PULSE 2023 Fall

5주차 – 그리디

목차

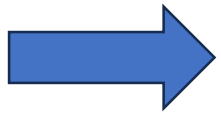
- PS 시 주의할 점
  - ❖자주 하는 실수 - 2
- 그리디 알고리즘

PS시 주의할 점

자주 하는 실수-2

- 배열에서 잘못된 인덱스로 접근하는 실수
  - 인덱스가 -1이 될 수 있음을 주의!

```
const int dx[] = {0, 1, 0, -1};
const int dy[] = {1, 0, -1, 0};
bool vis[55][55];
void dfs(int cx, int cy) {
  vis[cx][cy] = true;
  for (int i = 0; i < 4; ++i) {
    int nx = cx + dx[i];
    int ny = cy + dy[i];
    if (!vis[nx][ny]) dfs(nx, ny);
  }
}
```



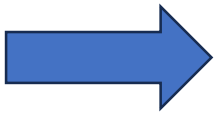
```
const int dx[] = {0, 1, 0, -1};
const int dy[] = {1, 0, -1, 0};
bool vis[55][55];
void dfs(int cx, int cy) {
  vis[cx][cy] = true;
  for (int i = 0; i < 4; ++i) {
    int nx = cx + dx[i];
    int ny = cy + dy[i];
    if (1 <= nx && nx <= n &&
        1 <= ny && ny <= n &&
        !vis[nx][ny]) dfs(nx, ny);
  }
}
```

자주 하는 실수-2

- for 문에서 strlen을 사용하는 실수
  - Strlen의 시간복잡도는 O(n)
  - 사용하고 있는 내장함수의 시간복잡도를 대략적으로 확인할 필요가 있음.

```
char s[5005];
scanf("%s", s);
for (int i = 0; i < strlen(s); ++i) {
  printf("%c", s[i]);
}
```

O(N^2)



```
char s[5005];
scanf("%s", s);
int n = strlen(s);
for (int i = 0; i < n; ++i) {
  printf("%c", s[i]);
}
```

O(N)

자주 하는 실수-2

- 느린 입출력(Input/Output) 사용

- C++의 cin. cout은 일반적으로 scanf, printf 보다 느림.

- 입출력이 많은 문제는 I/O 때문에 시간초과가 날 수 있으므로 알고리즘에 문제가 없다고 생각되면 I/O를 의심!

- 언어 별 빠른 I/O 방법
  - https://www.acmicpc.net/board/view/22716

그리디 알고리즘

그리디 알고리즘

- 탐욕(greedy) 알고리즘
  - 최적화 문제를 해결하는 알고리즘 중 하나
  - 해의 후보 중 최적이라고 생각되는 것을 정답으로 간주하고 진행 및 반복
  - 최종 결론에 도달
- 전체적인 상황을 고려하지 않고 현재 상황에서 가장 좋은 것을 선택함.
- **지역적으로 최적화**
  - 최적해를 구하는 알고리즘이 아님! → 하지만 최적해에 **해당하는 경우가 존재**
  - 어떻게 답이 될 수 있는지 증명 필요!!!

그리디 알고리즘

- 타 알고리즘에 비하여 빠르고 메모리를 적게 사용하여 최적해를 구할 수 있다.
  - 문제 상황이 그리디 알고리즘으로 해결할 수 있는지 확인 필요.
  - 이를 위해서는 반드시 그리디 알고리즘이 최적해를 구할 수 있음을 증명해야 함!

- Greedy 알고리즘 최적해 증명 방법
  1. 그리디 방식이 아니라 다른 방법으로 최적해를 구할 수 있다고 가정한다.
  2. 약간의 변형을 주어 그리디 방식으로 변경한다.
  3. 그리디 알고리즘의 정당성을 증명한다.
     1. Greedy choice property: 그리디 방법을 써도 해당 최적해를 구하는데 문제가 없음을 증명한다.
     2. Optimal substructure: 부분해의 최적해가 전체 답안의 최적해임을 증명한다.

그리디 알고리즘

- 거스름돈 문제(change-making problem)
  - 가장 적은 개수의 동전으로 거스름돈을 주는 문제
  - 그리디 알고리즘으로 항상 최적의 해를 구할 수 있는가?
    - 거스름돈 단위에서 선택한 두 개의 최대공약수는 항상 작은 단위가 되는 경우에는 항상 가능!
  - Ex) 7을 {1, 3, 4, 5}으로 거슬러 줄 때 → Greedy choice property 위배
    - Greedy: 5(1개), 1(2개) 총 3개
    - Optimal: 4(1개), 3(1개) 총 2개

그리디 알고리즘

보물

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	55544	36823	31319	68.481%

문제

옛날 옛적에 수학이 항상 큰 골칫거리였던 나라가 있었다. 이 나라의 국왕 김지민은 다음과 같은 문제를 내고 큰 상금을 걸었다.

길이가 N인 정수 배열 A와 B가 있다. 다음과 같이 함수 S를 정의하자.

$$S = A[0] \times B[0] + \ldots + A[N-1] \times B[N-1]$$

S의 값을 가장 작게 만들기 위해 A의 수를 재배열하자. 단, B에 있는 수는 재배열하면 안 된다.

S의 최솟값을 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 N이 주어진다. 둘째 줄에는 A에 있는 N개의 수가 순서대로 주어지고, 셋째 줄에는 B에 있는 수가 순서대로 주어진다. N은 50보다 작거나 같은 자연수이고, A와 B의 각 원소는 100보다 작거나 같은 음이 아닌 정수이다.

출력

첫째 줄에 S의 최솟값을 출력한다.

그리디 알고리즘

- 보물
  - A의 수를 재배열 하는 것만 가능함.
  - [전략] - 배열 A에서 작은 순서대로의 값과 배열 B에서 큰 순서대로의 값을 곱하는 것이 최선이다.

  - 그리디 알고리즘 정당성 증명
    - Greedy choice property
      - A = {a1, a2, … , an}, B = {b1, b2, … , bn}
      - a1*bn <= ai(i != 1)*bn, (ai-a1)*bn 만큼 작음
      - Ai*bj >= a1*bj, (ai-a1)*bj 만큼 큼
      - Bn이 bj보다 크거나 같으므로 [전략] 방식으로 최적해를 구할 수 있다.
    - Optimal substructure
      - 매 경우 최선의 선택을 한다고 가정할 때, 손해볼 것이 없음이 자명함.

그리디 알고리즘

```cpp
#include <bits/stdc++.h>
#define MAX 50

using namespace std;

int A[MAX];
int B[MAX];

int main(){
    int n, result = 0;
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> A[i];
    }
    for(int i = 0; i < n; i++){
        cin >> B[i];
    }
    sort(A, A + n);
    sort(B, B + n);
    for(int i = 0; i < n; i++){
        result += A[i] * B[n-i-1];
    }
    cout << result;
}
```

```python
n = int(input())

A = list(map(int,input().split()))
B = list(map(int,input().split()))

A.sort(reverse=True)
B.sort()

result = 0

for i in range(n):
  result += A[i] * B[i]

print(result)
```

그리디 알고리즘

- Prim, Kruskal, Dijkstra 알고리즘이 대표적인 그리디 알고리즘의 응용
  - 추후 graph 파트에서 다룰 예정

- Knapsack Problem
  - 배낭에 담을 수 있는 무게의 최댓값이 정해져 있고, 일정 가치와 무게가 있는 짐들을 배낭에 넣을 때, 가치의 합이 최대가 되도록 짐을 고르는 방법을 찾는 문제
  - Fractions(물건을 쪼갤 수 있음): 그리디 알고리즘이 항상 최적임을 보장함.
    - 무게 당 가치가 높은 순서대로 담아낼 수 있음.
  - No-Fractions(물건을 쪼갤 수 없음): 그리디 알고리즘 적용 불가....
    - 가치가 높은 순서나 무게가 적은 순서로만 선택하는 방식으로는 최적해를 구할 수 없음.
    - **Dynamic Programming!!**