



Maratona **Cln**

**Seletiva 2020**

# Programação Dinâmica (aula 2)

Aula 7

# Bitmask

# Bitmask

---



Maratona **CIn**

O que é ?

-> representação de estados/situações com Números binários.

Quando usar ?

-> Quando se faz necessário armazenar determinadas combinações.

Por que usar ?

-> Fácil armazenamento, operações bit-a-bit.



**Exemplo:** Existem 4 leds, eles podem ser ligados ou desligados, cada combinação possui um valor. É necessário guardar todas essas combinações com seus respectivos valores. O que fazer ?

# Bitmask

---



MaratonaCIn

**Exemplo:** Existem 4 leds, eles podem ser ligados ou desligados, cada combinação possui um valor. É necessário guardar todas essas combinações com seus respectivos valores. O que fazer ?

**Resposta:** Bitmask





# Bitmask




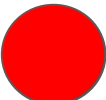
---



MaratonaCIn





Perceba que cada led possui dois estados: **On/off**  
Podemos representar um led ligado como 1, desligado como 0.



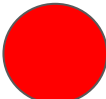

    = 1010

    = 0011

# Bitmask

Perceba que cada led possui dois estados: **On/off**  
Podemos representar um led ligado como 1, desligado como 0.

    = 1010

    = 0011

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

int main(){

    int cost[(1<<4)]; // 4 leds = 4 bits

    for(int i=0;i<(1<<4);i++){ // passeia por todas combinações possíveis
        int c;
        cin >> c;
        cost[i] = c;
    }
}
```



# Bitmask



MaratonaCIn

`__builtin_popcount(int mask)` = retorna a quantidade de bits setados em “mask”

`(valor << quantidade)` = shift left

`(valor >> quantidade)` = shift right

`(valor1 & valor2)` = and bit-a-bit

`(valor1 | valor2)` = or bit-a-bit

`(valor1 ^ valor2)` = xor bit-a-bit

`valor = ~valor;` not

**Cuidado com overflow !!**

Para valores maiores que  $10^9$ , usem `(ll(valor) << quantidade)`

ll = long long

# DP com Bitmask

# DP com Bitmask

---



MaratonaCIn

## Motivação:

Situações (combinações) são os **estados** de uma DP. A solução é aplicar a Bitmask nesses estados e programar a DP com um dos seus estados sendo a bitmask.

**Dica:** Problemas envolvendo bitmask são problemas em que combinações existem entre objetos e a quantidade desses objetos são pequenas.

$n \leq 20$  (pois será computado/armazenado  $2^n$ )



Exemplo:

Existem  $N$  caixas ( $n \leq 10$ ), a  $i$ -ésima caixa tem dimensões  $w \times h \times l$ ,  $w$  = largura,  $h$  = altura,  $l$  = comprimento. Não existem duas caixas com as mesmas dimensões. Determine qual o tamanho da maior pilha de caixas que é possível formar sem que a base da caixa superior seja maior que qualquer outra base de uma caixa inferior.



Exemplo:

Existem  $N$  caixas ( $n \leq 10$ ), a  $i$ -ésima caixa tem dimensões  $w \times h \times l$ ,  $w$  = largura,  $h$  = altura,  $l$  = comprimento. Não existem duas caixas com as mesmas dimensões. Determine qual o tamanho da maior pilha de caixas que é possível formar sem que a base da caixa superior seja maior que qualquer outra base de uma caixa inferior.

Quais são os estados dessa DP ?



Exemplo:

Existem  $N$  caixas ( $n \leq 10$ ), a  $i$ -ésima caixa tem dimensões  $w \times h \times l$ ,  $w$  = largura,  $h$  = altura,  $l$  = comprimento. Não existem duas caixas com as mesmas dimensões. Determine qual o tamanho da maior pilha de caixas que é possível formar sem que a base da caixa superior seja maior que qualquer outra base de uma caixa inferior.

Quais são os estados dessa DP ?

**Estado 1:** A última caixa colocada no topo da pilha

**Estado 2:** As dimensões da base escolhida da caixa no topo da pilha

**Estado 3:** Todas as caixas que ainda não foram colocadas na pilha

# DP com Bitmask



MaratonaCIn

Exemplo:

Existem  $N$  caixas ( $n \leq 10$ ), a  $i$ -ésima caixa tem dimensões  $w \times h \times l$ ,  $w$  = largura,  $h$  = altura,  $l$  = comprimento. Não existem duas caixas com as mesmas dimensões. Determine qual o tamanho da maior pilha de caixas que é possível formar sem que a base da caixa superior seja maior que qualquer outra base de uma caixa inferior.

Quais são os estados dessa DP ?

**Estado 1:** A última caixa colocada no topo da pilha

**Estado 2:** As dimensões da base escolhida da caixa no topo da pilha

**Estado 3:** Todas as caixas que ainda não foram colocadas na pilha

$N$  caixas

3 possíveis escolhas

$(2^n)$  ~usando bitmask

```
int DP[MAXN][3][(1<<MAXN)]; // dp necessária
```

# DP com Bitmask



MaratonaCIn

```
int dp[13][3][(1<<11) + 5]; // lembre de resetar a dp toda vez que for usar
typedef struct{
    int x;
    int y;
    int z;
}Box;
vector<Box> g;

int main(){

    int n;
    cin >> n;
    // lê a entrada e adiciona no vector (... resto do código)

    int bmask = 0;
    for(int i=0;i<n;i++){
        bmask = bmask|(1<<i);
        // seta todos os bits (caixas) como livre.
    }
    cout << solve(-1,0,bmask,n) << endl;
}
```



# DP com Bitmask



MaratonaCIn

```
int solve(int last, int op, int bmask, int n){

    if(bmask == 0) return 0; // acabaram as caixas livres
    if(dp[last][op][bmask] != -1) return dp[last][op][bmask]; // esse estado já foi calculado anteriormente

    int ret = 0;

    for(int i=0;i<n;i++){
        if( (1 << i) & bmask){ // se o resultado for != 0, então a caixa pode ser usada
            bmask = bmask^(1<<i); // seta a caixa como ocupada
            for(int j=0;j<3;j++){
                bool ok = valid(last,op,i,j); // verifica se eh possivel colocar a caixa i em cima da ultima caixa
                if(ok) ret = max(ret, 1 + solve(i,j,bmask,n)); // chama a recursao para uma nova combinacao
            }
            bmask = bmask|(1 << i); // a caixa que já foi usada, volta a ser livre
        }
    }

    dp[last][op][bmask] = ret; // atualiza a dp com o valor calculado desse estado
    return ret;
}
```

# DP com Bitmask

---

Link do problema original: <http://codeforces.com/gym/100642/attachments> (letra A)  
resolução: <https://pastebin.com/skfP79FT>

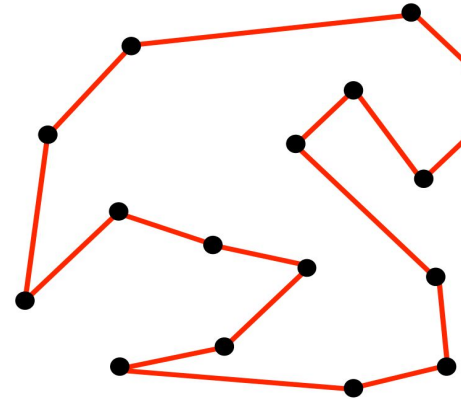
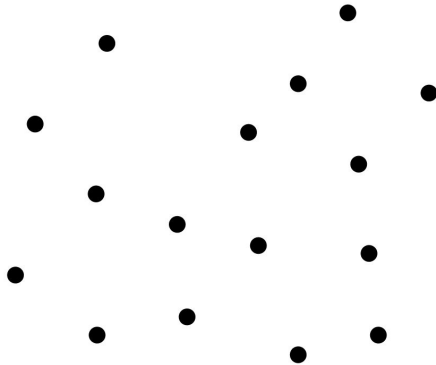


**TSP**

# TSP

Motivação:

Menor/Maior custo para visitar todos os vértices de um grafo  $G(V,E)$ , passando por cada vértice apenas uma vez.



# TSP



MaratonaCIn

Ressalvas:

- TSP é um problema **NP-completo**(até agora não existe solução polinomial)
  - Em contests geralmente o  $n \leq 20$
- Existem estados que se sobrepõem
  - Sol: PD
- Precisamos saber quais cidades foram visitadas previamente.
  - Sol: Bitmask



# TSP(Menor Custo)

## Código:

- Estados da dp:
  - Vértice atual
  - Conjunto de vértices já visitados
- Complexidade:
  - ?

```
#define ms 20
#define inf 0x3f3f3f3f
int v, dp[ms][1 << ms ], graph[ms][ms], n;

int min(int a, int b){ return (a > b) ? b : a; }

int solve(int pos, int visited){

    if(visited + 1 == (1<<n) )
        return 0;

    int ans = dp[pos][visited];

    if(~ans)
        return ans;

    ans = inf;

    for (int i = 0; i < n; i++) {

        if( !(visited & (1<<i)) ){

            ans = min(ans, graph[pos][i] + solve(i, visited|(1<<i)) );

        }

    }

}
```

# Digit DP



Motivação:

Calcular quantos números menores que **n** satisfazem uma certa propriedade.

Exemplo:

Quantos números menores que **n** possuem exatamente **k** ocorrências de um dígito **d**?  **$n \leq 10^{18}$ ,  $k \leq 18$**

E se for no intervalo  $[a, b]$ ?

$\text{solve}(b) - \text{solve}(a - 1)$



# Digit DP(exemplo)

Código:

E se o número  
fosse até  
 $10^{100}$ ?

```
vector<int> number;
int a, b, d, k;
int dp[18][18][2];

int solve(int pos, int qnt, int smaller) {
    if (qnt > k) return 0;
    if (pos == number.size()) return (qnt == k);
    if (dp[pos][qnt][smaller] != -1) return dp[pos][qnt][smaller];

    int ans = 0;
    int lim = (smaller == 0) ? number[pos] : 9;

    for (int digit = 0; digit <= lim; digit++) {
        int new_smaller = smaller;
        int new_qnt = qnt;
        if (!smaller && digit < lim) new_smaller = 1;
        if (digit == d) new_qnt++;
        ans += solve(pos + 1, new_qnt, new_smaller);
    }
    return dp[pos][qnt][smaller] = ans;
}
```



# Digit DP

---



Maratona**CIn**

Exemplo: