طراحی الگوریتم‌ها – برنامه نویسی پویا

Introduction to Algorithm

مهدی جوانمردی

مرور جلسه قبل

مرتب‌سازی سطلی BUCKET SORT

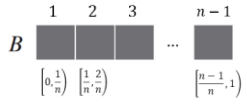
n-element array A

$0 \le A[i] < 1$

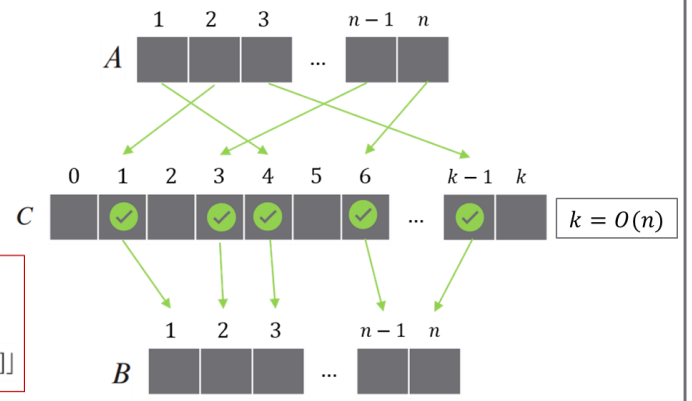
توزیع نرمال در بازه [0,1)

$B[0..n-1]$

سطل



$A[i]$ → $\lfloor nA[i] \rfloor$



$k = O(n)$

مرتب‌سازی خطی RADIX SORT

اطلاعات مازاد: اعداد دارای d رقم و هر رقم k مقدار متفاوت

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

```
RADIX-SORT(A, d)
for i = 1 to d
    use a stable sort to sort array A on digit i
```

- زمان اجرا: $\theta(d(n+k))$
- اگر $k = O(n)$ و $d = O(1)$ باشد زمان اجرای radix sort $\theta(n)$ خواهد بود

تحلیل زمانی BUCKET SORT

```
BUCKET-SORT(A)
let B[0..n − 1] be a new array
n = A.length
for i = 0 to n − 1
    make B[i] an empty list
for i = 1 to n
    insert A[i] into list B[⌊nA[i]⌋]
for i = 0 to n − 1
    sort list B[i] with insertion sort
concatenate the lists B[0], B[1], ..., B[n − 1] together in order
```

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

n_i متغیر تصادفی نشانگر تعداد المان‌ها در سطل B_i

$$= \Theta(n) + \sum_{i=0}^{n-1} O\left(2 - \frac{1}{n}\right) = \Theta(n)$$

تحلیل زمانی RADIX SORT

- اگر n عدد b بیتی داشته باشیم، برای هر عدد مثبت دلخواه $r \le b$ خواهیم داشت:

زمان اجرایی radix sort برای این اعداد $\theta((b/r)(n+2^r))$ خواهد بود

به شرطی که مرتب‌سازی پایدار استفاده شده $\theta(n+k)$ باشد

برای اعداد n و b تعیین $r \le b$ بگونه‌ای که زمان اجرای $(b/r)(n+2^r)$ را کمینه کند

حالت اول: $b < \lfloor \lg n \rfloor$ ← برای $r = b$ خواهیم داشت: $\Theta(n)$

حالت دوم: $b \ge \lfloor \lg n \rfloor$ ← برای $r = \lfloor \lg n \rfloor$ خواهیم داشت: $\Theta(bn/\lg n)$

درصورتی که $b = O(\lg n)$ باشد با انتخاب $r = \lfloor \lg n \rfloor$ زمان اجری radix sort برابر $O(n)$

فصل ۱۵ کتاب

- برنامه‌نویسی پویا
  - مسئله برش میله
  - ضرب زنجیره‌ای ماتریس
  - المان‌های برنامه‌نویسی پویا
  - طولانی‌تر زیر رشته مشترک
  - درخت دودویی جستجو بهینه

15 Dynamic Programming *359*

15.1 Rod cutting *360*

15.2 Matrix-chain multiplication *370*

15.3 Elements of dynamic programming *378*

15.4 Longest common subsequence *390*

15.5 Optimal binary search trees *397*

برنامه نویسی پویا Dynamic Programming

- یک روش برای حل مسئله با ترکیب حل زیرمسئله‌ها
- مشابه روش‌های تقسیم و حل

روش تقسیم و حل		روش برنامه‌نویسی پویا
تقسیم مسئله به زیرمسئله‌های مجزا و حل بازگشتی زیرمسئله‌ها ترکیب جواب‌ها و حل مسئله اصلی	⟷	تقسیم مسئله به زیرمسئله‌هایی با هم‌پوشانی بالا حل هر زیرمسئله فقط یکبار و ذخیره‌ی نتیجه استفاده از نتایج قبلی برای حل مسئله بزرگتر

- استفاده برنامه‌نویسی پویا در مسئله‌های بهینه سازی
- بهینه سازی: راه حل‌های متعددی برای حل مسئله دارد و ما جواب بهینه را میخواهیم
- هر جواب یک ارزش/هزینه‌ای دارد: جواب بهینه دارای ماکزیمم یا مینیمم ارزش/هزینه باشد

مراحل برنامه‌نویسی پویا

پایه‌های برنامه‌نویسی پویا

1. مشخص کردن ساختار یک جواب بهینه
2. تعیین مقدار برای یک جواب بهینه بصورت بازگشتی (زیرساختار بهینه)
3. محاسبه مقدار یک جواب بهینه، معمولا بصورت پایین به بالا
4. جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور

اگر فقط مقدار بهینه را میخواهیم و نه خود جواب بهینه، میتوان از مرحله چهارم صرف نظر کرد
برای بدست آوردن جواب بهینه می‌بایست برخی اطلاعات مازاد در مراحل قبلی نگهداری شوند

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution. (optimal substructure)
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

مسئله برش میله Rod Cutting

- برش یک میله بلند به قطعات کوچکتر به گونه ای که قیمت فروش آن بیشنیه شود!

- ورودی‌ها:
  - طول میله اولیه n
  - قیمت فروش بر حسب طول p_i
  - هزینه برش رایگان است
- خروجی‌ها:
  - بیشترین درآمد قابل حصول r_n
  - طول برش‌های مورد نیاز

جدول قیمت فروش بر حسب طول

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

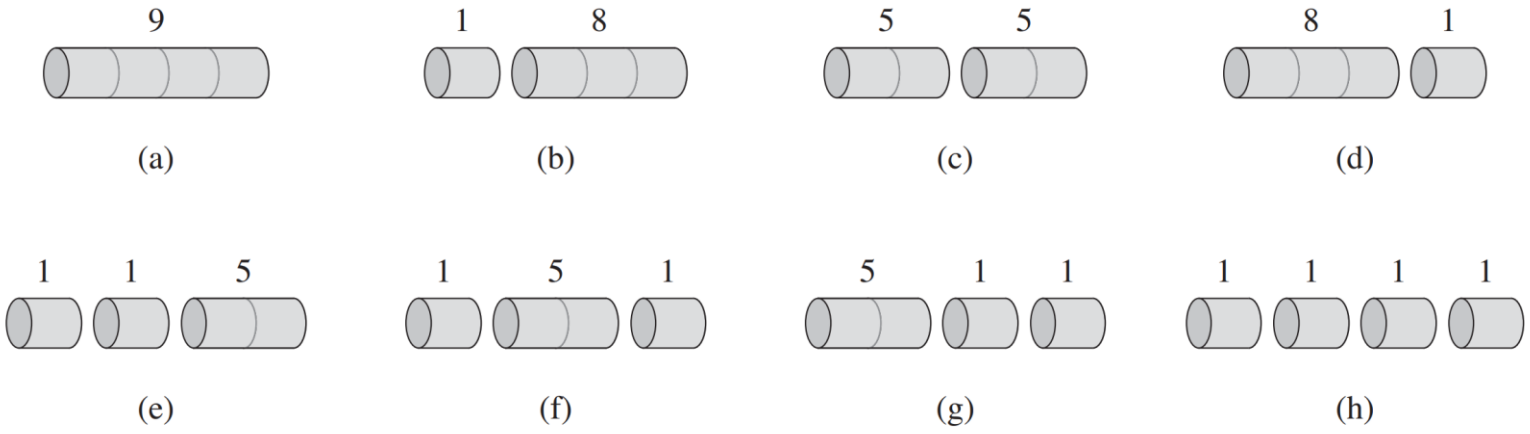
مثال

$$n = 4 \longrightarrow p_2 + p_2 = 5 + 5 = 10$$

مسئله برش میله Rod Cutting

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

جدول قیمت فروش بر حسب طول



حالت‌های ممکن برای برش میله به طول ۴

- تعداد حالات ممکن برای برش یک میله به طول n؟

We can cut up a rod of length n in 2^{n-1} different ways, since we have an independent option of cutting, or not cutting, at distance i inches from the left end,

جواب بهینه با مسئله برش میله

- اگر جواب بهینه میله را به k قسمت تقسیم کند، در این صورت خواهیم داشت:

$$1 \le k \le n$$

$$n = i_1 + i_2 + \cdots + i_k$$

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$$

به‌گونه‌ای‌که در آمد حاصله بیشینه باشد

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

جدول قیمت فروش بر حسب طول

$r_1 = 1$ from solution $1 = 1$ (no cuts) ,
$r_2 = 5$ from solution $2 = 2$ (no cuts) ,
$r_3 = 8$ from solution $3 = 3$ (no cuts) ,
$r_4 = 10$ from solution $4 = 2 + 2$,
$r_5 = 13$ from solution $5 = 2 + 3$,
$r_6 = 17$ from solution $6 = 6$ (no cuts) ,
$r_7 = 18$ from solution $7 = 1 + 6$ or $7 = 2 + 2 + 3$,
$r_8 = 22$ from solution $8 = 2 + 6$,
$r_9 = 25$ from solution $9 = 3 + 6$,
$r_{10} = 30$ from solution $10 = 10$ (no cuts) .

به دست آوردن ساختار جواب بهینه

- بیشترین در آمد حاصل از برش یک میله به طول n (r_n)
با فرض داشتن بیشترین درآمد حاصل از برش میله‌های کوچکتر $(r_1, r_2, \ldots, r_{n-1})$:

$$r_n = \max\left(\underline{p_n}, \underline{r_1 + r_{n-1}}, \underline{r_2 + r_{n-2}}, \ldots, \underline{r_{n-1} + r_1}\right)$$

بدون برش

اولین برش با طول ۱

اولین برش با طول ۲

اولین برش با طول n-1

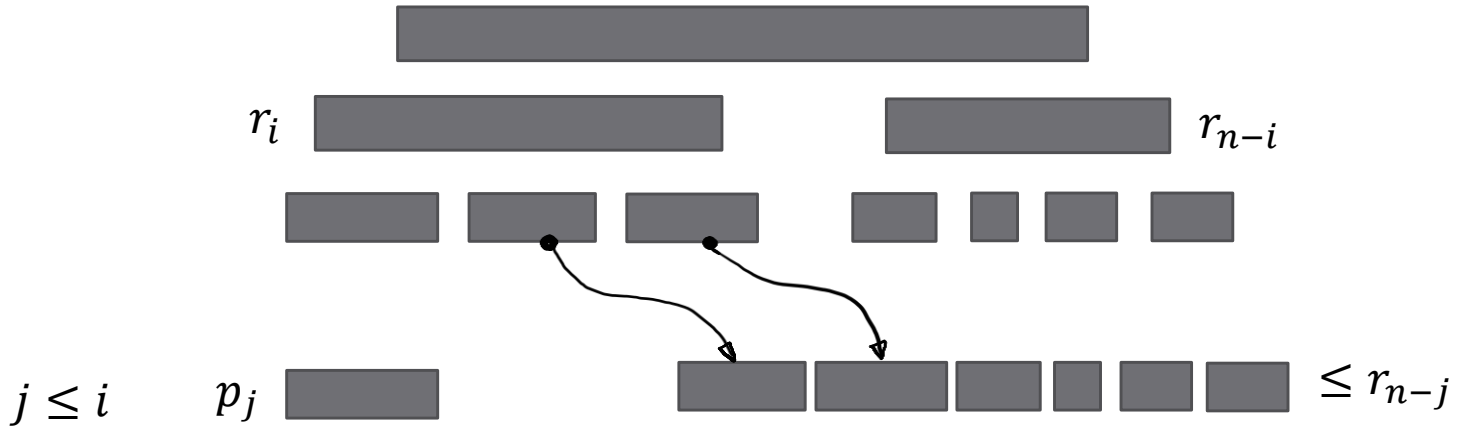
در ادامه، برش طرفین میله بصورت بهینه!

Optimal substructure
زیرساختار بهینه:
جواب بهینه مسئله را
می‌توان از جواب بهینه
زیرمسئله‌ها به دست آورد

ساختار کلی جواب بهینه

- بیشترین در آمد حاصل از برش یک میله به طول n (r_n)
با فرض داشتن بیشترین درآمد حاصل از برش میله‌های کوچکتر $(r_1, r_2, \dots, r_{n-1})$:

$$r_n = \max\left(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1\right)$$



کلی حالت‌های تکراری خواهیم داشت.. برش فقط یک طرف در ادامه همه حالات را ایجاد میکند!

ساختار کلی جواب بهینه

کمی ساده‌تر

$$r_n = \max_{1<i<n}\left(p_i + r_{n-i}\right)$$

حل بالا به پایین مسئله برش میله

- حل بالا به پایین مسئله بصورت بازگشتی

ساختار کلی جواب بهینه

$$r_n = \max_{1<i<n} (p_i + r_{n-i}))$$

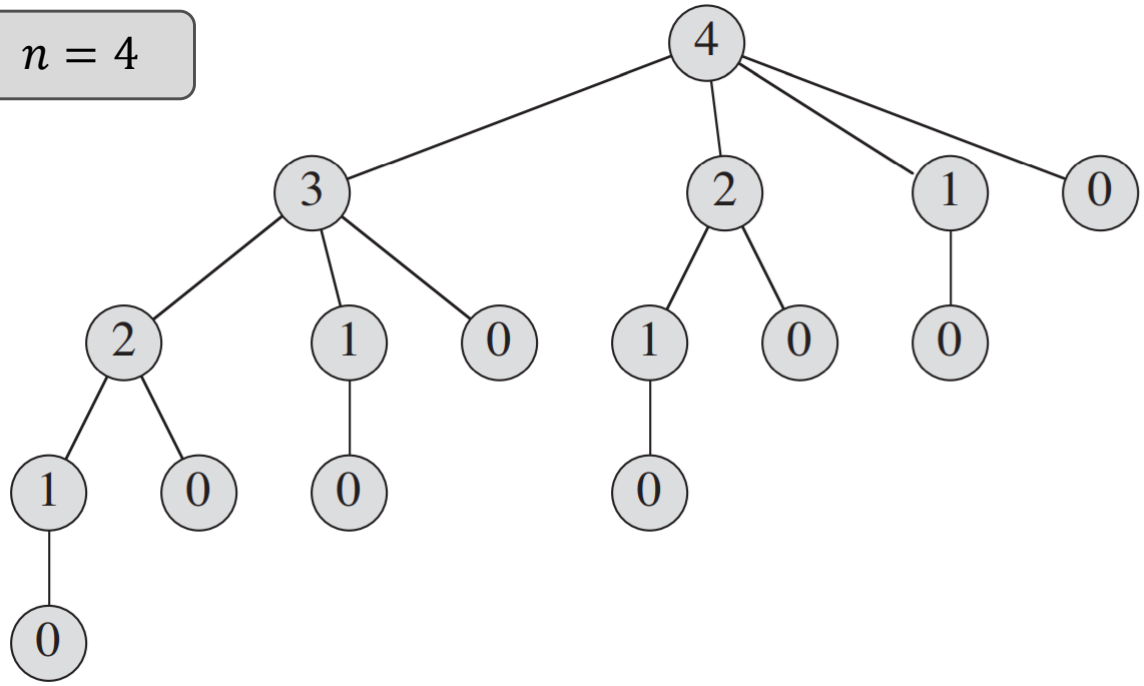
Recursive top-down approach

```
CUT-ROD(p, n)
if n == 0
    return 0
q = -∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n - i))
return q
```

بسیار زمان‌بر است! چرا؟؟

تحلیل زمان اجرا حل بالا به پایین مسئله برش میله

$n = 4$




```
CUT-ROD(p, n)
if n == 0
    return 0
q = -∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n - i))
return q
```

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) \longrightarrow T(n) = 2^n$$

For $n = 0$, this holds since $2^0 = 1$.

For $n > 0$, substituting into the recurrence, we have

$$\begin{aligned} T(n) &= 1 + \sum_{j=0}^{n-1} 2^j \\ &= 1 + (2^n - 1) \\ &= 2^n. \end{aligned}$$

برنامه نویسی پویا و مسئله برش میله

- ایده اصلی: کاری کنیم که هر زیرمسئله فقط یکبار حل شود!
- اگر در ادامه به زیرمسئله تکراری نیاز داریم از جواب قبلی استفاده کنیم نه محاسبه مجدد
- نیاز به حافظه اضافی برای نگهداری جواب‌های محاسبه شده (time-memory trade-off)

- روش‌های پیاده‌سازی

۱. حل بالا به پایین با حفظ کردن	۲. حل پایین به بالا
ساختار کلی مشابه روش قبل قبل از حل هر تابع بازگشتی چک میکند که آیا قبلا محاسبه شده یا نه اگر محاسبه شده فقط استفاده میکند در غیر اینصورت محاسبه و ذخیره میکند	حل مسئله از کوچک به بزرگ شروع از کوچکترین زیرمسئله برای حل هر زیرمسئله مقادیر کوچکتر آن قبلا محاسبه و ذخیره شده حل مسئله بزرگتر با مقادیر کوچکتر

برنامه نویسی پویا و مسئله برش میله

۱. حل بالا به پایین با حفظ کردن

ساختار کلی مشابه روش قبل

قبل از حل هر تابع بازگشتی چک میکند
که آیا قبلا محاسبه شده یا نه

اگر محاسبه شده فقط استفاده میکند
در غیر اینصورت محاسبه و ذخیره میکند

```
MEMOIZED-CUT-ROD(p, n)
let r[0..n] be a new array
for i = 0 to n
    r[i] = -∞
return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] ≥ 0
    return r[n]
if n == 0
    q = 0
else q = -∞
    for i = 1 to n
        q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
r[n] = q
return q
```

برنامه نویسی پویا و مسئله برش میله

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max(q, p[i] + r[j - i])
    r[j] = q
return r[n]
```

۲. حل پایین به بالا

حل مسئله از کوچک به بزرگ
شروع از کوچکترین زیرمسئله

برای حل هر زیرمسئله مقادیر کوچکتر آن
قبلا محاسبه و ذخیره شده

حل مسئله بزرگتر با مقادیر کوچکتر

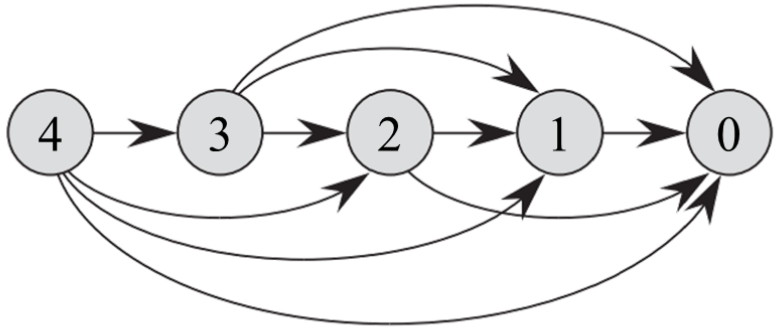
$$r_n = \max_{1<i<n} (p_i + r_{n-i}))$$

- تحلیل زمانی هر دو روش: به دلیل حلقه تو در تو $\Theta(n^2)$
- روش پایین به بالا دارای ثابت کوچکتر و در نتیجه سریع‌تر

درخت زیرمسئله‌ها

- برای حل مسائل بصورت برنامه‌نویسی پویا باید رابطه بین زیرمسئله‌ها را بدانیم
- درخت زیرمسئله‌ها میتواند اطلاعات کافی در این زمینه به ما بدهد

درخت زیرمسئله‌ها برای $n = 4$



- یال جهت‌دار از گره x به گره y یعنی:
  حل بهینه x نیازمند جواب بهینه y
- این گراف فشرده شده گراف حل بالا به پایین بازگشتی
- در روش پایین به بالا، گره y که از x به آن یالی وجود دارد
  باید قبل از حل گره x حتما حل شده باشد

Topological sort Sect. 21

بازسازی جواب بهینه

- تا الان: محاسبه بیشترین درآمد ممکن!
- اما روش بدست آوردن بیشترین درآمد؟ ← لیستی از طول قطعات

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[0..n] be new arrays
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        if q < p[i] + r[j - i]
            q = p[i] + r[j - i]
            s[j] = i
    r[j] = q
return r and s
```

- نیازمند ذخیره‌سازی نه تنها مقدار بهینه بلکه انتخاب بهینه!
- انتخاب بهینه:

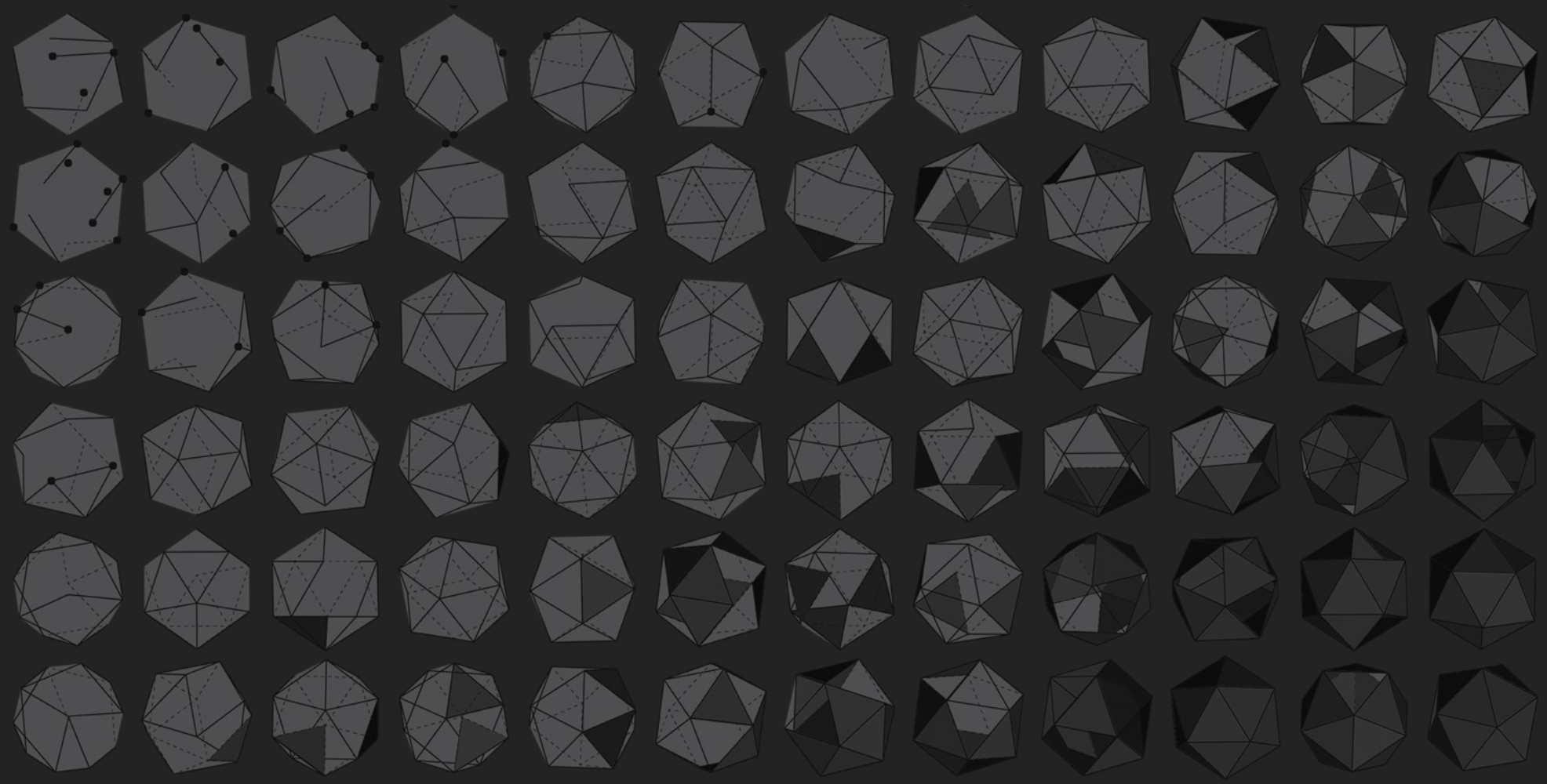
طول قطعه حاصل از اولین برش در حالت بهینه

بازسازی جواب بهینه

- تا الان: محاسبه بیشترین درآمد ممکن!
- اما روش بدست آوردن بیشترین درآمد؟ ← لیستی از طول قطعات

```
PRINT-CUT-ROD-SOLUTION(p, n)
(r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
while n > 0
    print s[n]
    n = n - s[n]
```

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10



ضرب زنجیره‌ای ماتریس‌ها

مرور مطالب قبل

Dynamic programming

روش برنامه‌نویسی پویا

تقسیم مسئله به زیرمسئله‌هایی با هم‌پوشانی
حل هر زیرمسئله فقط یکبار و ذخیره‌ی نتیجه
استفاده از نتایج قبلی برای حل مسئله بالاتر

روش تقسیم و حل

تقسیم مسئله به زیرمسئله‌های مجزا
و حل بازگشتی زیرمسئله‌ها
ترکیب جواب‌ها و حل مسئله اصلی

مسئله برش میله Rod cutting

برش یک میله بلند به قطعات کوچکتر به گونه ای که قیمت فروش آن بیشنیه شود!

ورودی‌ها

- طول میله اولیه n
- قیمت فروش بر حسب طول p_i
- قیمت برش رایگان

خروجی‌ها

- بیشترین درآمد قابل حصول r_n
- طول برش‌های مورد نیاز

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

مثال

$n = 4 \longrightarrow p_2 + p_2 = 5 + 5 = 10$

مراحل برنامه نویسی پویا

1. مشخص کردن ساختار یک جواب بهینه
2. تعیین مقدار برای یک جواب بهینه بصورت بازگشتی
3. محاسبه مقدار یک جواب بهینه، معمولا بصورت پایین به بالا
4. جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور

اگر فقط مقدار بهینه را میخواهیم و نه خود جواب بهینه، میتوان از مرحله چهارم صرف نظر کرد
برای بدست آوردن جواب بهینه میبایست برخی اطلاعات مازاد در مراحل قبلی نگهداری شوند

حل پایین به بالا و پویا مسئله برش میله

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max(q, p[i] + r[j - i])
    r[j] = q
return r[n]
```

حل مسئله از کوچک به بزرگ
شروع از کوچکترین زیرمسئله

برای حل هر زیرمسئله مقادیر کوچکتر آن
قبلا محاسبه و ذخیره شده

حل مسئله بزرگتر با مقادیر کوچکتر

مروری بر ماتریس و ضرب زنجیره‌ای ماتریس‌ها

ماتریس و نحوه نمایش آن:

ماتریس $n \times m$ بنام $A = [a[i,j]]$ یک آرایه دوبعدی بصورت

با n سطر و m ستون است

$$A = \begin{bmatrix} a[1,1] & a[1,2] & \cdots & a[1,m-1] & a[1,m] \\ a[2,1] & a[2,2] & \cdots & a[2,m-1] & a[2,m] \\ \vdots & \vdots & & \vdots & \vdots \\ a[n,1] & a[n,2] & \cdots & a[n,m-1] & a[n,m] \end{bmatrix}$$

ضرب ماتریس‌ها:

ضرب ماتریسی $C = AB$ دو ماتریس $p \times q$ و $q \times r$

که در آن $1 \le i \le p$ و $1 \le j \le r$

$$c[i,j] = \sum_{k=1}^{q} a[i,k]b[k,j]$$

$$A = \begin{bmatrix} 1 & 8 & 9 \\ 7 & 6 & -1 \\ 5 & 5 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 8 \\ 7 & 6 \\ 5 & 5 \end{bmatrix} \quad C = AB = \begin{bmatrix} 102 & 101 \\ 44 & 87 \\ 70 & 100 \end{bmatrix}$$

- مسئله بعدی برنامه‌نویسی پویا ← ضرب زنجیره‌ای ماتریس‌ها

ورودی	$\langle A_1, A_2, \ldots, A_n \rangle$	خروجی	$A_1 A_2 \cdots A_n$

ضرب دو ماتریس در یکدیگر

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

$$c[i,j] = \sum_{k=1}^{q} a[i,k]b[k,j]$$

- ضرب ماتریسی $C = AB$ دو ماتریس $p \times q$ و $q \times r$

```
MATRIX-MULTIPLY(A, B)
if A.columns ≠ B.rows
    error "incompatible dimensions"
else let C be a new A.rows × B.columns matrix
    for i = 1 to A.rows
        for j = 1 to B.columns
            c_ij = 0
            for k = 1 to A.columns
                c_ij = c_ij + a_ik · b_kj
    return C
```

تعداد ضرب مورد نیاز: $p \times q \times r$

ضرب زنجیره‌ای ماتریس‌ها

- ضرب زنجیره‌ای ماتریس‌ها

$$A_1A_2A_3\cdots A_{s-1}A_s = A_1(A_2(A_3\cdots(A_{s-1}A_s)))$$

- خاصیت شرکت پذیری ضرب ماتریسی

$$A_1A_2A_3 = (A_1A_2)A_3 = A_1(A_2A_3)$$

- تعداد کل ضرب مورد نیاز بستگی به انتخاب پرانتز گذاری دارد!

ضرب ماتریسی مورد نظر

$$A_1A_2A_3A_4 \longrightarrow \begin{array}{l} (A_1(A_2(A_3A_4))) \\ (A_1((A_2A_3)A_4)) \\ ((A_1A_2)(A_3A_4)) \\ ((A_1(A_2A_3))A_4) \\ (((A_1A_2)A_3)A_4) \end{array}$$

انواع پرانتز گذاری ممکن

تعداد کل ضرب‌ها در ضرب زنجیره‌ای ماتریس‌ها

- در ضرب ماتریسی تعداد کل ضرب‌ها بستگی به انتخاب پرانتز گذاری دارد!

$A_{i..j} = A_i A_{i+1} \cdots A_j$

$A_1 : 10 \times 100$

$A_2 : 100 \times 5$

$A_3 : 5 \times 50$

$A_4 : 50 \times 1$

1. $(A_1(A_2(A_3A_4)))$
2. $(A_1((A_2A_3)A_4))$
3. $((A_1A_2)(A_3A_4))$
4. $((A_1(A_2A_3))A_4)$
5. $(((A_1A_2)A_3)A_4)$

1 $(A_1(A_2(A_3A_4)))$

- $A_{34} = A_3A_4$, 250 mults, result is 5 by 1
- $A_{24} = A_2A_{34}$, 500 mults, result is 100 by 1
- $A_{14} = A_1A_{24}$, 1000 mults, result is 10 by 1
- Total is 1750

2 $((A_1(A_2A_3))A_4)$

- $A_{23} = A_2A_3$, 25000 mults, result is 100 by 50
- $A_{13} = A_1A_{23}$, 50000 mults, result is 10 by 50
- $A_{14} = A_{13}A_4$, 500 mults, results is 10 by
- Total is 75500

3 $((A_1A_2)(A_3A_4))$

- $A_{12} = A_1A_2$, 5000 mults, result is 10 by 5
- $A_{34} = A_3A_4$, 250 mults, result is 5 by 1
- $A_{14} = A_{12}A_{34})$, 50 mults, result is 10 by 1
- Total is 5300

4 $(A_1((A_2A_3)A_4))$

- $A_{23} = A_2A_3$, 25000 mults, result is 100 by 50
- $A_{24} = A_{23}A_4$, 5000 mults, result is 100 by 1
- $A_{14} = A_1A_{24}$, 1000 mults, result is 10 by 1
- Total is 31000

5 $(((A_1A_2)A_3)A_4)$

- $A_{12} = A_1A_2$, 5000 mults, result is 10 by 5
- $A_{13} = A_{12}A_3$, 2500 mults, result is 10 by 50
- $A_{14} = A_{13}A_4$, 500 mults, results is 10 by 1
- Total is 8000

ضرب زنجیره‌ای ماتریس‌ها بصورت بهینه!

صورت مسئله:

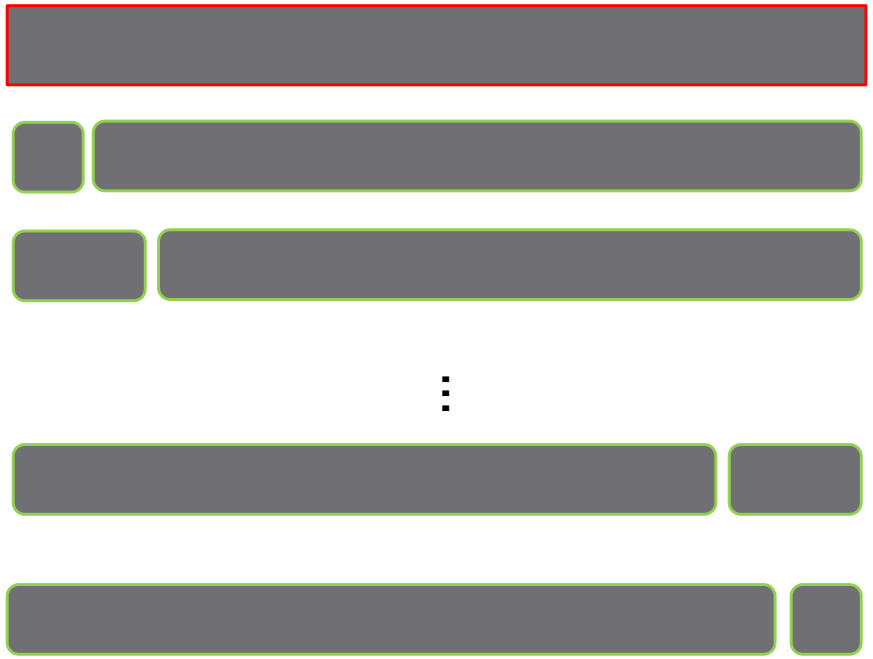
با فرض در اختیار داشتن زنجیره n تایی از ماتریس‌ها $\langle A_1, A_2, \dots, A_n \rangle$
که در آن ابعاد هر یک از A_i برابر باشد با $p_{i-1} \times p_i$

انتخاب یک پرانتزگذاری کامل برای ضرب زنجیره‌ای ماتریس‌های فوق بصورتیکه به کمترین تعداد ضرب نیاز داشته باشد

- در این مسئله در واقع با ضرب ماتریس‌ها کاری نداریم و هدف تعیین کم‌هزینه‌ترین ترتیب ممکن برای ضرب ماتریس‌ها است!
- طبیعتا هزینه پیدا کردن روش بهینه باید خیلی کمتر از آورده حاصل از ضرب به روش بهینه باشد!

تعداد پرانتز گذاری‌های ممکن

$P(n)$: تعداد پرانتز گذاری ممکن برای n ماتریس



	تعداد حالت‌های پرانتزگذاری
$A_1 A_2 \dots A_n$	
$A_1(A_2 \dots A_n)$	$P(1)P(n-1)$
$(A_1 A_2)(A_3 \dots A_n)$	$P(2)P(n-2)$
$\vdots$	
$(A_1 \dots A_{n-2})(A_{n-1} A_n)$	$P(n-2)P(2)$
$(A_1 \dots A_{n-1})\, A_n$	$P(n-1)P(1)$

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases} \longrightarrow \Omega(2^n)$$

✗ روش brute-force

برنامه‌نویسی پویا و حل ضرب زنجیره‌ای ماتریس‌ها

پایه‌های برنامه‌نویسی پویا

1. مشخص کردن ساختار یک جواب بهینه
2. تعیین مقدار برای یک جواب بهینه بصورت بازگشتی
3. محاسبه مقدار یک جواب بهینه، معمولا بصورت پایین به بالا
4. جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور

1. تعیین زیرساختار پرانتزگذاری بهینه
2. راه‌حل بازگشتی برای پرانتزگذاری بهینه
3. محاسبه هزینه (تعداد ضرب) یک پرانتزگذاری بهینه بصورت پایین به بالا
4. جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور

۱. تعیین زیرساختار پرانتزگذاری بهینه

تعیین ساختار پرانتزگذاری بهینه و استفاده از آن برای یافتن بهترین پاسخ

$$A_{i..j} = A_i A_{i+1} \cdots A_j$$

each matrix A_i is $p_{i-1} \times p_i$

درصورتی که $i < j$ باشد برای پرانتزگذاری $A_{i..j} = A_i A_{i+1} \cdots A_j$ باید آن را بین دو ماتریس A_k و A_{k+1} به دو قسمت $A_{i..k}$ و $A_{k+1..j}$ تقسیم کنیم و خواهیم داشت $i \leq k < j$

$$A_{i..j} \rightarrow A_{i..k} \quad A_{k+1..j}$$

اگر $i < j$

هزینه محاسبه $A_{i..j}$ با این روش پرانتزگذاری برابر است با

- هزینه محاسبه ماتریس $A_{i..k}$
- هزینه محاسبه ماتریس $A_{k+1..j}$
- هزینه ضرب این دو

ساختار بهینه: اگر پرانتزگذاری بهینه $A_{i..j}$ آن را به $A_{i..k}$ و $A_{k+1..j}$ تقسیم کرده باشد، می‌بایست پرانتزگذاری $A_{i..k}$ و $A_{k+1..j}$ نیز به خودی خود بهینه باشند

۲. راه حل بازگشتی برای پرانتزگذاری بهینه

- استفاده از زیرساختار بهینه برای تعریف راه حل بازگشتی برای پرانتزگذاری
- $m[i,j]$: کمترین تعداد ضرب مورد نیاز برای محاسبه $A_{i..j} = A_i A_{i+1} \cdots A_j$

each matrix A_i is $p_{i-1} \times p_i$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j \end{cases}$$

- $m[i,j]$ معرف هزینه بهینه است اما انتخاب بهینه را به ما نمی‌دهد
- $s[i,j]$: عدد k انتخاب شده که به موجب آن هزینه بهینه می‌شود
- می‌توان برای فرمول فوق یک برنامه بازگشتی نوشت $m[1,n]$ ← زمان اجرای نمایی

۳. محاسبه هزینه یک پرانتزگذاری بهینه

- تعداد زیرمسئله‌های $m[i,j]$ محدود است: با توجه به $1 \le i \le j \le n$ داریم $\Theta(n^2)$
- به جای راه حل بازگشتی، ارائه راه حل پایین به بالا

$A_1 A_2 \dots A_n$

$A_i : \quad p_{i-1} \times p_i \text{ for } i = 1, 2, \dots, n$

$p = \langle p_0, p_1, \dots, p_n \rangle$

$p.length = n + 1$

$m[1..n, 1..n]$ جدول هزینه‌ها

$s[1..n-1, 2..n]$ جدول انتخاب‌ها

$$m[i,j] = \min_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\}$$

طول زیرمسئله $m[i,j]$: $l = j - i + 1$

برای حل زیرمسئله‌هایی به طول l صرفا نیاز به زیرمسئله‌هایی با طول $1 < k < l$ داریم

Bottom-up approach

شروع به حل از زیرمسئله‌های بطول ۱، ۲ الی n

۳. محاسبه هزینه یک پرانتزگذاری بهینه

- تعداد زیرمسئله‌های $m[i,j]$ محدود است: با توجه به $1 \le i \le j \le n$ داریم $\Theta(n^2)$
- به جای راه حل بازگشتی، ارائه راه حل پایین به بالا

```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n              // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

$$A_i : \quad p_{i-1} \times p_i \text{ for } i = 1, 2, \ldots, n$$
$$p = \langle p_0, p_1, \ldots, p_n \rangle$$

$m[1..n, 1..n]$ جدول هزینه‌ها

$s[1..n-1, 2..n]$ جدول انتخاب‌ها

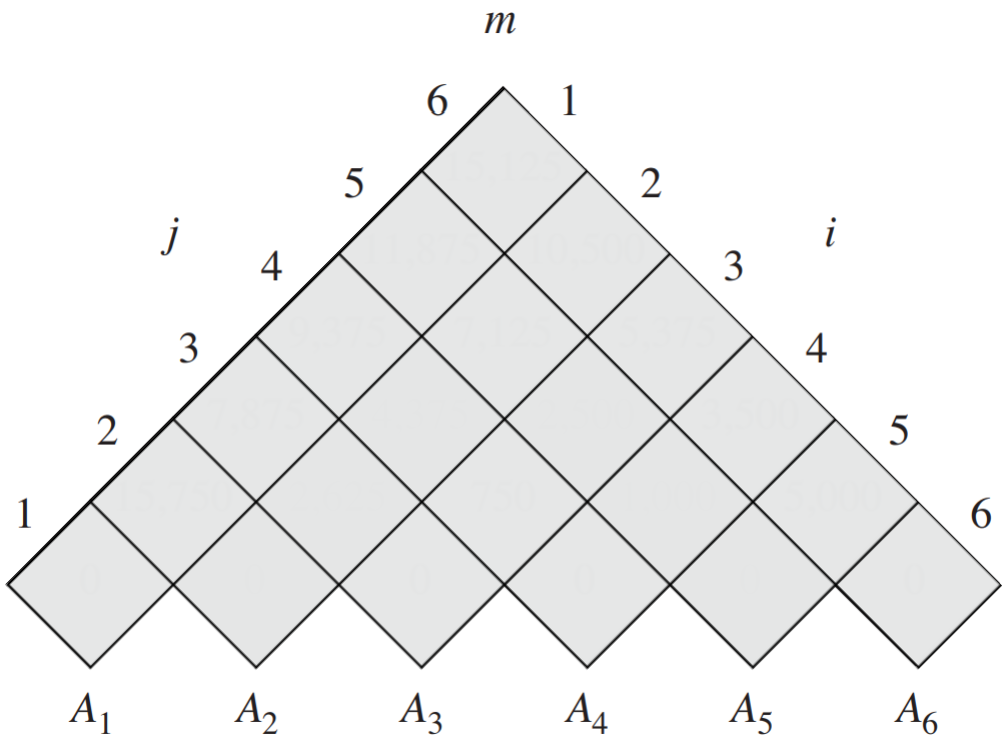
طول زیرمسئله $m[i,j]$: $l = j - i + 1$

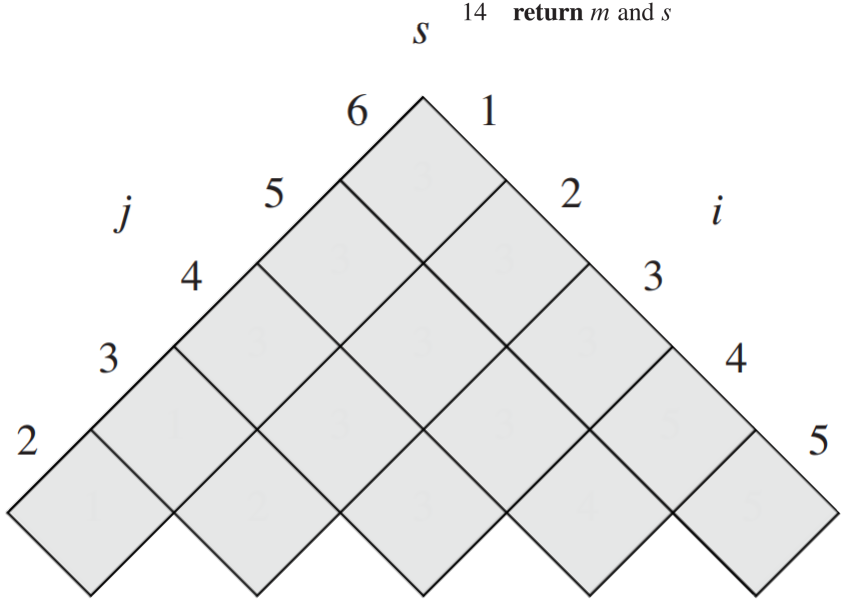
برای حل زیرمسئله‌هایی به طول l صرفا نیاز به زیرمسئله‌هایی با طول $1 < i < l$ داریم

۳. محاسبه هزینه یک پراتزگذاری بهینه

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n              // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

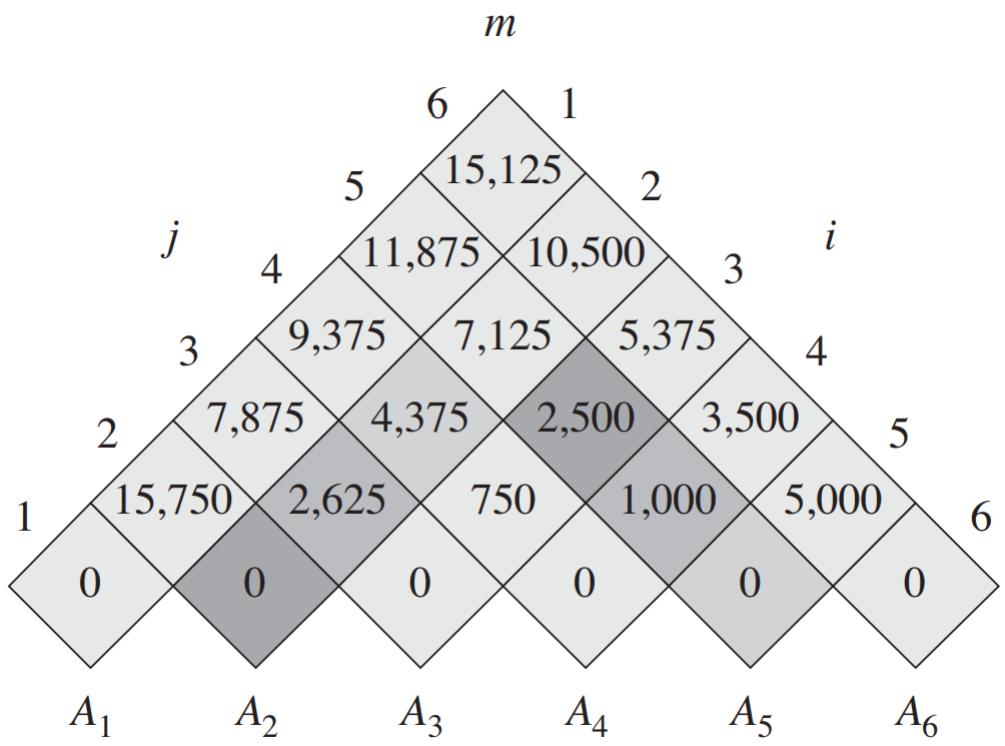


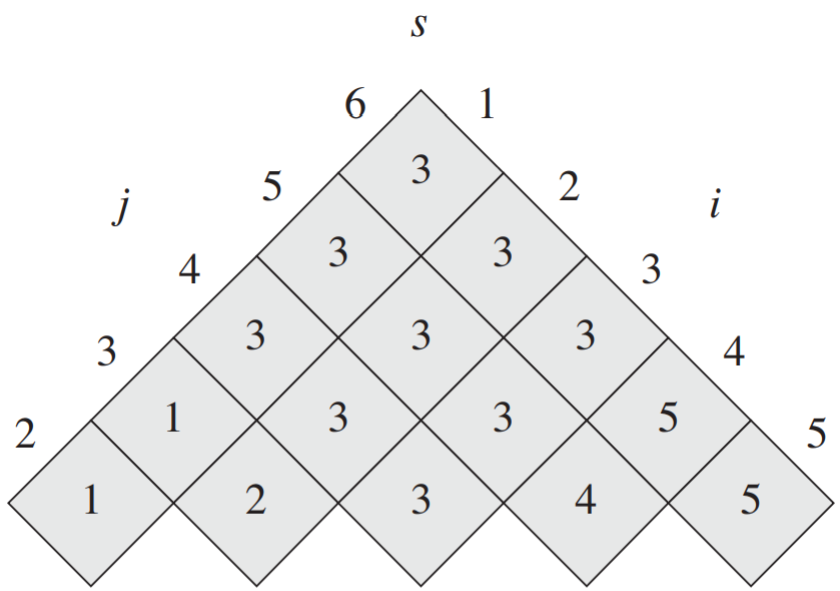


$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$

۳. محاسبه هزینه یک پرانتزگذاری بهینه

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25





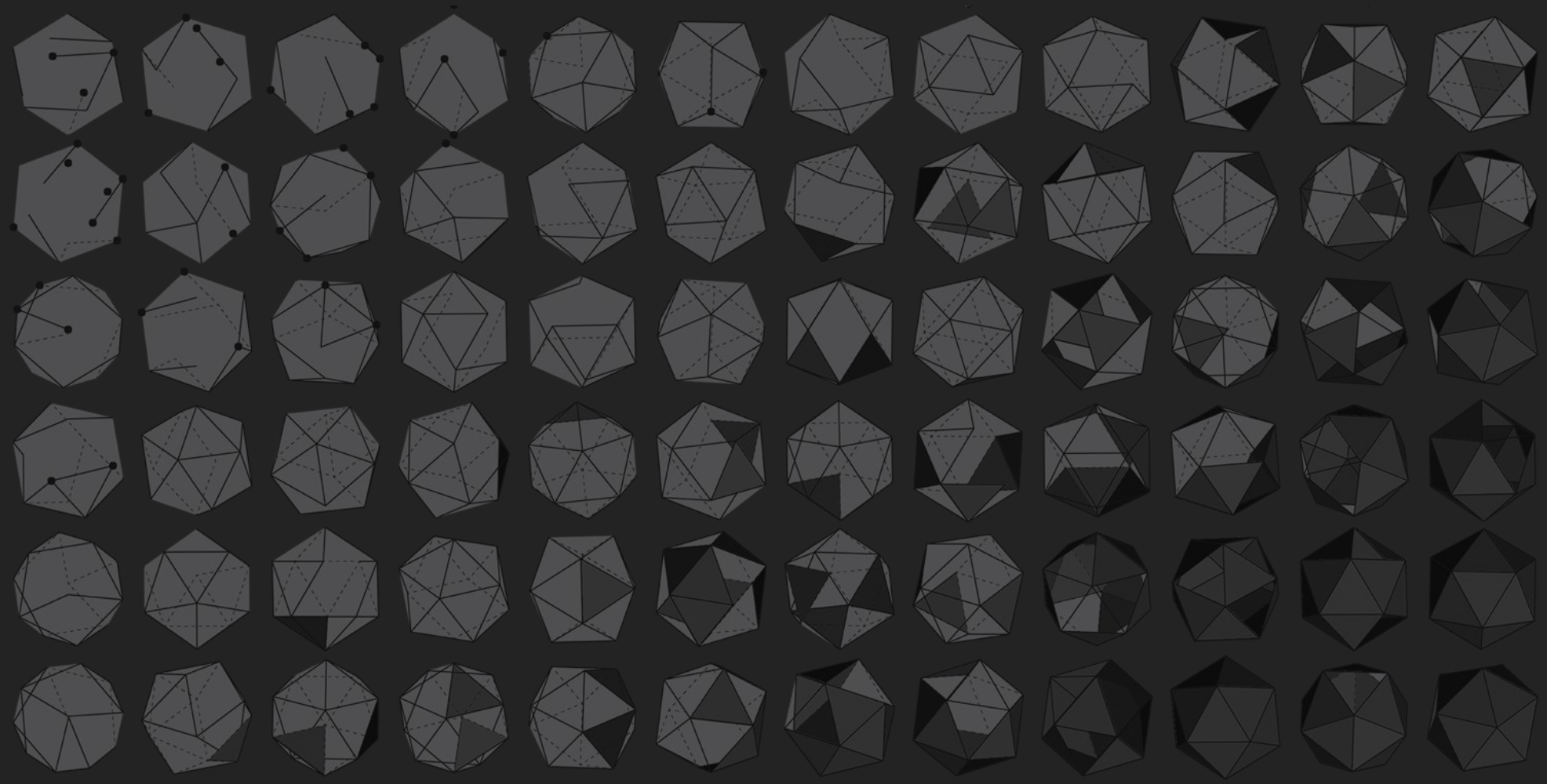
$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$

۴. بازسازی جواب بهینه را از روی اطلاعات

- شبیه کد بازسازی جواب بهینه از روی جدول s

```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```

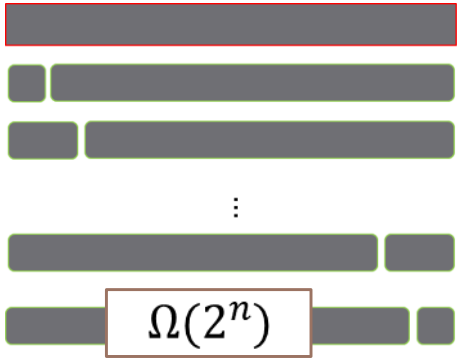
$\boxed{\text{PRINT-OPTIMAL-PARENS}(s, 1, n)}$



طولانی‌ترین زیردنباله مشترک LCS

مرور مطالب قبل

یافتن پرانتزگذاری با تعداد ضرب بهینه



تعداد حالت‌های پرانتزگذاری	$A_1A_2 \dots A_n$
$P(1)P(n-1)$	$A_1(A_2 \dots A_n)$
$P(2)P(n-2)$	$(A_1A_2)(A_3 \dots A_n)$
$\vdots$	
$P(n-2)P(2)$	$(A_1 \dots A_{n-2})(A_{n-1}A_n)$
$P(n-1)P(1)$	$(A_1 \dots A_{n-1})\,A_n$

$\Omega(2^n)$

حل پایین به بالا و پویا پرانتزگذاری بهینه

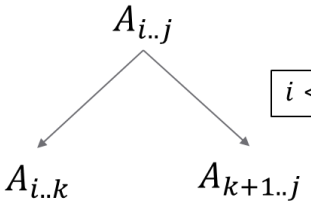
طول زیرمسئله $m[i,j]$: $l = j - i + 1$

برای حل زیرمسئله‌هایی به طول l صرفا نیاز به زیرمسئله‌هایی با طول $1 < i < l$ داریم

Bottom-up approach

شروع به حل از زیرمسئله‌های بطول ۱، ۲ الی n

$$A_{i..j} = A_iA_{i+1} \cdots A_j$$



اگر $i < j$

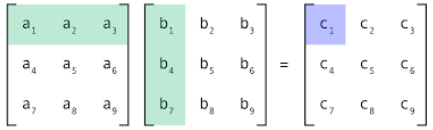
- هزینه محاسبه ماتریس $A_{i..k}$
- هزینه محاسبه ماتریس $A_{k+1..j}$
- هزینه ضرب این دو

جدول هزینه‌ها: $m[1..n, 1..n]$

جدول انتخاب‌ها: $s[1..n-1, 2..n]$

ضرب ماتریسی

- ضرب ماتریسی $C = AB$ دو ماتریس $p \times q$ و $q \times r$



```
MATRIX-MULTIPLY(A, B)
if A.columns ≠ B.rows
    error "incompatible dimensions"
else let C be a new A.rows × B.columns matrix
    for i = 1 to A.rows
        for j = 1 to B.columns
            c_ij = 0
            for k = 1 to A.columns
                c_ij = c_ij + a_ik · b_kj
    return C
```

تعداد ضرب مورد نیاز: $p \times q \times r$

ضرب زنجیره‌ای ماتریس‌ها و پرانتزگذاری

$$(A_1(A_2(A_3A_4)))$$
$$(A_1((A_2A_3)A_4))$$
$$((A_1A_2)(A_3A_4))$$
$$((A_1(A_2A_3))A_4)$$
$$(((A_1A_2)A_3)A_4)$$

انواع پرانتز گذاری ممکن

$$A_1A_2A_3 \cdots A_{s-1}A_s = A_1(A_2(A_3 \cdots (A_{s-1}A_s)))$$

تعداد کل ضرب مورد نیاز بستگی به انتخاب پرانتز گذاری دارد!

چه زمانی می‌توان از برنامه‌نویسی پویا استفاده کرد؟

- تا اینجای درس دو مثال از برنامه‌نویسی پویا دیدیم:

۱) برش میله

۲) ضرب ماتریسی

- سوال مهم و تکراری:

برای حل یک مسئله بهینه‌سازی به روش پویا آن مسئله باید چه ویژگی‌هایی داشته باشد؟

۱) زیرساختار بهینه (optimum substructure)

جواب بهینه یک مسئله از جواب‌های بهینه زیرمسئله‌ها به دست آید

۲) زیرمسئله‌های با هم‌پوشانی با (overlapping subproblems)

فضای جستجوی زیرمسئله‌ها باید کوچک باشد و روش بازگشتی بارها یک زیرمسئله را حل کند

مراحل عمومی کشف زیر ساختار بهینه

۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند

در این مرحله با یک انتخاب مناسب تا می‌توانید زیرمسئله‌های ساده ایجاد کنید و اگر نشد آن را تعمیم دهید

۲) فرض می‌کنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)

۳) با در اختیار داشتن انتخاب مرحله اول تعیین می‌کنیم چه زیرمسئله‌هایی منتج می‌شود

الف) چه تعداد زیرمسئله ایجاد میشود

ب) چه تعداد انتخاب باید امتحان کرد تا به جواب بهینه دست یافت؟

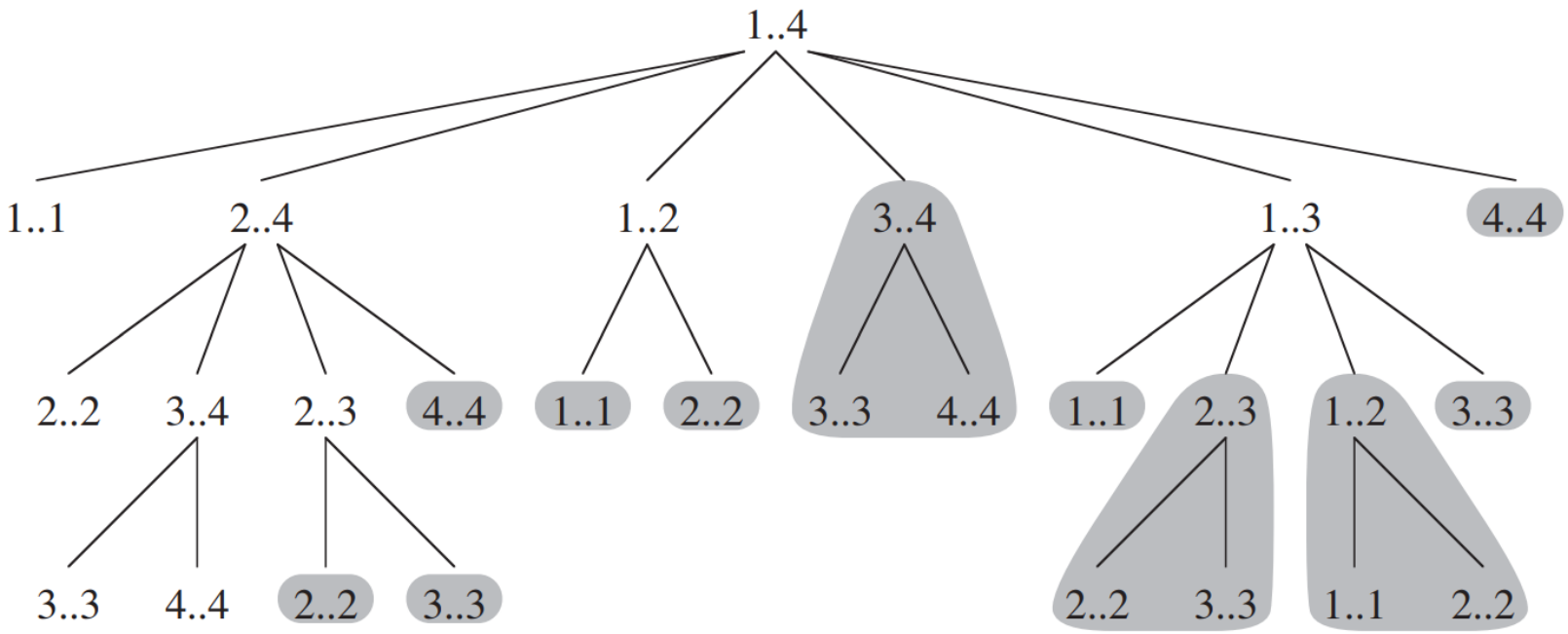
۴) نشان می‌دهیم برای دستیابی به راه حل بهینه می‌بایست جواب زیرمسئله‌ها نیز بهینه باشد

روش: فرض میکنیم جواب زیرمسئله‌ها بهینه نیستند و نتیجه می‌گیریم جواب مسئله نیز بهینه نیست

زمان اجرای غیر رسمی راه حل پویا: مجموع زیر مسئله‌ها $\times$ تعداد انتخاب‌هایی که باید برای یافتن جواب بهینه بررسی شود

زیرمسئله‌های با هم‌پوشانی

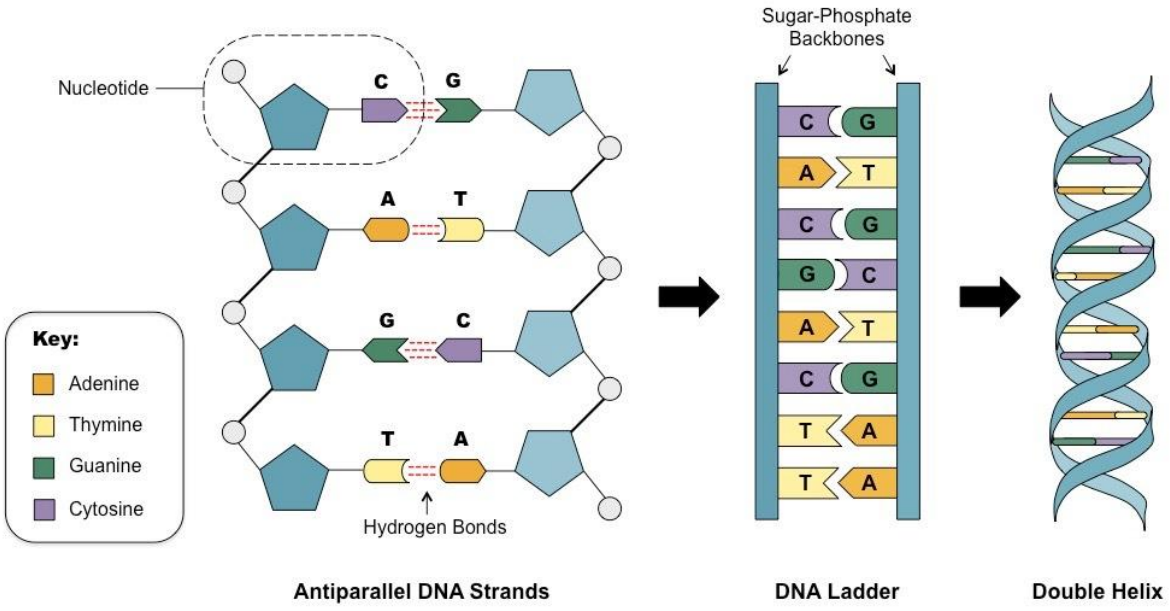
- عموما تعداد زیرمسئله‌ها از مرتبه چندجمله‌ای نسبت به n برخوردارند
- اما وقتی حل به روش بازگشتی نمایی می‌شود ← یعنی هر زیرمسئله چندبار حل شده!
- پس زیرمسئله‌های با هم‌پوشانی داریم



The recursion tree for the computation of RECURSIVE-MATRIX-CHAIN(p, 1, 4).

طولانی‌ترین زیردنباله مشترک LCS

- مثال استخراج شباهت بین DNA



adenine, guanine, cytosine, and thymine

$\{A, C, G, T\}$

$S_1 = \texttt{ACCGGTCGAGTGCGCGGAAGCCGGCCGAA}$

$S_2 = \texttt{GTCGTTCGGAATGCCGTTGCTCTGTAAA}$

$\boxed{\texttt{GTCGTCGGAAGCCGGCCGAA}}$

طولانی‌ترین زیردنباله مشترک LCS

- تعریف علمی زیردنباله:

$X = < x_1, x_2, \dots, x_m >$ $\quad x_i \in S$

$Z = < z_1, z_2, \dots, z_k >$ $\quad z_j \in S$

وجود دارد اندیس‌هایی $i_1 < i_2 < \cdots < i_k$ بطوریکه

برای تمام a در محدوده $1 \leq a \leq k$ داشته باشیم $z_a = x_{i_a}$

- تعریف زیر دنباله مشترک:

فرض کنید X و Y دو دنباله از مجموعه S باشند. در اینصورت دنباله Z درصورتی یک زیردنباله مشترک است که داشته باشیم:

۱) Z زیردنباله X باشد ۲) Z زیردنباله Y باشد

برنامه‌نویسی پویا و حل LCS

پایه‌های برنامه‌نویسی پویا

1. مشخص کردن ساختار یک جواب بهینه
2. تعیین مقدار برای یک جواب بهینه بصورت بازگشتی
3. محاسبه مقدار یک جواب بهینه، معمولا بصورت پایین به بالا
4. جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور

۱. تعیین ساختار جواب بهینه LCS

- راه حل brute-force
- یافتن همه زیردنباله‌های X
- چک کردن آن با Y
- نگهداری بلندترین مشترک‌ها
- تعداد زیردنباله‌های X: 2^m

دنباله‌های ورودی

$$X = < x_1, x_2, \dots, x_m > \quad x \in S$$

$$Y = < y_1, y_2, \dots, y_n > \quad y \in S$$

LCS دنباله‌های ورودی

$$Z = < z_1, z_2, \dots, z_k > \quad z \in S$$

آیا ویژگی زیرساختار بهینه وجود دارد؟

---

۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند

۲) فرض میکنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)

۳) با در اختیار داشتن انتخاب مرحله اول تعیین میکنیم چه زیرمسئله‌هایی منتج میشود

۴) نشان میدهیم برای دستیابی به راه حل بهینه میبایست جواب زیرمسئله‌ها نیز بهینه باشد

۱. تعیین ساختار جواب بهینه LCS

$$X = < x_1, x_2, \dots, x_m > \qquad x \in S$$
$$Y = < y_1, y_2, \dots, y_n > \qquad y \in S$$

$$Z = < z_1, z_2, \dots, z_k > \qquad z \in S$$

آیا ویژگی زیرساختار بهینه وجود دارد؟

۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند

۲) فرض میکنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)

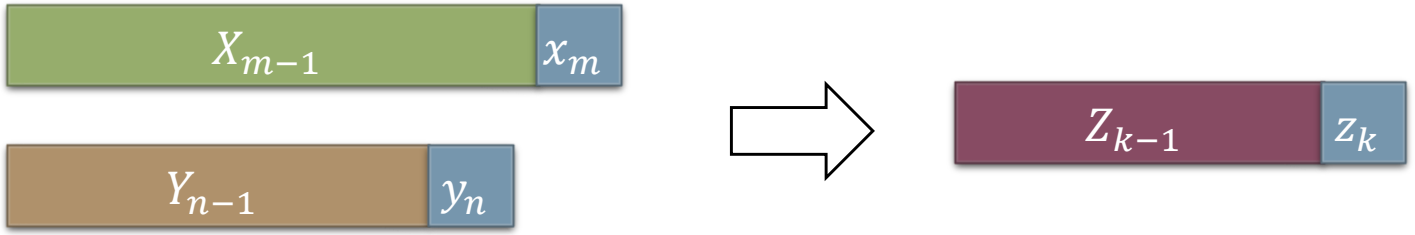
۳) با در اختیار داشتن انتخاب مرحله اول تعیین میکنیم چه زیرمسئله‌هایی منتج میشود

۴) نشان میدهیم برای دستیابی به راه حل بهینه میبایست جواب زیرمسئله‌ها نیز بهینه باشد

پیشوند i ام X

$$X = \langle x_1, x_2, \dots, x_m \rangle$$
$$X_i = \langle x_1, x_2, \dots, x_i \rangle$$

$$X = \langle A, B, C, B, D, A, B \rangle$$
$$X_4 = \langle A, B, C, B \rangle$$
$$X_0 = \langle \ \rangle$$



1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.

۱. تعیین ساختار جواب بهینه LCS

$$X = < x_1, x_2, \dots, x_m > \qquad x \in S$$
$$Y = < y_1, y_2, \dots, y_n > \qquad y \in S$$

$$Z = < z_1, z_2, \dots, z_k > \qquad z \in S$$

آیا ویژگی زیرساختار بهینه وجود دارد؟

۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند

۲) فرض میکنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)

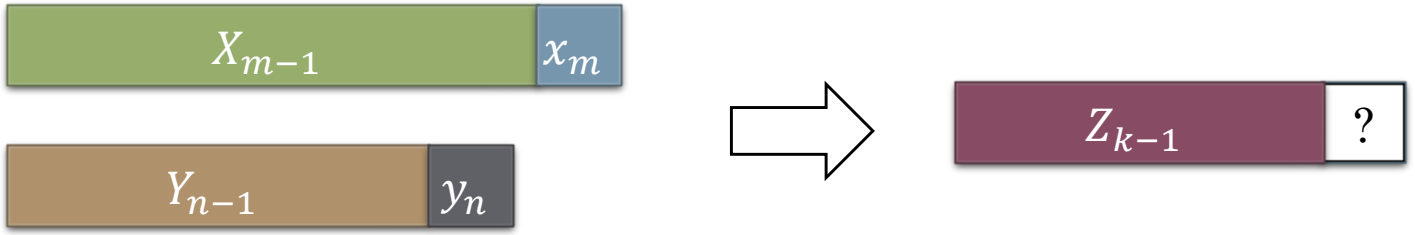
۳) با در اختیار داشتن انتخاب مرحله اول تعیین میکنیم چه زیرمسئله‌هایی منتج میشود

۴) نشان میدهیم برای دستیابی به راه حل بهینه میبایست جواب زیرمسئله‌ها نیز بهینه باشد

پیشوند i ام X

$$X = \langle x_1, x_2, \dots, x_m \rangle$$
$$X_i = \langle x_1, x_2, \dots, x_i \rangle$$

$$X = \langle A, B, C, B, D, A, B \rangle$$
$$X_4 = \langle A, B, C, B \rangle$$
$$X_0 = \langle \ \rangle$$



2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

۱. تعیین ساختار جواب بهینه LCS

$$X = < x_1, x_2, \dots, x_m > \qquad x \in S$$

$$Y = < y_1, y_2, \dots, y_n > \qquad y \in S$$

$$Z = < z_1, z_2, \dots, z_k > \qquad z \in S$$

آیا ویژگی زیرساختار بهینه وجود دارد؟

۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند

۲) فرض میکنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)

۳) با در اختیار داشتن انتخاب مرحله اول تعیین میکنیم چه زیرمسئله‌هایی منتج میشود

۴) نشان میدهیم برای دستیابی به راه حل بهینه میبایست جواب زیرمسئله‌ها نیز بهینه باشد

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

۲. تعیین یک جواب بازگشتی برای LCS

$X = < x_1, x_2, \dots, x_m > \quad x \in S$

$Y = < y_1, y_2, \dots, y_n > \quad y \in S$

$Z = < z_1, z_2, \dots, z_k > \quad z \in S$

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

$c[i, j]$	the length of an LCS of the sequences X_i and Y_j

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

۳. محاسبه جواب بهینه پایین به بالا!

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

فضای جستجوی زیرمسئله‌ها

$$O(2^{min(m,n)}) \leq O(T) \leq O(2^{n+m})$$

$$T(m,n) = T(m,n-1) + T(m-1,n)$$

فضای جستجوی زیرمسئله‌ها

$$\Theta(mn)$$

۳. محاسبه جواب بهینه پایین به بالا!

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

$b[i,j]$: ⟨ "←" "↑" "↖" ⟩

(3) (2) (1)

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

۴. بازسازی جواب بهینه!

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

$b[i,j]$: ⟨ "←" "↑" "↖" ⟩

(3) (2) (1)

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

مثال

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

$$X = \langle A, B, C, B, D, A, B \rangle$$

$$Y = \langle B, D, C, A, B, A \rangle$$

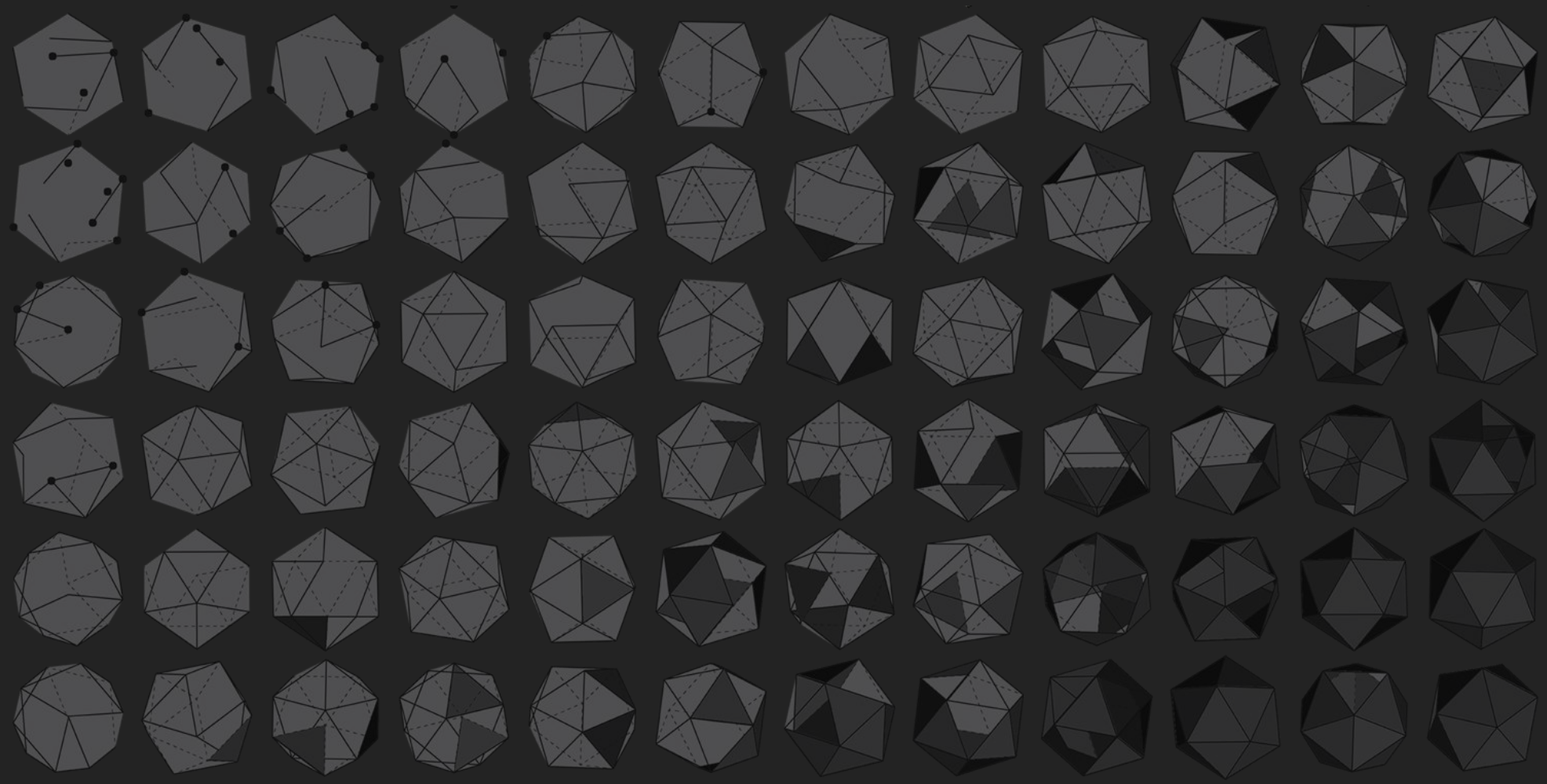
	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i							
1	A							
2	B							
3	C							
4	B							
5	D							
6	A							
7	B							

تحلیل زمان اجرا

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

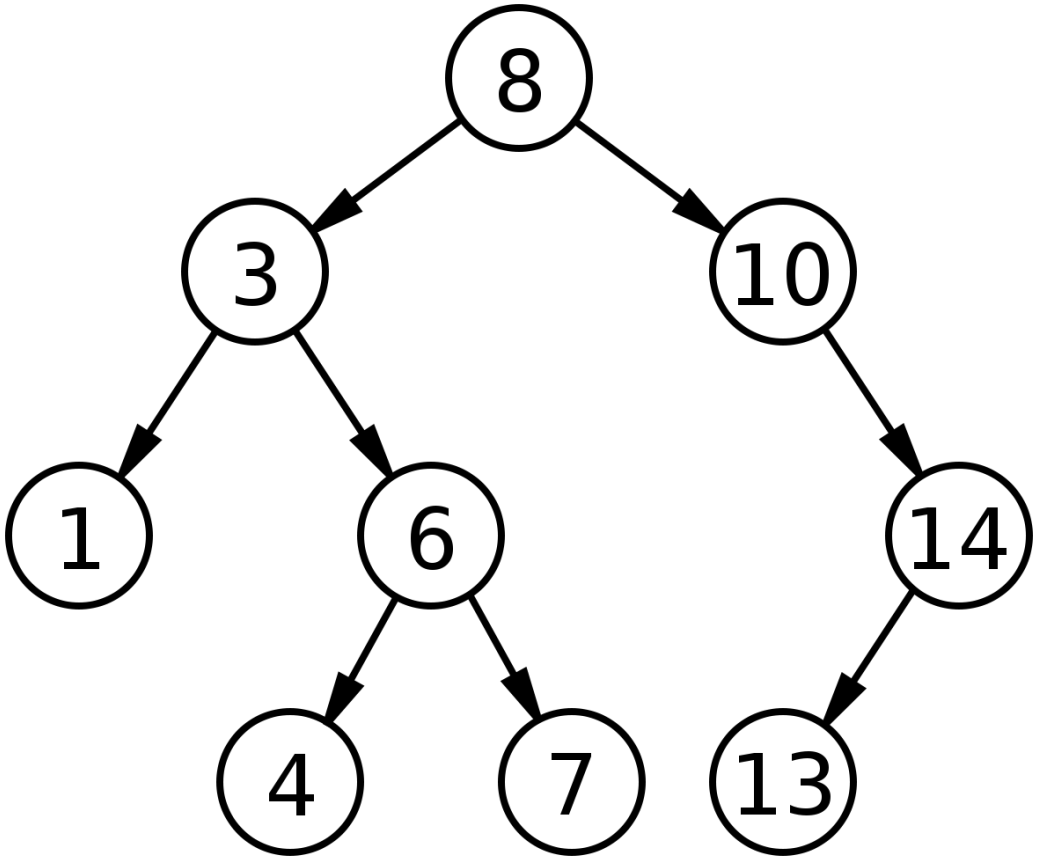
```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

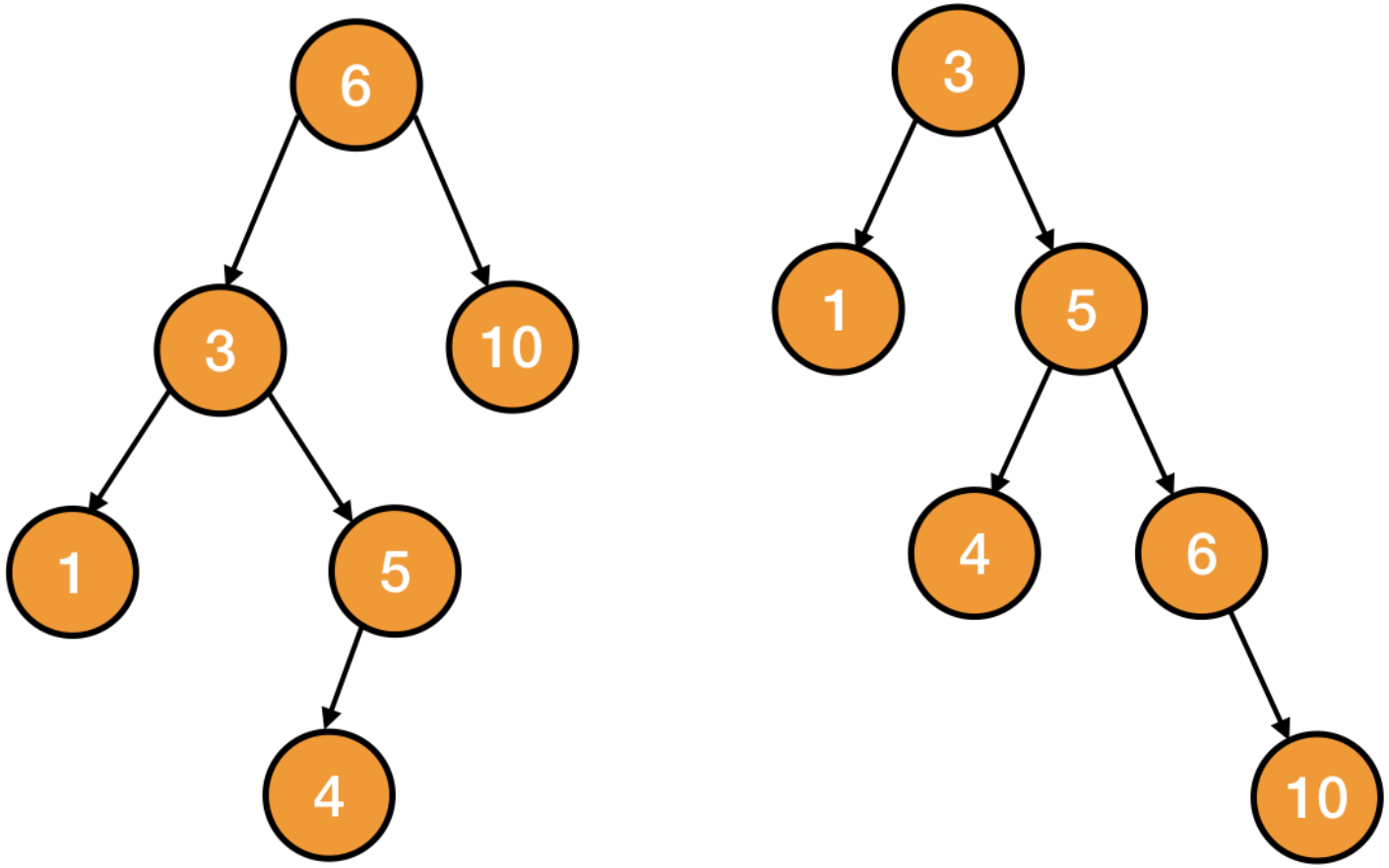


درخت دودویی جستجو BST

درخت دودویی جستجوی



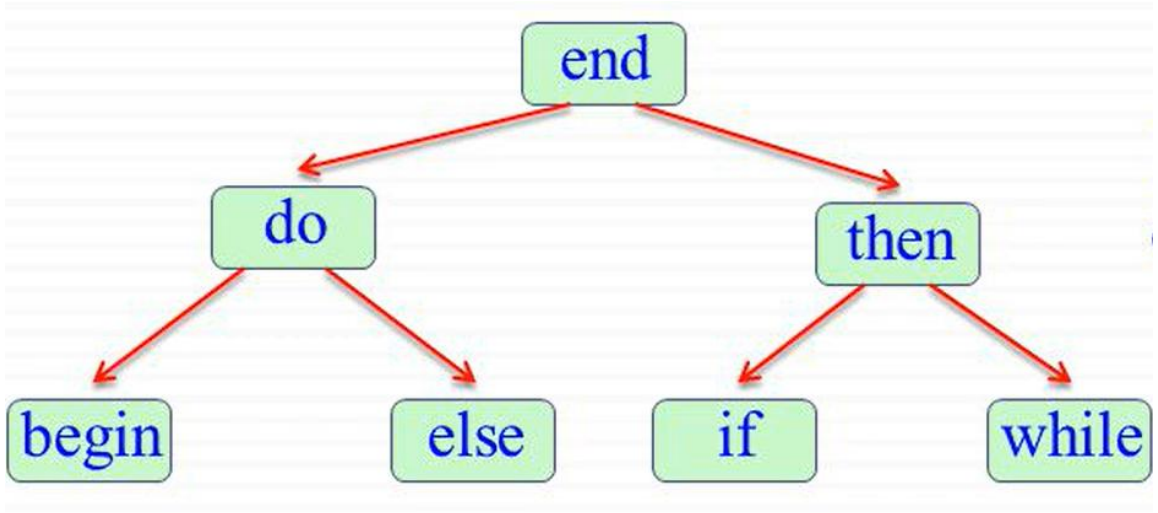
درخت دودویی جستجوی



Two Binary Search Trees Representing Same Set

مثال دیکشنری انگلیسی به فرانسه با BST

- مثال درخت دودویی جستجوی: ترجمه انگلیسی به فرانسوی
- ایجاد درخت دودویی جستجو با جفت لغت‌های انگلیسی و ترجمه فرانسه
  - key: لغت انگلیسی
  - data: ترجمه فرانسوی لغت key

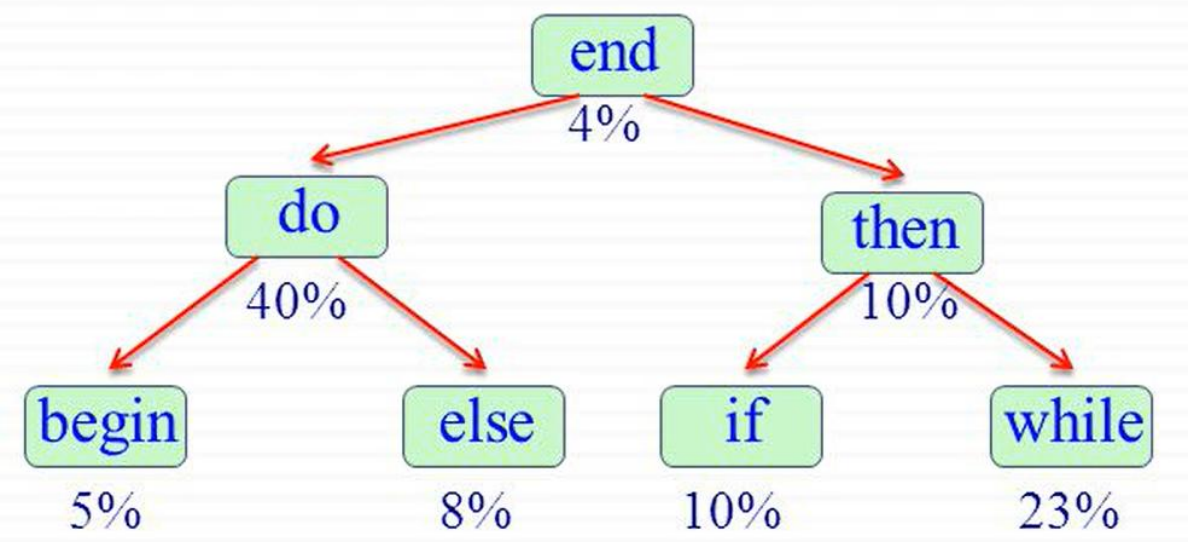


A B C D E F G H I
J K L M N O P Q
R S T U V W X Y Z

هزینه کل جستجو در دیکشنری

- فرض کنید که درصد رخداد لغت‌های انگلیسی در متن را بدانیم

begin	do	else	end	if	then	while
5%	40%	8%	4%	10%	10%	23%

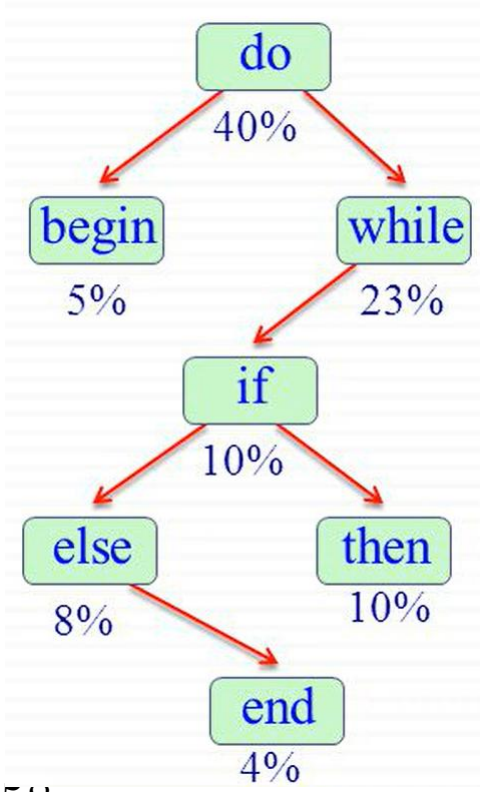


$$\boxed{Total\ cost = \sum_{i \in English\ words} [(depth(i) + 1) \times freq(i)]}$$

$$= 1 \times .04 + 2 \times .4 + 2 \times .1 + 3 \times .05 + 3 \times .08 + 3 \times .1 + 3 \times .23$$
$$= 2.42$$

درخت جستجوی دودویی بهینه

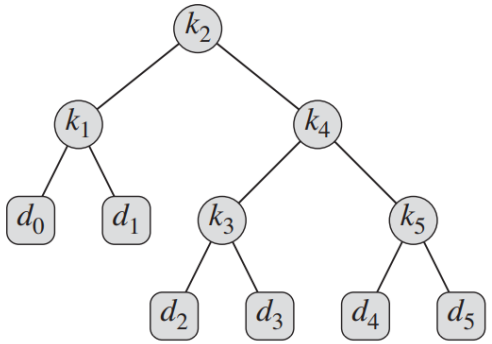
- ورودی: n عدد key مجزا مرتب شده صعودی $K = \langle k_1, k_2, \ldots, k_n \rangle$ $k_1 < k_2 < \cdots < k_n$
- برای هر k_i، احتمال p_i معرف احتمال جستجوی key
- خروجی: درخت جستجوی دودویی بهینه با کمترین هزینه جستجوی کل

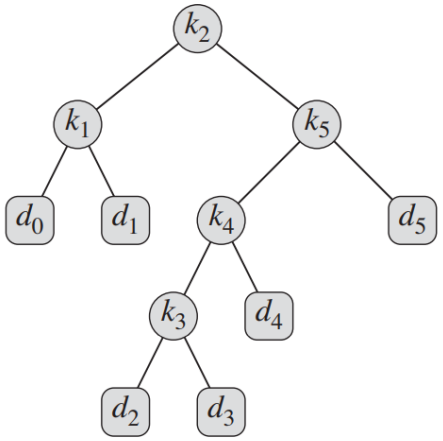


$$Total\ cost = 1 \times .4 + 2 \times .05 + 2 \times .23 + 3 \times .1 + 4 \times .08 + 4 \times .1 + 5 \times .04 = 2.18$$

درخت جستجوی دودویی بهینه

- ورودی: n عدد key مجزا مرتب شده صعودی $K = \langle k_1, k_2, \ldots, k_n \rangle$ $k_1 < k_2 < \cdots < k_n$
- برای هر k_i، احتمال p_i معرف احتمال جستجوی key
- خروجی: درخت جستجوی دودویی بهینه با کمترین هزینه جستجوی کل
- برخی جستجوها شاید در مجموعه K وجود نداشته باشد
- نیاز به $n + 1$ کلید بعنوان $dummy\ key$ $D = \langle\ d_0, d_1, d_2, \ldots, d_n\ \rangle$
- d_i معرف همه keyهایی که بین k_i و k_{i+1} قراردارد





$$K = \langle k_1, k_2, \ldots, k_n \rangle$$
$$D = \langle\ d_0, d_1, d_2, \ldots, d_n\ \rangle$$

درخت جستجوی دودویی بهینه

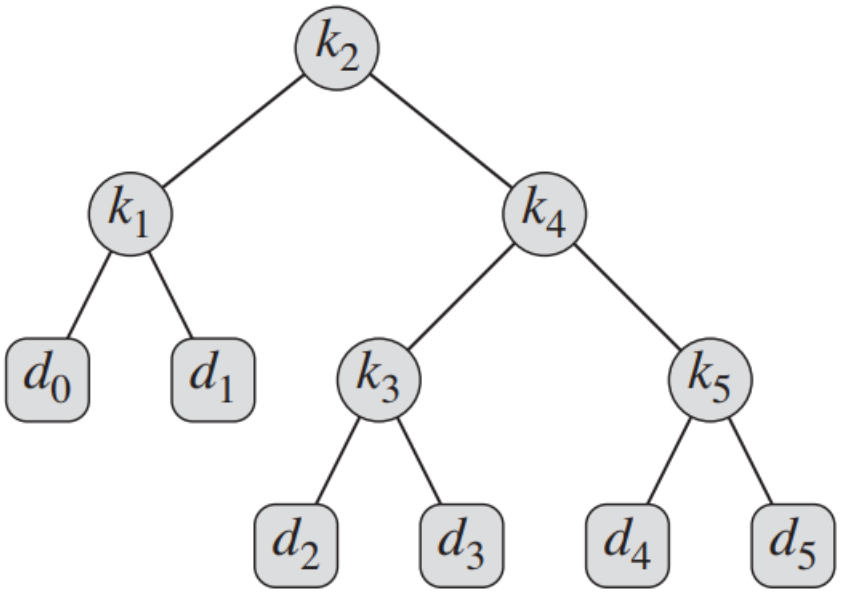
- ورودی: n عدد key مجزا مرتب شده صعودی $K = \langle k_1, k_2, \ldots, k_n \rangle$ $k_1 < k_2 < \cdots < k_n$
- برای هر k_i، احتمال p_i معرف احتمال جستجوی key
- $n+1$ عدد $dummy\ key$ $D = \langle\ d_0, d_1, d_2, \ldots, d_n\ \rangle$
- برای هر d_i، احتمال q_i معرف احتمال جستجوی $dummy\ key$

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

- خروجی: درخت جستجوی دودویی بهینه با کمترین هزینه جستجوی کل

$$\begin{aligned} \mathrm{E}\,[\text{search cost in } T] &= \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \mathrm{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_T(d_i) \cdot q_i \end{aligned}$$

درخت جستجوی دودویی بهینه

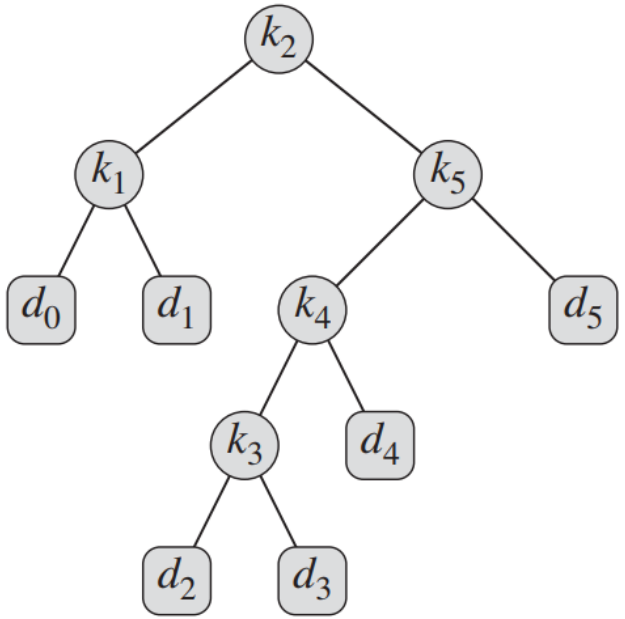


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

درخت جستجوی دودویی بهینه

Optimal binary search tree



i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

node	depth	probability	contribution
k_1	1	0.15	0.3
k_2	0	0.10	0.1
k_3	3	0.05	0.2
k_4	2	0.10	0.3
k_5	1	0.20	0.4
d_0	2	0.05	0.15
d_1	2	0.10	0.3
d_2	4	0.05	0.25
d_3	4	0.05	0.25
d_4	3	0.05	0.2
d_5	2	0.10	0.3
Total			2.75

- Overall height is not the smallest
- Key with the greatest probability is not at the root

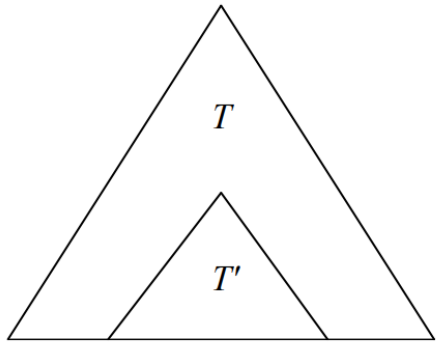
تعیین BST بهینه به روش برنامه‌نویسی پویا

Problem 12-4

- تعداد همه BST های ممکن با n گره نمایی است.

۱) تعیین ساختار بهینه

- هر زیردرختی در BST می‌بایست شامل keyهای پشت‌سرهم باشد $k_i, \ldots, k_j$ $1 \le i \le j \le n$
- درختی با keyهای $k_i, \ldots, k_j$ حتما شامل $dummy\ key$های $d_{i-1}, \ldots, d_j$ نیز هست



If T is an optimal BST and T contains subtree T' with keys $k_i, \ldots, k_j$, then T' must be an optimal BST for keys $k_i, \ldots, k_j$.

Proof Cut and paste. ■

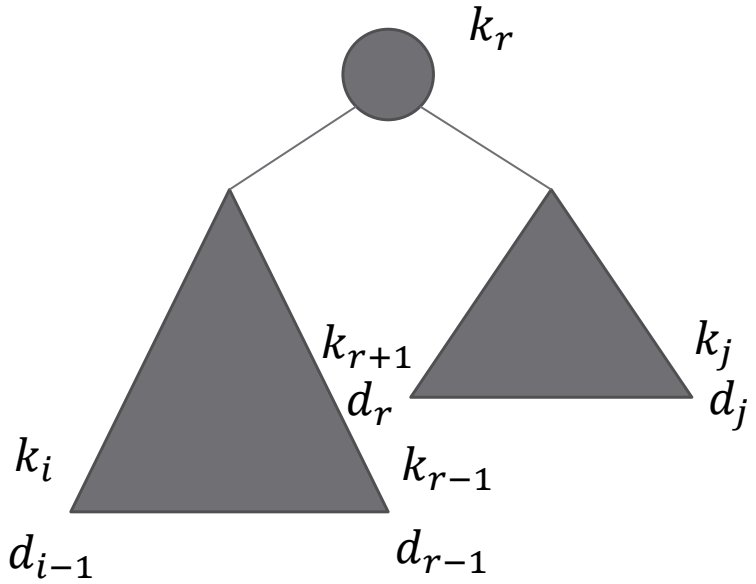
تعیین BST بهینه به روش برنامه‌نویسی پویا

Problem 12-4

- تعداد همه BST های ممکن با n گره نمایی است.

۱) تعیین ساختار بهینه

- هر زیردرختی در BST می‌بایست شامل keyهای پشت‌سرهم باشد $k_i, \ldots, k_j$ $1 \le i \le j \le n$
- درختی با keyهای $k_i, \ldots, k_j$ حتما شامل $dummy\ key$های $d_{i-1}, \ldots, d_j$ نیز هست



۱) انتخاب: تعیین ریشه درخت بهینه k_r

۲) در اختیار داشتن جواب برای زیردخت‌های بهینه

۳) تعیین زیرمسئله‌ها

۴) اثبات زیرساختار بهینه

- Given keys $k_i, \ldots, k_j$ (the problem).
- One of them, k_r, where $i \le r \le j$, must be the root.
- Left subtree of k_r contains $k_i, \ldots, k_{r-1}$.
- Right subtree of k_r contains $k_{r+1}, \ldots, k_j$.

تعیین BST بهینه به روش برنامه‌نویسی پویا

۲) تعیین مقدار یک جواب بهینه بصورت بازگشتی

$k_i, \ldots, k_j \quad i \geq 1,\ j \leq n,\ \text{and}\ j \geq i-1.$

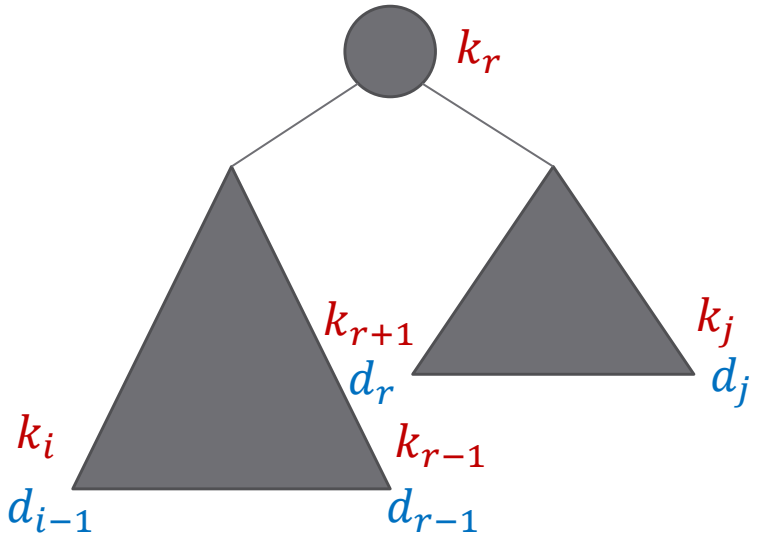
$j = i - 1$

هیچ key ای در درخت وجود ندارد!
(ولی یک dummy d_{i-1} وجود دارد)

$e[i, j]$ امید هزینه جستجو در درخت بهینه شامل $k_i, \ldots, k_j$

$e[1, n]$ امید هزینه جستجو در درخت بهینه شامل همه keyها

$e[i, i-1] = q_{i-1}$



$$e[i, j] = p_r + (e[i, r-1] + w(i, r-1)) + (e[r+1, j] + w(r+1, j))$$

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

تعیین BST بهینه به روش برنامه‌نویسی پویا

۲) تعیین مقدار یک جواب بهینه بصورت بازگشتی

$e[i, j]$ امید هزینه جستجو در درخت بهینه شامل $k_i, \ldots, k_j$

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

$$e[i, j] = p_r + (e[i, r-1] + w(i, r-1)) + (e[r+1, j] + w(r+1, j))$$

$$w(i, j) = w(i, r-1) + p_r + w(r+1, j)$$

$$e[i, j] = e[i, r-1] + e[r+1, j] + w(i, j)$$

$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i-1 , \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j . \end{cases}$$

تعیین BST بهینه به روش برنامه‌نویسی پویا

۳) محاسبه کمترین هزینه بصورت پایین به بالا

```
OPTIMAL-BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
for i = 1 to n+1
    e[i, i-1] = q_{i-1}
    w[i, i-1] = q_{i-1}
for l = 1 to n
    for i = 1 to n-l+1
        j = i+l-1
        e[i, j] = ∞
        w[i, j] = w[i, j-1] + p_j + q_j
        for r = i to j
            t = e[i, r-1] + e[r+1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] = t
                root[i, j] = r
return e and root
```

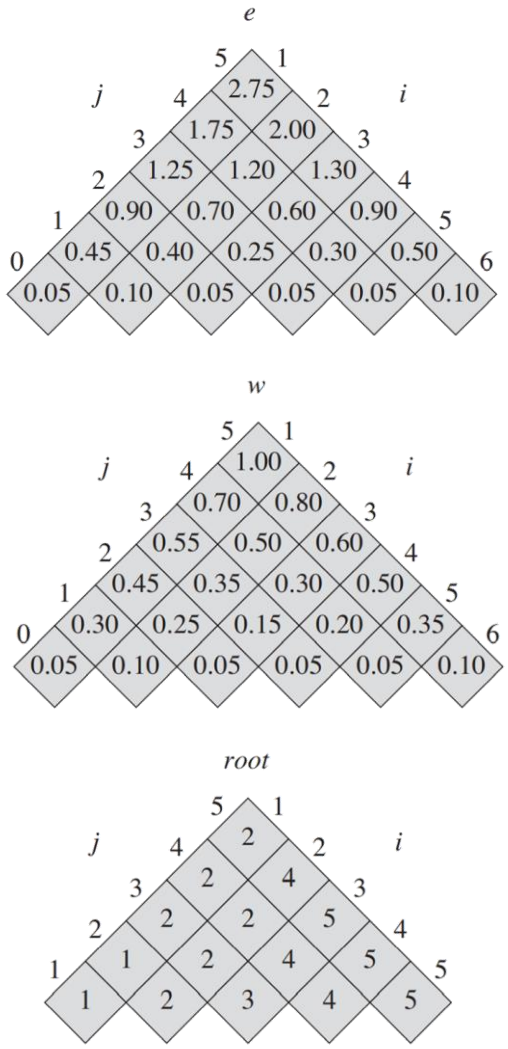
تعیین BST بهینه به روش برنامه‌نویسی پویا

```
OPTIMAL-BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
for i = 1 to n+1
    e[i, i-1] = q_{i-1}
    w[i, i-1] = q_{i-1}
for l = 1 to n
    for i = 1 to n-l+1
        j = i+l-1
        e[i, j] = ∞
        w[i, j] = w[i, j-1] + p_j + q_j
        for r = i to j
            t = e[i, r-1] + e[r+1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] = t
                root[i, j] = r
return e and root
```

node	probability
k_1	0.15
k_2	0.10
k_3	0.05
k_4	0.10
k_5	0.20
d_0	0.05
d_1	0.10
d_2	0.05
d_3	0.05
d_4	0.05
d_5	0.10

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 , \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j . \end{cases}$$



بازسازی BST بهینه

Construct an optimal solution

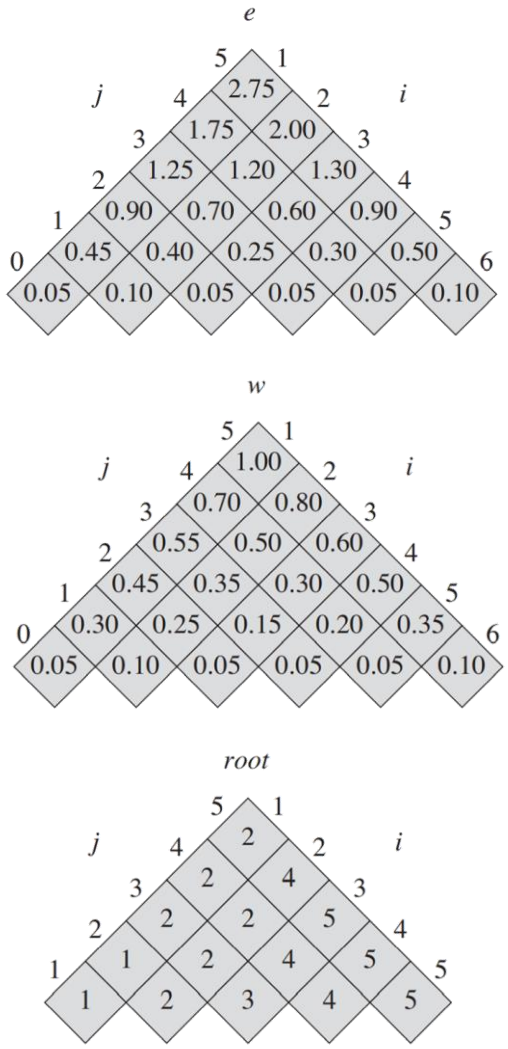
CONSTRUCT-OPTIMAL-BST($root$)

$r = root[1, n]$
print "k"$_r$ "is the root"
CONSTRUCT-OPT-SUBTREE$(1, r-1, r,$ "left", $root)$
CONSTRUCT-OPT-SUBTREE$(r+1, n, r,$ "right", $root)$

CONSTRUCT-OPT-SUBTREE$(i, j, r, dir, root)$

if $i \le j$
$t = root[i, j]$
print "k"$_t$ "is" dir "child of k"$_r$
CONSTRUCT-OPT-SUBTREE$(i, t-1, t,$ "left", $root)$
CONSTRUCT-OPT-SUBTREE$(t+1, j, t,$ "right", $root)$

node	probability
k_1	0.15
k_2	0.10
k_3	0.05
k_4	0.10
k_5	0.20
d_0	0.05
d_1	0.10
d_2	0.05
d_3	0.05
d_4	0.05
d_5	0.10

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i-1 , \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j . \end{cases}$$