CS301
Data Structure and Algorithms
Lecture 8: Applications of linked list

Pandav Patel
Assistant Professor

Computer Engineering Department
Dharmsinh Desai University
Nadiad, Gujarat, India

Objective

- To understand applications of linked list

Overview

Implement stack using linked list

- Can you implement stack using singly linked list?
  - Which end will you use for PUSH and POP operations?
  - What is the problem with using TAIL for POP?
- Can you implement stack using doubly linked list?
  - Which end will you use for PUSH and POP operations?
  - Is one end preferable over other?
- Is there any advantage in using circular (singly/doubly) linked list over non-circular (singly/doubly) linked list?

Implement queue using linked list

- Can you implement queue using singly linked list?
  - Which end will you use for INSERT and DELETE operations?
  - What is the problem with using TAIL for DELETE?
- Can you implement queue using doubly linked list?
  - Which end will you use for INSERT and DELETE operations?
  - Is one end preferable over other?
- Is there any advantage in using circular (singly/doubly) linked list over non-circular (singly/doubly) linked list?

Representation of polynomials using linked list

- $P(x, y, z) = 6x^{17}y^{12}z^{3} + 7x^{15}y^{20}z^{30} + 9x^{2}y^{21}z^{34} + x^{2}y^{21}z^{27}$
  - A single term of polynomial can be represented as one node of LL

COEF	POW_X	POW_Y	POW_Z	LINK

- $Q(x, y, z) = 9x^{15}y^{20}z^{30}$ - $6x^{17}y^{12}z^{3} + 16x^{3}y^{4}z^{2} + 17x^{3}$

- How to find $R(x, y, z) = P(x, y, z) + Q(x, y, z)$?
  - What would be the time complexity of your approach?
  - Can we reduce time complexity by storing the terms in order?

Representation of polynomials using linked list (cont...)

- Terms of polynomial can be stored in descending order
- Term $D_1 X^{A_1} Y^{B_1} Z^{C_1}$ will precede term $D_2 X^{A_2} Y^{B_2} Z^{C_2}$ if
  - $A_1 > A_2$ or
  - $A_1 = A_2$ and $B_1 > B_2$ or
  - $A_1 = A_2$ and $B_1 = B_2$ and $C_1 > C_2$
  - Assumption: For a given polynomial following situation will never arise
    - $A_1 = A_2$ and $B_1 = B_2$ and $C_1 = C_2$
- In previous slide, polynomial P is already in descending order, but Q is not

Addition of two polynomials using linked list

Cases

- Polynomial with zero terms
  - Both polynomials have zero terms
  - One of the two polynomials has zero terms
- Number of terms are same
- Number of terms are different
- Regarding terms
  - A term in P and a term in Q may have same power
  - For a term in P there may not be term with same power in Q
  - For a term in Q there may not be term with same power in P
  - Addition of two terms with same power from P and Q may result in zero coefficient

Addition of two polynomials using linked list (cont...)

Steps

1. Set CURRP and CURRQ to point to first terms of P and Q resp.
2. Repeat thru step 4 while there are terms left for processing in both polynomials
3. Obtain values for terms pointed by CURRP and CURRQ
4. If powers of both terms are equal (same power of all vars) then
   - If terms do not cancel (resultant coeff not zero)
     - Insert sum of terms at end of result
   - Advance CURRP and CURRQ to point to next terms of respective polynomials

   Else if power of term pointed by CURRP > power of term pointed by CURRQ
   - Insert term pointed by CURRP at end of result
   - Advance CURRP to point to next term of P

   Else
   - Insert term pointed by CURRQ at end of result
   - Advance CURRQ to point to next term of Q
5. Append remaining terms from non-empty polynomial to result and return

Algorithm: POLLY_ADD(P, Q)
Add polynomials represented by P and Q.
P, Q: Pointers to first nodes of linked list representing polynomials to be added.
RH, RT: Pointer to first node and last node of result.
Assumptions: RH, RT are passed by ref to INS_AT_TAIL
CURRP and CURRQ: Pointers to nodes representing current term.
A1, A2, B1, B2, C1, C2, D1, D2: Temporary variables.

1. [Initialize]
   CURRP ←P
   CURRQ ←Q
   RH ←RT ←NULL
2. [End of any polynomial?]
   Repeat thru step 4 while CURRP ≠ NULL and CURRQ ≠ NULL
3. [Get values for current terms of P and Q]
   A1 ←POW_X(CURRP)
   A2 ←POW_X(CURRQ)
   B1 ←POW_Y(CURRP)
   B2 ←POW_Y(CURRQ)
   C1 ←POW_Z(CURRP)
   C2 ←POW_Z(CURRQ)
   D1 ←COEF(CURRP)
   D2 ←COEF(CURRQ)
4. [Compare terms and add to result]
   If $A1 = A2$ *and* $B1 = B2$ *and* $C1 = C2$ then
      If $D1 + D2 \neq 0$ then
         INS_AT_TAIL(A1, B1, C1, D1 + D2, RH, RT)
      CURRP ←LINK(CURRP)
      CURRQ ←LINK(CURRQ)
   Else if $(A1 > A2)$ *or* $((A1 = A2)$ *and* $(B1 > B2))$
      or $((A1 = A2)$ *and* $(B1 = B2)$ *and* $(C1 > C2))$ then
      INS_AT_TAIL(A1, B1, C1, D1, RH, RT)
      CURRP ←LINK(CURRP)
   Else
      INS_AT_TAIL(A2, B2, C2, D2, RH, RT)
      CURRQ ←LINK(CURRQ)
5. [Terms remaining in any of the polynomials?]
   If CURRP ≠ NULL then
      LINK(RT) ←COPY(CURRP)
   Else if CURRQ ≠ NULL then
      LINK(RT) ←COPY(CURRQ)
   return(RT)

Addition of two polynomials using linked list (cont...)

Discussion

- What if in step 4, $D1 + D2 \neq 0$ is false?
- Why is step 5 required?
- Check if all cases discussed in earlier slide are working fine with above algorithm
  - Will it work fine when both P and Q has zero terms (NULL)?
    - Yes!
  - Will it work fine if either P or Q has zero terms (NULL) and other polynomial is non-empty (has at-least one term)?
    - It will fail. Can you fix it?

Insert polynomial term in ordered linked list

Cases

- Empty list
- Non-empty list
    - Insert node at **front** end of the linked list
    - Insert node at the **rear** end of the linked list
    - Insert node in the **middle** of the linked list

Insert polynomial term in ordered linked list (cont...)

Steps

1. Create a new node
2. Initialize node fields (POW_X, POW_Y, POW_Z and COEF)
3. Handle empty list case
4. Handle insert at front end
5. Find predecessor of node to be inserted
6. Add new node behind its predecessor and return

Algorithm: PINSERT(NX, NY, NZ, NCOEF, HEAD)

Assumptions: Node with given value of NX, NY, NZ does not exist in the list.

Add new node with NX, NY, NZ and NCOEF in ordered (descending) list.

HEAD: Pointers to first node of linked list

NEW: Pointer to newly created node

CURR: Temporary pointers to node

A, B, C: Temporary variables.

1. [Create a new node]
   NEW ←Create a new node
   If NEW = NULL then
      Write("New node not created")
      return(HEAD)
2. [Initialize node fields]
   POW_X(NEW) ←NX
   POW_Y(NEW) ←NY
   POW_Z(NEW) ←NZ
   COEF(NEW) ←NCOEF
3. [Handle empty list case]
   If *HEAD = NULL* then
      LINK(NEW) ←NULL
      return(NEW)
4. [Does new node precede first node in the list?]
   A ←POW_X(HEAD)
   B ←POW_Y(HEAD)
   C ←POW_Z(HEAD)
   If $(NX > A)$ *or* $((NX = A)$ *and* $(NY > B))$
      or $((NX = A)$ *and* $(NY = B)$ *and* $(NZ > C))$ then
      LINK(NEW) ←HEAD
      return(NEW)
5. [Find predecessor of new node to be inserted]
   CURR ←HEAD
   While LINK(CURR) ≠ NULL do
      A ←POW_X(LINK(CURR))
      B ←POW_Y(LINK(CURR))
      C ←POW_Z(LINK(CURR))
      If $(NX < A)$ *or* $((NX = A)$ *and* $(NY < B))$
         or $((NX = A)$ *and* $(NY = B)$ *and* $(NZ < C))$ then
         CURR ←LINK(CURR)
      Else
         Exitloop
6. [Add new node behind predecessor and return]
   LINK(new) ←LINK(CURR)
   LINK(CURR) ←NEW
   return(HEAD)

Insert polynomial term in ordered linked list (cont...)

Discussion

- What if node with given powers already exist in the list?
- Can you rewrite this algorithm so that it works fine when node with given powers already exist in the list? (i.e. for some term present in the list, A = NX and B = NY and C = NZ)

Real world applications of linked list

- To implement Undo and Redo functionalities
- To provide navigation of previous and next photo/link in photo viewer/web browser
- Task scheduling by operating system in round robin fashion
- Playlist of songs with next, previous, loop functionality
- Let us implement a playlist where user can set his/her liking of a song in range 1 to 1 million. Following functionalities are expected. Which data structure should be preferred? Think of complexity for each of these operations.
    - Add new song to the list
    - Start listening from most favourite song to least favourite song
    - Delete least favourite song
    - Add new song to the list (liking of it can range from 1 to 1 million)
    - Play songs in a loop

Real world applications of linked list (cont...)

- What is we want to add following functionality to our solution in previous slide?
  - Start listening from least favourite song to most favourite song
- What is we want to add one more functionality to our solution?
  - Play songs randomly. Make sure that song is not repeated. (Question asked to me in Google interview in 2012)