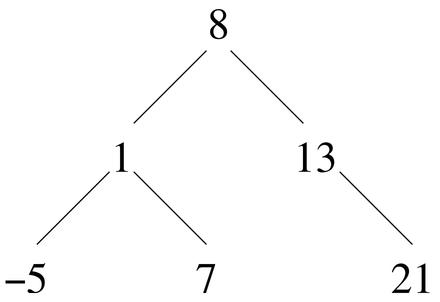
# Optimal BST

- When the probabilities of searching each key is known, it easy to compute the expected cost of accessing the tree.
- An OBST is a BST which has minimal expected cost.

| Key | -5 | 1 | 8 | 7 | 13 | 21 |
|---|---|---|---|---|---|---|
| Probabilities | 1/8 | 1/32 | 1/16 | 1/32 | 1/4 | 1/2 |




- Access cost for each node $i$ is $(1 + d_i) \cdot p_i$ where $d_i$ is the depth of the node.
- Therefore, the expected value of a search is:

$$E[cost] = 1 \cdot \frac{1}{16} + 2 \cdot (\frac{1}{32} + \frac{1}{4}) + 3 \cdot (\frac{1}{8} + \frac{1}{32} + \frac{1}{2})$$

# Optimal BST

- The tree shown was not optimal.
- Let us put 21 closer to the root. Then with its high probability, the tree will have a lower expected cost.
- Criterion for an optimal tree:

Each optimal binary search tree is composed of a root and (at most) two optimal subtrees, the left and the right.

# Optimal BST

- Optimal substructure property is thus quite evident.
- All we do now is to find a recursive formulation.
- The recursive formulation is also easy.
- We use the same notation as in case of matrix chain or polygon triangulation, i.e., let $B_{i\ldots j}$ represent the OBST formed from words $w_i, w_{i+1}, \ldots w_j$. We consider the words in sorted order.

