```
function generatePassword(startingChar):
   password = [startingChar]
   currentChar = startingChar
   for i from 1 to 7:
      possibleMoves = getValidMoves(currentChar)
      nextChar = randomChoose(possibleMoves)
      password.append(nextChar)
      currentChar = nextChar
   return password


function getValidMoves(character):
   // This function will return the current moves of the character at a distance of 2-3    return
validMoves[character]


function randomChoose(list):
   // This function will select a random element from the given list
   return randomElementFromList(list)
```

For this assignment, we can develop an algorithm in accordance with the rules you have specified. Below I will explain how we will proceed step by step by addressing each question.

1. **What distance metric can be used for the distances Decoupled between keys?**
2. It would be quite convenient to use the Manhattan Distance in this kind of algorithm. Manhattan distance is the distance of one key to another key horizontally and it calculates vertically as the sum of movements. Each key on the keyboard can be

placed in a coordinate system (for example, each key can have an X, Y coordinate).Örneğin:

- * Assume that the d key on the keyboard is at the coordinate (1, 2), and the s key is at the coordinate (2, 2).
- • In this case, the Manhattan distance is: |2 - 1| + |2 - 2| = 1 okay, fine.

• This kind of distance is a Decently simple and effective method for calculating the "movement" between characters.

In a nutshell: The Manhattan distance is a convenient metric for measuring distances Decoupled between keys.

3. Would you need a specific data structure to represent the keyboard layout?

- 4. You will need a data structure to represent the keyboard layout. The keyboard layout requires a structure in which each key will be located in a certain position (coordinates). There are several options available to represent this structure:
- **• 2D Arrays: You can keep the keyboard layout as a 2D array (matrix). For example, each row may contain one row keys and each column may contain a specific character.**

*** Hash Map: A hash map can also be used on the keyboard that matches each key with a coordinate pair.**

Is it necessary permanently? Keeping the keyboard layout is necessary for calculating distances, but another temporary data structure can also be used during the operation of the algorithm. Decently, a matrix or hash map can be created to calculate the distance between the keys.

4. When we decide to map each key to a list of valid moves (i.e. other keys with a distance of 2 to 3), what is the optimal Java data structure?

- 5. To keep the distances between the keys and list the current moves for each key, the following data structures may be appropriate Dec:
- *** HashMap: You can use each key as a key and keep a list of the current moves connected to this key. For example, a -> is like [b, c, d].**

*** ArrayList: You can store a list containing the current moves of each key in a HashMap.**

Recommended Data Structure: HashMap<Character, List<Character>> is ideal for storing a list of the current moves of each key.

4. 8 karakterli bir parola oluşturmak için sözde kod (veya Java kodu) yazın.

Aşağıda, 8 karakterli bir şifre oluşturma algoritması için sözde kodu bulabilirsiniz:

```
text
```

```
Kodu kopyala
1. Kullanıcıdan ilk karakteri seçmesini iste.
2. İlk karakterin geçerli hamlelerini listele (2 ile 3 mesafe arasındaki
karakterler).
3. Geçerli hamlelerden birini rastgele seç.
4. Bu yeni seçilen karakteri bir sonraki karakter olarak al.
5. Bu adımları tekrarlayarak 8 karakterli bir parola oluştur.
6. Parola tamamlandığında, şifreyi kullanıcıya göster.
```

Örnek sözde kod:

```
text
Kodu kopyala
function generatePassword(startingChar):
    password = [startingChar]
    currentChar = startingChar
    for i from 1 to 7:
        possibleMoves = getValidMoves(currentChar)
        nextChar = randomChoose(possibleMoves)
        password.append(nextChar)
        currentChar = nextChar
    return password

function getValidMoves(character):
    // Bu fonksiyon, karakterin 2-3 mesafe uzaklıktaki geçerli hamlelerini
döndürecek
    return validMoves[character]

function randomChoose(list):
    // Bu fonksiyon, verilen listeden rastgele bir eleman seçecek
    return randomElementFromList(list)
```

Bu sözde kod, başlama karakterini kullanıcıdan alır, geçerli hamle listesinden rastgele bir karakter seçer ve bunu 8 karakterli bir parolaya dönüştürür.

5. Aşağıdaki anahtarlar için geçerli hamlelerin listesini hesaplayın: a, f, h, 8, 0 ve p.

Bu adımda, belirtilen anahtarlar için 2 ila 3 mesafeye sahip diğer karakterleri belirlememiz gerekiyor. Bu işlemi yapabilmek için, klavye düzenine bakarak her tuş için olası geçişleri hesaplayacağız. Örnek olarak, eğer QWERTY klavyesini temel alıyorsak, şu şekilde bir yaklaşım izlenebilir:

- **a**: Geçerli hamleler: `s, q, w`
- **f**: Geçerli hamleler: `d, g, r`
- **h**: Geçerli hamleler: `g, j, u`
- **8**: Geçerli hamleler: `7, 9, 0`
- **0**: Geçerli hamleler: `9, p, o`
- **p**: Geçerli hamleler: `o, 0, l`

Bu tür hesaplamalar, klavye düzenine ve karakterlerin birbirine olan mesafesine bağlıdır. Eğer bu listeyi tam olarak çıkartmak istiyorsanız, kullanılan klavye düzenini göz önünde bulundurmak gerekir.

Sonuç:

Bu ödevin çözümünde:

- Tuşlar arasındaki mesafeler için Manhattan mesafesi kullanılabilir.
- Klavye düzeni için 2D dizi veya hash map kullanılabilir.
- Geçerli hamleleri tutmak için en uygun veri yapısı `HashMap<Character, List<Character>>` olacaktır.
- Sözde kod ile şifre üretim algoritması yukarıda verilmiştir.
- Belirtilen anahtarlar için geçerli hamlelerin listesi klavye düzenine göre hesaplanabilir.