CS360 Lecture ghi chú - Chức năng Gấp đôi Linked

- Jim Plank
- Thư mục: **~ ván / tấm ván / classes / cs360 / 360 / www-home / ghi chú / Dllists**
- Bài giảng: **http://www.cs.utk.edu/~plank/plank/classes/cs360/360/notes/Dllists**
- Ghi chú Original: *Wed 25 Tháng Tám 1999 10:14:37 EDT*
- Sửa đổi lần cuối: *Wed 25 tháng một năm 2012 10:11:15 EST*

Chức năng kép liên kết

Danh sách liên kết kép là như danh sách liên kết đơn lẻ, trừ mỗi node có hai con trỏ - một đến nút tiếp theo, và một đến nút trước đó. Điều này làm cho cuộc sống tốt đẹp trong nhiều cách:

- Bạn có thể đi qua các danh sách phía trước và lạc hậu.
- Bạn có thể chèn bất cứ nơi nào trong danh sách dễ dàng. Điều này bao gồm việc chèn trước khi một nút, sau khi một nút, ở phía trước của danh sách, và ở cuối danh sách.
- Bạn có thể xóa các nút rất dễ dàng.

Các API cho các danh sách liên kết kép trong **dllist.h** . Nó định nghĩa một nút danh sách gấp đôi được liên kết:

```
typedef struct {dllist
  struct dllist * flink;
  struct dllist * chớp;
  Jval val;
} * Dllist;
```

Dưới đây là các hoạt động hỗ trợ bởi **dllist.o** :

- **New_dllist Dllist ()** : Phân bổ và trả về một danh sách gấp đôi liên kết mới.
- **free_dllist (Dllist l)** : Phá hủy danh sách, gọi **miễn phí ()** trên tất cả các cấp phát bộ nhớ trong danh sách. Danh sách này không phải là trống rỗng.
- **dll_prepend (Dllist l, Jval val)** : Thêm một nút mới vào đầu danh sách. Giá trị của nút này là **val** . **Dll_prepend ()** không có giá trị trả về.
- **dll_append (Dllist l, Jval val)** : Thêm một nút mới vào cuối danh sách. Giá trị của nút này là **val** . **Dll_append ()** không có giá trị trả về.
- **dll_insert_b (Dllist n, Jval val)** : Thêm một nút mới vào danh sách ngay trước khi nút được chỉ định. Giá trị của nút này là **val** .
- **dll_insert_a (Dllist n, Jval val)** : Thêm một nút mới vào danh sách ngay sau khi các nút được chỉ định. Giá trị của nút này là **val** .

- **Dllist dll_nil (Dllist l)** : Trả về một con trỏ trỏ tới **nil** (sentinel) nút cho danh sách. Bạn có thể nghĩ về **con số không** như là một nút không có giá trị, bắt đầu và kết thúc danh sách. Vì **l** điểm đến các hạch, **dll_nil** trả **l** . Bạn không cần phải gọi **dll_nil ()** để truy cập các nút trọng điểm. Bạn chỉ có thể sử dụng **l** , mặc dù nó làm cho mã của bạn dễ đọc hơn nếu bạn sử dụng **dll_nil ()** .
- **Dllist dll_first (Dllist l)** : Trả về một con trỏ đến nút đầu tiên trong danh sách. Nếu danh sách rỗng, này trả về sentinel. Như với **dll_nil ()** , bạn không cần phải gọi **dll_first (l)** - bạn chỉ có thể sử dụng **l-> flink .**
- **Dllist dll_last (Dllist l)** : Trả về một con trỏ đến nút cuối cùng trong danh sách. Nếu danh sách rỗng, này trả về sentinel. Như với **dll_nil ()** , bạn không cần phải gọi **dll_last (l)** - bạn chỉ có thể sử dụng**l-> blink .**
- **Dllist dll_next (Dllist n)** : Trả về một con trỏ đến nút tiếp theo trong danh sách sau khi **n .** Nếu **n** là nút cuối cùng trong danh sách, sau đó **dll_next (n)** trả về sentinel. Như với **dll_first ()** , bạn không cần phải gọi **dll_next (n)** - bạn chỉ có thể sử dụng **n-> flink .**
- **Dllist dll_prev (Dllist n)** : Trả về một con trỏ đến nút trước đó trong danh sách trước khi **n .** Nếu **n** là nút đầu tiên trong danh sách, sau đó **dll_prev (n)** trả về sentinel. Như với **dll_last ()** , bạn không cần phải gọi **dll_prev (n)** - bạn có thể chỉ cần sử dụng **n-> blink .**
- **int dll_empty (Dllist l)** : Trả về cho dù **l** là trống rỗng.
- **Dll_val Jval (Dllist n)** : Trả về nút **n** 's **val** trường. Một lần nữa, bạn không cần phải sử dụng này, nhưng đôi khi nó có ích.
- **int dll_delete_node (Dllist n)** : Xóa và giải phóng nút **n** .

Cuối cùng, có hai macro để vượt qua **dllists** phía trước và phía sau. **ptr** phải là một **Dllist** và **danh sách** phải là một **Dllist** :

```
#define dll_traverse (ptr, danh sách) \
  for (ptr = (danh sách) -> flink;! ptr = (danh sách); ptr = ptr-> flink)
#define dll_rtraverse (ptr, danh sách) \
  for (ptr = (danh sách) -> blink; ptr = (danh sách); ptr = ptr-> chớp!)
```

Thực hiện

Việc thực hiện của mỗi **dllists** là như là một danh sách gấp đôi liên kết tròn với một nút trọng điểm. Mã này là trong **dllist.c** .

Các **typedef** cho một **dllist** nút là:

```
typedef struct {dllist
  struct dllist * flink;
  struct dllist * chớp;
  Jval val;
} * Dllist;
```

Lưu ý rằng mỗi node có hai con trỏ - một liên kết chuyển tiếp (**flink**) đến nút tiếp theo trong danh sách, và một liên kết ngược (**blink**) đến nút trước đó trên danh sách. Một **Dllist** là một con trỏ đến nút trọng điểm.

Danh sách này là tròn ở cả hai hướng - của sentinel **flink** điểm đến nút đầu tiên trong danh sách, và nó **chớp** điểm đến nút cuối cùng trong danh sách. Nút đầu tiên của **blink** điểm đến trọng điểm, cũng như các nút cuối cùng của **flink** .

Một số nghệ thuật ascii: Dưới đây là một danh sách trống **l** :

```
l ------------- + -> | ----------- |
                | | Flink --------- \
                | | Blink ------- \ |
                | | Val =? | | |
                | | ----------- | | |
                | | |
                \ ------------------- + - /
```

Và đây là danh sách đó sau khi gọi **dll_append (l, new_jval_i (3));** : (hoặc **dll_prepend (l, new_jval_i (3))** cho rằng vấn đề).

```
l ------------- + -> | ----------- | / - + -> | ----------- |
                | | Flink ------- / | | flink --------- \
                | | Blink --------- / | blink ------- \ |
                | | Val =? | | Val.i = 3 | | |
                | | ----------- | | ----------- | | |
                | | |
                \ ------------------------------------- + - /
```

Trên thực tế, nó làm cho các bản vẽ sạch hơn để có các liên kết trở lại đi ngược:

```
l ----------------> | ----------- | | ----------- |
                / ---> | Flink -----------> | flink --------- \
                | / ------ Chớp | <----------- chớp | <- \ |
                | | | Val =? | | Val.i = 3 | | |
                | | | ----------- | | ----------- | | |
                | | | |
                | \ ---------------------------------- / |
                | |
                \ ---------------------------------------- /
```

Dưới đây là danh sách đó sau khi gọi **dll_append (l, new_jval_i (5));** :

```
l ----------> | ----------- | | ----------- | | ----------- |
        / ---> | Flink -----------> | flink -----------> | flink --------- \
        | / ------ Chớp | <----------- chớp | <----------- chớp | <- \ |
        | | | Val =? | | Val.i = 3 | | val.i = 5 | | |
        | | | ----------- | | ----------- | | ----------- | | |
        | | | |
        | \ ------------------------------------------------ -------- / |
        | |
        \ ------------------------------------------------ ------------ /
```

Tôi sẽ không đi qua nhiều ví dụ với ascii art. Bạn cần phải nhận được hang này bởi bây giờ.

Nhiều người trong số các thủ tục triển khai các thủ tục tầm thường hoặc macro:

```
New_dllist Dllist ()
{
  Dllist d;

  d = (Dllist) malloc (sizeof (struct dllist));
  d-> flink = d;
```

```
  d-> blink = d;
  trở d;
}
```

```
dll_empty (Dllist l)
{
  trở lại (l-> flink == l);
}
```

```
free_dllist (Dllist l)
{
  while (! dll_empty (l)) {
    dll_delete_node (dll_first (l));
  }
  miễn phí (l);
}
```

```
#define dll_first (d) ((d) -> flink)
#define dll_next (d) ((d) -> flink)
#define dll_last (d) ((d) -> blink)
#define dll_prev (d) ((d) -> blink)
#define dll_nil (d) (d)
```

Các chỉ mảnh tinh tế của mã là **dll_insert_b ()** và **dll_delete_node** . Với **dll_insert_b (n, v)** , chúng tôi **malloc ()** một nút mới, thiết lập giá trị của nó **v** , và sau đó liên kết nó vào danh sách ngay trước khi **n** .Điều này có nghĩa rằng chúng ta thiết lập các nút mới của **flink** trường để **n** , và nó **blink** trường để **n-> blink** . Sau đó, chúng tôi thiết lập **n-> blink** đến nút mới và cũ **n-> blink** 's **flink** trường vào các nút mới. Đây là các mã:

```
dll_insert_b (Dllist nút, Jval v) / * Chèn một nút cho trước * /
{
  Dllist mới;

  mới = (Dllist) malloc (sizeof (struct dllist));
  mới-> val = v;

  mới-> flink = nút;
  mới-> blink = node-> blink;
  mới-> flink-> blink = new;
  mới-> blink-> flink = new;
}
```

Một khi chúng ta đã **dll_insert_b ()** thói quen chèn ba danh sách khác chỉ đơn giản là gọi đến **dll_insert_b ()** :

```
dll_insert_a (Dllist n, Jval val) / * Chèn sau khi một nút cho trước * /
{
  dll_insert_b (n-> flink, val);
}

dll_append (Dllist l, Jval val) / * Chèn vào cuối danh sách * /
```

```
{
  dll_insert_b (l, val);
}

dll_prepend (Dllist l, Jval val) / * Chèn vào đầu danh sách * /
{
  dll_insert_b (l-> flink, val);
}
```

Xóa là khá dễ dàng quá. Trước tiên, bạn phải gỡ bỏ các nút **n** 's từ danh sách bằng cách thiết lập **n-> flink-> blink** để **n-> blink** và bằng cách thiết lập **n-> blink-> flink** để **n-> flink** . Sau đó, bạn tự do **n** :

```
dll_delete_node (Dllist nút) / * Xóa một iTerm tùy ý * /
{
  node-> flink-> blink = node-> blink;
  node-> blink-> flink = node-> flink;
  miễn phí (node);
}
```

Ví dụ sử dụng

Ví dụ đầu tiên là một trong những tiêu chuẩn của chúng tôi: đảo ngược đầu vào tiêu chuẩn. Điều này đơn giản là đủ để không cần giải thích. Đó là trong **dllreverse.c** :

```
#include <stdio.h>
#include <string.h>
#include "fields.h"
#include "dllist.h"

main ()
{
  IS;
  Dllist l;
  Tmp Dllist;

  là = new_inputstruct (NULL);
  l = new_dllist ();

  while (get_line (là)> = 0) {
    dll_append (l, new_jval_s (strdup (là-> text1)));
  }

  dll_rtraverse (tmp, l) printf ("% s", jval_s (tmp-> val));
}
```

Ví dụ thứ hai là một chuẩn khác: in ấn cuối cùng **n** dòng đầu vào tiêu chuẩn. Chúng tôi làm điều này bằng cách đọc vào tiêu chuẩn vào một **Dllist** , và đảm bảo rằng các **Dllist** luôn có ít nhất **n** nút. Sau đó, chúng tôi in ra: Mã này là trong **dlltail.c** :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
#include "fields.h"
#include "dllist.h"

main (int argc, char ** argv)
{
  IS;
  int n;
  Dllist l;
  Tmp Dllist;

  if (argc! = 2) {
    fprintf (stderr, "sử dụng: dlltail n \ n");
    xuất cảnh (1);
  }
  n = atoi (argv [1]);
  if (n <0) {
    fprintf (stderr, "sử dụng: dlltail n - n phải> = 0 \ n");
    xuất cảnh (1);
  }

  là = new_inputstruct (NULL);
  l = new_dllist ();

  while (get_line (là)> = 0) {
    dll_append (l, new_jval_s (strdup (là-> text1)));
    if (là-> dòng> n) {
      tmp = dll_first (l);
      miễn phí (jval_s (dll_val (tmp)));
      dll_delete_node (tmp);
    }
  }

  dll_traverse (tmp, l) printf ("% s", jval_s (tmp-> val));
}
```

Một vài ghi chú về chương trình này. Đầu tiên, bạn phải gọi **strdup ()** để sao chép chuỗi mà bạn đã đọc từ đầu vào tiêu chuẩn. Nếu không, **get_line ()** sẽ ghi đè lên các dây, và tất cả các đường dây của bạn sẽ giống nhau (chúng tôi đã đi qua khái niệm này nhiều lần, nhưng tôi muốn giữ vào server của nhà điểm).

Thứ hai, khi bạn gọi **dll_delete_node ()** nó loại bỏ các nút từ danh sách và giải phóng các node. Tuy nhiên, nếu **val** điểm trường dữ liệu đã được cấp phát bằng **malloc ()** , sau đó dữ liệu được tự động giải phóng không. Đó là bởi vì các **dllist** thư viện không có ý tưởng những gì các dữ liệu. Có lẽ bạn muốn nó được giải phóng, hoặc có lẽ bạn không làm, vì nó là trên mutiple cấu trúc dữ liệu.

Trong chương trình này, bạn chắc chắn nhất muốn giải phóng dữ liệu - giả sử bạn đọc một tập tin có 10G giá trị của văn bản và bạn chỉ đơn giản là muốn đọc 10 dòng cuối cùng - nếu bạn không giải phóng các dữ liệu khi bạn xóa một nút , bạn sẽ đốt cháy 10G bộ nhớ. Nếu bạn làm giải phóng các dữ liệu, sau đó bạn chỉ lưu trữ 10 dòng trong bộ nhớ tại một thời điểm.

Một ví dụ cuối

Việc chuyển nhượng dưới đây là từ CS140, trong C ++:

COS: Cột dây và tăng gấp đôi

Bạn sẽ ghét chương trình này. Xin lỗi. Bạn sẽ viết **cos.cpp** . Này đọc từ từ đầu vào tiêu chuẩn và phân vùng chúng thành đôi và không đánh đôi. On line 1, nó sẽ in đầu tiên không phải đôi và lần đầu tiên đôi. On line 2, nó sẽ in thứ hai không đôi và đôi thứ hai. Và như vậy. Định dạng của mỗi dòng sẽ là:

- Các phi-double, độn đến 30 ký tự và trái lý.
- Một không gian.
- Các đôi, độn đến 20 ký tự, phải được thực hiện với bốn chữ số sau dấu thập phân.

Như vậy, mỗi dòng sẽ là 51 ký tự. Nếu có cùng số lượng tăng gấp đôi và không tăng gấp đôi, sau đó cuộc sống là dễ dàng. Tuy nhiên, nếu có những đôi hơn không đôi, bạn chỉ cần in 30 không gian khi bạn nhận được các dòng mà không có sự đánh đôi. Nếu có nhiều phi đôi hơn gấp đôi, sau đó bạn chỉ cần in 20 không gian nơi các đôi sẽ đi.

Bạn có thể giả định rằng từ trong tiêu chuẩn đầu vào là 30 ký tự hoặc ít hơn.

Ví dụ:

```
UNIX> mèo input1.txt
1 2 3 Fred Binky Dontonio
UNIX> cos <input1.txt
Fred 1.0000
Binky 2,0000
Dontonio 3,0000
UNIX> mèo input2.txt
1 2 3 Fred
UNIX> cos <input2.txt
Fred 1.0000
                                                2,0000
                                                3,0000
UNIX> mèo input3.txt
1 2 Fred Binky Dontonio
UNIX> cos <input3.txt
Fred 1.0000
Binky 2,0000
Dontonio
UNIX> cos <input3.txt | cat -e
Fred $ 1.0000
Binky $ 2,0000
Dontonio $
UNIX>
```

Trong trường hợp bạn đang tự hỏi, "mèo -e" bản in đầu vào tiêu chuẩn trên đầu ra tiêu chuẩn, và đưa ra một '$' ở cuối dòng. Nó là tốt đẹp để có thể nhìn thấy không gian ở cuối dòng.

Các giải pháp là sử dụng hai Dllists - một cho đôi và một cho chuỗi. Khi bạn đọc stdin, bạn sử dụng **sscanf ()** để xác định xem một từ là một đôi, và nếu được, bạn đặt nó vào danh sách gấp đôi. Nếu không, bạn gọi **strdup ()** và đặt chuỗi vào danh sách chuỗi. Sau đó bạn in ra theo ba giai đoạn. Trong lần đầu tiên, bạn in tất cả các dòng trong đó có cả đôi và dây. Khi bạn đang thực hiện với vòng lặp, bạn có thể tăng gấp đôi hoặc chuỗi còn sót lại. Hai tuyến còn lại xử lý từng trường hợp. Mã này là trong **cos.c** :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "fields.h"
#include "dllist.h"

main ()
{
  Dl Dllist, dtmp; / * Các danh sách cho đôi * /
  Sl Dllist, STMP; / * Các danh sách cho các chuỗi * /
  double d;
  int i;
  IS;

  dl = new_dllist ();
  sl = new_dllist ();
  là = new_inputstruct (NULL);

  while (get_line (là)> = 0) {
    for (i = 0; i <là-> NF; i ++) {
      if (sscanf (là-> Trường [i], "% lf", & d) == 1) {
        dll_append (dl, new_jval_d (d));
      } Else {
        dll_append (sl, new_jval_s (strdup (là-> Trường [i])));
      }
    }
  }

  / * In các dòng có cả đôi và dây: * /

  dtmp = dl-> flink;
  STMP = SL-> flink;

  while (dtmp! = dl && STMP! = sl) {
    printf ("% - 30% 20.4lf \ n", stmp-> val.s, dtmp-> val.d);
    dtmp = dtmp-> flink;
    STMP = stmp-> flink;
  }

  / * Bây giờ in các dòng mà chỉ có dây: * /

  while (STMP! = sl) {
    printf ("% - 30% 20 \ n", stmp-> val.s, "");
    STMP = stmp-> flink;
  }

  / * Và in các dòng mà chỉ có đôi: * /
```

```
  while (dtmp! = dl) {
    printf ("% - 30% 20.4lf \ n", "", dtmp-> val.d);
    dtmp = dtmp-> flink;
  }

  exit (0);
}
```