5.분할 정복, DP

안해성

배울 내용

- 분할 정복
  풀게 될 문제 : 사각형으로 분할 {1992}, 규칙찾고 분할{2447}
  + 히스토그램{1725}, 반전 수 세기

- DP
  풀게될 문제 : 중복 부분 문제{1699}, 최적 부분 구조{2294},
  차원 줄이기{1309}
  +다차원 배열

분할 정복

Divide and Conquer

분할 정복 예시 1

- 문제 요약 :
  음이 아닌 정수 a, b가 있다고 해보자. ($0 \leq a, b \leq 10^9$)
  a^b 를 쉽게 구할 수 있을까?

  제한 시간 1초

- 풀이 :
  이 문제는 기초 정수론를 배울 때, 예시로 들었던 문제이다.
  그때 O(Log b)에 해결할 수 있다는 것을 배웠다!

a^b 를 작게 만들기

- $a^b = \begin{cases} a^{b/2} \times a^{b/2}\ (b\ is\ even) \\ a^{b/2} \times a^{b/2} \times a\ (b\ is\ odd) \end{cases}$ 라는 것,

  즉 재귀적으로 이 문제를 해결 할 수 있다 것을 알 수 있다.

- 그렇다면, 어째서 위 방법으로 했을 때, 시간 복잡도가 O(log b)로 줄어들게 되었을까?

a^b 관찰하기

- a^b가 어떻게 구해질 수 있는 지 관찰 해보자.

- a^b 에서 b가 짝, 홀수 일 경우에는 아까 본 식과 같이 문제를 나눌 수 있고 $a^{b/2}$를 구했으면 $a^{b/2} \times a^{b/2} \times (a\{b\ is\ odd\})$를 통해 a^b 를 구할 수 있다.

- 추가적으로, b가 0이라면?
  a^b 는 1이라는 것이 자명하다.

결과

- a^b에서 어떤 b이든지 간에 2로 나누다 보면 언젠가는 0이 될 것이다.
  그때, b가 0이면 a^b는 1이라는 기저 조건이 있으니,
  무조건 나눠서 풀 수 있다!

- f(n)을 a^n이라고 하면,

  f(n) = (f(n/2) * (f(n/4) * (f(n/8)* … f(1) * (f(0)=1*a)…)))

- () = f(1), () = f(n/4), () = f(n/2), ()=f(n)

분할 정복

- 분할 정복은 방금 문제와 같이,
  a^b를 $a^{b/2}$로 나눈다.(분할)
  구해진 $a^{b/2}$을 b의 값에 잘 처리한다. (정복)

  을 통해 문제를 해결하는 방식이다.

- 투 포인터처럼,
  어떻게 분할하고 정복하느냐에 따라 다양한 문제를 해결 할 수 있다.

분할 정복 예시 2

- 문제 :
  자연수 n개가 주어진다. $(1 \le n \le 10^9)$ 이를 오름차순으로 정렬 코드를 직접 작성하여라. (즉, std::sort 불가능)

  제한시간 1초

- 풀이 :
  외우고 있을 간단하게 구현할 수 있는 정렬은 대부분 O(n^2)의 시간 복잡도를 가지고 있어서 TLE가 날 것이다.

문제 나눠보기

- 정렬하는데 문제를 어떻게 나눌 수 있을까?
  예시 1처럼 어떤 수학적 공식이 있을까?

  아니면 그냥 예시 1처럼 절반씩 나누고,
  그 작아진 배열을 정렬해볼까?

  나눴다고 해도, 나눠진 배열은 그럼 어떻게 합칠 수 있을까?

문제 나눠보기

- 분할 정복을 이용해서 문제를 풀기 위해선, 방금의 “어떻게 분할하지?”와 “어떻게 분할 한 것을 합치지?”과 같은 질문을 해결 할 수 있어야 한다.

- 병합 정렬.
  정렬할 배열이 있을 때,
  반반 나눠서 정렬하고 합치는 알고리즘.

병합 정렬

- 배열을 절반 씩 나눈다고 치자, 그렇다면 정복은 어떻게 해야 할까?

- 예시 1처럼 정렬이라는 f(lt, rt)함수를 분석해보자
  //f(lt, rt)는 배열의 lt인덱스부터 rt인덱스까지 정렬하는 함수

- f(lt, rt)에서 (lt < rt)일 때,
  f(lt, mid), f(mid+1, rt), 합치기(lt, rt) 를 하면 정렬이 된다.

병합 정렬

- f(lt, rt)에서 (lt == rt)일 때,
  배열의 원소가 1개인 경우로, 이는 이미 정렬되었다고 할 수 있다.
- f(lt, rt)에서 (lt > rt)일 때,
  올바른 배열 범위가 아니므로, 정렬 할 필요도 없다.

- 즉 lt, rt를 계속 절반으로 쪼개서 나누다 보면,
  언젠간 lt == rt에 도달하고, 이는 이미 정렬된 것이니, 어떻게 합치기만
  잘하면 정렬이 완성될 것이다!

합치는 방법

- 합치는 걸 $O(n^2)$에 하기 싫어서 이러고 있으니, 이보단 빨라야 한다.

- 이용할 수 있는 조건,
  우리가 두 배열을 합칠 때, 그 두 배열은 무조건 정렬되어 있다.

- [2, 5, 7, 9]   [1, 3, 4, 8, 10, 11]

코드

- 출처 justiceHui

```cpp
#include <bits/stdc++.h>
using namespace std;

int N, A[1010101];

void Merge(int s, int m, int e){
    static int tmp[1010101];

    int i = s, j = m + 1, idx = s;
    while(i <= m && j <= e){
        if(A[i] < A[j]) tmp[idx++] = A[i++];
        else tmp[idx++] = A[j++];
    }
    while(i <= m) tmp[idx++] = A[i++];
    while(j <= e) tmp[idx++] = A[j++];
    for(int k=s; k<=e; k++) A[k] = tmp[k];
}

void MergeSort(int s, int e){
    if(s == e) return;
    int m = (s + e) / 2;
    MergeSort(s, m);
    MergeSort(m+1, e);
    Merge(s, m, e);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    MergeSort(1, N);
    for(int i=1; i<=N; i++) cout << A[i] << "\n";
}
```

1992.쿼드트리

- 문제를 요약할 자신이 없다. 같이 보고 오자.

- 풀이:
  분할 정복을 이용하면 쉽게 해결 가능하다.

  f(x,y,n)
  //사각형의 왼쪽 위의 점이 x, y이고 길이가 n인 사각형이고,
  // 만약 전부 한 숫자로 차 있다면 그 값을 출력한다.
  = f(x/2, y/2, n/2), f(x/2+n/2, y/2, n/2), f(x/2, y/2 +n/2, n/2),
  f(x/2+n/2, y/2 +n/2, n/2)

1992.쿼드트리

- f(x, y, 1)일 때는 무조건 한 숫자로 채워져 있는 것이기 때문에 그 숫자로 출력하면 된다.

- n는 2의 재곱수이기 때문에 나누다 보면, 무조건 1이 나온다.

  즉 계속 나눠가면서 풀 수 있다.

2447.별찍기 - 10

- 이 역시 설명할 자신이 없다;;

- 풀이,
  문제 자체가 이미 정사각형을 9등분 해서 가운데 빼고 별을 출력하게 하면 된다.
- f(x, y, n)
  //사각형의 왼쪽 위 좌표가 x, y 길이가 n인 사각형 별찍기
  방금 문제처럼 잘 나누고 더해주면 된다.

2447.별찍기 - 10

- n이 1이면 가운데가 뚫릴 수도 없는 사각형도 아니기 때문에, 무조건 별을 출력하면 된다.

- n은 3의 제곱수이기 때문에 나누다 보면 언젠가는 무조건 1이 나온다.

- 즉 분할 정복으로 해결 가능

더 어려운 문제에서는?

- 히스토그램에서 가장 넓은 사각형{1725}
- 반전 수 세기

다이나믹 프로그래밍

DP(Dynamic Programming)

DP 예시 1

- 문제 :
  오늘 처음 봤던 그 문제를 조금 변형해서 ,
  자연수 n 개의 b에 대해 a^b 를 구해보자 ($0 \leq n \leq 10^8$)
  (n개의 b에 대해 a는 한 자연수로 고정)

  제한시간 1초

- 풀이 :
  분할 정복을 이용해서 풀어보려고 해도,
  시간 복잡도는 O(log b) * n가 되어서 TLE.

a^b 관찰해보기

- $a^{14} = a^7 \times a^7$이고 $a^{15} = a^7 \times a^7 \times a$이다.
- $a^7 = a^3 \times a^3 \times a$ 이고 $a^6 = a^3 \times a^3$ 이다.

- 위처럼 a^b를 구하다 보면, 이미 한 번 계산을 했지만 또 다시 계산을 하게 되는 경우가 있다.

  이 모든 a^b 의 결과 값을 저장해준다면, 훨씬 계산을 적게 할 수 있지 않을까?

DP 예시 2

- 14916.거스름돈 문제를 봐 보자.

- 문제 요약 :
  2, 5를 최대한 적게 사용해서 자연수 n을 만들면 된다. 그 때의 사용한 2, 5의 개수를 구하라. ($1 \le n \le 10^5$)
  제한시간 (1초)

- 풀이 :
  그리디하게 풀 수 있을면 O(1)에 해결가능할 것 같다.

14961.거스름돈

- 일단 그리디는 불가능하다.
  그리디로 푼다고 하면, 가장 큰 5를 최대한 쓰고 2를 쓰는 방식인데,
  6이라는 반례가 존재한다.
  n이 6일 때는 5를 쓰면 안되기 때문이다.

- f(n)을 n이 n일 때의 2, 5를 가장 적게 쓴 개수라고 정의해 보자.

- 그렇다면 f(n) = min(f(n-2), f(n-5))+1이 아닐까?

DP

- DP는 2 종류의 문제를 잘 풀 수 있는 알고리즘이다.
  "중복 부분 문제",
  "최적 부분 구조" 라는 문제를 풀 수 있고,
  각각 예시 1번 문제와, 예시 2번 문제이다.

- DP가 무엇이길래 저 2문제를 잘 풀 수 있는 걸까?

DP

- DP는 문제를 나누고, 그 나눠진 작은 문제를 해결한 결과 값을 저장해서 큰 문제의 풀이에 이용하는 방법을 말한다 할 수 있다.

- 예시 1에서는 a^b를 $a^{b/2}$로 나누고, 이 결과 값을 저장하고 a^b를 푸는데 이용한다.

- 예시2에서는 f(n)을 f(n-2), f(n-5)로 나누고 저장한다음, f(n)을 구하는데 이용한다.

예시 2 증명

- 예시 1은 바로 보이니 패스하고 예시 2를 봐 보자.

- 어떤 n을 만든다고 할 때, 동전을 최대한 적게 추가해서 만드는 것이 이득임은 당연한다.

- 따라서 동전 1개를 추가한다고 할 때,
  n을 만들 수 있는 경우는 n-2, n-5 밖에 없다.

  즉 f(n-2), f(n-5)중 작은 값을 이용해서 f(n)을 만들면,
  f(n)을 이보다 더 작게 만들 수는 없다.

예시 2 구현 팁

- 예시 2번을 구현한다고 한다면, 어떻게 하는 게 좋을까?

  직접 함수 작성해서 f(n)을 만들면 될까?

- 더 편하게 하는 방식이 있다!

배열을 함수처럼

- int arr[n]이 있다고 하자.
  이를 단순히 1차원 배열이라고 생각하지 말고,

  n에 만들 동전 값을 넣어주면 그에 해당하는 결과값이 나오는
  함수라고 생각하자.

- 그렇다면 f(n) = min(f(n-2),f(n-5))는
  arr[n] = min(arr[n-2], arr[n-5]) 처럼 쓸 수 있을 것이다.

차원이 늘어나면?

- 문제가 마지막으로 사용한 동전이 2아님 5인지 구별하고 그 경우의 수를 구해야 한다고 해보자.

- 그렇다면 int arr[n][2]같이 표현할 수 있다.
- arr[n][0] =
  마지막으로 사용한 동전이 2일 때 값어치를 n을 만드는 경우의 수.

- arr[n][1] =
  arr[n][1]과 같지만 마지막으로 사용한 동전이 5인 경우의 수.

최적 부분 구조, 중복 부분 문제

- 이처럼 큰 문제를 작은 문제로 나누고,

  "그 작은 문제의 최적해가 큰 문제의 최적해에 포함된다면"

  그 문제를 최적 부분 구조라고 한다.

- 중복 부분 문제는
  같은 계산을 여러 번 해야 하는 문제를 의미한다.

DP 문제 풀기

- 예시 2처럼 함수 f(n)을 정의하고, 어떻게 해야 f(n)을 만들 수 있는 지를 중심으로 생각하면 조금은 더 수월하게 풀 수 있다.

- 구현은 방금 봤던 배열을 함수처럼 이용하면 쉽게 할 수 있다.

  또 배열에서 차원이 몇 개든 상관이 없던 것 처럼, f(n)의 매개 변수도 마구 늘어도 된다.
  (단지 시간 복잡도가 늘어날 뿐..)

1699.제곱수의 합

- f(n) : n을 만드는데 썼던 최소 항의 개수
- f(n) = min(f(n-a)) + 1 {a = n보다 작은 제곱수}

```cpp
#include <iostream>

using namespace std;

int dp[100100];

int main() {
    int n;

    cin >> n;

    dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        int res = 1e9;
        for (int k = 1; k * k <= i; k++) {
            res = min(res, dp[i - k * k]);
        }
        dp[i] = res + 1;
    }

    cout << dp[n];

    return 0;
}
```

1309.동물원

- f(n, pos) :
  세로로 n칸째의 우리에 pos{0 : 배치 X ,1 한쪽에 배치, 2 : 다른 한쪽}
  에 배치했을 때의 경우의 수

- f(n, 0) = f(n-1, 0) + f(n-1, 1) + f(n-1,2)
- f(n, 1) = f(n-1, 0) + f(n-1, 2)
- f(n, 2) = f(n-1, 0) + f(n-1, 1)

1309.동물원

+ 식을 정리하면, int arr[n][3]이 아니라
  int arr[n][2]를 통해 해결할 수 있다.

```cpp
#include <iostream>

using namespace std;

#define MOD 9901

int dp[100100][2];

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

    dp[0][0] = 1;
    dp[1][0] = 1;
    dp[1][1] = 1;
    for (int i = 2; i <= n; i++) {
        dp[i][0] = ((dp[i - 1][0] + dp[i - 1][1])%MOD + (dp[i - 2][0] + dp[i - 2][1])%MOD)%MOD;
        dp[i][1] = ((dp[i - 1][0] + dp[i - 2][0])%MOD + dp[i - 2][1])%MOD;
    }

    cout << ((dp[n][0] + dp[n][1])%MOD + (dp[n - 1][0] + dp[n - 1][1])%MOD)%MOD;

    return 0;
}
```

더 어려운 문제는?

- 12865.평범한 배낭 (유명한 문제인 냅색(knapsack) 문제)
- 2098.외판원 순회 (비트 마스킹 DP)
- 2618.경찰차 (차원 줄이기)

정리

- 분할 정복
  문제 여러가지 방법으로 나누어서 풀기

- DP
  "중복되는 부분 문제",
  "최적 부분 구조"인 문제를
  작은 문제들의 값을 저장한 것을 이용해 구할 수 있다.