자료구조 과제 4

20231609 정희선

Problem 1.

sparse matrix를 linked list를 사용하여 관리하는 프로그램이다.

- **data structure**

1. Matrix node

각 node는 matrix의 header node 또는 실제 matrix item을 나타낼 수 있다. tagfield type의 tag는 node가 header인지 item인지 구분한다. down과 right pointer는 각각 다음 row와 다음 column으로 connection된다. union structure는 node가 header일 때 다음 header node를 가리키거나, item일 때는 row, column, value 정보를 저장한다.

2. hnode array

각 header node에 대한 pointer array로, matrix의 각 row와 column을 관리한다.

- **Algorithm**

1. mread()

파일로부터 matrix 데이터를 읽어들여 linked list 형태로 sparse matrix를 구성한다.
matrix의 크기와 nonzero entry의 수를 먼저 읽은 후, 각 entry에 대한 information를 읽어서 적절한 위치에 node를 삽입한다.
각 row와 column의 header node를 초기화하고, 모든 row와 column에 대해 linked list를 생성한다.

2. mwrite()

matrix를 파일로 출력한다. linked list를 따라가면서 각 row의 entry들을 파일에 순차적으로 쓴다.

3. mearase()

생성된 sparse matrix의 메모리를 해제한다. 각 row와 column에 대해 할당된 node들을 순차적으로 해제한다.

4. mtranspose()

sparse matrix의 transpose를 계산한다.
기존 matrix를 파일로 쓰고, 다시 읽어들인 후에 row와 column을 교환하여 새로운 linked list를 생성한다.
transposed matrix는 새로운 연결 structure로 재구성되어 파일에 저장되고, 다시 읽힌다.

Problem 2

이 프로그램은 polynomial을 linked list를 사용하여 표현하고 다룬다.

- **data structure**

1. polyNode

각 node는 polynomial의 한 term을 나타낸다. 이 struct는 coef, expon, 그리고 다음 node를 가리키는 link로 구성된다.

2. poly_pointer

polyNode struct를 가리키는 pointer다. 이 pointer는 polynomial의 개별 term을 연결하는 데 사용된다.

- **Algorithm**

1. makeP()

file로부터 data를 읽어 linked list를 생성한다. 각 term의 coef와 expon를 읽어 list에 insertion한다.

2. attach()

주어진 coef와 expon를 가진 새로운 node를 생성하고, 주어진 list의 마지막에 이 node를 추가한다.

3. padd()

두 polynomial을 더하는 function이다. 두 list를 traversal하면서 expon이 같은 term은 coef를 더하고, expon이 다른 term은 그대로 새 list에 추가한다.

4. pmult()

두 polynomial을 곱하는 function이다. 첫 번째 polynomial의 각 term에 대하여 두 번째 polynomial의 모든 term과 multiplication을 수행하고, result를 누적하여 최종 polynomial을 구성한다. 이 때, 각 multiplication result는 padd function을 통해 누적된 result polynomial에 더해진다.

5. pwrite()

polynomial을 file로 output하는 function이다. linked list를 traversal하면서 각 term의 coef와 expon를 file에 쓰고, polynomial의 전체 term 개수도 file에 기록한다.

Problem 3

이 프로그램은 maze를 탐색하고 path를 찾는다.

- **data structure**

1. struct element

Maze 내의 각 location과 direction을 나타내는 데 사용된다. row와 col은 현재 position을, dir은 next 이동 direction을 표시한다.

2. struct offsets

이동할 수 있는 8가지 direction을 define한다. 각 direction은 vertical과 horizontal의 변화량으로 표현된다.

3. struct Node

doubly circular linked list에서 각 node를 정의한다. 각 node는 element struct를 item으로 갖고, 이전과 다음 node를 가리키는 link가 있다.

4. 2-dimension array maze, mark

Maze array는 각 cell에 wall(1) 또는 path(0)의 정보를 저장한다. Mark array는 해당 location을 visit했는지의 여부를 표시하는 데 사용된다.

- **Algorithm**

1. path ()

Maze의 entrance(1,1)에서 시작하여 destination까지 path를 찾는다. 각 step에서 doubly(circular) linked list에 현재 position을 저장하고, 가능한 모든 direction을 explore하여 path를 찾는다.
만약 다음 이동 position이 wall이 아니고 아직 visit하지 않은 position이라면, 그 location으로 이동하고 현재 position과 direction 정보를 circular list에 저장한다. 이 과정을 반복하면서 destination에 도달하면 path를 output한다.

2. dinsert ()

doubly(circular) linked list에 새 node를 insertion하는 기능을 수행한다. List가 비어있으면 자기 자신을 가리키게 하고, 그렇지 않으면 list의 마지막에 insertion한다.

3. ddelete()

doubly(circular) linked list에서 node를 remove하고, removed된 node의 element 정보를 return한다.

4. printCircularList()

최종 path를 file로 output한다. doubly(circular) linked list를 traverse하면서 각 position의 coordinates를 file에 기록한다.