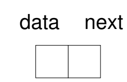
CSE 326
Lecture 6

Spring Quarter 1995

Implementing Records and Pointers using Arrays

- This will be needed in many languages such as Fortran and Basic.
- It is very simple when the number records that will be needed is known in advance.
- Consider a very basic node type with two fields "data" (integer) and "next" (pointer).



- We know that no more than n records will ever be needed at one time.
- We delare two arrays D[1..n] and N[1..n] each which holds and integer. D is initialize to 0 and N is initialized such that N[k] = k-1 for k = 1 to n.
- We also delare an integer variable Free which will always be in the range 0 to N. Free is initialized to n.

Initial Data Structure

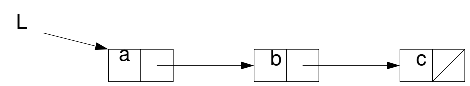
	D	N
1		0
2		1
3		2
4		3
5		4
6		5
n-1		n-2
n		n-1

Free = n

Basic Correspondences

- pointer = some integer from 0 to n
- null = 0
- p.data = D[p]
- p.next = N[p]
- The index Free is the pointer to the "list of free nodes".

Data Structure in Use



	D	N
1		3
2	c	0
3		0
4	a	6
5		8
6	b	2
7		5
8		1

L = 4

Free = 7

- Insert(x) {

  local integer q;
  if not(Free = 0) then q := Free
  else "overflow";
  Free := N[Free];
  D[q] := x;
  N[q] := L;
  L := q }

Data Structure in Use (Cont.)

	D	N
1		3
2	c	0
3		0
4	a	6
5		8
6	b	2
7		5
8		

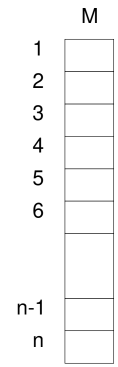
p = L

Free = 7

- Delete_First {

  local integer q;
  q := L;
  L := N[L];
  N[q] := Free;
  Free := q }

Global Memory Management



- M is the virtual address space.
- A record with 2 integer fields will take M[k] and M[k+1] in the memory.
- An object that requires m memory locations will take the block M[k], M[k+1], ... , M[k+m-1].
- Extra storage might be necessary for marking the block for memory management functions.
- Typical memory management functions

  -keeping a "heap" of free unstructured storage.

  - keeping free lists of objects of the same type.

  - moving unused objects back to the heap, perhaps merging objects of the same or different type.

  - garbage collection.

Analysis of Algorithms

- Analysis of an algorithm provides insight into how long the program will run and how much storage it will need

  - Time Complexity

  - Space Complexity.
- It is not uncommon that an analysis provides insight into alternative algorithms.
- Input size is usually indicated by a number n. Sometimes there are multiple inputs.
- Running time is a function of n such as:

  n, n^2, $n \log n$, 2^n
- Computer Scientist and Engineers typically use order notation so as not to bother with constant factors and low order terms.

  $20n^2 + 500n + 10000$

  low order terms

Rates of Growth

- Suppose a computer executes 10^{12} operations per second.

	10	**100**	**1,000**	**10,000**
n	**10^{-11} s**	**10^{-10} s**	**10^{-9} s**	**10^{-8} s**
n log n	**10^{-11} s**	**10^{-9} s**	**10^{-8} s**	**10^{-7} s**
n^2	**10^{-10} s**	**10^{-8} s**	**10^{-6} s**	**10^{-4} s**
n^3	**10^{-9} s**	**10^{-6} s**	**10^{-3} s**	**1 s**
n^4	**10^{-8} s**	**10^{-4} s**	**1 s**	**10^{4} s**
2^n	10^{-9} s	**10^{18} s**	**10^{289} s**	

10^4 s = 2.8 hr

10^{18} s = 30 billion years

Order Notation

- $T(n) = O(f(n))$ if there are constants c and n_0 such that $T(n) \leq cf(n)$ for all $n \geq n_0$.
- $T(n) = \Omega(f(n))$ if there are constants c and n_0 such that $T(n) \geq cf(n)$ for all $n \geq n_0$.
- $T(n) = \Theta(f(n))$ if $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$.
- Examples:

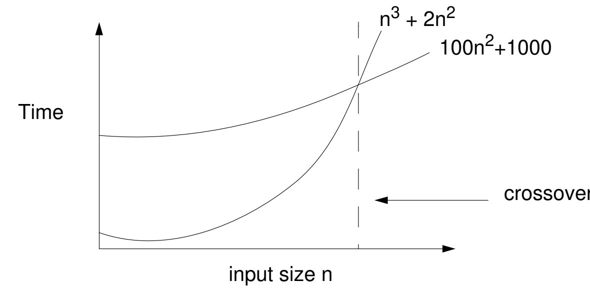
  $10{,}000\ n^2 + 25\ n = \Theta(n^2)$

  $10^{-10}\ n^2 = \Theta(n^2)$

  $n \log n = O(n^2)$

  $n \log n = \Omega(n)$

Understanding Orders of Magnitude



Some Basic Relationships

- Polynomial time is time $O(n^k)$ for some k.
- Exponential time is time $O(c^n)$ for some $c > 1$.
- Logarithmic time is time $O(\log n)$

  Note that in order notation the base of the logarithm is ignored. Why? Because $\log_k n = \log_k 2 \log_2 n$ for any k.
- $c^n = \Omega(n^k)$ for any $c > 1$ and any k.
- $\log^k n = O(n)$ for any k.

Kinds of Analysis

- Time or Space
- Upper bounds or lower bounds
  - $T(n) = O(f(n))$ implies $f(n)$ is an upper bound
  - $T(n) = \Omega(f(n))$ implies $f(n)$ is a lower bound
- Worst case - worst over all possibilites
- Average case - assuming some distribution of inputs.
- "Common" case
- Example Quicksort
  - worst case is n^2
  - average case is n log n

Example 1

- For loops - assume k = 0 initially.

  for i = 1 to n do
  for j = 1 to n do
  k := k + 1

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{n} 1\right) = \sum_{i=1}^{n} n = n^2$$

Example 2

- for i = 1 to n do
  for j = i to n do
  k := k + 1

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

$$\frac{n(n+1)}{2} = O(n^2)$$

Example 3

- Conditional

  if C then S1 else S2

  time = time of C + max(time of S1, time of S2)

- While loop

  while C do S

  time = sum of times of each iteration of the loop

Example 4

- Recursion
- Analysis amost always yields a recurrence of some form.
- Recursive max.

  $T(0) \leq b$

  $T(n) \leq c + T(n-1)$

- Analysis

  $T(n) \leq c + c + T(n-2)$ by substitution

  $\leq c + c + c + T(n-3)$ by substitution again

  $\leq kc + T(n - k)$ for all k from 1 to n.

  $\leq nc + T(0)$

  $\leq cn + b$