Programação Dinâmica de Dígitos (Digit DP)
Professor Wladimir

Programação Dinâmica de Dígitos (Digit DP)

A **Programação Dinâmica de Dígitos** é uma técnica que permite resolver problemas que envolvem a contagem ou o somatório de propriedades específicas relacionadas aos dígitos dos números dentro de um intervalo $[L, R]$.

Por exemplo, considere o problema de encontrar a soma de todos os dígitos de todos os números no intervalo $[L, R]$, para $0 \leq L < R \leq 10^{15}$.

Para $L = 1$ e $R = 13$, temos que a soma dos dígitos de todos os números é igual a:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + (1 + 0) + (1 + 1) + (1 + 2) + (1 + 3) = 55.$$

De maneira geral, a Programação Dinâmica de Dígitos pode ser usada para:

- contar números dentro de um intervalo que satisfaçam determinadas restrições;
- encontrar números que possuam propriedades específicas em seus dígitos;
- calcular somas, médias ou frequências de ocorrência de padrões relacionados aos dígitos.

A ideia central da técnica consiste em processar um dígito por vez, do mais significativo para o menos significativo, rastreando se estamos construindo um número que ainda segue as restrições do limite superior ou se já ultrapassamos esse limite.

Exemplo intuitivo:

Considere que queremos calcular alguma propriedade para todos os números menores ou iguais a 4578. Suponha que os dois primeiros dígitos, da esquerda para a direita, já foram escolhidos.

- **Caso 1:** Quando os dois primeiros dígitos escolhidos coincidem com os dígitos do limite superior, ou seja, temos o prefixo `45xy`. Nesse caso:
  - se $x \in \{0, 1, 2, 3, 4, 5, 6\}$, então $y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ (isto é, y não está restrito);
  - se $x = 7$, então $y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ (isto é, y está restrito).

  Note que, enquanto o próximo dígito for igual ao dígito correspondente do limite superior, a restrição permanece ativa.

- **Caso 2:** Quando o prefixo formado pelos dígitos escolhidos é menor do que o prefixo do limite superior, os próximos dígitos deixam de ser restritos. Por exemplo, $42xy < 4578$. Assim:

$$x, y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

Formulação do subproblema:

Dado um número $N = d_n d_{n-1} \ldots d_1$, representando o limite superior, queremos calcular a soma dos dígitos de todos os números entre 1 e N.

Suponha que já construímos parcialmente um número da forma:

$$e_n e_{n-1} e_{idx+1} ? \ldots ?$$

tal que

$$e_n + e_{n-1} + \ldots + e_{idx+1} = \text{sum},$$

Assim, para cada estado parcialmente construído, associamos um subproblema da forma:

$$dp(idx, sum, tight),$$

onde:

- `idx` indica a posição atual do dígito que estamos processando;
- `sum` representa a soma parcial dos dígitos escolhidos até o momento;
- `tight` é uma variável booleana que indica se o número atual ainda é limitado pelo prefixo de N.

Mais formalmente:

$$tight = \begin{cases} 1, & \text{se } e_n e_{n-1} e_{idx+1} = d_n d_{n-1} d_{idx+1}, \\ 0, & \text{se } e_n e_{n-1} e_{idx+1} < d_n d_{n-1} d_{idx+1}. \end{cases}$$

Relação de recorrência

A relação de recorrência é definida como:

$$dp(-1, sum, \_) = sum$$

Esse é o caso base: quando todos os dígitos já foram processados ($idx = -1$), a função retorna a soma acumulada dos dígitos formados até o momento.

Quando não há restrição em relação ao limite superior ($tight = 0$), todos os próximos dígitos podem variar livremente de 0 a 9:

$$dp(idx, sum, 0) = \sum_{d=0}^{9} dp(idx - 1,\ sum + d,\ 0).$$

Quando ainda há restrição ($tight = 1$), o próximo dígito d pode assumir valores de 0 até d_{idx}, onde d_{idx} é o dígito correspondente na decomposição de N:

$$dp(idx, sum, 1) = dp(idx - 1,\ sum + d_{idx},\ 1)\ +\ \sum_{d=0}^{d_{idx}-1} dp(idx - 1,\ sum + d,\ 0).$$

Interpretação da recorrência:

- O termo $dp(idx - 1,\ sum + d_{idx},\ 1)$ corresponde ao caso em que o dígito atual escolhido é igual ao dígito do limite superior d_{idx}, mantendo assim a restrição ($tight = 1$).
- O somatório $\sum_{d=0}^{d_{idx}-1} dp(idx-1,\ sum+d,\ 0)$ representa os casos em que o dígito atual é menor que d_{idx}, o que faz com que os próximos dígitos deixem de estar restritos ao limite ($tight = 0$).

Em resumo:

- O caso $tight = 1$ mantém a restrição e divide-se entre o dígito igual ao limite e os menores que ele;
- O caso $tight = 0$ ignora o limite, permitindo livre escolha dos dígitos subsequentes;
- O caso base retorna a soma total obtida após processar todos os dígitos.

```cpp
#include <bits/stdc++.h>

using namespace std;

long long int soma_digitos(vector <int> digitos, int idx, int soma, int apertado){

    if(idx == -1){
        return soma;
    }

    if(!apertado){
        long long int ret = 0;

        for(int i = 0; i <= 9; i++){
            ret += soma_digitos(digitos, idx-1, soma + i, 0);
        }

        printf("idx = %d, apertado = %d, ret = %lld\n", idx, apertado, ret);

        return ret;
    }else{
        //                    1 + soma_digitos(-1)
        long long int ret = soma_digitos(digitos, idx-1, soma + digitos[idx],  1);

        for(int i = 0; i < digitos[idx]; i++){
            ret += soma_digitos(digitos, idx-1, soma+ i, 0);
        }

        printf("idx = %d, apertado = %d, ret = %lld\n", idx, apertado, ret);

        return ret;

    }
}

int main(){
    string s = "23";

    vector <int> digitos;
    digitos.resize(s.size());
    int pos = s.size() - 1;
    for(int i = 0; i < (int)s.size(); i++)
        digitos[pos--] = s[i] - '0';

    cout << soma_digitos(digitos, digitos.size() - 1, 0, 1);

}

/*
idx = 0, apertado = 1, ret = 14
idx = 0, apertado = 0, ret = 45
idx = 0, apertado = 0, ret = 55
idx = 1, apertado = 1, ret = 114
114
*/
```

Sobreposição de subproblemas

Considere que estamos calculando a soma dos dígitos de todos os números menores ou iguais a 99786. Observe que resolver o subproblema correspondente a $56xyz$ e o subproblema $83pqr$ pode ser considerado equivalente, desde que as somas parciais dos dígitos escolhidos até o momento sejam iguais e ambos os casos estejam na situação *sem restrições* (`tight` = 0).

Em outras palavras, se a soma acumulada até o ponto atual é a mesma e os próximos dígitos podem variar livremente de 0 a 9, as possíveis escolhas para (x, y, z) e (p, q, r) serão idênticas.

Assim, basta resolver apenas um desses subproblemas e armazenar (memorizar) o resultado obtido, de modo que o outro possa reutilizar essa resposta sem necessidade de recomputação. Essa é a essência da técnica de **memorização** na Programação Dinâmica de Dígitos, que evita o cálculo redundante de subproblemas equivalentes e melhora significativamente o desempenho do algoritmo.

```cpp
#include <bits/stdc++.h>
using namespace std;

#define max_digit 20
#define max_sum max_digit*9
long long dp[max_digit][max_sum][2];

long long int soma_digitos(vector <int> digitos, int idx, int soma, int apertado){
    if(idx == -1){
        return soma;
    }
    if( dp[idx][soma][apertado] != -1)
        return dp[idx][soma][apertado];
    if(!apertado){
        long long int ret = 0;
        for(int i = 0; i <= 9; i++){
            ret += soma_digitos(digitos, idx-1, soma + i, 0);
        }
        return dp[idx][soma][apertado] = ret;
    }else{
        long long int ret = soma_digitos(digitos, idx-1, soma + digitos[idx],  1);

        for(int i = 0; i < digitos[idx]; i++){
            ret += soma_digitos(digitos, idx-1, soma+ i, 0);
        }
        return dp[idx][soma][apertado] = ret;
    }
}

int main(){
    string s = "23";

    vector <int> digitos;
    digitos.resize(s.size());
    int pos = s.size() - 1;
    for(int i = 0; i < (int)s.size(); i++)
        digitos[pos--] = s[i] - '0';

    memset(dp, -1, sizeof(dp));

    cout << soma_digitos(digitos, digitos.size() - 1, 0, 1);

}
```

Contagem de números binários sem dígitos consecutivos iguais a 1

O programa a seguir tem como objetivo contar a quantidade de números binários menores ou iguais a um número N cuja representação binária **não contém dois bits consecutivos iguais a 1**. Por exemplo, para $N = 10_{10} = 1010_2$, os números válidos são:

$$1,\ 10,\ 100,\ 101,\ 1000,\ 1001,\ 1010,$$

totalizando 7 números.

A ideia consiste em processar o número bit a bit, da direita para a esquerda, e em cada posição decidir qual bit (0 ou 1) pode ser colocado sem violar a restrição de não haver dois bits 1 consecutivos.

Definimos a função:

$$dp[idx][prev][tight]$$

onde:

- `idx` é o índice do bit atual sendo processado (0 é o bit menos significativo);
- `prev` indica se o bit anterior foi 1 (`true`) ou 0 (`false`);
- `tight` indica se o prefixo atual está *preso* ao prefixo de N, isto é, se ainda não excedemos os bits de N.

O valor armazenado em `dp[idx][prev][tight]` representa a quantidade de números válidos que podem ser formados considerando as restrições acima a partir da posição `idx` até o bit menos significativo.

Condições base e recorrência

A condição base é:

$$dp[-1][prev][tight] = 1$$

pois, quando todos os bits já foram processados, temos exatamente uma configuração válida (o número formado até o momento).

A relação de recorrência é:

$$dp[idx][prev][tight] = \sum_{digit=0}^{limit} \begin{cases} 0, & \text{se } prev = 1 \text{ e } digit = 1 \\ dp[idx-1][digit = 1][nextTight], & \text{caso contrário} \end{cases}$$

onde

$$limit = \begin{cases} digitos[idx], & \text{se } tight = 1, \\ 1, & \text{caso contrário}, \end{cases}$$

e

$$nextTight = tight \wedge (digit = digitos[idx]).$$

Implementação em C++

```cpp
#define max_digit 32
long long dp[max_digit][2][2];

long long conta_numeros(vector<int> digitos, int idx, bool prev, bool apertado) {
    if (idx == -1) return 1;
    if (dp[idx][prev][apertado] != -1) return dp[idx][prev][apertado];

    int limit = apertado ? digitos[idx] : 1;
    long long count = 0;

    for (int digit = 0; digit <= limit; digit++) {
        if (prev && digit == 1) continue; // evita 11 consecutivos
        bool nextApertado = apertado && (digit == digitos[idx]);
        count += conta_numeros(digitos, idx - 1, digit == 1, nextApertado);
    }

    return dp[idx][prev][apertado] = count;
}
```

Conversão do número e chamada principal

O número N é convertido em sua forma binária, armazenando os bits em um vetor. Em seguida, o vetor é processado pela função de DP. O resultado é decrementado em 1, pois o número 0 é contado, mas não deve ser considerado na resposta final.

```cpp
int main() {
    int n = 1234;
    vector<int> digitos;
    while (n > 0) {
        digitos.push_back(n % 2);
        n /= 2;
    }
    memset(dp, -1, sizeof(dp));
    cout << conta_numeros(digitos, digitos.size() - 1, false, true) - 1 << endl;
}
```

Verificação por força bruta

O código inclui também uma função de *força bruta* que percorre todos os números de 1 a N, verificando bit a bit se há dois 1 consecutivos. Essa função serve apenas para validar o resultado da DP.

```cpp
bool biton(int n, int pos) {
    return (n & (1 << pos));
}

long long brute_force(int n) {
    long long cont = 0;
    for (int k = 1; k <= n; k++) {
        bool ok = true;
        for (int j = 0; j < 31; j++) {
            if (biton(k, j) && biton(k, j + 1)) ok = false;
        }
        if (ok) cont++;
    }
    return cont;
}
```

Problemas Clássicos de Programação Dinâmica em Dígitos (Digit DP)

A seguir apresentamos uma lista de problemas clássicos que podem ser resolvidos utilizando a técnica de **Programação Dinâmica de Dígitos (Digit DP)**. Essa abordagem é útil para resolver problemas que envolvem a contagem ou soma de propriedades numéricas sobre intervalos grandes, onde uma iteração direta seria inviável.

1. SPOJ – PR003004 (Digit Sum Interval)

Enunciado: Calcular a soma de todos os dígitos de todos os números dentro do intervalo $[L, R]$.
Exemplo: $L = 1$, $R = 13 \Rightarrow 55$.
Dica: Usar uma função auxiliar $f(N)$ que calcula a soma dos dígitos de 1 até N e então $S(L, R) = f(R) - f(L-1)$. A DP deve armazenar a soma acumulada de dígitos parciais e o estado de restrição (*tight*).

https://www.spoj.com/problems/PR003004/

2. SPOJ – CPCRC1C (Sum of Digits)

Enunciado: Dado um número N, calcular a soma dos dígitos de todos os números de 1 até N.
Dica: A função recursiva $dp(idx, sum, tight)$ considera o índice do dígito, a soma parcial dos dígitos e se o número construído até o momento é igual ao prefixo de N. É um dos exemplos mais clássicos de Digit DP.

https://www.spoj.com/problems/CPCRC1C/

3. SPOJ – KOPC12H (OE Numbers)

Enunciado: Contar quantos números em $[L, R]$ têm a soma dos dígitos pares maior do que a soma dos dígitos ímpares.
Dica: A DP pode ser modelada como $dp(idx, diff, tight)$ onde *diff* representa a diferença entre a soma dos dígitos pares e ímpares. No final, apenas estados com $diff > 0$ devem ser contados.

https://www.spoj.com/problems/KOPC12H/

4. SPOJ – NY10E (Non-Decreasing Digits)

Enunciado: Dado N, contar quantos números de N dígitos (permitindo zeros à esquerda) possuem dígitos não decrescentes, ou seja, $d_1 \leq d_2 \leq \ldots \leq d_N$.
Dica: A DP pode ser modelada como $dp(pos, lastDigit)$, onde *lastDigit* representa o último dígito escolhido. Em cada passo, só é permitido escolher um dígito $\geq$ ao anterior.

https://www.spoj.com/problems/NY10E/