Aula 3 · Busca Completa e Matemática

Desafios de Programação

Fernando Kiotheka Vinicius Tikara Date

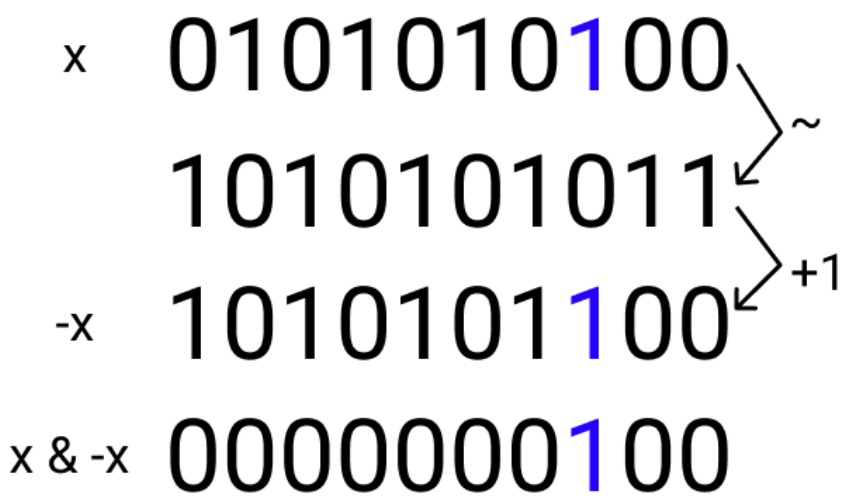
UFPR

18/09/2024

Operando com bits em C++

Operações em bits

- Multiplicar/Dividir por potências de dois:
  - `x << y` $= 2^y x$.
  - `x >> y` $= \lfloor 2^{-y} x \rfloor$.
- Verificar o bit i: `x & (1 << i) != 0`.
- Ligar o bit i: `x | (1 << i)`.
- Desligar o bit i: `x & ~(1 << i)`.
- Alternar o bit i: `x ^ (1 << i)`
- Todos os n bits ligados: `(1 << n) - 1`.
- Obter o bit menos significativo ligado: `x & -x`.



Contagem de bits!

É útil as vezes contar quantos 1s existem na representação binária de um número. Fazer esse código não é difícil, mas o GCC nos dá algumas funções que podem ser usadas para obter essas contagens:

Função	Descrição
`popcount`	Conta o número de 1s.
`__builtin_ctzll`	Conta o número de zeros à direita (0 é indefinido).
`std::__lg`	Posição do bit mais significativo (lg), cuidado que é indefinido para 0.
`ffsll`	Posição do primeiro bit menos significativo indexado em 1.

Problema de Deslocamento de Partículas

Um quadrado tem seus vértices nas coordenadas $(0, 0)$, $(0, 2^N)$, $(2^N, 2^N)$, $(2^N, 0)$. Cada vértice tem um atrator. Uma partícula é inicialmente colocada na posição $(2^{N-1}, 2^{N-1})$. Cada atrator pode ser ativado individualmente, qualquer número de vezes. Quando um atrator na posição (i, j) é ativado, se a partícula está na posição (p, q), ela será movida no ponto do meio entre (i, j) e (p, q).

Dado um N $(1 \leq N \leq 20)$ e um ponto (x, y) com $0 < x, y < 2^N$, calcule o menor número de vezes que você precisa ativar os atratores para que a partícula acabe na posição (x, y). Você pode assumir para todos os casos de teste que uma solução existe.

Solução do Deslocamento de Partículas

Código particles.cpp

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

int main() {
    cin.tie(0);
    ios_base::sync_with_stdio(0);

    int n, x, y; cin >> n >> x >> y;
    assert((x&-x) == (y&-y));
    cout << n-1 - __lg(x&-x) << "\n";
}
```

Mas eu quero mais de 64 bits!

Alguns problemas exigem mais bits do que um inteiro possui, e também precisam de um espaço compacto para armazenar esses bits (exemplo: 10^9). Como alternativa você pode usar o `vector<bool>` ou o tipo especializado `bitset<N>` do C++. Ele não cresce dinamicamente como o `vector<bool>`, mas pode ser impresso pelo `cout` e tem mais operações legais:

- `bitset<n> m (string s)` [$\mathcal{O}(n)$]: Cria um `bitset` com n bits e inicializado com s (composto de `` `0's `` e `` `1's ``).
- `bitset<n> m (ull x = 0)` [$\mathcal{O}(n)$]: Cria um `bitset` com n bits e inicializado com x (alinhado a direita).
- `m[size_t i]` [$\mathcal{O}(1)$]: Acessa o bit i.
- `m.count()` [$\mathcal{O}(n)$]: Conta quantidade de bits 1.
- `m.reset()` [$\mathcal{O}(n)$]: Coloca 0 em todos os bits.
- `~m` [$\mathcal{O}(n)$]: Alterna o valor de todos os bits.
- `m |, &, ~, <<, >>` [$\mathcal{O}(n)$]: As operações bit a bit usuais.

Exemplo de bitset

Código bitset.cpp

```
#include <bits/stdc++.h>
using namespace std;
using b20 = bitset<20>;
void p(b20 b) { cout << b << " c=" << b.count() << "\n"; }
int main() {
    b20 bits ("10001001110011101001");
    b20 mask = ~(b20(0b111) << 10 | b20(0b111) << 5);
    p(bits); p(bits & mask); bits.reset();
    bits[0] = 1; bits[1] = true; bits[3] = 1;
    bits[8] = 1; bits[8] = false; bits[18] = 1;
    p(bits); p(bits ^ mask);
}
```

Entrada	Saída
	10001001110011101001 c=10 10001000000000001001 c=4 01000000000000001011 c=4 10111110001100010100 c=10

Busca Completa

Força bruta

- A dita “solução trivial”.
- Tentar todas as possibilidades, mas tem que saber contar!
- Frequentemente é necessário usar isso como parte da solução.
- Essa solução pode ser mais fácil de codar e passa no tempo limite.

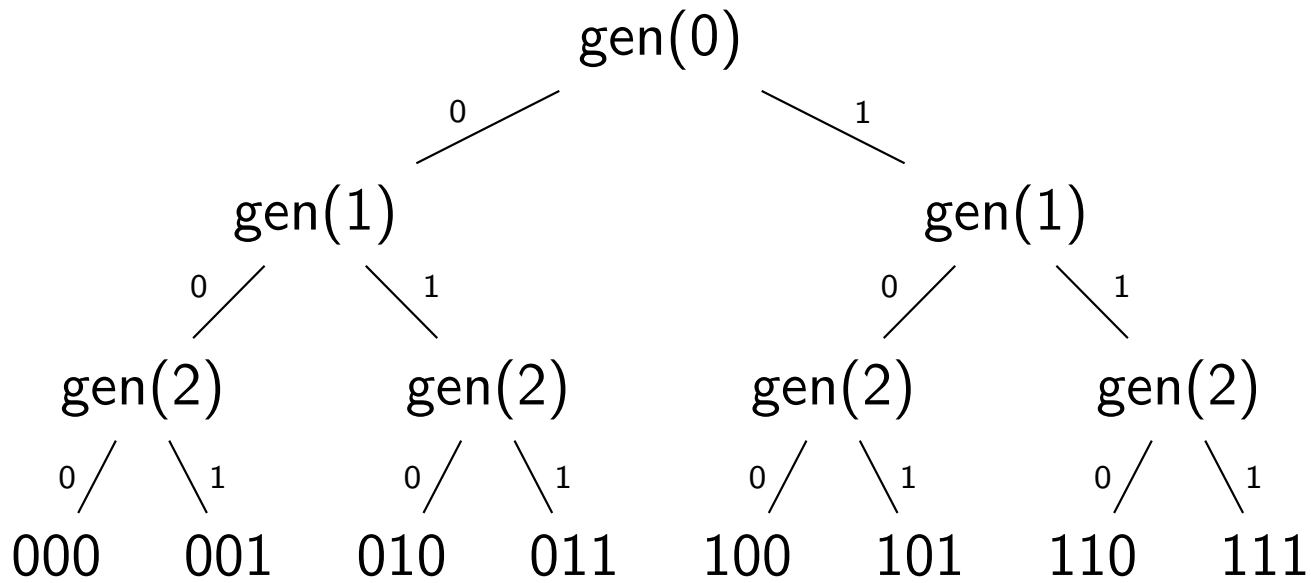
Testando todos subconjuntos

Suponha que temos n elementos. Quantos subconjuntos existem com esses elementos? A matemática discreta nos traz a resposta:

$$2 \times 2 \times \cdots \times 2 \times 2 = 2^n$$

- Se tivermos um elemento 1, temos os subconjuntos $\{1\}$ e $\{\}$.
- Se tivermos dois elementos 1 e 2, temos os subconjuntos $\{1, 2\}$, $\{1\}$, $\{2\}$ e $\{\}$.
- E assim por diante, como enumerar?
- Se não quisermos o subconjunto vazio, é só ignorá-lo.

Recursão!


gen(0)
0
1
gen(1)
gen(1)
0
1
0
1
gen(2)
gen(2)
gen(2)
gen(2)
0
1
0
1
0
1
0
1
000
001
010
011
100
101
110
111

Solução recursiva para enumerar subconjuntos

Código `setsrec.cpp`

```
#include <bits/stdc++.h>
using namespace std;
#define PN (t == (1<<n)-1 || t++ % 4 == 3 ? "\n" : " ")
vector<bool> s (100); int n, t;
void gen(int k) {
    if (k == n) {
        for (int i = 0; i < n; i++) { cout << s[i]; }
        cout << PN; return; }
    s[k] = 0; gen(k+1); s[k] = 1; gen(k+1);
}
int main() { while (cin >> n) { t = 0; gen(0); } }
```

Entrada	Saída
1 2 3	0 1 00 01 10 11 000 001 010 011 100 101 110 111

Serve um bit?

O jeito mais simples, claro, é usar a representação binária.

Código `sets.cpp`

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    while (cin >> n)
    for (int s = 0; s < (1<<n); s++) {
        for (int i = 0; i < n; i++)
            cout << bool(s & (1<<i));
        cout << (s == (1<<n)-1 || s == 0b11 ? "\n" : " ");
    }
}
```

Entrada	Saída
1 2 3	0 1 00 10 01 11 000 100 010 110 001 101 011 111

Conjuntos como bits

As operações em conjuntos representados por bits são as seguintes:

- Conjunto $\{i\}$: `1 << i`.
- Complemento A^c: `~A`.
- União $A \cup B$: `A | B`.
- Intersecção $A \cap B$: `A & B`.
- Diferença simétrica $A \triangle B$: `A ^ B`.
- Subtração de conjuntos $A \setminus B$: `A & ~B`.

Iterando sobre subconjuntos

Algo que extremamente útil quase trata de conjuntos é iterar sobre os subconjuntos de algum conjunto.

Código `subset.cpp`

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int ss, mask = 0b00101001;
    for (ss = mask; /* ss > 0 */; ss = (ss-1) & mask) {
        cout << bitset<8>(ss) << (ss & 1 ? " " : "\n");
        if (ss == 0) { break; }
    }
}
```

Entrada	Saída
	00101001 00101000 00100001 00100000 00001001 00001000 00000001 00000000

Permutações

Suponha que temos n elementos em uma **sequência**. De quantos jeitos diferentes podem rearranjar essa sequência? De novo, a matemática discreta nos traz a resposta:

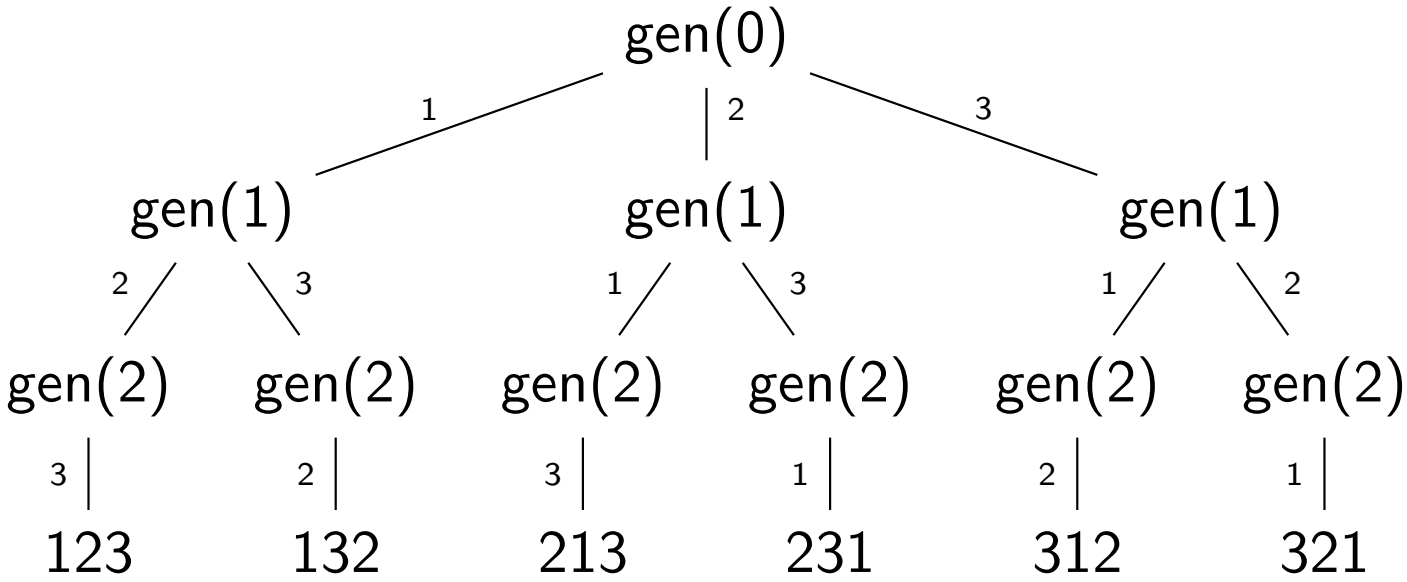
$$n!$$

Uma sequência $[1, 2, 3]$ por exemplo pode ser ordenada de $3! = 3 \cdot 2 \cdot 1 = 6$ jeitos diferentes:

- $[1, 2, 3]$
- $[1, 3, 2]$
- $[2, 1, 3]$
- $[2, 3, 1]$
- $[3, 1, 2]$
- $[3, 2, 1]$

Como enumerar todas?

Recursão! (novamente)



Solução recursiva para enumeração de permutações

Código permutationrec.cpp

```
#include <bits/stdc++.h>
using namespace std;
#define PN cout << "\n"
vector<int> s (100); vector<bool> chosen (101); int n, t;
void gen(int k) {
    if (k == n) {
        for (int i = 0; i < n; i++) { cout << s[i]; }
        cout << " "; return; }
    for (int i = 1; i <= n; i++) if (!chosen[i]) {
        chosen[i] = 1; s[k] = i; gen(k+1); chosen[i] = 0;
    }
}
int main() { while (cin >> n) { t = 0; gen(0); PN; } }
```

Entrada	Saída
1 2 3	1 12 21 123 132 213 231 312 321

Usando next_permutation

Código permutation.cpp

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n; while (cin >> n) {
        vector<int> a (n);
        for (int i = 0; i < n; i++) { a[i] = i+1; }
        do {
            for (int x : a) { cout << x; } cout << " ";
        } while (next_permutation(a.begin(), a.end()));
        cout << "\n";
    }
}
```

Entrada	Saída
1	1
2	12 21
3	123 132 213 231 312 321

Problema das Equações Simples

Você quer dado os inteiros A, B, C $(1 \leq A, B, C \leq 10^4)$ saber se existem números inteiros distintos x, y e z que satisfazem as equações:

$$x + y + z = A$$
$$xyz = B$$
$$x^2 + y^2 + z^2 = C$$

Como resolver? Na força bruta é claro. Mas note que:

- Como C é no máximo 10^4, para atingir o C máximo, x, y e z podem ser no máximo 10^2.
- Soluções com x, y e z negativos são possíveis.

Solução das Equações Simples com busca iterativa completa

Código `equations.cpp`

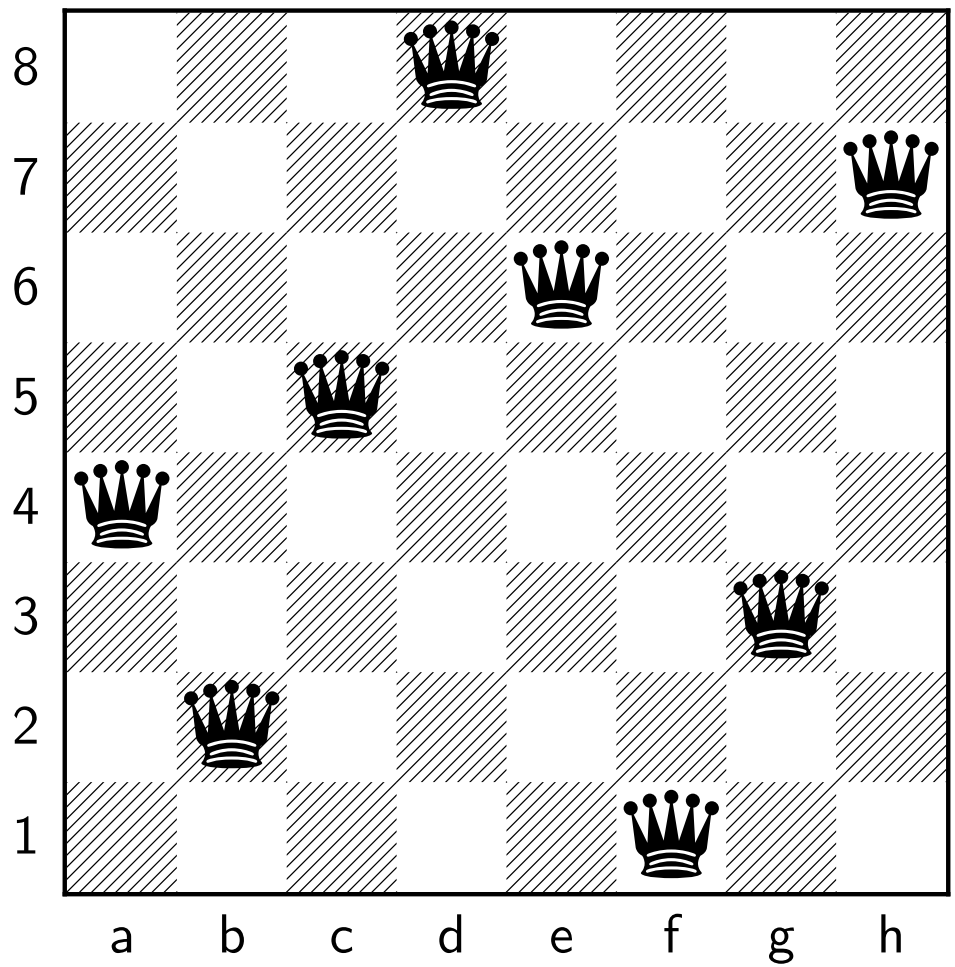
```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int a, b, c; while (cin >> a >> b >> c) {
        bool ok = false;
        for (int x = -100; x <= 100; x++) {
            for (int y = -100; y <= 100; y++) {
                for (int z = -100; z <= 100; z++) {
                    if (x != y && x != z && y != z &&
                            x + y + z == a &&
                            x * y * z == b &&
                            x*x + y*y + z*z == c && !ok) {
                        cout << "S\n"; ok = true;
                    } } } }
        if (!ok) { cout << "N\n"; } }
}
```

Exemplos do problema Equações Simples

Entrada	Saída
10 3978 4126	S
91 3108 5681	S
1 2 3	N
3 3 3	N
3 1 3	N
1 8 21	S
17 108 121	S
10 1 100	N
47 240 1637	S
57 3468 1481	S
52 2210 1350	S
1 4 9	N
5 1 13	N
46 980 1290	S

Problema das n-rainhas

Quantos jeitos de colocar n rainhas, uma em cada coluna, no tabuleiro de xadrez $n \times n$?



Vamos resolver usando *backtracking*

Código `queens.cpp`

```
#include <bits/stdc++.h>
using namespace std;
int n, s; vector<int> q (1000);
void sol(int cc) {
    if (cc == n) { s++; return; }
    for (int r = 0; r < n; r++) {
        bool ok = true;
        for (int c = 0; c < cc; c++)
            ok &= q[c] != r && abs(r-q[c]) != abs(cc-c);
        if (ok) { q[cc] = r; sol(cc+1); }
    }
}
int main() {
    while (cin >> n) { s = 0; sol(0); cout << s << " "; }
}
```

Entrada	Saída
8 9 10 11 12 13	92 352 724 2680 14200 73712

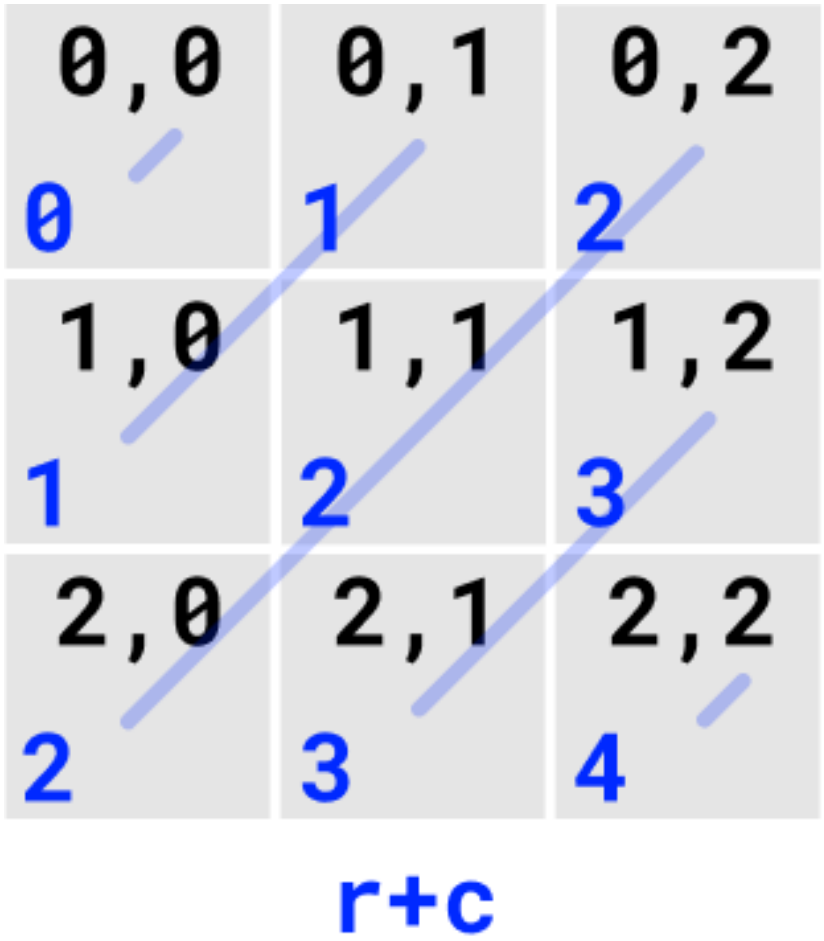
Dá pra melhorar um pouco né?

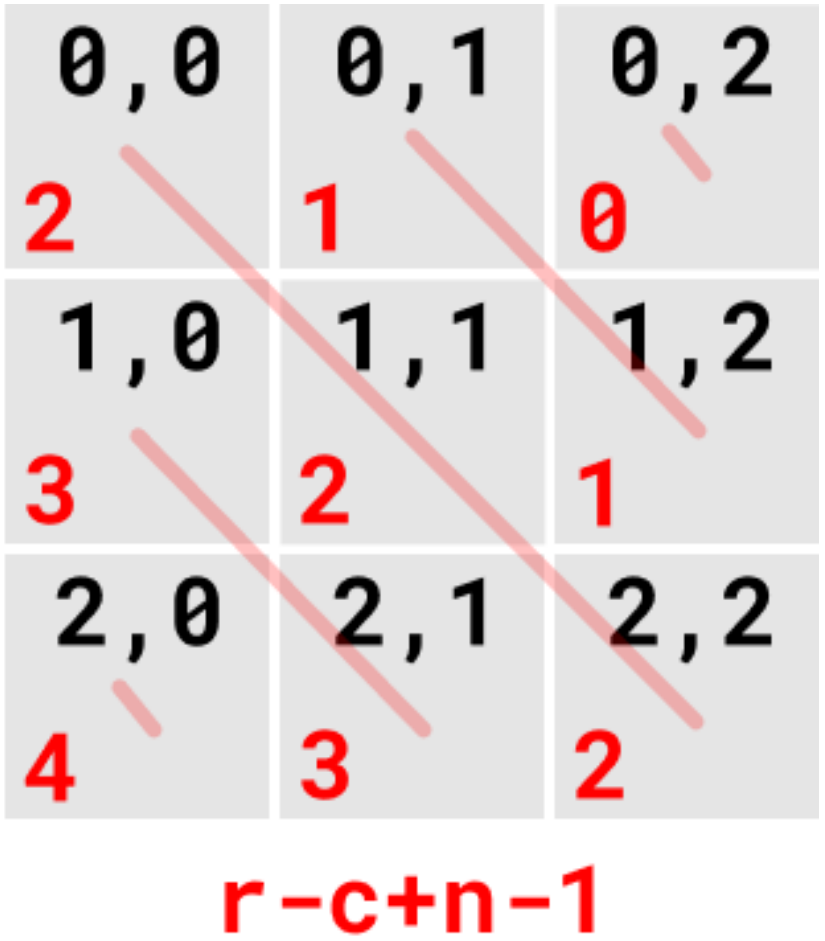
Código queens2.cpp

```
#include <bits/stdc++.h>
using namespace std;
int n, s; vector<bool> q(1000), d1 (1000), d2 (1000);
void sol(int c) {
    if (c == n) { s++; return; }
    for (int r = 0; r < n; r++) {
        if (q[r] || d1[r+c] || d2[r-c+n-1]) { continue; }
        q[r] = d1[r+c] = d2[r-c+n-1] = true;
        sol(c+1);
        q[r] = d1[r+c] = d2[r-c+n-1] = false;
    }
}
int main() {
    while (cin >> n) { s = 0; sol(0); cout << s << " "; }
}
```

Entrada	Saída
10 11 12 13 14	724 2680 14200 73712 365596

O que foi isso?





Melhor ainda?!

Código queens3.cpp

```
#include <bits/stdc++.h>
using namespace std; using ll = long long;
int n, s;
void sol(int i, ll rw = 0, ll ld = 0, ll rd = 0) {
    if (rw == (1<<n)-1) { s++; return; }
    ll pos = ((1<<n)-1) & (~(rw | ld | rd));
    while (pos) {
        ll p = pos & -pos; pos -= p;
        sol(i+1, rw | p, (ld | p) << 1, (rd | p) >> 1);
    }
}
int main() {
    while (cin >> n) { s = 0; sol(0); cout << s << " "; }
}
```

Entrada	Saída
12 13 14 15	14200 73712 365596 2279184

Ainda assim exponencial!

- O problema de contar n rainhas é difícil, e atualmente a melhor solução pra se fazer isso é fazer busca exaustiva.
- O maior n que se conhece é 27, são 23490796715412252 8 combinações.
- O problema de **completar** um tabuleiro parcialmente preenchido de rainhas foi provado NP-Completo.
- Como diz o sábio Macacário de Maratona de Programação do ITA: "Não é hora de tentar ganhar o prêmio nobel".

Economizando passos

Fazer uma busca completa completa é perigoso, pois pode não passar no tempo limite. Mas se o seu programa nas piores entradas aparenta estar na reta do tempo limite, podemos tentar:

Poda Abandone as árvores que não tem chance de sucesso cedo. No caso das rainhas por exemplo, abandonamos quando há um conflito entre rainhas já colocadas.

Simetria São 92 soluções para um tabuleiro de 8 rainhas, porém são só 12 soluções únicas espelhadas e rotacionadas. Porém, simetria é difícil.

Pré-calcular O famoso método *Halim*. Se não existirem muitas respostas, coloque todas direto no código!

Otimização usual Aproveite da *cache* acessando os elementos na ordem que eles estão dispostos na memória. Use estruturas de dados menores, etc. Isso ajuda por exemplo no CSES (juiz com problemas excelentes).

Algoritmos Vai ver... busca completa não é a solução?

Matemática

Quando matemática é a solução...

Quando os limites de um problema são na ordem 10^{10}, quase nenhum outro tipo de solução se aplica. Tome por exemplo o problema Conta:

$$S = 1 - 1 + 1 - 1 + 1 - 1 + 1 - + \ldots$$

Dado um número N ($0 \leq N \leq 10^{10}$), a quantidade de termos da soma, qual é o resultado da soma dos N termos da expressão?
É óbvio notar nesse caso que o que importa é verificar a paridade.
Mas e pra progressões mais conhecidas?

Progressão aritmética

$$a_n = a_k + r(n - k)$$

- r: Razão aritmética
- k: Termo conhecido
- n: Termo que se quer obter

$$S_n = \frac{n(a_1 + a_n)}{2}$$

Progressão geométrica

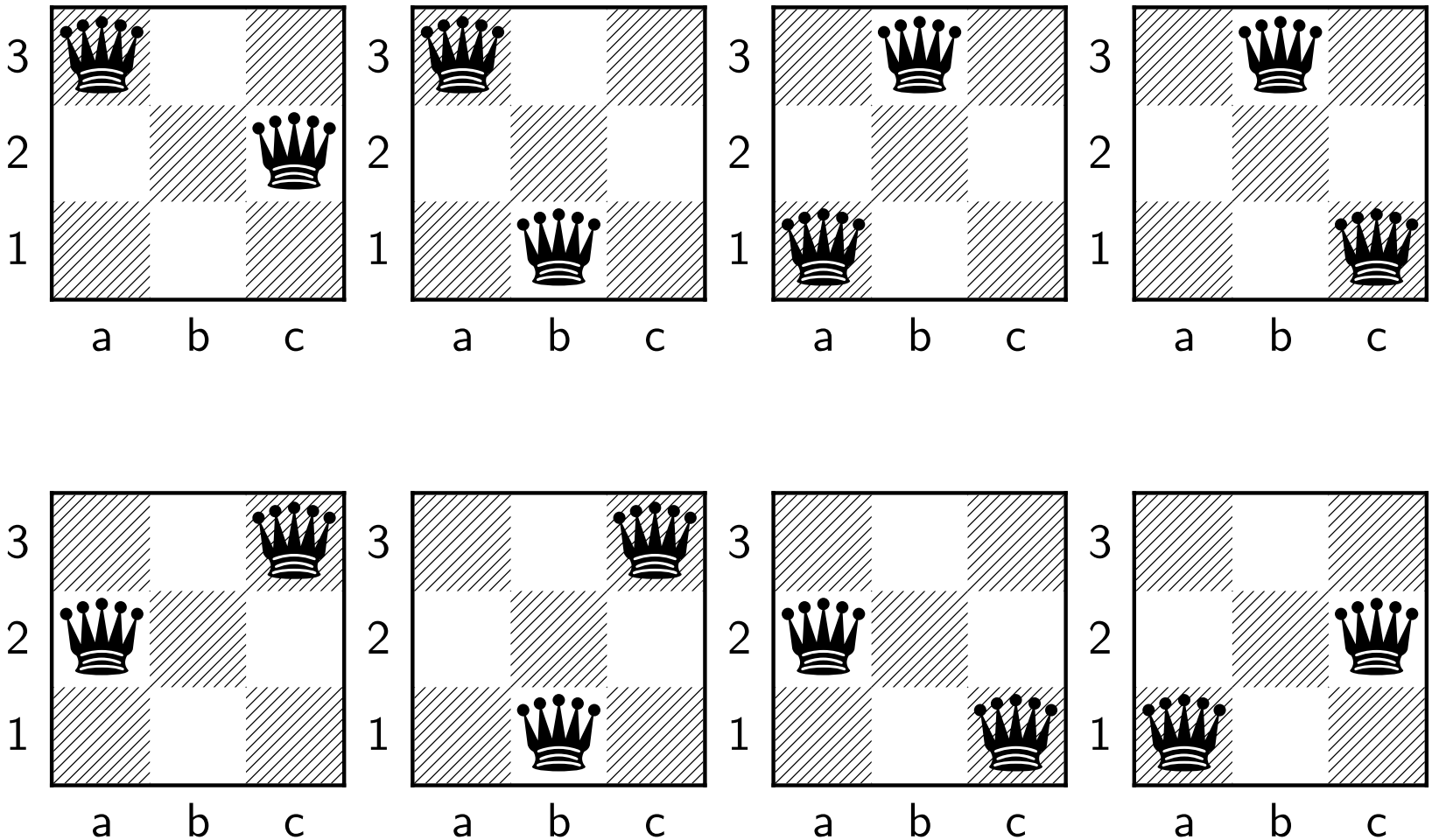
$$a_n = a_k q^{(n-k)}$$

- q: Razão geométrica
- k: Termo conhecido
- n: Termo que se quer obter

$$S_n = \frac{a_1(q^n - 1)}{q - 1}$$

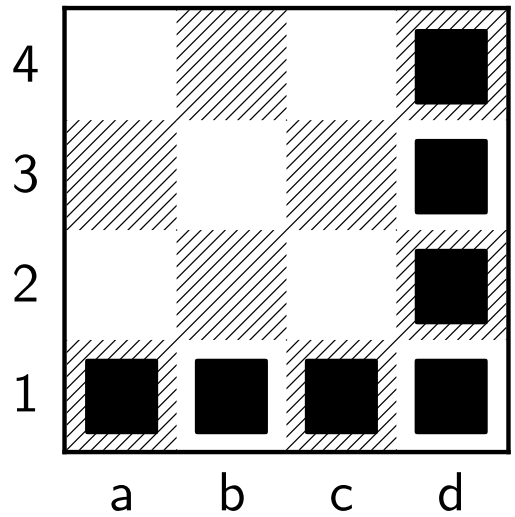
Problema das Duas Rainhas

Quantos jeitos de colocar duas rainhas em um tabuleiro $n \times n$?



Vamos pensar em resolver isso em $\mathcal{O}(1)$

Vamos definir $q(n)$ como sendo a solução desse problema.



- Se nenhuma rainha está na área marcada, $q(n-1)$.
- Colocamos então uma rainha em uma das $2n-1$ posições.
- Essa rainha ataca $3(n-1)$ posições do tabuleiro.
- Sobram $n^2 - 3(n-1) - 1$ posições.
- Finalmente, excluímos o que contamos duas vezes, quando estão na última linha ou coluna: $(n-1)(n-2)$.

Fórmula!

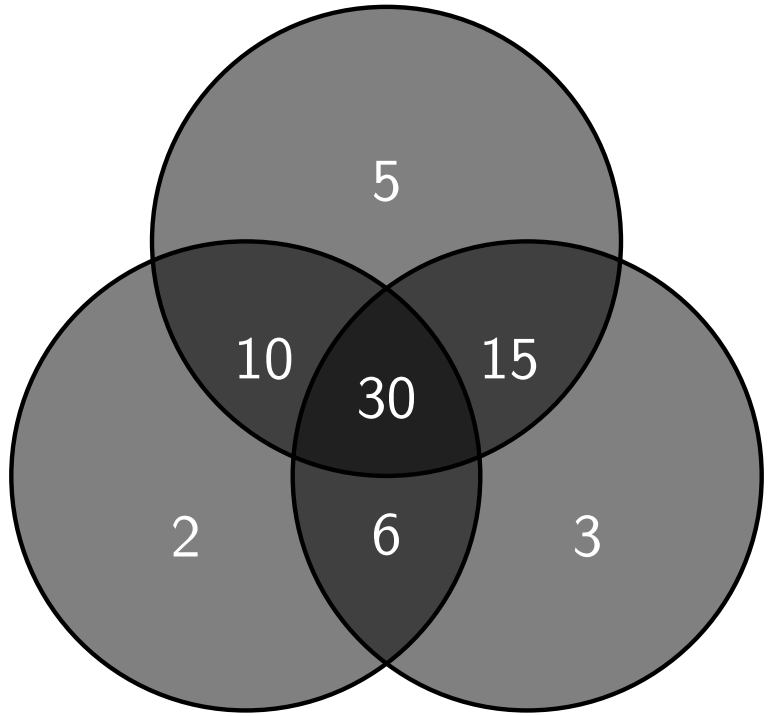
$$\begin{aligned} q(n) &= q(n-1) + (2n-1)(n^2 - 3(n-1) - 1) - (n-1)(n-2) \\ &= q(n-1) + 2(n-1)^2(n-2) \end{aligned}$$

Resolvendo usando piruetas de Discreta...

$$q(n) = \frac{n^4}{2} - \frac{5n^3}{3} + \frac{3n^2}{2} - \frac{n}{3}.$$

Princípio da inclusão-exclusão

Dado um problema onde queremos achar quantos números de um intervalo $[1, n]$ são múltiplos de 2, 3 ou 5, como faz? Temos que imaginar os múltiplos do intervalo $[1, n]$ como um diagrama:



Assim, podemos concluir que a quantidade é dada por:

$$C_{2,3,5}(N) = C_2 + C_3 + C_5 - C_{10} - C_{15} - C_6 + C_{30}$$

Onde C_k é o número de múltiplos de k em $[1, n]$, ou seja, $\lfloor n/k \rfloor$.

Princípio da inclusão-exclusão generalizado

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{i<j} |A_i \cap A_j| + \ldots + (-1)^n \sum_{i<j} |A_1 \cap \ldots \cap A_n|$$

- Como implementar computacionalmente?
- Essa conta se resume a:
  - Pegar todos os subconjuntos.
  - Verificar o tamanho do subconjunto atual e determinar o sinal.
  - Obter o resultado da intersecção nos elementos.
  - Multiplicar o resultado pelo sinal.
  - Somar no total.
- Implementação é deixada ao leitor.