Relatório
1640. O Problema da Contagem

A primeira abordagem ao problema foi tratar da maneira mais simples a contagem de cada ocorrência dos dígitos: listar todos os números dentro do intervalo e para cada um realiza-se a contagem de forma simples usando o “mod” para separar cada um dos algarismos e computá-los. O trecho do código a baixo mostra o 'core' da solução que só não inclui como definir o limite superior e inferior e a leitura do arquivo.

```
//dado limiteA e limiteB, sendo que limiteA < limiteB
static String solucaoSimplista(int limiteA, int limiteB){
    long[] ocorrencias = new long[10];

    //iteração sobre todos os itens do intervalo
    for(int i = limiteA;  i <= limiteB; i++ ){
        int iCopia = i;
        while(iCopia > 0){
            int sobra = iCopia%10; //separa o ultimo algarismo

            ocorrencias[sobra] += 1; //conta a ocorrencia do numero separado
            iCopia = iCopia/10;
        }
    }

    String saida = "";
    for(int i = 0; i < ocorrencias.length; i++){
        saida += ocorrencias[i] + " ";
    }

    return saida.trim();
}
```

O problema com essa solução é o custo. O custo de iterar sobre os *n* item já é alto o bastante, mas o custo de iterar sobre os m dígitos de cada um dos números aumenta ainda mais tornando a solução de complexidade *O(n*m)*. Ao submeter essa solução, o tempo limite para os testes no julgamento do *UVa Online Judge* foi esgotado, embora tenha acertado todos os casos testes feitos.

A segunda ideia foi usar técnicas de programação dinâmica. Para cada intervalo dado, conferir se um subintervalo dele já foi calculado anteriormente e calcular o resto do intervalo da mesma maneira que foi proposta para a primeira solução.

Como por exemplo: dado os inteiros 0 e 100, caso não houver nenhum intervalo limitado por *a* e *b,* sendo que 0 <= a, b <= 100, as ocorrências são calculadas listando todos os números entre 0 e 100.

Caso haja, por exemplo, o intervalo limitado por 30 e 70, só precisamos calcular as ocorrências dos intervalos [0, 30[e depois]70, 100], e depois somar com as ocorrências já calculadas no intervalo [30, 70]. Abaixo seguem as partes mais importantes da solução que implementa essa estratégia:

```
  static Map<Integer, Intervalo> intervalosCalculados = new HashMap<>();
  static String solucaoProgramacaoDinamica(int limiteA, int limiteB){
      long[] ocorrencias = new long[10];

      for(int i = limiteA;  i <= limiteB; i++ ){
          //Conferir se um intervalo a partir de i já foi calculado
          Intervalo intervalo = intervalosCalculados.get(i);
          if(intervalo != null){
              //Subintervalo já calculado
              int intervaloLimiteB = intervalo.getLimiteB();
              if(intervaloLimiteB < limiteB){
                  //adicionar as ocorrencias do intervalo já calculado as ocorrencias corrente
                  long[] intervaloOcorrencias = intervalo.getOcorrencias();
                  for(int j = 0; j < ocorrencias.length; j++){
                      ocorrencias[j] += intervaloOcorrencias[j];
                  }

                  i = intervaloLimiteB;
                  continue;
              }
          }

          int iCopia = i;
          while(iCopia > 0){
              int sobra = iCopia%10;

              ocorrencias[sobra] += 1;
              iCopia = iCopia/10;
          }
      }
```

```
          String saida = "";
          for(int i = 0; i < ocorrencias.length; i++){
              saida += ocorrencias[i] + " ";
          }

          //adicionar intervalo as Ocorrencias dos Intervalos Calculados
          Intervalo intervaloCorrente = intervalosCalculados.get(limiteA);
          if(intervaloCorrente == null){
              intervaloCorrente = new Intervalo(limiteA, limiteB, ocorrencias);
              intervalosCalculados.put(limiteA, intervaloCorrente);
          } else{
              if(intervaloCorrente.getLimiteB() < limiteB){
                  intervaloCorrente.setLimiteB(limiteB);
                  intervaloCorrente.setOcorrencias(ocorrencias);
                  intervalosCalculados.put(limiteA, intervaloCorrente);
              }
          }


          return saida.trim();
      }
  }

  class Intervalo{
      private int limiteA;
      private int limiteB;
      private long[] ocorrencias;

      public Intervalo(int limiteA, int limiteB, long[] ocorrencias){
          this.limiteA = limiteA;
          this.limiteB = limiteB;
          this.ocorrencias = ocorrencias;
      }
```

No melhor cenário, esse intervalo já foi calculado ou, imaginando que não há requisição de mesmos intervalos, que apenas os números que delimitam o intervalo precisam ter suas ocorrências calculadas. A complexidade seria O(n), sendo n a soma dos algarismos dos dois limites. No pior dos casos, nenhum intervalo é requisitado é sub limite do outro e portanto caímos dentro do caso da primeira abordagem com a mesma complexidade O(n*m).

Por ainda ter um custo tão caro no pior dos casos, essa solução não foi aceita também pelo *UVa*, pela mesma razão da primeira solução *Runtime error*, apesar de ter passado nos casos de testes do enunciado e alguns outros elaborados.

A solução evoluiu para uma terceira ideia que diminuiu a complexidade para todos os casos. A ideia foi calcular quantos mini intervalos (de 0 a 10) já previamente calculados cabem no intervalo que desejamos calcular as ocorrências. Dessa forma, só precisamos iterar sobre os números que são a sobra e não puderam ser cobertos pelos mini intervalos.

Por exemplo, no intervalo [17, 137], primeiro iteramos sobre os números seguintes ao limite inferior até esse ser divisível por 10 calculando do modo simplista as ocorrências e depois iteramos sobre os números anteriores ao limite superior até ser divisível por 10 para calcular do mesmo jeito também.

Após esses dois passos, ainda precisamos calcular as ocorrências no intervalo de [20, 130]. A partir daqui podemos usar a estratégia dos mini intervalos de 0 a 10. Vemos que nesse novo intervalo cabem (130 - 20)/10 = 11 mini intervalos inteiros de 0 a 10, por tanto, podemos concluir que na unidade ocorrem [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] x 11 de cada digito.

Depois de calcular todas as ocorrências nas unidades, podemos “retirá-las” do nosso intervalo, passando a encarar que precisamos calcular apenas o intervalo [2, 13] e que cada adição a partir de agora no número de ocorrências precisamos multiplicar por dez. Para calcular esse novo intervalo, repetimos o mesmo processo do começo do exemplo, até que os dois limites atinjam 0 e 0.

Dessa maneira, podemos imaginar que vamos repetir esses passos para cada intervalo *n* vezes, sendo *n* o número de algarismos do maior dos limites. Nesse problema o número máximo de *n* é oito, pois a e b < 100000000.

O primeiro passo, para tornar os dois limites divisíveis por 10, se repete no máximo 9 vezes para cada um dos limites. O segundo passo é constante já que o número de algarismos que se quer contabilizar as ocorrências é sempre dez.

No máximo fazemos *n**9*11 iterações. Portanto podemos dizer que a complexidade dessa solução é O(n). As próximas imagens mostram como fica o algoritmo para essa solução:

```
static String solucaoAlternativa(int limiteA, int limiteB){
    long[] ocorrencias = new long[10];

    int ordem = 1;

    while(limiteA > 0 || limiteB > 0){
        while(limiteA <= limiteB && limiteA%10 != 0){
            for(int i = limiteA; i > 0; i = i/10){
                int sobra = i%10;
                ocorrencias[sobra] += 1 * ordem;
            }

            limiteA++;
        }

        while(limiteA <= limiteB && limiteB%10 != 0){
            for(int i = limiteB; i > 0; i = i/10){
                int sobra = i%10;
                ocorrencias[sobra] += 1 * ordem;
            }

            limiteB--;
        }
        if(limiteB%10 == 0){
            for(int i = limiteB; i > 0; i = i/10){
                int sobra = i%10;
                ocorrencias[sobra] += 1 * ordem;
            }
        }


        int gap = (limiteB - limiteA)/10;
        for(int i = 0; i < ocorrencias.length; i++){
            ocorrencias[i] += gap * ordem;
        }
```

```
        limiteA = limiteA/10;
        limiteB = limiteB/10 - 1;

        ordem = ordem*10;
    }

    String saida = "";
    for(int i = 0; i < ocorrencias.length; i++){
        saida += ocorrencias[i] + " ";
    }

    return saida.trim();
}
```