#include <stdio.h>

#include <stdlib.h>

// Estrutura para representar uma solução e funções relacionadas.

typedef struct {

int \* notas;

int tam;

} Solucao;

Solucao \* nova\_solucao(){

Solucao \* sol = (Solucao \*) malloc(sizeof(Solucao));

sol->notas = (int \*) malloc(20 \* sizeof(int));

sol->tam = 0;

return sol;

}

Solucao \* copia(Solucao \* sol){

int i;

Solucao \* nova = NULL;

if(sol){

nova = (Solucao \*) malloc(sizeof(Solucao));

nova->notas = (int \*) malloc(20 \* sizeof(int));

nova->tam = sol->tam;

for(i = 0; i < nova->tam; i++) nova->notas[i] = sol->notas[i];

}

return nova;

}

Solucao \* adiciona(Solucao \* sol, int x){

if(sol){

sol->notas[sol->tam] = x;

sol->tam++;

}

return sol;

}

Solucao \* melhor(Solucao \* a, Solucao \* b){

if(a && b) return a->tam <= b->tam ? a : b;

else return a ? a : b;

}

void print(Solucao \* sol){

int i;

if(sol){

printf("Solução:");

for(i = 0; i < sol->tam; i++){

printf(" %d", sol->notas[i]);

}

printf("\n");

}

else {

printf("Sem solução!\n");

}

}

// implementação gulosa para o problema do troco.

Solucao \* troco\_guloso(int n, int \* v, int k){

int i;

Solucao \* sol = nova\_solucao();

for(i = 0; i < k; i++){

while(n - v[i] >= 0) {

adiciona(sol, v[i]);

n -= v[i];

}

}

return sol;

}

// implementação tentativa e erro.

Solucao \* troco(int n, int \* v, int k){

int i;

Solucao \* m = NULL;

if(n == 0) return nova\_solucao();

for(i = 0; i < k; i++){

if(n - v[i] >= 0){

m = melhor( adiciona( troco(n - v[i], v, k), v[i] ), m );

}

}

return m;

}

// implementação iterativa usando progrmação dinâmica (solução bottom-up)

Solucao \* troco\_din\_it(int n, int \* v, int k){

int s, i;

Solucao \*\* solucoes = (Solucao \*\*) malloc((n + 1) \* sizeof(Solucao \*));

Solucao \* m;

solucoes[0] = nova\_solucao();

for(s = 1; s <= n; s++){

m = NULL;

for(i = 0; i < k; i++){

if(s - v[i] >= 0){

m = melhor( adiciona( copia(solucoes[s - v[i]]), v[i] ), m );

}

}

solucoes[s] = m;

}

return solucoes[n];

}

// implementação recursiva usando progrmação dinâmica (solução top-down)

Solucao \* troco\_din\_rec(int n, int \* v, int k, Solucao \*\* solucoes){

int i;

Solucao \* m = NULL;

if(!solucoes[n]){

if(n == 0){

solucoes[n] = nova\_solucao();

}

else {

for(i = 0; i < k; i++){

if(n - v[i] >= 0){

m = melhor( adiciona( copia(troco\_din\_rec(n - v[i], v, k, solucoes)), v[i] ), m );

}

}

solucoes[n] = m;

}

}

return solucoes[n];

}

Solucao \* troco\_din\_rec\_init(int n, int \* v, int k){

int i;

Solucao \*\* solucoes = (Solucao \*\*) malloc((n + 1) \* sizeof(Solucao \*));

for(i = 0; i <= n; i++) solucoes[i] = NULL;

return troco\_din\_rec(n, v, k, solucoes);

}

int main(){

int v1[] = { 50, 20, 10, 5, 2, 1 };

int v2[] = { 50, 20 };

int v3[] = { 4, 3, 1 };

print(troco\_guloso(80, v1, 6));

print(troco\_guloso(80, v2, 2));

print(troco\_guloso(6, v3, 3));

printf("----------------------------------------------------------\n");

print(troco(80, v1, /\*\*/ 4 /\*\*/)); // <--- cuidado com esse parâmetro, pois a versão tentativa e erro implementada, além de demorada, possui um vazamento de memória sério!

print(troco(80, v2, 2));

print(troco(6, v3, 3));

printf("----------------------------------------------------------\n");

print(troco\_din\_it(80, v1, 6));

print(troco\_din\_it(80, v2, 2));

print(troco\_din\_it(6, v3, 3));

printf("----------------------------------------------------------\n");

print(troco\_din\_rec\_init(80, v1, 6));

print(troco\_din\_rec\_init(80, v2, 2));

print(troco\_din\_rec\_init(6, v3, 3));

return 0;

}