

Ordenação de arrays de cadeias e de arrays de estruturas

Pedro Guerreiro

Universidade do Algarve

8 de março de 2021

Há um aforismo de programação que diz: “se não sabes o que fazer, ordena.” Ordena o array, bem entendido.

De facto, quando trabalhamos com arrays no âmbito de problemas de programação, frequentemente surge a necessidade de os ordenar. Daí, o aforismo: se não sabemos o que fazer ainda, ordenamos o array logo, e assim adiantamos as coisas...

Com arrays de números, já estudámos dois algoritmos de ordenação: o *insertion-sort* e o *mergesort*. Vamos agora descobrir como usar esses algoritmos com arrays de cadeias e com arrays de estruturas.

Voos

O tema desta divagação são os voos à partida num certo aeroporto. Concretizando, à data em que escrevo, observo no sítio do aeroporto de Lisboa, a seguinte lista de voos:

Day	Time	Terminal	Flight Number		Destination	Airline	Status
06/03/2021	05:00	T1	KL 1692	+5	Amsterdam	KLM	Departed 05:12
06/03/2021	05:30	T1	AF 1125	+9	Paris, Ch. de Gaulle	AIR FRANCE	Departed 05:54
06/03/2021	06:00	T1	S4 121		Ponta Delgada	AZORES AIRLINES	Departed 06:07
06/03/2021	08:00	T1	S4 141		Pico	AZORES AIRLINES	Departed 08:07
06/03/2021	08:25	T1	IB 8765		Madrid	IBERIA	Departed 08:26
06/03/2021	08:35	T1	DS 1446		Geneva	EASYJET SWITZERLAND	Departed 08:34
06/03/2021	09:10	T1	TP 1545		Praia	TAP PORTUGAL	Departed 09:36
06/03/2021	09:25	T1	TP 436	+1	Paris, Orly	TAP PORTUGAL	Departed 09:29

Isto é só a primeira parte da lista, com os primeiros voos da manhã (que, aliás, já partiram, à hora em que escrevo).

Na verdade, através do *browser*, fui capaz de criar uma folha de cálculo com a lista de voos. A seguinte figura mostra a parte inicial dessa folha:

	A	B	C	D	E	F	G
1	06/03/2021	05:00	T1	KL 16925	Amsterdam	KLM	Departed 05:12
2	06/03/2021	05:30	T1	AF 11259	Paris, Ch. de Gaulle	AIR FRANCE	Departed 05:54
3	06/03/2021	06:00	T1	S4 121	Ponta Delgada	AZORES AIRLINES	Departed 06:07
4	06/03/2021	08:00	T1	S4 141	Pico	AZORES AIRLINES	Departed 08:07
5	06/03/2021	08:25	T1	IB 8765	Madrid	IBERIA	Departed 08:26
6	06/03/2021	08:35	T1	DS 1446	Geneva	EASYJET SWITZERLAND	Departed 08:34
7	06/03/2021	09:10	T1	TP 1545	Praia	TAP PORTUGAL	Departed 09:36
8	06/03/2021	09:25	T1	TP 4361	Paris, Orly	TAP PORTUGAL	Departed 09:29

Vemos que as colunas correspondem às colunas da tabela na página: dia, hora, terminal, número de voo, destino, companhia aérea e estado do voo.

Há, ao todo, 45 voos, no dia de hoje, 6 de março de 2021¹.

Com esta lista de voos, queremos, como exercício, calcular o conjunto das companhias aéreas que operam no aeroporto de Lisboa. Mais precisamente, quere-

¹ É um número pequeno, mas estamos em pandemia e há pouca gente a viajar, 🤔.

mos criar um array com os nomes das companhias aéreas que constam na lista de voos. No array, os nomes devem estar por ordem alfabética, sem repetições.

Precisamos, para começar, de um tipo para representar os voos. De entre os elementos de informação presentes na folha de cálculo, dispensamos o dia, o terminal e o estado do voo. Ficamos com a hora de partida, o número de voo, o aeroporto de destino e a companhia aérea. O número de voo, apesar de se chamar “número” é uma cadeia de caracteres. O destino e a companhia aérea também são cadeias de caracteres. A hora de partida, que surge na forma de cadeia de caracteres, por exemplo `08:35`, ganha em ser representada internamente por um número. Aliás, antecipando a necessidade de calcular o intervalo entre voos consecutivos, percebemos que convirá exprimir a hora de partida por meio do número de minutos que decorreram desde a meia-noite.

Logo, o tipo para os voos poderá ficar assim:

```
typedef struct {
    int departure;
    const char *code;
    const char *destination;
    const char *airline;
} Flight;
```

Equipemo-lo com as operações habituais: construtor e funções de escrita:

```
Flight flight(int departure, const char *code,
              const char* destination, const char *airline)
{
    Flight result = {departure, code, destination, airline};
    return result;
}

void flight_print(FILE *f, Flight x)
{
    fprintf(f, "[%d]<%s><%s><%s>",
            x.departure, x.code, x.destination, x.airline);
}

void flight_println(FILE *f, Flight x)
{
    flight_print(f, x);
    fprintf(f, "\n");
}
```

}

Apesar de, para calcular apenas as companhias aéreas não serem precisos os outros campos, o melhor é mesmo guardar toda a informação num array de voos, de onde depois “extrairemos” as companhias.

Para criar esse array, temos de ser capazes de ler os dados que estão na folha de cálculo. Ora, em geral, aos ficheiros que contêm as folhas de cálculo estão num formato próprio, ilegível. Portanto, é indispensável começar por criar uma cópia da folha possa sim ser lida diretamente como texto.

Na prática, usa-se o formato CSV, do inglês “*comma-separated values*”. Eis as oito primeiras linhas da folha de cálculo, neste formato:

```
06/03/2021,05:00,T1,KL 16925,Amsterdam,KLM,Departed  
05:12,06/03/2021  
06/03/2021,05:30,T1,AF 11259,"Paris, Ch. de Gaulle",AIR  
FRANCE,Departed 05:54,06/03/2021  
06/03/2021,06:00,T1,S4 121,Ponta Delgada,AZORES  
AIRLINES,Departed 06:07,06/03/2021  
06/03/2021,08:00,T1,S4 141,Pico,AZORES AIRLINES,Departed  
08:07,06/03/2021  
06/03/2021,08:25,T1,IB 8765,Madrid,IBERIA,Departed  
08:26,06/03/2021  
06/03/2021,08:35,T1,DS 1446,Geneva,EASYJET  
SWITZERLAND,Departed 08:34,06/03/2021  
06/03/2021,09:10,T1,TP 1545,Praia,TAP PORTUGAL,Departed  
09:36,06/03/2021  
06/03/2021,09:25,T1,TP 4361,"Paris, Orly",TAP  
PORTUGAL,Departed 09:29,06/03/2021
```

Constatamos que, tal como a expressão *comma-separated values* sugere, os campos estão separados cada um do seguinte por meio de vírgulas. Há uma complicação quando os valores eles próprios têm vírgulas. Nesse caso são colocados entre aspas, por exemplo, "Paris, Ch. de Gaulle". Sem as aspas, a vírgula a seguir a Paris seria interpretada com terminando o valor, mas não queremos isso.

Portanto, o problema que temos entre mãos neste momento é ler um ficheiro CSV para um array de voos. O protótipo da função será:

```
int flights_read(FILE *f, Flight *a);
```

Leremos linha a linha. Para cada linha criamos um array de cadeias, uma cadeia para cada valor, separando as cadeias pelas vírgulas. Depois selecionamos as ca-

deias que nos interessam e criamos um objeto de tipo `Flight`, ajustando consoante necessário.

Esta análise põe em evidência um novo subproblema: dada uma cadeia, parti-oná-la em subcadeias, separadas por vírgulas. Aliás, para mais generalidade, podemos parametrizar o separador: hoje é vírgula, amanhã poderá se ponto e vírgula, depois de amanhã pode ser uma barra, etc. Eis o protótipo:

```
int str_split(const char *s, char separator, const char **a)
```

Usaremos a técnica do *count-while*: contamos os caracteres desde o início da cadeia até à primeira vírgula; selecionamos a subcadeia inicial com o número de caracteres contados, copiamos dinamicamente para o array; depois saltamos a vírgula, se houver; e recomeçamos com o resto da cadeia, isto é, com a subcadeia que começa logo a seguir à vírgula.

Há uma complicação, no entanto: se aparecer uma vírgula numa cadeia entre aspas, essa não deve contar. Precisamos, portanto, de uma função *count-while* especializada: se o primeiro caractere da cadeira não for o sinal de aspas, então conta até à vírgula; se for, então conta até à vírgula que vem a seguir às próximas aspas, inclusive:

```
int str_count_while_csv(const char *s, char separator)
{
    int result = 0;
    if (*s == '"')
    {
        result = 1 + str_count_while_not(s+1, '"');
        if (s[result] == '"')
            result++;
    }
    else
        result = str_count_while_not(s, separator);
    return result;
}
```

Esta função é subtil. Não descuremos a respetiva função de teste unitário:

```
void unit_test_str_count_while_csv(void)
{
    const char *s1 = "01234,abcdefghi,xxx";
```

```

assert(str_count_while_csv(s1, ',') == 5);
assert(str_count_while_csv(s1+6, ',') == 9);
assert(str_count_while_csv(s1+6+10, ',') == 3);
const char *s2 = ".,.,.";
assert(str_count_while_csv(s2, ',') == 0);
assert(str_count_while_csv(s2+1, ',') == 0);
assert(str_count_while_csv(s2+2, ',') == 0);
assert(str_count_while_csv(s2+3, ',') == 0);
const char *s3 = "\"abcde\",1234567890,";
assert(str_count_while_csv(s3, ',') == 7);
assert(str_count_while_csv(s3+8, ',') == 10);
assert(str_count_while_csv(s3+8+11, ',') == 0);
}

```

Com recurso à função `str_count_while_csv`, a função `str_split` tem o aspeto habitual:

```

int str_split(const char *s, char separator, const char **a)
{
    int result = 0;
    int i = 0;
    while (s[i])
    {
        int x = str_count_while_csv(s+i, separator);
        a[result++] = str_ndup(s+i, x);
        i += x;
        if (s[i] == separator)
            i++;
    }
    return result;
}

```

Também vale a pena considerar a função de teste unitário:

```

void unit_test_str_split(void)
{
    const char *s1 = "01234,abcdefghi,xxx";
    const char *a1[10];
    int n1 = str_split(s1, ',', a1);
    const char *b1[3] = {"01234", "abcdefghi", "xxx"};
    assert(strings_equal(a1, n1, b1, 3));
    const char *s2 = ".,.,.";
    const char *a2[10];
    int n2 = str_split(s2, ',', a2);
    const char *b2[4] = {"", "", "", ""};
    assert(strings_equal(a2, n2, b2, 4));
}

```

```

const char *s3 = "\"abcde\",1234567890,";
const char *a3[10];
int n3 = str_split(s3, ',', a3);
const char *b3[2] = {"\"abcde\"", "1234567890"};
assert(strings_equal(a3, n3, b3, 2));
}

```

A função `strings_equal`, aqui usada, verifica se dois arrays de cadeias de caracteres são iguais, e é análoga a funções que estudamos para arrays de números:

```

int strings_equal(const char **a, const int n, const char **b,
int m)
{
    int result = n == m;
    int i = 0;
    while (result && i < n)
        if (strcmp(a[i], b[i]) != 0)
            result = 0;
        else
            i++;
    return result;
}

```

Ainda assim, repare na função de biblioteca `strcmp`, usada para comparar cadeias de caracteres. Ela dará zero como resultado se as duas cadeias forem iguais, dará um valor negativo se a primeira for “menor” que a segunda e dará um valor positivo se a primeira for “maior” que a segunda. Nesta frase, “menor” não significa menor em comprimento, mas sim menor “lexicograficamente”. Por outras palavras, quando dizemos que uma cadeia é “menor” que outra, queremos significar que a palavra representada por essa cadeia viria antes da palavra representada pela outra, num dicionário. Analogamente para “maior”. De facto, é esta ordenação que nos convém.

Testemos a função `str_split` com o ficheiro CSV de voos.

```

void test_str_split(void)
{
    const char *a[1000];
    int n = strings_read(stdin, a);
    for (int i = 0; i < n; i++)
    {
        const char *b[8];
        int m = str_split(a[i], ',', b);
    }
}

```

```

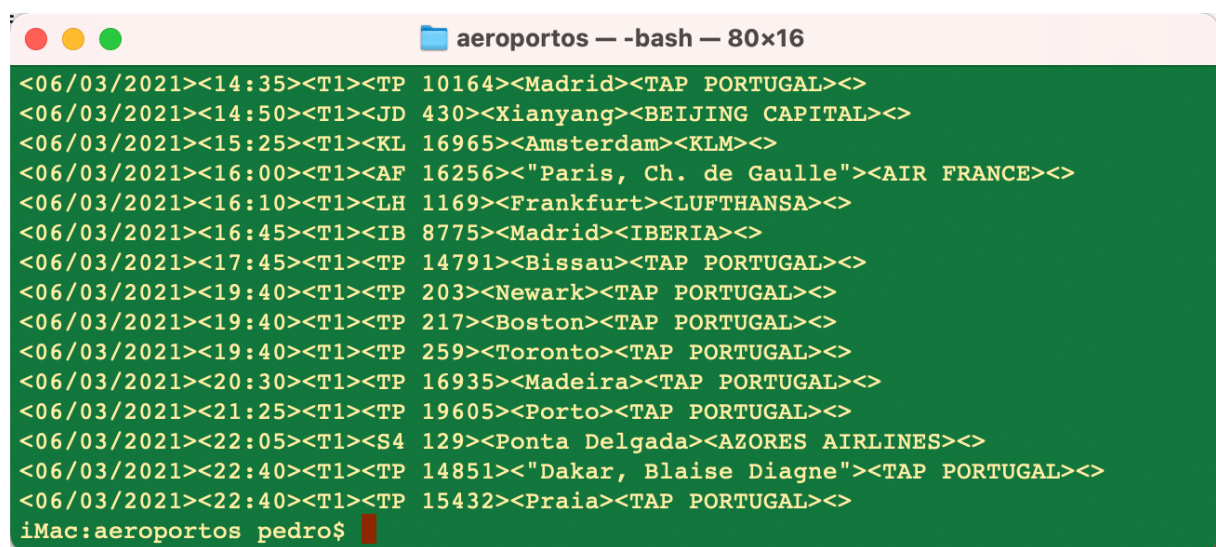
    strings_fprintf(stdout, b, m, "<%s>");
}
}

```

Fixando que a esta função corresponderá a opção **A** na linha de comando, invocaremos o programa, na diretoria de trabalho, assim:

```
$ ../../sources/a.out A < partidas_lisboa_20210306.csv
```

O programa escreve 45 linhas, uma para cada voo. Após o teste, só ficam visíveis na consola as últimas linhas escritas:



```

<06/03/2021><14:35><T1><TP 10164><Madrid><TAP PORTUGAL><>
<06/03/2021><14:50><T1><JD 430><Xianyang><BEIJING CAPITAL><>
<06/03/2021><15:25><T1><KL 16965><Amsterdam><KLM><>
<06/03/2021><16:00><T1><AF 16256><"Paris, Ch. de Gaulle"><AIR FRANCE><>
<06/03/2021><16:10><T1><LH 1169><Frankfurt><LUFTHANSA><>
<06/03/2021><16:45><T1><IB 8775><Madrid><IBERIA><>
<06/03/2021><17:45><T1><TP 14791><Bissau><TAP PORTUGAL><>
<06/03/2021><19:40><T1><TP 203><Newark><TAP PORTUGAL><>
<06/03/2021><19:40><T1><TP 217><Boston><TAP PORTUGAL><>
<06/03/2021><19:40><T1><TP 259><Toronto><TAP PORTUGAL><>
<06/03/2021><20:30><T1><TP 16935><Madeira><TAP PORTUGAL><>
<06/03/2021><21:25><T1><TP 19605><Porto><TAP PORTUGAL><>
<06/03/2021><22:05><T1><S4 129><Ponta Delgada><AZORES AIRLINES><>
<06/03/2021><22:40><T1><TP 14851><"Dakar, Blaise Diagne"><TAP PORTUGAL><>
<06/03/2021><22:40><T1><TP 15432><Praia><TAP PORTUGAL><>
iMac:aeroportos pedro$

```

Confirmamos que cada linha tem sete campos. Só nos interessam o segundo, o quarto, o quinto e o sexto. Mesmo assim, o segundo, com a hora de partida, interessa-nos, mas na forma de um número inteiro e não nesta forma textual. Portanto, precisamos de uma função para calcular o número de minutos, um valor de tipo **int**, a partir da cadeia de caracteres:

```

int minutes_from_timestring(const char *s)
{
    assert(strlen(s) >= 5);
    assert(s[2] == ':');
    int h = atoi(s);
    int m = atoi(s+3);
    return h * 60 + m;
}

```

A função de biblioteca **atoi** converte um cadeia para número inteiro. Aqui precisamos de converter duas cadeias, uma para as horas e outra para os minutos.

Na verdade, a função `atoi`, “vai convertendo” enquanto consegue, isto é, até ao fim da cadeia ou até aparecer um caractere que não é um algarismo. É isso que nos convém: a primeira chamada, `atoi(s)`, apanha os algarismos da hora (até ao sinal de dois pontos) e a segunda, `atoi(s+3)`, apanha os algarismos dos minutos (até ao fim da cadeia).

O quarto campo, que identifica o aeroporto de destino, é o tal que no ficheiro às vezes vem entre aspas. Ora, não queremos as aspas no nosso processamento.

Para as retirar, usamos a seguinte função:

```
// Return a dynamic copy of `s` without the initial quote,
// which must exist, and the final quote, if it exists
const char* str_dup_unquoting(const char *s)
{
    assert(*s == '"');
    int x = (int)strlen(s);
    if (s[x] == '"') x--;
    return str_ndup(s+1, x-2); // two quote chars will be
    removed.
}
```

Na verdade, não retiramos as aspas: criamos é uma cópia sem as aspas.

Já temos todos os ingredientes para programar a leitura dos voos, do ficheiro para o array:

```
int flights_read(FILE *f, Flight *a)
{
    int result = 0;
    char line[max_line_length];
    while (str_readline(f, line) != EOF)
    {
        const char *w[16];
        int m = str_split(line, ',', w);
        assert(m == 7);
        int minutes = minutes_from_timestring(w[1]);
        const char *s = w[4];
        if (*s == '"')
            s = str_dup_unquoting(w[4]);
        a[result++] = flight(minutes, w[3], s, w[5]);
    }
    return result;
}
```

Experimentemos, com a seguinte função de teste:

```
void test_flights_read_write(void)
{
    Flight a[max_flights];
    int n = flights_read(stdin, a);
    flights_print(stdout, a, n);
}
```

A função `flights_print` escreve o array de voos, recorrendo à função `flight_print`, para escrever cada um dos voos:

```
void flights_print(FILE *f, Flight *a, int n)
{
    for (int i = 0; i < n; i++)
        flight_println(f, a[i]);
}
```

Eis o resultado da experiência na consola:

A terminal window titled 'aeroportos — -bash — 80x16' with a green background. It displays a list of flight records, each on a new line. The records are: [875]<TP 10164><Madrid><TAP PORTUGAL>, [890]<JD 430><Xianyang><BEIJING CAPITAL>, [925]<KL 16965><Amsterdam><KLM>, [960]<AF 16256><Paris, Ch. de Gaulle><AIR FRANCE>, [970]<LH 1169><Frankfurt><LUFTHANSA>, [1005]<IB 8775><Madrid><IBERIA>, [1065]<TP 14791><Bissau><TAP PORTUGAL>, [1180]<TP 203><Newark><TAP PORTUGAL>, [1180]<TP 217><Boston><TAP PORTUGAL>, [1180]<TP 259><Toronto><TAP PORTUGAL>, [1230]<TP 16935><Madeira><TAP PORTUGAL>, [1285]<TP 19605><Porto><TAP PORTUGAL>, [1325]<S4 129><Ponta Delgada><AZORES AIRLINES>, [1360]<TP 14851><Dakar, Blaise Diagne><TAP PORTUGAL>, and [1360]<TP 15432><Praia><TAP PORTUGAL>. The prompt 'iMac:aeroportos pedro\$' is visible at the bottom.

```
iMac:aeroportos pedro$ [875]<TP 10164><Madrid><TAP PORTUGAL>
[890]<JD 430><Xianyang><BEIJING CAPITAL>
[925]<KL 16965><Amsterdam><KLM>
[960]<AF 16256><Paris, Ch. de Gaulle><AIR FRANCE>
[970]<LH 1169><Frankfurt><LUFTHANSA>
[1005]<IB 8775><Madrid><IBERIA>
[1065]<TP 14791><Bissau><TAP PORTUGAL>
[1180]<TP 203><Newark><TAP PORTUGAL>
[1180]<TP 217><Boston><TAP PORTUGAL>
[1180]<TP 259><Toronto><TAP PORTUGAL>
[1230]<TP 16935><Madeira><TAP PORTUGAL>
[1285]<TP 19605><Porto><TAP PORTUGAL>
[1325]<S4 129><Ponta Delgada><AZORES AIRLINES>
[1360]<TP 14851><Dakar, Blaise Diagne><TAP PORTUGAL>
[1360]<TP 15432><Praia><TAP PORTUGAL>
iMac:aeroportos pedro$
```

Agora que já temos o array com os voos, fica relativamente simples construir a lista das companhias aéreas: a partir do array dos voos, construímos o array dos nomes das companhias aéreas. Nesse array, haverá nomes repetidos. Ordenamos o array. Ao ordenar, os nomes repetidos ficarão de seguida. Usando o esquema de remoção de duplicados, que conhecemos desde os tempos dos arrays de números inteiros, logo conseguimos o resultado pretendido.

Para obter o array dos nomes das companhias, fazemos assim:

```

int get_airlines(const Flight *a, int n, const char **b)
{
    int result = 0;
    for (int i = 0; i < n; i++)
        b[result++] = a[i].airline;
    return result;
}

```

Testemos:

```

void test_get_airlines(void)
{
    Flight a[max_flights];
    int n = flights_read(stdin, a);
    const char *b[n];
    int m = get_airlines(a, n, b);
    strings_fprintfn(stdout, b, m, "<%s>");
}

```

Eis o resultado na consola:



```

iMac:aeroportos pedro$ ../../sources/a.out C < partidas_lisboa_20210306.csv
<KLM><AIR FRANCE><AZORES AIRLINES><AZORES AIRLINES><IBERIA><EASYJET SWITZERLAND>
<TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTU
GAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP P
ORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TURKISH AIRLINES><AIR EUROPA><TAP PORTUGAL>
<TAP PORTUGAL><TAP PORTUGAL><AIR FRANCE><TAP PORTUGAL><TAP PORTUGAL><EMIRATES><T
AP PORTUGAL><TAP PORTUGAL><BEIJING CAPITAL><KLM><AIR FRANCE><LUFTHANSA><IBERIA><
TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUG
AL><AZORES AIRLINES><TAP PORTUGAL><TAP PORTUGAL>
iMac:aeroportos pedro$

```

Para ordenar arrays de cadeias de caracteres, vamos usar, neste exercício, o algoritmo *insertionsort*. Com arrays de inteiros, era assim:

```

void ints_isort(int *a, int n)
{
    for (int i = 1; i < n; i++)
    {
        int j = i;
        while (j > 0 && a[j-1] > a[j])
        {
            ints_exchange(a, j-1, j);
            j--;
        }
    }
}

```

```
}
```

A função `ints_exchange` havia sido definida anteriormente:

```
void ints_exchange(int *a, int x, int y)
{
    int m = a[x];
    a[x] = a[y];
    a[y] = m;
}
```

Com arrays de cadeias de caracteres, é parecido. Precisamos de uma função para trocar dois elementos num array de cadeias de caracteres:

```
void strings_exchange(const char **a, int x, int y)
{
    const char *m = a[x];
    a[x] = a[y];
    a[y] = m;
}
```

A função de ordenação terá o seguinte protótipo:

```
void strings_isort(const char **a, int n);
```

No corpo da função, onde antes estava `ints_exchange`, agora será `strings_exchange`. A única diferença notável reside na maneira de comparar os dois elementos `a[j-1]` e `a[j]` na condição da instrução `while`. Não se usa o operador `>`, que, sendo aplicável, não tem o sentido pretendido, mas sim a função `strcmp`, que vimos há pouco. Para exprimir que `a[j-1]` é “maior” `a[j]` (maior em sentido lexicográfico, bem entendido), escrevemos agora `strcmp(a[j-1], a[j]) > 0`. Portanto, a função fica assim:

```
void strings_isort(const char **a, int n)
{
    for (int i = 1; i < n; i++)
    {
        int j = i;
        while (j > 0 && strcmp(a[j-1], a[j]) > 0)
        {
            strings_exchange(a, j-1, j);
            j--;
        }
    }
}
```

```

    }
}

```

Recorde que as cadeias do array estão em memória dinâmica. No array existem os endereços das cadeias. Quando trocamos no array, trocamos os endereços, não as cadeias propriamente ditas. Aliás, as cadeias ficam sossegadinhas em memória, nunca saindo das posições que ocupam.

A função de teste é análoga à anterior, acrescentando a ordenação:

```

void test_strings_isort(void)
{
    Flight a[max_flights];
    int n = flights_read(stdin, a);
    const char *b[n];
    int m = get_airlines(a, n, b);
    strings_isort(b, m);
    strings_fprintfn(stdout, b, m, "<%s>");
}

```

Confirmamos que as companhias aéreas agora vêm por ordem alfabética, mas ainda repetidas:

```

iMac:aeroportos pedro$ ../../sources/a.out D < partidas_lisboa_20210306.csv
<AIR EUROPA><AIR FRANCE><AIR FRANCE><AIR FRANCE><AZORES AIRLINES><AZORES AIRLINE
S><AZORES AIRLINES><BEIJING CAPITAL><EASYJET SWITZERLAND><EMIRATES><IBERIA><IBER
IA><KLM><KLM><LUFTHANSA><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL>
<TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTU
GAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP P
ORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><T
AP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGAL><TAP PORTUGA
L><TAP PORTUGAL><TAP PORTUGAL><TURKISH AIRLINES>
iMac:aeroportos pedro$

```

O esquema de remoção de duplicados é um clássico da programação:

```

int strings_unique(const char **a, int n, const char **b)
{
    int result = 0;
    int i = 0;
    while (i < n)
    {
        int z = strings_count_while(a+i, n-i, a[i]);
        b[result++] = a[i];
        i += z;
    }
}

```

```

    }
    return result;
}

```

Com isto, a função que a partir de uma array de voos cria a o array ordenado das companhias aéreas sem duplicados fica assim:

```

int unique_airlines(const Flight *a, int n, const char **b)
{
    int result = get_airlines(a, n, b);
    strings_isort(b, result);
    result = strings_unique(b, result, b);
    return result;
}

```

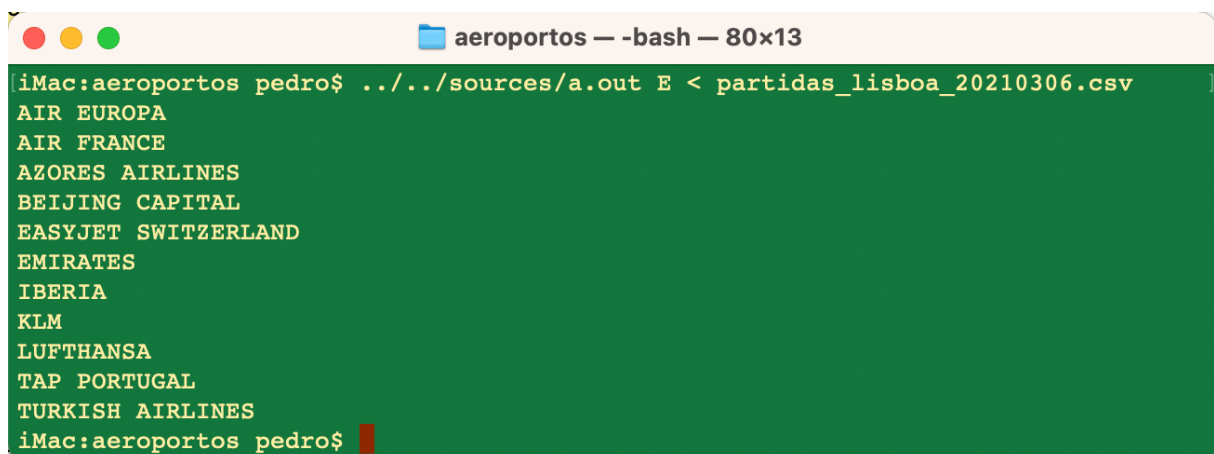
Para concluir, a respetiva função de teste:

```

void test_unique_airlines(void)
{
    Flight a[max_flights];
    int n = flights_read(stdin, a);
    const char *b[n];
    int m = unique_airlines(a, n, b);
    strings_fprint_basic(stdout, b, m);
}

```

Eis o resultado, com uma companhia por linha:



```

aeroportos — -bash — 80x13
iMac:aeroportos pedro$ ../../sources/a.out E < partidas_lisboa_20210306.csv
AIR EUROPA
AIR FRANCE
AZORES AIRLINES
BEIJING CAPITAL
EASYJET SWITZERLAND
EMIRATES
IBERIA
KLM
LUFTHANSA
TAP PORTUGAL
TURKISH AIRLINES
iMac:aeroportos pedro$

```

Neste exercício, ordenámos o array dos nomes das companhias aéreas, que havíamos extraído do array dos voos. Como fazer para ordenar o próprio array dos voos?

A técnica é análoga. Usemos, como exemplo, o mesmo algoritmo, o *insertionsort*, para ordenar o array pelo aeroporto de destino. O protótipo da função de ordenação será:

```
void flights_sort_by_destination(Flight *a, int n);
```

Precisamos de uma função para trocar dois voos, no array de voos:

```
void flights_exchange(Flight *a, int x, int y)
{
    Flight m = a[x];
    a[x] = a[y];
    a[y] = m;
}
```

A comparação dos voos é feita mediante os membros *destination*, que são cadeias de caracteres. Observe:

```
void flights_sort_by_destination(Flight *a, int n)
{
    for (int i = 1; i < n; i++)
    {
        int j = i;
        while (j > 0 && strcmp(a[j-1].destination,
                               a[j].destination) > 0)
        {
            flights_exchange(a, j-1, j);
            j--;
        }
    }
}
```

E é tudo.

Confirmemos, com a seguinte função de teste:

```
void test_sort_by_destination(void)
{
    Flight a[max_flights];
    int n = flights_read(stdin, a);
    flights_print(stdout, a, n);
    printf("-----\n");
    flights_sort_by_destination(a, n);
    flights_print(stdout, a, n);
}
```

```
}
```

Para melhor controlo, a função escreve o ficheiro antes de ordenar e depois de ordenar. Como o resultado é relativamente extenso, é melhor guardar o output num ficheiro:

```
$ ./../sources/a.out F < partidas_lisboa_20210306.csv > out.txt
```

Eis as primeira linhas da parte ordenada.

```
[300]<KL 16925><Amsterdam><KLM>
[700]<TP 6746><Amsterdam><TAP PORTUGAL>
[925]<KL 16965><Amsterdam><KLM>
[720]<TP 10385><Barcelona><TAP PORTUGAL>
[1065]<TP 14791><Bissau><TAP PORTUGAL>
[1180]<TP 217><Boston><TAP PORTUGAL>
[595]<TP 6465><Brussels><TAP PORTUGAL>
[1360]<TP 14851><Dakar, Blaise Diagne><TAP PORTUGAL>
[815]<EK 1921><Dubai><EMIRATES>
[740]<TP 1324><Dublin><TAP PORTUGAL>
[970]<LH 1169><Frankfurt><LUFTHANSA>
[515]<DS 1446><Geneva><EASYJET SWITZERLAND>
[650]<TP 9465><Geneva><TAP PORTUGAL>
[680]<TK 17561><Istanbul><TURKISH AIRLINES>
[585]<TP 472><Lyon, St. Exupery><TAP PORTUGAL>
[600]<TP 16992><Madeira><TAP PORTUGAL>
[1230]<TP 16935><Madeira><TAP PORTUGAL>
[505]<IB 8765><Madrid><IBERIA>
[595]<TP 10145><Madrid><TAP PORTUGAL>
[685]<UX 1150><Madrid><AIR EUROPA>
[875]<TP 10164><Madrid><TAP PORTUGAL>
[1005]<IB 8775><Madrid><IBERIA>
[870]<TP 11383><Malaga><TAP PORTUGAL>
```

Constatamos, com esta listagem parcial, que o array está ordenado pelo nome do aeroporto de destino, conforme queríamos. Também constatamos que os casos de empate, isto é, os troços em que o aeroporto de destino é o mesmo, estão ordenado por ordem da hora de partida. Isso acontece porque essa era a ordenação inicial, no que respeita a esses elementos empatados. O *insertionsort*, ao fazer o seu trabalho, nunca troca elementos empatados e isso garante que a ordenação inicial desses elementos não é comprometida. Ainda assim, é importante

perceber desde já que outros algoritmos de ordenação não são tão “cuidadosos” como o *insertionsort*.

Portanto, se a nossa intenção tivesse sido desempatar por ordem da hora de partida, bastava mesmo fazer como fizemos, e mais nada.

Suponhamos, para clarificar a questão, que queremos desempatar, não por hora de partida, mas por número de voo. O número de voo é uma cadeia de caracteres e estamos a pensar na ordem lexicográfica.

Deve ser evidente que a comparação usada na função `flights_sort_by_destination`, que é `strcmp(a[j-1].destination, a[j].destination) > 0`, não vai bastar. Faz falta é uma função de comparação *ad hoc* que compare primeiro pelo destino e depois, em caso de empate, pelo número de voo. Observe, com muita atenção:

```
int flight_cmp_by_destination_then_code(Flight x, Flight y)
{
    int result = strcmp(x.destination, y.destination);
    if (result == 0)
        result = strcmp(x.code, y.code);
    return result;
}
```

Este é o esquema geral das funções de comparação *multicritério*: primeiro compara-se pelo critério principal. Se houver empate, essa comparação dá zero e então compara-se pelo critério secundário.

Percebido isto, a função de ordenação não tem mistério:

```
void flights_sort_by_destination_then_code(Flight *a, int n)
{
    for (int i = 1; i < n; i++)
    {
        int j = i;
        while (j > 0 &&
0)          flight_cmp_by_destination_then_code(a[j-1], a[j]) >
        {
            flights_exchange(a, j-1, j);
            j--;
        }
    }
}
```

```
}  
}
```

A função de teste é como a anterior, substituindo a ordenação e omitindo a escrita do array antes da ordenação:

```
void test_sort_by_destination_then_code(void)  
{  
    Flight a[max_flights];  
    int n = flights_read(stdin, a);  
    flights_sort_by_destination_then_code(a, n);  
    flights_print(stdout, a, n);  
}
```

Correndo como anteriormente, obtemos um ficheiro cujas primeiras linhas são assim:

```
300]<KL 16925><Amsterdam><KLM>  
[925]<KL 16965><Amsterdam><KLM>  
[700]<TP 6746><Amsterdam><TAP PORTUGAL>  
[720]<TP 10385><Barcelona><TAP PORTUGAL>  
[1065]<TP 14791><Bissau><TAP PORTUGAL>  
[1180]<TP 217><Boston><TAP PORTUGAL>  
[595]<TP 6465><Brussels><TAP PORTUGAL>  
[1360]<TP 14851><Dakar, Blaise Diagne><TAP PORTUGAL>  
[815]<EK 1921><Dubai><EMIRATES>  
[740]<TP 1324><Dublin><TAP PORTUGAL>  
[970]<LH 1169><Frankfurt><LUFTHANSA>  
[515]<DS 1446><Geneva><EASYJET SWITZERLAND>  
[650]<TP 9465><Geneva><TAP PORTUGAL>  
[680]<TK 17561><Istanbul><TURKISH AIRLINES>  
[585]<TP 472><Lyon, St. Exupery><TAP PORTUGAL>  
[1230]<TP 16935><Madeira><TAP PORTUGAL>  
[600]<TP 16992><Madeira><TAP PORTUGAL>  
[505]<IB 8765><Madrid><IBERIA>  
[1005]<IB 8775><Madrid><IBERIA>  
[595]<TP 10145><Madrid><TAP PORTUGAL>  
[875]<TP 10164><Madrid><TAP PORTUGAL>  
[685]<UX 1150><Madrid><AIR EUROPA>  
[870]<TP 11383><Malaga><TAP PORTUGAL>
```

As diferenças são subtis mas importantes.