CSE 326
Lecture 22

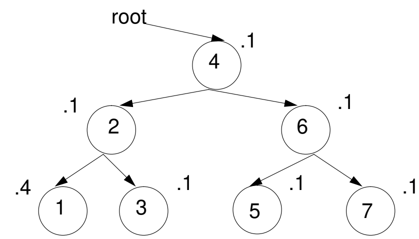
Spring Quarter 1995

Optimum Binary Search Tree

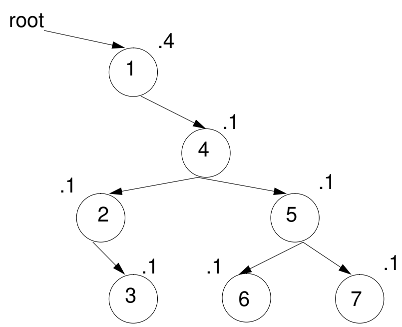
- Suppose we know the frequencies for each potential look-up. Then the optimum binary search tree is a potential option.
- Suppose we have the following frequencies

1	2	3	4	5	6	7
.4	.1	.1	.1	.1	.1	.1

- Ave time per look-up = .4x3 + .1x2 + .1x3 + .1x1 + .1x3 + .1x2 + .1x3 = 2.6



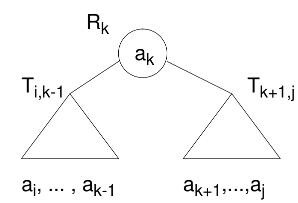
Optimum Binary Search Tree Example



Ave time per look-up = .4x1 + .1x3 + .1x4 + .1x2 + .1x4 + .1x3 + .1x4 = 2.4

Algorithm for Optimum Binary Search Tree

- Suppose the data items are $a_1, a_2, \ldots, a_n$ with frequencies $f_1, f_2, \ldots, f_n$.
- To construct the optimal binary search tree for $a_i, a_2, \ldots, a_j$, for each k we find the optimal binary search trees for $a_i, a_2, \ldots, a_{k-1}$ and for $a_{k+1}, a_{k+2}, \ldots, a_j$ and compute the cost of the binary search tree with a_k at the root. We then choose the k which minimizes that cost.



- $\text{cost}(R_k) = f_i + f_{i+1} + \ldots + f_j + \text{cost}(T_{i,k-1}) + \text{cost}(T_{k+1,j})$
- $T_{i,j} = R_k$ which minimizes $\text{cost}(R_k)$

Optimum Binary Search Tree Calculation

- Frequencies are .4 .1 .1 .1 .1 .1 .1

cost(i,j) = cost($T_{i,j}$)

j

i	1	2	3	4	5	6	7
1	**.4 , 1**	**.6, 1**	**.9, 1**	**1.2, 1**	**1.6, 1**		
2		**.1, 2**	**.3, 2**	**.5, 3**	**.8, 3**	**1.1, 3**	
3			**.1, 3**	**.3, 3**	**.5, 4**	**.8, 4**	**1.1, 4**
4				**.1, 4**	**.3, 4**	**.5, 5**	**.8, 5**
5					**.1, 5**	**.3, 5**	**.5, 6**
6						**.1, 6**	**.3, 6**
7							**.1, 7**

cost(1,6) = .9 + $\min_k$(cost(1,k-1) + cost(k+1,6)) = .9 + cost(1,0) + cost(2,6) = 2
cost(2,7) = .6 + $\min_k$(cost(2,k-1) + cost(k+1,7)) = .6 + cost(2,3) + cost(5,7) = 1.4
cost(1,7) = 1 + $\min_k$(cost(1,k-1) + cost(k+1,7)) = 1 + cost(1,3) + cost(5,7) = 2.4

Constructing the Tree from the Table

	1	2	3	4	5	6	7
1	**.4 , 1**	**.6, 1**	**.9, 1**	**1.2, 1**	**1.6, 1**	**2, 1**	**2.4, 3**
2		**.1, 2**	**.3, 2**	**.5, 3**	**.8, 3**	**1.1, 3**	**1.4, 4**
3			**.1, 3**	**.3, 3**	**.5, 4**	**.8, 4**	**1.1, 4**
4				**.1, 4**	**.3, 4**	**.5, 5**	**.8, 5**
5					**.1, 5**	**.3, 5**	**.5, 6**
6						**.1, 6**	**.3, 6**
7							**.1, 7**

Optimum Binary Search Tree

- The basic optimum binary search tree algorithm is $O(n^3)$
- Optimum binary search tree can be found in $O(n^2)$.
- There is a fancy O(n log n) algorithm if all the data only at the leaves.
- There is a slightly more general problem where look-ups can fail. In this case we have frequencies $g_0, g_1, \ldots, g_n$ in addition to $f_1, f_2, \ldots, f_n$.

  g_i is the probability that a search fails between a_i and a_{i+1}.

  The optimum binary search tree algorithm can be modified to solve this problem.

The Splay Tree Look-up Table

- Basic Splay Tree

  1. Put the items into a splay tree.

  2. On each look-up splay item found to root. If there are failed look-ups the last node found should be splayed to the root also.

  This method has the good properties of the optimum binary search tree with no preprocessing.

- Splay Tree Training. Suppose the look-up table will be used for a long time with some relative frequencies. Can we remove the overhead of splaying?

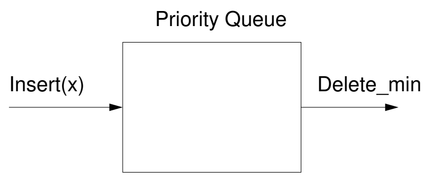
  Yes, to some degree. Run the splay tree method for a while, then leave the tree as a static binary search tree.

Priority Queues

- Priority queues support the two operations

  1. Insert(x) which inserts the key (number x) into the priority queue

  2. Delete_min which removes the smallest item in the priority queue and return it.



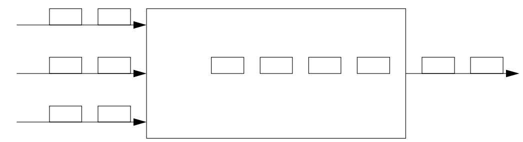
Applications of Priority Queues

- Queues with inherent priorities

  - Run-time queue in an operating system

  - Printer queue

  - Network multiplexer



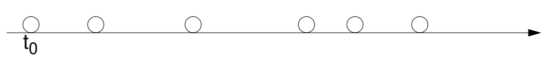
Discrete Event Simulation

- An event is a pair (x,t) where x is a description of the event and t is the time it is supposed to occur.
- A discrete event simulator (DES) maintains a set S of events which it intends to simulate in time order.
- Simulation engine of a DES:

  repeat

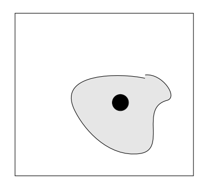
  Find and remove the event (x,t_0) from S such that t_0 is minimal;
  Do whatever x says to do, in the process new events (x_1,t_1),
  $(x_2,t_2),\ldots,(x_k,t_k)$ may be generated;
  Insert the new events into S.

  until the simulation has gone long enough or S is empty



The N-Body Problem in Astrophysics

- We have millions of stars clustered in galaxies and we wish to see what happens when the galaxies collide or get close to each other.
- If there are N bodies then there are N^2 gravitational interactions. To simulate one time increment would take far too long.
- Put the stars into a quad-tree or 3-d tree. A node of the tree represents a cluster of stars. Such a cluster can be approximated by its center of mass.



Continuing N-Body Simulation

- Each node n has an error $\varepsilon(n)$ associated with it. $\varepsilon(n)$ is the error in approxmating the stars in the cluster by its center of mass.
- Suppose we have some acceptable error δ for one step of the simulation. We maintain a set of nodes (clusters) C whose sum of errors is ε. Initially C contains only the root and $\varepsilon = \varepsilon(\text{root})$.
- Cluster Finding

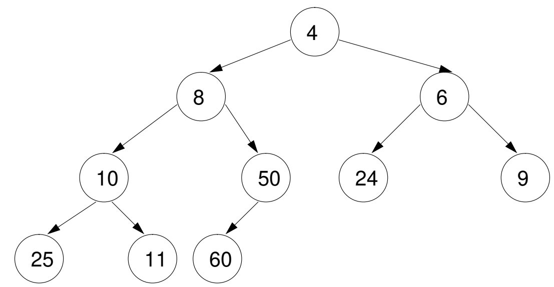
  while $\varepsilon > \delta$ do

  Find and remove the member n of C such that $\varepsilon(n)$ is maximal;
  Insert $n_1, n_2, \ldots, n_k$ the children of n which have positive mass, into C;
  $\varepsilon := \varepsilon - \varepsilon(n) + \varepsilon(n_1) + \ldots + \varepsilon(n_k)$

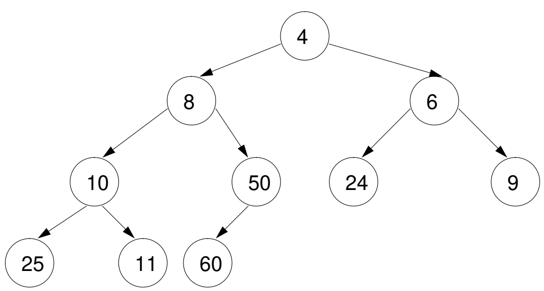
  endwhile

The Heap Data Structure

- A binary heap is a complete binary tree with the "heap property".
- Heap property: The key of the parent is smaller than the keys of its children.



Storage of a Complete Binary Tree



1	2	3	4	5	6	7	8	9	10
4	8	6	10	50	24	9	25	11	60

Finding Children and Parent

- Children of i are in 2i and 2i+1
- Parent of i is in i/2