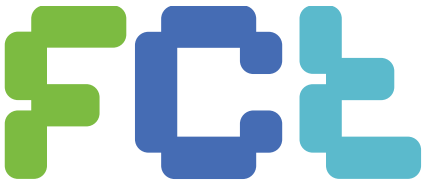

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Análise e Desenho de Algoritmos

Relatório do 1º Trabalho

Ana Garcia, MIEI 46783

Março 2015

Conteúdo

1 Apresentação do Problema

Team Formations é um problema de combinações, onde as ocupações estão delimitadas, isto é, em vez de um simples problema combinatório, é nos colocado o desafio de achar as combinações possíveis, para espaços delimitados por extremos.

A ideia inicial consiste na colocação de jogadores em campo, *P players*, em posições delimitadas por número mínimo, ou máximo, de jogadores. Neste caso, teremos então *R roles*, ou seja, R número de posições delimitadas por uma regra.

O nosso objectivo, é então, o de achar o número de diferentes formas possíveis, de colocar os jogadores em campo.

2 Resolução do Problema

Inicialmente foi preciso uma leitura mais cuidada, afim de obter e organizar a informação dada no enunciado do problema.

Após a leitura, sabemos então que temos duas variáveis, P jogadores e R regras, onde existe pelo menos um jogador, até um máximo de 50, e existe pelo menos uma regra, até um máximo de 20 regras. Também nos é dito que as regras de delimitação podem ser mínimo/máximo de 0 jogadores até o número de jogadores, P.

Numa primeira abordagem, vamos eliminar os mínimos, retirando ao número de jogadores a quantidade existente nas restrições de mínimos. Se no fim obtivermos um número de jogadores menor que 0, não existe qualquer tipo de combinação possível, o nosso algoritmo termina, e envia 0 como resultado.

Ao mesmo tempo foi criado um vector de máximos para cada regra, sendo que, onde existia um mínimo, existe agora um máximo com o número 50, sendo este o número máximo de jogadores.

Neste ponto, temos então uma simplificação do problema inicial, apenas com delimitações do tipo máximos.

Colocamos então a seguinte questão, existe algum algoritmo eficiente para a resolução deste problema? E a resposta é não. É necessário calcular tudo, vejamos então a recursividade deste problema.

$$f(vecMax, role, players) = \begin{cases} 1 & \text{if } players = 0 \\ 1 & \text{if } role + 1 = |vecMax| \wedge vecMax[role] \geq players \\ 0 & \text{if } role + 1 = |vecMax| \wedge vecMax[role] < players \\ \sum_{i=players-vecMax[n]}^{players} f(vecMax, role + 1, i) & \text{else} \end{cases}$$

Foi contruido um algoritmo, o qual é apresentado no Anexo A, de modo a apresentar uma tabela de R*P, número da Regra por número de Jogadores.

Utilizando **programação dinâmica**, foi então refeito o algoritmo, onde se inicializou a primeira coluna a 1, e a primeira linha a 1 ou 0 de acordo com o seu máximo.

Assim foi preenchido um vector bidimencional, pela ordem linha, coluna, onde: se a posição players(jogadores) não ultrapassar o máximo, soma a posição de linha anterior, com a posição de coluna anterior. Caso ultrapasse, a esta soma é subtraida a posição da linha anterior coluna da diferença da quantidade de players (jogadores) com máximo desta posição -1 (a contagem das colunas começa em 0, dai ser necessário subtrair 1 a esta diferença).

$$f(role, players) = \begin{cases} 1 & \text{if } players = 0 \\ 1 & \text{if } role = 0 \wedge vecMax[role] \geq players \\ 0 & \text{if } role = 0 \wedge vecMax[role] < players \\ f(role-1, players) + f(role, players-1) & \text{if } role! = 0 \wedge players! = 0 \wedge vecMax[role] \geq players \\ f(role-1, players) + f(role, players-1) \\ -f(role-1, players - vecMax[role] - 1) & \text{if } rol!e = 0 \wedge players! = 0 \wedge vecMax[role] < players \end{cases}$$

3 Implementação do Algoritmo

Na estrutura do programa foi usada apenas uma classe Main, com uma função que calcula o número de combinações possíveis, long possible_combinations(int[] vecMax, int players).

As estruturas de dados utilizadas neste programa foi um vector unidimencional de inteiros, para guardar os máximos das roles (regras), cujo o tamanho é o número de roles(regras); e um vector bidimencional de longs, para guardar o cálculo de combinações para cada posição, cujo o tamanho é o número de roles(regras) por número de jogadores(players).

4 Análise do Algoritmo

Tomando R como número de roles (regras) e P como número de players (jogadores).

A complexidade espacial desta implementação é Θ(R) para o vector de Máximos e Θ(R*P) para o vector bidimencional de resultados.

A complexidade temporal é de Θ(R) na leitura das roles localizada na main, e de Θ(R*P) na função possible_combinations. No total temos uma combinação temporal de Θ(R*P).

5 Conclusões

Em suma, a programação dinâmica deste problema, reduziu bastante o custo de complexidade, deixando de ser exponencial para ser polinomial. Existe outra solução estudada, passando por uma recursividade controlada por memorização, mas por nao acrescer qualquer valor, não foi implementada.

6 Anexo A

```java

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Main_exponencial {

  private static final int MAX_X = 50;

  public static void main(String[] args) throws IOException {

    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

    int players = Integer.parseInt(in.readLine());

    int nRoles = Integer.parseInt(in.readLine());
    int[] vecMax = new int[nRoles];

    // leitura das roles, eliminacao dos minimos
    for (int i = 0; i < nRoles; i++) {

      String[] role = in.readLine().split("␣");

      if (role[0].equals("MIN")) {
        int min = Integer.parseInt(role[1]);
        players -= min;
        vecMax[i] = MAX_X;

      } else {
        int max = Integer.parseInt(role[1]);
        vecMax[i] = max;
      }

    }

    if (players < 0) {
      System.out.println("0");

    } else {

      for (int len = 1; len <= vecMax.length; len++) {
        for (int pl = 0; pl <= players; pl++) {
          long result = possible_combinations(vecMax, len, 0, pl);
          System.out.print(result + "\t");
      }
      System.out.println();
    }
  }

}
```

```
1
2   /**
3   *
4   * @param vecMax
5   *            vetor de maximos
6   * @param n
7   *            posicao actual do vetor
8   * @param players
9   *            numero de jogadores
10  * @return combinacoes possiveis
11  */
12
13  public static long possible_combinations(int[] vecMax, int len, int n, int players) {
14    long result = 0;
15
16    if (players == 0) {
17      result = 1;
18    }
19    // estou na ultima posicao do vec?
20    else if (n == len - 1) {
21      // posso por todos os jogadores?
22      if (vecMax[n] >= players) {
23        result = 1;
24      } else {
25        result = 0;
26      }
27    } else {
28      // i de sumario
29      int i = Math.max(0, players - vecMax[n]);
30
31      for (; i <= players; i++) {
32        result += possible_combinations(vecMax, len, n + 1, i);
33      }
34
35    }
36
37    return result;
38  }
39 }
```

7 Código Fonte Completo

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Main {
  private static final int MAX_X = 50;

  public static void main(String[] args) throws IOException {

    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

    int players = Integer.parseInt(in.readLine());

    int nRoles = Integer.parseInt(in.readLine());
    int[] vecMax = new int[nRoles];

    // leitura das roles, eliminacao dos minimos
    for (int i = 0; i < nRoles; i++) {

      String[] role = in.readLine().split("␣");

      if (role[0].equals("MIN")) {
        int min = Integer.parseInt(role[1]);
        players -= min;
        vecMax[i] = MAX_X;

      } else {
        int max = Integer.parseInt(role[1]);
        vecMax[i] = max;
      }

    }

    if (players < 0) {
      System.out.println("0");

    } else {

      long result = possible_combinations(vecMax, players);
      System.out.println(result);

    }

  }
```

```
1
2   private static long possible_combinations(int[] vecMax, int players) {
3
4     int nRoles = vecMax.length;
5     long vecResult[][] = new long[nRoles][players + 1];
6
7     int max = vecMax[0];
8     // inicializar a primeira linha
9     for (int p = 0; p <= players; p++) {
10      if (p <= max) {
11        vecResult[0][p] = 1;
12      } else {
13        vecResult[0][p] = 0;
14      }
15    }
16
17    for (int r = 1; r < nRoles; r++) {
18      max = vecMax[r];
19      // inicializacao da coluna 0
20      vecResult[r][0] = 1;
21
22      for (int p = 1; p <= players; p++) {
23        vecResult[r][p] = vecResult[r - 1][p] + vecResult[r][p - 1];
24        if (p > max) {
25          vecResult[r][p] -= vecResult[r - 1][p - max - 1];
26        }
27      }
28    }
29    return vecResult[nRoles - 1][players];
30  }
31 }
```