AED III
Pesquisa Completa

Pesquisa Completa

A pesquisa completa é um método geral que pode ser usado para resolver quase qualquer problema de algoritmo. A ideia é gerar todas as soluções possíveis para o problema usando a **força bruta**, e depois selecionar a melhor solução ou contar o número de soluções, dependendo do problema.

A pesquisa completa é uma boa técnica se houver tempo suficiente para passar por todas as soluções, porque a pesquisa geralmente é fácil de implementar e sempre dá a resposta correta.

Gerando Permutações

Consideramos o problema de gerar todas as permutações de um conjunto de n elementos. Por exemplo, as permutações de {0, 1, 2} são (0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0 , 1) e (2, 1, 0). Existem duas abordagens: podemos usar a recursão ou passar pelas permutações iterativamente.

Método 1

Como subconjuntos, permutações podem ser geradas usando recursão. A busca de função a seguir passa pelas permutações do conjunto {0, 1,. . . , n - 1}. A função cria uma permutação vetorial que contém a permutação e a pesquisa começa quando a função é chamada sem parâmetros.

```
void search()
{
  if(permutation.size() == n)
  {
    // process permutation
  }
  else
  {
    for (int i = 0; i < n; i++)
    {
      if (chosen[i]) continue;
      chosen[i] = true;
      permutation.push_back(i);
      search();
      chosen[i] = false;
      permutation.pop_back();
    }
  }
}
```

Cada chamada de função adiciona um novo elemento à permutação. A matriz escolhida indica quais elementos já estão incluídos na permutação. Se o tamanho da permutação for igual ao tamanho do conjunto, uma permutação foi gerada.

Método 2

Outro método para gerar permutações é começar com a permutação {0, 1,. . . , n - 1} e use repetidamente uma função que constrói a próxima permutação em ordem crescente. A biblioteca padrão C++ contém a função next_permutation que pode ser usada para isso:

```
vector<int> permutation;
for (int i = 0; i < n; i++)
{
   permutation.push_back(i);
}
do
{
   // process permutation
} while(next_permutation(permutation.begin(),permutation.end()));
```

Exercício Resolvido

Gerar permutações sempre foi um problema importante na ciência da computação. Neste problema, você terá de gerar todas as permutações de uma dada string, em ordem lexicográfica crescente. Lembre-se que seu algoritmo deve ser eficiente.

Entrada:

A primeira linha da entrada contém um inteiro n, indicando o número de strings que seguem. As próximas n linhas contém uma string cada. Cada string conterá apenas caracteres alfanuméricos, e nunca conterá espaços. O tamanho máximo de uma string é 10.

Exemplo de Entrada	Exemplo de Saída
3 ab abc bca	ab ba abc acb bac bca cab cba abc acb bac bca cab cba

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  int n;
  string line;
  scanf ("%d", &n);
  for (int t = 0; t < n; ++t)
  {
    cin >> line;
    sort(line.begin(), line.end());
    do
    {
        printf("%s\n", line.c_str());
    } while (next_permutation(line.begin(), line.end()));
    printf("\n");
  }
  return 0;
}
```

Se o sort() não existisse?

Imagina a seguinte situação: É preciso ler três variáveis e imprimir estes valores em ordem crescente. Porém, a função sort(), que faz tal ordenação não existe. Como proceder?

Deve-se verificar todas as 6 combinações possíveis (3!)

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  int a,b,c;
  scanf("%d %d %d",&a,&b,&c);
  if((a<=b)&&(b<=c)) printf("%d %d %d\n",a,b,c);
  else if((a<=c)&&(c<=b)) printf("%d %d %d\n",a,c,b);
  else if((b<=a)&&(a<=c)) printf("%d %d %d\n",b,a,c);
  else if((b<=c)&&(c<=a)) printf("%d %d %d\n",b,c,a);
  else if((c<=a)&&(a<=b)) printf("%d %d %d\n",c,a,b);
  else if((c<=b)&&(b<=a)) printf("%d %d %d\n",c,b,a);
}
```

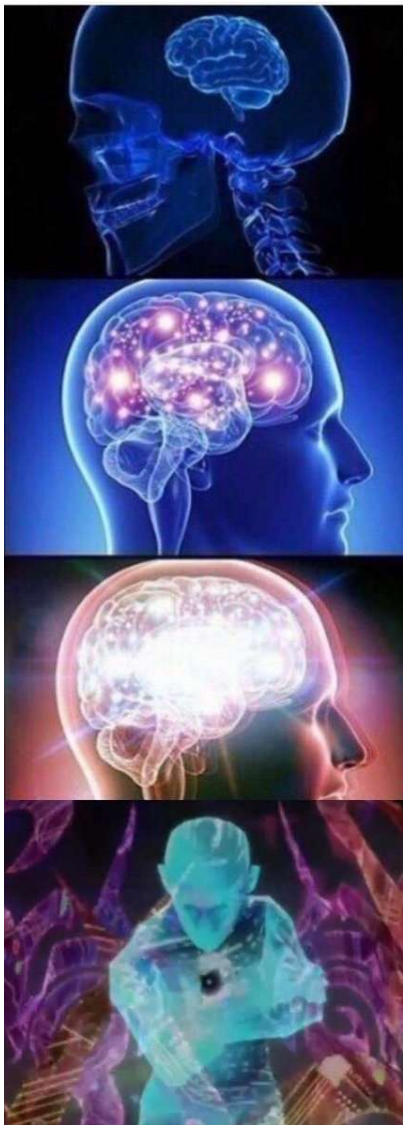
4! = 24

5! = 120

6! = 720

7! = 5040

...



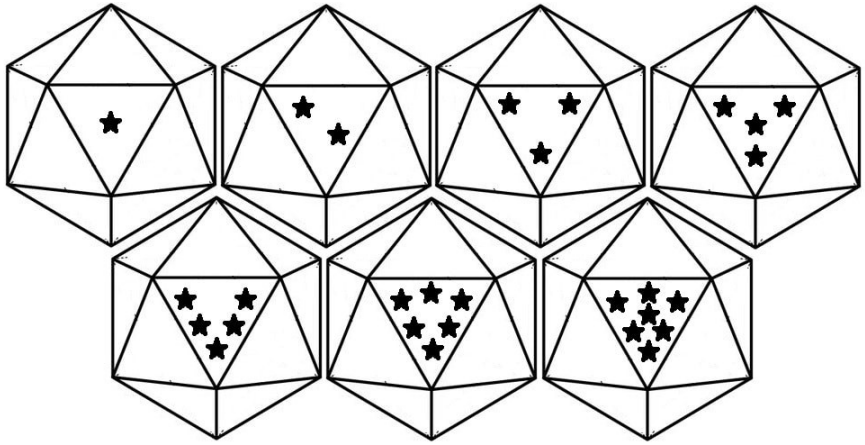
```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  int n;
  string line;
  char name[100][2];
  char resposta[100], r[100];
  char parte[100];
  bool question;
  map<string, int> numbers;
  scanf ("%d", &n);
  for (int t = 0; t < n; ++t)
  {
    cin >> line;
    for(int i=0; i<line.length();i++)
    {
      name[i][0]=line[i];
      name[i][1]='\0’;
      scanf("%d",&numbers[name[i]]);
    }
```

```
do
{
   for(int i=0; i<line.length();i++)
   {
     name[i][0]=line[i];
     name[i][1]='\0’;
   }
   question = true;
   for(int i=0; i<line.length()-1;i++)
   {
     question = question && (numbers[name[i]]<=numbers[name[i+1]]);
   }
   strcpy(resposta,"");
   if(question)
   {
     sprintf(parte,"%d",numbers[name[0]]);
     strcat(resposta,parte);
     for(int i=1; i<line.length();i++)
     {
       sprintf(parte," %d",numbers[name[i]]);
       strcat(resposta,parte);
     }
```

```
            strcat(resposta,"\n");
            strcpy(r,resposta);
          }
      } while (next_permutation(line.begin(), line.end()));
      printf("%s",r);
    }
    return 0;
}
```

Exercício

1. Reza a lenda que os Icosaedros do Lagarto são cristalinos e podem invocar o Lagarto Shen Long Int, que tem a habilidade de conceder desejos para quem conseguir juntar os sete icosaedros. Os icosaedros vêm em conjuntos de sete com cada mostrando o número de estrelas. Estes são difíceis de encontrar porque eles se dispersam ao redor do planeta, e se desativam por um ano a cada desejo, se transformando em pedras. Para ter a oportunidade de invocar o Lagarto, um tem que viajar ao redor do globo para encontrá-las. Certo dia, Kogu ganha o icosaedro de quatro estrelas de seu avô, Hogan. Ao saber da lenda, Kogu resolve ir atrás dos outros icosaedros, para que possa perder o medo de injeção. Como a tecnologia progrediu, isso ficou mais fácil, e bem simples com a invenção de sua amiga Mulba, o Radar do Lagarto. Só que, para ganharem tempo, precisam saber qual ordem de busca dos icosaedros eles devem seguir, de modo que percorram a menor distância possível, com a sua nuvem voadora.



Escreva um programa que, dadas as coordenadas dos icosaedros, informe o caminho de menor distância, saindo de sua casa, capture todos os icosaedros e volte para casa, além de informar a distância de tal caminho.

Exemplo de Entrada	Exemplo de Saída
2 11 11 1 15 14 2 19 17 3 23 20 5 23 2 6 19 5 7 15 8 11 11 7 15 14 6 19 17 5 23 20 3 23 2 2 19 5 1 15 8	Caso 1: 4->1->2->3->5->6->7->4: 48.00000 Caso 2: 4->1->2->3->5->6->7->4: 48.00000