자료구조 (Data Structure)
2주차: 링크드 리스트

리스트

- 정의: 데이터들에 순서를 붙여서 관리
- 특징: 순서에 따라 모든 데이터에 방문할 수 있다
- 기능: 데이터 추가, 삭제, 탐색

배열 리스트

- 커다란 메모리에 데이터를 왼쪽부터 이어서 저장
- 빅데이터는 메모리 확보 어려움
- 가변 크기 RESIZE에서 쓰지 않는 메모리를 점유

링크드 리스트

- 순서의 최소 정보는 각 데이터의 다음 데이터
- 데이터마다 별도로 메모리 할당
- 첫번째 노드만 알면 모든 노드 방문 가능
- 메모리 사용량은 가변 크기 배열 리스트와 비슷하다

노드

- 데이터와 다음 데이터 정보의 묶음
- val: 저장된 데이터
- next: 다음 노드 정보
- 마지막 데이터는 next가 비어있다

INSERT - 제일 뒤에 추가

- 직관적으로 봤을 때, 노드를 가장 쉽게 추가하려면
- 기존의 가장 마지막 노드의 next에 연결
- 마지막 노드를 찾는 비용 있음

INSERT - 제일 앞에 추가

- 가장 빠르게 리스트에 노드를 추가하는 방법
- 새 노드의 next에 기존의 첫번째 노드 연결
- 새 노드를 첫번째 노드로 변경

INSERT - 특정 노드 뒤에 추가

- 추가할 위치를 알면 빠른 추가 가능
- 다음 노드 정보를 잃지 않게 주의

INSERT_AFTER(p, x)

- 노드 p 뒤에 노드 x를 추가한다.

- 동작: x.next ← p.next

  p.next ← x

INSERT_AFTER(p, x)의 시간 효율

- p와 x의 next만 대입해서 상수 시간

DELETE - 제일 뒤를 삭제

- 제일 뒤에 있는 노드를 삭제하려면
- prev(마지막 직전)를 찾아 prev.next ← NULL로 만든 후
- 마지막 노드를 free

DELETE - 제일 앞을 삭제

- 제일 앞에 있는 노드는 다른 노드의 next에 연결 없음
- 삭제 하기 전 다음 노드 정보만 저장

DELETE - 특정 노드 뒤를 삭제

- 삭제할 때 next 정보를 잃지 않게 주의
- 삭제 하기 전 next 정보를 먼저 업데이트

DELETE_AFTER(p)

- 노드 p의 다음 노드를 삭제한다.
- 조건: p.next != NULL
- 동작: tmp ← p.next

  p.next ← tmp.next

  free(tmp)

DELETE_AFTER(p)의 시간 효율

- p.next만 바꾸고, 삭제해서 상수 시간

RETRIEVE(head, k)

- head로부터 k ≥ 1번째 노드 반환(head가 1번째)

- 동작: node ← head

  반복: node != NULL이고, k > 1인 동안

  node ← node.next

  k ← k - 1

- 반환: node (없으면 NULL)

RETRIEVE(head, k)의 시간 효율

- 특정 k에 대해 ~ k, 최악(끝까지 가면) ~ n

제일 앞에 더미 추가하기

- 모든 데이터 노드들이 다른 노드 뒤에 있음
- 더미에 저장된 데이터는 의미가 없고 next만 쓰인다
- 데이터가 없으면 head.next ← NULL

원형 + 더미(head)

- 원형: 마지막.next ← head
- 데이터 없으면: head.next ← head
- 연결을 하나 끊어도 노드들이 모두 이어져 있다

삭제된 노드들의 리스트

- 노드를 삭제할 때 free를 하지말고 따로 모아둔다
- 노드를 새로 만들 때 모아둔 노드에서 가져다 쓴다
- 모아둔 노드가 없을 때만 메모리를 할당받는다

PUSH_FREE(avail, node)

- 삭제 노드를 free list(원형+더미) 제일 앞에 추가

- 동작: node.next ← avail.next

  avail.next ← node

CREATE_DUMMY()

- head ← 새로운 노드
- head.val ← 0
- head.next ← head
- 반환: head

RETRIEVE_CIR(head, k)

- node ← head.next

- 반복: node != head이고, k > 1인 동안

  node ← node.next

  k ← k-1

- 반환: node // node == head면 NOT_FOUND(없음)

DELETE_AFTER_CIR(head, p, avail)

- 만약: p.next == head

  함수 종료

- tmp ← p.next
- p.next ← p.next.next
- PUSH_FREE(avail, tmp)

CREATE_NODE(avail)

- 만약: avail.next == avail

  node ← 새로운 노드

- 그외:

  node ← avail.next

  avail.next ← avail.next.next

- 반환: node

DELETE_CIR(head, avail)

- first ← head.next
- last ← head
- last.next ← avail.next
- avail.next ← first

원형 양방향 링크드 리스트

- 양방향 이동 가능
- 파라미터로 받은 노드를 삭제 가능
- val: 저장된 데이터
- prev: 이전 노드 정보
- next: 다음 노드 정보

DELETE_NODE_CD(node, avail)

- 조건: node ≠ head(더미)

- node.prev.next ← node.next
- node.next.prev ← node.prev
- PUSH_FREE(avail, node)

예제: 다항식 저장하기

- $2x^2 + 3x$를 저장하려면,
- 1. 항 노드를 만든다 (계수 3, 지수 1, 다음 없음)
- 2. 항 노드를 만든다 (계수 2, 지수 2, 다음 3x)

항 노드

- coeff: 계수
- exp: 지수
- next: 다음 항

CREATE(coefficient, exponent, next_node)

- node ← 새로운 항 노드
- node.coeff ← coefficient
- node.exp ← exponent
- node.next ← next_node
- 반환: node

INSERT_TAIL(tail, coefficient, exponent)

- tail.next ← CREATE(coefficient, exponent, NULL)

ADD(X, Y) 1/5

- X, Y는 (exp 내림차순, 동차 지수 없음) 리스트로 가정

- dummy ← 항 노드
- dummy.next ← NULL
- tail ← dummy
- 반복: X != NULL 그리고 Y != NULL

ADD(X, Y) 2/5

- 만약: X.exp > Y.exp

  INSERT_TAIL(tail, X.coeff, X.exp)

  X ← X.next

  tail ← tail.next

  다음 반복으로

ADD(X, Y) 3/5

- 만약: X.exp < Y.exp

  INSERT_TAIL(tail, Y.coeff, Y.exp)

  Y ← Y.next

  tail ← tail.next

  다음 반복으로

ADD(X, Y) 4/5

- sum ← X.coeff + Y.coeff
- 만약: sum != 0

  INSERT_TAIL(tail, sum, Y.exp)

  tail ← tail.next
- X ← X.next
- Y ← Y.next
- 다음 반복으로

ADD(X, Y) 5/5

- 만약: X != NULL

  tail.next ← X

- 그외:

  tail.next ← Y

- 반환: dummy.next

요약

- 링크드 리스트에서 데이터 추가/삭제
- 더미, 원형구조, 삭제 노드 리스트 활용
- 다항식 예제
- 다음 시간: 링크드 리스트를 C 언어로 구현해 본다

MUL(X, Y) 1/2

- A ← NULL

- 반복: px ← X부터 끝까지

  dummy ← 항 노드

  dummy.next ← NULL

  tail ← dummy

MUL(X, Y) 2/2

- 반복: py ← Y부터 끝까지

  INSERT_TAIL(tail,

  px.coeff * py.coeff,

  px.exp + py.exp)

  tail ← tail.next
- A ← ADD(A, dummy.next)
- 반환: A

INVERT(head)

- 포인터 방향을 모두 뒤집어 head↔tail을 바꾼다
- (prev, cur, nxt) 3포인터로 순차 갱신: cur.next ← prev

INVERT(head) 1/2

- 만약: head.next == NULL

  반환: head

- prev ← head
- cur ← head.next
- prev.next ← NULL // 첫 링크를 먼저 끊어 순환 방지

INVERT(head) 2/2

- 반복: cur != NULL인 동안

  nxt ← cur.next

  cur.next ← prev

  prev ← cur

  cur ← nxt

- 반환: prev

CONCATENATE(X, Y)

- Y의 모든 노드를 X의 끝에 붙인다

- 만약: X == NULL → 반환: Y
- tail ← X
- 반복: tail.next != NULL

  tail ← tail.next

- tail.next ← Y