1 الگوریتم‌های تقسیم و غلبه [1]

یکی از راه‌های حل مسئله روش تقسیم و غلبه(حل) است؛ که مبنای شمار زیادی از الگوریتم‌های سریع در حوزه‌های مختلف از جمله جستجو است. این روش بر اساس چندین بار استفاده همزمان از الگوریتم‌های بازگشتی پایه‌ریزی شده است. سه مرحله کلی این روش عبارت‌اند از:

- **تقسیم**[2]: تقسیم کردن مسئله به زیر مسئله‌هایی که اندازه کوچکتری نسبت به مساله اولیه دارند.
- **غلبه / حل**[3]: حل کردن هر مسئله به صورت بازگشتی، تا جایی که به زیر مسئله‌هایی برسیم که به صورت مستقیم قابل حل باشند.
- **متحد کردن**[4]: ترکیب کردن پاسخ زیر مسئله‌ها و ایجاد مسئله اصلی.

برخی از الگوریتم‌ها مساله را صرفا به یک مساله با اندازه کوچکتر تبدیل میکنند ولی برخی دیگر آن را به چندین زیر مساله تقسیم می‌کنند.

در هر مرحله مساله موجود یا به صورت بازگشتی حل می‌شود (حالت بازگشتی) و یا مستقیم (حالت پایه.) مساله اینکه چه زمانی یک مسئله به اندازه کافی کوچک است تا مستقیم حل شود، می‌تواند بر زمان اجرای آن تاثیر بگذارد.
در بسیاری از الگوریتم‌ها حالت‌های پایه از میان کوچکترین حالت‌های ممکن انتخاب می‌شوند زیرا که در این صورت تعداد حالت‌هایی که باید بررسی شوند کمتر شده و حل هر کدام از آن‌ها به صورت مستقیم بسیار ساده تر می‌باشد. برای مثال در جستجوی سریع * حالت پایه لیست خالی در نظر گرفته می‌شود. (و یا الگوریتم FFT، زمانی که ورودی تنها یک نمونه باشد (؟))
در حالی که در برخی از الگوریتم‌ها اگر حالت پایه طوری انتخاب شود که اندازه مساله نه خیلی کوچک و نه خیلی بزرگ باشد،سرعت و بازدهی الگوریتم بهبود می‌یابد. به این روش‌ها که قسمتی از مساله را به صورت بازگشتی و بقیه آن ا به صورت مستقیم حل می‌کنند الگوریتم‌های پیوندی [5]گفته می‌شود. این استراتژی باعث می‌شود که هزینه اضافی برای توابع بازگشتی که کار قابل‌توجهی انجام نمی‌دهند پرداخت نشود. برای مثال در پیمایش درخت دودویی اگر قبل از اینکه به یک گره وارد شویم مطمئن شویم که NULL نیست، تعداد اجرای تابع پیمایش نصف می‌شود.

---

[1] Divide and Conquer
[2] Divide
[3] Conquer
[4] Combine
[5] Hybrid algorithm

1.1 پیچیدگی زمانی و درستی روش

درستی الگوریتم‌های تقسیم و غلبه معمولاً با استفاده از استقرای ریاضی نشان داده می‌شود. همچنین برای به دست آوردن پیچیدگی زمانی آن‌ها همانند آنچه که در فصل ؟ نشان داده شد از رابطه‌های بازگشتی استفاده می‌شود.

1.2 کاهش و غلبه[6]

در روش تقسیم و غلبه هر مساله چندین زیر مساله ایجاد می‌کند در حالی که در سری دیگری از روش‌ها (که بعضاً با نام کاهش و غلبه هم شناخته می‌شوند) هر مسئله تنها به یک زیر مسئله با اندازه کوچکتر تبدیل می‌شود. از مهم‌ترین مثال‌های این الگوریتم‌ها می‌توان به جستجوی دودویی و برخی روش‌های بهینه‌سازی اشاره کرد که در هر مرحله صرفاً فضای جستجو را کوچکتر می‌کنند.

1.3 مثال‌های از روش تقسیم و غلبه

- *جستجوی دودویی (کاهش و غلبه)*
  - تقسیم : در هر مرحله فضای جستجو نصف می‌شود(لیست چپ یا راست)
  - غلبه: به صورت بازگشتی هر مساله حل می‌شود تا به لیست خالی یا کلید جستجو برسیم

- *مرتب‌سازی سریع*
  - تقسیم: با استفاده از یک محور هر لیست به دو لیست یک عناصر کوچکتر و دیگری عناصر بزرگ‌تر از محور تقسیم می‌شود.
  - غلبه: هر کدام از لیست‌ها به صورت بازگشتی حل می‌شوند.

- *مرتب‌سازی ادغامی*
  - تقسیم: هر لیست نصف می‌شود.
  - غلبه: حل به صورت بازگشتی.
  - متحد کردن: با مقایسه خانه به خانه دو لیست مرتب شده لیست اصلی ساخته می‌شود.

- *تعداد نابجایی‌های آرایه*[7]

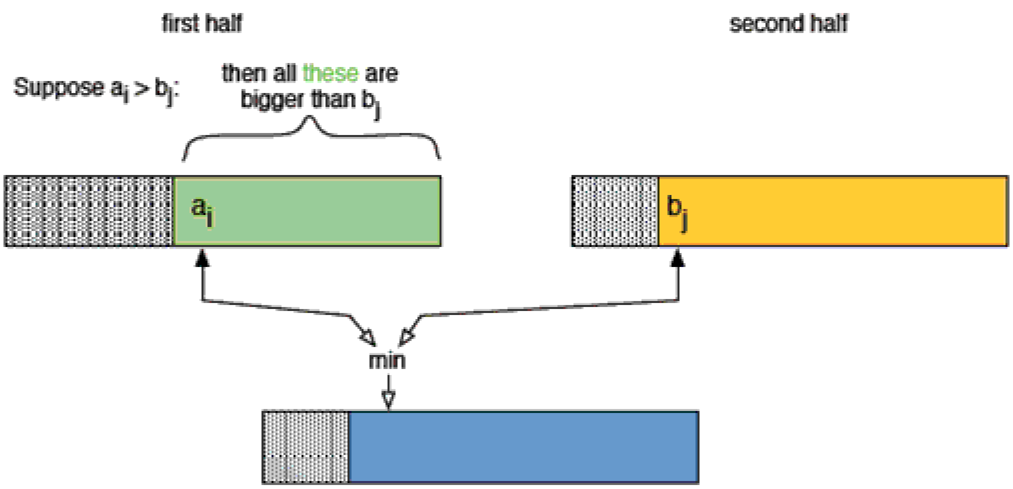
[6] Decrease and conquer
[7] Array inversions

مقدار نابجایی آرایه، معیاری است از اینکه آن آرایه چه قدر تا مرتب بودن فاصله دارد. اگر آرایه مرتب باشد این مقدار صفر و اگر برعکس مرتب شده باشد این مقدار بیشینه خواهد بود. نابه‌جایی دو عنصر این‌گونه تعریف می‌شود که عنصر بزرگ‌تر قبل از عنصر کوچکتر قرار داشته باشد یا به عبارتی:

$a[i] > a[j]$ و $i < j$

الگوریتم محاسبه این مقدار بر اساس الگوریتم مرتب‌سازی ادغامی طراحی شده است.

- تقسیم: هر لیست نصف می‌شود.
- غلبه: حل به صورت بازگشتی.
- متحد کردن: تعداد نابجایی‌ها برابر است با نابجایی‌ها در دو زیر-لیست به اضافه نابجایی‌های بین دو زیر-لیست. به این صورت که هنگام ادغام کردن[8] (مرتب‌سازی ادغامی) اگر عنصر چپ بزرگ‌تر از عنصر راست بود، آن موقع تمام عناصر باقی‌مانده در لیست چپ از عنصر راست بزرگ‌تر هستند و نابجایی محسوب می‌شوند.



- *الگوریتم ضرب ماترس استراسن*[9]

این الگوریتم دو ماتریس $n * n$ را در یکدیگر ضرب می‌کند $O(n^{\ln 7})$ که در مقایسه با روش معمول (سه حلقه تودرتو) $O(n^3)$ سریع‌تر است.

- *الگوریتم پیدا کردن نزدیک‌ترین نقاط*[10]

[8] Merge
[9] Strassen's algorithm for matrix multiplication
[10] Closest Pair of Points

هدف پیدا کردن جفت نقطه‌ای در یک مجموعه است که کمترین فاصله را از یکدیگر دارند. با استفاده از روش تقسیم و غلبه می‌توان این مساله را در زمان $O(n \log n)$ در مقایسه با زمان $O(n^2)$ (روش بدیهی مقایسه دو به دو نقاط) حل کرد.

1.4 ویژگی‌ها و برتری‌های روش تقسیم و غلبه

- روشی برای بسیاری از مسائل پیچیده
- بازدهی و سرعت الگوریتم ها
- قابلیت اجرای موازی [11] و همزمان الگوریتم
- دسترسی به حافظه
  - معمولاً اندازه حالت‌های پایه آن‌قدر کوچک است که روی حافظه نهانی جا می‌شوند که خود باعث افزایش چشمگیر سرعت اجرای الگوریتم می‌شود.
- خطای تجمعی کمتر در عملیات‌های ریاضی [12] (ناشی از نحوه نگهداری اعداد اعشاری در کامپیوتر)

[11] Parallelism
[12] http://en.wikipedia.org/wiki/Divide_and_conquer_algorithms#Roundoff_control

2 برنامه‌ریزی پویا[13]

برنامه‌ریزی پویا همانند روش تقسیم و غلبه مسائل را با پیدا کردن پاسخ به زیر مسئله‌ها حل می‌کند. روش تقسیم و غلبه مسائل را با تقسیم آن‌ها به زیر-مساله‌هایی جدای از هم تقسیم می‌کند، در حالی که روش برنامه‌ریزی پویا در حین حل مساله بارها و بارها زیر-مسائل یکسانی را حل می‌کند. به عبارتی دیگر زیر-مسائل برنامه‌ریزی پویا همپوشانی دانند. به همین دلیل یک الگوریتم برنامه‌ریزی پویا پس از حل هر زیر مساله پاسخ آن را در حافظه ذخیره می‌کند تا در دفعات بعدی، دوباره آن را حل نکند و صرفا با زمان دسترسی ثابت، آن را از حافظه بخواند.

روش برنامه‌ریزی پویا اغلب در حل مسائل بهینه‌سازی استفاده می‌شود. این گونه مسائل لزوماً پاسخ یکتایی ندارند و هدف پیدا کردن مقدار بهینه است. در طراحی الگوریتم برنامه‌ریزی پویا برای این گونه مسائل معمولاً ۴ مرحله زیر طی می‌شوند:

1. مشخص کردن ساختار جواب بهینه
2. تعریف بازگشتی پاسخ بهینه
3. محاسبه مقدار پاسخ بهینه
4. ایجاد پاسخ بهینه با استفاده از مقادیر محاسبه شده

2.1 ویژگی‌های پاسخ بهینه

مسائلی که با استفاده از برنامه‌ریزی پویا حل می‌شوند معمولاً دو ویژگی زیر را دارند:

2.1.1 زیر مسائل همپوشان

همان‌طور که ذکر شد در برنامه‌ریزی پویا وقتی کار دارد که زیر مسئله‌ها چندین بار استفاده شوند و اگر در حین تقسیم همیشه زیر مسئله‌های جدید به وجود آید این روش کاربرد نخواهد داشت. برای مثال جستجوی دودویی زیر مسائل مشترک ندارد و ذخیره پاسخ‌های آن بی‌فایده است؛ در حالی که برای محاسبه اعداد سری فیبوناچی در روش بازگشتی معمولی بارها و بارها زیر مسائل تکراری حل می‌شوند.

---

[13] Dynamic Programming

منظور برنامه‌نویسی کامپیوتری نیست و درواقع یک روش ذخیره جدول است

```
                   fib(5)
                 /        \
           fib(4)              fib(3)
          /      \            /      \
     fib(3)      fib(2)      fib(2)    fib(1)
    /     \      /    \      /    \
 fib(2)  fib(1) fib(1) fib(0) fib(1) fib(0)
 /    \
fib(1) fib(0)
```

درخت فراخوانی تابع فیبوناچی برای ششمین عدد

دو روش کلی برای ذخیره‌سازی پاسخ‌های زیر مسائل وجود دارد

2.1.1.1 مموایز کردن[14](/ روش نت برداری / روش به خاطر سپاری)

در این روش در هر بار اجرای تابع، ابتدا بررسی می‌شود که آیا مقدار این تابع قبلاً محاسبه شده است یا نه و اگر نشده باشد محاسبه انجام می‌شود، در غیر این صورت مقدار ذخیره شده بازگردانده خواهد شد.

```
int fib(int n)
{
   if(lookup[n] == NIL)
   {
    if ( n <= 1 )
      lookup[n] = n;
    else
      lookup[n] = fib(n-1) + fib(n-2);
   }

   return lookup[n];
}
```

[14] Memoization

2.1.1.2 ذخیره‌سازی جدولی[۱۵]

در این روش یک جدول برای ذخیره‌سازی تمام پاسخ‌ها در نظر گرفته می‌شود و سپس جدول از پایین(ساده‌ترین / کوچک‌ترین حالت) به بالا [۱۶]پر می‌شود. در این روش همه خانه‌های جدول پر خواهند شد در صورتی که روش مموایز کردن خانه‌ها را برحسب نیاز پر می‌کند.

```
int fib(int n)

{
  int f[n+1];
  int i;
  f[0] = 0;    f[1] = 1;
  for (i = 2; i <= n; i++)
      f[i] = f[i-1] + f[i-2];

  return f[n];
}
```

2.1.2 اصل بهینگی (زیر ساختار بهینه)

یک مسئله ساختار بهینه دارد اگر پاسخ بهینه مسئله ، خود شامل پاسخ‌های بهینه زیر مسائل آن باشد.

2.2 مثال‌هایی از برنامه‌ریزی پویا

- الگوریتم‌های کوتاه‌ترین مسیر که در فصل ؟ بررسی شدند
    - بلمن-فورد
    - دایسترا
    - فلوید-وارشال
- پیدا کردن زیرمجموعه متوالی در آرایه با بیشترین مجموع
- مسئله پول خرد [۱۷]

---

[15] Tabular

[16] Bottom Up

[17] Coin change problem, change making problem

3 الگوریتم های حریصانه[18]

در حل بسیاری از مسائل بهینه سازی در هر مرحله با توجه به اطلاعات موجود تصمیماتی گرفته می‌شود و از بین گزینه های ممکن یکی انتخاب می‌شود. در اکثر این مسائل نیازی به استفاده از روش برنامه‌ریزی پویا نیست و الگوریتم های حریصانه به تنهایی کافی و موثر هستند.

الگوریتم‌های حریصانه در هر مرحله گزینه‌ای را انتخاب می‌کنند که در آن لحظه بهترین است به این امید که انتخاب این بهینه‌های محلی منجر به پیدا کردن پاسخ بهینه کلی خواهد شد.

به طور کلی الگوریتم های حریصانه شامل ۵ جزء زیر هستند:

- مجموعه گزینه ها[19]، که پاسخ بهینه با استفاده از آن‌ها ساخته می‌شود.
- تابع انتخاب، که بهترین گزینه برای اضافه کردن به جواب را انتخاب می‌کند.
- تابع امکان،که مشخص میکند آیا یک گزینه می‌تواند برای بدست آوردن جواب استفاده شود.
- تابع هدف، که مقداری را به هر جواب (کامل یا ناکامل) نسبت می‌دهد.
- تابع پاسخ،که مشخص خواهد کرد که چه زمانی به جواب رسیده‌ایم.

الگوریتم های حریصانه معمولاً <<کوتاه بین>> و <<غیر قابل نجات>> دسته بندی می‌شوند.

3.1 مثال ها:

- الگوریتم های پیدا کردن درخت پوشای کمینه
  - کروزکال
  - پریم
- الگوریتم فشرده سازی هافمن
- الگوریتم های کوتاهترین مسیر دایسترا
- پول خرد

یک از روش های حل مساله پول خرد برای پیدا کردن حداقل سکه مورد نیاز یک روش حریصانه است به این ترتیب که در هر مرحله بزرگترین سکه ممکن با مقدار کمتر از مقدار مساله را انتخاب می‌کند.

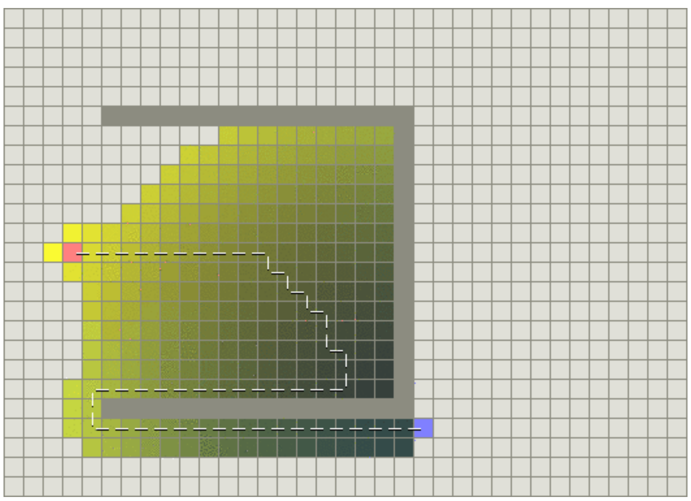
---

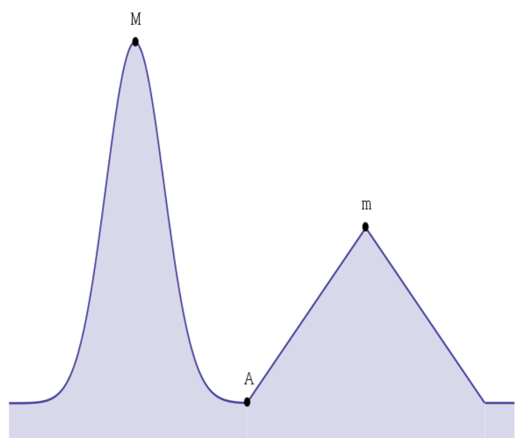
[18] Greedy Algorithms
[19] Candidates

3.2 پاسخ اشتباه

در برخی موارد الگوریتم های حریصانه صرفا یکی از پاسخ های بهینه محلی را می‌یابند و آن را به عنوان جواب نهایی معرفی میکنند. برای نمونه در مساله پول خرد الگوریتم ذکر شده نمیتواند با سکه های {۲۵، ۱۰، ۴} مقدار ۴۱ را بسازد (بعد از استفاده از یک سکه ۲۵ تایی و یک سکه ۱۰ تایی، معلوم میشود که نمیتوان مقدار ۶ را با سکه های ۴ تایی ساخت)؛ در حالی که یک الگوریتم پیچیده تر میتواند با یک سکه ۲۵ تایی و ۴ سکه ۴ تایی این مقدار را بسازد.

به عنوان مثالی دیگر می‌توان به الگوریتم مسیریابی A star که تابع اکتشافی[20] آن حریصانه است اشاره کرد





[20] Heuristic