Bruteforce

시간복잡도, 공간복잡도 계산과 브루트포스

Dosawas 2024-02-18

목차

차례

1주차 review

- 첫시간인데 너무 욕심부렸습니다. 죄송합니다.
  - 시간복잡도 계산이 필요한 문제들이 좀 많았음…, 그리고 그리디 좀 어렵죠?..ㅠㅠ
  - 제가 풀 때 생각을 좀 해야했던 문제들을 담았습니다. (낙담 하지 말것!!)
- fast_io 할 때, endl 대신 “\n” 쓰기
- 28323 스케이트 연습 - 뒤집어서 생각할 수 있는가?
- 1105 팔 - 그리디한 아이디로 모든 경우를 다 세지 않고 문제를 풀 수 있는가?
- 1439 뒤집기 - 배열을 스위핑하면서 처리할 수 있는가?

시간복잡도

컴퓨터의 장점

- 매우 쉬운 문제 : 3, 100, 6, 8, 12, 16 중에서 12가 있습니까?
- 사람 : 네 (1초안에) , 컴퓨터 : 네(1초안에)
- 그렇다면 다음과 같은 문제는 어떨까요?

시간복잡도

컴퓨터의 장점

- 6807 5249 73 3658 8930 1272 7544 878 7923 7709 4440 8165 4492 3042 7987 2503 2327 1729 8840 2612 4303 3169 7709 7157 9560 933 3099 278 1816 5335 9097 7826 3512 9267 3810 7633 979 9149 6579 8821 1967 672 1393 9336 5485 1745 5228 4091 194 6357 5001 1153 6708 7944 5668 1490 8124 2196 9530 903 7722 4666 8549 8024 7801 6853 977 7408 8228 4933 298 8981 8635 8013 3865 9814 9063 4536 9425 1669 4115 94 5629 6501 6517 4195 105 404 9451 4298 2188 1123 9505 6882 6752 1566 6716 337 4438 3144 12 6807 5249 73 3658 8930 1272 7544 878 7923 7709 4440 8165 4492 3042 7987 2503 2327 1729 8840 2612 4303 3169 7709 7157 9560 933 3099 278 1816 5335 9097 7826 3512 9267 3810 7633 979 9149 6579 8821 1967 672 1393 9336 5485 1745 5228 4091 194 6357 5001 1153 9063 4536 9425 1669 4115 94 5629 6501 6517 4195 105 404 9451 4298 2188 1123 9505 6882 6752 1566 6716 337 4438 314 중에 12가 있을까요?
- 인간 : …..
- 컴퓨터 : 네(1초 안에)

시간복잡도

컴퓨터가 얼마나 더 빠른가 / 빅O notation

- 컴퓨터는 1초에 1억번 계산한다고 생각합시다.
  - 요즘 컴퓨터 좋아져서 더 빠르긴합니다만…열악한 환경으로 계산하는게 맘 편합니다.
- 입출력은 오래 걸립니다. 그래서 fast_io를 사용하는 것입니다.
- 최악의 상황으로 계산한다.. 그래서 빅 O 표기법을 사용합니다.
  - 자신의 코드에 나오는 반복문, 재귀를 살펴보며 계산합니다.
  - for(int I=0;i<N;i++) for(int j = 0;j<N;j++) 이라고 하면 $O(N^2)$ 이 되겠죠.
- 재귀, copy, max_element(), str.find(), 배열복사 등 시간복잡도 알고 쓰기

공간복잡도

메모리 초과

- PS를 할 때 요긴하게 사용하는 것 중 하나는 전역 변수입니다.
- Main 함수에 선언을 하면 stack memory 초과가 되기 쉽습니다.
- 그래서 vector 등 동적 배열 컨테이너를 사용하는 것이 좋은 것입니다.

```
int main(){
    fast_io
    const int max = 1e7;
    int arr1[max];
}
```

Segmentation fault를 받음

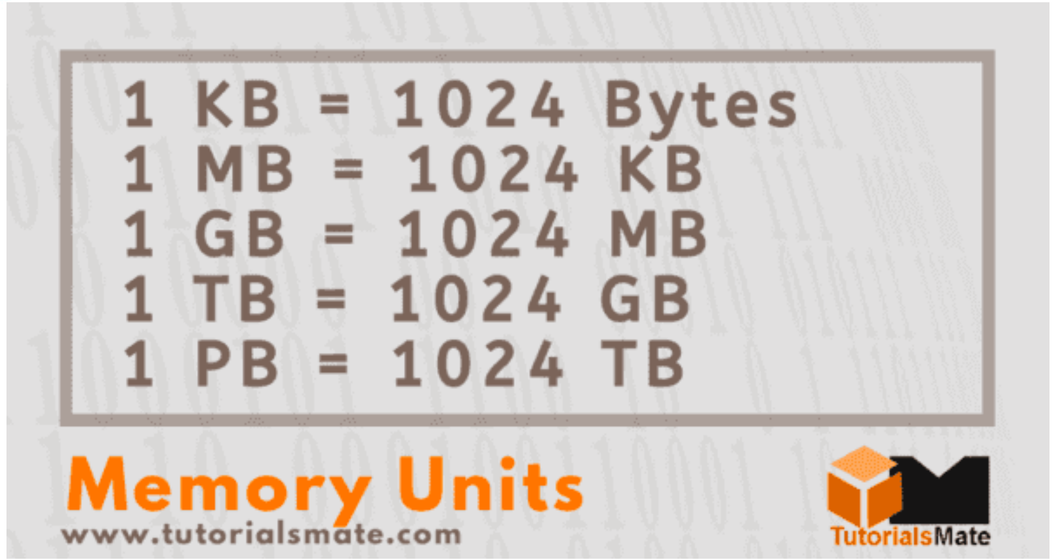
```
int main(){
    fast_io
    const int max = 1e7;
    // int arr1[max];
    vector<int> arr2(max);
}
```

정상적으로 잘 작동함

공간복잡도

메모리 대략적인 계산

- int는 4byte입니다.
- 1byte = 8bit
- 만약 메모리 제한이 128MB라면 128 * 1024 * 1024 byte를 사용할 수 있구나.
- 그러면 int형은 약 백만개 넘게 쓸 수 있겠네~
- 하지만 천만개 넘게는 저장하기 힘들겠다…



구현

2. 유용한 도구 1 - 여러가지 유용한 함수들

- min, max
- max_element(시작주소, 끝주소)
- find(), count(), fill(), memset(0, 1, -1로만 초기화 가능)
- gcd(greatest common divisor), lcm(least common multiple)
- 너무 많아서 다 못씀 검색해서 쓰세요, 직접 구현도 해보면 좋음(특히, gcd, lcm)

구현

2. 유용한 도구 1 - 여러가지 유용한 함수들

- #include <algorithm>
  - sort(시작주소, 끝주소, 비교함수)
  - reverse(시작주소, 끝주소), swap(a , b)
- #include <string> - 문자열 문제에 정말 유용함
  - stoi, stoll - string to int, string to longlong
  - 문자열 덧셈, str.substr(시작주소, 크기)
  - Str.find()
- 이런거 말고도 매우 많지만 다 담을 수 없고 종종 코드를 넣으면서 보여드리겠습니다.

구현

2. 유용한 도구 2 - auto와 foreach문(C++14이상 쓰세요!)

- C계열 언어의 두려운점은 자료형이 존재한다는 것이다. (사실 그렇게 어렵지도 않지만)

- 참조 연산자 &를 잘 이용해보자!

```cpp
int main(){
    fast_io
    string s; cin >> s;
    int n; cin >> n;
    vector<pii> p(n);
    for(auto &[x, y] : p) cin >> x >> y;
    for(auto c : s){
        for(auto [x, y] : p){
            cout << x << c << y << ' ';
        }
        cout << '\n';
    }
}
```

```
+-*/
3
1 4
5 6
7 8
1+4 5+6 7+8
1-4 5-6 7-8
1*4 5*6 7*8
1/4 5/6 7/8
```

구현

2. 유용한 도구 3 - vector(가변 배열)

```cpp
int main(){
    fast_io
    int n; cin >> n;
    // 1차원 배열 초기화
    vector<int> a = {1, 2, 3, 4, 5};
    vector<int> b(5, 1);
    vector<int> c(n);
    // 2차원 배열 초기화
    int h, w; cin >> h >> w;
    vector<vector<int>> ab = {{1, 2, 3}, {4, 5, 6}};
    vector<vector<int>> cd(h, vector<int>(w, 1));
    vector ef = vector(h, vector<int>(w));
    // 4차원 배열
    int a1, a2, a3, a4; cin >> a1 >> a2 >> a3 >> a4;
    vector<vector<vector<vector<int>>>> not_good(a1, vector<vector<vector<int>>>(a
    vector good = vector(a1, vector(a2, vector(a3, vector<int> (a4, 1)))));

    cout << ":finish:\n";
}
```

```cpp
// 가변배열
a.push_back(1); // 맨 뒤에 1을 추가
a.pop_back(); // 맨 뒤 원소 지우기
a.size(); // 크기 리턴
// 원소나 주소 접근
a.front(); // 맨 앞 원소
a.back(); //맨 뒤 원소
a.begin(); // 맨앞원소 주소
a.end(); // 맨 뒤에서 하나 더 간 주소
// 각종 함수들
a.clear(); // 모든 원소 없애버리기
a.empty(); // 비어있으면 1
a.erase(a.begin()); // 주소에 있는 거 지우기
```

BruteForce

무식하게 대입하기

- 암호학에는 brute-force attack (무차별 대입 공격)이라는 말도 있습니다.
  - 무식하게 때려박다보면 답이 나온다는 것이지요. (이것이 컴퓨터의 큰 강점입니다)
- 만약에 컴퓨터가 4자리 숫자로만 이루어진 암호를 맞추려면 $10 \times 10 \times 10 \times 10$
  - 따라서 0.5초도 안걸릴겁니다.
- 하지만 인간이 하나하나 해보려면….하나당 1초라고 쳐도 2시간이 넘게 걸리겠네요
- 여기서 숫자 한자리만 알아내도 10배 더 빨리 구할 수 있는 것도 ps에서 중요한 컨셉이죠.

BruteForce

문제 풀이

- Naive한 풀이 생각 -> 시간 복잡도 계산 -> 어? 그냥 돌리면 되겠는데?

```
1 2 3
```

```
vector<int> ARR = {100, 200, 300, 400, 500};
vector<int> choose = {0, 0, 0, 1, 1};
do{
    for (int i=0;i<ARR.size();i++) if(!choose[i]) cout <
    cout << '\n';
} while (next_permutation(choose.begin(), choose.end()))
```

```
100 200 300
100 200 400
100 200 500 ';
100 300 400
100 300 500
100 400 500
200 300 400
200 300 500
200 400 500
300 400 500
```

BruteForce

순열과 조합

```cpp
int main(){
    fast_io
    vector<int> arr = {1, 2, 3};
    arr.push_back(4);
    do{
        for(int a : arr) cout << a << ' ';
        cout << '\n';
    }while(next_permutation(arr.begin(), arr.end()));
    cout << '\n';

    vector<int> ARR = {100, 200, 300, 400, 500};
    vector<int> choose = {0, 0, 0, 1, 1};
    do{
        for (int i=0;i<ARR.size();i++) if(!choose[i]) cout << ARR[i] << ' ';
        cout << '\n';
    } while (next_permutation(choose.begin(), choose.end()));
}
```

BruteForce

순열과 조합 이용 문제

- 1145 적어도 대부분의 배수(https://www.acmicpc.net/problem/1145)
- 5개 중에 3개를 고르는 문제입니다.
- 아까 배운 조합을 써도 되고 3중 for문을 써도 됩니다.

문제

다섯 개의 자연수가 있다. 이 수의 적어도 대부분의 배수는 위의 수 중 적어도 세 개로 나누어 지는 가장 작은 자연수이다.

서로 다른 다섯 개의 자연수가 주어질 때, 적어도 대부분의 배수를 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 다섯 개의 자연수가 주어진다. 100보다 작거나 같은 자연수이고, 서로 다른 수이다.

BruteForce

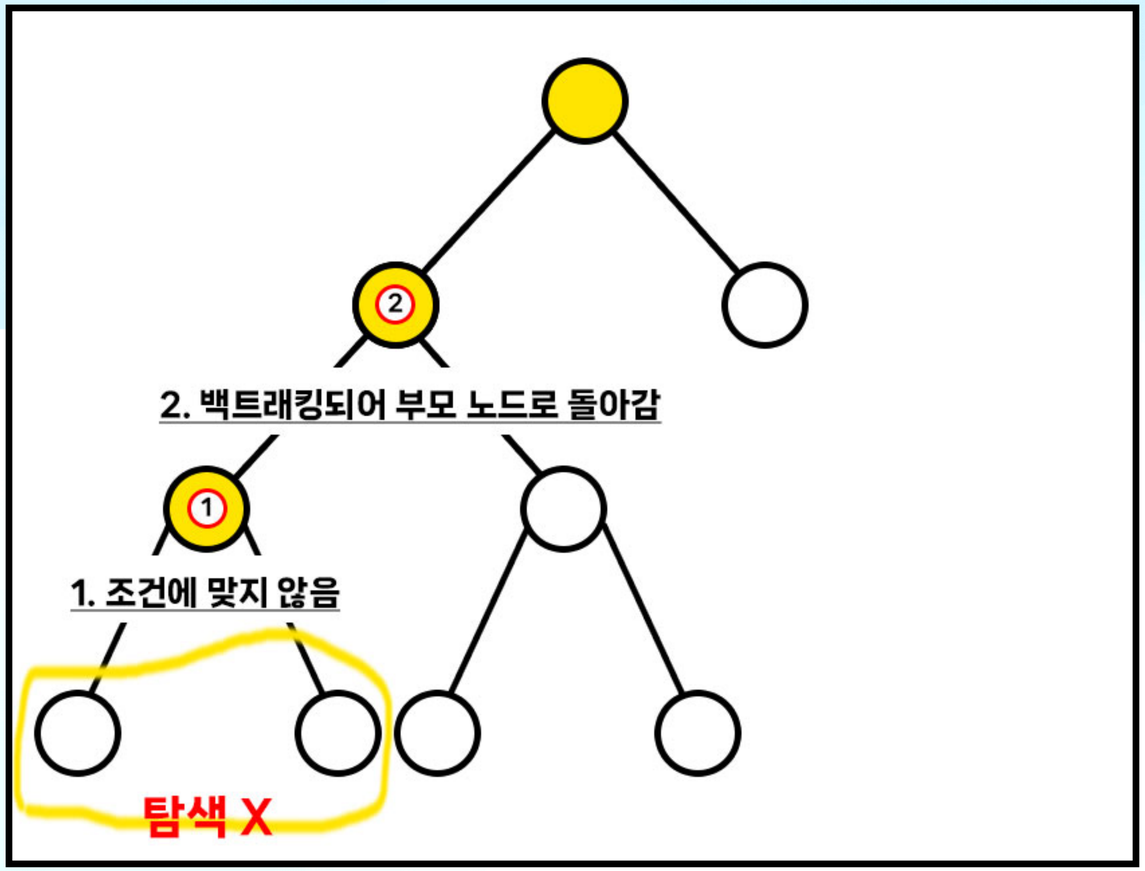
순열과 조합 이용 문제

```cpp
int main()
{
    int res = 1000000 + 1;
    int a[5];
    for (int i = 0; i < 5; i++) cin >> a[i];
    vector<int> c = {0, 0, 1, 1, 1};
    do{
        int x = 1;
        for(int i=0;i<5;i++){
            if(c[i]) x = lcm(x, a[i]);
        }
        res = min(res, x);
    } while (next_permutation(c.begin(), c.end()));
    cout << res;
}
```

백트래킹(Back-Tracking)

1. 원리

- 그림으로 알아보자



출처 : https://velog.io/@rekoding/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-%EB%B0%B1%ED%8A%B8%EB%9E%98%ED%82%B9-BackTracking

백트래킹(Back-Tracking)

2. 문제풀이

- 매우 유명한 백트래킹 문제가 있습니다. (N과 M 대부분 다 풀어볼 것을 권장합니다)

문제

자연수 N과 M이 주어졌을 때, 아래 조건을 만족하는 길이가 M인 수열을 모두 구하는 프로그램을 작성하시오.

- 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 수열

입력

첫째 줄에 자연수 N과 M이 주어진다. (1 ≤ M ≤ N ≤ 8)

백트래킹(Back-Tracking)

2. 문제풀이

- used[i] : i라는 숫자를 지금까지 길 따라오면서 썼는가?
- ans : 결과 저장 배열
- 전역변수를 쓴다.
  - 안되는 코드 ->

```cpp
void back_tracking2(vector<int> arr){
    if(arr.size() == M){
        for(int r : arr) cout << r << ' ';
        cout << '\n';
        return ;
    }

    for(int i=1;i<=N;i++){
        if(!used[i]){
            used[i] = true;
            arr.push_back(i);
            back_tracking2(arr);
            used[i] = false;
        }
    }
}
```

오래걸리는 코드

```cpp
int N, M;
int used[9], ans[9];

void back_tracking(int cnt){
    if(cnt == M){
        for(int i=0;i<M;i++){
            cout << ans[i] << ' ';
        }
        cout << '\n';
        return ;
    }

    for(int i=1;i<=N;i++){
        if(!used[i]){
            used[i] = true;
            ans[cnt] = i;
            back_tracking(cnt+1);
            used[i] = false;
        }
    }
}

int main(){
    fast_io
    cin >> N >> M;
    back_tracking(0);
}
```

정답 코드

백트래킹(Back-Tracking)

2. 문제풀이2

- [18429 근손실](https://www.acmicpc.net/prob
- 백트래킹에 조건이 추가 되었습니다.
  - 단순히 이 문제처럼 매개변수로 처리할 수 있다면
  - 그렇게 하는게 좋습니다.

```cpp
int n, k;
int a[9], used[9];
int res = 0;

void back_tracking(int idx, int sum){
    if(idx == n){
        res++;
        return ;
    }

    for(int i=0;i<n;i++){
        if(!used[i] && sum + a[i] - k >= 500){
            used[i] = true;
            back_tracking(idx+1, sum + a[i] - k);
            used[i] = false;
        }
    }
}

int main(){
    fast_io
    cin >> n >> k;
    for(int i=0;i<n;i++) cin >> a[i];
    back_tracking(0, 500);
    cout << res;
}
```

마무리

- 시간 복잡도와 공간 복잡도를 생각하며 앞으로의 구현 방향을 설계한다.
- Naive한 방법을 생각해내는 것이 문제 풀이의 시작이다.
- 컴퓨터는 무식하게 대입하면서 푸는 것을 참 잘한다.
- C++11, C++17, C++20에서 지원하는 여러 기능들을 적극 이용해보자.
  - 각종 라이브러리에 있는 함수들
  - auto
  - foreach문