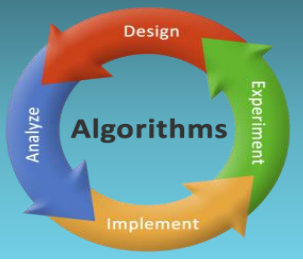
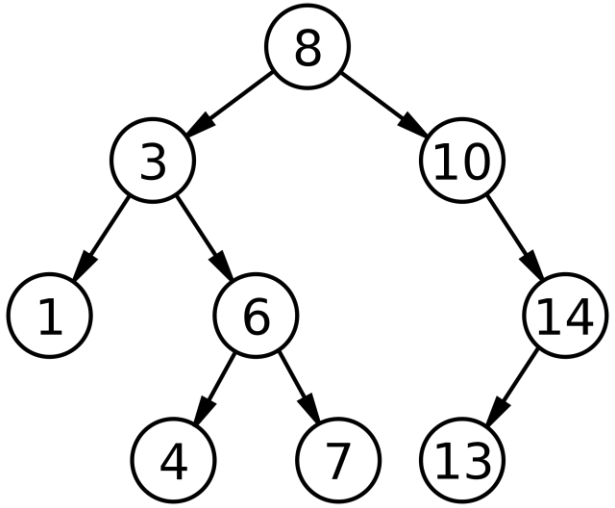
برنامه‌نویسی پویا

درخت جستجوی دودویی بهینه

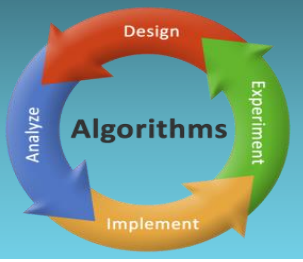
درخت جست‌وجوی دودویی



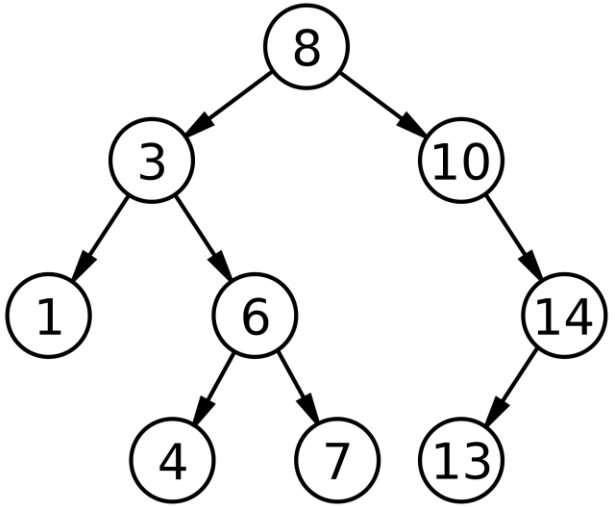
- درخت جست‌وجوی دودویی یا Binary Search Tree (BST)، یک درخت دودویی است که در هر گره داریم:
  - ✓ مقدار تمام عناصر در زیردرخت سمت راست بزرگتر از گره والد است،
  - ✓ و مقدار تمام عناصر در زیردرخت سمت چپ کمتر از گره والد است.



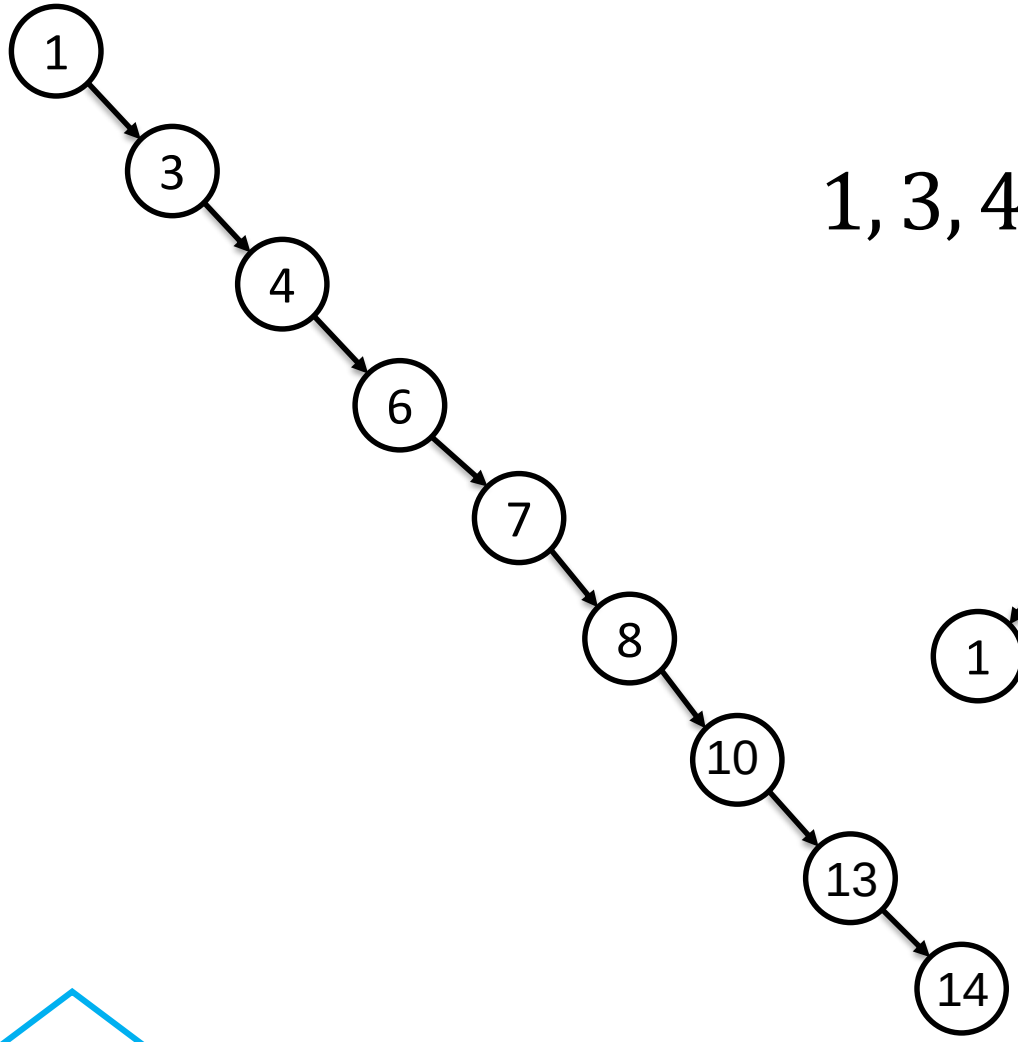
جست‌وجو در یک BST




```
TREE-SEARCH(x, k)
1   if x == NIL or k == x.key
2         return x
3   if k < x.key
4         return TREE-SEARCH(x.left, k)
5   else return TREE-SEARCH(x.right, k)
```

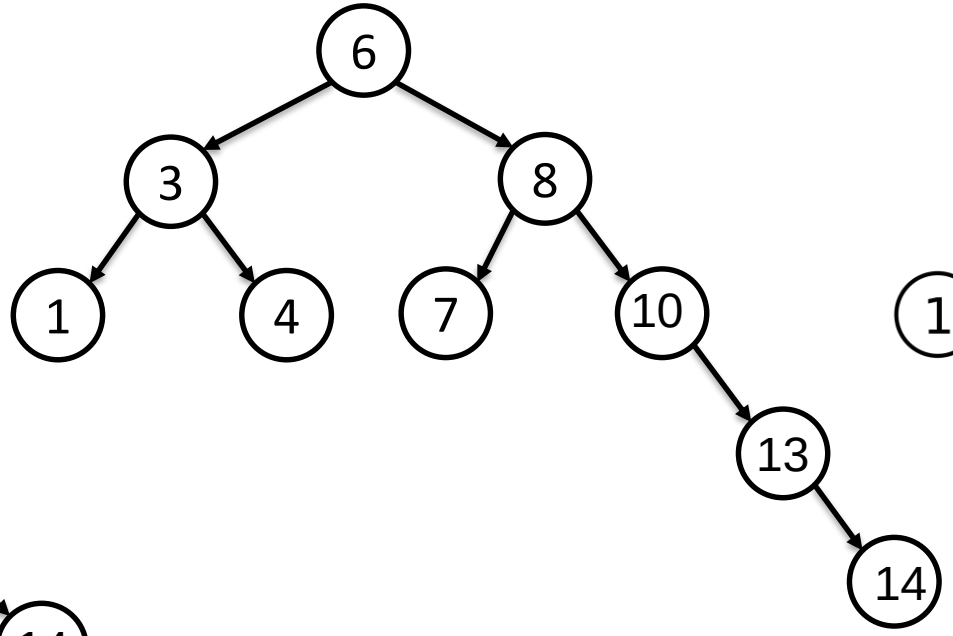


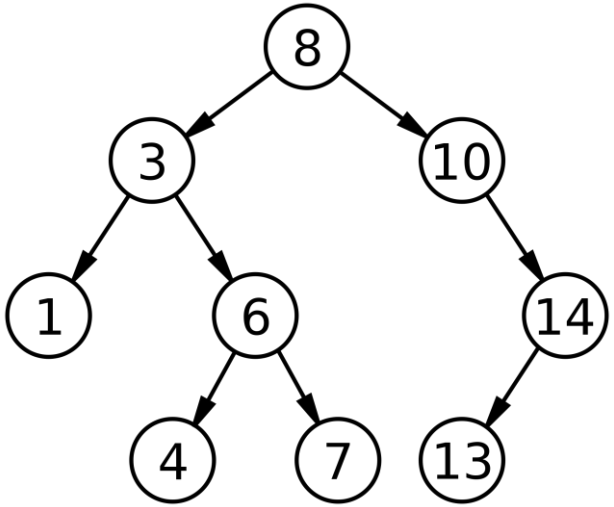
تنوع BST



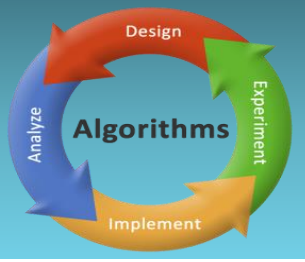
- برای هر مجموعه از کلیدها BSTهای متنوعی قابل ساخت است:

1, 3, 4, 6, 7, 8,10, 13 , 14





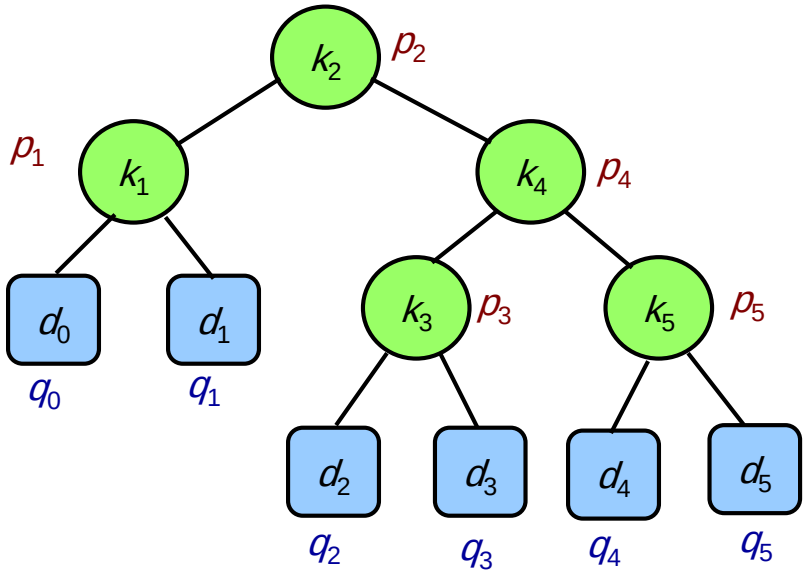
یک مسئله



- حال تصور کنید یک متن به زبان انگلیسی داریم و می‌خواهیم به صورت کلمه‌به‌کلمه به فارسی ترجمه کنیم.
- کلمات زبان انگلیسی به همراه معادل فارسی‌شان در یک BST ذخیره شده‌اند. در واقع درختی مانند یک فرهنگ‌لغت داریم.
- کلمات هر زبان با فراوانی‌های مختلفی در متون آن زبان ظاهر می‌شوند.
  - ✓ به عنوان مثال کلمه "است" در زبان فارسی در هر متنی چندین بار تکرار می‌شود.
  - ✓ ولی کلمه "دریانورد" را به ندرت خواهیم دید.
- فرض کنید احتمال رخداد هر کلمه انگلیسی برای ما مشخص است. (چگونه؟)
- هنگام ترجمه، با دیدن هر کلمه انگلیسی در یک متن، باید آن را در فرهنگ‌لغت بیابیم (جستجو در BST).
- می‌دانیم برای هر مجموعه کلید (کلمه) BSTهای مختلفی می‌توان ساخت. برای این مسئله کدام درخت بهترین است؟
  - ✓ بدیهی است که برای بهبود سرعت در جست‌وجو بهتر است کلماتی که تکرار بیشتری دارند، نزدیکتر به ریشه قرارداده شوند.

- حال مسئله ما این است که چگونه ساختار دقیق درخت را مشخص کنیم؟

معرفی نمادها



- مجموعه کلیدها (کلمات):

$$K = \langle k_1, k_2, \dots, k_n \rangle, \qquad k_1 < k_2, < \cdots < k_n$$

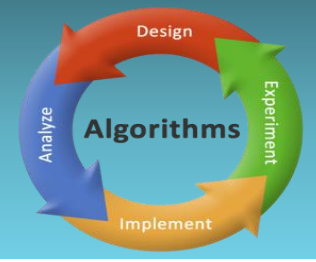
- احتمال رخداد کلمات در زبان:

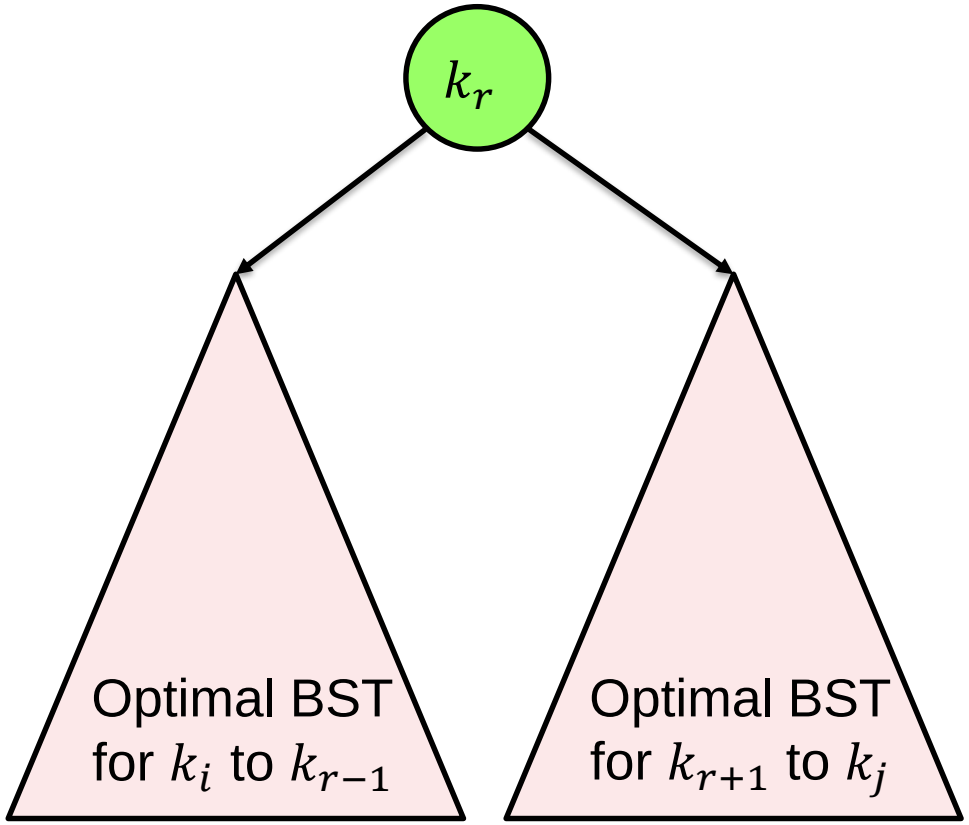
$$P = \langle p_1, p_2, \dots, p_n \rangle$$

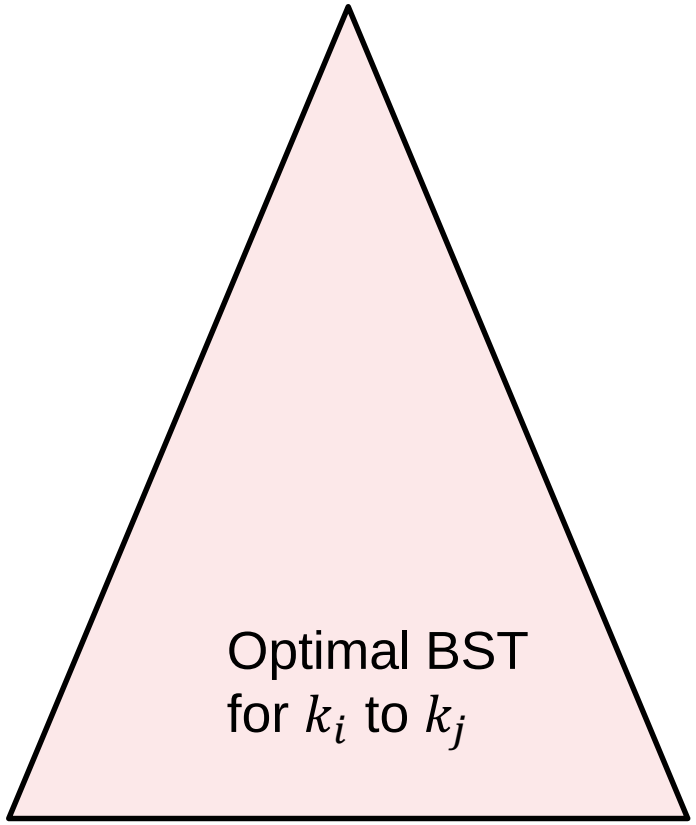
- کلیدهای ساختگی d_0 تا d_n برای جستجوهای ناموفق در نظر گرفته شده‌اند و دارای احتمالات q_0 تا q_n تا هستند.

$$D = \langle d_0, d_2, \dots, d_n \rangle, \qquad Q = \langle q_0, q_2, \dots, q_n \rangle$$

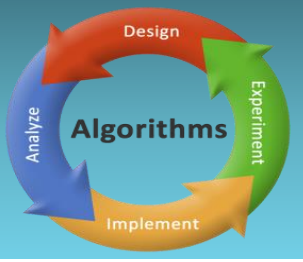
ماهیت بازگشتی مسئله





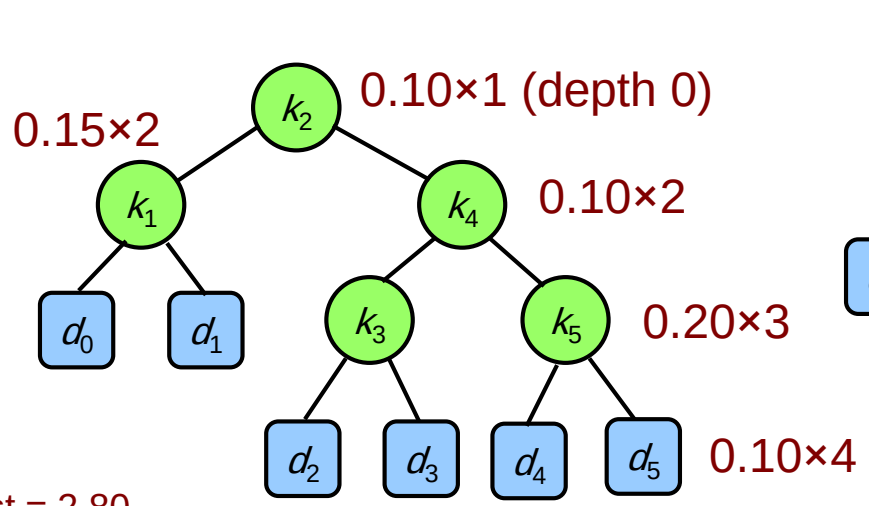


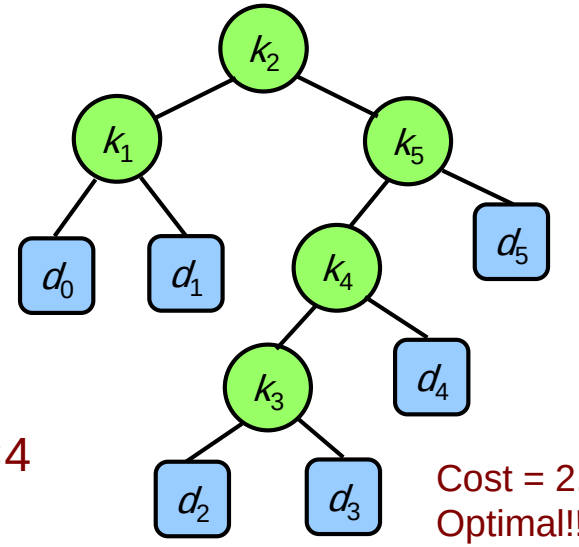
محاسبه هزینه



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

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

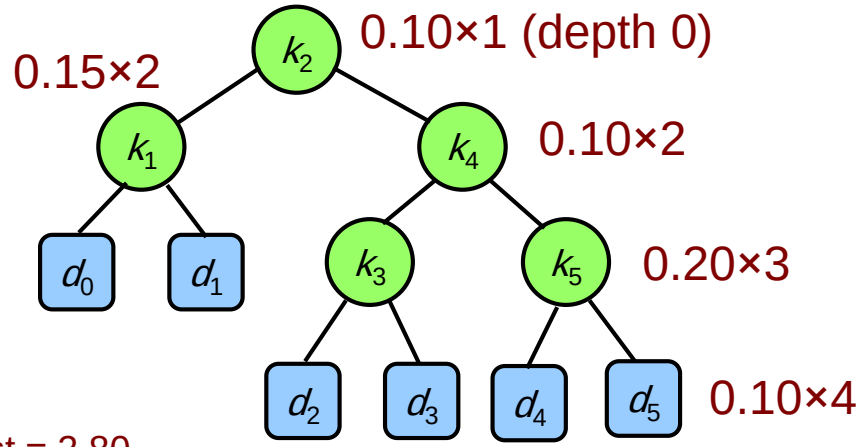




$$E[search\ cost\ in\ T] = \sum_{i=1}^{n} (depth(k_i) + 1).p_i + \sum_{i=0}^{n} (depth(d_i) + 1).q_i$$

$$= 1 + \sum_{i=1}^{n} depth(k_i).p_i + \sum_{i=0}^{n} depth(d_i).q_i$$

محاسبه هزینه با جزئیات بیشتر



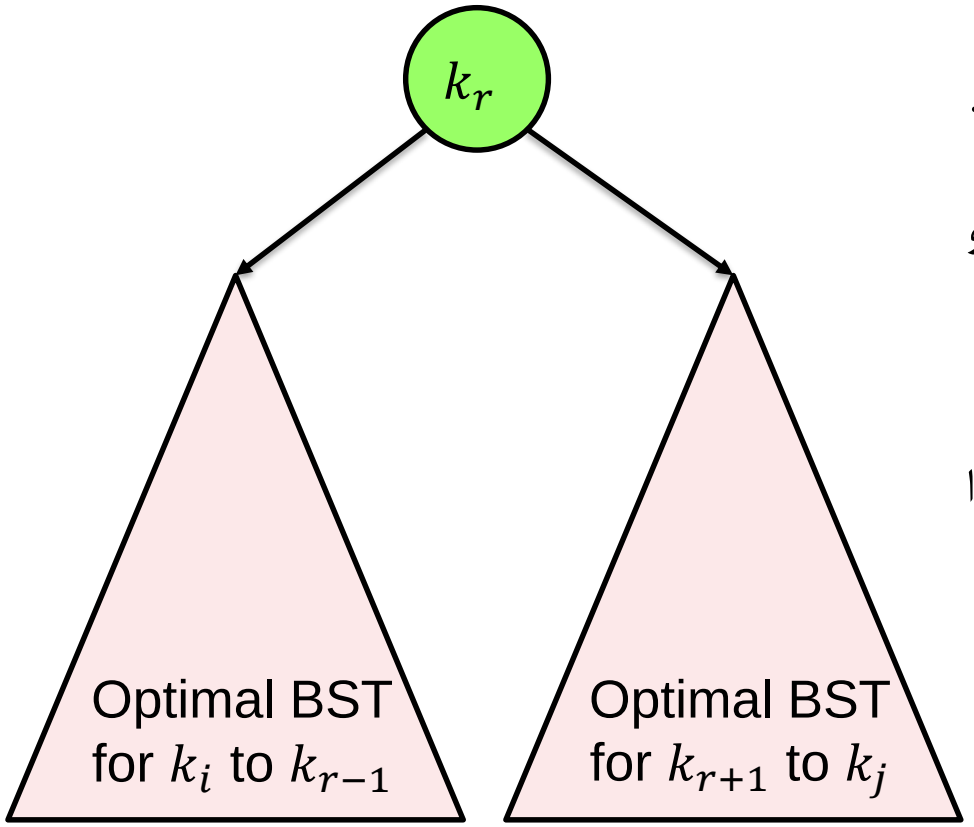
Cost = 2.80

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

(a)

راه‌حل بازگشتی

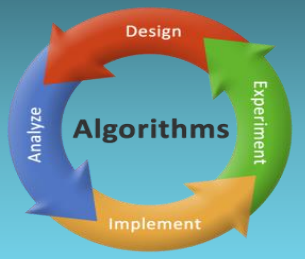


- برای اینکه درخت جستوجو، بهینه باشد باید هر شاخه(زیر درخت) آن نیز بهینه باشد.
از این نظر می‌توانیم مسئله را از پایین به بالا حل کنیم.
- $e[i,j]$ را اینگونه تعریف می‌کنیم که هزینه‌ی درخت بهینه‌ی حاوی مقادیر k_i تا k_j است.
در نهایت هزینه‌ی کل در $e[1,n]$ خواهد بود.
- حالت پایه زمانی اتفاق می‌افتد که درخت شامل هیچ کلیدی نباشد یعنی $j = i - 1$ و آنگاه:

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

- وقتی $j > i$ باشد باید k_r را از بین k_i تا k_j به نحوی پیدا کرد کمترین هزینه جستجو را داشته باشیم.
- فعلا فرض می‌کنیم که r را پیدا کرده‌ایم و جواب بهینه زیرمسائل را داریم.

راه‌حل بازگشتی



پس از پیدا کردن r حالا باید هزینه درخت جدید را حساب کنیم، می‌دانیم:

$$E[a\ tree\ of\ keys\ from\ k_i to\ k_j] = \sum_{l=i}^{j}(depth(k_l)+1).p_l + \sum_{l=i-1}^{j}(depth(d_l)+1).q_l$$

پس داریم:

$$e[i,j] = p_r + \left(e[i,r-1] + \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l\right) + \left(e[r+1,j] + \sum_{l=r+1}^{j} p_l + \sum_{l=r}^{j} q_l\right)$$

اگر $w(i,j)$ مجموع احتمالات یک درخت حاوی گره‌های k_i تا k_j به صورت زیر تعریف می‌شود:

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

بعد از ادغام، ارتفاع هر گره ۱ واحد زیاد می‌شود.

خواهیم داشت:

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

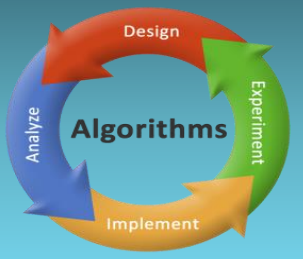
همچنین:

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

پس:

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

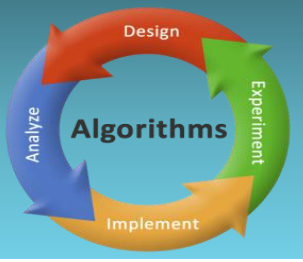
راه‌حل بازگشتی



بخاطر اینکه نمی‌دانیم r برابر چند است داری:

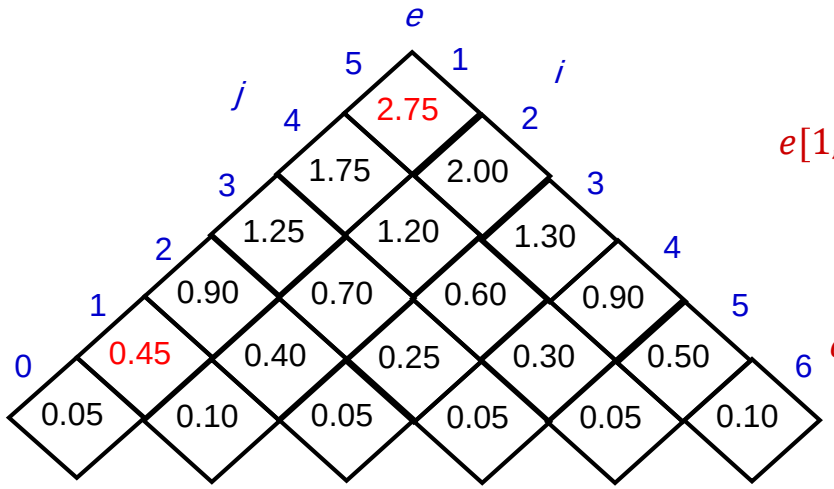
$$e[i,j] = \begin{cases} q_{i-1} & j = i-1 \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & j \ge i \end{cases}$$

الگوریتم برنامه‌نویسی پویا




```
OPTIMAL-BST(p, q, n)
 1  let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
 2  for i = 1 to n + 1
 3      e[i, i − 1] = q_{i−1}
 4      w[i, i − 1] = q_{i−1}
 5  for l = 1 to n
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          e[i, j] = ∞
 9          w[i, j] = w[i, j − 1] + p_j + q_j
10          for r = i to j
11              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
12              if t < e[i, j]
13                  e[i, j] = t
14                  root[i, j] = r
15  return e and root
```

Example



$$e[1,5] = e[1,1] + e[3,5] + w(1,5)$$
$$= 0.45 + 1.30 + 1.00 = 2.75$$

$$e[1,1] = e[1,0] + e[2,1] + w(1,1)$$
$$= 0.05 + 0.10 + 0.3 = 0.45$$

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