اصول طراحی کامپایلر

درس‌نامه‌ی ۱۸

کاظم فولادی
http://kazim.fouladi.ir

ویراست اول: ۱۳۸۸
ویراست دوم: ۱۳۹۳

فهرست مطالب

۱۸ بهینه‌سازی کد میانی ۱

۱۸-۱ مقدمه ۱

۱۸-۱-۱ ضوابط تبدیل‌های بهبود کد ۲

۱۸-۲ یک برنامه‌ی نمونه: مرتب‌سازی سریع (QuickSort) ۳

۱۸-۳ بلاک‌های پایه و گراف‌های جریان ۴

۱۸-۴ منابع اصلی بهینه‌سازی ۵

۱۸-۴-۱ حذف زیرعبارت‌های مشترک ۵

گراف‌های جهت‌دار بدون دور برای تعیین زیرعبارت‌های مشترک ۶

۱۸-۴-۲ انتشار کپی ۷

۱۸-۴-۳ حذف کد مرده ۸

۱۸-۴-۴ جادادن ثابت‌ها ۸

۱۸-۵ بهینه‌سازی حلقه‌ها ۸

۱۸-۵-۱ جابجایی کد ۹

۱۸-۵-۲ کاهش دشواری ۹

۱۸-۵-۳ متغیرهای استقرایی ۹

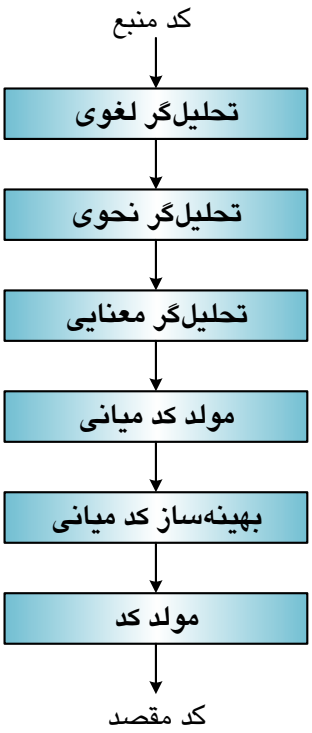
❊ تمرین ۱۱

۱۸ بهینه‌سازی کد میانی

INTERMEDIATE CODE OPTIMIZATION

۱۸-۱ مقدمه

بهینه‌سازی کد میانی، پنجمین مرحله‌ی فرایند کامپایل است.



شکل ۱۸-۱: بهینه‌سازی کد میانی به عنوان پنجمین مرحله‌ی فرایند کامپایل

مرحله‌ی بهینه‌سازی کد میانی سعی در بهبود کد میانی دارد، به منظور:

- سرعت اجرای بالاتر
- مصرف حافظه‌ی کمتر
- مصرف انرژی کمتر

شکل ۱۸-۲: مرحله‌ی بهینه‌سازی کد میانی

مثال

نمونه‌ای از بهینه‌سازی کد میانی:

کد میانی سه‌آدرسی

```
temp1 := inttofloat(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

↓

کد میانی سه‌آدرسی

```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

- هدف از بهینه‌سازی کد میانی، یافتن کدی است که زمان اجرای پایین‌تر، فضای مصرفی کم‌تر و ... داشته باشد.
- بهینه‌سازی، تضمین نمی‌کند که کد حاصل بهترین کد ممکن باشد.
- در اینجا تنها **بهینه‌سازی‌های مستقل از ماشین** در نظر گرفته می‌شود: یعنی هیچ ملاحظه‌ای در مورد خصوصیات ماشین مقصد وارد نمی‌شود.
- تکنیک‌های مورد استفاده ترکیبی از تحلیل‌های **جریان کنترل** و **جریان داده‌ها** است:
  - **تحلیل جریان کنترل (Control-Flow Analysis):**
    شناسایی حلقه‌ها درگراف جریان برنامه (حلقه‌ها معمولاً کاندیداهای خوبی برای بهبود هستند).
  - **تحلیل جریان داده‌ها (Data-Flow Analysis):**
    جمع‌آوری اطلاعاتی در مورد چگونگی استفاده از متغیرها در برنامه.

۱۸-۱-۱ ضوابط تبدیل‌های بهبود کد

بهترین تبدیل‌ها، آنهایی هستند که حداکثر منفعت را با حداقل تلاش به دست می‌دهند.

- یک تبدیل باید معنای برنامه را حفظ کند.
- یک تبدیل باید در حالت متوسط، سرعت برنامه را به یک میزان قابل اندازه‌گیری افزایش دهد.
- از بهینه‌سازی کد برای برنامه‌هایی که گاهی اجرا می‌شوند و همچنین در هنگام اشکال‌زدایی برنامه بایستی اجتناب شود.
- **تذکر:** بهبودهای اساسی با بهبود کد منبع برنامه به وجود می‌آید. برنامه‌نویس همیشه مسئول یافتن بهترین ساختمان داده‌ها و الگوریتم‌ها برای حل یک مسئله است.

۱۸-۲ یک برنامه‌ی نمونه: مرتب‌سازی سریع (QuickSort)

برای نمایش تاثیر تکنیک‌های مختلف بهینه‌سازی، از برنامه‌ی زیر استفاده می‌کنیم:

```
void quicksort(m,n)
int m,n;
{
    int i,j,v,x;
    if (n <= m) return;
    i = m-1; j = n; v = a[n];    /* fragment begins here */
    while (1) {
          do i = i+1; while (a[i]<v);
          do j = j-1; while (a[j]>v);
          if (i>=j) break;
          x = a[i]; a[i] = a[j]; a[j] =x;
    }
    x = a[i]; a[i] = a[n]; a[n] =x;    /* fragment ends here */
    quicksort(m,j); quicksort(i+1,n);
}
```

کد سه آدرسی این برنامه‌ی نمونه، به صورت زیر به دست آمده است:

```
(1)    i = m-1              (16)    t7 = 4*i
(2)    j = n                (17)    t8 = 4*j
(3)    t1 = 4*n             (18)    t9 = a[t8]
(4)    v = a[t1]            (19)    a[t7] = t9
(5)    i = i+1              (20)    t10 = 4*j
(6)    t2 = 4*i             (21)    a[t10] = x
(7)    t3 = a[t2]           (22)    goto (5)
(8)    if t3<v goto (5)     (23)    t11 = 4*i
(9)    j = j-1              (24)    x  = a[t11]
(10)   t4 = 4*j             (25)    t12 = 4*i
(11)   t5 = a[t4]           (26)    t13 = 4*n
(12)   if t5>v goto (9)     (27)    t14 = a[t13]
(13)   if i>=j goto (23)    (28)    a[t12] = t14
(14)   t6 = 4*i             (29)    t15 = 4*n
(15)   x = a[t6]            (30)    a[t15] = x
```

۱۸-۳ بلاک‌های پایه و گراف‌های جریان

مرحله‌ی بهینه‌سازی کد «مستقل از ماشین»:

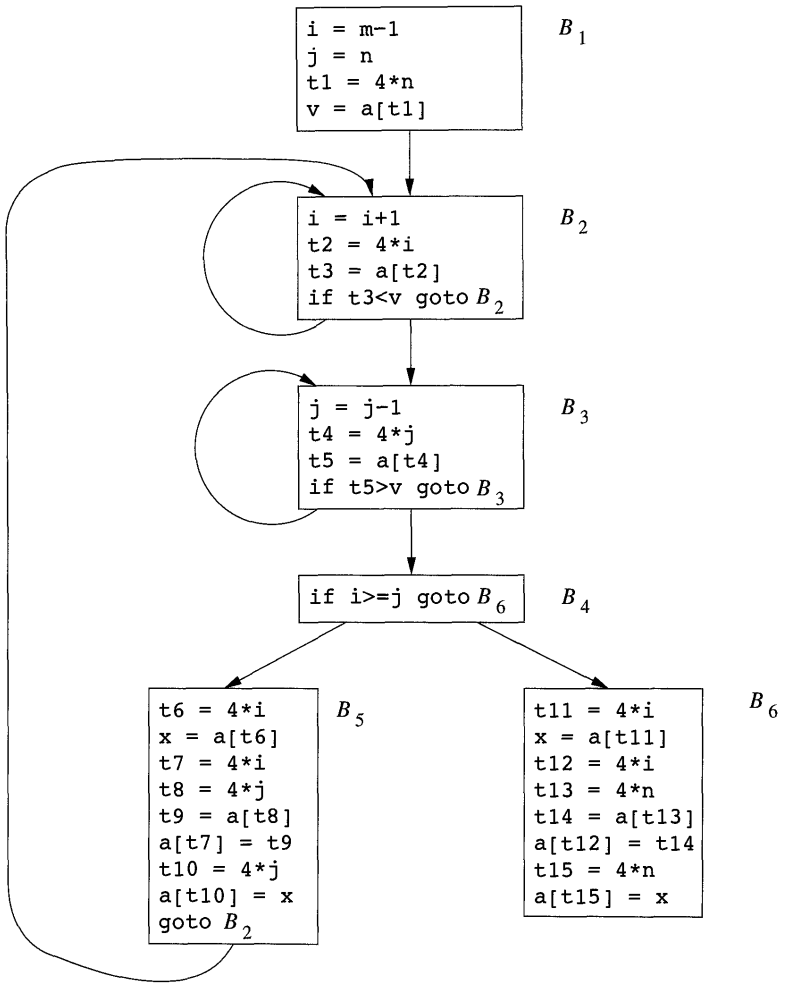
- تحلیل جریان کنترل
- تحلیل جریان داده‌ها

در طول تحلیل جریان کنترل، یک برنامه به صورت **گراف جریان (flow-graph)** نمایش داده می‌شود:

- هر گره یک **بلاک پایه (Basic Block)** است: دنباله‌ای از دستورالعمل‌های پی‌درپی که جریان کنترل از ابتدای آن آغاز می‌شود و در انتهای آن بدون پرش یا توقف، خاتمه می‌یابد.
- هر یال، یک جریان کنترل را نشان می‌دهد.

مثال

شکل زیر، گراف جریان را برای کد سه آدرسی مرتب‌سازی سریع نشان می‌دهد که در آن هر B_i یک بلاک پایه است:



۱۸-۴ منابع اصلی بهینه‌سازی

- **بهینه‌سازی محلی (local)**: تنها با دستورات موجود در یک بلاک پایه سر و کار دارد.
- **بهینه‌سازی سراسری (global)**: با کل برنامه سر و کار دارد.

یک بلاک پایه، مجموعه‌ای از عبارت‌ها را محاسبه می‌کند:
می‌توان تعدادی تبدیل را بر روی بلاک پایه اعمال کرد، بدون اینکه عبارت محاسبه شده توسط آن بلاک تغییر کند. نمونه‌ی این تبدیل‌ها عبارت است از:

۱) حذف زیرعبارت مشترک (Common Sub-expression Elimination)

۲) انتشار کپی (Copy Propagation)

۳) حذف کد مرده (Dead-Code Elimination)

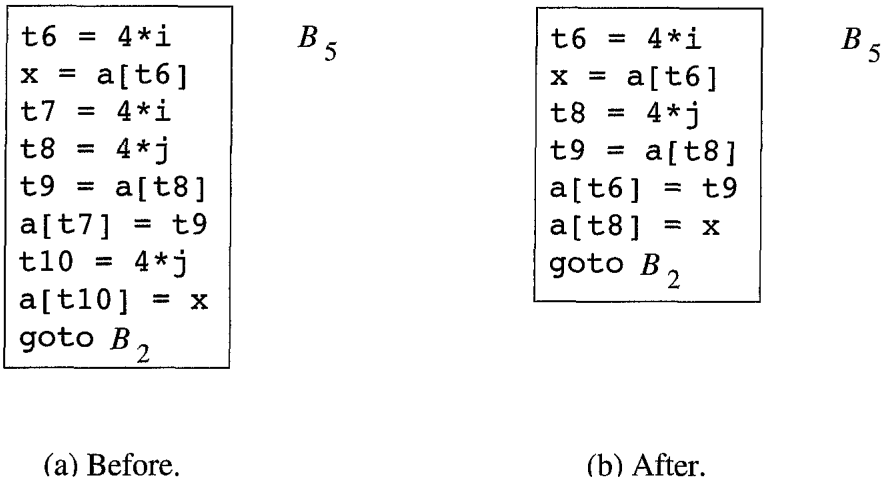
۴) جادادن ثابت (Constant Folding)

۱۸-۴-۱ حذف زیرعبارت‌های مشترک

یک وقوع عبارت E، زیرعبارت مشترک (common sub-expression) نامیده می‌شود، اگر E قبلاً محاسبه شده باشد و مقادیر متغیرهای موجود در E از محاسبه‌ی قبلی تغییر نکرده باشد.

مثال

بلاک B_5 را در نظر بگیرید. پس از حذف زیرعبارت‌های مشترک B_5 به صورت زیر تبدیل می‌شود:



(a) Before. (b) After.

پس از حذف محلی زیرعبارت مشترک، B_5 هنوز $4*i$ و $4*j$ را ارزیابی می‌کند که زیرعبارت‌های مشترک هستند.

- $4*j$ در بلاک B_3 توسط t_4 ارزیابی می‌شود. بنابراین، دستورات

$$t_8 := 4*j;\ t_9 := a[t_8];\ a[t_8] := x$$

می‌تواند با دستورات زیر جایگزین شود:

$$t_9 := a[t_4];\ a[t_4] := x$$

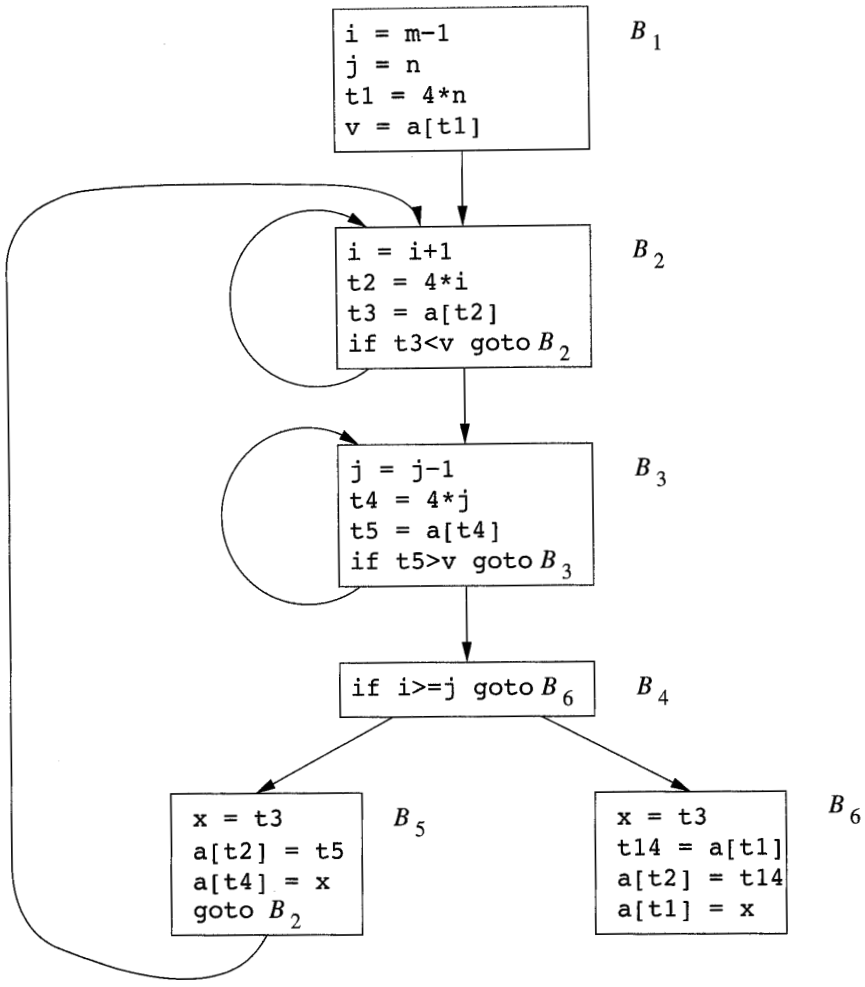
- حال، $a[t_4]$ نیز یک زیرعبارت مشترک خواهد بود که در B_3 توسط t_5 محاسبه می‌شود. بنابراین، دستورات

$$t_9 := a[t_4];\ a[t_6] := t_9$$

می‌تواند با دستور $a[t_6] := t_5$ جایگزین شود.

- به طور مشابه، مقدار x مانند مقدار نسبت داده شده به t_3 در بلاک B_3 است: t_6 می‌تواند حذف شود و با t_2 جایگزین شود.

گراف جریان حاصل از حذف زیرعبارت‌های مشترک محلی و سراسری در بلاک‌های پایه‌ی B_5 و B_6:



گراف‌های جهت‌دار بدون دور برای تعیین زیرعبارت‌های مشترک

برای تعیین زیرعبارت‌های مشترک، یک بلاک پایه را به صورت یک گراف جهت‌دار بدون دور (DAG) بازنمایی می‌کنیم تا چگونگی استفاده‌ی مجدد زیرعبارت‌ها در یک بلاک مشخص شود.

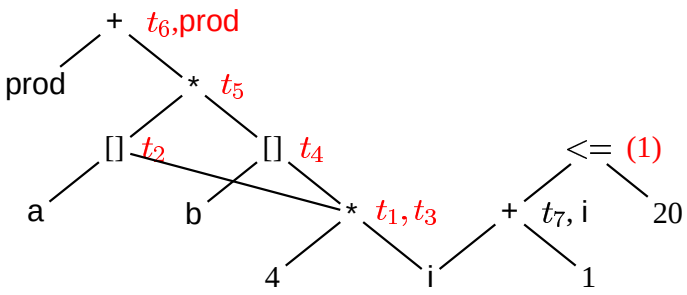
DAG برای یک بلاک پایه، دارای مشخصات زیر است:

- برگ‌ها حاوی شناسه‌های یکتا هستند (نام متغیرها یا ثوابت).
- گره‌های داخلی، حاوی یک نماد عملگر هستند.
- گره‌ها می‌توانند دارای لیستی از متغیرهای وابسته باشند تا نشان دهد آن متغیرها دارای مقداری هستند که در آن گره محاسبه می‌شود.

مثال

گراف‌های جهت‌دار بدون دور برای یک بلاک پایه

(1) $t_1 := 4 * i$
(2) $t_2 := a[t_1]$
(3) $t_3 := 4 * i$
(4) $t_4 := b[t_3]$
(5) $t_5 := t_2 * t_4$
(6) $t_6 := prod + t_5$
(7) $prod := t_6$
(8) $t_7 := i + 1$
(9) $i := t_7$
(10) if $i <= 20$ goto (1)



۱۸-۴-۲ انتشار کپی

قاعده‌ی انتشار کپی: با داشتن دستور کپی $x := y$، پس از این دستور هر جا ممکن بود به جای x از y استفاده می‌کنیم.

مثال

با اعمال انتشار کپی بر روی بلاک B_5 از حاصل مثال قبل، خواهیم داشت:

$$x := t_3$$
$$a[t_2] := t_5$$
$$a[t_4] := t_3$$
$$\text{goto} B_2$$

این تبدیل به همراه حذف کد مرده در مجموع امکان حذف انتساب $x := t_3$ را فراهم می‌کند.

۱۸-۴-۳ حذف کد مرده

- یک متغیر در یک نقطه از برنامه **زنده (live)** است، اگر مقدار آن بتواند متعاقباً استفاده شود، وگرنه **مرده (dead)** است.
- بخشی از کد **مرده** است، اگر داده‌های محاسبه شده در آن در هیچ جای دیگری استفاده نشود.
- کد مرده ممکن است حاصل تبدیل انتشار کپی باشد.
- کاربرد حذف کد مرده به همراه انتشار کپی می‌تواند مناسب باشد.

مثال

در بلاک B_5 پس از انتشار کپی، می‌توانیم ببینیم که x در هیچ جای دیگری از آن بلاک استفاده نمی‌شود. بنابراین x یک متغیر مرده است و می‌توانیم انتساب $x := t_3$ را از B_5 حذف کنیم.

$$x := t_3$$
$$a[t_2] := t_5$$
$$a[t_4] := t_3$$
$$\text{goto} B_2$$

۱۸-۴-۴ جادادن ثابت‌ها

ممکن است در زمان کامپایل بتوانیم متوجه شویم که مقدار یک عبارت (یا یک متغیر) ثابت است.

- **جادادن ثابت‌ها (Constant Folding)** تبدیلی است که یک عبارت را با یک ثابت جایگزین می‌کند.
- جادادن ثابت‌ها برای کشف کد مرده مفید است.

مثال

دستور شرطی if x goto L را در نظر می‌گیریم. اگر با جادادن ثابت‌ها متوجه شویم که x همیشه نادرست است، می‌توانیم شرط آزمون if و پرش به L را حذف کنیم.

۱۸-۵ بهینه‌سازی حلقه‌ها

اگر مقدار دستورالعمل‌های موجود در یک حلقه را کاهش بدهیم، در زمان اجرای برنامه می‌تواند بهبود حاصل شود.

برای این کار، سه تکنیک مفید وجود دارد:

- جابجایی کد (Code Motion)
- کاهش دشواری (Reduction in Strength)
- حذف متغیر استقرایی (Induction-Variable Elimination)

۱۸-۵-۱ جابجایی کد

اگر محاسبه‌ی یک عبارت **مستقل از حلقه (loop-invariant)** باشد، این تبدیل چنین محاسبه‌ای را قبل از حلقه قرار می‌دهد.

مثال

دستور while زیر را در نظر می‌گیریم:

$$\textbf{while } i <= limit - ۲ \textbf{ do}$$

عبارت $limit - ۲$ مستقل از حلقه است. تبدیل جابجایی کد به نتیجه‌ی زیر منجر می‌شود:

$$t := limit - ۲; \qquad \textbf{while } i <= t \textbf{ do}$$

۱۸-۵-۲ کاهش دشواری

منظور از کاهش دشواری، جایگزینی یک محاسبه با یک محاسبه‌ی ارزان‌تر است.

مثال

انتساب $t_۴ := ۴ * j$ را در بلاک $B_۳$ در نظر می‌گیریم.
j در هر مرتبه ۱ واحد کاهش می‌یابد، پس $t_۴ := ۴ * j - ۴$.
بنابراین می‌توانیم $t_۴ := ۴ * j$ را با $t_۴ := t_۴ - ۴$ جایگزین کنیم.

مشکل: باید پیش از ورود به بلاک $B_۳$، $t_۴$ را با $t_۴ = ۴ * j$ مقداردهی کنیم.
نتیجه: جایگزینی یک ضرب با یک تفریق، موجب افزایش سرعت کد حاصل می‌شود.

۱۸-۵-۳ متغیرهای استقرایی

متغیر x یک متغیر استقرایی (induction variable) یک حلقه است اگر هر بار که مقدار x تغییر می‌کند، به اندازه‌ی یک مقدار ثابت افزایش یا کاهش پیدا کند.

یک موقعیت متداول آن است که یک متغیر استقرایی مانند i یک آرایه را اندیس‌گذاری کند و یک متغیر استقرایی دیگر مانند t آفست واقعی برای دسترسی به آرایه باشد:

- اغلب می‌توانیم از i خلاص شویم.

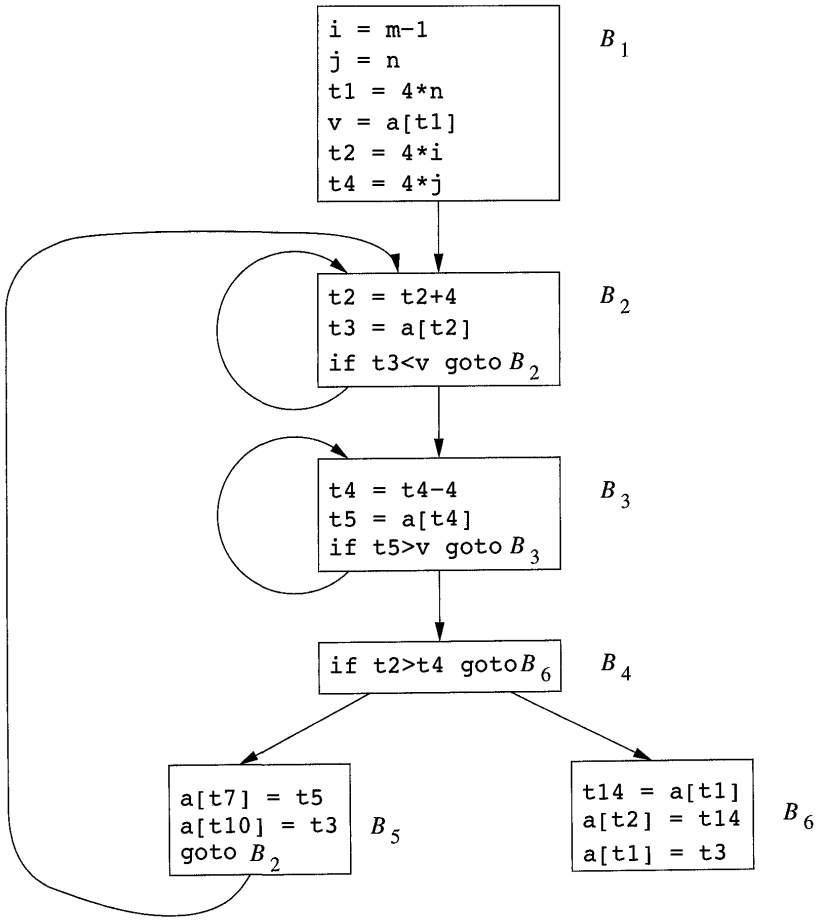
- در حالت کلی، وقتی دو یا چند متغیر استقرایی داریم، می‌توان از همه‌ی آنها بجز یکی خلاص شد.

مثال

حذف متغیرهای استقرایی: مثال

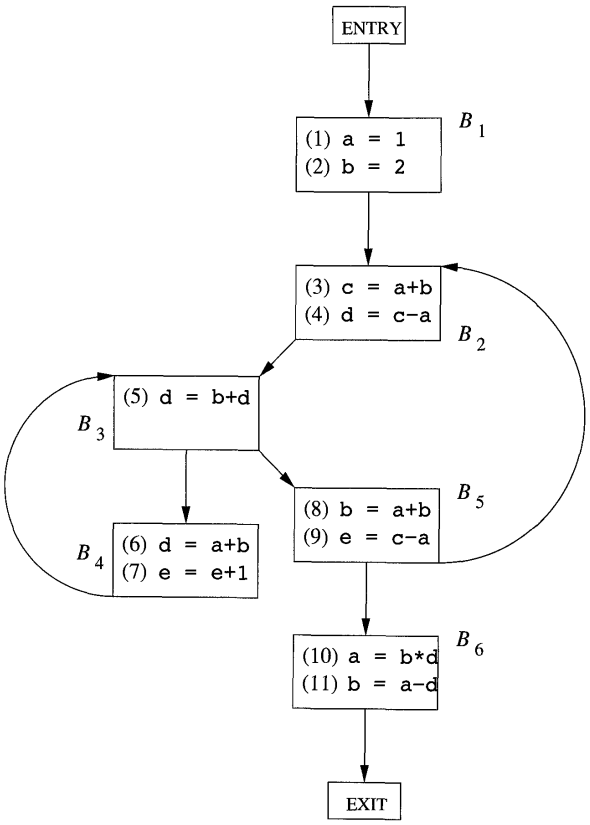
- حلقه‌ی بلاک B_3 را در نظر می‌گیریم. متغیرهای j و t_4 متغیرهای استقرایی هستند.
  همین مورد برای متغیرهای iو t_2 در بلاک B_2 صدق می‌کند.
- پس از کاهش پیچیدگی اعمال شده بر t_2و t_4، تنها مورد استفاده از i و j، برای تعیین تست در B_4 خواهد بود.
- از آنجا که $t_2 := 4 * i$ و $t_4 := 4 * j$ است، تست $t_2 > t_4$ هم‌ارز با $i > j$ است.
- پس از این جایگزینی در تست، هم i (در بلاک B_2) و هم j (در بلاک B_3)، متغیرهای مرده می‌شوند و می‌توانند حذف شوند.

گراف جریان پس از کاهش دشواری و حذف متغیرهای استقرایی به صورت زیر در می‌آید:



◄ تمرین

۱. با توجه به گراف جریان زیر،



آ) حلقه‌ها را مشخص کنید.

ب) زیرعبارت‌های مشترک محلی را مشخص کنید و حذف آنها را انجام دهید.

پ) زیرعبارت‌های مشترک سراسری را مشخص کنید و حذف آنها را انجام دهید.

ت) امکان انجام انتشار کپی را در این گراف جریان بررسی کنید و در صورت امکان آن را اعمال کنید.

ث) جایگذاری ثابت‌ها را در این گراف جریان انجام دهید.

ج) حذف کد مرده را در این گراف جریان انجام دهید.

چ) متغیرهای استقرایی را برای هر حلقه مشخص کنید.