자료구조 (Data Structure)
3주차: 스택

데이터 접근에 제한을 둔 자료구조

- 리스트: 모든 데이터에 순서로 접근 가능
- 데이터를 하나씩 꺼낼 수만 있으면 될 때도 있다.
- 데이터를 꺼낼 때는 정해진 규칙에 따라서 꺼낸다
  - 추가된 순서대로 꺼내기
  - 최신 데이터부터 꺼내기
  - 우선순위에 따라 꺼내기

스택

- 특징: 최신 데이터(top)부터 꺼낸다
- 정의: 리스트의 특별한 케이스. 한쪽 끝에서만(top) 추가/삭제 가능
- 기능: 데이터 추가(PUSH), 최신 데이터 읽기(PEEK) / 꺼내기(POP)

단방향 링크드 리스트로 구현한 스택

- 처음 노드가 마지막 노드보다 추가/삭제가 쉽다
- top: 리스트의 처음 노드
- PUSH: 단방향 링크드 리스트의 INSERT_FIRST
- PEEK: top의 데이터를 반환
- POP: PEEK 후 DELETE_FIRST

스택에 저장된 정보

- top: 현재 스택에서 가장 최근에 추가된 노드
- size: 노드 개수(항상 $0 \leq size$)
- 유효 구간: top부터 next로 서로 연결된 size개 노드

노드(Node)

- 데이터와 다음 노드 정보의 묶음
- val: 저장된 데이터
- next: 다음 노드 정보(마지막 노드는 NULL)

NODE_CREATE(val, next)

- 새 노드를 만들고 정보를 저장해서 반환한다

- 동작: x에 새로운 노드를 저장한다

  x의 val에 매개변수 val을 저장한다

  x의 next에 매개변수 next를 저장한다

- 반환: x

PUSH(stack, val)

- 데이터가 val인 새 노드를 만들고 top 앞에 연결한다

- 동작: x에 NODE_CREATE(val, NULL)를 저장한다
  x의 next에 stack의 top을 저장한다
  stack의 top에 x를 저장한다
  stack의 size를 1 증가시킨다

PUSH(stack, val)의 시간 효율

- NODE_CREATE: 상수 시간 가정
- 포인터 대입(연결 변경): 상수 시간
- 결론: 상수 시간

POP(stack)

- stack의 top을 삭제하고, 데이터를 반환한다
- 조건: stack이 비어있지 않음
- 동작: old_t에 stack의 top을 저장한다

  v에 old_t의 val을 저장한다

  stack의 size를 1 감소시킨다

  stack의 top에 old_t의 next를 저장한다

  old_t를 삭제한다
- 반환: v

POP(stack)의 시간 효율

- 포인터 한 번 교체 + free: 상수 시간
- 결론: 상수 시간

PEEK(stack)

- 삭제 작업 없이 top에 있는 데이터를 조회만 한다
- 조건: stack이 비어있지 않음
- 반환: stack의 top의 val

PEEK(stack)의 시간 효율

- top의 val에 접근: 상수 시간
- 결론: 상수 시간

stack 경계/에러 처리

- PUSH: 노드의 메모리 할당에 실패할 수 있음
- POP: stack의 top이 NULL이면, UNDERFLOW
- PEEK: stack의 top이 NULL이면, UNDERFLOW
- 새로운 노드나 top에만 접근하므로 상수 시간

배열 리스트로 구현한 스택

- 배열 리스트에서는 마지막 원소가 첫번째 원소보다 추가/삭제가 쉽다
- PUSH: 배열 리스트의 INSERT(A, A.size + 1)
- POP: 배열 리스트의 DELETE(A, A.size)
- PEEK: 배열 리스트의 RETRIEVE(A, A.size)
- 링크드 리스트처럼 첫번째 원소를 top으로 하려면...

배열로 구현한 스택

- 데이터를 배열의 오른쪽부터 순서대로 저장한다
- data: 배열
- capacity: 최대 저장량
- top: 마지막에 추가된 데이터의 인덱스
  (스택이 비어있으면, top == capacity + 1)

ASTACK 상태/크기

- 비어있음 : top == capacity + 1
- 가득 차있음 : top == 1
- size : capacity - top + 1

ASTACK_INITIALIZE(stack, cap)

- cap개 데이터를 저장가능하게 stack을 초기화한다

- 조건: cap > 0
- 동작: stack의 data에 새로운 배열을 저장한다

  (배열 크기: cap + 1)

  stack의 capacity에 cap을 저장한다

  stack의 top에 cap + 1을 저장한다

ASTACK_PUSH(stack, val)

- stack에 val을 추가한다
- 조건: stack의 top > 1
- 동작: stack의 data[stack의 top - 1]에 val을 저장한다
  stack의 top을 1 감소시킨다

ASTACK_PUSH(stack, val)의 시간 효율

- 배열에 대입 1번
- 결론: 상수 시간

ASTACK_POP(stack)

- stack에서 top을 삭제하고 데이터를 반환한다
- 조건: stack이 비어있지 않음 (top ≤ capacity)
- 동작: v에 stack의 data[stack의 top]을 저장한다
  stack의 top을 1 증가시킨다
- 반환: v

ASTACK_POP(stack)의 시간 효율

- 배열에서 데이터 읽기 1번
- 결론: 상수 시간

ASTACK_PEEK(stack)

- top의 데이터를 반환한다
- 조건: stack이 비어있지 않음 (top ≤ capacity)
- 반환: stack의 data[stack의 top]

ASTACK_PEEK(stack)의 시간 효율

- 배열에서 데이터 읽기 1번
- 결론: 상수 시간

ASTACK 경계/에러 처리

- PUSH: if (top == 1) OVERFLOW (저장할 공간 없음)
- POP: if (top == capacity + 1) UNDERFLOW (없음)
- PEEK: if (top == capacity + 1) UNDERFLOW (없음)
- 모든 연산은 배열 한 칸 접근만 하므로 상수 시간

예제: 미로 찾기 문제

- 시작점 s부터 도착지 d까지 경로 찾기
- *은 막힌 지점, 비어 있는 곳은 방문 가능
- 미로 예시:

```
**********
*s  *    **
** * ** **
*    *    *
**** *** *
*       d*
**********
```

미로에서 출구를 찾기 위한 규칙

- 간 곳은 표시해놓는다(-)
- 갈 수 있는 곳은 표시해놓고(o) 언젠가 간다
- 시작점 처리: 다음 선택지는 o 한 개 있음

```
**********
*-o *    **
** * ** **
*    *    *
**** *** *
*        d*
**********
```

다음에 어디로 갈지 선택하는 규칙

- 갈 곳들을 저장한 자료구조에서 갈 곳을 하나 꺼내서 거기로 간다
- 유일했던 선택지 처리: 다음 선택지는 o 두 개 있음

```
**********
*--o*    **
**o* ** **
*    *   *
**** *** *
*       d*
**********
```

막다른 곳의 규칙

- o가 나올 때까지 돌아가며 표시해놓는다(x)

- 오른쪽으로 갔다가 다시 돌아옴

```
**********
*--x*    **
**o* ** **
*    *   *
**** *** *
*       d*
**********
```

종료 조건

- d에 도달하면 종료 (찾음)
- 갈 곳 o가 없으면 종료 (못찾음)

```
**********
*--x*    **
**-* ** **
* ---*    *
****-*** *
*    ----d*
**********
```

미로 찾기를 하며 저장할 정보

- o 모음 (갈 곳들): 스택 (최신 위치가 가장 가깝다)
- - 모음 (간 곳들): 리스트 (경로 확인용)
- x,-,* 모음 (안갈 곳들): 미로 정보에 표시

미로 정보: 2차원 배열 (배열의 배열)

- 미로[i][j]: (행 i, 열 j) 위치의 상태
- 0 : 가도 됨
- 1 : 갈 수 없거나 이미 갔음
- 처음 주어진 미로 정보를 수정해도 된다고 가정

좌표 구조체

- row: 행 좌표
- col: 열 좌표
- prev_size: 현재 점을 탐색 경로에 추가하기 직전에 경로의 길이 (시작점은 0)

STATUS(maze, point)

- 반환: maze[point의 row][point의 col]

// 0 : 처리하지 않은 통로
// 1 : 벽/이미 방문/방문 예정 (모두 방문 계획 금지)

MARK(maze, point)

- maze[point의 row][point의 col]에 1 저장

// 방문했거나 방문 예정 표시

좌표 리스트/스택

- points: 좌표 구조체의 배열
- size: 저장된 좌표 개수
- capacity: 최대 저장량

CREATE(cap)

- list에 새 좌표 리스트/스택 저장
- list의 points에 새 배열 저장 (크기: cap + 1)
- list의 capacity에 cap 저장
- list의 size에 0 저장
- 반환: list

PUSH(list, point)

- 조건: list의 size < list의 capacity
- list의 points[size + 1]에 point 저장
- size 1 증가

POP(list)

- 조건: list의 size > 0
- tmp에 list의 points[size] 저장
- list의 size 1 감소
- 반환: tmp

DELETE_TO_SIZE(list, new_size)

- 조건: 0 ≤ new_size ≤ list의 size
- list의 size를 new_size로 변경

미로찾기 함수

- PUSH_NBRS(maze, to_visit, point)

  미로 maze에서 point의 이웃 점들 중에 아직 확인하지 않은 점들을 스택 to_visit에 추가한다.

- FIND(maze, start, end, len)

  가로세로 크기 len인 미로 maze에서 시작점 start에서 도착점 end까지 경로를 반환한다.

PUSH_NBRS(maze, to_visit, point)

- 현재 위치 point의 인접 좌표 중에서 아직 갈 계획도 없었던 좌표들을 to_visit에 추가한다
- maze: 미로 정보 (갈 수 없거나 확인한 점이 표시됨)
- to_visit: 갈 곳들(스택)
- point: 현재 위치
- 이동은 상/하/좌/우 4-이웃만 허용

PUSH_NBRS(maze, to_visit, point) 동작

- n_added에 0 저장
- 반복: point의 상하좌우 좌표 next에 대하여
  만약: STATUS(maze, next) == 0 이면
    // next의 prev_size는 point까지 경로 길이
    next의 prev_size에 point의 prev_size + 1 저장
    MARK(maze, next)
    PUSH(to_visit, next)
    n_added 1 증가
- 반환: n_added // 추가된 좌표 개수

STATUS/MARK 시점 정리

- STATUS==0인 이웃만 후보로 본다
- 후보를 스택에 넣기 전에 MARK=1로 바꿔 '방문 예정' 표시
- 효과: 중복 PUSH 방지(각 칸은 최대 한 번 스택에 들어감)

FIND(maze, start, end, len)

- start에서 end까지 가는 경로를 찾아서 반환한다
- maze: 미로 정보 (갈 수 없거나 확인한 점이 표시됨)
- start: 시작점 좌표
- end: 도착점 좌표
- len: 정사각형 미로의 한 변 길이 (행과 열 개수)

FIND(maze, start, end, len) 가정

- 미로는 정사각형, 한 변 길이는 len
- 시작점과 도착점은 미로 내부의 통로(0)
- 가장자리는 모두 1(벽)로 둘러싸여 있어서 범위체크 생략 가능
- 미로 정보는 수정해도 됨
- start ≠ end

FIND(maze, start, end, len) 변수

- to_visit: 갈 곳 후보 스택(최근 이웃부터 처리)
- maze에 표시된 좌표는 to_visit에 추가하지 않음
- path: 시작점부터 현재 좌표까지의 경로 리스트.
- path는 다음 좌표로 갈 때마다 갱신됨
- going_back: 막다른 곳에서 거꾸로 왔는지 체크.
- 거꾸로 온 경우 path를 되돌려야 함

FIND(maze, start, end, len) 특징

- to_visit에 있는 좌표들은 모두 maze에서 1(방문 예정)
- path에 있는 좌표들도 모두 maze에서 1(path에 추가되기 전 to_visit에 있었음)
- 막다른 길에서 돌아갈 때 시작점부터 돌아간 점까지 path가 보존되어 있음(스택의 특성)

FIND(maze, start, end, len) 동작

- to_visit에서 갈 좌표를 하나 꺼낸다.
- 꺼낸 좌표가 도착점이면, path를 반환하고 종료.
- 꺼낸 좌표의 이웃 좌표들을 PUSH_NBRS로 to_visit에 추가.
- to_visit에 좌표가 있는 한 계속 반복.
- to_visit이 비어있으면, NULL 반환(못찾음).

FIND(maze, start, end, len) 1/4

- to_visit에 CREATE(len * len) 저장 // 갈 곳들
- path에 CREATE(len * len) 저장 // 경로 기록
- going_back에 0 저장 // 막다른 곳 체크
- start의 prev_size에 0 저장
- MARK(maze, start)
- PUSH(to_visit, start)

FIND(maze, start, end, len) 2/4

- 반복: to_visit이 비어있지 않음

  point에 POP(to_visit) 저장

  만약: going_back == 1

    DELETE_TO_SIZE(path, point의 prev_size)

  PUSH(path, point)

FIND(maze, start, end, len) 3/4

- 만약: point와 end의 좌표가 동일

  반환: path // 도착점 포함 경로

- n에 PUSH_NBRS(maze, to_visit, point) 저장
- 만약: n == 0

  going_back에 1 저장 (막다른 곳 처리)

FIND(maze, start, end, len) 4/4

- 그외:

  going_back에 0 저장

- 다음 반복으로

- 반환: NULL (못찾음)

미로 찾기 복잡도 분석

- 시간: ~ len^2 각 칸은 최대 한 번 방문/마킹
- 공간: ~ len^2 스택(to_visit)과 path(최대 칸 수)

이웃 탐색 순서와 경로 형태

- 스택은 이웃을 추가하는 순서에 따라 같은 미로라도 다른 경로가 나올 수 있다
- 예: 상/하/좌/우 순서로 이웃을 추가하면, 오른쪽으로 쭉 간다
- 예: 우/상/좌/하 순서로 이웃을 추가하면, 아래로 쭉 간다

요약

- 스택: 최신 데이터부터 꺼내는 자료구조
- 단방향 링크드 리스트/배열로 구현 가능
- 모든 연산이 상수 시간
- 미로 찾기: 스택을 사용해서 경로 탐색

과제

- 미니 퀴즈의 연장선
- 제출 기한: 9월 26일 금요일 수업시간
- 제출 방법: 이메일로 pdf 파일 제출

  수업시간에 미니퀴즈와 함께 제출