```
/*
SUBCONJUNTOS DE SOMAS IGUAIS A K
O objetivo do algoritmo e:
Dado um inteiro n, um conjunto com n elementos inteiros positivos e um inteiro k, o algoritmo
imprimirá os subconjuntos que sejam iguais a k e k é um número passado pelo usuário
Obs: n, k e o conjunto de inteiros e devem ser positivos
*/

#include<stdio.h>
#include<stdlib.h>

/*
	Realiza a impressão de uma expressão conjunto com tam_conj posições que será igual a k
*/

void printsoma(int *conjunto,int tam_conj,int k){

	int i = 0;

	while(i<tam_conj){
		if(i==0)
			printf("%d",conjunto[i]);
		else
			printf(" + %d",conjunto[i]);
		i++;
	}
	printf(" = %d\n",k);
}

/*
	A função realiza a soma com backtracking dos subconjuntos de inteiros

	int pos		-> posição do elemento a ser somado do conjunto principal
	int soma	-> soma dos elementos
	int k		-> valor de busca definido pelo usuário
	int *conjunto	-> conjunto principal
	int *conjuntoaux -> conjunto auxiliar para impressão dos subconjuntos
	int tam_conj	-> tamanho do conjunto principal
	int posaux	-> posição do elemento no conjunto auxiliar
*/

void somaSub(int pos,int soma,int k,int *conjunto,int *conjuntoaux,int tam_conj,int posaux){

	int tam_conjaux = posaux+1;		// tamanho atual do conjunto auxiliar

	soma += conjunto[pos];
	conjuntoaux[posaux] = conjunto[pos];
	posaux++;

	// caso for encontrado é escrito o subconjunto
	if(soma == k)
		printsoma(conjuntoaux,tam_conjaux,k);
```

```
	// se a soma for maior que o k, aquela ramificação é descartada para otimização do algoritmo
	else if(soma<k){

		// O while faz as possíveis ramificações com os possíveis subconjuntos de soma da
esquerda
		//	para direita, cima para baixo
		while(++pos<tam_conj)
			somaSub(pos,soma,k,conjunto,conjuntoaux,tam_conj,posaux);
	}
}

int main(){

	int tam_conj = 0;	// tamanho do conjunto
	int *conjunto = NULL;	// conjunto
	int *conjuntoaux = NULL; // conjunto auxiliar para guardar os elementos no backtracking
	int k = 0;		// valor mínimo necessário
	int i = 0;

	scanf("%d",&tam_conj);
	getchar();  // '\n'

	conjunto = (int *) malloc(tam_conj*sizeof(int));
	conjuntoaux = (int *) malloc(tam_conj*sizeof(int));

	while(i < tam_conj){
		scanf("%d",&conjunto[i]);
		i++;
	}

	scanf("%d",&k);

	i = 0;

	// para que ocorra todas as ramificações possíveis é usado o comando abaixo
	while(i < tam_conj){
		somaSub(i,0,k,conjunto,conjuntoaux,tam_conj,0);
		i++;
	}

	free(conjunto);
	free(conjuntoaux);

	return 0;

}
```

ENCONTRAR A SAIDA DE UM LABIRINTO USANDO BACKTRACKING

```
#include <stdio.h>
#include <stdbool.h>

#define DIM 4

void print(int mtz[][DIM])
{
   for(int i = 0; i < DIM; i++)
   {
      for(int j = 0; j < DIM; j++)
         printf("%d ", mtz[i][j]);
      printf("\n");
   }
}

bool acharCaminho(int x, int y, int mtz[DIM][DIM], int sol[DIM][DIM])
{
   if(x < 0 || x > DIM-1) //se estiver fora dos limites
      return false;

   if(y < 0 || y > DIM-1) //se estiver fora dos limites
      return false;

   if(mtz[y][x] == 0)     //se for um obstaculo
      return false;

   if(x == DIM-1 && y == DIM-1)  //ENCONTREI A SAIDA
   {
      sol[x][y] = 1;
      return true;
   }

   sol[y][x] = 1;        //ok, caminho me leva a uma possivel solução

   if(acharCaminho(x+1, y, mtz, sol) == true)  //andar para direita(frente)
      return true;

   if(acharCaminho(x, y+1, mtz, sol) == true) //andar para baixo
      return true;

   sol[x][y] = 0; //backtracking (tudo falhou)
   return false;
}

bool begin(int mtz[DIM][DIM])
{
   int sol[DIM][DIM] = {0};
   if(acharCaminho(0,0,mtz, sol) == false)
      return false;
```

```
    print(sol);
    return true;
}

int main(void)
{
    int maze[DIM][DIM] = {0};
    for(int i = 0; i < DIM; i++)
        for(int j = 0; j < DIM; j++)
            scanf(" %d", &maze[i][j]);

    printf("%s\n", begin(maze) == true ? "SUCESSO" : "FRACASSO");
    return 0;
}
```

IMPRIMIR TODOS OS POSSIVEIS SUBCONJUNTOS

```
/*
O objetivo do algoritmo e receber da stdin o tamanho de um conjunto de caracteres e o conjunto,
a partir destes dados, imprimir na stdout todos os possíveis subconjuntos do conjunto usando a
metodologia de resolução de problemas Backtracking
*/

#include<stdio.h>
#include<math.h>
#include<stdlib.h>

void backtracking(int posicao, char *conjunto, char *conj_bool, int tam_conj);

int main(){

        int tam_conj = 0;       // tamanho do conjunto
        char *conj_bool = NULL; // conjunto de boolenos
        char *conjunto = NULL;  // conjunto
        int i = 0;

        scanf("%d",&tam_conj);
        getchar();  // '\n'

        conj_bool = (char *) malloc(tam_conj*sizeof(char));
        conjunto  = (char *) malloc(tam_conj*sizeof(char));

        while(i < tam_conj){
                scanf("%c",&conjunto[i]);
                i++;
        }

        backtracking(0,conjunto,conj_bool,tam_conj);

        return 0;
```

```
}
/*
	int posicao     -> indice atual de arranjos a serem modificados na função
	char *conjunto  -> arranjo de caracteres, ou elementos do conjunto
	char *conj_bool -> arranjo de valores de verdade para realizar a impressão dos subconjuntos
do conjunto
	int tam_conj    -> tamanho do conjunto a ser impressos seus subconjuntos
*/
void backtracking(int posicao, char *conjunto, char *conj_bool, int tam_conj){

	int i = 0;
	int j = 0;

	if(posicao == tam_conj){
		while(i < tam_conj){
			if(conj_bool[i] == 1){
				printf("%c",conjunto[i]);
				j = 1;
			}
			i++;
		}
		if(j == 1)
			printf("\n");
	}

	else{
		conj_bool[posicao] = 1;
		backtracking(posicao+1,conjunto,conj_bool,tam_conj);
		conj_bool[posicao] = 0;
		backtracking(posicao+1,conjunto,conj_bool,tam_conj);
	}
}
```

IMPRIMIR TODAS AS PERMUTAÇÕES(INCLUSIVE COM REPETIÇÕES) DE UMA DADA STRING

```
#include <stdio.h>
#include <string.h>

//troca de valores entre ponteiros
void swap(char *a, char *b)
{
    char temp = *a;
    *a = *b;
    *b = temp;
}

/* Imprime todas as permutações de uma da string
   1. String
   2. indice do primeiro elemento da string
   3. indice do ultimo elemento da string
```

```
*/
void permute(char *s, int l, int r)
{
   int i = 0;
   if (l == r)
     printf("%s\n", s);
   else
   {
       for (i = l; i <= r; i++)
       {
          swap((s+l), (s+i));
          permute(s, l+1, r);
          swap((s+l), (s+i)); //backtrack
       }
   }
}

int main()
{
    char str[] = "ABC";
    int n = strlen(str);
    permute(str, 0, n-1);
    return 0;
}
```