Bitmask em Programação Dinâmica

Laboratório de Programação Competitiva - 2020

Pedro Henrique Paiola

Manipulação de bits

- Linguagens como C e C++ permitem manipular os bits de uma certa variável, normalmente numérica.
- Operadores bit-a-bit em C e C++:
  - NOT: ~
  - AND: &
  - OR: |
  - XOR: ^
  - Left shift: <<
  - Right shift: >>

Manipulação de bits

- **NOT**: o operador ~ é um operador unário que inverte o valor de todos os bits de um número. Aplica a negação para cada bit.
- Exemplo:

$$N = 5 = (101)_2$$
$$\sim N = \sim 5 = \sim (101)_2 = (010)_2 = 2$$

x	~x
0	1
1	0

Manipulação de bits

- **AND**: o operador **&** é um operador binário que opera em duas palavras de bits de mesmo tamanho, aplicando a operação lógica E bit-a-bit.
- Exemplo:

$A = 5\ (101)_2$

$B = 3\ (011)_2$

$A\ \&\ B = (101)_2\ \&\ (011)_2 = (001)_2 = 1$

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

Manipulação de bits

- **OR**: o operador | é um operador binário que opera em duas palavras de bits de mesmo tamanho, aplicando a operação lógica OU bit-a-bit.
- Exemplo:

$A = 9\ (1001)_2$

$B = 3\ (0011)_2$

$A \mid B = (1001)_2 \& (0011)_2 = (1011)_2 = 11$

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

Manipulação de bits

- **XOR**: o operador ^ é um operador binário que opera em duas palavras de bits de mesmo tamanho, aplicando a operação lógica OU-EXCLUSIVO bit-a-bit.
- Exemplo:

$A = 9\ (1001)_2$

$B = 3\ (0011)_2$

$A\ \hat{}\ B = (1001)_2 \& (0011)_2 = (1010)_2 = 10$

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

Manipulação de bits

- **Left Shift**: o operador **<<** é um operador binário que desloca **N** vezes para a esquerda os bits de uma palavra **X** (X << N). Numericamente, é equivalente a multiplicar um número por 2^n.
- Exemplo:

$3 = (0011)_2$

$3 << 1 = (0110)_2 = 6$

$3 << 2 = (1100)_2 = 12$

Manipulação de bits

- **Right Shift**: o operador **>>** é um operador binário que desloca **N** vezes para a direita os bits de uma palavra **X** (X >> N). Numericamente, é equivalente a dividir um número por 2^n.
- Exemplo:

$12 = (1100)_2$

$12 >> 1 = (0110)_2 = 6$

$12 >> 2 = (0011)_2 = 3$

$12 >> 3 = (0001)_2 = 1$

Bitmask

- Podemos aproveitar estas operações de diversas formas. De forma geral, utilizamos a técnica de **mascaramento** ou **máscara de bits**, em que aplicamos uma operação bit a bit a uma certa variável e uma constante (máscara) de forma a extrair a informação desejada de forma simples e eficiente.
- Em Programação Competitiva, normalmente utilizamos bitmasks para representar e operar conjuntos.
    - Utilizamos uma máscara de bits em que cada bit representa um certo elemento. Se o bit for 1, este elemento está incluso no conjunto, se for 0, não está.

Bitmask

- Exemplo:

  Conjunto: {a, b, c, d, e, f, g, h}

  bit 7 = a, bit 6 = b, ... , bit 0 = h

  Subconjuntos:

  {b, c, f, h} = 01100101

  {a} = 10000000

  {} = 00000000

Bitmask

- **Adicionar um elemento no conjunto**: vamos setar o bit correspondente ao elemento.

```
void addElement(int &set, int elemPos){
    set = set | (1 << elemPos)
}

Exemplo: set = 10001    elemPos = 2
10001 | 00100 = 10101
```

Bitmask

- **Remover um elemento no conjunto**: vamos resetar o bit correspondente ao elemento.

```
void removeElement(int &set, int elemPos){
    set = set & ~(1 << elemPos)
}

Exemplo: set = 10101    elemPos = 2
10101 & 11011 = 10001
```

Bitmask

- **Checar se um conjunto contém um elemento**: checar se o bit que representa o elemento está setado.

```
bool hasElement(int set, int elemPos){
    return ((set & (1 << elemPos)) != 0);
}

Exemplo: set = 01001    elemPos = 3
01001 & 01000 = 01000 != 0
```

Bitmask

- **União de dois conjuntos**: um elemento estará presente na união de dois conjuntos sse pelo menos um dos conjuntos contiver o elemento. Com as bitmasks que representam os conjuntos, basta aplicar o operador OR

```
int union(int setA, int setB){
    return (setA | setB);
}
```

```
Exemplo: setA = 01100   setB = 00101
01100 | 00101 = 01101
```

Bitmask

- **Intersecção de dois conjuntos**: um elemento estará presente na intersecção de dois conjuntos sse ele está presente em ambos os conjuntos. Com as bitmasks que representam os conjuntos, basta aplicar o operador AND

```
int intersection (int setA, int setB){
    return (setA & setB);
}
```

```
Exemplo: setA = 01100   setB = 00101
01100 & 00101 = 00100
```

Bitmask

- **Complemento**: um elemento estará presente no complementar de um conjunto sse ele não pertence ao conjunto. Para isso, podemos utilizar o operador NOT

```
int complement(int set){
    return ~set;
}

Exemplo: set = 01100
         ~set = 10011
```

Bitmask - Exemplo

- **Problema:** Você, professor, possui **n** questões para elaborar uma prova, sendo que a **i**-ésima questão possui dificuldade $\mathbf{c_i}$. Agora, você tem que preparar uma prova que a dificuldade total das questões seja pelo menos **l** e no máximo **r,** e que a diferença entre a questão mais difícil e a mais fácil seja pelo menos **x**. Calcule a quantidade de maneiras de preparar a prova
- **Restrições**
  - $2 \leq n \leq 15$
  - $1 \leq l \leq r \leq 10^9$
  - $1 \leq x \leq 10^6$

Bitmask - Exemplo

- **Solução:** Vamos enumerar todos os subconjuntos possíveis de questões, verificando quais atendem as restrições impostas. Para enumerar cada subconjunto, pasta incrementarmos uma variável contador de 0 (nenhum elemento no conjunto) até (1 << n) - 1 (todos os elementos no conjunto)

```
0  = 0000
1  = 0001
2  = 0010
3  = 0011
...
14 = 1110
15 = 1111
```

Bitmask - Exemplo

```
for(int mask = 0; mask < (1 << n); mask++){
    long long s = 0;
    int maior = 0; int menor = inf;
    for(int i = 0; i < n; i++) { //percorrendo todos os bits
        if (hasElement(mask, i)){
            s += c[i];
            maior = max(maior, v[i]);
            menor = min(menor, v[i]);
    }
    if (s >= l && s <= r && (maior - menor) >= x)
        resp++;
}
```

Programação Dinâmica com Bitmask

- O uso de bitmasks pode nos auxiliar em diversas tarefas, em especial em problemas de programação dinâmica em que um conjunto de elementos faz parte do estado do nosso problema.
- Comum em problemas de seleção em que a escolha a ser determinada em um certo estado (selecionar ou não selecionar) depende ou se relaciona de alguma forma com as escolhas anteriores.

PD com Bitmask - Atribuição de tarefas

- **Problema:** temos **N** pessoas e **N** tarefas, sendo que cada tarefa deve ser alocada para apenas uma pessoa. Temos também uma matriz ***cost*** de tamanho **NxN**, em que **cost[i][j]** denota o custo para a pessoa **i** desempenhar a tarefa **j**.

PD com Bitmask - Atribuição de tarefas

- **Força bruta:** por esta abordagem, vamos testar todas as atribuições possíveis

```
assign(N,  cost)
    for i = 0 to N
        assignment[i] = i   //assigning task i to person i
    res = INFINITY
    for j = 0 to factorial(N)
        total_cost = 0
        for i = 0 to N
            total_cost = total_cost + cost[i][assignment[i]]
        res = min(res, total_cost)
        generate_next_greater_permutation(assignment)
    return res
```

PD com Bitmask - Atribuição de tarefas

- **Programação Dinâmica:** podemos modelar como um problema de programação dinâmica, tendo como estado **(k, mask)** em que **k** que as pessoas de **0 a k-1** já tiveram uma tarefa atribuída, e **mask** representa o conjunto de tarefas que já foram atribuídas a alguém.
- Supondo que temos a resposta de (k, mask), temos o seguinte passo, considerando que **i** é uma tarefa disponível:

$$dp(k+1, mask|(1 << i)) = min(dp(k+1, mask|(1 << i)), dp[k][mask] + cost[k][i])$$

PD com Bitmask - Atribuição de tarefas

- Melhorando um pouco esta solução, podemos eliminar o parâmetro **k**, pois **k** é sempre o número de elementos no conjunto representado pela máscara, sendo assim, ficamos com o seguinte passo:

$$dp(mask|(1 << i)) = min(dp(mask|(1 << i)), dp[mask] + cost[x][i])$$

em que x = número de bits setados na máscara

- Complexidade da solução: $O(2^n n)$

PD com Bitmask - Atribuição de tarefas

```
assign(N, cost)
    for i = 0 to power(2,N)
        dp[i] = INFINITY
    dp[0] = 0
    for mask = 0 to power(2, N)
        x = count_set_bits(mask)
        for j = 0 to N
            if jth bit is not set in mask
                dp[mask|(1<<j)] = min(dp[mask|(1<<j)],
                                      dp[mask]+cost[x][j])
    return dp[power(2,N)-1]
```

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask =
x =
j =
mask | (1 << j) =
dp[mask] + cost[x][j] =

i		dp[i]
0	000	0
1	001	inf
2	010	inf
3	011	inf
4	100	inf
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 000
x = 0
j = 0
mask | (1 << j) = 001
dp[mask] + cost[x][j] = 0 + 15

i		dp[i]
0	000	0
1	001	inf
2	010	inf
3	011	inf
4	100	inf
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 000
x = 0
j = 0
mask | (1 << j) = 001
dp[mask] + cost[x][j] = 15

i		dp[i]
0	000	0
1	001	15
2	010	inf
3	011	inf
4	100	inf
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 000
x = 0
j = 1
mask | (1 << j) = 010
dp[mask] + cost[x][j] = 0 + 10

i		dp[i]
0	000	0
1	001	15
2	010	inf
3	011	inf
4	100	inf
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 000
x = 0
j = 1
mask | (1 << j) = 010
dp[mask] + cost[x][j] = 10

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	inf
4	100	inf
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 000
x = 0
j = 2
mask | (1 << j) = 100
dp[mask] + cost[x][j] = 0 + 9

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	inf
4	100	inf
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 000
x = 0
j = 2
mask | (1 << j) = 100
dp[mask] + cost[x][j] = 9

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	inf
4	100	9
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 001
x = 1
j = 1
mask | (1 << j) = 011
dp[mask] + cost[x][j] = 15 + 15

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	inf
4	100	9
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 001
x = 1
j = 1
mask | (1 << j) = 011
dp[mask] + cost[x][j] = 30

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	30
4	100	9
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 001
x = 1
j = 2
mask | (1 << j) = 101
dp[mask] + cost[x][j] = 15 + 10

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	30
4	100	9
5	101	inf
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 001
x = 1
j = 2
mask | (1 << j) = 101
dp[mask] + cost[x][j] = 25

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	30
4	100	9
5	101	25
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 010
x = 1
j = 0
mask | (1 << j) = 011
dp[mask] + cost[x][j] = 10 + 9

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	30
4	100	9
5	101	25
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

taref / pessoa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 010
x = 1
j = 0
mask | (1 << j) = 011
dp[mask] + cost[x][j] = 19

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	**19**
4	100	9
5	101	25
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 010
x = 1
j = 2
mask | (1 << j) = 110
dp[mask] + cost[x][j] = 10 + 10

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	25
6	110	inf
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 010
x = 1
j = 2
mask | (1 << j) = 110
dp[mask] + cost[x][j] = 20

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	25
6	110	20
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 011
x = 2
j = 2
mask | (1 << j) = 111
dp[mask] + cost[x][j] = 19 + 8

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	25
6	110	20
7	111	inf

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 011
x = 2
j = 2
mask | (1 << j) = 111
dp[mask] + cost[x][j] = 27

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	25
6	110	20
7	111	27

PD com Bitmask - Atribuição de tarefas

taref / pessoa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 100
x = 1
j = 0
mask | (1 << j) = 101
dp[mask] + cost[x][j] = 9 + 9

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	25
6	110	20
7	111	27

PD com Bitmask - Atribuição de tarefas

tarefa / pessoa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 100
x = 1
j = 0
mask | (1 << j) = 101
dp[mask] + cost[x][j] = 18

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	**18**
6	110	20
7	111	27

PD com Bitmask - Atribuição de tarefas

taref / pessoa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 100
x = 1
j = 1
mask | (1 << j) = 110
dp[mask] + cost[x][j] = 9 + 15

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	18
6	110	20
7	111	27

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 100
x = 1
j = 1
mask | (1 << j) = 110
dp[mask] + cost[x][j] = 24

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	18
6	110	20
7	111	27

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 101
x = 2
j = 1
mask | (1 << j) = 111
dp[mask] + cost[x][j] = 18 + 12

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	18
6	110	20
7	111	27

PD com Bitmask - Atribuição de tarefas

taref / pessoa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 101
x = 2
j = 1
mask | (1 << j) = 111
dp[mask] + cost[x][j] = 30

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	18
6	110	20
7	111	27

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	0	1	2
0	15	10	9
1	9	15	10
2	10	12	8

mask = 110
x = 2
j = 0
mask | (1 << j) = 111
dp[mask] + cost[x][j] = 20 + 10

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	18
6	110	20
7	111	27

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	10	9
1	9	15	10
2	10	12	8

mask = 110
x = 2
j = 0
mask | (1 << j) = 111
dp[mask] + cost[x][j] = 30

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	18
6	110	20
7	111	27

PD com Bitmask - Atribuição de tarefas

pessoa \ tarefa	**0**	**1**	**2**
0	15	**10**	9
1	**9**	15	10
2	10	12	**8**

mask = 111

resp = dp[111] = **27**

i		dp[i]
0	000	0
1	001	15
2	010	10
3	011	19
4	100	9
5	101	18
6	110	20
7	111	27

Caminho Hamiltoniano

- **Problema:** Temos **n** cidades e **m** estradas de mão dupla entre elas. Queremos saber o menor custo para visitar todas as cidades sem passar por uma mesma ciadade mais de uma vez, começando da cidade 0.
- **Restrições:**
  - $1 \leq n \leq 15$
  - $n - 1 \leq m \leq n(n-1)/2$
  - A distância entre duas cidades é menor que 10^3
  - As cidades são enumeradas de 0 à n-1

Caminho Hamiltoniano

- **Solução:** aplicaremos programação dinâmica com auxílio de bitmask, com o estado (mask, i) onde dp[mask][i] representa o custo mínimo para visitar todas as cidades que estão marcadas na bitmask sendo que a última cidade visitada é a i.
- Nesse caso, temos que verificar quais os possíveis caminhos a serem tomadas.

$$dp[mask][i] = \begin{cases} 0 & \text{se } mask = 2^n - 1 \\ min_{0 \leq v \leq n-1}(dp[mask|1 << v][v] + C[i][v]) & \text{caso contrário} \end{cases}$$

- Complexidade: $O(2^n n^2)$

Caminho Hamiltoniano

```
int solve(int mask, int i){
    if (mask == (1 << n) - 1)
        return dp[mask][i] = 0;
    if (dp[mask][i] != -1)
        return dp[mask][i];
    int ans = inf;
    for(int v = 0; v < n; v++){
        if ((mask & (1 << v)) || !mat[i][v]) continue;
        ans = min(ans, solve(mask | (1 << v), v) + mat[i][v]);
    }
    return dp[mask][i] = ans;
}
```

Pebbles (EOlymp - 1218)

- **Problema:** dado um tabuleiro de tamanho n x n ($3 \leq n \leq 15$) em que cada posição possui um valor entre 10 e 99, queremos distribuir peças no tabuleiro de forma que a soma das posições ocupadas seja máxima.
- Porém, duas peças não podem ocupar a mesma posição, e nem duas posições adjacentes (seja na horizontal, vertical ou diagonal).

Pebbles (EOlymp - 1218)

- **Solução:** vamos aplicar programação dinâmica. Vamos varrer o tabuleiro, de cima para baixo e da esquerda para a direita, e para cada posição teremos duas opções para avaliar: colocar uma peça ou não colocar.
- Porém, a opção de colocar uma peça depende desta posição estar livre, o que por sua vez depende das peças anteriores que já foram colocadas.
- Por este motivo, e buscando avaliar todas as soluções possíveis, os parâmetros do nosso estado deve compreender não só a posição atual (i,j), mas também o conjunto de posições já ocupadas por peças.

Pebbles (EOlymp - 1218)

- Na prática, não precisamos de todas as posições anteriores, apenas aquelas que podem influenciar na nossa escolha. Por isso, se nosso tabuleiro tem tamanho n x n, vamos armazenar as escolhas das últimas n + 1 posições.

	n	n-1	n-2	...
...	0			

Pebbles (EOlymp - 1218)

- Para verificar se um movimento é possível, os bits 0, n-2, n-1 e n da bitmask não podem estar setados, pois eles representam as posições adjacentes a posição atual.

	n	n-1	n-2	...
...	0			

Pebbles (EOlymp - 1218)

- Relação de recorrência:

$$dp[mask][pos] = \begin{cases} 0 & \text{se } pos \text{ está fora dos limites} \\ dp[mask << 1][prox(pos)] & \text{se } !estaLivre(pos) \\ min(dp[mask << 1][prox(pos)], & \\ dp[mask << 1 + 1][prox(pos)] + value[pos]) & \text{caso contrário} \end{cases}$$

C++ STL: Bitset

- Temos um problema em utilizar bitmask: estamos limitados ao número de bits de nossas variáveis numéricas.
- Exemplo: sizeof(long long int) → 8 bytes = 64 bits
- Em muitos problemas de PD, isso não chega a ser um problema porque no caso em que precisamos de conjuntos para definir os estados, normalmente a complexidade será exponencial. Sendo assim, as dimensões de nossa entrada não podem ser muito grandes.
- Mas, se for necessário, a STL do C++ nos oferece uma estrutura de dados interessante: o **bitset**

C++ STL: Bitset

- Na prática, podemos trabalhar com ele quase como se fosse um vetor de valores booleanos. Ao instanciarmos um objeto bitset precisamos definir o número de bits necessários. O tamanho deve ser uma constante.

```
bitset<100> bset1;  //Todos os bits começam com o valor 0
bitset<100> bset2(20); //inicializado com os bits de 20

cout << bset1 << endl; //000000...000000
cout << bset2 << endl; //000000...010100
```

C++ STL: Bitset

- Podemos consultar e alterar valores em um bitset utilizando o operador [].

```
bset1[i] = 1;
cout << bset1[i] << endl;
```

- Ou podemos utilizar os métodos:
  - `set()`, `set(pos)` ou `set(pos, val)`
  - `reset()` ou `reset(pos)`
  - `flip()` ou `flip(pos)`
  - `test(pos)`

C++ STL: Bitset

- Outros métodos que podem ser bastante úteis são:
  - `count()`: retorna a quantidade de bits setados
  - `size()`
  - `any()`: testa se pelo menos algum bit está setado
  - `none()`: teste se nenhum bit está setado
  - `all()`: testa se todos os bits estão setados

C++ STL: Bitset

- Além disso, todos os operadores bit a bit estão sobrecarregados e podem ser utilizados em bitsets.

```
bitset<4> bset1(9); // bset1 contains 1001
bitset<4> bset2(3); // bset2 contains 0011

// comparison operator
cout << (bset1 == bset2) << endl; // false 0
cout << (bset1 != bset2) << endl; // true  1

// bitwise operation and assignment
cout << (bset1 ^= bset2) << endl; // 1010
cout << (bset1 &= bset2) << endl; // 0010
cout << (bset1 |= bset2) << endl; // 0011
```

```
// left and right shifting
cout << (bset1 <<= 2) << endl; // 1100
cout << (bset1 >>= 1) << endl; // 0110

// not operator
cout << (~bset2) << endl; // 1100

// bitwise operator
cout << (bset1 & bset2) << endl; // 0010
cout << (bset1 | bset2) << endl; // 0111
cout << (bset1 ^ bset2) << endl; // 0101
```

Referências

https://github.com/UnBalloon/programacao-competitiva/tree/master/Opera%C3%A7%C3%B5es%20bit%20a%20bit

https://www.geeksforgeeks.org/bitmasking-and-dynamic-programming-set-1-count-ways-to-assign-unique-cap-to-every-person/

https://oprofessorleandro.wordpress.com/tag/mascara-de-bits/

https://neps.academy/lesson/228

https://www.hackerearth.com/practice/algorithms/dynamic-programming/bit-masking/tutorial/

https://noic.com.br/materiais-informatica/ideias/bitmask/

https://www.geeksforgeeks.org/c-bitset-and-its-application/