Derleme

Kırmızı-Siyah Ağaç Kütüphanesini kullanmak için ***/blugreen/homes/plank/cs360/include*** da bulabileceğiniz "jrb.h" dosyasını dahil edin. C dosyanızın içindeki tam yolu dahil etmek yerine sadece şunu yapıp;

#include "jrb.h"

Sonra da ***gcc -I/blugreen/homes/plank/cs360/include*** ile birlikte programı derleyin. Yürütülebilir bir hale getirmek için nesne dosyaları bağladığınızda, libfdr ders notlarındaki yönergeleri izleyin. Bu dizindeki makefile sizin için bu işleri yapar.

Kırmızı-Siyah (RB) Ağaçlar

RB Ağaçlar; dengeli ikili ağaç yapısına dayalı veri yapılarıdır. Bu işi yapmak için onların nasıl çalıştığını bilmenize gerek yoktur. Ağaçtaki her **n** eleman sayısı için tüm işlemleri **0(log(n))** zamanı içinde sizin yerinize gerçekleştirir. (Eğer gerçekten kırmızı-siyah ağaçlar hakkında daha fazla bilgi edinmek istiyorsanız,bana haber verin ben de size bunlar üzerine bazı metinler göstereyim.)

RB Ağaçlar için temel yapı JRB'dir. Dllist de olduğu gibi tüm RB Ağaçlar bir baş düğüme sahiptir. Boş bir RB Ağacın baş düğümünün adresini döndüren **make_jrb()** ile yeni bir RB Ağaç oluşturabilirsiniz. Bu baş düğüm RB Ağacının ana gövdesini gösterir. Ayrıca Ağacın ilk ve son dış düğümleri için de endişelenmenize gerek yoktur. Bu dış düğümler flink ve blink işaretçileri ile sıralanır; böylece Dllistlerde olduğu gibi RB Ağaçlarını sıralı bir şekilde görebilir ve ağaç içindeki herhangi bir düğümü 0(log(n)) zamanında bulabilirsiniz.

Dllistlerde olduğu gibi ağaç içindeki her düğüm Jval olan bir val alanına sahiptir.Ayrıca her düğüm bir Jval anahtar alanına sahiptir. RB ağaçları düğümlerin sıralı olup olmadığından, sıralı ise neye göre sıralı olduğundan emin olur.

_str, _int, _dbl, _gen

jrb.h/jrb.c deki JRB Ağacı rutinleri ekleme/arama olmak üzere 4 rutin içerir. Ekleme terimleri şunlardır:

- **JRB jrb_insert_str(JRB tree, char *key, Jval val)**: Standart bir karakter dizesi kullanarak ağaca yeni bir düğümü anahtar olarak yerleştirir. Strcmp() karşılaştırma fonksiyonu olarak kullanılır. Jrb_insert_str() ile yapılan standart giriş sıralaması ile ilgili basit bir örnek için strsort.c dosyasına bakınız.

  Bu komutun yeni bir Jrb Ağaç düğümünün adresini döndürdüğünü unutmayın. Ayrıca ağaçta bir anahtar olsa bile yeni bir düğüm üretip bu düğümü ağaca eklediğini unutmayın. Yinelenen anahtarın göreli sıralamalarına ilişkin hiçbir garanti vermez.

  Anahtar bir string olsa bile yeni düğümde Jval e dönüştürülecektir. Eğer s düğümünün anahtarını almak isterseniz jval_s(s->key) veya s->key.s komutlarını kullanmalısınız.
- **JRB jrb_insert_int(JRB tree, int key, Jval val):**anahtar olarak integer kullanıp ağaca yerleştirir.

- **JRB jrb_insert_dbl(JRB tree, double key, Jval val):**anahtar olarak double kullanıp ağaca yerleştirir.
- **JRB jrb_insert_gen(JRB tree, Jval key, Jval val, int (*func)(Jval, Jval))**:Şimdi senin anahtarın bir 'jval'.Argumanler ve geri dönüş değerleri olarak 2 Jval değeri alan func() karşılaştırma fonksiyonu sağlarsın.

Eger ilk anahtar ikincisinden daha az ise negatif integer

Eger ilk anahtar ikincisinden daha fazla ise pozitif anahtar

Eşitse sıfırdır.

Bu integer ve stringlerle basit tiplerden daha sofistike yapımlara izin verir.Örneğin; 'strisort.c' string tipleri ancak durumu önemsemez.'strrsort2.c' string tipleri terstendir.Bunları oku.

Sen karıştıramazsın ve aynı ağaçın içinde karşılaştırma fonksiyonlarını eşleştir.Başka bir deyişle, jrb_insert_str() ve jrb_insert_int() ile bazı anahtarları yerleştirmemen gerekir.Bunu yapmak için bir çekirdek çöplüğü için yalvarıyor olacaksın.

Anahtarları bulmak için , jrb_find_str(), **jrb_find_int()**, **jrb_find_dbl()** ya da **jrb_find_gen()** kullanılır.Açık olarak, **jrb_insert_str()** ile anahtarları yerleştirirsen, senin onları bulmak için sonradan **jrb_find_str**() kullanman gerekir.Aradığın anahtarlar ağaçta yoksa , **jrb_find_*xxx*()** NULL değeri döndürür.

Son olarak **jrb_find_gte_str()**, **jrb_find_gte_int()**, **jrb_find_gte_dbl()** ve **jrb_find_gte_gen()** fonksiyonlarımız mevcut.Bunlar özel anahtar için eşit , küçük ya da büyük olma durumuna göre ağaç düğümünde değer döndürür.Özel anahtar ağaçtan daha büyükse , başlangıç düğümü için bir pointer döndürecek. Anahtar buldu ya da bulmadıysa sana ayarlanmış bir 'found' argümanı vardır.

jrb_first(), **jrb_last()**, **jrb_prev()** ve **jrb_next()** de sadece dllist kütüphanesinde yapıldığı gibi makrolar kullanabilirsin.

Örnek Programlar

strsort.c: Standart girişi öncelikli sıralamak için red-black ağacı kullanılır.

strrsort1.c: Standart girişi tersten öncelikli sıralamak için red-black ağacı kullanılır. Tersten ağaç geçme yapar.

strrsort2.c: Standart girişi tersten öncelikli sıralamak için red-black ağacı kullanılır. Yeni bir **revcomp** karşılaştırma fonksiyonu oluşturarak basit değer döndüren **strcmp()** i yapar.

strusort.c: Standart girişi öncelikli sıralamak için red-black ağacı kullanılır ve kopya satırları kaldırır. Ağaca onu yerleştirmeden önce bu hat için kontrol yapar.

strisort.c: Standart girişi öncelikli sıralamak için red-black ağacı kullanılır, üst veya alt bölümleri önemsemez. Yeni bir **ucomp** karşılaştırma fonksiyonu oluşturarak, bölümleri(alt ve üst) önemsemeyen **strcmp()** fonksiyonunu yapar.

nsort.c: sort –n gibi sıralama için red-black ağacını kullanır.Her satıra integer olarak davranır ve bu şekilde sıralar.Satırlar integer değilse ya da tekrar satırları varsa , hiç bir şey yapmaz.

nsort2.c: sort –n gibi sıralamalar için red-black ağacını kullanır. 2 satır aynı **atoi()** değerine sahipse, öncelikli sıralanırlar.Bunu **jrb_insert_gen()** kullanır.

nsort3.c: nsort2 in aynısı, fakat 2 düzey red-black ağacı kullanılır.Açıklama için aşağıya bakın.

2 Düzey Ağaç Örneği

atoi() değeri tarafından metnin sıralama satırlarını istediğimizi varsayalım, fakat 2 dizi aynı **atoi()** değerine sahipse, onları öncelikli sırala.Bunu yapmanın bir yolu,güçlendirilmiş bir karşılaştırma fonksiyonu kullanmak ve **nsort2.c** içinde satırları **jrb_insert_gen()** ile yerleştirmek**.input_n2** üzerinde bunu dene.

2. bir yol da 2 Düzey Ağacın olması.İlk ağacın anahtar olarak integerleri olsun ve her satır değerini **atoi()** üzerinden kalıtım alsın, diğer red-black ağacının her düğümün **val** değeridir.Bu red-black ağacı düğümün anahtarında eşit olan **atoi()** değerinin her satırını içerir, öncelikli sıralar. Böylece satırı okuduğun zaman, onun **atoi()** değeri ağaçta ise ilk göreceksin.Görmezsen, anahtarı atoi() olan ve val alanı yeni olan ,boş red-black ağacını ağaca yeni bir düğüm olarak ekle. Şimdi anahtarının string değeri **atoi()** olan düğümün **val** alanında red-black ağacı için senin bir pointerin var. Şimdi yapacağımız şey, **jrb_insert_str()** kullanarak bu 2. Red-black ağacına dizi yerleştirmek.Bunu yaptığın zaman , büyük bir 2 Düzey Red-Black ağacın olacak.En yüksek düzey ağaç geçmesi tarafından onu çaprazlama geçersin ve bu ağaçtaki her düğüm için, sen içinde **val** değeri ve dizi çıktılarının olduğu ağacı çaprazlama geçersin.Koda bak.Bu **nsort3.c** nin kodu.

Diğer Örnek : “Golf”

Red-Black ağacı kullanılan tipik bir örnek aşağıda mevcut.Farzedelim, bizim golf skorlarıyla bir dosya demetimiz olsun.Örnekler **1998_Majors** ve **1999_Majors** de mevcut.Bu dosyaların formatı:

Name sunday-score F **total-score**

Örneğin, **1999_Majors/Masters** ın ilk birkaç satırı:

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

Numarası olmayan kelimelerinin isimlerini not al.

Şimdi, bu dosyalar üzerinde birkaç işlenebilir veri yapmak istediğimizi varsayalım. Örneğin, her oyuncuyu sıralamak istediğimizi varsayalım ki ilk olarak en çok turnuva oynayan oyuncuları yazarız ve sonra bunları en düşük averaj skorlu oyuncuya göre sıralarız.

Bu da **golf.c** in yapılışı.Komut satırında skor dosyalarını alır sonra oyuncuları ve skorların hepsini okur.Sonra onları turnuva veya averaj skorlarına göre sıralar ve o sırada onları her turnuvada aldıkları puan ile birlikte yazar.Örneğin, **score1** e bakın:

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

Ve **score2:**

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

Golf programı bu 2 dosyanın içindekileri okur ve turnuva numarasına göre 4 oyuncuyu derecelendirir ve sonra averaj skoru:

```
UNIX> golf score1 score2
Greg Norman                  :   2 tournaments :    2.00
   -5 : score1
    9 : score2
Davis Love III               :   2 tournaments :    2.50
   -6 : score1
   11 : score2
Jose Maria Olazabal              :   1 tournament  :   -8.00
   -8 : score1
David Frost                  :   1 tournament  :   10.00
   10 : score2
```

Tamam, şimdi **golf** nasıl çalıştırılır? Üç aşamada çalışır.İlk aşama , her golfçü için bir yapı oluşturmak için giriş dosyaları okunur. Aşağıdaki tanımda **Golfer** yapılarının val alanlarının golfçülerin isimlerinin üzerinde red-black ağacı anahtarlı olması için yapılmış bir veri yapısı mevcut:

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

İlk 3 alan belli. Golfçülerin skorlarının listesi son alanda mevcut. Aşağıdaki tanım ile bir **Score** yapısı için puan listesinin her öğesi mevcut:

```
typedef struct {
  char *tname;          /* File name */
  int score;           /* Total score */
} Score;
```

Her dosyada Pazar skorlarını yoksayacağımızı not alın.

Golfçüleri okumak için , jrb ağaç **golfers** oluştururuz ve sonra her dosya giriş satırını okuruz. Her satır için, golfçülerin ismini kurarız ve sonra eğer golfçü **golfers** ağacında bir girişe sahipse, görmek için bakarız. Giriş mevcut değilse bir tane oluştururuz. Eklenen dosya için skor olan giriş bir kere bulundu veya oluşturuldu.Tüm dosyalar okunduğu zaman, Aşama 1 tamamlandı:

```
Golfer *g;
  Score *s;
  JRB golfers, rnode;
  int i, fn;
  int tmp;
  IS is;
  char name[1000];
  Dllist dnode;

  golfers = make_jrb();

  for (fn = 1; fn < argc; fn++) {
    is = new_inputstruct(argv[fn]);
    if (is == NULL) { perror(argv[fn]); exit(1); }

    while(get_line(is) >= 0) {

      /* Her satır için hata kontroü */

      if (is->NF < 4 || strcmp(is->fields[is->NF-2], "F") != 0 ||
          sscanf(is->fields[is->NF-1], "%d", &tmp) != 1 ||
          sscanf(is->fields[is->NF-3], "%d", &tmp) != 1) {
        fprintf(stderr, "File %s, Line %d: Not the proper format\n",
          is->name, is->line);
        exit(1);
      }

      /* Golfçünün ismi yapıldı */
      strcpy(name, is->fields[0]);
      for (i = 1; i < is->NF-3; i++) {
        strcat(name, " ");
        strcat(name, is->fields[i]);
      }

      /* Adı aranıyor */

      rnode = jrb_find_str(golfers, name);

      /* Giriş yoksa giriş oluşturuluyor. */

      if (rnode == NULL) {
        g = (Golfer *) malloc(sizeof(Golfer));
        g->name = strdup(name);
        g->ntourn = 0;
        g->tscore = 0;
        g->scores = new_dllist();
        jrb_insert_str(golfers, g->name, new_jval_v(g));
```

```
    } else {
      g = (Golfer *) rnode->val.v;
    }

    /* Golfçünün yapısı için bilgi ekle */

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

   Bir sonraki dosyaya git */

  jettison_inputstruct(is);
 }
```

Şimdi, bu, golfçülerin üzerinden bütün bilgileri bize verir fakat turnuva numarasına ya da averaj skoruna göre değil de golfçülerin isimlerine göre sıralandılar. Böylece, 2. Aşamada, biz golfçülerin doğru bir şekilde sıralanacağı bir adet 2. Red-black ağacı yaparız.Bunu yapmak için, turnuva numarası veya averaj skoruna göre golfçüleri karşılaştıran kendi karşılaştırma fonksiyonumuzu yapmaya ihtiyacımız olması gerekir. Aşağıda bu karşılaştırma fonksiyonu mevcuttur:

```
int golfercomp(Jval j1, Jval j2)
{
  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;
}
```

Ve burada 2. Red-black ağacının oluşturulduğu yerdeki **main** in parçası mevcut:

```
  sorted_golfers = make_jrb();

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

jrb_insert_gen için bir **Jval** geçeriz.(Not al)

Son olarak , 3. Aşama **sorted_golfers** ağacını çapraz geçmek, her golfçü için doğru bilgiyi yazmaktır. Bu basittir ve aşağıda yapılmıştır:

```
jrb_rtraverse(rnode, sorted_golfers) {
    g = (Golfer *) rnode->key.v;
    printf("%-40s : %3d tournament%1s : %7.2f\n", g->name, g->ntourn,
          (g->ntourn == 1) ? "" : "s",
          (float) g->tscore / (float) g->ntourn);
    dll_traverse(dnode, g->scores) {
      s = (Score *) dnode->val.v;
      printf("  %3d : %s\n", s->score, s->tname);
    }
  }
```

Bunu dene. Tiger Woods'un bu yıl 4 büyüklerin hepsi içinde en iyisini yaptığını göreceksin:

```
UNIX> golf 1999_Majors/*
Tiger Woods                    :   4 tournaments :    0.25
   10 : 1999_Majors/British_Open
    1 : 1999_Majors/Masters
  -11 : 1999_Majors/PGA_Champ
    1 : 1999_Majors/US_Open
Colin Montgomerie              :   4 tournaments :    3.75
   12 : 1999_Majors/British_Open
   -1 : 1999_Majors/Masters
   -6 : 1999_Majors/PGA_Champ
   10 : 1999_Majors/US_Open
Davis Love III                 :   4 tournaments :    4.50
   10 : 1999_Majors/British_Open
   -6 : 1999_Majors/Masters
    5 : 1999_Majors/PGA_Champ
    9 : 1999_Majors/US_Open
Jim Furyk                      :   4 tournaments :    4.50
   11 : 1999_Majors/British_Open
    0 : 1999_Majors/Masters
   -4 : 1999_Majors/PGA_Champ
   11 : 1999_Majors/US_Open
Nick Price                     :   4 tournaments :    4.75
   17 : 1999_Majors/British_Open
   -3 : 1999_Majors/Masters
   -7 : 1999_Majors/PGA_Champ
   12 : 1999_Majors/US_Open
...
```