# CS360 Bài giảng - Red-Black Trees (JRB)

- James S. Plank
- Thư mục: ~ **ván / tấm ván / classes / cs360 / 360 / www-home / ghi chú / JRB**
- Bài giảng: **http://www.cs.utk.edu/~plank/plank/classes/cs360/360/notes/JRB**
- Ghi chú bằng văn bản ban đầu Tháng Tám, 1999.
- Phiên bản mới nhất: tháng một, năm 2015.

---

---

### Biên dịch

Để sử dụng các thư viện cây đỏ-đen, bạn nên bao gồm các tập tin "jrb.h", có thể được tìm thấy trong **/ home / ván / cs360 / include** . Thay vì bao gồm tên đường dẫn đầy đủ trong tập tin C của bạn, chỉ cần làm:

```
#include "jrb.h",
```

và sau đó biên dịch chương trình với:

```
gcc -I / home / ván / cs360 / bao gồm
```

Khi bạn liên kết các tập tin đối tượng của bạn để tạo một thực thi, thực hiện theo các hướng dẫn trong các ghi chú bài giảng libfdr .

Makefile trong thư mục này không có gì cả những điều này cho bạn.

## Cây Red-Black

Rb-cây là những cấu trúc dữ liệu dựa trên cây nhị phân cân đối. Bạn không cần phải biết làm thế nào họ làm việc - chỉ là họ làm việc, và tất cả các hoạt động đều trong *O (log (n))* thời gian, nơi *n* là số phần tử trong cây. (Nếu bạn thực sự muốn biết thêm về cây đỏ-đen, cho tôi biết và tôi có thể chỉ cho bạn một số văn bản về họ).

Các cấu trúc chính cho rb-cây là **JRB** . Giống như dllists, tất cả rb-cây có một nút tiêu đề. Bạn tạo một rb-cây bằng cách gọi **make_jrb ()** , trả về một con trỏ đến nút tiêu đề của một sản phẩm nào rb-tree.Tiêu đề này điểm để các cơ quan chính của rb-tree, mà bạn không cần phải quan tâm, và để các nút bên ngoài đầu tiên và cuối cùng của cây. Những nút bên ngoài được nối với nhau bằng flink và nháy con trỏ, để bạn có thể xem rb-cây như là dlists với tài sản mà họ được sắp xếp, và bạn có thể tìm thấy bất kỳ nút trong cây trong *O (log (n))* thời gian.

Giống như dllists, mỗi nút trong cây có một **val** trường, là một Jval. Ngoài ra, mỗi node có một **chính** trường, mà còn là một Jval. Cây rb-cây làm cho chắc chắn rằng các phím được sắp xếp. Làm thế nào chúng được sắp xếp phụ thuộc vào cây.

---

## _str, _int, _dbl, _gen

Các thói quen cây JRB trong **jrb.h / jrb.c** thực hiện bốn loại chèn / tìm kiếm các thói quen. Các thói quen chèn là:

- **JRB jrb_insert_str (JRB cây, char * key, Jval val)** : chèn một nút mới vào cây sử dụng một chuỗi ký tự chuẩn như là chìa khóa. **strcmp ()** được sử dụng như các chức năng so sánh. Xem **strsort.c**cho một ví dụ đơn giản về phân loại tiêu chuẩn đầu vào thứ tự từ điển với **jrb_insert_str ()** .

  Lưu ý rằng nó sẽ trả về một con trỏ đến nút cây JRB mới. Cũng lưu ý rằng, nếu chính là đã có trong cây, sau đó nó vẫn tạo ra một nút mới và đặt nó vào trong cây. Không có bảo đảm được thực hiện liên quan đến trật tự tương đối của các phím trùng lặp.

  Mặc dù chính là một chuỗi, nó sẽ được chuyển đổi thành một **Jval** trong nút mới. Vì vậy, nếu bạn muốn có được tại các trọng điểm của nút **s** , bạn nên sử dụng một trong hai **jval_s (s-> key)** hay **s-> key.s** .

- **JRB jrb_insert_int (JRB cây, int key, Jval val)** : chèn vào cây sử dụng một số nguyên như là chìa khóa. Xem **nsort.c** cho một ví dụ về điều này.
- **JRB jrb_insert_dbl (JRB cây, chìa khóa đôi, Jval val)** : chèn vào cây sử dụng một **đôi** như là chìa khóa.
- **JRB jrb_insert_gen (JRB cây, phím Jval, Jval val, int (* func) (Jval, Jval))** : Bây giờ, chính bạn là một **jval** . Bạn cung cấp một chức năng so sánh **func ()** , mà phải mất hai **Jval** s 'như các đối số, và trả về:
  - một số nguyên âm nếu phím đầu tiên là ít hơn lần thứ hai.
  - một số nguyên dương nếu chìa khóa đầu tiên lớn hơn lần thứ hai.
  - bằng không nếu các phím đều bình đẳng.

  Điều này cho phép bạn làm những điều phức tạp hơn so với chỉ đơn giản là phân loại với số nguyên và chuỗi. Ví dụ, **strisort.c** sắp xếp dây nhưng bỏ qua trường hợp. **strrsort2.c** sắp xếp chuỗi theo thứ tự ngược. Đọc những hơn.

Bạn không thể trộn và kết hợp các chức năng so sánh trong cùng một cây. Nói cách khác, bạn không cần phải chèn thêm một số phím với **jrb_insert_str ()** và một số với **jrb_insert_int ()** . Để làm như vậy sẽ van xin một bãi chứa lõi.

Để tìm thấy chìa khóa, bạn sử dụng một trong **jrb_find_str ()** , **jrb_find_int ()** , **jrb_find_dbl ()** hoặc **jrb_find_gen ()** . Rõ ràng, nếu bạn chèn phím với **jrb_insert_str ()** , sau đó bạn nên sử dụng**jrb_find_str ()** để tìm thấy chúng. Nếu chìa khóa mà bạn đang tìm đã không có trong cây, sau đó **jrb_find_** ***xxx*** **()** trả về NULL.

Cuối cùng, cũng có: **jrb_find_gte_str ()** , **jrb_find_gte_int ()** , **jrb_find_gte_dbl ()** và **jrb_find_gte_gen ()** . Những trả lại nút cây JRB mà chính là một trong hai bằng phím nhất định, hoặc có trọng là nhỏ nhất lớn hơn các khóa này. Nếu khóa này là lớn hơn bất kỳ trong cây, nó sẽ trả về một con trỏ đến nút trọng điểm. Nó đã một đối số **phát** được thiết lập để cho bạn biết nếu phím đã được tìm thấy hay không.

Bạn có thể sử dụng các macro **jrb_first ()** , **jrb_last ()** , **jrb_prev ()** và **jrb_next ()** , giống như đối tác của họ trong thư viện dllist.

Ngoài ra, để xóa một cây, bạn sử dụng **jrb_free_tree () .** Điều này không gọi **miễn phí ()** trên phím và Vals - nó chỉ đơn giản là xóa tất cả bộ nhớ kết hợp với các nút trên cây.

---

## Chương trình ví dụ:

- **strsort.c** : Sử dụng cây đỏ-đen để sắp xếp thứ tự từ điển đầu vào tiêu chuẩn.
- **strrsort1.c** : Sử dụng cây đỏ-đen để sắp xếp tiêu chuẩn đầu vào thứ tự từ điển theo thứ tự ngược. Nó làm điều này bằng cách đi qua cây theo thứ tự ngược.
- **strrsort2.c** : Sử dụng cây đỏ-đen để sắp xếp tiêu chuẩn đầu vào thứ tự từ điển theo thứ tự ngược. Nó làm điều này bằng cách tạo ra một chức năng mới so **revcomp** , mà chỉ đơn giản trả **-strcmp ()** .Bây giờ các loại cây trái ngược nhau, vì vậy nó được đi qua hướng về phía trước.
- **strusort.c** : Sử dụng cây đỏ-đen để sắp xếp thứ tự từ điển đầu vào tiêu chuẩn, và nó loại bỏ các dòng trùng lặp. Nó làm điều này bằng cách kiểm tra với một dòng trước khi chèn nó vào trong cây.
- **strisort.c** : Sử dụng cây đỏ-đen để sắp xếp thứ tự từ điển đầu vào tiêu chuẩn, bỏ qua trường hợp trên và dưới. Nó làm điều này bằng cách tạo ra một chức năng mới so **ucomp** , có thể sao **strcmp ()**chức năng 's nhưng bỏ qua trường hợp.
- **nsort.c** : Sử dụng cây đỏ-đen để sắp xếp như **sort -n** - tức là nó đối xử với mỗi dòng là một số nguyên, và sắp xếp nó theo cách đó. Nếu các dòng là số nguyên không, hoặc có những dòng trùng lặp, bất cứ điều gì đi.
- **nsort2.c** : Sử dụng cây đỏ-đen để sắp xếp như **sort -n** chỉ bây giờ nếu hai dòng có cùng **atoi ()** giá trị, sau đó chúng được sắp xếp thứ tự từ điển. Này sử dụng **jrb_insert_gen () .**
- **nsort3.c** : Tương tự như **nsort2** , nhưng thay vào đó nó sử dụng một hai cấp cây đỏ-đen. Nếu đây là khó hiểu đối với bạn, xin vui lòng đọc các ví dụ tiếp theo, mà không một điều rất giống nhau.

---

## Một ví dụ cây hai cấp

Giả sử chúng ta đang đọc đầu vào bao gồm tên và điểm số. Tên có thể là bất kỳ số từ, và điểm số là các số nguyên. Mỗi dòng có một tên tiếp theo một số điểm, và có thể là bất kỳ số lượng khoảng trắng giữa các từ trong các tập tin đầu vào. Một ví dụ là **đầu vào-nn.txt** . Như bạn có thể nhìn thấy từ 10 dòng đầu tiên, đó là loại lộn xộn, nhưng nó phù hợp với các định dạng:

```
UNIX> đầu -n 10 đầu vào-nn.txt
     Molly lên trời 60
Taylor mây che 47
   Brody trễ 56
Tristan Covenant 75
```

```
Adam nhuộm 38
Brianna miền 54
     Jonathan Value 5
        Max Head 48
Adam Bobbie 68
        Jack không thể miêu tả 99
UNIX>
```

Giả sử chúng ta muốn xử lý tập tin đầu vào này bằng cách tạo ra một **Person** cấu trúc cho mỗi dòng có tên của người đó và số điểm:

```
typedef struct {
  char * name;
  int điểm;
} Người;
```

Và sau đó giả sử chúng ta muốn in các người, trước tiên là theo điểm số, và sau đó bằng tên. Chúng tôi muốn các định dạng đầu ra của chúng tôi để được tên, trái lý và độn đến 40 ký tự, theo sau là số các đệm để hai nhân vật. Tôi sẽ viết chương trình này ba lần. Tôi tin rằng cuối cùng của ba là tốt nhất, nhưng nó là một bài tập tốt để đi qua tất cả ba.

## ni_sort1.c - tạo ra một chìa khóa phân loại

Chương trình đầu tiên là *ni_sort1.c* . Nó đọc mỗi người vào một cấu trúc và sau đó tạo ra một chuỗi mà nó sử dụng như là một chuỗi so sánh. Key có chứa số điểm, phải chứng minh và độn đến mười nhân vật, và sau đó tên. Vì vậy, khi bạn sử dụng phím để chèn người vào một cây đỏ-đen, cây được sắp xếp theo thứ tự mà bạn muốn.

Hãy nhìn vào các chương trình:

```
typedef struct {
  char * name;
  int điểm;
  char * trọng điểm;
} Người;

main ()
{
  JRB t, tmp;
  IS;
  Person * p;
  int nDung lượng, i;

  là = new_inputstruct (NULL);
  t = make_jrb ();

  while (get_line (là)> = 0) {
    if (là-> NF> 1) {

      / * Mỗi dòng là tên tiếp theo chỉ số. Tỉ số là dễ dàng để có được. * /

      p = malloc (sizeof (Person));
```

```
      p-> score = atoi (là-> Trường [là-> NF-1]);

      / * Tên là một vấn đề khác nhau, bởi vì tên này có thể được bao gồm bất
kỳ
         số từ với bất kỳ số lượng khoảng trắng. Chúng tôi muốn tạo ra một
         tên chuỗi mà có mỗi từ của tên cách nhau bởi một dấu cách.

         Nhiệm vụ đầu tiên của chúng tôi là để tính toán kích thước của tên
của chúng tôi. * /

      nDung lượng = strlen (là-> Trường [0]);
      for (i = 1; i <là-> NF-1; i ++) nDung lượng + = (strlen (là-> Trường
[i]) + 1);

      / * Chúng tôi sau đó phân bổ các chuỗi và sao chép từ đầu tiên vào
chuỗi. * /

      p-> name = (char *) malloc (sizeof (char) * (nDung lượng + 1));
      strcpy (p-> tên, là-> Trường [0]);

      / * Chúng tôi sao chép trong các từ còn lại, nhưng lưu ý cách chúng tôi
làm như vậy bằng cách gọi strcpy
         vào vị trí chính xác của nơi tên đi, chứ không phải là, nói, nhiều
lần
         gọi strcat () như chúng ta sẽ làm gì trong C ++ - như giải pháp. Đây
là nhiều hơn nữa
         hiệu quả (không đề cập đến bất tiện) vì sử dụng strcat (). * /

      nDung lượng = strlen (là-> Trường [0]);
      for (i = 1; i <là-> NF-1; i ++) {
        p-> tên [nDung lượng] = '';
        strcpy (p-> tên + nDung lượng + 1, là-> Trường [i]);
        nDung lượng + = strlen (p-> tên + nDung lượng);
      }

      / * Chúng tôi tạo ra một chìa khóa để chèn vào cây đỏ-đen. Điều đó sẽ
         để có số điểm, độn đến 10 ký tự, theo sau bởi tên. Chúng tôi
         phân bổ (nDung lượng + 12) ký tự: nDung lượng cho tên, 10 cho điểm
số,
         một cho không gian, và một cho các ký tự null. * /

      p-> key = (char *) malloc (sizeof (char) * (nDung lượng + 12));
      sprintf (p-> key, "% 10ngày% s", p-> điểm, p-> tên);

      jrb_insert_str (t, p> key, new_jval_v ((void *) p));
    }
  }

  jrb_traverse (tmp, t) {
    p = (Person *) tmp-> val.v;
    printf ("% - 40% 2d \ n", p-> tên, p-> điểm);
  }
  exit (0);
}
```

Trong một cơn hiếm của niceness, tôi có nhận xét chương trình này. Bạn nên chú ý đến cách tôi tạo ra tên, vì nó như thế nào bạn làm một điều như vậy có hiệu quả trong C. Bạn sẽ bị cám dỗ để chỉ đơn giản là phân bổ một chuỗi khổng lồ và sau đó sử dụng **strcat ()** để tạo ra tên. Đó là mô hình mà bạn muốn sử dụng trong C ++. Tuy nhiên, đó là không hiệu quả vì **strcat ()** (xem các bài bình luận trên **strcat ()** trongcác bài giảng ).

Thay vào đó, bạn thực hiện một đường chuyền qua tên để tính toán kích thước của chuỗi, và sau đó bạn phân bổ các chuỗi. Sau đó, bạn sử dụng **strcpy ()** để sao chép mỗi từ của tên vào đúng chỗ của nó. Có, mã là xấu xí, nhưng nó là cách hiệu quả nhất để làm điều đó.

Sau khi tạo tên, chúng tôi tạo ra các phím so sánh, và lưu ý như thế nào, chúng ta phải tính toán kích thước của nó và phân bổ nó. Chúng tôi chèn chìa khóa vào cây với các cấu trúc như một người val, và khi chúng tôi đi qua nó, chúng tôi nhận được đơn đặt hàng mà chúng tôi muốn. Tôi sử dụng "% 10D" cho điểm số, bởi vì tôi biết các số nguyên maxiumum là $2^{31}$ , đó là khoảng 2000000000. Tôi muốn tất cả các điểm liên kết và quyền chính đáng trong các phím, vì vậy nó sẽ sắp xếp các số nguyên đúng. Điều này là bởi vì không gian có giá trị ASCII thấp hơn so với con số.

Nó luôn luôn tốt để sanity kiểm tra các chương trình của bạn để đảm bảo rằng sản lượng có ý nghĩa, và rằng bạn không có lỗi hoặc lỗi chính tả. Dưới đây, tôi làm như sau. Trước tiên, tôi đảm bảo rằng các tập tin đầu vào và đầu ra có cùng một số đường dây và lời:

```
UNIX> ni_sort1 <input-nn.txt> đầu ra-1.txt
UNIX> wc đầu vào-đầu ra nn.txt-1.txt
  500 1583 24446 đầu vào-nn.txt
  500 1583 22000 đầu ra-1.txt
 1000 3166 46.446 tổng số
UNIX>
```

Chúng khác nhau về số lượng các ký tự, bởi vì họ định dạng chữ và điểm số khác nhau. Tất cả là tốt cho đến nay. Tiếp theo, tôi tỉnh táo kiểm tra bắt đầu và kết thúc để đảm bảo rằng họ nhìn bên phải:

```
UNIX> đầu ra-1.txt
Addison Paige Chain 0
Eli gneis 0
Ella craftsperson 0
Lilly Gianna Zen 0
Matthew cứng 0
Evan thô lỗ 1
Isaiah Metabolism 1
Mason Fourier 1
Xavier Agave 1
Daniel Berman 2
UNIX> đuôi ra-1.txt
Layla Lựa chọn 96
Lucas Fay Jr 96
Madeline công tác 96
Sofia Nitrous 96
Gianna Sinh 97
Lucy Quaternary 97
Sophia ngược dòng 97
Charlie Lucas Vine 98
```

```
Jack không thể miêu tả 99
Lily Span 99
UNIX>
```
Sau đó, tôi làm mẫu, để đảm bảo rằng một trong các giá trị số có đầu ra bên phải. Ở đây, tôi làm điều đó với 96:
```
UNIX> grep 96 đầu ra-1.txt
Alexander Bstj 96
Grace Globulin 96
Jonathan Blanket Esq 96
Kaitlyn sự chia nhau 96
Layla Lựa chọn 96
Lucas Fay Jr 96
Madeline công tác 96
Sofia Nitrous 96
UNIX> grep 96 đầu vào-nn.txt
   Grace Globulin 96
        Jonathan Blanket Esq 96
Alexander Bstj 96
     Madeline công tác 96
     Layla Lựa chọn 96
     Kaitlyn sự chia nhau 96
     Sofia Nitrous 96
        Lucas Fay Jr 96
UNIX> grep 96 đầu vào-nn.txt | sed 's / ^ * //' | sort
Alexander Bstj 96
Grace Globulin 96
Jonathan Blanket Esq 96
Kaitlyn sự chia nhau 96
Layla Lựa chọn 96
Lucas Fay Jr 96
Madeline công tác 96
Sofia Nitrous 96
UNIX>
```
Ok, tôi là tốt. Lưu ý, đó không phải là một bài kiểm tra kết luận. Dưới đây là một bài kiểm tra kết luận hơn (chỉ nhìn vào điều này nếu bạn đang thực sự quan tâm. Tôi làm công cụ này tất cả các thời gian, vì vậy tôi rất tốt ở đó. Tuy nhiên, tôi sẽ buộc bạn phải tìm hiểu sed và awk, bởi vì họ là những siêu . mạnh mẽ) (Oh, và tôi không kiểm tra bạn về công cụ này, tôi chỉ cố gắng để giúp bạn trở nên hiệu quả hơn với các công cụ Unix):
```
UNIX> sed 's / ^ * //' input-nn.txt | head
Molly lên trời 60
Taylor mây che 47
Brody trễ 56
Tristan Covenant 75
Adam nhuộm 38
Brianna miền 54
Jonathan Value 5
Max Head 48
Adam Bobbie 68
Jack không thể miêu tả 99
UNIX> sed 's / ^ * //' input-nn.txt | sed 's / * / - / g' | head
Molly-trời-60-
Taylor-mây che-47
Brody-trễ-56
Tristan-Covenant-75
Adam-Nhuộm-38
```

```
Brianna-Domain-54
Jonathan-Value-5
Max-Head-48
Adam-Bobbie-68
Jack-khôn tả-99
UNIX> sed 's / ^ * //' input-nn.txt | sed 's / * / - / g' | sed 's / - \ ([0-
9] \) / \ 1 /' | head
Molly-trời 60-
Taylor-mây che 47
Brody-trễ 56
Tristan-Covenant 75
Adam-nhuộm 38
Brianna-Domain 54
Jonathan-Value 5
Max-Head 48
Adam-Bobbie 68
Jack-99 không thể tả
UNIX> sed 's / ^ * //' input-nn.txt | sed 's / * / - / g' | sed 's / - \ ([0-
9] \) / \ 1 /' | sed ' s / - $ // '| head
Molly-60 lên trời
Taylor-mây che 47
Brody-trễ 56
Tristan-Covenant 75
Adam-nhuộm 38
Brianna-Domain 54
Jonathan-Value 5
Max-Head 48
Adam-Bobbie 68
Jack-99 không thể tả
UNIX> sed 's / ^ * //' input-nn.txt | sed 's / * / - / g' | sed 's / - \ ([0-
9] \) / \ 1 /' | sed ' s / - $ // '| awk' {print $ 2, $ 1} '| head
60-Molly lên trời
47 Taylor-mây che
56-Brody trễ
75 Tristan-Covenant
38 Adam-nhuộm
54 Brianna-Domain
5 Jonathan-Value
48 Max-Head
68 Adam-Bobbie
99-Jack không thể tả
UNIX> sed 's / ^ * //' input-nn.txt | sed 's / * / - / g' | sed 's / - \ ([0-
9] \) / \ 1 /' | sed ' s / - $ // '| awk' {print $ 2, $ 1} '| sort -n | head
0 Addison-Paige-Chain
0 Eli-gneis
0 Ella-craftsperson
0 Lilly-Gianna-Zen
0 Matthew-cứng
1 Evan-thô lỗ
1 Isaiah-Chuyển hóa
1 Mason-Fourier
1 Xavier-Agave
2-Daniel Berman
UNIX> sed 's / ^ * //' input-nn.txt | sed 's / * / - / g' | sed 's / - \ ([0-
9] \) / \ 1 /' | sed ' s / - $ // '| awk' {print $ 2, $ 1} '| sort -n | awk'
{printf "% -40s% 2d \ n", $ 2, $ 1} '| head
Addison-Paige-Chain 0
```

```
Eli-gneis 0
Ella-craftsperson 0
Lilly-Gianna-Zen 0
Matthew-cứng 0
Evan-thô lỗ 1
Isaiah-Chuyển hóa 1
Mason-Fourier 1
Xavier-Agave 1
Daniel-Berman 2
UNIX> sed 's / ^ * //' input-nn.txt | sed 's / * / - / g' | sed 's / - \ ([0-
9] \) / \ 1 /' | sed ' s / - $ // '| awk' {print $ 2, $ 1} '| sort -n | awk'
{printf "% -40s% 2d \ n", $ 2, $ 1} '| sed' s / - / / g '| head
Addison Paige Chain 0
Eli gneis 0
Ella craftsperson 0
Lilly Gianna Zen 0
Matthew cứng 0
Evan thô lỗ 1
Isaiah Metabolism 1
Mason Fourier 1
Xavier Agave 1
Daniel Berman 2
UNIX> sed 's / ^ * //' input-nn.txt | sed 's / * / - / g' | sed 's / - \ ([0-
9] \) / \ 1 /' | sed ' s / - $ // '| awk' {print $ 2, $ 1} '| sort -n | awk'
{printf "% -40s% 2d \ n", $ 2, $ 1} '| sed' s / - / / g '> junk.txt
UNIX> openssl junk.txt md5 ra-1.txt
MD5 (junk.txt) = 4eee1503231b23c0052d9b3c57b1cd50
MD5 (output-1.txt) = 4eee1503231b23c0052d9b3c57b1cd50
UNIX>
```

Bây giờ, chúng ta hãy viết chương trình một lần thứ hai, chỉ có thời gian này, chúng tôi chỉ đơn giản là chèn **Person** cấu trúc như một chìa khóa, và viết một chức năng so sánh để so sánh các phím. Chương trình này là trong **ni_sort2.c** , và đây là những bộ phận có liên quan:

```
typedef struct {
  char * name;
  int điểm;
} Người;

int so sánh (Jval j1, j2 Jval)
{
  Person * p1, * p2;

  p1 = (Person *) j1.v;
  p2 = (Person *) j2.v;

  if (p1-> score> p2-> điểm) return 1;
  if (p1-> điểm <p2-> điểm) return -1;
  trở strcmp (p1-> tên, p2-> tên);
}

main ()
{
  JRB t, tmp;
  IS;
  Person * p;
  int nDung lượng, i;
```

```
  là = new_inputstruct (NULL);
  t = make_jrb ();

  while (get_line (là)> = 0) {

      .... Làm việc đọc và sáng tạo của con người

      / * Chúng tôi hiện chèn sử dụng jrb_insert_gen, với các cấu trúc người
như một chìa khóa. * /

      jrb_insert_gen (t, new_jval_v ((void *) p), new_jval_v (NULL), so
sánh);
    }
  }

  jrb_traverse (tmp, t) {
    p = (Person *) tmp-> key.v;
    printf ("% - 40% 2d \ n", p-> tên, p-> điểm);
  }
  exit (0);
}
```

Điều này không đòi hỏi nhiều bình luận, ngoại trừ phải nhớ rằng họ là jvals phím, vì vậy bạn phải định kiểu họ để loại mà bạn muốn trong các chức năng so sánh. Thật dễ dàng để khẳng định rằng đầu ra của chương trình này là giống như là cuối cùng:

```
UNIX> ni_sort2 <input-nn.txt | head
Addison Paige Chain 0
Eli gneis 0
Ella craftsperson 0
Lilly Gianna Zen 0
Matthew cứng 0
Evan thô lỗ 1
Isaiah Metabolism 1
Mason Fourier 1
Xavier Agave 1
Daniel Berman 2
UNIX> ni_sort2 <input-nn.txt> đầu ra-2.txt
UNIX> đầu ra openssl md5 -. * txt
MD5 (output-1.txt) = 4eee1503231b23c0052d9b3c57b1cd50
MD5 (output-2.txt) = 4eee1503231b23c0052d9b3c57b1cd50
UNIX>
```

Các chương trình cuối cùng là **ni_sort3.c** , và nó sử dụng một cây hai cấp. Mức đầu tiên là keyed trên điểm số, và chỉ có một nút cây mỗi điểm. Các val của mỗi nút là một cây đỏ-đen keyed vào tên, với các cấu trúc người trong val. Hãy xem cách chúng tôi tạo ra cây và in nó ra. Cá nhân tôi thích giải pháp này là tốt nhất, và đó là thực hành tốt cho bạn, bởi vì bạn sẽ tạo ra các cấu trúc dữ liệu như thế này tất cả các thời gian:

```
main ()
{
  JRB t, tmp, person_tree, t2;
```

```
  IS;
  Person * p;
  int nDung lượng, i;

  là = new_inputstruct (NULL);
  t = make_jrb ();

  while (get_line (là)> = 0) {

    if (là-> NF> 1) {
      ... Đọc và tạo ra người

      / * Để chèn người, đầu tiên chúng ta kiểm tra để xem nếu tỉ số là trong
         cây. Nếu không, chúng tôi tạo ra nó với một màu đỏ-đen cây rỗng như
một val.
         Trong cả hai trường hợp, chúng ta chèn tên vào cây thứ hai cấp. * /

      tmp = jrb_find_int (t, p> điểm);
      if (tmp == NULL) {
        person_tree = make_jrb ();
        jrb_insert_int (t, p> điểm, new_jval_v ((void *) person_tree));
      } Else {
        person_tree = (JRB) tmp-> val.v;
      }

      jrb_insert_str (person_tree, p-> tên, new_jval_v ((void *) p));
    }
  }

  / * Để in dân, chúng ta cần phải làm một lồng nhau, hai cấp đệ quy * /

  jrb_traverse (tmp, t) {
    person_tree = (JRB) tmp-> val.v;
    jrb_traverse (t2, person_tree) {
      p = (Person *) T2-> val.v;
      printf ("% - 40% 2d \ n", p-> tên, p-> điểm);
    }
  }
  exit (0);
}
```

Một lần nữa, chúng ta có thể kiểm tra để chắc chắn rằng nó là chính xác:

```
UNIX> ni_sort3 <input-nn.txt | head
Addison Paige Chain 0
Eli gneis 0
Ella craftsperson 0
Lilly Gianna Zen 0
Matthew cứng 0
Evan thô lỗ 1
Isaiah Metabolism 1
Mason Fourier 1
Xavier Agave 1
Daniel Berman 2
UNIX> ni_sort3 <input-nn.txt> đầu ra-3.txt
```

```
UNIX> đầu ra openssl md5 -. * txt
MD5 (output-1.txt) = 4eee1503231b23c0052d9b3c57b1cd50
MD5 (output-2.txt) = 4eee1503231b23c0052d9b3c57b1cd50
MD5 (output-3.txt) = 4eee1503231b23c0052d9b3c57b1cd50
UNIX>
```

---

## Một ví dụ khác: `` Golf ''

Dưới đây là một ví dụ điển hình của việc sử dụng một cây đỏ-đen. Nó không làm được gì nhiều ngoài những ví dụ cuối cùng, nhưng tôi cho nó để thực hành. Giả sử chúng ta có một loạt các tập tin với điểm số sân golf. Ví dụ như trong **1998_Majors** và **1999_Majors** . Các định dạng của các file này là:

**Tên chủ nhật-score** F **-tổng số điểm**

Ví dụ, một vài dòng đầu tiên của **1999_Majors / Masters** là:

```
Jose Maria Olazabal -1 F -8
Davis yêu III -1 -6 F
Greg Norman 1 F -5
Bob Estes 0 F -4
Steve Pate 1 F -4
David Duval -2 -3 F
Phil Mickelson -1 -3 F
...
```

Lưu ý rằng tên có thể có bất kỳ số lượng từ.

Bây giờ, giả sử rằng chúng ta muốn làm một số xử lý dữ liệu trên các tập tin. Ví dụ, giả sử chúng ta muốn sắp xếp mỗi người chơi để chúng tôi đầu tiên in ra các cầu thủ đã chơi hầu hết các giải đấu, và sau đó trong đó, chúng tôi sắp xếp theo các cầu thủ với số điểm trung bình thấp nhất.

Đây là những gì **golf.c** không. Nó có điểm số tập tin trên dòng lệnh, sau đó đọc tất cả các cầu thủ và điểm số. Sau đó, nó sắp xếp chúng theo số lượng các giải đấu / Điểm số trung bình, và in chúng ra theo thứ tự, cùng với điểm số của họ cho mỗi giải đấu. Ví dụ, nhìn vào **score1** :

```
Jose Maria Olazabal -1 F -8
Davis yêu III -1 -6 F
Greg Norman 1 F -5
```

và **score2** :

```
Greg Norman 1 F 9
David Frost 3 F 10
Davis yêu III -2 F 11
```

Các **golf** chương trình đọc trong hai tập tin, và đứng thứ bốn cầu thủ bởi số lượng các giải đấu, và sau đó điểm trung bình:

```
UNIX> golf score1 score2
Greg Norman: 2 giải đấu: 2.00
   -5: Score1
    9: score2
Davis yêu III: 2 giải đấu: 2.50
   -6: Score1
   11: score2
Jose Maria Olazabal: 1 giải đấu: -8,00
   -8: Score1
David Frost: 1 giải đấu: 10.00
```

```
   10: score2
```

Ok, bây giờ không bao **golf** làm việc? Vâng nó hoạt động trong ba giai đoạn. Trong giai đoạn đầu, nó đọc các tập tin đầu vào để tạo ra một cấu trúc cho mỗi người chơi golf. Các cấu trúc dữ liệu cho điều này là một cây đỏ-đen keyed vào tên của tay golf, và có val trường là **Golfer** cấu trúc mà có defintion sau đây:

```
typedef struct {
  char * name;
  int ntourn;
  int tscore;
  Điểm Dllist;
} Golfer;
```

Ba trường đầu tiên là hiển nhiên. Các trường cuối cùng là một danh sách các điểm của tay golf. Mỗi phần tử của danh sách những điểm đến một **số** cấu trúc với các định nghĩa sau đây:

```
typedef struct {
  char * tname; / * Tên tập tin * /
  int điểm; / * Tổng số điểm * /
} Điểm;
```

Lưu ý, trong mỗi tập tin, chúng ta sẽ bỏ qua trình `` điểm chủ nhật. "

Vì vậy, để đọc trong những người chơi golf, chúng tôi tạo ra cây JRB **chơi golf** , và sau đó đọc trong mỗi dòng của mỗi tập tin đầu vào. Đối với mỗi dòng, chúng ta xây dựng tên của tay golf, và sau đó chúng ta nhìn thấy nếu các tay golf có một mục trong các **golfer** cây. Nếu không có mục đó, sau đó người ta được tạo ra. Một khi vào được tìm thấy / tạo ra, tỷ số cho tập tin được thêm vào. Khi tất cả các tập tin đã được đọc, giai đoạn 1 đã hoàn thành:

```
  Golfer * g;
  Điểm * s;
  Golfer JRB, rnode;
  int i, fn;
  int tmp;
  IS;
  char name [1000];
  Dnode Dllist;

  golfer = make_jrb ();

  for (fn = 1; fn <argc; fn ++) {
    là = new_inputstruct (argv [fn]);
    if (là == NULL) {perror (argv [fn]); xuất cảnh (1); }

    while (get_line (là)> = 0) {

      / * Lỗi kiểm tra mỗi dòng * /

      if (là-> NF <4 || strcmp (là-> Trường [là-> NF-2], "F")! = 0 ||
          sscanf (là-> Trường [là-> NF-1], "% d", & tmp)! = 1 ||
          sscanf (là-> Trường [là-> NF-3], "% d", & tmp)! = 1) {
        fprintf (stderr, "Tập tin% s, Line% d: Không đúng định dạng \ n",
          là-> tên, là-> line);
        xuất cảnh (1);
      }
```

```
      / * Xây dựng tên của tay golf. Đây là mã lười biếng mà không hiệu quả,
bằng cách này * /

      strcpy (tên, là-> Trường [0]);
      for (i = 1; i <là-> NF-3; i ++) {
        strcat (tên, "");
        strcat (tên, là-> Trường [i]);
      }

      / * Tìm kiếm tên * /

      rnode = jrb_find_str (người chơi golf, tên);

      / * Tạo một mục nếu không tồn tại. * /

      if (rnode == NULL) {
        g = (Golfer *) malloc (sizeof (Golfer));
        g-> name = strdup (tên);
        g-> ntourn = 0;
        g-> tscore = 0;
        g-> điểm = new_dllist ();
        jrb_insert_str (người chơi golf, g-> tên, new_jval_v (g));
      } Else {
        g = (Golfer *) rnode-> val.v;
      }

      / * Thêm các thông tin của người chơi golf struct * /

      s = (Điểm *) malloc (sizeof (Score));
      s-> tname = argv [fn];
      s-> score = atoi (là-> Trường [là-> NF-1]);
      g-> ntourn ++;
      g-> tscore + = s-> điểm;
      dll_append (g-> điểm, new_jval_v (s));
    }

    / * Đi vào các tập tin kế tiếp * /

    jettison_inputstruct (là);
  }
```

Bây giờ, điều này cho chúng ta tất cả những thông tin về những người chơi golf, nhưng họ được sắp xếp theo tên của người chơi golf, không phải bằng số lượng các giải đấu / Điểm số trung bình. Vì vậy, trong giai đoạn 2, chúng ta xây dựng một màu đỏ-đen cây thứ hai mà sẽ sắp xếp những người chơi golf một cách chính xác. Để làm được điều này, chúng ta cần phải xây dựng chức năng so sánh của riêng của chúng tôi so sánh các golf thủ bởi số lượng các giải đấu / Điểm số trung bình. Dưới đây là các chức năng so sánh:

```
int golfercomp (Jval j1, j2 Jval)
{
  Golfer * g1, g2 *;

  g1 = (Golfer *) j1.v;
  g2 = (Golfer *) j2.v;
```

```
  if (g1-> ntourn> g2-> ntourn) return 1;
  if (g1-> ntourn <g2-> ntourn) return -1;
  if (g1-> tscore <g2-> tscore) return 1;
  if (g1-> tscore> g2-> tscore) return -1;
  return 0;
}
```

Và đây là một phần trong **chính** nơi mà các cây đỏ-đen thứ hai được xây dựng:

```
  sorted_golfers = make_jrb ();

  jrb_traverse (rnode, người chơi golf) {
    jrb_insert_gen (sorted_golfers, rnode-> val, JNULL, golfercomp);
  }
```

Lưu ý, bạn vượt qua một **Jval** để **jrb_insert_gen** .

Cuối cùng, giai đoạn thứ ba là để đi qua các **sorted_golfers** cây, in ra các thông tin chính xác cho mỗi người chơi golf. Điều này là dễ hiểu, và thực hiện dưới đây:

```
  jrb_rtraverse (rnode, sorted_golfers) {
    g = (Golfer *) rnode-> key.v;
    printf ("% - 40:% 3d% giải đấu 1s:% 7.2f \ n", g-> tên, g-> ntourn,
           (G-> ntourn == 1)? "": "S",
           (Float) g-> tscore / (float) g-> ntourn);
    dll_traverse (dnode, g-> điểm) {
      s = (Điểm *) dnode-> val.v;
      printf ("% 3d:% s \ n", s-> điểm, s-> tname);
    }
  }
```

Hãy thử nó ra. Bạn sẽ thấy rằng Tiger Woods đã làm tốt nhất trong tất cả bốn chuyên ngành năm nay:

```
UNIX> golf 1999_Majors / *
Tiger Woods: 4 giải đấu: 0.25
   10: 1999_Majors / British_Open
    1: 1999_Majors / Masters
  -11: 1999_Majors / PGA_Champ
    1: 1999_Majors / US_Open
Colin Montgomerie: 4 giải đấu: 3,75
   12: 1999_Majors / British_Open
   -1: 1999_Majors / Masters
   -6: 1999_Majors / PGA_Champ
   10: 1999_Majors / US_Open
Davis yêu III: 4 giải đấu: 4.50
   10: 1999_Majors / British_Open
   -6: 1999_Majors / Masters
    5: 1999_Majors / PGA_Champ
    9: 1999_Majors / US_Open
Jim Furyk: 4 giải đấu: 4.50
   11: 1999_Majors / British_Open
    0: 1999_Majors / Masters
   -4: 1999_Majors / PGA_Champ
   11: 1999_Majors / US_Open
Nick Giá: 4 giải đấu: 4,75
   17: 1999_Majors / British_Open
   -3: 1999_Majors / Masters
```

```
   -7: 1999_Majors / PGA_Champ
   12: 1999_Majors / US_Open
...
```