Lập Trình C (cơ bản)

Chương 07. Sắp xếp & đo thời gian

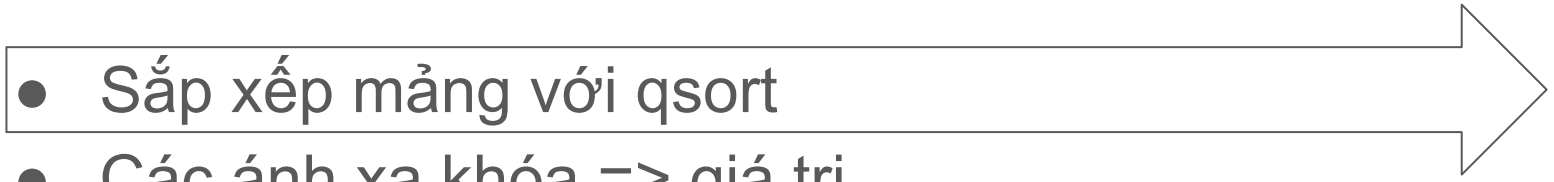
Soạn bởi: TS. Nguyễn Bá Ngọc

2021

Nội dung

- Sắp xếp mảng với qsort
- Các ánh xạ khóa => giá trị
- Đo thời gian xử lý

Nội dung



- Sắp xếp mảng với qsort
- Các ánh xạ khóa => giá trị
- Đo thời gian xử lý

Hàm qsort

Thư viện: stdlib.h

extern void

qsort (void *base, size_t nmemb, size_t size, compar_t cmp);

Tham số: **base** - Con trỏ tới phần tử đầu tiên của mảng

nmemb - Số lượng phần tử

size - Kích thước phần tử (sizeof)

cmp - Hàm so sánh, sử dụng định dạng trả về như strcmp cho kết quả tăng dần.

Nguyên mẫu: **int (*) (const void*, const void*)**;

Ví dụ 7.1. Sắp xếp mảng int với qsort

vd7-1.c

```
11 #define to_int(p) (*((const int*)p))
12 int cmp_inc_i(const void *a, const void *b) {
13   return to_int(a) - to_int(b);
14 }
15
16 int main() {
17   int a[] = {3, 2, 1, 6, 5, 8};
18   int n = sizeof(a)/sizeof(a[0]);
19   printf("Trước sắp xếp: ");
20   print_a(a, n);
21   qsort(a, n, sizeof(a[0]), cmp_inc_i);
22   printf("Sau Sắp xếp: ");
23   print_a(a, n);
24   return 0;
25 }
```

qsort truyền cho cmp_inc_i 2 con trỏ tới các vùng nhớ chứa giá trị kiểu int/các phần tử của mảng a.

```
bangoc:$gcc -o prog vd7-1.c
bangoc:$./prog
Trước sắp xếp:  3 2 1 6 5 8
Sau Sắp xếp:  1 2 3 5 6 8
bangoc:$
```

Bài tập 7.1. Sắp xếp mảng số thực với qsort

Viết chương trình hỏi người dùng nhập vào 1 số nguyên dương n, sau đó hỏi người dùng nhập vào n số thực.

Yêu cầu: 1) Lưu các số thực trong mảng được phát động.

2) Sắp xếp các giá trị được nhập vào với qsort và sau đó đưa kết quả ra màn hình.

3) Giải phóng bộ nhớ đã được cấp phát.

Bài tập 7.2. Sắp xếp mảng chuỗi ký tự

Viết chương trình hỏi người dùng nhập số nguyên dương n, sau đó hỏi người dùng nhập vào n chuỗi ký tự.

Yêu cầu: 1) Lưu các chuỗi ký tự do người dùng nhập vào trong mảng cấp phát động.

2) Sắp xếp các chuỗi ký tự theo thứ tự bảng chữ cái và in kết quả ra màn hình.

3) Giải phóng bộ nhớ đã được cấp phát.

Gợi ý: char **a = malloc(n * sizeof(char*)); // char *a[n];
a[i] = strdup(s);
!qsort sẽ truyền địa chỉ phần tử của mảng **a**, trong trường hợp này là địa chỉ vùng nhớ chứa giá trị kiểu **char***.

Bài tập 7.3. Xử lý dữ liệu văn bản

Cho bộ dữ liệu văn bản trong tệp có ở địa chỉ:
https://github.com/bangoc/c-basic/blob/master/text/lisa_all_text.txt
Trong bài tập này chúng ta định nghĩa từ là chuỗi ký tự liên tiếp không chứa khoảng trắng.

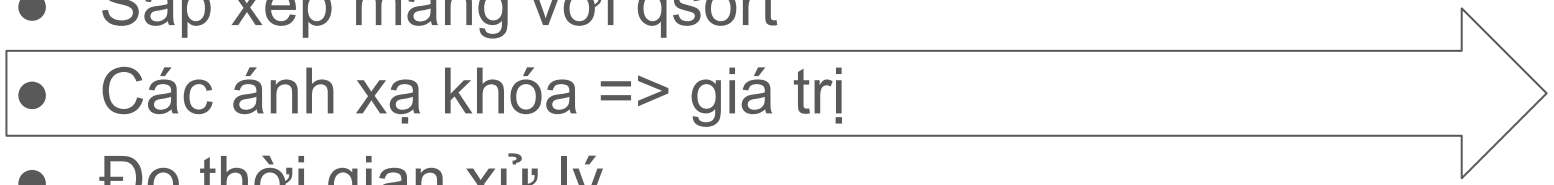
Yêu cầu: Đưa ra số lượng từ duy nhất và 10 từ xuất hiện thường xuyên nhất.

Gợi ý: Có thể đọc từng dòng với cgetline rồi tách từ theo khoảng trắng hoặc đọc từng từ bằng fscanf với khóa %s

Phương án 1: B1) Lưu các từ đọc được vào danh sách móc nối đơn; B2) Tạo mảng động để lưu các từ (tương tự 7.2); B3) Sắp xếp mảng rồi thực hiện thống kê; B4) Tạo mảng cấu trúc chứa các từ và số lần xuất hiện rồi sắp xếp theo số lần xuất hiện. B5) Xuất ra các dữ liệu được yêu cầu.

Nội dung

- Sắp xếp mảng với qsort
- Các ánh xạ khóa => giá trị
- Đo thời gian xử lý



Cấu trúc lưu trữ dạng ánh xạ khóa=>giá trị

- Có thể coi mảng là 1 ánh xạ đơn giản nhất với khóa là số nguyên (chỉ số) và giá trị là phần tử mảng
  - Ví dụ: int a[100]; // a[i] là 1 phần tử có kiểu int
- Ưu điểm và nhược điểm của mảng:
  - Ưu điểm: Tốc độ truy cập nhanh (O(1)).
  - Nhược điểm: Các chỉ số phải là các số nguyên liên tiếp
    - Không hiệu quả nếu giá trị khóa có phân bố thưa.
    - Không sử dụng được dữ liệu khác làm khóa.
- Ánh xạ dựa trên hàm băm: Có thể sử dụng nhiều kiểu dữ liệu khác nhau làm khóa.
  - Các khóa thường được chuyển thành số nguyên bằng hàm băm và **không được sắp xếp** thứ tự. Trong điều kiện lý tưởng có thể truy cập các phần tử với độ phức tạp O(1).
- Ánh xạ dựa trên cây nhị phân tìm kiếm: Có thể sử dụng nhiều kiểu dữ liệu khác nhau làm khóa.
  - Các khóa được **sắp xếp** theo thứ tự trong cây. Với các triển khai dựa trên cây cân bằng có thể truy cập các phần tử với độ phức tạp $O(\log_2(N))$.

(Tham khảo triển khai **rbm** *dựa trên cây đỏ-đen có trong* **cgen***)*

Một số hàm cơ bản trong triển khai rbm

rbm là một triển khai ánh xạ dựa trên cây đỏ-đen với khóa và giá trị có kiểu gtype.

- rbm_t rbm_create(
  bn_compare_t cmp // Hàm so sánh khóa kiểu gtype
  );
  Tạo đối tượng rbm và trả về (con trỏ đến) đối tượng được tạo nếu thành công, hoặc trả về NULL nếu phát sinh lỗi.
- rbm_node_t rbm_insert(rbm_t t, // Đối tượng rbm
  gtype key, gtype value // Khóa và giá trị được thêm vào
  );
  Trả về nút mới được tạo nếu khóa chưa tồn tại, hoặc trả về nút đã có trong t có khóa bằng khóa đang được thêm vào.
- gtype *rbm_vref(rbm_t t, // Đối tượng rbm
  gtype key // Khóa
  );
  Trả về con trỏ đến value được gắn với key trong t nếu tìm thấy, hoặc NULL nếu ngược lại.

Một số hàm cơ bản trong triển khai rbm(2)

- rbm_node_t rbm_search(rbm_t t, // Đối tượng rbm
  gtype key // Khóa cần tìm
  );
  Trả về nút có khóa == key trong t nếu tìm thấy, hoặc trả về NULL nếu ngược lại.
- rbm_node_t rbm_delete(rbm_t t, // Đối tượng rbm
  gtype key // Khóa của nút cần xóa
  );
  Nếu tìm được nút có khóa == key trong t thì tách nút đó ra khỏi t (xóa các liên kết nhưng không giải phóng bộ nhớ) và trả về nút tìm được, nếu ngược lại thì trả về NULL.

Một số macro hữu ích:

- rbm_traverse(cur, m) - Duyệt các nút trong đối tượng rbm;
- rbm_free(m) - Giải phóng bộ nhớ của đối tượng rbm. Người dùng cần giải phóng bộ nhớ động có trong key và value của các nút nếu có, trước khi gọi rbm_free;
- rbm_node_key(n) - key của nút n;
- rbm_node_value(n) - value của nút n.

Ví dụ 7.2. Ánh xạ dựa trên cây đỏ-đen

vd7-2.c

```c
#include "cgen.h"
#include "ext/io.h"
#include <stdio.h>
#include <string.h>
int main() {
  char buff[1024];
  printf("Nhập chuỗi ký tự (hoặc chỉ bấm Enter): \n");
  rbm_t words = rbm_create(gtype_cmp_s);
  do {
    remove_tail_lf(fgets(buff, 1024, stdin));
    if (buff[0] == '\0') {
      break;
    }
    char *s = strdup(buff);
    rbm_node_t n = rbm_insert(words, gtype_s(s), gtype_i(-1));
    if (rbm_node_value(n).i < 0) {
      rbm_node_value(n).i = 1;
    } else {
      rbm_node_value(n).i++;
      free(s);
    }
  } while (1);
  printf("Các từ đã nhập và số lần nhập: \n");
  rbm_traverse(cur, words) {
    printf("%s: %ld\n",
      rbm_node_key(cur).s, rbm_node_value(cur).i);
  }
  rbm_traverse(cur, words) {
    free(rbm_node_key(cur).s);
  }
  rbm_free(words);
  return 0;
}
```

```
Nhập chuỗi ký tự (hoặc chỉ bấm Enter):
aa
bb
aa
cc
bb
dd
ee

Các từ đã nhập và số lần nhập:
aa: 2
bb: 2
cc: 1
dd: 1
ee: 1
```

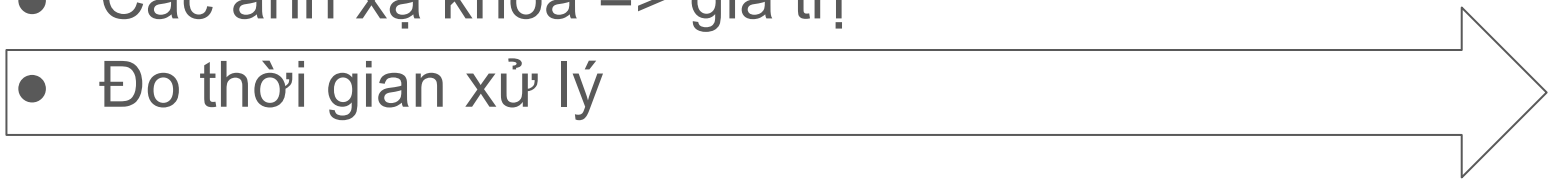
Bài tập 7.4. Xử lý dữ liệu văn bản

Sử dụng các yêu cầu và dữ liệu đầu vào như trong bài tập 7.3, nhưng giải quyết vấn đề theo cách khác (phương án 2).

Phương án 2: B1) Đọc từng từ và lưu vào 1 đối tượng rbm với khóa là từ và giá trị là số lần từ xuất hiện; B2) Duyệt đối tượng rbm thu được ở bước 1 để tạo mảng các các cấu trúc bao gồm từ duy nhất cùng với số lần xuất hiện của nó; B3) Sắp xếp mảng thu được ở B2 theo số lần xuất hiện rồi xuất các thông tin được yêu cầu.

Nội dung

- Sắp xếp mảng với qsort
- Các ánh xạ khóa => giá trị
- Đo thời gian xử lý



Đo thời gian xử lý trong C

Hàm clock, thư viện time.h

```
extern clock_t clock (void);
```

Trả về thời gian sử dụng bộ vi xử lý từ thời điểm bắt đầu thực hiện chương trình. Để lấy giá trị thời gian tính bằng *s* chúng ta cần chia cho CLOCKS_PER_SEC.

- Chúng ta đo thời gian ở thời điểm bắt đầu và thời điểm kết thúc xử lý sau đó lấy hiệu của 2 kết quả đo.
  - Ví dụ:

```
clock_t t1 = clock();  // có kiểu số nguyên
/* Xử lý */
clock_t t2 = clock();
// Thời gian xử lý = (double)(t2 - t1)/CLOCKS_PER_SEC
```

Để có kết quả ổn định chúng ta có thể đo nhiều lần và lấy trung bình kết quả thu được từ các lần đo.

Ví dụ 7.3. So sánh thời gian sắp xếp

vd7-3.c

```
11  void selsort(int *a, const int n) {
12    for (int i = 0; i < n - 1; ++i) {
13      for (int j = i + 1; j < n; ++j) {
14        if (a[i] > a[j]) {
15          int tmp = a[i];
16          a[i] = a[j];
17          a[j] = tmp;
18        }
19      }
20    }
21  }
22  int cmp_inc_i(const void *v1, const void *v2) {
23    return *((const int*)v1) - *((const int*)v2);
24  }
25  int main(int argc, char *argv[]) {
26    if (argc != 2) {
27      printf("Usage: ./prog 100000\n");
28      return 1;
29    }
30    int n;
31    sscanf(argv[1], "%d", &n);
32    if (n < 0) {
33      printf("Số lượng phần tử = %d\n", n);
34      return 1;
35    }
36    srand(time(NULL));
37    int *a = malloc(n * sizeof(int)),
38        *b = malloc(n * sizeof(int));
39    for (int i = 0; i < n; ++i) {
40      a[i] = b[i] = rand();
41    }
42    BENCH("Sắp xếp chọn", 1, selsort(a, n));
43    BENCH("Sắp xếp nhanh", 1,
44      qsort(a, n, sizeof(a[0]), cmp_inc_i));
45    return 0;
46  }
```

```
bangoc:$./prog 1000
Sắp xếp chọn - Trung bình 1 lần = 0.004958 s
Sắp xếp nhanh - Trung bình 1 lần = 2.9e-05 s
bangoc:$./prog 10000
Sắp xếp chọn - Trung bình 1 lần = 0.1979 s
Sắp xếp nhanh - Trung bình 1 lần = 0.000235 s
bangoc:$./prog 100000
Sắp xếp chọn - Trung bình 1 lần = 23.263 s
Sắp xếp nhanh - Trung bình 1 lần = 0.002728 s
bangoc:$
```

Qua ví dụ này có thể thấy tốc độ vượt trội của qsort *so với* selsort *(sắp xếp chọn). Cũng có thể thấy thời gian sắp xếp bằng* qsort *tăng chậm hơn nhiều so với thời gian sắp xếp bằng* selsort.

Bài tập 7.5. Đo thời gian xử lý

Tái cấu trúc kết quả thu được ở Bài tập 7.3 và Bài tập 7.4: Đưa các xử lý vào hàm và chuyển sang 1 đơn vị biên dịch riêng không chứa hàm main, và được ghép nối vào phần còn lại (có chứa hàm main).

Gợi ý: Viết chương trình đo thời gian xử lý với các hàm thu được sau khi tái cấu trúc.