Dynamic Programming

- Several problems
- Principle of dynamic programming
  - Structure analysis of optimal solutions
  - Defining values of optimal solutions
  - Top-down or bottom-up computation of values
  - Computation of optimal solution from computed values
- Longest Common Subsequences
- Optimal binary search trees

Longest Common Subsequence

- ***Problem:*** Given 2 sequences, $X = \langle x_1,\ldots,x_m \rangle$ and $Y = \langle y_1,\ldots,y_n \rangle$, find a common subsequence whose length is maximum.

Springtime ncaa tournament basketball

printing north carolina krzyzewski

Subsequence needn't be consecutive, but must be in order.

Other sequence questions

- ***Edit distance:*** Given 2 sequences, $X = \langle x_1,\ldots,x_m \rangle$ and $Y = \langle y_1,\ldots,y_n \rangle$, what is the minimum number of deletions, insertions, and changes that you must do to change one to another?

$$ED = |\text{operations}| = |X| + |Y| - 2\,|\text{LCS}|$$

Example: for the first pair of sequences, we have

$$|\text{operations}| = |X| + |Y| - 2\,|\text{LCS}|$$
$$= 10 + 8 - 2 \times 6 = 6$$

Other sequence questions

- ***DNA sequence alignment:*** Given a score matrix M on amino acid pairs with $M(a, b)$ for $a, b \in \{\Lambda\} \cup A$ ($A = \{A, T, C, G\}$, Λ - space symbol), and 2 DNA sequences, $X = \langle x_1,\ldots,x_m\rangle \in A^m$ and $Y = \langle y_1,\ldots,y_n\rangle \in A^n$, find the alignment with highest score.

$$
\begin{array}{c|c}
 & \begin{array}{ccccc} A & T & C & G & \Lambda \end{array} \\
\begin{array}{c} A \\ T \\ C \\ G \\ \Lambda \end{array} &
\begin{bmatrix}
1 & -1 & -1 & -1 & -2 \\
-1 & 1 & -1 & -1 & -2 \\
-1 & -1 & 1 & -1 & -2 \\
-1 & -1 & -1 & 1 & -2 \\
-2 & -2 & -2 & -2 & 1
\end{bmatrix}
\end{array}
$$

G ATCG GCAT

CAAT GTGAATC

-1-2+1+1-2+1-2+1-1+1+1-2 = -4

More problems

Optimal BST: Given sequence $K = k_1 < k_2 < \cdots < k_n$ of n sorted keys, with a search probability p_i for each key k_i, build a binary search tree (BST) with minimum expected search cost.

Matrix chain multiplication: Given a sequence of matrices $A_1\ A_2\ \ldots\ A_n$, with A_i of dimension $m_i \times n_i$, insert parenthesis to minimize the total number of scalar multiplications.

$$((\mathrm{A}_1 \times (\mathrm{A}_2 \times \mathrm{A}_3)) \times (\mathrm{A}_4 \times \mathrm{A}_5)) \text{ or } ((\mathrm{A}_1 \times \mathrm{A}_2) \times \mathrm{A}_3) \times (\mathrm{A}_4 \times \mathrm{A}_5))$$

Which is fast?

Number of scaler multiplications of $A_{mk} \times A_{kn}$:

$$m \times k \times n.$$

$(A_{10,100} \times A_{100,5}) \times A_{5,50}$:

$$10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500.$$

$B_{10,5} \times A_{5,50}$

$A_{10,100} \times (A_{100,5} \times A_{5,50})$:

$$10 \times 100 \times 50 + 100 \times 5 \times 50 = 75000.$$

$A_{10,100} \times C_{100,50}$

Dynamic Programming

- Dynamic Programming is an algorithm design technique for optimization problems: often minimizing or maximizing.
- Like divide and conquer, DP solves problems by combining solutions to subproblems.
- Unlike divide and conquer, subproblems are not independent.
  - » Subproblems may share subsubproblems,
  - » However, solution to one subproblem may not affect the solutions to other subproblems of the same problem. (More on this later.)
- DP reduces computation by
  - » Solving subproblems in a bottom-up fashion.
  - » Storing solution to a subproblem the first time it is solved.
  - » Looking up the solution when subproblem is encountered again.
- Key: determine structure of optimal solutions

Steps in Dynamic Programming

1. Characterize structure of an optimal solution.
2. Define value of optimal solution recursively.
3. Compute optimal solution values either top-down with caching or bottom-up in a table.
4. Construct an optimal solution from computed values.

We'll study these with the help of examples.

Longest Common Subsequence

- ***Problem:*** Given 2 sequences, $X = \langle x_1,\ldots,x_m \rangle$ and $Y = \langle y_1,\ldots,y_n \rangle$, find a common subsequence whose length is maximum.

springtime | ncaa tournament | basketball

printing | north carolina | krzyzewski

Subsequence needn't be consecutive, but must be in order.

Naïve Algorithm

- For every subsequence of X, check whether it's a subsequence of Y .
- Time: $\Theta(n2^m)$.
  - » 2^m subsequences of X to check.
  - » Each subsequence takes $\Theta(n)$ time to check: scan Y for first letter, for second, and so on.

Optimal Substructure

Theorem
Let $Z = \langle z_1, \ldots, z_{k-1}, z_k \rangle$ be any LCS of X and Y.
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 either $z_k \neq x_m$ and Z is an LCS of X_{m-1} and Y.
3. or $z_k \neq y_n$ and Z is an LCS of X and Y_{n-1}.

Notation:

prefix $X_i = \langle x_1,\ldots,x_i \rangle$ is the first i letters of X.

prefix $Y_j = \langle y_1,\ldots,y_j \rangle$ is the first j letters of Y.

Springtimeg	pringtime
Printing	Printing
LCS: printing	LCS: print

Theorem

Let $Z = \langle z_1, \ldots, z_{k-1}, z_k \rangle$ be any LCS of X and Y .

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 either $z_k \neq x_m$ and Z is an LCS of X_{m-1} and Y .
3. or $z_k \neq y_n$ and Z is an LCS of X and Y_{n-1}.

Case 1:

springtimeg

printing

springtime

printin

LCS: printin + g

Case 2:

pringtime

printing

pringtim pringtime

printing printin

LCS: print = max{print, print}

Optimal Substructure

Theorem

Let $Z = \langle z_1, \ldots, z_k\rangle$ be any LCS of X and Y.

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

Proof: (case 1: $x_m = y_n$)

Any common sequence Z' that does not end in $x_m = y_n$ can be made longer by adding $x_m = y_n$ to the end. Therefore,

(1) longest common subsequence (LCS) Z must end in $x_m = y_n$.

(2) $Z_{k\text{-}1}$ is a common subsequence of $X_{m\text{-}1}$ and $Y_{n\text{-}1}$, and

(3) there is no longer CS of $X_{m\text{-}1}$ and $Y_{n\text{-}1}$, or Z would not be an LCS.

Optimal Substructure

Theorem

Let $Z = \langle z_1, \ldots, z_k \rangle$ be any LCS of X and Y.

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

Proof: (case 2: $x_m \neq y_n$, and $z_k \neq x_m$)

Since Z does not end in x_m,

(1) Z is a common subsequence of $X_{m\text{-}1}$ and Y, and

(2) there is no longer CS of $X_{m\text{-}1}$ and Y, or Z would not be an LCS.

(case 2: $x_m \neq y_n$, and $z_k \neq y_n$) Since Z does not end in y_n,

(3) Z is a common subsequence of $Y_{n\text{-}1}$ and X, and

(4) there is no longer CS of $Y_{n\text{-}1}$ and X, or Z would not be an LCS.

Recursive Solution

- Define $c[i, j]$ = length of LCS of X_i and Y_j .
- We want to get $c[m,n]$.

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

This gives a recursive algorithm and solves the problem.
But does it solve it well?

Recursive Solution

$$c[\alpha,\beta]=\begin{cases}0 & \text{if } \alpha \text{ empty or } \beta \text{ empty},\\ c[\mathit{prefix}\,\alpha,\mathit{prefix}\,\beta]+1 & \text{if } \mathrm{end}(\alpha)=\mathrm{end}(\beta),\\ \max(c[\mathit{prefix}\,\alpha,\beta],c[\alpha,\mathit{prefix}\,\beta]) & \text{if } \mathrm{end}(\alpha)\neq\mathrm{end}(\beta).\end{cases}$$

c[springtime, printing]

c[springtim, printing] *c*[springtime, printin]

same subproblem

[springti, printing] [springtim, printin] [springtim, printin] [springtime, printi]

[springt, printing] [springti, printin] [springtim, printi] [springtime, print]

Recursive Solution

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

- Keep track of $c[\alpha,\beta]$ in a table of nm entries
- Top-down
- Bottom-up

X = springtime

Y = printing

		p	r	i	n	t	i	n	g
	0	0	0	0	0	0	0	0	0
s	0								
p	0								
r	0								
i	0								
n	0								
g	0								
t	0								
i	0								
m	0								
e	0								

Computing the length of an LCS

LCS-LENGTH (X, Y)

```
m ← length[X]
n ← length[Y]
for i ← 1 to m
      do c[i, 0] ← 0
for j ← 0 to n
      do c[0, j ] ← 0
for i ← 1 to m
       do for j ← 1 to n
             do if x_i = y_j
                      then c[i, j ] ← c[i–1, j–1] + 1
                                b[i, j ] ← "↖"
                      else if c[i–1, j ] ≥ c[i, j–1]
                               then 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
```

initialization

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

$b[i, j]$ points to table entry whose subproblem we used in solving LCS of X_i and Y_j.

i-1,j-1	$i-1, j$
i, j -1	

$c[m,n]$ contains the length of an LCS of X and Y.

Time: $O(mn)$

Recursive Solution

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

X = ABCBDAB

Y = BDCABA

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0						
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

Recursive Solution

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

X = ABCBDAB

Y = BDCABA

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	↑0	↑0	↑0	↖1	←1	↖1
B	0	↖1	←1	←1	↑1	↖2	←2
C	0	↑1	↑1	↖2	←2	↑2	↑2
B	0	↖1	↑1	↑2	↑2	↖3	←3
D	0	↑1	↖2	↑2	↑2	↑3	↑3
A	0	↑1	↑2	↑2	↖3	↑3	↖4
B	0	↖1	↑2	↑2	↑3	↖4	↑4

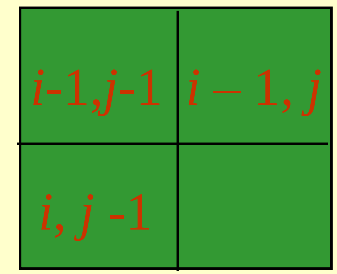
X = ABCBDAB

Y = BDCABA

LCS: BCBA

Constructing an LCS

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



- Initial call is PRINT-LCS(b, X, m, n).
- When $b[i, j\,] = \nwarrow$, we have extended LCS by one character. So LCS = number of entries with $\nwarrow$ in them.
- Time: $O(m + n)$

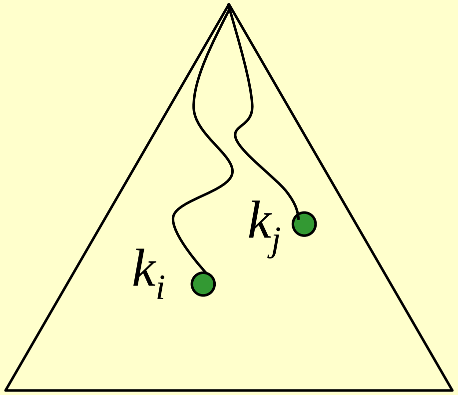
Steps in Dynamic Programming

1. Characterize structure of an optimal solution.
2. Define value of optimal solution recursively.
3. Compute optimal solution values either top-down with caching or bottom-up in a table.
4. Construct an optimal solution from computed values.

We'll study these with the help of examples.

Optimal Binary Search Trees

- Problem
  - » Given sequence $K = k_1 < k_2 < \cdots < k_n$ of n sorted keys, with a search probability p_i for each key k_i.
  - » Want to build a binary search tree (BST) with minimum expected search cost.
  - » Actual cost = # of items (nodes in the tree) examined.
  - » For key k_i, $cost(k_i)$ = depth$_T(k_i)$ +1, where depth$_T(k_i)$ = depth of k_i in BST T .



Expected Search Cost

$$E[\text{search cost in } T]$$

Mathematical expectation of searching costs of all nodes in T

$$= \sum_{i=1}^{n} cost(k_i) \cdot p_i$$

$$= \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) \cdot p_i \qquad (15.16)$$

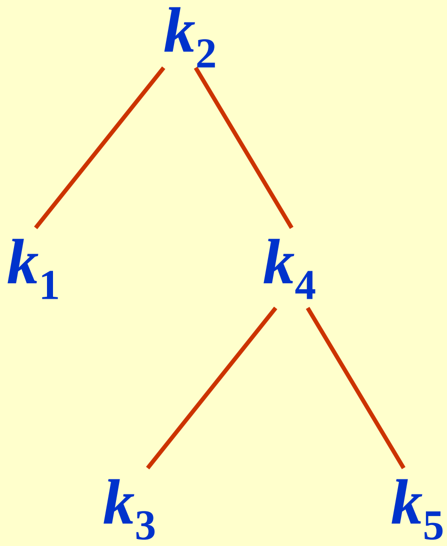
$$= \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=1}^{n} p_i$$

$$= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i$$

Sum of probabilities is 1.

Example

- Consider 5 keys with these search probabilities:
$p_1 = 0.25$, $p_2 = 0.2$, $p_3 = 0.05$, $p_4 = 0.2$, $p_5 = 0.3$.



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i)\cdot p_i$
1	1	0.25
2	0	0
3	2	0.1
4	1	0.2
5	2	0.6
		1.15

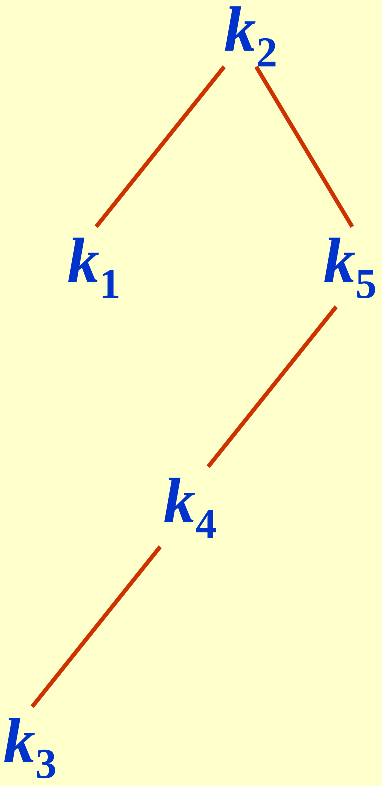
$k_1 < k_2 < \cdots < k_5$

Therefore, E[search cost]

$$= 1 + \sum_1^5 \text{depth}_T(k_i) \cdot p_i$$
$$= 1 + 1.15 = 2.15.$$

Example

- $p_1 = 0.25$, $p_2 = 0.2$, $p_3 = 0.05$, $p_4 = 0.2$, $p_5 = 0.3$.



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	0.25
2	0	0
3	3	0.15
4	2	0.4
5	1	0.3
		1.10

Therefore, E[search cost] = 2.10.

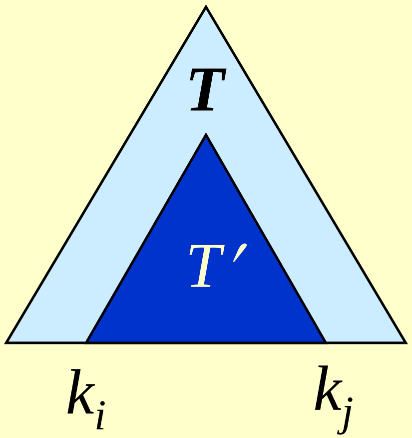
This tree turns out to be optimal for this set of keys.

Observation

- **Observations:**
  - » Optimal BST may not have smallest height.
  - » Optimal BST may not have highest-probability key at root.
- Build by exhaustive checking?
  - » Construct each *n*-node BST.
  - » For each,
    assign keys and compute expected search cost.
  - » But there are $\Omega(4^n/n^{3/2})$ different BSTs with *n* nodes.

Optimal Substructure

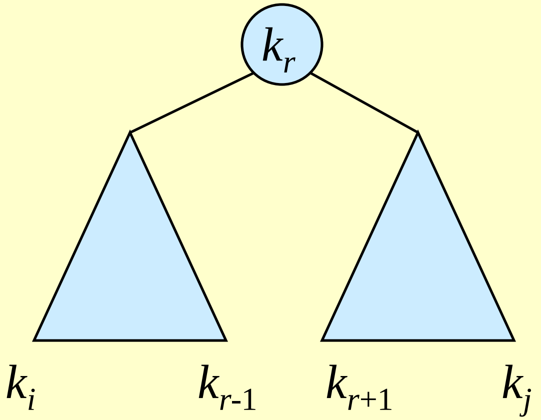
- Any subtree of a BST contains keys in a contiguous range $k_i, \ldots, k_j$ for some $1 \le i \le j \le n$.



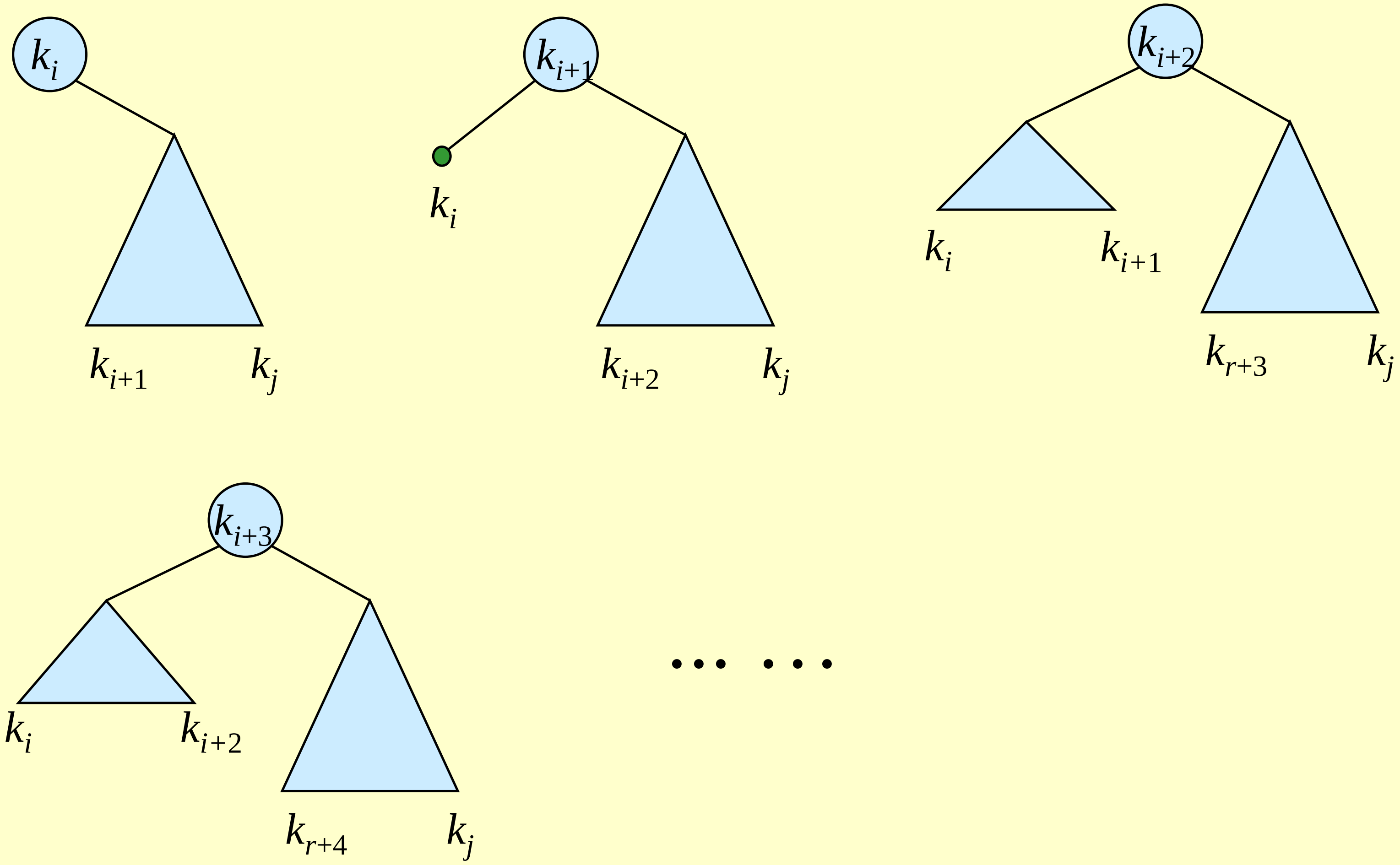
- If T is an optimal BST and
  T contains subtree T' with keys $k_i, \ldots, k_j$,
  then T' must be an optimal BST for keys $k_i, \ldots, k_j$.
- **Proof:** … … .

Optimal Substructure

- One of the keys in k_i, …, k_j, say k_r, where $i \le r \le j$, must be the root of an optimal subtree for these keys.
- Left subtree of k_r contains k_i,…, k_{r-1}.
- Right subtree of k_r contains k_{r+1}, …, k_j.



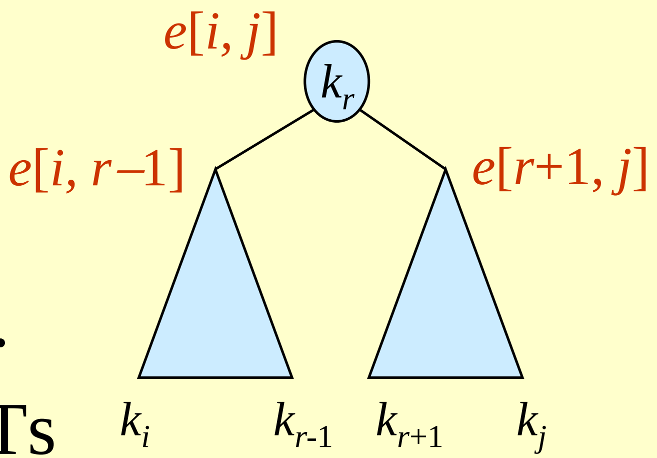
- To find an optimal BST:
  » Examine all candidate roots k_r, for $i \le r \le j$
  » Determine all optimal BSTs containing k_i, …, k_{r-1} and all optimal BSTs containing k_{r+1}, …, k_j



k_i
k_{i+1}
k_j
k_{i+1}
k_i
k_{i+2}
k_j
k_{i+2}
k_i
k_{i+1}
k_{r+3}
k_j
k_{i+3}
k_i
k_{i+2}
k_{r+4}
k_j
… …

Recursive Solution

- Find optimal BST for $k_i, \ldots, k_j$, where $i \geq 1, j \leq n, j \geq i-1$. When $j = i - 1$, the tree is empty.
- Define $e[i, j]$ = expected search cost of optimal BST for $k_i, \ldots, k_j$.



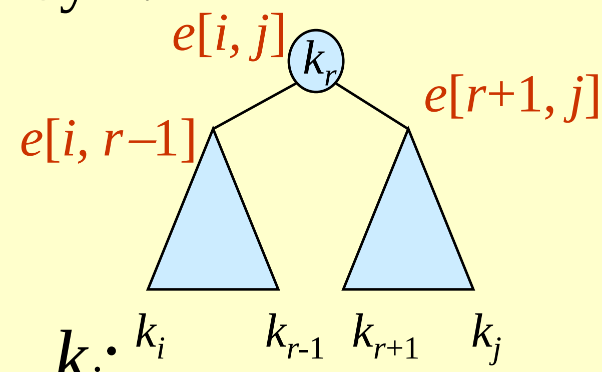
- If $j = i - 1$, then $e[i, j] = 0$.
- If $j \geq i$,
  - » Select a root k_r, for some $i \leq r \leq j$.
  - » Recursively make an optimal BSTs
    - for $k_i, \ldots, k_{r-1}$ as the left subtree, $e[i, r - 1]$ and
    - for $k_{r+1}, \ldots, k_j$ as the right subtree, $e[r + 1, j]$.

Recursive Solution

- When the OPT subtree becomes a subtree of a node:
  - » Depth of every node in OPT subtree goes up by 1.
  - » Expected search cost increases by

$$\boxed{w(i, j) = \sum_{l=i}^{j} p_l}$$

from (15.16)



- If k_r is the root of an optimal BST for $k_i ,..,k_j$:
  - » $e[i, j] = p_r + (e[i, r–1] + w(i, r–1)) + (e[r+1, j] + w(r+1, j))$

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

- But, we don't know k_r. Hence,

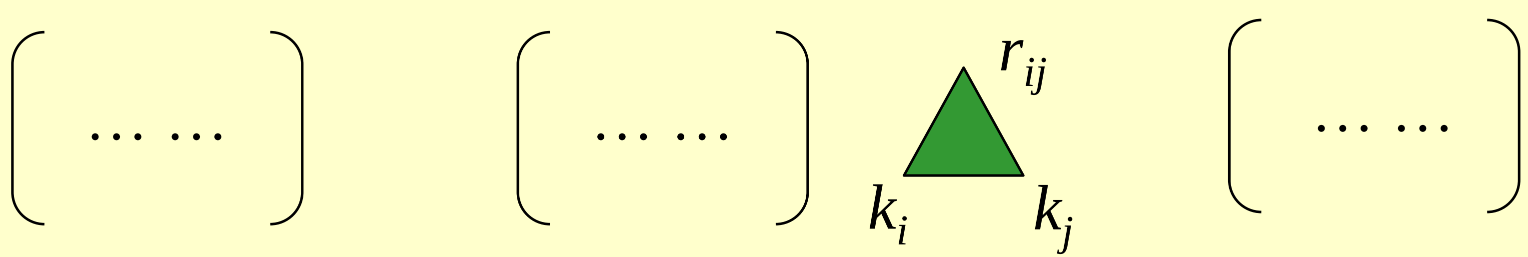
$$\boxed{e[i, j] = \begin{cases} 0 & \text{if } j = i-1 \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}}$$

Computing an Optimal Solution

For each subproblem (i, j), store:

- expected search cost in a table $e[1..n + 1, 0..n]$
  - » Will use only entries $e[i, j\ ]$, where $j \geq i–1$.
- $\text{root}[i, j]$ = root of subtree with keys $k_i, \ldots, k_j$, for $1 \leq i \leq j \leq n$.
- $w[1..n + 1, 0..n]$ = sum of probabilities
  - » $w[i, i - 1] = 0$ for $1 \leq i \leq n$.
  - » $w[i, j] = w[i, j - 1] + p_j$ for $1 \leq i \leq j \leq n$.

$e[1..n + 1, 0..n]$ $root[1..n, 1..n]$ $w[1..n + 1, 0..n]$



Pseudo-code

```
OPTIMAL-BST(p, n)
for i ← 1 to n + 1
    do e[i, i–1] ← 0
       w[i, i – 1] ← 0
for l ← 1 to n
    do for i ← 1 to n – l + 1
        do j ← i + l – 1
           w[i, j] ← w[i, j – 1] + p_j
           e[i, j] ← ∞
           for r ← i to j
               do t ← e[i, r–1] + e[r + 1, j] + w[i, j ]
                  if t < e[i, j]
                  then e[i, j] ← t
                       root[i, j] ← r
return e and root
```

Consider all trees with *l* keys.

Fix the first key.

Fix the last key

$$w(i, j) = \sum_{l=i}^{j} p_l$$

Determine the root of the optimal (sub)tree

$$e[i, j] = \begin{cases} 0 & \text{if } j = i - 1 \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$

Time: $O(n^3)$

Example

Construct an optimal binary search tree over five key values k1 < k2 < k3 < k4 < k5 with access probability 0.3, 0.2, 0.1, 0.15, and 0.25, respectively.

Pseudo-code

e[i,j]

	j=0	1	2	3	4	5
i=1	0					
2		0				
3			0			
4				0		
5					0	
6						0

w[i,j]

	j=0	1	2	3	4	5
i=1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Example

Construct an optimal binary search tree over five key values k1 < k2 < k3 < k4 < k5 with access probability 0.3, 0.2, 0.1, 0.15, and 0.25, respectively.

1. w[i,j] $l = 1$

	j=0	1	2	3	4	5
i=1	0	0.3				
2		0	0.2			
3			0	0.1		
4				0	0.15	
5					0	0.25
6						0

OPTIMAL-BST(p, n)

for $i \leftarrow 1$ **to** $n + 1$
  do $e[i, i-1] \leftarrow 0$
    $w[i, i-1] \leftarrow 0$
for $l \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** $n - l + 1$
    do $j \leftarrow i + l - 1$
      $w[i, j] \leftarrow w[i, j-1] + p_j$
      $e[i, j] \leftarrow \infty$
      for $r \leftarrow i$ **to** j
        do $t \leftarrow e[i, r-1] + e[r + 1$
        $+ w[i, j]$
      if $t < e[i, j]$
      then $e[i, j] \leftarrow t$
      $root[i, j] \leftarrow r$
return e and $root$

1. e[i,j] $l = 1$

	j=0	1	2	3	4	5
i=1	0	0.3				
2		0	0.2			
3			0	0.1		
4				0	0.15	
5					0	0.25
6						0

1. r[i,j] $l = 1$

	j=0	1	2	3	4	5
i=1		1				
2			2			
3				3		
4					4	
5						5
6						

OPTIMAL-BST(p, n)

1. **for** $i \leftarrow 1$ **to** $n + 1$
2. **do** $e[i, i-1] \leftarrow 0$
3. $w[i, i-1] \leftarrow 0$
4. **for** $l \leftarrow 1$ **to** n
5. **do for** $i \leftarrow 1$ **to** $n - l + 1$
6. **do** $j \leftarrow i + l - 1$
7. $w[i, j] \leftarrow w[i, j-1] + p_j$
8. $e[i, j] \leftarrow \infty$
9. **for** $r \leftarrow i$ **to** j
10. **do** $t \leftarrow e[i, r-1] + e[r + 1$
    $+ w[i, j]$
11. **if** $t < e[i, j]$
12. **then** $e[i, j] \leftarrow t$
13. $root[i, j] \leftarrow r$
10. return e and $root$

$$e[i, j] = \begin{cases} 0 & \text{if } j = i - 1 \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$

2. w[i,j] $l = 2$

	j=0	1	2	3	4	5
i=1	0	0.3	0.5			
2		0	0.2	0.3		
3			0	0.1	0.25	
4				0	0.15	0.4
5					0	0.25
6						0

2. e[i,j] $l = 2$

	j=0	1	2	3	4	5
i=1	0	0.3	0.7			
2		0	0.2	0.4		
3			0	0.1	0.35	
4				0	0.15	0.55
5					0	0.25
6						0

2. r[i,j] $l = 2$

	j=0	1	2	3	4	5
i=1		1	1			
2			2	2		
3				3	4	
4					4	5
5						5
6						

3. w[i,j] $l = 3$

	j=0	1	2	3	4	5
i=1	0	0.3	0.5	0.6		
2		0	0.2	0.3	0.45	
3			0	0.1	0.25	0.5
4				0	0.15	0.4
5					0	0.25
6						0

3. e[i,j] $l = 3$

	j=0	1	2	3	4	5
i=1	0	0.3	0.7	1		
2		0	0.2	0.4	0.8	
3			0	0.1	0.35	0.85
4				0	0.15	0.55
5					0	0.25
6						0

3. r[i,j] $l = 3$

	j=0	1	2	3	4	5
i=1		1	1	2		
2			2	2	3	
3				3	4	5
4					4	5
5						5
6						

4. w[i,j] $l = 4$

	j=0	1	2	3	4	5
i=1	0	0.3	0.5	0.6	0.75	
2		0	0.2	0.3	0.45	0.7
3			0	0.1	0.25	0.5
4				0	0.15	0.4
5					0	0.25
6						0

4. e[i,j] $l = 4$

	j=0	1	2	3	4	5
i=1	0	0.3	0.7	1	1.4	
2		0	0.2	0.4	0.8	1.35
3			0	0.1	0.35	0.85
4				0	0.15	0.55
5					0	0.25
6						0

4. r[i,j] $l = 4$

	j=0	1	2	3	4	5
i=1		1	1	2	2	
2			2	2	3	4
3				3	4	5
4					4	5
5						5
6						

5. w[i,j] $l = 5$

	j=0	1	2	3	4	5
i=1	0	0.3	0.5	0.6	0.75	1
2		0	0.2	0.3	0.45	0.7
3			0	0.1	0.25	0.5
4				0	0.15	0.4
5					0	0.25
6						0

5. e[i,j] $l = 5$

	j=0	1	2	3	4	5
i=1	0	0.3	0.7	1	1.4	2.15
2		0	0.2	0.4	0.8	1.35
3			0	0.1	0.35	0.85
4				0	0.15	0.55
5					0	0.25
6						0

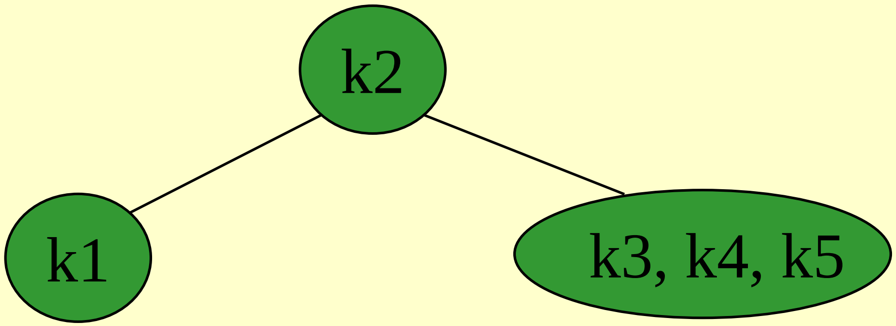
5. r[i,j] $l = 5$

	j=0	1	2	3	4	5
i=1		1	1	1	2	2
2			2	2	2	4
3				3	4	4
4					4	5
5						5
6						

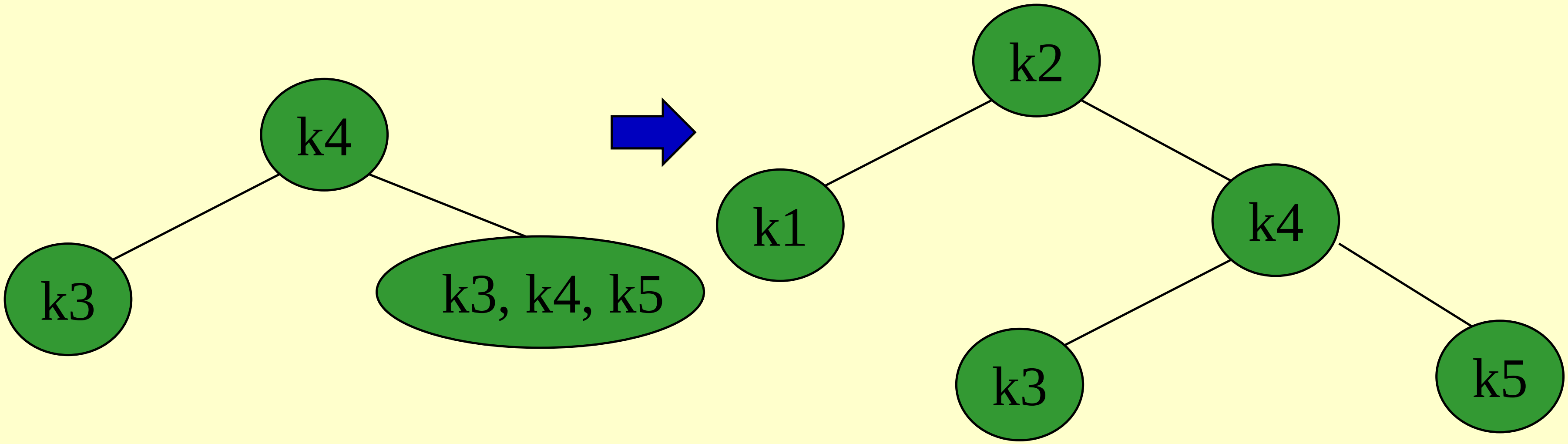
r[i,j]

	j=0	1	2	3	4	5
i=1		1	1	1	2	2
2			2	2	2	4
3				3	4	4
4					4	5
5						5
6						

r[1, 5] = 2 shows that the root of the tree over k1, k2, k3, k4, k5 is k2.



r[3, 5] = 4 shows that the root of the subtree over k3, k4, k5 is k4.



Elements of Dynamic Programming

- Optimal substructure
- Overlapping subproblems

Optimal Substructure

- Show that a solution to a problem consists of making a choice, which leaves one or more subproblems to solve.
- Suppose that you are given this last choice that leads to an optimal solution.
- Given this choice, determine which subproblems arise and how to characterize the resulting space of subproblems.
- Show that the solutions to the subproblems used within the optimal solution must themselves be optimal. Usually use cut-and-paste.
- Need to ensure that a wide enough range of choices and subproblems are considered.

Optimal Substructure

- Optimal substructure varies across problem domains:
  - » 1. *How many subproblems* are used in an optimal solution.
  - » 2. *How many choices* in determining which subproblem(s) to use.
- Informally, running time depends on (# of subproblems overall) × (# of choices).
- How many subproblems and choices do the examples considered contain?
- Dynamic programming uses optimal substructure **bottom up**.
  - » *First* find optimal solutions to subproblems.
  - » *Then* choose which to use in optimal solution to the problem.

Optimal Substucture

- Does optimal substructure apply to all optimization problems? No.
- Applies to determining the shortest path but NOT the longest simple path of an unweighted directed graph.
- Why?
  - » Shortest path has independent subproblems.
  - » Solution to one subproblem does not affect solution to another subproblem of the same problem.
  - » Subproblems are not independent in longest simple path.
    - Solution to one subproblem affects the solutions to other subproblems.
  - » Example:

Overlapping Subproblems

- The space of subproblems must be "small".
- The total number of distinct subproblems is a polynomial in the input size.
  - » A recursive algorithm is exponential because it solves the same problems repeatedly.
  - » If divide-and-conquer is applicable, then each problem solved will be brand new.

Question: What kind of trees will be created, if the search probabilities of all the key words are the same?

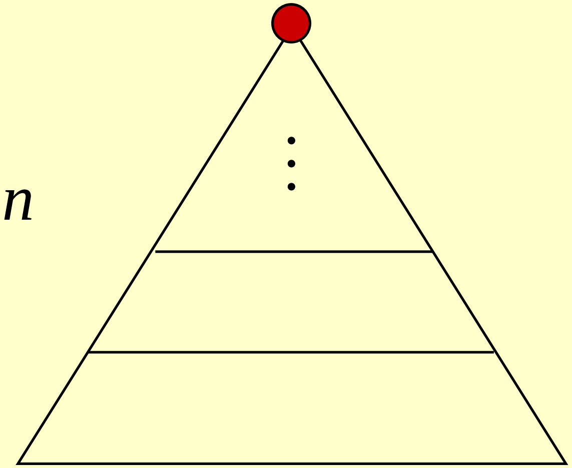
Answer: A balanced binary search tree.

Reason: In this case, the mathematical expectation is:

$$\frac{1}{n}\sum_i depth_T(k_i).$$

This value reaches minimum when the tree is balanced.

depth(*a leaf*) = $\log_2 n$

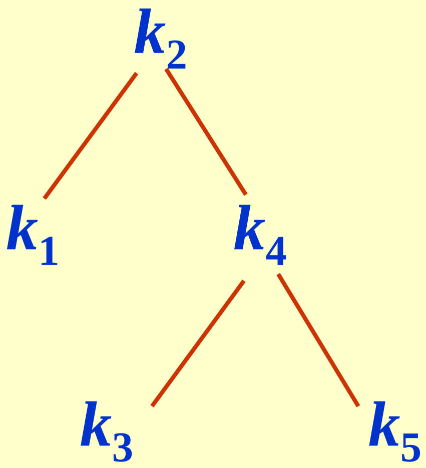


$\leq \lceil n/2^{2+1} \rceil$

$\leq \lceil n/2^{1+1} \rceil$

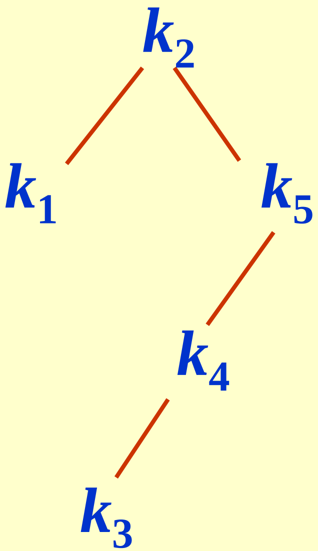
$\leq \lceil n/2^{0+1} \rceil$

$$\frac{1}{n}\sum_i depth_T(k_i) = \frac{1}{n} O(n \log_2 n) = O(\log_2 n)$$



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	0.2
2	0	0
3	2	0.4
4	1	0.2
5	2	0.4
		1.2

Therefore, E[search cost] = 2.2.



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	0.2
2	0	0
3	3	0.6
4	2	0.4
5	1	0.2
		1.4

Therefore, E[search cost] = 2.4.