فصل ۳

طراحی الگوریتم با استقرا

برای حل مسایلی که ماهیت بازگشتی دارند، استقرا یک روش بسیار قوی است. به کمک استقرا می‌توان برای چنین مسایلی الگوریتم مناسب طراحی کرد، درستی الگوریتم را اثبات و آن را تحلیل کرد. در این جا برای چند مسئله‌ی این مرحله‌ها را نشان می‌دهیم

۱.۳ کُد گری

می‌خواهیم به n شئ کدهای متمایزی اختصاص دهیم. ساده‌ترین روش برای این کار آن است که به آنها اعداد ۱ تا n اختصاص دهیم و با توجه به آن که تمام کارهای ما در سیستم دودویی انجام می‌گیرد این n عدد را به صورت دودویی نمایش دهیم. بدیهی است که در این حالت تعداد بیت‌ها برای هر کد برابر است با $\lceil \log_2 n \rceil$ که به‌ترین حالت است.

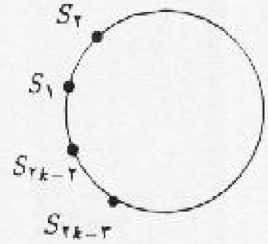
ولی به علت محدودیت‌های سخت‌افزاری در شمارنده‌ها می‌خواهیم که کدهای متوالی فقط در یک بیت اختلاف داشته باشد. یکی از این دلایل این محدودیت جلوگیری از ایجاد Hazard در مدارهای منطقی‌ای است که این کدها را پشت سر هم می‌شمارد.

بنابراین مسئله که می‌خواهیم حل کنیم این است: **چگونه می‌توان به n شئ کد دودویی با تعداد بیت کمینه (یعنی $\lceil \log_2 n \rceil$ بیت) اختصاص داد تا کدهای متوالی فقط در یک بیت اختلاف داشته باشند؟** توجه کنید که این توالی برای کدها «دورانی» است؛ یعنی هر کد دقیقاً دو کد قبل و بعد دارد. به چنین کدی «کد گری[1]» می‌گویند.

کدهای گری دو نوع هستند؛ بسته و باز.

کد گریِ بسته[2]: هرگاه هر کد با هرکدام از دو کدِ بعدی و قبلی‌اش دقیقاً در یک بیت اختلاف داشته باشد. بنابراین کدها تشکیل یک دور می‌دهند. مثلاً، $\{0000 \to 0010 \to 0011 \to 0111 \to 0110 \to 1110 \to 1111 \to 1011 \to 1010 \to 1000\}$ یک کد بسته برای ۱۰ عنصر است. توجه کنید که ۰۰۰۰ و ۱۰۰۰ هم در یک بیت با هم اختلاف دارند.

Gray code[1]
close Gray code[2]



شکل ۱.۳: نمایش کد گری

کد گری باز[۳]: همان کد بسته ست مگر بین تنها دو کد متوالی که فقط اختلاف آن دو بیش از یک بیت است.
مثلاً، $\{۰۰۰۰ \to ۰۰۱۰ \to ۰۰۱۱ \to ۰۱۱۱ \to ۰۱۱۰ \to ۱۱۱۰ \to ۱۱۱۱ \to ۱۰۱۱ \to ۱۰۱۰\}$
یک کد باز برای ۹ عنصر است.

حالت‌های ساده‌ی زیر را در نظر بگیرید:

۱. برای $n = ۲$ کد بسته‌ی بهینه داریم: $\{۰ \to ۱\}$

۲. برای $n = ۴$ نیز کد بسته‌ی بهینه داریم : $\{۰۰ \to ۰۱ \to ۱۱ \to ۱۰\}$

۳. برای $n = ۳$ کد بسته نداریم ولی کد باز بهینه داریم: $\{۰۰ \to ۰۱ \to ۱۱\}$

لم ۱. **ثابت کنید که برای تعداد زوجی عناصر ($n = ۲k$) می‌توان کد گری بسته ایجاد کرد.**

اثبات. این لم را به کمک استقرا به صورت «سازنده»[۴] اثبات می‌کنیم؛ **یعنی هم اثبات می‌کنیم که این کار شدنی است و هم نشان می‌دهیم که چگونه و با چه الگوریتمی می‌توان کدها را تولید کرد.**

پایه‌ی استقرا: $n = ۲$ ($k = ۱$) که بدیهی است.

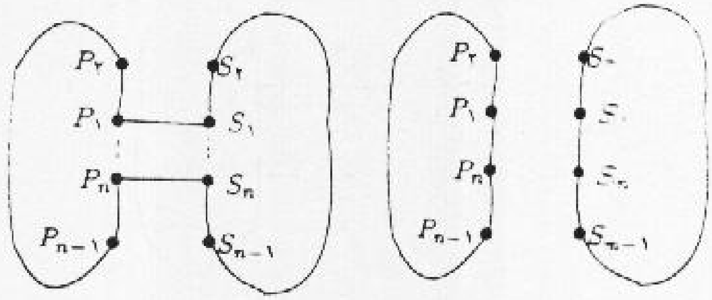
فرض استقرا: برای $n = ۲k - ۲$ کد بسته وجود دارد.

حکم استقرا: برای $n = ۲k$ هم کد بسته داریم.

بنا به فرض استقرا برای $n = ۲K - ۲$ کد بسته‌ی $\{S_۱, S_۲, \cdots, S_{۲k-۲}\}$ **را داریم که می‌توان آن را به‌صورت دور** بسته‌ای مانند شکل ۱.۳ نشان داد. رابطه‌ی بین دو کد متمایز را که **در یک بیت با هم اختلاف دارند را با خط تیره** نشان می‌دهیم (شکل ۱.۳).

ما یک بیت صفر به همه‌ی کدهای موجود اضافه کنیم؛ **باز هم رابطه‌ی فوق برقرار است. کدهای $S_{۲k-۱}$ و $S_{۲k}$ را به این صورت اضافه می کنیم: کد $S_{۲k}$ همان کد $S_۱$ است، با این اختلاف که بیت صفر اضافه شده به $S_۱$** یک باشد. کد $S_{۲k-۱}$ همان کد $S_{۲k-۲}$ **باشد با این اختلاف که بیت صفر اضافه شده به $S_{۲k-۲}$ یک باشد.** با این

---
[۳] Open Gray Code
[۴] constructive



شکل ۲.۳: ساخت کد گری به طول دو برابر

ترتیب، S_1 و S_{2k}، S_{2k-2} و S_{2k-1}، S_{2k-1} با S_{2k} فقط در یک بیت اختلاف دارند. بنابراین کد حاصل برای $n = 2k$ هم بسته است. □

توجه: در اثبات فوق الگوریتمی ارایه دادیم که در آن، برای اضافه کردن هر دو عنصر به مجموعه‌ی عناصر یک بیت به کدهای قبلی اضافه شد. بنابراین با این الگوریتم، برای $n = 2k$ عنصر کد بسته‌ای با $\frac{n}{2}$ بیت ایجاد می‌کنیم که بهینه نیست.

لم ۲. برای $n = 2^k$ عنصر می‌توان کد گری بسته‌ی k بیتی ایجاد نمود.

اثبات. اثبات به کمک استقرا.

پایه: برای $n = 2$ بدیهی است.

فرض: برای $n = 2^{k-1}$ کد بسته‌ی $k - 1$ بیتی وجود دارد.

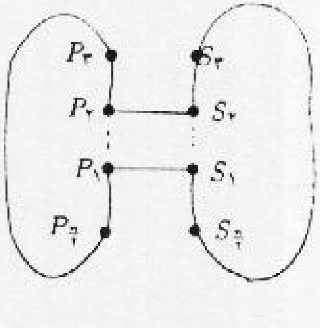
حکم: برای $n = 2^k$ کد بسته‌ی k بیتی وجود دارد.

مجموعه‌ی کدهای بسته‌ی $k - 1$ بیتی فرض استقرا (برای $n = 2^{k-1}$ عنصر) را $S = \{S_1, \cdots S_n\}$ می‌نامیم. فرض کنید $F = \{P_1, \cdots P_n\}$ یک کد دیگر مشابه‌ی S است (کدهای P_i و S_i یکسان هستند.) این دو کد بسته را به صورت دور می‌توان نشان داد (شکل ۲.۳).

به دور S یک بیتِ ۱ و به دور P یک بیتِ ۰ (صفر) اضافه می‌کنیم تا کدهای $S' = \{S'_1, \cdots S'_n\}$ و $S' = \{P'_1, \cdots P'_n\}$ ایجاد شوند. بدیهی‌است که مجموعه‌ی $\{S'_1, \cdots S'_n, P'_n, P'_{n-1}, \cdots P'_1\}$ یک کد بسته و بهینه‌ی k بیتی برای $n = 2^k$ عنصر است. □

لم ۳. برای $n = 2k + 1$ عنصر، کد گری بسته وجود ندارد.

اثبات. فرض کنید که برای این تعداد عنصر یک کد گری بسته داشته باشیم. در این صورت، اگر از یک کد شروع کنیم در یک دور زدن باید به همان کد برسیم. با توجه به آن که از یک کد به کد بعدی فقط یک بیت تغییر می‌کند، باید تعداد تغییرات این بیت‌ها زوج باشد تا بتوان به همان کد اولیه برسیم. یعنی اگر یک بیت صفر به یک



شکل ۳.۳: ساخت کد گری

تبدیل شود باید در جای دیگر همان بیت یک دوباره به‌صفر تبدیل شود. ولی این کار با توجه به تعداد فرد عناصر ممکن نیست. □

قضیه ۴. برای n عدد می‌توان کد گری $\lceil \log_2 n \rceil$ بیتی ایجاد کرد. اگر n زوج باشد این کد بسته و اگر n فرد باشد باز است.

اثبات. با استقرا اثبات می‌کنیم.

حالت اول: فرض می‌کنیم تعداد عناصر زوج است ($n = 2k$)

$\frac{n}{2} = k$ زوج است یا فرد. طبق فرض استقرا، اگر k زوج باشد، برای k عنصر کد گری بسته، و اگر فرد باشد، کد گری باز، هر دو به طول $\lceil \log_2 k \rceil$ بیت وجود دارد. اگر k زوج باشد، درست مانند اثبات لم ۲ می‌توان برای n عنصر کد گری بسته‌ی $\lceil \log_2 n \rceil = 1 + \lceil \log_2 \frac{n}{2} \rceil$ بیتی ایجاد کرد. اگر k فرد باشد، دو کد باز و مشابه‌ی $S = \{S_1, \cdots S_k\}$ و $P = \{P_1, \cdots P_k\}$ را برای k عنصر در نظر بگیرید، و فرض کنید که تنها کدهای S_1 و S_2 (و نیز P_1 و P_2) با هم در بیش از یک بیت اختلاف دارند. این دو کد را می‌توان به صورت دو دور «باز» مانند شکل ۳.۳ نشان داد.

به دور S یک بیتِ ۱ و به دور P یک بیتِ ۰ (صفر) اضافه می‌کنیم تا کدهای $S' = \{S'_1, \cdots S'_n\}$ و $S' = \{P'_1, \cdots P'_n\}$ ایجاد شوند. بدیهی است که مجموعه‌ی $\{S'_1, \cdots S'_n, P'_n, P'_{n-1}, \cdots P'_1\}$ یک کد بسته است و تعداد بیت‌های آن برابر است با

$$\lceil \log_2 \frac{n}{2} \rceil + 1 = \lceil \log_2 n \rceil$$

حالت دوم: فرض می‌کنیم تعداد عناصر فرد است ($n = 2k + 1$)

برای $(2k + 2)$ عنصر یک کد بسته‌ی $\lceil \log_2 2k + 2 \rceil$ بیتی ایجاد می‌کنیم. کافی است یک عنصر دلخواه را حذف کنیم؛ کد بسته به کد باز و با طول بهینه تبدیل می‌شود، چون $\lceil \log_2 2k + 1 \rceil = \lceil \log_2 2k + 2 \rceil$

□

۲.۳ رابطه‌ی مستقل از حلقه برای اثبات درستی الگوریتم

استقرا، یک راه خوب برای اثبات الگوریتم‌هاست. برای مثال اگر یک الگوریتم حاوی یک حلقه باشد، می‌توانیم بااستقرا روی متغیر حلقه، درستی نتیجه رااثبات کنیم. دراین صورت رابطه‌ی استقرایی برپایه‌ی متغیرهای حلقه به گونه‌ی مستقلی از حلقه(Loop Invariant) بیان می‌گردد. این رابطه را اگر برای همه‌ی مقادیر حلقه اثبات کنیم، در واقع درستی آن را اثبات کرده‌ایم.

جهت روشن‌تر شدن مطلب به مثال زیر توجه کنید:

```
Convert-to-Binary(n)
    ▷ Input: n (A positive number)
    ▷ Outpurt: b (Array of bits)
    t ← n
    k ← 0
    while t > 0
        do k ← k + 1
           b[k] ← t mod 2
           t ← t div 2
```

الگوریتم فوق عمل تبدیل عدد n را به رشته‌ی دودویی معادل را بر عهده دارد.

برای اثبات درستی این الگوریتم، بر روی k تعداد دفعاتی که حلقه تکرارشده است، استقرا انجام می‌دهیم. اما قبل از آن باید رابطه‌ی مستقل از حلقه بیابیم که وابسته به متغیرهای حلقه (از جمله k) باشد و استقرا بر روی آن برای k های متفاوت انجام دهیم. به رابطه‌ی ذکر شده invariant می‌گوییم.

در این الگوریتم رابطه‌ی استقرایی را به صورت زیر حدس می‌زنیم:

$$n = t۲^k + m$$

که در آن m عددی است که آرایه‌ی b نمایش می‌دهد.

اثبات.

پایه‌ی استقرا : اگر $k = ۰$ آن‌گاه $m = ۰$ پس $n = t۲^۰ + ۰$

فرض استقرا : اگر $k = i$ آن‌گاه $n = t۲^i + m$

حکم استقرا: اگر $k = i + ۱$ آن‌گاه $n = t'۲^{i+۱} + m'$ که t' و m' با رابطه‌ی زیر بیان می گردند:

$$t' = \lfloor \frac{t}{۲} \rfloor \qquad m' = m + (t \bmod ۲)۲^i$$

با توجه به روابط فوق و فرض استقرا داریم:

$$n = \lfloor \frac{t}{۲} \rfloor ۲^{i+۱} + m + (t \bmod ۲)۲^i = ۲^i(۲\lfloor \frac{t}{۲} \rfloor + (t \bmod ۲)) + m$$

از طرفی

$$۲\lfloor\frac{t}{۲}\rfloor + (t \bmod ۲) = t$$

پس داریم:

$$n = t۲^{i} + m$$

واین رابطه بنا به فرض استقرا صحیح است.

□

۳.۳ محاسبه‌ی مقدار یک چندجمله‌ای

هدف محاسبه‌ی مقدار یک چندجمله‌ای $P_n(x)$ از درجه‌ی n بر حسب x است.

$$P_n(x) = a_n x^n + a_{n-۱}x^{n-۱} + \cdots + a_۱ x + a_۰$$

ورودی: x، n و $a_n, \ldots, a_۰$

خروجی: مقدار $P_n(x)$

راه حل اول : چندجمله‌ای را به صورت بازگشتی تعریف می‌کنیم: $P_n(x) = P_{n-۱}(x) + a_n x^n$ و مطابق آن الگوریتمی بازگشتی برای حل مسئله می‌نویسیم. در این صورت، هزینه‌ی الگوریتم برابر $n-۱$ عمل جمع خواهد بود به اضافه‌ی تعداد عملیات‌های ضرب که برابر است با:

$$T(n) = T(n-۱) + n = \frac{n(n+۱)}{۲}$$

راه حل دوم : (الگوریتم هورنر[۵])
تعریف را به این صورت تغییر می‌دهیم:

$$P'_{n-۱}(x) = a_n x^{n-۱} + a_{n-۱}x^{n-۲} + \cdots + a_۲ x + a_۱$$

$$P_n(x) = xP'_{n-۱}(x) + a_۰$$

بنابراین:

$$T(n) = T(n-۱) + ۱ = n$$

[۵] Horner's a,gprothm

```
HORNER()
1  p ← 0
2  for i ← n downto 0
3       do p ← p * x + a[i]
```

۴.۳ زیرگراف القایی بیشینه

هدف به دست آوردن بزرگترین زیرگرافی است که درجه‌ی هر رأس آن حداقل k باشد. به این زیرگراف زیرگراف القایی بیشینه[۱] می‌گوییم.

یک مثال کاربردی برای این مسئله: عده‌ای را می‌خواهیم به یک مهمانی دعوت کنیم. هریک از این افراد تعدادی از دیگران را می‌شناسد و رابطه‌ی «شناسایی» رابطه‌ای دوطرفه است. می‌خواهیم از بین این افراد، می‌خواهیم بیشترین تعدادی را دعوت کنیم که هرکدامشان لااقل k نفر دیگر را بشناسد. مسئله به صورت گراف مدل می‌شود و هدف به دست آوردن زیرگراف القایی است با درجه‌ی حداقل k.

به عبارت دیگر، در گراف $G = (V, E)$ زیرمجموعه‌ی $S \subset V$ را با حداکثر تعداد رأس‌ها می‌خواهیم پیدا کنیم که زیر گراف القایی آن دارای رأس‌هایی با درجه‌ی حداقل k باشد.

روش پیاده‌سازی: استفاده از ماتریس مجاورت

حل: اگر درجه‌ی همه‌ی گره‌ها از k بیشتر باشد مسئله حل شده است، در غیر این صورت گره‌ای مانند v وجود دارد به طوری که $\text{Degree}(v) < k$. گره‌ی مورد نظر و یال‌های مربوط به آن را حذف می کنیم. این کار را ادامه می‌دهیم تا وقتی که درجه‌ی هر گره‌ی باقی‌مانده حداقل k باشد.

۵.۳ نگاشت یک‌به‌یک

مجموعه‌ی A و تابع $f : A \to A$ داده‌شده است. می‌خواهیم بزرگترین زیرمجموعه‌ی $S \subset A$ را پیدا کنیم که f بر روی آن زیرمجموعه یک‌به‌یک باشد، یعنی $f : S \mapsto S$.

تابع f بر روی A را می‌توان به صورت یک گراف سودار نشان داد. شکل ۴.۳ یک تابع را بر روی $A = \{۱, ۲, ۳, ۴, ۵, ۶, ۷\}$ نشان می‌دهد.

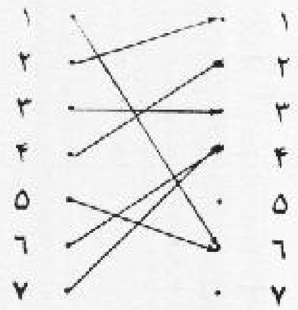
هدف پیدا کردن بزرگ‌ترین زیرگرافی است که درجه‌ی ورودی و خروجی هر گره در آن ۱ باشد. این مسئله را می‌توان با استقرا حل کرد.

اگر درجه‌ی ورودی گره‌های این گراف را به دست آوریم، باید گره‌ای به نام $v \in A$ با درجه‌ی ورودی صفر وجود داشته باشد، در غیر این صورت $S = A$ جواب است (چرا؟) v نمی‌تواند عضوی از جواب باشد، پس v و یال‌هایی که از آن خارج می‌شوند را حذف می‌کنیم و در گراف حاصل مسئله را به صورت استقرا حل می‌کنیم.

برنامه‌ی زیر این کار را انجام می‌دهد.

---

[۱] Maximal Induced Subgraph



شکل ۴.۳: مثالی از نگاشت یک‌به‌یک

```
ONE-TO-ONE-MAP(A)
    ▷ Input: A is an array [1..n] with values of 1..n
    ▷ Output: S a subset of A = {1..n}
S ← A
for j ← 1 to n
    do C[j] ← 0            C[j] is the indegree of j
for j ← 1 to n
    do Inc (C[F[j]])
for j ← 1 to n
    do if C[j] = 0
          then put j in a queue
while queue is not empty
      do remove i from queue
          S ← S − {i}
          Dec(C[F[i]])
          if C[F[i]] = 0
             then put F[i] in the queue
```

۶.۳ زیردنباله‌ی متوالی بیشینه

دنباله‌ی n عدد (مثبت و منفی) $P_n = a_1, a_2, \ldots, a_n$ داده شده‌اند. می‌خواهیم زیردنباله‌ای متوالی از این دنباله با بیش‌ترین مجموع (Maximum Consecutive Subsequence) را پیدا کنیم. یعنی i و j ای را پیدا کنیم که مقدار $\sum_{k=i}^{j} a_k$ بیشینه شود.

بدیهی است که الگوریتم $O(n^2)$ برای این مسئله وجود دارد. هدف پیدا کردن الگوریتم خطی است.

اگر به‌صورت استقرایی به مسئله نگاه کنیم، نمی‌توان از جواب مسئله برای $P_{i-1} = a_1, a_2, \ldots, a_{i-1}$، و با کاری به‌اندازه‌ی $O(1)$ به حل آن برای $P_i = a_1, a_2, \ldots, a_i$ رسید، چرا که ممکن است جواب P_i شامل a_i باشد

و نتوان آن‌را از جواب زیرمسئله‌ی P_{i-1} به‌دست آورد، چون جواب P_{i-1} ممکن است شامل a_{i-1} نباشد. برای رفع این مشکل لازم است که استقرا را قوی‌تر کنیم و برای زیر مسئله‌ی P_i دو جواب به‌درست بیاوریم؛ یکی به‌ترین جواب، و دیگری جوابی که شامل a_i هم باشد که به آن جواب «پسوندی» می‌گوییم. این دو جواب را با $GlobalMax$، و $SuffixMax$ هر جواب شامل مقدار بیشینه و اندیس‌های شروع و پایان زیردنباله است. برای سادگی، در این‌جا فقط مقدار بیشینه در نظر گرفته می‌شود. رویه‌ای که این مسئله را حل می‌کند، به‌صورت زیر است.

```
Maximum-Consequtive-Subsequence(A)
    ▷ Input: A an array [1..n] with positive and negative values
    ▷ Output: the maximum value of ∑_{k=i}^{j} A_i for some i and j
GlobalMax ← 0
SuffixMax ← 0
for i ← 1 to Length(A)
    do if A[i] + SuffixMax > GlobalMax
          then SuffixMax ← SuffixMax + A[i]
               GlobalMax ← SuffixMax
          else if A[i] + SuffixMax > 0
                  then SuffixMax ← A[i] + SuffixMax
                  else SuffixMax ← 0
```

بدیهی است که الگوریتم فوق خطی است.

۷.۳ مسئله‌ی «ستاره‌ی مشهور»

می‌خواهیم با حداقل تعداد پرسش از نفر n فردی که ویژگی‌های یک «ستاره‌ی مشهور[۲]» را دارد، در صورت وجود، پید کنیم. یک نفر ستاره است اگر بقیه او را از قبل بشناسند و او با آن‌ها آشنا نباشد. ما مجاز هستیم از a بپرسیم که آیا b را می‌شناسد؟ این یک پرسش است. توجه کنید در این مسئله رابطه‌ی شناختن یک رابطه‌ی یک‌طرفه است.

چون $n(n-۱)/۲$ گروه دو نفری داریم پس اگر پرسش‌ها به طور دل‌خواه باشند، در بدترین حالت نیاز به $n(n-۱)$ پرسش داریم.

راه حل اول: فرض کنیم که ما می‌توانیم ستاره را بین $n-۱$ نفر اول توسط استقرا به دست آوریم. چون حداکثر یک ستاره می‌توانیم داشته باشیم، سه حالت ممکن است اتفاق بیفتد:

۱. ستاره بین $n-۱$ نفر اول است.

۲. نفر nام ستاره است.

۳. ستاره نداریم.

---

[۲] celebrity

در حالت اول فقط باید بررسی کنیم که نفر nام ستاره را می‌شناسد و ستاره او را نمی‌شناسد. در دو حالت بعد، حداکثر به $۲(n-۱)$ پرسش نیاز داریم؛ چرا که باید روشن کنیم که آیا هریک از $n-۱$ نفر بقیه نفر nام می‌شناسد و نفر nام او را نمی‌شناسد. بنابراین، جمع کل پرسش‌ها $n(n-۱)$ است که در بدترین حالت هم به آن رسیدیم.

راه‌حل بهتر: به روش حذفی عمل می‌کنیم، در هر پرسش یک نفر را از مجموعه حذف کنیم و با استفاده از استقرا راه حل را دنبال می‌کنیم. برای این کار فرض کنید که ما از A بپرسیم که آیا B را می‌شناسد یا نه؟ اگر جواب مثبت باشد، A نمی‌تواند ستاره باشد، و اگر جواب منفی باشد B نمی‌تواند ستاره باشد؛ در هر صورت یکی از این افراد حذف می‌شوند. حال کافی است بین $n-۱$ نفر باقی‌مانده ستاره را پیدا کنیم. با $n-۱$ پرسش به یک نفر به نام s می‌رسیم؛ تنها s می‌تواند ستاره باشد، و با $۲(n-۱)$ پرسش این امر را می‌توان مشخص کرد. بنابراین تعداد کل پرسش‌ها می‌شود:

$$n - ۱ + ۲(n - ۱) = ۳(n - ۱)$$

حال الگوریتم این راه حل را می‌نویسیم. ورودی `Know` یک ماتریس مجاورت $n \times n$ است. مقدار `Know[i,j]` برابر یک است اگر فرد `i` فرد `j` را بشناسد، وگرنه صفر است. هدف پیدا کردن شماره‌ی `s` است به گونه‌ای که تمامی عناصر ستون `s`ام (به جز `[s,s]`) یک و تمامی عناصر سطر `s`ام (به جزمولفه `[s,s]`) صفر باشند:

```
CELEBRITY(Know, n)
    ▷ Input: Know (an n × n Boolean matrix)
    ▷ Output: the celebrity
i ← 1;  j ← 2;  next ← 3
    ▷ in the first phase we eliminate all but one candidate
while next ≤ n + 1
      do if Know[i, j]
            then i ← next
            else j ← next
         next := next + 1
         ▷ one of either i or j is eliminated
if i = n + 1
   then candidate ← j
   else candidate ← i
▷ Now we check that the candidate is indeed the celebrity
wrong ← false;  k ← 1
Know[candidate, candidate] ← false      ▷ a dummy variable to pass the test
while not wrong and k ≤ n
      do if Know[candidate, k]
            then wrong ← true
         if not Know[k, candidate]
            then if candidate ≠ k
                    then wrong ← true
         k ← k + 1
if not wrong
   then celebrity ← candidate
   else celebrity ← 0    ▷ no celebrity
```

البته این تعداد پرسش‌ها کمینه نیست و می‌توان به‌تر از این عمل کرد. ما می‌توانیم کاری کنیم که فردی به‌نام c که به‌عنوان کاندید ستاره‌ی مشهور انتخاب می‌شود، در مرحله‌ی اول حداقل به تعداد قابل توجهی مورد سؤال قرار گرفته باشد. اگر نتیجه‌ی همه‌ی پرسش‌ها را یادداشت کنیم، در مرحله‌ی دوم الگوریتم لازم نیست که این پرسش‌ها را مجدداً بپرسیم. برای این کار، توجه کنید که هر پرسش را می‌توان به‌صورت یک گره با دو فرزند نشان داد که «برنده»ی هر پرسش (کسی که احتمال ستاره بودن او هست) به‌عنوان پدر در درخت بالا می‌رود. با این ترتیب، مجموعه پرسش‌های مرحله‌ی اول الگوریتم تشکیل یک درخت می‌دهد که ریشه‌ی آن c است. در الگوریتم ارائه شده معلوم نیست که در چه مرحله‌ای و چند بار c مورد پرسش قرار گرفته است؛ ممکن است c در آخرین پرسش این مرحله درگیر شده باشد.

اگر درخت پرسش‌ها را به‌صورت یک درخت دودویی متوازن در بیاوریم، به‌طوری که همه‌ی افراد در اولین مرحله‌ی (یا حداکثر مرحله‌ی بعد) از پرسش‌ها قرار بگیرند، مطمئن خواهیم بود که c حتماً در تعدادی پرسش که متناسب با ارتفاع قرار داشته است. برای این کار ابتدا افراد را به دسته‌های دوتایی تقسیم می‌کنیم (اگر n فرد باشد، یک‌نفر تنها می‌ماند). از یکی از افراد هر دسته می‌پرسیم که آیا نفر دیگر آن‌دسته را می‌شناسد یا خیر. «برندگان» این پرسش‌ها به مرحله‌ی دوم می‌روند. اگر فردی در مرحله‌ی اول تنها باشد، در این مرحله حتماً در یک دسته‌ی دوتایی قرار می‌گیرد. همین کار را تکرار می‌کنیم تا به یک نفر برسیم که این فرد کاندید ماست. عمق هر برگ در این درخت نشان‌دهنده‌ی تعداد پرسش‌هایی است که از او شده است. ارتفاع این درخت برابر $\lceil \lg n \rceil$ است و با توجه به احتمال تنها افتادن کاندید یک بار در هر دو مرحله از دسته‌بندی‌ها، مطمئن هستیم که این فرد در $k = \lceil \frac{\lg n}{2} \rceil$ سؤال حتماً درگیر بوده است. یعنی در مرحله‌ی دوم الگوریتم، از $2(n-1)$ پرسش، مطمئنیم که kتای آن‌را در مرحله‌ی اول پرسیده‌ایم. پس تعداد کل پرسش‌ها برابر $3(n-1) - \lceil \frac{\lg n}{2} \rceil$ می‌شود.

۸.۳ مسئله‌ی «سربازها»

n سرباز در یک دریف هرکدام به دل‌خواه به‌سمت چپ یا راست ایستاده‌اند. با هر سوت فرمانده، هر سربازی که روبه‌روی سرباز دیگری ایستاده است ۱۸۰ درجه می‌چرخد. می‌خواهیم بدانیم که آیا این سربازها پس از تعدادی سوت به حالت «پایدار» می‌رسند که از آن پس هیچ سربازی جهت خود را تغییر ندهد و این تعداد سوت چند تاست؟

اگر سربازی که در جهت چپ یا راست ایستاده است را به‌ترتیب با صفر (۰) یا ۱ نشان دهیم، حالت ورودی دنباله‌ای از یک و صفر است. در هر سوت هر ۱۰ به ۰۱ تبدیل می‌شود، یعنی صفرها به‌سمت چپ و ۱ها به‌سمت راست حرکت می‌کنند، و یا در جای خود می‌ایستند. با توجه به محدودیت تعداد ارقام، بدیهی است که سربازها حتماً به حالت پایدار می‌رسند. هم‌چنین نشان می‌دهیم که تعداد سوت‌ها حداکثر $n - 1$ است.

لم ۴. در مسئله‌ی سربازها با n سرباز، برای رسیدن به حالت پایدار، $n - 1$ سوت کافی و همین تعداد سوت در برخی حالات لازم است.

اثبات. این مطلب را با استقرا اثبات می‌کنیم. برای این کار، نحوه‌ی حرکت و به‌مقصد رسیدن ۱ و صفرها را تحلیل می‌کنیم. حالت پایدار وقتی است که دنباله‌ی ورودی مرتب شود. کافی است نشان دهیم که همه‌ی ۱ها پس از $n - 1$ سوت در جای نهایی خود قرار می‌گیرند. برای این کار نشان می‌دهیم که iمین سمت راست‌ترین ۱، حتماً از سوت iام به‌بعد به‌سوی مقصد خود که محل $n - i + 1$ است، در حال حرکت است و بین ۱ تا وقتی که

به مقصد نرسیده باشد توقف نمی‌کند (یعنی با هر سوت یک محل به‌سمت راست می‌رود). اگر این مطلب را اثبات کنیم، با توجه به این که بیش‌ترین فاصله‌ی این ۱ با مقصدش وقتی است که در ابتدای صف باشد (یعنی نیاز به حداکثر $n-i$ سوت اضافی دارد،) به‌وضوح پس از سوت $n-1$ام این ۱ به مقصد نهایی‌اش خواهد رسید.

پایه‌ی استقرا برای $i = 1$ است. بدیهی است که اولین سمت راست‌ترین ۱ با اولین سوت به‌سمت راست حرکت می‌کند (مگر آن‌که در آخر صف باشد،) و حداکثر پس از $n-1$ سوت به‌مقصدش می‌رسد. حال فرض کنید که این مطلب برای همه‌ی $1 \le k < i$امین سمت راست‌ترین ۱ درست است، نشان می‌دهیم که برای iامین سمت‌راست‌ترین ۱ هم درست خواهد بود. در این‌جا نکته در آن است اولاً، ۱ ها از هم عبور نمی‌کنند و ثانیاً، تنها $i-1$امین ۱ می‌تواند جلوی حرکت iامین ۱ را بگیرد (وقتی که کنار هم باشند). با فرض استقرا، $i-1$امین ۱ در سوت $i-1$ام حرکت کرده است پس در انتهای این سوت به‌جای آن یک صفر قرار می‌گیرد. پس iامین ۱ می‌تواند در سوت بعدی حرکتش را شروع کند. چون طبق فرض استقرا $i-1$امین ۱ از سوت $i-1$ام به بعد در هیچ سوتی نمی‌ایستد، این مطلب برای iامین ۱ هم درست است. □