그리디, 자료구조 (1)

25.05.05 (월) 온라인

목차

그리디

그리디 | 01 개요

- **그리디**
  - Greedy : [형용사] 탐욕스러운, **[알고리즘] 현재 상황에서 가장 좋아 보이는 선택만을 반복하는 방식**
  - 현재 상황에서 가장 좋아 보이는 선택만을 반복해서 최종 해답을 구하는 알고리즘
  - 현재 상태에서 가장 좋은 선택(국소 최적해)이 전체 범위에서도 가장 좋은 선택(전역 최적해)이라면, 그리디 기법을 사용할 수 있음.
- **그리디 vs DP**
  - 그리디 : 매 순간 국소 최적 선택을 해 나감. 탐욕적 선택이 항상 최선인지 증명이 필요함.
  - DP : 모든 경우를 비교해서 최적 선택을 함. 점화식만 올바르면 자동으로 최적해가 보장됨.
- **정당성 증명**
  - 정당성 증명 : 탐욕적으로 선택한 해가 최적해임을 증명하는 것을 말한다. 귀류법으로 증명할 수 있다.
  - 귀류법 : 명제의 결론이 부정이라고 가정했을 때, 모순이 발생함을 보여 원래 명제가 참임을 증명하는 논증법이다.
    - 즉, 그리디한 선택이 최적해가 아님을 가정한 후 모순을 유도함으로써 그리디한 선택이 최적해임을 증명하는 것이다!

그리디 | 01 개요

귀류법 예시 | "루트 2가 무리수임을 증명하라"

1. 루트 2가 유리수라고 가정하자.
2. 서로소인 정수 a, b에 대해 '루트 2 = a / b' 라 표현할 수 있다.
3. 양 변을 제곱하면, 2 = a^2 / b^2 -> a^2 = 2 * b^2 이다. a^2는 짝수임을 알 수 있다.
4. a^2이 짝수이면, a도 짝수다. 따라서, a = 2k라 놓을 수 있다. (k는 임의의 정수)
5. a = 2k를 대입하면 (2k)^2 = 2 * b^2 -> b^2 = 2k^2이다. b도 짝수임을 알 수 있다.
6. a, b는 모두 2로 나누어 떨어지므로 서로소라는 가정에 모순된다.
7. 따라서 루트 2는 무리수이다.

그리디 | 02 예제 (1) : 백준 11047번 동전 0

- **문제 요약**
  - N종류의 동전을 가지고 있고, 각각의 동전을 매우 많이 가지고 있다. 1원짜리 동전은 항상 가지고 있다.
  - 동전을 적절히 사용해서 그 가치의 합을 K로 만들려고 한다. 이때 필요한 동전 개수의 최솟값을 구해라.
- **입력**
  - 첫째 줄에 N과 K가 주어진다. $(1 \le N \le 10, 1 \le K \le 100{,}000{,}000)$
  - 둘째 줄부터 N개의 줄에 동전의 가치 A_i가 오름차순으로 주어진다. $(1 \le A_i \le 1{,}000{,}000, A_1 = 1, i \ge 2$인 경우에 A_i는 A_{i-1}의 배수)
- **출력**
  - 첫째 줄에 K원을 만드는데 필요한 동전 개수의 최솟값을 출력한다.

그리디 | 02 예제 (1) : 백준 11047번 동전 0

- **풀이**
  - **가격이 큰 동전부터 최대한 많이 사용하는 전략**을 취해본다.
- **정당성 증명**
  - 귀류법을 사용하여, 위 전략이 항상 최적이 아니라고 가정하자.
  - 즉, 어떤 금액 K에 대해 위 전략으로 만든 동전 리스트 G보다 더 적은 개수의 동전 리스트 O가 존재한다고 가정하자.
    - 가치를 기준으로 내림차순으로 동전을 정렬한다. -> C[i] = {i번째로 큰 동전의 가치}
    - G, O가 처음으로 다른 지점 i가 존재한다. 비싼 동전부터 최대한 많이 가져가는 전략에 의해 G[i] > O[i]이다.
    - C[i] * (G[i] – O[i])원 만큼의 차이를 채우기 위해 C[i]보다 싼 동전을 사용할 텐데, C{i]의 약수면서 싼 동전을 C[i] * (G[i] – O[i])보다 많은 동전이 필요하므로 O는 B보다 동전을 더 적게 사용할 수 없다.
    - 따라서 더 적은 개수의 동전 리스트는 존재하지 않으므로, 전략은 최적이다.
  - 가치가 큰 동전을 작은 동전을 대체하면, 동전 수는 줄어들 수 없으므로 모순이 발생한 것이다.
  - 따라서, 애초에 그런 O는 존재하지 않는다.

그리디 | 02 예제 (1) : 백준 11047번 동전 0

- 구현

```cpp
#include <iostream>
using namespace std;

int main() {
    int n, k, a[10], cnt = 0;
    cin >> n >> k;
    for (int i = 0; i < n; ++i) cin >> a[i];
    for (int i = n - 1; i >= 0; --i) {
        cnt += k / a[i];
        k %= a[i];
    }
    cout << cnt;
}
```

그리디 | 02 예제 (2) : 백준 11399번 ATM

- **문제 요약**
  - ATM 기계 앞에 N명이 줄을 서 있다.
  - 각 사람 i가 돈을 인출하는 데 Pi분이 걸린다.
  - 앞 사람이 끝나야 다음 사람이 인출할 수 있어서, 기다리는 시간이 누적된다.
  - 입력으로 N과 P_1, P_2, …, P_n이 주어질 때, 돈을 인출 하는데 시간의 총합의 최솟값을 구하라.
- **입력**
  - 첫째 줄에 사람의 수 N($1 \le N \le 1,000$)이 주어진다. 둘째 줄에는 각 사람이 돈을 인출하는데 걸리는 시간 Pi가 주어진다. ($1 \le Pi \le 1,000$)
- **출력**
  - 첫째 줄에 각 사람이 돈을 인출하는데 필요한 시간의 합의 최솟값을 출력한다.

그리디 | 02 예제 (2) : 백준 11399번 ATM

- **풀이**
  - **돈을 인출하는데 걸리는 시간이 작은 사람부터 줄을 서는 전략**을 취해보자.
- **정당성 증명**
  - 위 전략이 최적이 아니라고 가정하자.
    - P를 위 전략으로 구성한 순서를 담은 리스트라고 하자.
    - 최적이 아니라는 것은, i <= j인 P[i]와 P[j]에 대해 P[j]가 더 먼저 인출하는 게 최적이라는 의미다.
    - Time(P[i]) <= Time(P[j])이므로, 순서를 바꾸면 총합이 커진다. 모순이 발생했다.
      - P[i] -> P[j] 순서일때, P[i]는 앞선 모든 사람의 시간과 P[i]만 기다리면 된다.
      - P[j] -> P[i] 순서일때, P[j]는 앞서 모든 사람의 시간과 P[j]만 기다리면 된다.
    - 따라서, 돈을 인출하는데 걸리는 시간이 작은 사람부터 정렬하는 것이 최적의 전략이다.

그리디 | 02 예제 (2) : 백준 11399번 ATM

- 구현

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int n, a[1000], sum = 0, res = 0;
    cin >> n;
    for (int i = 0; i < n; ++i) cin >> a[i];
    sort(a, a + n);
    for (int i = 0; i < n; ++i) {
        sum += a[i];
        res += sum;
    }
    cout << res;
}
```

그리디 | 02 예제 (3) : 백준 2217번 로프

- **문제 요약**
  - 총 N개의 로프가 있다.
  - 각 로프는 최대 w_i 킬로그램까지 들 수 있다.
  - k개의 로프를 병렬로 연결하면, 물체의 무게 w는 고르게 나뉘어 각 로프에 w/k의 하중이 걸린다.
  - 모든 로프를 사용할 필요는 없다.
  - 이 로프들을 이용하여 들어올릴 수 있는 물체의 최대 중량을 구해내는 프로그램을 작성해라.
- **입력**
  - 첫째 줄에 정수 N이 주어진다. 다음 N개의 줄에는 각 로프가 버틸 수 있는 최대 중량이 주어진다. 이 값은 10,000을 넘지 않는 자연수이다.
- **출력**
  - 답을 출력하라.

그리디 | 02 예제 (3) : 백준 2217번 로프

- **풀이**
  - 임의의 K개의 로프를 이용해 들 수 있는 무게는 min({K개의 로프 각각이 버틸 수 있는 중량}) * K 이다.
  - 이를 이용해 다음과 같은 전략을 생각할 수 있다. 먼저, 로프를 내림차순으로 정렬한다.
  - i번째 로프까지 사용할 경우, 가능한 최대 중량은 w[i] * (i + 1)이다. 이 값을 전부 계산해보고, 그 중 최댓값을 출력하면 된다.

그리디 | 02 예제 (3) : 백준 2217번 로프

- **정당성 증명**
  - k개의 로프를 사용할 때, 가장 약한 로프가 전체 중량의 하중을 제한한다. 즉, {k개의 로프 중 가장 작은 하중} x k가 최대 무게이다.
    1. 로프를 정렬한 상태(weight[0] ≥ weight[1] ≥ … ≥ weight[n-1])에서 앞에서 부터 k개의 로프를 선택했을 때 최대 하중은 weight[k – 1] x k 이다. 왜냐하면 weight[k – 1]이 가장 약한 로프이기 때문이다.
    2. 모든 k(1 ≤ k ≤ n)에 대해 이 값을 계산하고, 그 중 최댓값이 가능한 모든 조합 중 최적의 값이다.
  - **연속된 상위 k개만 보는 것이 항상 최선의 전략**이다.
    - 상위 k개를 보는 것이 최선이 아니라고 가정하자.
    - 원소가 k개인 로프의 집합 A에 대해 min(A) * k는 상위 k개를 보는 것보다 항상 작다.
    - 모순이 발생했으므로, 연속된 상위 k개만 보는 것이 항상 최선이다.

그리디 | 02 예제 (4) : 백준 1931번 회의실 배정

- **문제 요약**
  - 한 개의 회의실과 N개의 회의가 주어진다.
  - 각 회의는 시작 시간과 종료 시간이 있으며, 한 회의실에서 하나의 회의만 진행 가능하다.
  - 회의는 시작 시간과 종료 시간이 같아도 된다.
  - 겹치지 않게 선택할 수 있는 회의의 최대 개수를 구해라.
- **입력**
  - 첫째 줄에 회의의 수 $N(1 \le N \le 100,000)$이 주어진다. 둘째 줄부터 $N + 1$ 줄까지 각 회의의 시작 시간과 끝나는 시간이 주어진다.
- **출력**
  - 겹치지 않게 선택할 수 있는 회의의 최대 개수를 구해라.

그리디 | 02 예제 (4) : 백준 1931번 회의실 배정

- **풀이**
  - **가장 먼저 끝나는 회의를 선택하는 것이 항상 최적해로 이어진다.**
- **정당성 증명**
  - 최적해 S가 있다고 가정하자. (가장 많은 회의를 스케줄링한 정답 집합 S)
  - 우리의 알고리즘은 $A = [a_1, a_2, \ldots, a_k]$를 선택한다. 여기서 각 a_i는 끝나는 시간 기준으로 빠른 순서대로 선택한 회의이다.
  - S와 A는 같지 않다고 가정하자. (즉, 적어도 하나의 회의는 다르다)
  - 첫 번째로 A와 S가 다르게 선택한 시점에서, S는 b라는 회의를 선택했고, A는 a를 선택했으며 a.end < b.end이다.
  - 이때 a가 b보다 일찍 끝나므로, b 대신 a를 선택해도 이후 회의 배치에 영향을 주지 않는다.
  - 즉, b를 a로 바꿔도 S는 여전히 유효한 회의 집합이며 개수도 동일하거나 더 많아질 수 있다. (S를 A와 더 유사한 형태로 만들 수 있다)
  - 이 과정을 반복하면 결국 A와 동일한 선택을 한 S'가 존재하므로, A도 최적해인 것이다.
  - 같지 않다는 가정에 모순되므로 위 전략은 최적이다.

그리디 | 02 예제 (4) : 백준 1931번 회의실 배정

- 구현

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<pair<int, int>> meetings(n);

    for (int i = 0; i < n; ++i) {
        cin >> meetings[i].second >> meetings[i].first; // 끝나는 시간, 시작 시간
    }

    sort(meetings.begin(), meetings.end()); // 끝나는 시간 기준 정렬

    int count = 0, endTime = 0;
    for (const auto& [end, start] : meetings) {
        if (start >= endTime) {
            endTime = end;
            ++count;
        }
    }

    cout << count << '\n';
    return 0;
}
```

자료구조

직접 구현하는 방법은 다루지 않고, 실전 문제 풀이에 사용할 수 있을 정도로 간단히 다룹니다!

자료구조 | 00 자료구조란?

- **자료구조는 데이터를 저장하고, 조작하는 방법을 제공한다.**
  - 스택 : 마지막에 삽입된 데이터가 가장 먼저 삭제되는 연산을 지원하는 자료구조
    - DFS 알고리즘
  - 큐 : 먼저 삽입된 데이터가 먼저 삭제되는 연산을 지원하는 자료구조
    - BFS, 시뮬레이션 알고리즘
  - 덱 : 양쪽 끝에서 삽입과 삭제가 가능한 연산을 지원하는 자료구조
    - 슬라이딩 윈도우 알고리즘
  - 트리 : 부모-자식 관계 기반 탐색과 삽입, 삭제 연산을 지원하는 자료구조
    - 세그먼트 트리(자료구조)
  - 힙 : 최댓값/최솟값을 즉시 꺼내는 연산을 지원하는 자료구조
    - 우선순위 큐(자료구조), 다익스트라 알고리즘
  - 해시 테이블 : 키를 통한 빠른 삽입, 검색, 삭제 연산을 지원하는 자료구조
  - **이외에도 많다!**

자료구조 스택 | 01 정의와 개념 소개

- **정의**
  - stack은 가장 마지막에 넣은 원소가 가장 먼저 빠져나오는 자료구조이다.
  - LIFO (Last In, First Out) : 나중에 들어간 것이 먼저 나온다.
  - 예시) 재귀함수, 웹 브라우저 뒤로 가기 등
- **연산**

  모든 연산은 평균적으로 O(1)의 시간복잡도를 가진다.
  - push(x) : 스택 위에 x를 추가
  - pop() : 스택 가장 위의 값을 제거
  - top() : 스택 가장 위의 값을 조회
  - empty() : 스택이 비어 있는지 확인
  - size() : 스택에 쌓인 원소 개수 반환

자료구조 스택 | 01 정의와 개념 소개

- **std::stack<T> name;**
  - C++에서 stack은 <stack> 헤더에 정의되어 있고, STL(Standard Template Library)에서 제공하는 컨테이너이다.
- **사용법**

```cpp
#include <iostream>
#include <stack>
using namespace std;

int main() {
    stack<int> s;

    // 1. push(x): 값 추가
    s.push(10);
    s.push(20);
    s.push(30);

    // 2. top(): 맨 위 값 조회
    cout << "Top: " << s.top() << endl; // 30

    // 3. pop(): 맨 위 값 제거
    s.pop(); // 30 제거
    cout << "Top after pop: " << s.top() << endl; // 20

    // 4. size(): 원소 개수
    cout << "Size: " << s.size() << endl; // 2

    // 5. empty(): 비었는지 확인
    if (s.empty()) {
        cout << "Stack is empty.\n";
    } else {
        cout << "Stack is not empty.\n";
    }
```

```cpp
    // 6. emplace(x): 스택에 값 추가 (push와 유사)
    s.emplace(40);
    cout << "Top after emplace: " << s.top() << endl; // 40

```

자료구조 스택 | 02 예제 (1) : 백준 10828번 스택

- **문제 요약**

  다음의 5가지 명령(연산)을 지원하는 스택을 구현하라.

  1. push X: 정수 X를 스택에 넣는 연산이다.
  2. pop: 스택에서 가장 위에 있는 정수를 빼고, 그 수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.
  3. size: 스택에 들어있는 정수의 개수를 출력한다.
  4. empty: 스택이 비어있으면 1, 아니면 0을 출력한다.
  5. top: 스택의 가장 위에 있는 정수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.

- **입력**
  - 첫째 줄에 주어지는 명령의 수 N ($1 \le N \le 10,000$)이 주어진다. 둘째 줄부터 N개의 줄에는 명령이 하나씩 주어진다.

- **출력**
  - 출력해야하는 명령이 주어질 때마다, 한 줄에 하나씩 출력한다.

자료구조 스택 | 02 예제 (1) : 백준 10828번 스택

- **문제 풀이**
  - STL에 구현된 스택을 이용하면 쉽게 풀 수 있다.

```cpp
#include <stack>
#include <string>
#include <iostream>
using namespace std;
#define fastio ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);

int main(){
    fastio;

    int N, tmp; cin >> N;
    stack<int> s;
    string str;

    for(int i = 0; i < N; i++){
        cin >> str;

        if(str == "push"){
            cin >> tmp;
            s.push(tmp);
        }else if(str == "pop"){
            if(s.empty()){
                cout << "-1" << "\n";
            }else{
                cout << s.top() << "\n";
                s.pop();
            }
        }else if(str == "size"){
            cout << s.size() << "\n";
        }else if(str == "empty"){
            if(s.empty()){
                cout << "1" << "\n";
            }else{
                cout << "0" << "\n";
            }
        }else if(str == "top"){
            if(s.empty() == 1){
                cout << "-1" << "\n";
            }else{
                cout << s.top() << "\n";
            }
        }
    }
}
```

자료구조 스택 | 02 예제 (2) : 백준 9012번 괄호

- **문제 요약**
  - 괄호 문자열이 올바르게 구성되었다면 "YES"를 출력하고 그렇지 않다면 "NO"를 출력하라.
- **입력**
  - 입력의 첫 번째 줄에는 입력 데이터의 수를 나타내는 정수 T가 주어진다.
  - 각 테스트 데이터의 첫째 줄에는 괄호 문자열이 한 줄에 주어진다.
  - 하나의 괄호 문자열의 길이는 2 이상 50 이하이다.
- **출력**
  - 괄호 문자열이 올바르게 구성되었다면 "YES"를 출력하고 그렇지 않다면 "NO"를 출력하라.

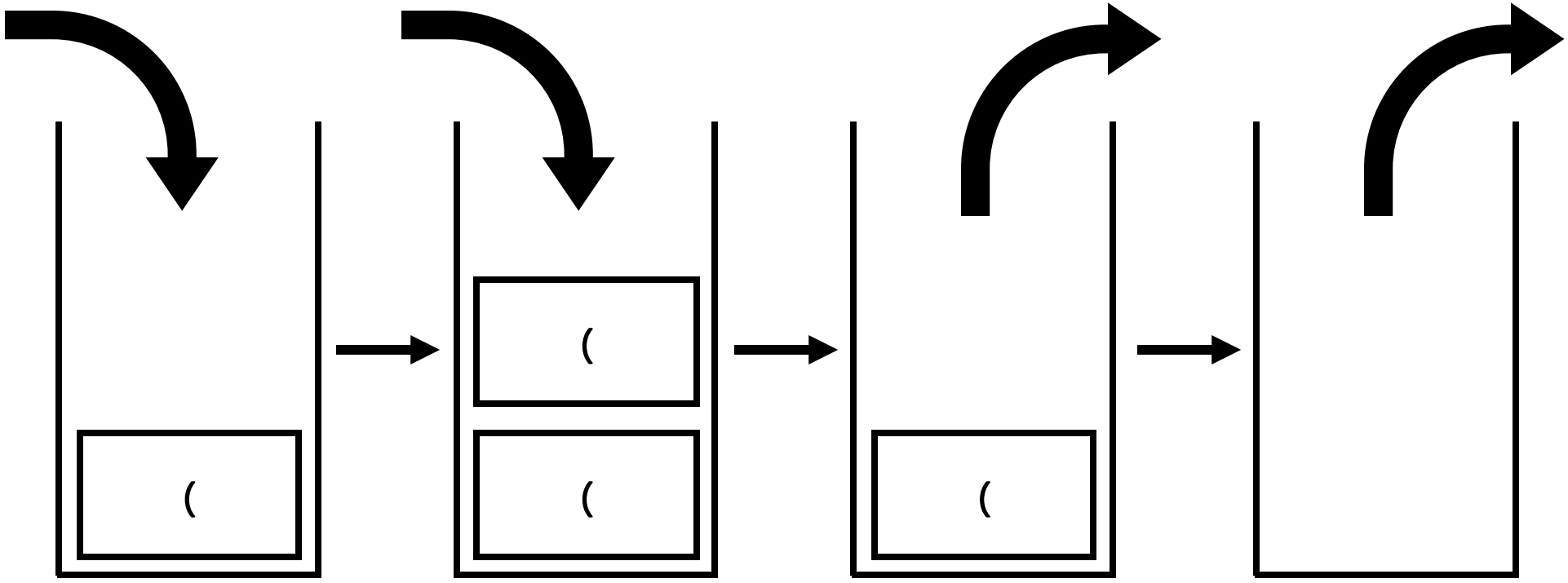
자료구조 스택 | 02 예제 (2) : 백준 9012번 괄호

- **문제풀이**

  규칙 1. 여는 괄호가 들어오면, 괄호를 스택에 넣는다.

  규칙 2. 닫는 괄호가 들어오면, 괄호가 비어있는지 검사하고 그렇지 않다면 여는 괄호를 스택에 뺀다.

  예시) 문자열 "(())"이 입력으로 주어졌을 때 스택의 변화



규칙 1 규칙 1 규칙 2 규칙 2

자료구조 스택 | 02 예제 (2) : 백준 9012번 괄호

- 구현

```cpp
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int main() {
    int T;
    cin >> T;

    while (T--) {
        string str;
        cin >> str;
        stack<char> s;
        bool isValid = true;

        for (char ch : str) {
            if (ch == '(') {
                s.push(ch);
            } else if (ch == ')') {
                if (s.empty()) {
                    isValid = false;
                    break;
                } else {
                    s.pop();
                }
            }
        }
        if (!s.empty()) isValid = false;

        cout << (isValid ? "YES" : "NO") << '\n';
    }

    return 0;
}
```

자료구조 스택 | 02 예제 (3) : 백준 17298번 오큰수

- **문제 요약**
  - 크기가 N인 수열 A = A1, A2, ..., AN이 있다.
  - 수열의 각 원소 Ai에 대해서 오큰수 NGE(i)를 구하려고 한다.
  - Ai의 오큰수는 오른쪽에 있으면서 Ai보다 큰 수 중에서 가장 왼쪽에 있는 수를 의미한다. 그러한 수가 없는 경우에 오큰수는 -1이다.
- **입력**
  - 첫째 줄에 수열 A의 크기 N ($1 \le N \le 1,000,000$)이 주어진다. 둘째 줄에 수열 A의 원소 A1, A2, ..., AN ($1 \le Ai \le 1,000,000$)이 주어진다.
- **출력**
  - 총 N개의 수 NGE(1), NGE(2), ..., NGE(N)을 공백으로 구분해 출력한다.

자료구조 스택 | 02 예제 (3) : 백준 17298번 오큰수

- **문제 풀이**
  - 예제를 풀면서, 감을 잡아보자.
  - **NEG(i) = {A[i]보다 오른쪽에 있으면서 A[i]보다 큰 수 중에서 가장 왼쪽에 있는 수}** 라고 하자.
  - NEG(3)은 -1이다. 7보다 오른쪽에 있으면서 A[i]보다 큰 수가 없기 때문이다.
    - 다른 말로 하면, 스택이 비어있기 때문에 -1을 출력한다. 스택이 비어있기에 7을 스택에 넣는다.
  - NEG(2)은 7이다. 스택의 위에 있는 값이 A[2]보다 크기 때문이다.
    - 2를 스택에 넣는다.
  - NEG(1)은 7이다. 스택의 위에 있는 값이 A[1]보다 크기 때문이다.
    - 스택 위에 있는 값이 A[1]보다 클 때까지 스택에 pop 연산을 해준다. 7를 출력하고, 5를 스택에 넣으면 된다.
  - NEG(0)은 5이다. 스택의 위에 있는 값이 A[1]보다 크기 때문이다.
- 일반화하면, 스택에는 현재까지 나온 값 중에서 '오큰수 후보'만을 남긴다.
  - 스택에 있는 값이 현재 탐색 중인 값보다 작거나 같으면 오큰수가 될 수 없으므로 꺼낸다.
  - 스택에 남아 있는 가장 위의 값이 처리 중인 값보다 크면, 그것이 오큰수이다.

예제 입력 1 복사

```
4
3 5 2 7
```

예제 출력 1 복사

```
5 7 7 -1
```

자료구조 스택 | 02 예제 (3) : 백준 17298번 오큰수

- 구현

```cpp
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

int main() {

    int n;
    cin >> n;

    vector<int> A(n);
    vector<int> answer(n);
    stack<int> s;

    for (int i = 0; i < n; ++i) {
        cin >> A[i];
    }

    for (int i = n - 1; i >= 0; --i) {
        // 오큰수가 될 수 없는 값들 제거
        while (!s.empty() && s.top() <= A[i]) {
            s.pop();
        }

        // 스택이 비었으면 오큰수가 없다
        if (s.empty()) {
            answer[i] = -1;
        } else {
            answer[i] = s.top();
        }

        // 현재 값을 스택에 push
        s.push(A[i]);
    }

    for (int i = 0; i < n; ++i) {
        cout << answer[i] << " ";
    }

    return 0;
}
```

자료구조 큐 | 01 정의와 개념 소개

- **정의**
  - queue는 선입선출(FIFO) 구조의 자료구조이다.
  - FIFO (First In, First Out) : 먼저 들어간 것이 먼저 나온다.
  - 예시) 줄 서서 입장하기
- **연산**

  모든 연산은 평균적을 O(1)의 시간복잡도를 가진다.
  - push(x) : 뒤에 넣기
  - pop() : 앞의 원소 빼기
  - front() : 가장 앞 원소 확인
  - back() : 가장 뒤 원소 확인
  - empty() : 큐가 비었는지 확인
  - size() : 큐의 원소 개수 반환

자료구조 큐 | 01 정의와 개념 소개

- **std::queue<T> name;**
  - C++에서 queue는 <queue> 헤더에 정의되어 있고, STL(Standard Template Library)에서 제공하는 컨테이너이다.
- **사용법**

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> q;

    q.push(1);
    q.push(2);
    q.push(3);  // 큐 상태: [1, 2, 3]

    cout << "front: " << q.front() << '\n'; // 1
    cout << "back: " << q.back() << '\n';   // 3
    cout << "size: " << q.size() << '\n';   // 3

    q.pop(); // 1 제거
    cout << "새 front: " << q.front() << '\n'; // 2

    // 전체 비우기
    while (!q.empty()) {
        cout << "pop: " << q.front() << '\n';
        q.pop();
    }
}
```

자료구조 큐 | 02 예제 (1) : 백준 10845번 큐

- **문제 요약**
  - 다음 6가지 연산을 지원하는 큐를 구현하라.
    - push X: 정수 X를 큐에 넣는 연산이다.
    - pop: 큐에서 가장 앞에 있는 정수를 빼고, 그 수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
    - size: 큐에 들어있는 정수의 개수를 출력한다.
    - empty: 큐가 비어있으면 1, 아니면 0을 출력한다.
    - front: 큐의 가장 앞에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
    - back: 큐의 가장 뒤에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
- **입력**
  - 첫째 줄에 주어지는 명령의 수 N ($1 \le N \le 10,000$)이 주어진다. 둘째 줄부터 N개의 줄에는 명령이 하나씩 주어진다.
- **출력**
  - 출력해야하는 명령이 주어질 때마다, 한 줄에 하나씩 출력한다.

자료구조 큐 | 02 예제 (1) : 백준 10845번 큐

- **문제 풀이**
  - STL에 구현된 큐를 이용하면 쉽게 구현할 수 있다.

```cpp
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int N;
    cin >> N;
    queue<int> q;
    string command;
    while (N--) {
        cin >> command;
        if (command == "push") {
            int x;
            cin >> x;
            q.push(x);
        } else if (command == "pop") {
            if (q.empty()) {
                cout << -1 << '\n';
            } else {
                cout << q.front() << '\n';
                q.pop();
            }
        } else if (command == "size") {
            cout << q.size() << '\n';
        } else if (command == "empty") {
            cout << q.empty() << '\n';
        } else if (command == "front") {
            if (q.empty()) {
                cout << -1 << '\n';
            } else {
                cout << q.front() << '\n';
            }
        } else if (command == "back") {
            if (q.empty()) {
                cout << -1 << '\n';
            } else {
                cout << q.back() << '\n';
            }
        }
    }
    return 0;
}
```

자료구조 큐 | 02 예제 (2) : 백준 2164번 카드2

- **문제 요약**
  - 1부터 N까지 번호가 적힌 카드가 순서대로 한 줄로 놓여 있습니다.
  - 다음의 과정을 카드가 한 장 남을 때까지 반복합니다:
    1. 제일 위에 있는 카드를 버립니다.
    2. 그 다음 제일 위에 있는 카드를 제일 아래로 옮깁니다.
  - 마지막에 남는 카드의 번호를 출력합니다.
- **입력**
  - 정수 N(1 <= N <= 500,000)이 주어진다.
- **출력**
  - 남게 되는 카드의 번호를 출력한다.

자료구조 큐 | 02 예제 (2) : 백준 2164번 카드2

- **문제 풀이**
  - 1부터 N까지의 수를 순서대로 큐에 넣고, 다음 과정을 큐의 크기가 1이될 때까지 진행하면 된다.
    1. 제일 위에 있는 카드를 버립니다. = pop 연산을 한다.
    2. 그 다음 제일 위에 있는 카드를 제일 아래로 옮깁니다. = pop 연산과 push 연산을 한다.
  - 큐의 크기가 1이될 때, pop 연산을 하여 정답을 출력한다.

```cpp
#include <iostream>
#include <queue>

int main() {
    int N;
    std::cin >> N;
    std::queue<int> q;

    // 1부터 N까지의 숫자를 큐에 삽입
    for (int i = 1; i <= N; ++i) {
        q.push(i);
    }
    // 카드가 한 장 남을 때까지 반복
    while (q.size() > 1) {
        q.pop(); // 제일 위의 카드 버리기
        q.push(q.front()); // 다음 제일 위의 카드를 제일 아래로 이동
        q.pop(); // 이동한 카드는 다시 제거
    }

    // 마지막에 남은 카드 출력
    std::cout << q.front() << std::endl;

    return 0;
}
```

자료구조 덱 | 01 정의와 개념 소개

- **정의**
  - Deque(Double-Ended Queue)는 양쪽 끝에서 삽입과 삭제가 모두 가능한 자료구조이다.
- **연산**

  모든 연산은 O(1)의 시간복잡도를 가진다.
  - push_front(x) : 앞에 x 추가
  - push_back(x) : 뒤에 x 추가
  - pop_front() : 앞에서 제거하고 출력
  - pop_back() : 뒤에서 제거하고 출력
  - front() : 앞 원소 조회
  - back() : 뒤 원소 조회
  - size() : 덱의 원소 개수 반환
  - empty() : 덱이 비었는지 확인

자료구조 덱 | 01 정의와 개념 소개

- **std::deque<T> name;**
  - C++에서 deque는 <deque> 헤더에 정의되어 있고, STL(Standard Template Library)에서 제공하는 컨테이너이다.
  - 덱은 [] 연산자로 O(1)의 임의 접근이 가능하다.
- **사용법**

```cpp
#include <iostream>
#include <deque>
using namespace std;

int main() {
    deque<int> dq;

    dq.push_back(1);    // [1]
    dq.push_front(2);   // [2, 1]

    cout << "front: " << dq.front() << '\n';  // 2
    cout << "back: " << dq.back() << '\n';    // 1
    cout << "size: " << dq.size() << '\n';    // 2

    dq.pop_back();      // [2]
    dq.pop_front();     // []

    cout << "empty: " << dq.empty() << '\n';  // 1 (true)
}
```

자료구조 덱 | 02 예제 (1) : 백준 10866번 덱

- **문제 요약**

  다음의 8가지 명령(연산)을 지원하는 덱을 구현하라.

  1. push_front X: 정수 X를 덱의 앞에 넣는다.
  2. push_back X: 정수 X를 덱의 뒤에 넣는다.
  3. pop_front: 덱의 가장 앞에 있는 수를 빼고, 그 수를 출력한다. 만약, 덱에 들어있는 정수가 없는 경우에는 -1을 출력한다.
  4. pop_back: 덱의 가장 뒤에 있는 수를 빼고, 그 수를 출력한다. 만약, 덱에 들어있는 정수가 없는 경우에는 -1을 출력한다.
  5. size: 덱에 들어있는 정수의 개수를 출력한다.
  6. empty: 덱이 비어있으면 1을, 아니면 0을 출력한다.
  7. front: 덱의 가장 앞에 있는 정수를 출력한다. 만약 덱에 들어있는 정수가 없는 경우에는 -1을 출력한다.
  8. back: 덱의 가장 뒤에 있는 정수를 출력한다. 만약 덱에 들어있는 정수가 없는 경우에는 -1을 출력한다.

- **입력**
  - 첫째 줄에 주어지는 명령의 수 N ($1 \le N \le 10,000$)이 주어진다. 둘째 줄부터 N개의 줄에는 명령이 하나씩 주어진다.
- **출력**
  - 출력해야하는 명령이 주어질 때마다, 한 줄에 하나씩 출력한다.

자료구조 덱 | 02 예제 (1) : 백준 10866번 덱

- **문제 풀이**
  - STL에 구현된 덱을 사용하면 된다.

```cpp
#include <iostream>
#include <deque>
#include <string>

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int N;
    std::cin >> N;

    std::deque<int> dq;
    std::string command;

    while (N--) {
        std::cin >> command;

        if (command == "push_front") {
            int x;
            std::cin >> x;
            dq.push_front(x);
        } else if (command == "push_back") {
            int x;
            std::cin >> x;
            dq.push_back(x);
        } else if (command == "pop_front") {
            if (dq.empty()) {
                std::cout << -1 << '\n';
            } else {
                std::cout << dq.front() << '\n';
                dq.pop_front();
            }
        } else if (command == "pop_back") {
            if (dq.empty()) {
                std::cout << -1 << '\n';
            } else {
                std::cout << dq.back() << '\n';
                dq.pop_back();
            }
        } else if (command == "size") {
            std::cout << dq.size() << '\n';
        } else if (command == "empty") {
            std::cout << dq.empty() << '\n';
        } else if (command == "front") {
            if (dq.empty()) {
                std::cout << -1 << '\n';
            } else {
                std::cout << dq.front() << '\n';
            }
        } else if (command == "back") {
            if (dq.empty()) {
                std::cout << -1 << '\n';
            } else {
                std::cout << dq.back() << '\n';
            }
        }
    }

    return 0;
}
```

자료구조 덱 | 02 예제 (2) : 백준 1021번 회전하는 큐

- **문제 요약**
  - 지민이는 N개의 원소를 포함하고 있는 양방향 순환 큐를 가지고 있다. 지민이는 이 큐에서 몇 개의 원소를 뽑아내려고 한다.
  - 지민이는 이 큐에서 다음과 같은 3가지 연산을 수행할 수 있다.
    1. 첫 번째 원소를 뽑아낸다. 이 연산을 수행하면, 원래 큐의 원소가 a1, ..., ak이었던 것이 a2, ..., ak와 같이 된다.
    2. 왼쪽으로 한 칸 이동시킨다. 이 연산을 수행하면, a1, ..., ak가 a2, ..., ak, a1이 된다.
    3. 오른쪽으로 한 칸 이동시킨다. 이 연산을 수행하면, a1, ..., ak가 ak, a1, ..., ak-1이 된다.
- **입력**
  - 큐에 처음에 포함되어 있던 수 N이 주어진다.
  - 지민이가 뽑아내려고 하는 원소의 위치가 주어진다. (이 위치는 가장 처음 큐에서의 위치이다.)
- **출력**
  - 이때, 그 원소를 주어진 순서대로 뽑아내는데 드는 2번, 3번 연산의 최솟값을 출력하는 프로그램을 작성하시오.

자료구조 덱 | 02 예제 (2) : 백준 1021번 회전하는 큐

- **문제 풀이**
  - deque에 1 ~ N을 넣는다.
  - 뽑아내려고 하는 수의 위치에 대해 2번 연산과 3번 연산 중 어떤 것이 더 저렴한지 판단하고 해당 연산을 실행한다.
  - 뽑아내려는 수가 deque의 front에 있으면 pop_front 연산을 하면 된다.

```cpp
#include <bits/stdc++.h>

int main() {
    int N, M;
    std::cin >> N >> M;

    std::deque<int> dq;
    for (int i = 1; i <= N; ++i) {
        dq.push_back(i);
    }

    int count = 0;
    while (M--) {
        int target;
        std::cin >> target;

        int idx = std::find(dq.begin(), dq.end(), target) - dq.begin();
        int left = idx;
        int right = dq.size() - idx;

        if (left <= right) {
            while (dq.front() != target) {
                dq.push_back(dq.front());
                dq.pop_front();
                ++count;
            }
        } else {
            while (dq.front() != target) {
                dq.push_front(dq.back());
                dq.pop_back();
                ++count;
            }
        }
        dq.pop_front();
    }

    std::cout << count << std::endl;
    return 0;
}
```

문제

- **아래 문제와 스터디에 다룬 예제들로 STL 사용방법을 익혀보고 나서, 직접 구현하는 방법도 살펴보시면 좋습니다.**
- **기본**
  - 백준 1874번 스택 수열
  - 백준 10773번 제로
- **심화**
  - 백준 5430번 AC
  - 백준 2493번 탑

오늘 스터디는 여기서 마무리하겠습니다.

백준 많이 푸세요! 수고하셨습니다!