V_i[a_i] = \min_{\substack{\{i \in m\} \\ \langle a_i \rangle \in S}} O_m(s_m) + \sum_{i=k}^{m-1} O_k(a_k, s_k)$$ (Bellman if modifying string A)

$$V_i[b_i] = \min_{\substack{\{i \in n\} \\ \langle b_i \rangle \in S}} O_n(s_n) + \sum_{i=k}^{n-1} O_k(b_k, s_k)$$ (Bellman if modifying string B)

(7)

HW3 Final Draft

(2)

```
def dynamic_weighted_edit_distance(str1, str2, l1, l2)
    min_table = arr[l1][l2]
    fill min_table with 0's
    for i from 1 to l1+1
        for j from 1 to l2+1
            if i == 1
                min_table[i][j] = sum of ASCII values of
                                  str2[1 to j]

            elif j == 1
                min_table[i][j] = sum of ASCII values of
                                  str1[1 to i]

            else
                choice = min(min_table[i-1][j] // delete
                             min_table[i-1][j-1] // substitute
                             min_table[i][j-1] // insert
                if choice == min_table[i-1][j] // delete
                    min_table[i][j] = choice + 1
                                      ASCII of str1[i-1]
```

(8)

HW3 Final Draft

```
else if choice == min_table[i][j-1] //insert
  min_table[i][j] = choice + ASCII of str2[j-1]

else // substitute
  min_table[i][j] = choice + int(mean of ASCII of str1[i-1], str2[j-1])

return min_table[l1][l2]
```

(3)

```
def greedy_edit_distance(str1, str2, l1, l2)
  min_table = arr[l1][l2]

  fill min_table with 0s

  if l1 == 0
    return sum of ASCII str2 values
  else if l2 == 0
    return sum of ASCII str1 values

  for i from 1 to l1+1
   for j from 1 to l2+1
    if str1[i-1] == str2[j-1]
      min_table[i][j] = min_table[i-1][j-1]
```

(9)

```
else
  choice = min(ASCII of str1[i-1] // delete
               ASCII of str2[j-1] // insert
         round down → int(mean(ASCII of str1[i-1], str2[j-1]))
  if choice == str1[i-1] // delete
     min_table[i][j] = min_table[i-1][j] + choice
  else if choice == str2[j-1] // insert
     min_table[i][j] = min_table[i][j-1] + choice
  else
     min_table = min_table[i-1][j-1] + choice

return min_table[l1][l2]
```

(10)

HW3 Final Draft

(non weighted)

			A	L	G	O	R	I	T	H	M
		0	1	2	3	4	5	6	7	8	9
	A	1	0	1	2	3	4	5	6	7	8
	L	2	1	0	1	2	3	4	5	6	7
	K	3	2	1	1	2	3	4	5	6	7
	H	4	3	2	2	2	3	4	5	6	7
	W	5	4	3	3	3	4	5	6	7	8
	A	6									
	R	7									
	I	8									
	Z	9									
	M	10									
	I	11									

"ALGORITHM"

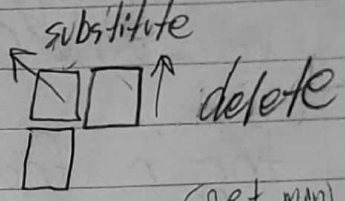
A+"LGORITHM"

AL+"GORITHM"

ALK+"ORITHM"

ALKH+"RITHM"

i/p



(get min)

ALGORITHM

(11)

HW 3 Final Draft

3)

```
def Longest_Palindromic_Subsequence(str, n)
    max_subseq[][] = arr[n, n]

    for r from 1 to n
        max_subseq[r, r] = 1
        for c from r+1 to n
            if(str[r] = str[c])
                max_subseq[r, c] = max_subseq[r+1, c-1] + 2
            else
                max_subseq[r, c] = max(max_subseq[r+1, c], max_subseq[r, c-1])

    return max_subseq[1, n-1]
```

(12)

HW3 Final Draft

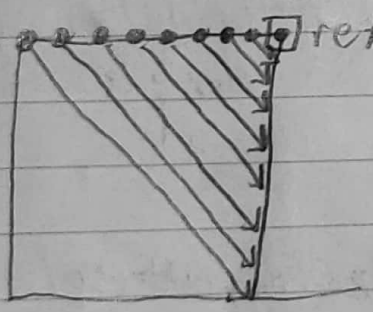
4)

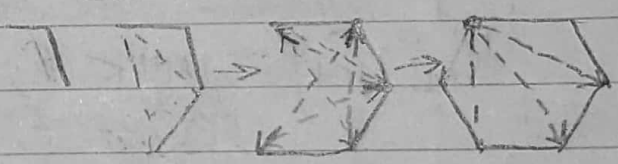
```
def distance(p1, p2) // distance of two points
    return sqrt((p1.x - p2.x)^2 + (p1.y - p2.y)^2)

def cost(points[], n1, n2, n3) // cost of 1 triangle
    p1 = points[n1]                // in triangulation
    p2 = points[n2]
    p3 = points[n3]

    return distance(p1, p2) + distance(p2, p3) + distance(p1, p3)

// 'main' function
def Dynamic_Convex_Polygon_Triangulation(points[], p_size)
    if(p_size < 3)
        return 0

    table[][] = arr[p_size, p_size]

    for r=0 to p_size
        for c=0 to p_size
            table[r, c] = 0
```

(13)

HW3 Final Draft

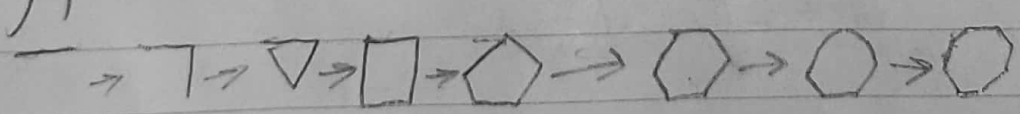
```
for gap from 1 to p_size
    r = 1
    for c from gap to p_size
        if c > r+1
            table[r, c] = MAX integer size
            for q from r+1 to c
                weight = cost(points, r, c, q)
                table[r, c] = min(table[r, c], weight + table[r, q]
                                                    + table[c, q])
        r++
return table[1, p_size]
```

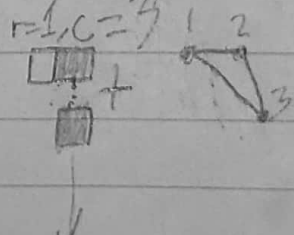




gap - increase # of polygons until you get the full polygon size

gap (1 to 8)





(gap to p_size)