Matemática

Permutações

Prof. Edson Alves
Faculdade UnB Gama

Princípio Multiplicativo

- O princípio multiplicativo está relacionado ao número de elementos do produto cartesiano de dois conjuntos
- Se A e B são dois conjuntos finitos não vazios, com $|A| = n$, $|B| = m$, então o produto cartesiano $A \times B$ terá nm elementos
- Este princípio é útil em contagem de n-uplas de elementos, onde cada elemento da n-upla vem de um conjunto distinto, ou de um conjunto já utilizado, excluídos os elemento já escolhidos
- Deste princípio derivam os conceitos de permutação, arranjo e combinação

Permutações

Seja A um conjunto com n elementos distintos. Uma **permutação** consiste em um ordenação destes elementos tal que duas permutações são distintas se dois ou mais elementos ocuparem posições distintas.

Notação: $P(n)$

Exemplo: se $A = \{1, 2, 3\}$, há 6 permutações distintas, a saber:

$$123, 132, 213, 231, 312, 321$$

Cálculo do número de permutações

- Considere um conjunto com n elementos distintos
- Para a primeira posição há n escolhas
- Para a segunda, $(n-1)$ escolhas, uma vez que o primeiro elemento já foi escolhido
- Pelo mesmo motivo, há $(n-2)$ escolhas para o terceiro elemento, e assim sucessivamente, até restar uma única escolha para o último elemento
- Portanto,

$$P(n) = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 = n!$$

Caracterização das permutações

- Em combinatória é útil associar os conceitos de contagem à situações práticas e tentar encontrar soluções por analogia
- As permutações, por exemplo, podem ser visualizadas como a retirada de n bolas distintas de uma caixa, sem reposição
- Veja que tanto as bolas quanto a ordem de retirada importam, no sentido que duas permutações são distintas se a ordem de alguma das bolas é diferente

Permutações com repetição

- Um permutação com repetição consiste em uma ordenação de n elementos, não necessariamente distintos
- Considere um conjunto de k elementos distintos, onde cada um deles ocorre n_i vezes, com $i = 1, 2, \ldots, k$, de forma que $n_1 + n_2 + \ldots + n_k = n$
- Dentre as $n!$ permutações dos n elementos, várias delas serão repetidas
- De fato, como o elemento i se repete n_i vezes, uma permutação p em particular se repetirá $n_i!$ vezes

Permutações com repetição

- Isto porque todas as permutações de posições dentre as cópias de i levam a uma mesma permutação
- Por exemplo, para o conjunto $A = \{1, 2, 1\}$, apenas 3 das 6 permutações são distintas: $112, 121, 211$
- Assim, o número de permutações distintas, com repetições, é dado por

$$PR(n; n_1, n_2, \ldots, n_k) = \frac{n!}{n_1! \times n_2! \times \ldots \times n_k!}$$

Implementação da permutação com repetição

```
template<typename T>
long long permutations(const vector<T>& A)
{
    map<T, int> hist;

    for (auto a : A)
        ++hist[a];

    long long res = factorial(A.size());

    for (auto [a, ni] : hist)
        res /= factorial(ni);

    return res;
}
```

Permutações circulares

- Se, em uma permutação, os objetos devem ser dispostos em uma formação circular, sem uma marcação clara de início de fim, algumas permutações se tornam idênticas, a menos de uma rotação
- Para contabilizar apenas as permutações que não podem ser geradas a partir de rotações das demais, é preciso fixar um elemento em uma dada posição e permutar os demais nas posições restantes
- Deste modo, o número de permutações circulares de n elementos distintos é dado por

$$PC(n) = P(n-1) = (n-1)!$$

Enumeração das permutações

- É possível enumerar todas as possíveis permutações de n elementos por meio de *backtracking*
- A função `next_permutation()` da biblioteca `algorithm` do C++ também enumera as permutações distintas
- Ela retorna verdadeiro se é possível gerar a próxima permutação, na ordem lexicográfica, a partir da permutação atual, e falso, caso contrário
- Assim, para enumerar todas as permutações distintas, é preciso começar com a primeira permutação na ordem lexicográfica, que consiste em todos os elementos ordenados

`prev_permutation()`

- A biblioteca `algorithm` também contém a função `prev_permutation()`, que também enumera permutações
- Contudo, ela o faz em sentido oposto em relação à `next_permutation()`
- Assim, para listar todas as permutações distintas usando `prev_permutation()`, é preciso iniciar na última permutação, segundo a ordem lexicográfica
- Ambas funções tem complexidade $O(N)$

Exemplo de enumeração das permutações

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

int main()
{
    vector<int> A { 5, 3, 4, 1, 2 };

    sort(A.begin(), A.end());           // Primeira permutação na ordem lexicográfica

    do {
        for (size_t i = 0; i < A.size(); ++i)
            cout << A[i] << (i + 1 == A.size() ? '\n' : ' ');
    } while (next_permutation(A.begin(), A.end()));

    return 0;
}
```

Permutações em competições

- Listar todas as permutações tem complexidade $O(n \times n!)$
- A enumeração de todas as permutações só é viável para valores pequenos de n (por exemplo, $n \approx 10$)
- Em problemas que envolvam permutações sujeitas a uma série de restrições, caso seja possível, listar todas elas e filtrá-las individualmente é mais simples de implementar do que computar as permutações desejadas diretamente

Problemas

- AtCoder
  1. ABC 103A - Task Scheduling Problem
  2. ABC 123B - Five Dishes
- Codeforces
  1. 222B - Cosmic Tables
  2. 612E - Square Root of Permutation
  3. 961C - Chessboard

Referências

1. **SANTOS**, José Plínio O., **MELLO**, Margarida P., **MURARI**, Idani T. *Introdução à Análise Combinatória*, Editora Ciência Moderna, 2007.