# Synthesis Assignment #3

## CS 5800, Fall 2022
## Dr. Lindsay Jamieson

### Chapter 6: Greedy Algorithms (Continued -)

#### 6.1 Encodings & Huffman's Encoding Algorithm

Before we get to know the famous compression algorithm Huffman coding based on the prefix coding theory, we have to learn what the prefix code is. In computer science, a prefix code complies with the property that no code for one single word is the prefix of any other words. In this case, Huffman coding can prevent some coding errors when executing encoding or decoding. For example, let's say the Huffman code for the character 'A' is 1101, and 11 for character 'B', and 01 for 'C". If we do generate any Huffman code like this, it would cause conflicts during compression or decompression. Since 1101 can be translated into 'BC' but not A which is not what we expected.

Addtionally, effiency is the other feature of Huffman coding. To some extent, it can utilize a binary tree to generate the code for each character and guarantee that the character with a higher frequency can always generate a shorter code. Under this circumstance, the total length for the file would be optimized as the shortest and hence creating a compressed file.

```
Pseudocode from Dusgupta:

function Huffman(f):
Input: An array f[1...n] of frequencies, n = number of symbols
Output: An encoding tree with n leaves
let H be a priority queue of integers, ordered by f
for i = 1 to n: enqueue(H, i)
for k = n+1 to 2n−1:
  i = deletemin(H), j = deletemin(H)
  create a node numbered k with children i, j
  f[k] = f[i] + f[j]
  enqueue(H, k)
```

Exercise: Run a step-by-step execution of Huffman's Encoding Algorithm with a list of frequencies of characters given below.

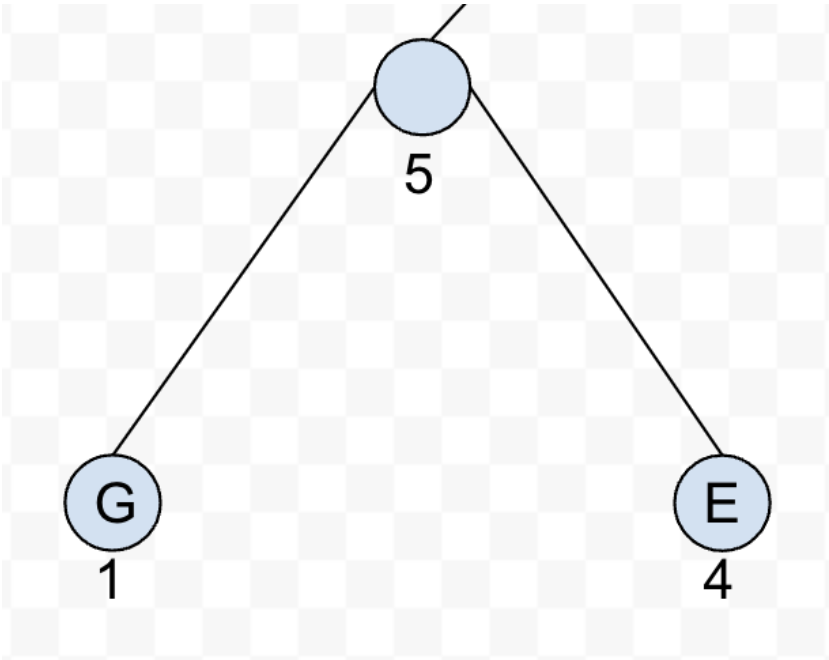
{A = 5, B = 23, C = 12, D = 36, E = 4, F = 19, G = 1}

1. At first, the frequency stack shows as below:

   H = {A = 5, B = 23, C = 12, D = 36, E = 4, F = 19, G = 1}

   i = dequeue(G), j = dequeue(E)

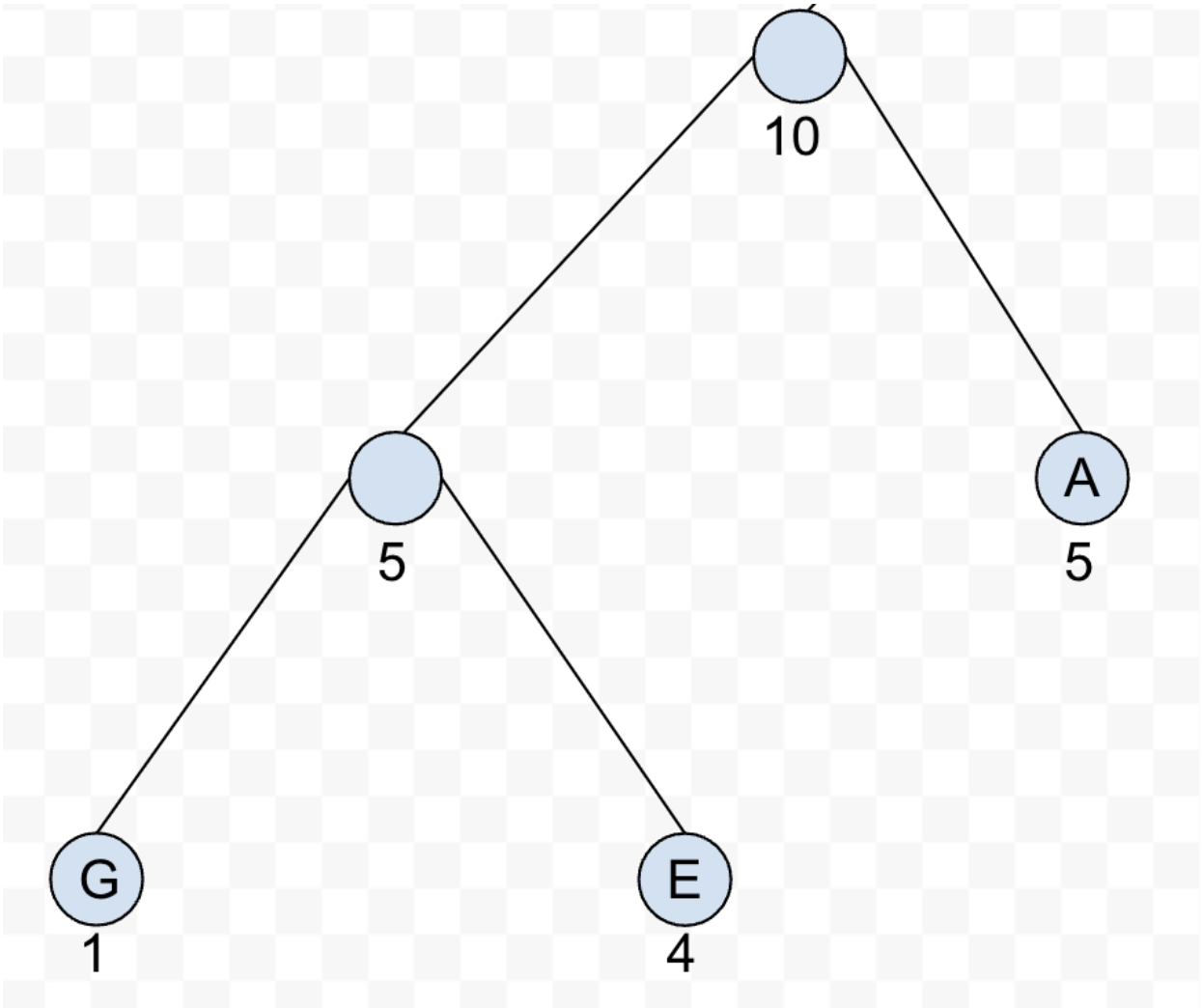
   And we create a new tree node based on i and j




And enqueue treenode 5

```
H = {A = 5, B = 23, C = 12, D = 36, F = 19, treenode = 5}
```

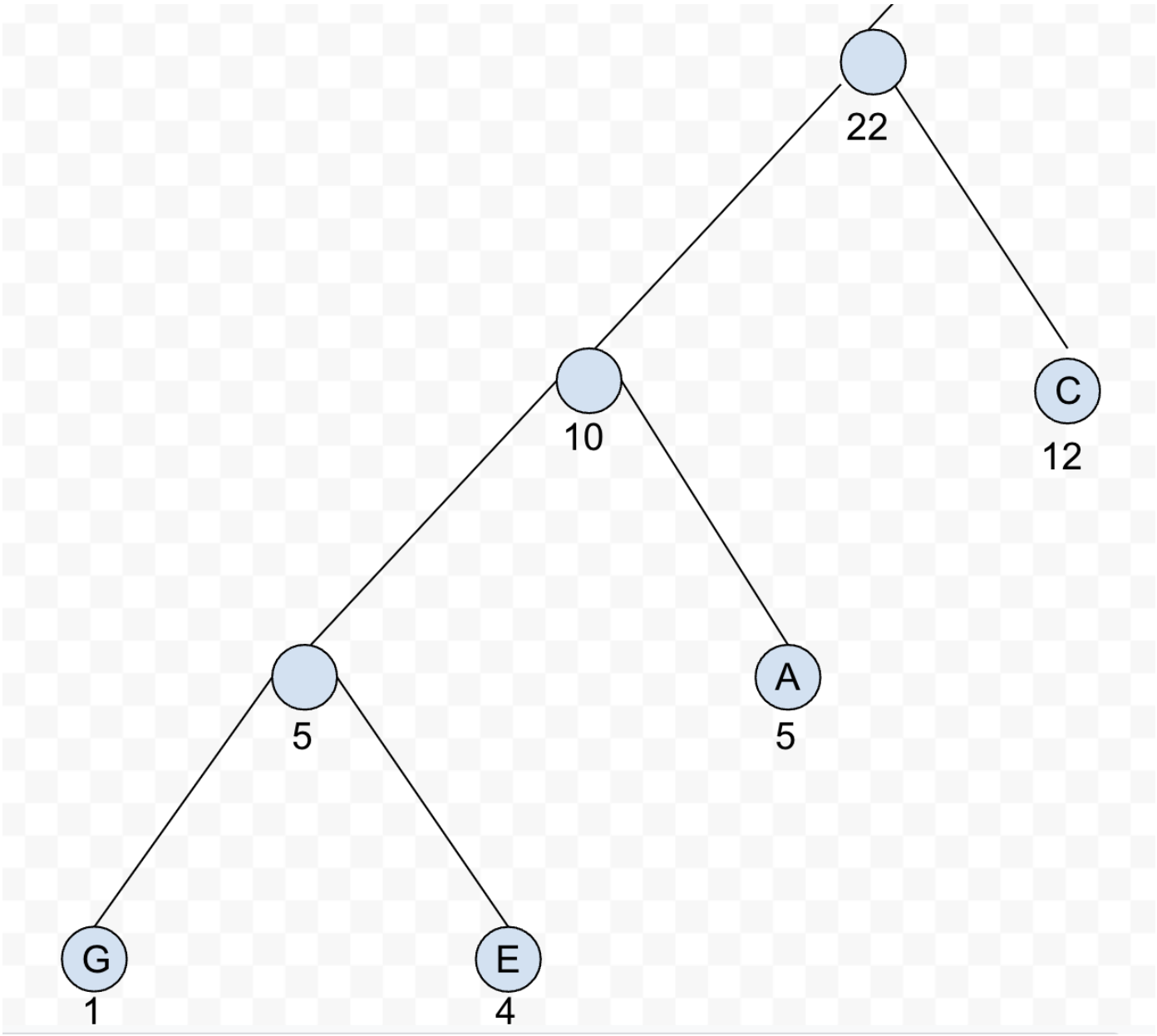
2. i = dequeue(treenode5), j = dequeue(A) = 5 And we create a new tree node based on i and j



And enqueue treenode 10

```
H = {B = 23, C = 12, D = 36, F = 19, treenode = 10}
```

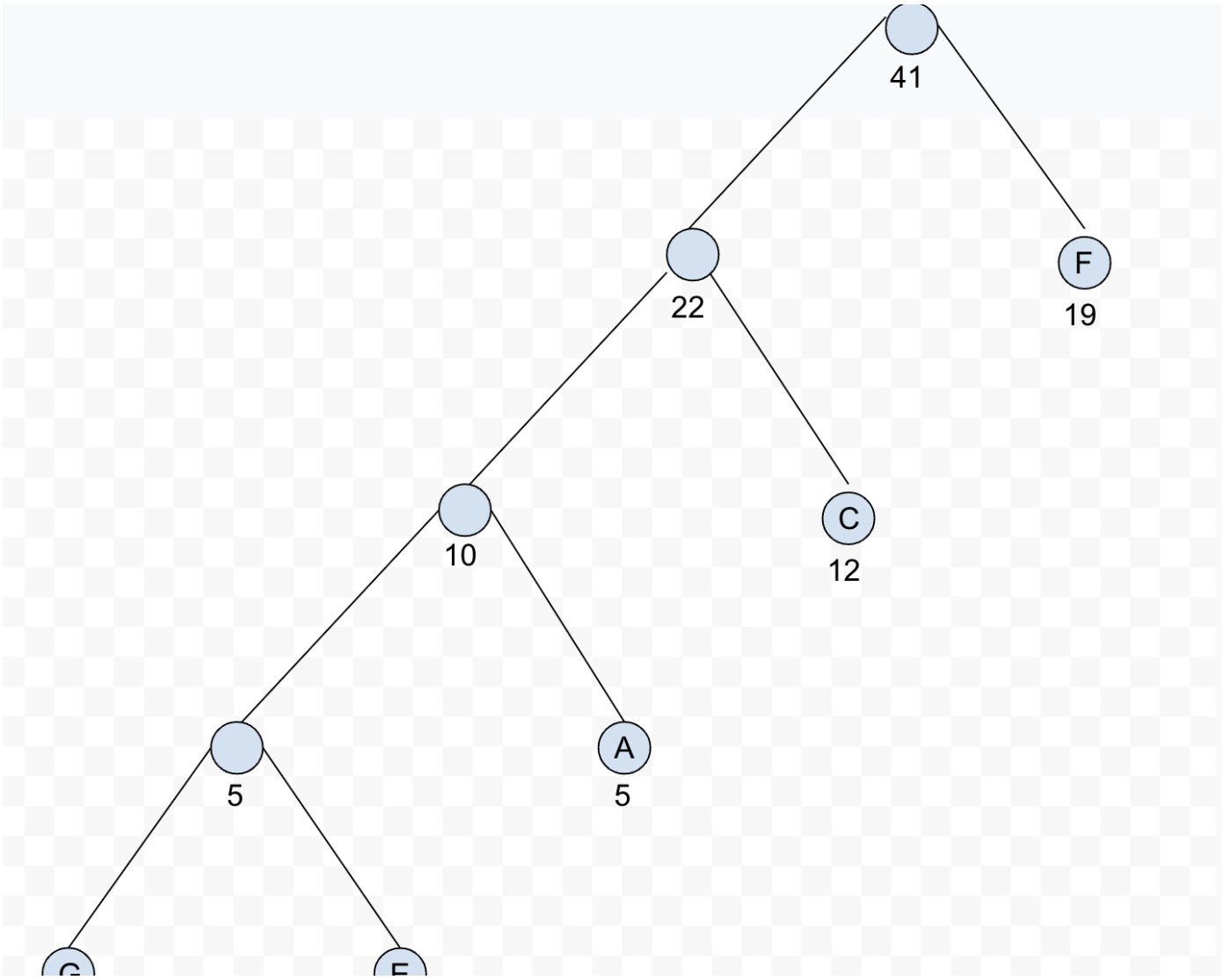
3. i = dequeue(treenode10), j = dequeue(C) = 12 And we create a new tree node based on i and j




And enqueue treenode 22

```
H = {B = 23, D = 36, F = 19, treenode = 22}
```

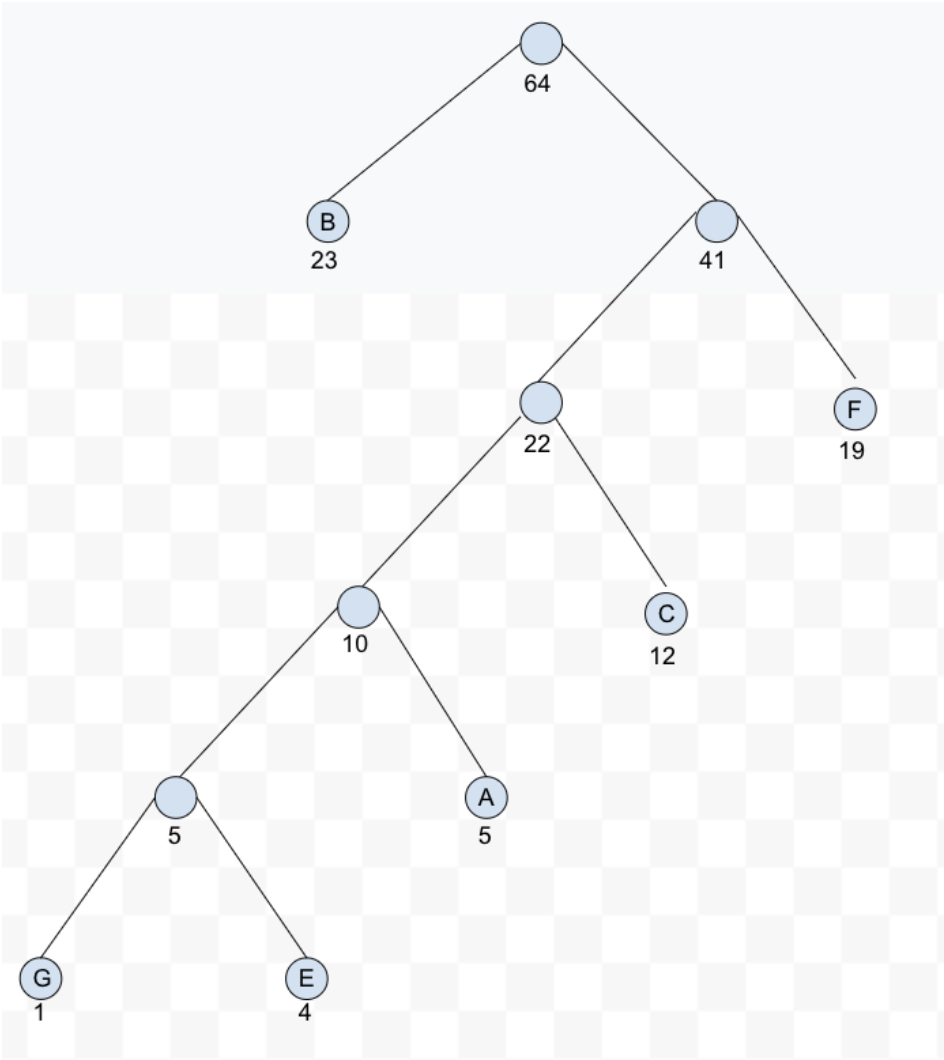
4. i = dequeue(F) = 19, j = dequeue(treenode 22) And we create a new tree node based on i and j




And enqueue treenode 41

```
H = {B = 23, D = 36, treenode = 41}
```

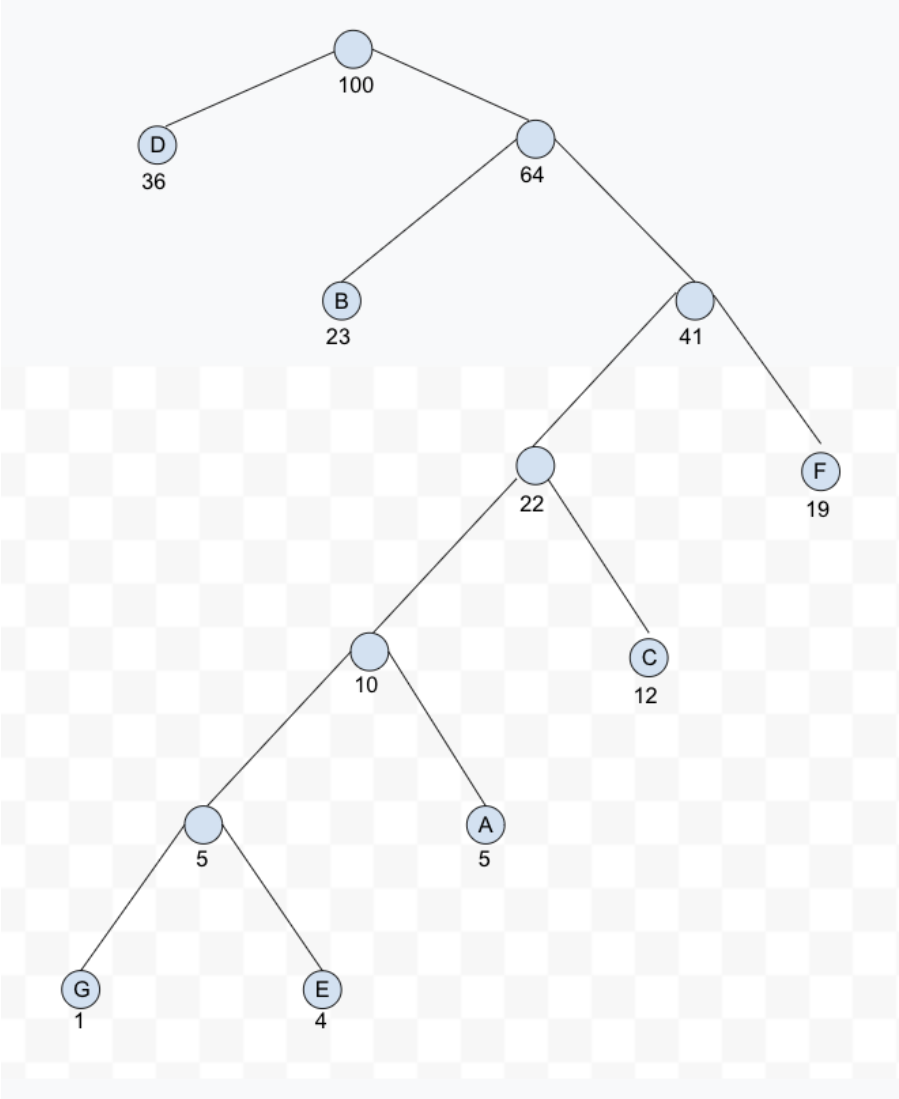
5. i = dequeue(B) = 23, j = dequeue(treenode 41) And we create a new tree node based on i and j




And enqueue treenode 64

```
H = {D = 36, treenode = 64}
```

6. i = dequeue(D) = 36, j = dequeue(treenode 64) And we create a new tree node based on i and j




And enqueue treenode 64

```
H = {treenode = 100}
```

Therefore, A = 11001, B = 10, C = 1101, D = 0, G = 110000, E = 110001, F =

| Node | Value |
|---|---|
| A | 11001 |
| B | 10 |
| C | 1101 |
| D | 0 |
| E | 110001 |
| F | 111 |
| G | 110000 |

### 6.2 Greedy Algorithm Design

Problem - Given a table of jobs where each job is assign a deadline and profit if the job is finished before the deadline. And also, each job takes one single period unit. Try to maximize the profit in a way that all jobs in the given schedule can be finished on time.

| Job | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| Deadline | 2 | 2 | 3 | 2 | 3 | 6 |
| Profit | 10 | 5 | 100 | 20 | 40 | 35 |

Solution:

Here's the pseudocode for the job sequencing algorithm:

```
Input : array Job[N], Hashmap Deadline(Job,i), Hashmap Profit(Job, j)

Output : Maximum profit

for i = 1 to N
priority_queue Q.enqueue(job[i]) // Initialize the job priority queue

while Q != empty
  job = Q.dequeue
  for j = deadline[i] to 1
    if deadline[j] is empty
      ans[j] = job

for i = 1 to N
    maximum += lookupProfit(ans)//lookupProfit means to look up the profit in the Hashmap

return maximum
```

Explanation: First, we should determine whether the current job can fit in the current slot before the deadline. If so, add it to the position that starts from the deadline in order to facilitate more profitable jobs. Otherwise, we just skip the current job until we reach the end of the priority queue.

Here's the step-by-step execution of the algorithm:

| **Job** | **A** | **B** | **C** | **D** | **E** | **F** |
|---|---|---|---|---|---|---|
| Deadline | 2 | 2 | 3 | 2 | 3 | 6 |
| Profit | 10 | 5 | 100 | 20 | 40 | 35 |

| **Job** | **C** | **E** | **F** | **D** | **A** | **B** |
|---|---|---|---|---|---|---|
| Profit | 100 | 40 | 35 | 20 | 10 | 5 |
| Deadline | 3 | 3 | 6 | 2 | 2 | 2 |

1. Job = C deadline = 3

| | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|
| Answer | | | C | | | |

table updated

| **Job** | **E** | **F** | **D** | **A** | **B** |
|---|---|---|---|---|---|
| Profit | 40 | 35 | 20 | 10 | 5 |
| Deadline | 3 | 6 | 2 | 2 | 2 |

2. job = E deadline = 3

| | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|
| Answer | | E | C | | | |

table updated

| **Job** | **F** | **D** | **A** | **B** |
|---|---|---|---|---|
| Profit | 35 | 20 | 10 | 5 |
| Deadline | 6 | 2 | 2 | 2 |

3. job = F deadline = 6

| | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|
| Answer | | E | C | | | F |

table updated

| **Job** | **D** | **A** | **B** |
|---|---|---|---|
| Profit | 20 | 10 | 5 |

| Job | D | A | B |
|---|---|---|---|
| Deadline | 2 | 2 | 2 |

4. job = D deadline = 2

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Answer | D | E | C | | | F |

table updated

| Job | A | B |
|---|---|---|
| Profit | 10 | 5 |
| Deadline | 2 | 2 |

5. job = A deadline = 2

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Answer | D | E | C | | | F |

Since slots before 2 are all occupied, we just skip job A.

table updated

| Job | B |
|---|---|
| Profit | 5 |
| Deadline | 2 |

6. job = B deadline = 2

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Answer | D | E | C | | | F |

Since slots before 2 are all occupied, we just skip job B.

table updated

| Job |
|---|
| Profit |
| Deadline |

maximum profit = 195

Time complexity: O(N)

## Chapter 7: Dynamic Programming

### 7.1 Technical definition

In computer science, it is common that some algorithms may turn out to be significantly time-consuming such as recursions. In order to solve theses complicated problems, a new technique was developed by mathematicians. It is called dynamic programming.

Essencially, it trades space for time, indicating that dynamic programming would take extra spaces in order to store the results of sub-problems. More specifically, we are breaking the original problem into several sub-problems and make use of the realtion between the sub-problems and the original problem to obtain the final answer.

### 7.2 Edit-distance

When we are solving the Edit-distance problem by the approach of dynamic programming, we should look for the sub-problem of it. To begin with, we should be aware that there are 3 ways to edit distance between 2 strings --- replace, insert, delete, respectively. And the cost for each of them is assumed to be 1 unit. Given theses inforamtion, our primary goal is to find the sub-problem.

Let's say we are given two strings "be" and "bp"

| | x | y |
|---|---|---|
| j | a | b |
| k | c | |

If we want to figure out the number for the blank space, we should determine whether 'k' is equal to 'y' first. If so, the edit distance between "xy" and "jk" should be the edit distance between "j" and "k" + 0 which should be (a + 1),since k and y are equal. Otherwise, the edit distance between "xy" and "jk" should be the minimun value among (the edit distance e1 between "xy" and "j_" + 1) or (the edit distance e2 between "x_" and "j_" + 1) or (the edit distance e3 between "jk" and "x_" + 1).

For example, string A[0,5] = "shady", string B[0,5] = "windy". There are actually 4 ways to calculate the distance. The first solution is conditional, because it requires that the character at the current position '5' is equal to the other one. In our case, solution 1 is (A[0,4] to B[0,4]) + 0. Solution2 is by repacing. A[0,2] to B[0,2] can be broken down into (A[0,1] to B[0,1]) + 1(1 means replace 'a' to 'n'). Solution3 is executed by insersion. In this case, edit distance from A[0,2] to B[0,2] can be transformed to edit distance from A[0,2] to B[0,1]("sha" to "wi") + the insersion of B2. Solution4(deletion) is pretty similar to solution3, whereas the final step is replaced by deletion.

Therefore, the sub-problem should be : if(row[i] == row[j]) return matrix[i - 1][j - 1] else return MIN{(matrix[i-1][j] + 1),(matrix[i][j-1] + 1), (matrix[i-1][j-1] + 1)]

Exercise: Run a step-by-step execution of the edit-distance algorithm with the following two strings.

A = "Lunette", B = "Phoebe"

Solution: First of all, we are supposed to create a table where each pair represents the edit distance between A[0,i] to B[0,j]

| | **" "** | **L** | **u** | **n** | **e** | **t** | **t** | **e** |
|---|---|---|---|---|---|---|---|---|
| " " | | | | | | | | |
| P | | | | | | | | |
| h | | | | | | | | |
| o | | | | | | | | |
| e | | | | | | | | |
| b | | | | | | | | |
| e | | | | | | | | |

Initialize the first row and first column(the edit distance from " " to any string should be the length of the string itself)

| | **" "** | **L** | **u** | **n** | **e** | **t** | **t** | **e** |
|---|---|---|---|---|---|---|---|---|
| " " | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| P | 1 | | | | | | | |
| h | 2 | | | | | | | |
| o | 3 | | | | | | | |
| e | 4 | | | | | | | |
| b | 5 | | | | | | | |
| e | 6 | | | | | | | |

Then, we should build up the table according to the sub-problem

| | **" "** | **L** | **u** | **n** | **e** | **t** | **t** | **e** |
|---|---|---|---|---|---|---|---|---|
| " " | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| P | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| h | 2 | 2 | 2 | 3 | 4 | 5 | 6 | 7 |
| o | 3 | 3 | 3 | 3 | 4 | 5 | 6 | 7 |
| e | 4 | 4 | 4 | 4 | 3 | 4 | 5 | 6 |
| b | 5 | 5 | 5 | 5 | 4 | 4 | 5 | 6 |
| e | 6 | 6 | 6 | 6 | 5 | 5 | 5 | 5 |

And the final answer should be table[7,6] = 5.

### 7.3 knapsack (both 0-1 and unrestrained)

#### 1. 0/1 Knapsack

In this particular approach, we are given a bunch of items with their value and weight, respectively. And we need to find out the largest value with a certain combination of chosen items(each item can only be picked at most once) and the total weight of items should be less than the given amount.

In order to solve this problem, we can find out the sub-problem of it while we are building up the whole table.

Items:

| wt | val |
|---|---|
| 1 | 3 |
| 2 | 4 |
| 3 | 6 |
| 5 | 8 |

answer table S, while each row represents the weight and value of current item and each column repsents the given total weight.

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | | | | | | | | |
| 2/4 | | | | | | | | |
| 3/6 | | | | | | | | |
| 5/8 | | | | | | | | |

Initialization: the first column should be initialized as 0.

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | | | | | | | |
| 2/4 | 0 | | | | | | | |
| 3/6 | 0 | | | | | | | |
| 5/8 | 0 | | | | | | | |

Next, we try to fill out the first row. Since one item can only get picked once, all pairs would be updated as 3.

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2/4 | 0 | | | | | | | |
| 3/6 | 0 | | | | | | | |
| 5/8 | 0 | | | | | | | |

When it comes to the second row, we should determine whether the weight of current pickable item is larger than or equal to the given toal weight. In this case, S[2/4, 1] would still be 3 since its weight is larger than the total weight 1.

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2/4 | 0 | 3 | | | | | | |
| 3/6 | 0 | | | | | | | |
| 5/8 | 0 | | | | | | | |

When it comes to S[2/4, 2], let's say i stands for the row(weight and value for current item), and j stands for the column(given total weight), and the total value should be :

MAX(get_val(i) + get_val_prev[j - get_wt(i)], Get_val_prev(i - 1,j) = MAX(4 + 0, 3) = 4. Explanation: When current item can fit the current total weight, we should subtract the item weight from the total weight, and find the value with the item that is given the subtracted weight. And compare the above result with the result from previous row(the permutation excludes the current item)

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2/4 | 0 | 3 | 4 | | | | | |
| 3/6 | 0 | | | | | | | |
| 5/8 | 0 | | | | | | | |

Likewise, we can finish the second row with the algorithm mentioned above

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2/4 | 0 | 3 | 4 | 7 | 7 | 7 | 7 | 7 |

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 3/6 | 0 | 3 | 4 | | | | | |
| 5/8 | 0 | | | | | | | |

As for the third row, let's say we want to calculate the S[3/6, 3].

S[3/6, 3] = MAX(get_val(i) + get_val_prev[j - get_wt(i)], get_val_prev(i-1, j)) = MAX(6+0, 7) = 7. And the rest of the row can be finished similarly.

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2/4 | 0 | 3 | 4 | 7 | 7 | 7 | 7 | 7 |
| 3/6 | 0 | 3 | 4 | 7 | 9 | 10 | 13 | 13 |
| 5/8 | 0 | | | | | | | |

Actually, the approach to the last row is pretty similar to what is done before, I'll just skip the explanation process.

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2/4 | 0 | 3 | 4 | 7 | 7 | 7 | 7 | 7 |
| 3/6 | 0 | 3 | 4 | 7 | 9 | 10 | 13 | 13 |
| 5/8 | 0 | 3 | 4 | 7 | 9 | 10 | 13 | 13 |

### 2. unrestrained Knapsack

Slightly different from 0/1 Knapsack, an unrestrained Knapsack will not put a limit on the number of each single item.

Again, we are given the same items: Items:

| wt | val |
|---|---|
| 1 | 3 |
| 2 | 4 |
| 3 | 6 |
| 5 | 8 |

answer table S, while each row represents the weight and value of current item and each column repsents the given total weight.

Initialization:

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | | | | | | | |
| 2/4 | 0 | | | | | | | |
| 3/6 | 0 | | | | | | | |
| 5/8 | 0 | | | | | | | |

The first row: Since we can pick one item as many times as we want, the total value would be : S[j] / S[i.wt] * S[i.val]

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| 2/4 | 0 | | | | | | | |
| 3/6 | 0 | | | | | | | |
| 5/8 | 0 | | | | | | | |

When it comes to the second row, the value in column '1' should be copied from the previous row since the weight of current item is larget than the target weight.

And the value for the rest would be :

MAX(get_val(i) + get_val[j - get_wt(i)], get_val_prev(i - 1,j).

For example, S[2/4, 2] = MAX(4 + 0, 6) = 6

Explanation: When current item can fit the current total weight, we should subtract the item weight from the total weight, and find the value with the current item that is given the subtracted weight. And compare the above result with the result from previous row(the permutation excludes the current item)

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| 2/4 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| 3/6 | 0 | | | | | | | |
| 5/8 | 0 | | | | | | | |

The third row:

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| 2/4 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| 3/6 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| 5/8 | 0 | | | | | | | |

The last row:

| wt / val | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1/3 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| 2/4 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| 3/6 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| 5/8 | 0 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |

### 6.3 dynamic programming algorithm design

In this section, I will use the money change problem as an example again. But the way to solve it is dilivered by dynamic programming. Remember how it is solved in the previous section? We solve the problem by using greedy algorithm. And each time when we are picking a coin, we pick the qualified coin with the largest value in order to minimize the number of coins making change for the target. However, when it comes to the dynamic approach, we should figure out the sub-problem.

Let's say we are given an array of coins [1,2,5] and make change for 11 dollars. If we look at the last picked coin, we will have 3 choices : 1, 2, 5. And the cost for picking each of them is 1. Let's say we pick 5 as the last coin, and the total cost for making change for 11 is 1 + make_change(11 - 5). Similarly, we repeatedly do this on picking 2 and 1 as the last coin. And the final result is MIN(1 + make_change(10), 1 + make_change(9), 1 + make_change(6)). And this is just the sub-problem we are looking for.

Exercise: Use a list of coins given below to make change for 12. Run a step-by-step execution by dynamic programming. Coins: [1,2,5,7] total: 12

Solution:

First, we initializa a table indexed from 0 to 12 and fill out the table with INT_MAX

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

1. We start off from 0, and obviously none of coins can be picked to make change for 0.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

2. At index 1, we can only pick '1' from coins, and the total number of coins would be 1 + make_change( 1 - 0 ) = 1 + 0 = 1.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

3. At index 2, we can only pick '1' and '2" from coins, and the total number of coins would be MIN(pick'1' and pick'2') = MIN(1 + make_change(1), 1 + make_change(0)) = MIN(1 + 1, 1 + 0) = 1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

4. At index 3, we can only pick '1' and '2" from coins, and the total number of coins would be MIN(pick'1' and pick'2') = MIN(1 + make_change(2), 1 + make_change(1)) = MIN(1 + 1, 1 + 1) = 2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

5. At index 4, we can only pick '1' and '2" from coins, and the total number of coins would be MIN(pick'1' and pick'2') = MIN(1 + make_change(3), 1 + make_change(2)) = MIN(1 + 2, 1 + 1) = 2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

6. At index 5, we can pick '1', '2' and '5' from coins, and the total number of coins would be MIN(pick'1', pick'2' and pick '5') = MIN(1 + make_change(4), 1 + make_change(3), 1 + make_change(0)) = MIN(1 + 2, 1 + 2, 1 + 0) = 1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 1 | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

7. At index 6, we can pick '1', '2' and '5' from coins, and the total number of coins would be MIN(pick'1', pick'2' and pick '5') = MIN(1 + make_change(5), 1 + make_change(4), 1 + make_change(1)) = MIN(1 + 1, 1 + 2, 1 + 1) = 2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 1 | 2 | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

8. At index 7, we can pick '1', '2', '5' and '7' from coins, and the total number of coins would be MIN(pick'1', pick'2', pick '5' and pick '7') = MIN(1 + make_change(6), 1 + make_change(5), 1 + make_change(2), 1 + make_change(0)) = MIN(1 + 2, 1 + 1, 1 + 1, 1 + 0) = 1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | INT_MAX | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

9. At index 8, we can pick '1', '2', '5' and '7' from coins, and the total number of coins would be MIN(pick'1', pick'2', pick '5' and pick '7') = MIN(1 + make_change(7), 1 + make_change(6), 1 + make_change(3), 1 + make_change(1)) = MIN(1 + 1, 1 + 2, 1 + 2, 1 + 1) = 2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | 2 | INT_MAX | INT_MAX | INT_MAX | INT_MAX |

10. At index 9, we can pick '1', '2', '5' and '7' from coins, and the total number of coins would be MIN(pick'1', pick'2', pick '5' and pick '7') = MIN(1 + make_change(8), 1 + make_change(7), 1 + make_change(4), 1 + make_change(2)) = MIN(1 + 2, 1 + 1, 1 + 2, 1 + 1) = 2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | 2 | 2 | INT_MAX | INT_MAX | INT_MAX |

11. At index 10, we can pick '1', '2', '5' and '7' from coins, and the total number of coins would be MIN(pick'1', pick'2', pick '5' and pick '7') = MIN(1 + make_change(9), 1 + make_change(8), 1 + make_change(5), 1 + make_change(3)) = MIN(1 + 2, 1 + 2, 1 + 1, 1 + 2) = 2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | 2 | 2 | 2 | INT_MAX | INT_MAX |

12. At index 11, we can pick '1', '2', '5' and '7' from coins, and the total number of coins would be MIN(pick'1', pick'2', pick '5' and pick '7') = MIN(1 + make_change(10), 1 + make_change(9), 1 + make_change(6), 1 + make_change(4)) = MIN(1 + 2, 1 + 2, 1 + 2, 1 + 2) = 3

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | 2 | 2 | 2 | 3 | INT_MAX |

13. At index 12, we can pick '1', '2', '5' and '7' from coins, and the total number of coins would be MIN(pick'1', pick'2', pick '5' and pick '7') = MIN(1 + make_change(11), 1 + make_change(10), 1 + make_change(7), 1 + make_change(5)) = MIN(1 + 3, 1 + 2, 1 + 1, 1 + 1) = 2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | 2 | 2 | 2 | 3 | 2 |

Therefore the number of coins needed for makeing change of 12 is 2.

## Chapter 8: Linear Programming

1. Technique definition In computer science, many problems we want to solve are optimization tasks. If all equations for this problem is linear and our goal is to maximize or minimize the target funtion with the given constraints, that is the definition of linear programming. In general, there are two prerequisites for linear programming : (1) Optimize a given linear function. (2) all given variables should satisfy all the equations or inequations of the problem.
2. problem specifications In order to solve real-world problems with linear programming, we have to convert the problem description into our linear equations and inequations first and hence we have to extract variables and functions from the problem. In short, we are supposed to figure out the constraints and funtion to be optimized.

Exercise: A protein pod manufacturer is mixing two types of protein, brand X and brand Y for the new product. If each pod is required to contain at least 20 grams of protein and 100 milligrams of sodium, where brand X has 10 grams of protein and 70 milligrams of sodium which cost 50 cents per unit and brand Y contains 15 grams of protein and 90 milligrams of sodium which cost 90 cents per unit. How much of each brand should be added to minimize the total cost for the new product?

Solution:

To begin with, we have to make the problem more specific by translating the description into constraints and variables.

Let's say the number of units of brand X is x while the number of units of brand Y is y.

The constraints would be :

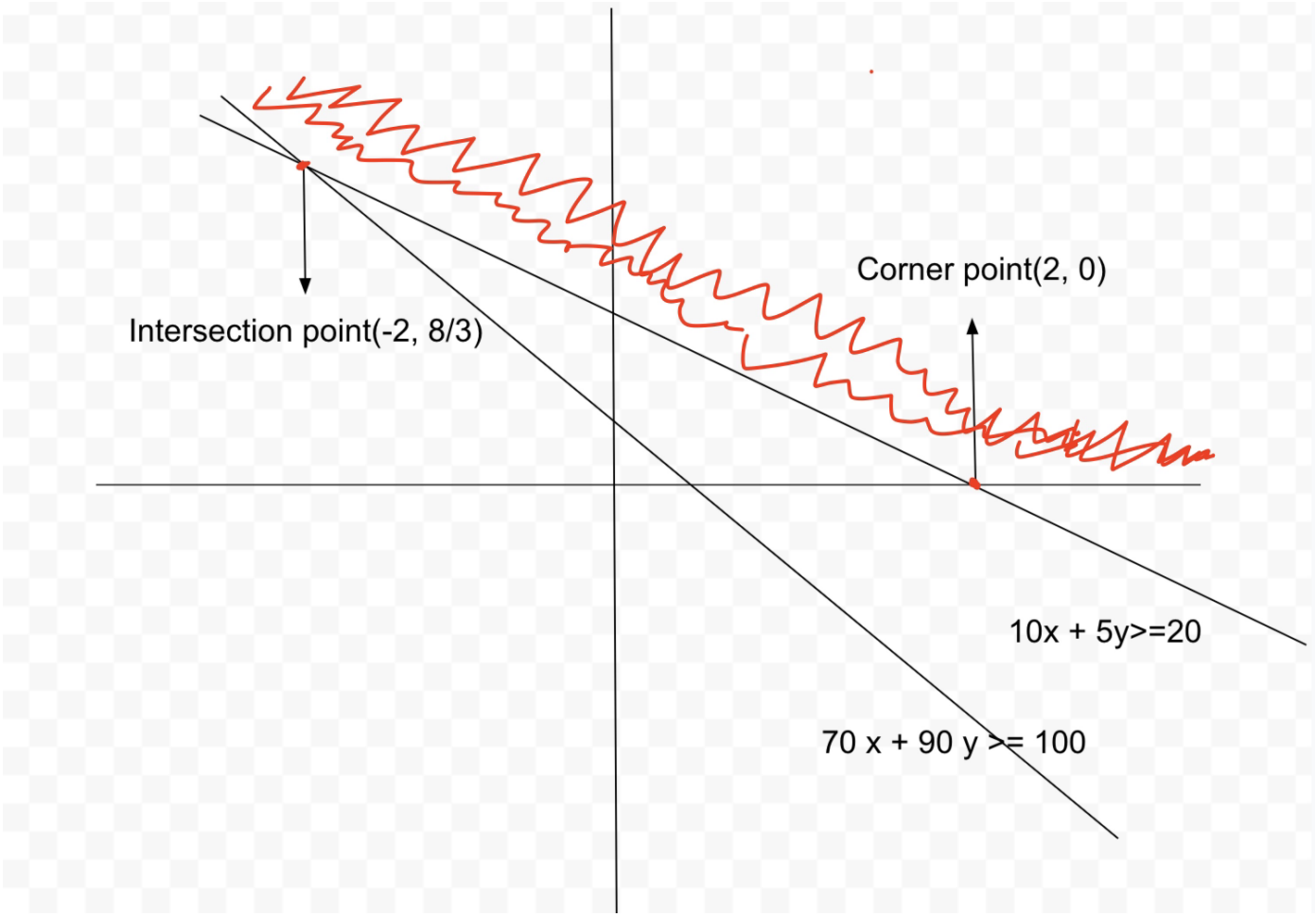
10x + 15y >=20

70x + 90y >= 100

x >= 0, y >= 0

The cost to be minimized : 0.5 x + 0.9 y

Next, we should draw the boundaries for the possible area and find the minimal value with all the corner points




For the intersection point(corner point as well): Profit p1 = 0.5 * (-2) + 0.9 * 8/3 = 1.4

For the corner point(2,0): Profit p2 = 0.5 * 2 + 0.9 * 0 = 1

Therefore, the minimal cost for the new product is 1 dollar, given the number of units of brand A is 2, number of units of brand B is 0.

## Chapter 9 : NP - Completeness

### 9.1 Definition

Before we get to know the definition of NP - Completeness, we should have a basic understanding of the complexity theory. In general, the types of complexity are divided into two levels: the one is something like O(n) and the other is like O($2^n$) where the latter is in any case much larger than the former: one is O(1),O(log(n)),O(n^a), etc., and we call it polynomial-level complexity because its scale n appears at the bottom; the other is O(a^n) and O(n!) type complexity, which is non-polynomial-level and computers can't often handle them. When we are solving a problem, the algorithm we choose usually needs to be of polynomial complexity; non-polynomial complexity takes too much time and tends to time out, unless the data size is very small. And unfortunately, not all the problems can be solved within a polynomial complexity.

Before we step further into some details, we should know the definition of P problems and NP problems. The exact definition for a P problem is : A problem can be solved by a solution which takes a polynomial time complexity. NP problems are problems for which a solution can be verified in polynomial time. For example, I'm taking a guess for the answer of a certain hard problem and fortunately the answer turns out to be true after a verification done in polynomial time. Obviously all problems belong to NP problems since if you can solve a P problem in polynomial time, there's no doubt you can verify the answer of it in polynomial time.

Finally, here comes with the definition of a NP - Completeness(Non-deterministic):First, it has to be an NP problem; then, all NP problems can be reduced to it. Proving that a problem is an NPC problem is also simple. First prove that it is at least an NP problem, and then prove that one of the known NPC problems can be reduced to it.

To recap :

Definition 1 : A problem $S \in P$ if it can be solved in polynomial time.

Definition 2 : A problem $S \in NP$ the answer can be verified in polynomial time

Definition 3: A problem $S \in NP - Complete$ if

1. $S \in NP$
2. Every problem $S' \in NP$ can be reduced to S in polynomial time

To sum up, the relationship among P, NP and NP-complete can be : P $\in$ NP $\in$ NP-C

### 9.2 SAT and 3 SAT

Before we get to know the definition of SAT and 3SAT, we should gain the awareness of several additional definitions.

Definition 1. A literal is either a $\overline{x_i}$ or $x_i$.

Definition 2. A Boolean clause is an OR operation on literals($x_1 \bigvee \overline{x_2} \bigvee x_3$)

Definition 3. A boolean expression is called in Conjunctive Normal Form(CNF) if the expression is an AND operation on Boolean clauses.($x_1 \bigvee \overline{x_2} \bigvee x_3$) $\bigwedge$ ($x_2\bigvee x_4$)

Definition 4. A boolean expression is called in 3-CNF if the expression is an AND operation on Boolean clauses with an exact length of 3 individually.($x_1 \bigvee \overline{x_2} \bigvee x_3$) $\bigwedge$ ($x_2\bigvee x_4 \bigvee x_6$)

A SAT problem(Boolean Satisfiability Problem) requires us to determine whether there's a way of satisfifying a boolean expression in CNF where all variables can be valued as a TRUE or FALSE.

Likewise, A 3 SAT problem requires us to determine whether there's a way of satisfying a boolean expression in 3-CNF where all variables can be valued as a TRUE or FALSE.

Exercise:

1. Prove the given boolean expression is satisfiable. ($x_1 \bigvee \overline{x_2} \bigvee x_3$) $\bigwedge$ ($x_2 \bigvee \overline{x_4}$) $\bigwedge$ ($x_4 \bigvee \overline{x_1}$)

   Solution: Since what we want as a result is a TRUE on the output of the given boolean expression, we have to make sure that the 3 clauses are all true.

   In this case, we have to make sure that at least one literal in each clause should be a TRUE.

   For the first clause: Let's say we let $x_1$ ot be true, and $\overline {x_2}$ and $x_3$ can take any value.

   For the second clause: Let's assume $x_2$ to be true, and $\overline {x_4}$ can take any value.

   For the third clause: Since $x_1$ is already true, $\overline{x_1}$ would be false. But it does not affect the final result as long as we set $x_4$ to be true.

   Therefore, if $x_1$ = true, $x_2$ = true and $x_4$ = true, the output of the given expression in CNF can be true and hence it is satisfiable.

2. Prove the given boolean expression (3-CNF) is satisfiable. ($x_1 \bigvee \overline{x_2} \bigvee x_3$) $\bigwedge$ ($x_2 \bigvee \overline{x_4} \bigvee \overline{x_1}$) $\bigwedge$ ($x_2 \bigvee \overline{x_3} \bigvee x_4$)

   Solution: pretty much the same as what we have done before, we just need to make sure the assinment of variables leads to an output of TRUE.

   For the first clause: Let's say we have $x_1$ to be true and the first clause is hence to be true.

   For the second clause: Let's say we have $x_2$ to be true, given that $x_1$ is true. In this case the second clause is obviously true.

   For the last clause: Let's say we have $x_3$ to be false, given that $x_1$ and $x_2$ to be true. In this case, the last clause turns out to be true.

   Therefore, the given expression is satisfiable.

### 9.3 Proving NP - Completeness

Exercise :

Independent Set Problem: Given a graph G, does G contain an independent set of size J?

Prove the given problem above is NP - complete.

Solution: According to the Definition 3 in section 9.2:

A problem $S \in NP - Complete$ if

1. $S \in NP$
2. Every problem $S' \in NP$ can be reduced to S in polynomial time

To begin with, we should prove the Independent Set Problem is a NP: Suppose we are given a graph G with M vertives and N edges. the solution to the given independent set problem is G' with J vertices.

In order to verify the solution to the problem, we should check whether each pair of vertices in G' exists in the edges of G:

There are $O(M^2)$ possible pairs of vertices in G'. To verify whether each of pairs exists in the edge set of G, we need to do it N times since there are N edges in G.

The total time to verify the answer is $O(M^2N)$ which turns out to be polynomial and hence prove the problem $\in$ NP.

Next, we are supposed to prove Every problem $S' \in NP$ can be reduced to S in polynomial time. This approach seems a bit tricky in essense, since it's terrible for us to prove all problems in NP can be reduced to S in polynomial time. But if we prove a NP - compeleteness can be reduced to the Independent Set Problem, it can yields the same result because a NP - complete problem can be reduced to any NPs.

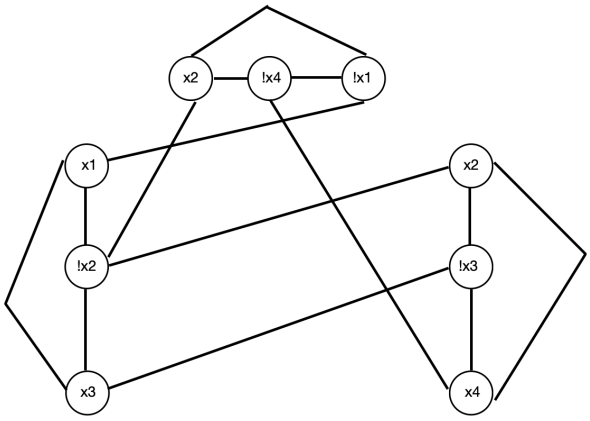
For the sake of simplicity, we choose the 3 SAT problem as the NP-completeness to prove our answer.

3 SAT Problem : Given a boolean exression in 3-CNF, is there any way of satisfying it?

Let's say we are given a boolean expression $\psi$ = ($x_1 \bigvee \overline{x_2} \bigvee x_3$) $\bigwedge$ ($x_2 \bigvee \overline{x_4} \bigvee \overline{x_1}$) $\bigwedge$ ($x_2 \bigvee \overline{x_3} \bigvee x_4$) and we express $\psi$ as a graph so that if $\psi$ is satisfiable, the given graph does have an independent set.

The following steps would be creating a graph using the literals and relations:

1. Create a vertex for each literal in the expression
2. Connect all literals one-by-one in each clause
3. Connect each literal to its negation.



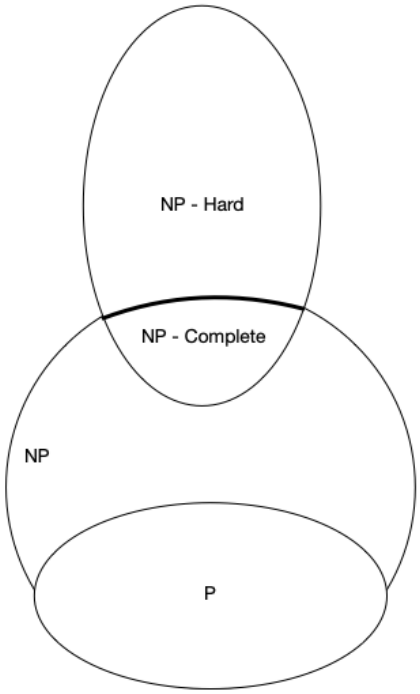

From the graph drawn above , we can easily find out an independent set : $x_1$, $x_4$, $x_2$.

Note that the independent set has a size of 3 and $\psi$ is satisfiable only when the clause has exacly 3 literals. And when we pick $x_1$, $x_4$, $x_2$ as the independent set, they are both assumed to be true. And we can verify that $\psi$ is true after the assignment of the given literals. The graph can be constructed in polynomial time.

Therefore, the Independent Set Problem is NP - Complete.

### 9.4 Getting around NP - Completeness

In addition to NP - Completeness, there are problems that are the same hard as NP - Compeleteness problems (even harder). They are called NP - Hard. The definition of a NP - Hard problem is that a NP - Complete problem can be reduced to it in polynomial time. If it is a NP - Hard problem, it should also belong to NP - Completeness. Here's a diagram of the relationship among P, NP, NP - Compeleteness, NP - Hard:



In the end, why do we need to dig into the NP - Completeness? The answer is simple. If we find a polynomial solution to a NP - Completeness problem, all NP prolems can be solved in polynomial time according to the relationship between them.