1

프로그래밍 문제 접근하기

1.1 시그마 BOJ #2355

두 정수 A와 B가 주어지면 두 정수 사이에 있는 모든 수의 합을 구해야 합니다.

간단히 생각한다면 그냥 A부터 B까지의 모든 수를 `for` 루프를 돌려 더할 수 있을 것 같습니다. 하지만 문제에서 A, B의 제한이 -21.4억 $\leq A, B \leq 21.4$억 입니다. 최악의 경우 42억번의 연산을 해야 되고, 시간 제한이 0.25초이므로 이렇게 풀면 당연하게도 TLE를 받게 됩니다.

다행히도 우리는 고등학교 때 이런 식을 배운 적이 있습니다.

$$1+2+3+\cdots+(n-1)+n=\sum_{k=1}^{n}k=\frac{k(k+1)}{2}$$

이를 응용해 문제의 식을 세워볼 수 있습니다. 만일 $A \leq B$라면

$$A+(A+1)+\cdots+(B-1)+B=\underbrace{\frac{B(B+1)}{2}-\frac{A(A+1)}{2}}_{\substack{B\text{까지의 합}-A\text{까지의 합}\\=(A+1)\text{부터 }B\text{까지의 합}}}+A$$

입니다. 이를 그대로 코드로 구현하면 시간 복잡도는 $\mathcal{O}(1)$로 어떤 수가 들어와도 풀 수 있습니다.

주의할 점이 있는데, 하나는 문제에 $A \leq B$라고 명시되어 있지 않다는 점입니다. $A > B$라면 두 정수를 `swap`해 주거나, `if`문을 통해 따로 계산해 주는 등의 처리가 필요합니다. 제 경우에는 처음 입력받은 두 수를 u와 v라 두고 $A=\min(u,v)$, $B=\max(u,v)$로 계산했습니다. 이를 처리하는 방법은 여러 가지가 있으니 마음에 드는 방법으로 풀면 됩니다.

또한 Python 사용자가 아니라면 주의해야 할 것은 A와 B를 `int` 형으로 두면 안 된다는 점입니다. A와 B, 그리고 계산 결과 모두 `int` 자료형에 들어가지만 $\frac{B(B+1)}{2}$와 같은 중간 계산값은 `int`에 들어갈 수 없을지도 모르기 때문입니다. 다행히도 `long long` 등의 64비트 정수형을 이용해 해결할 수 있습니다.

정답 코드

실행 시간 0ms, 메모리 1,988KB

```cpp
#include <iostream>
#include <algorithm> // min(), max()

using namespace std;
using ll = long long;

int main() {
    ll u, v;
    cin >> u >> v;

    ll a = min(u, v), b = max(u, v);
    cout << (b * (b + 1) / 2) - (a * (a + 1) / 2) + a;
    return 0;
}
```

1.2 달팽이는 올라가고 싶다 BOJ #2869

달팽이가 Vm의 나무를 올라갑니다. 하루 Am 올라가지만 자는 동안 Bm 내려갑니다. 정상에 올라간 후에는 미끄러지지 않는다고 합니다.

이 문제도 상수 제한이 너무 커서 직접 시뮬레이션해 볼 수는 없을 것 같습니다. 앞서 했던 것처럼 수학을 이용해 풀어봅시다.

달팽이는 하루에 $(A-B)$m 올라갑니다. 마지막 날에는 1m에서 Am 사이만큼 올라갈 것입니다. 따라서 마지막 날을 제외하고 생각한다면 $(V-A)$m의 막대를 하루에 Dm씩 올라갔을 때 며칠이 걸리는지를 계산하는 문제로 바뀝니다. 이는

$$\left\lceil \frac{V-A}{A-B} \right\rceil + 1 \quad (\text{일})$$

입니다. 이제 $V-A$와 $A-B$ 각각을 실수로 바꾼 뒤 `ceil` 연산을 하면 되겠습니다.

하지만 프로그래밍 문제에서 정수가 등장하면 최대한 정수로만 계산해 주는 게 좋습니다. **실수 오차**가 생길 수 있기 때문입니다. 이런 계산도 정수 나눗셈으로 하는 게 좋은데, 안타깝게도 정수 나눗셈은 나머지를 무조건 버려버립니다.

다행히도 p와 q가 모두 자연수라면 아래 식이 성립합니다.

$$\left\lceil \frac{p}{q} \right\rceil = \left\lfloor \frac{p+q-1}{q} \right\rfloor$$

이를 이용해 위의 식을 이렇게 바꿀 수 있습니다.

$$\left\lceil \frac{V-A}{A-B} \right\rceil + 1 = \left\lfloor \frac{V-A+A-B-1}{A-B} \right\rfloor + 1$$
$$= \left\lfloor \frac{V-B-1}{A-B} \right\rfloor + 1$$

$$\lceil x \rceil = \begin{cases} \lfloor x \rfloor & x \in \mathbb{Z} \\ \lfloor x+1 \rfloor & \text{otherwise} \end{cases}$$

물론 생각하기 어렵다면 나누어 떨어질 때를 `if` 문으로 거르는 방법을 이용할 수도 있습니다. 여하튼 이 식을 코드로 구현하면 $\mathcal{O}(1)$으로 AC를 받습니다.

정답 코드

실행 시간 0ms, 메모리 1,988KB

```cpp
#include <iostream>

using namespace std;

int main() {
    int a, b, v;
    cin >> a >> b >> v;
    cout << (v - b - 1) / (a - b) + 1;
    return 0;
}
```

1.3 블랙잭 BOJ #2798

수가 적힌 카드 여러 장을 보고, 합이 M을 넘지 않으면서 M에 최대한 가까운 카드 3장의 합을 출력하는 문제입니다. 다행스럽게도 카드의 개수는 100개를 넘지 않습니다. 따라서 이 문제는 가능한 카드 3개의 조합을 모두 시도해 보는 $\mathcal{O}(n^3)$ 알고리즘으로도 쉽게 풀립니다.

정답 코드

실행 시간 0ms, 메모리 1,988KB

```cpp
#include <iostream>
#include <algorithm> // max()

using namespace std;

int arr[100];

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
```

```
14
15      int mx = 0;
16      for (int i = 0; i < n; i++) {
17          for (int j = 0; j < n; j++) {
18              if (i == j) continue;
19              for (int k = 0; k < n; k++) {
20                  if (i == k) continue;
21                  if (j == k) continue;
22                  int s = arr[i] + arr[j] + arr[k];
23                  if (s > m) continue;
24                  mx = max(mx, s);
25              }
26          }
27      }
28
29      cout << mx;
30      return 0;
31  }
```

여담으로, 위 코드의 16–27번째 줄을 `continue` 없이 작성하면 아래와 같이 됩니다.

```
16      for (int i = 0; i < n; i++) {
17          for (int j = 0; j < n; j++) {
18              if (i ≠ j) {
19                  for (int k = 0; k < n; k++) {
20                      if (i ≠ k) {
21                          if (j ≠ k) {
22                              int s = arr[i] + arr[j] + arr[k];
23                              if (s ⩽ m) {
24                                  mx = max(mx, s);
25                              }
26                          }
27                      }
28                  }
29              }
30          }
31      }
```

```php
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

마치 왼쪽에서 류リュウ가 아도겐波動拳을 날린 것 같아 보이는 아름다운 코드입니다. 외국에선 실제로 이런 코드를 **들여쓰기 아도겐**Indent Hadouken 이라는 이름으로 부를 정도입니다.

이런 코드는 가독성이 상당히 떨어지고, 에디터 창을 왼쪽 오른쪽 왔다갔다 해야 하기 때문에 짜다가도 내가 뭘 짜고 있는지 잊어버리기 쉬우며 디버깅하기도 어렵습니다. 이런 이유로 `for` 루프 내부에서 조건을 처리할 때는 `if` 블록으로 감싸는 대신 `continue`를 적극적으로 활용하는 게 좋습니다.

1.4 설탕 배달 BOJ #2839

알고리즘을 잘 하려면 설탕도 배달할 줄 알아야 합니다. 3kg 봉지와 5kg 봉지가 있는데, 최대한 적은 봉지를 이용해 정확히 Nkg의 설탕을 배달해야 합니다.

이런 문제는 모든 경우의 수를 시도해 보고 싶어도 애매합니다. 하지만 5kg 봉지를 최대한 많이 가져가면 봉지의 수를 아낄 수 있음을 짐작할 수 있습니다. 일단 N이 5로 나눠진다면 전부 5kg 봉지를 들고 가면 됩니다.

그러면 N을 5로 나눴을 때의 나머지에 따라 가져가는 설탕 봉지의 개수가 달라질 것 같습니다. 1이 남을 때, 2가 남을 때, 등등을 전부 생각해 보면 아래와 같이 정리됩니다. $N = 5 \times k + r$이라면

r	식	총 봉지 수
0	$N = 3 \times 0 + 5 \times k$	k
1	$N = 3 \times 2 + 5 \times (k-1)$	$2 + (k-1)$
2	$N = 3 \times 4 + 5 \times (k-2)$	$4 + (k-2)$
3	$N = 3 \times 1 + 5 \times k$	$1 + k$
4	$N = 3 \times 3 + 5 \times (k-1)$	$3 + (k-1)$

이 됩니다. 하지만 k가 음수가 될 수는 없으므로 예외적으로 4kg과 7kg은 어떤 조합으로도 만들 수 없음을 알 수 있습니다. 따라서 N이 4 혹은 7일 때만 예외 처리를 해 주고, 나머지는 n을 5로 나눈 나머지에 따라 `switch` 문 등을 이용해 코드로 구현하면 정답입니다.

정답 코드

실행 시간 0ms, 메모리 1,988KB

```cpp
#include <iostream>

using namespace std;

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

    if (n == 4 || n == 7) {
        cout << -1;
        return 0;
    }

    int r = n % 5, k = n / 5;
    switch (r) {
        case 1:
        case 3:
            cout << 1 + k;
            break;
        case 2:
        case 4:
            cout << 2 + k;
            break;
        default:
            cout << k;
            break;
    }

    return 0;
}
```

16–17번째 줄에서 case 문이 이런 식으로 겹쳐져 있으면 $r = 1$이거나 $r = 3$일 경우 18번째 줄이 실행됩니다. 20–21번째 줄의 경우도 같습니다.

1.5 그룹 단어 체커 BOJ #1316

어떤 단어가 그룹 단어인지 체크해야 합니다. 알파벳이 떨어져서 나타나면 안 된다고 합니다. 다시 말하면, 앞에서 등장한 알파벳이 뒤에서 다시 등장하면 안 된다는 뜻입니다.

알파벳이 앞에서 등장했는지 여부를 알파벳 개수만큼의 배열을 만들어 판단할 수 있습니다. 인접한 알파벳들의 경우 for 루프를 돌리면서 지금 확인하고 있는 알파벳이 직전에 확인한 알파벳과 같다면 무시하고 지나가 버리면 됩니다.

정답 코드

실행 시간 0ms, 메모리 1,988KB

```cpp
#include <iostream>
#include <string>
#include <cstring> // memset()

using namespace std;

bool occur['z' + 1];

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

    int s = 0;
    while (n--) {
        memset(occur, 0, sizeof(occur)); // initialize array

        string str;
        cin >> str;

        bool flag = true;
        for (int i = 0; i < str.length(); i++) {
            if (i != 0 && str[i] == str[i - 1]) continue;
            if (occur[str[i]]) {
                flag = false;
                break;
            } else {
                occur[str[i]] = true;
            }
        }
        if (flag) s++;
    }

    cout << s;
    return 0;
}
```

`cstring` 헤더의 `memset`은 연속된 메모리를 빠르게 초기화해 줍니다.

1.6 체스판 다시 칠하기 BOJ #1018

N과 M은 모두 50 이하입니다. $N = M = 50$이라면 그 중 8×8 체스판을 고르는 경우의 수는 $(50 - 8 + 1)^2 = 1849$개입니다. 따라서 이 문제도 상수가 크지 않기 때문에 가능한 경우를 전부 시도해 볼 수 있습니다.

정답 코드

실행 시간 0ms, 메모리 1,988KB

```cpp
#include <iostream>
#include <string>
#include <algorithm> // min()

using namespace std;

string b[50]; // board

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        cin >> b[i];
    }

    int mn = 987654321;
    for (int i = 0; i <= n - 8; i++) {
        for (int j = 0; j <= m - 8; j++) {
            // (i, j): upper left corner
            for (int k = 0; k < 2; k++) {
                // k = 0: WBWB ... , k = 1: BWBW ...
                int diff = 0;
                for (int x = i; x < i + 8; x++) {
                    for (int y = j; y < j + 8; y++) {
                        int val = (x + y + k) % 2;
                        char c = (val == 1 ? 'B' : 'W');
                        if (b[x][y] != c) diff++;
                    }
                }

                mn = min(mn, diff);
            }
        }
    }

    cout << mn;
    return 0;
}
```

1.7 어린 왕자 BOJ #1004

원의 경계가 맞닿거나 교차하는 경우가 없다는 조건에 주목합시다. 조금만 생각해 보면 어떤 원에 대해 출발점과 도착점 중 한 점만 그 원에 들어가 있다면, 좋든 싫든 그 원의 경계를 넘어야 된다는 것을 알 수 있습니다. 두 점 모두가 원에 들어가 있으면 원 안에서만 움직이면 되고, 두 점 모두가 원 밖에 있다면 그 원을 피해가면 됩니다.

따라서 모든 원에 대해 출발점이나 도착점 중 하나만 원에 들어가 있는 경우만 세 주면 정답입니다.

점이 원 안에 들어가 있는가의 여부는 점과 원의 중심과의 거리와 원의 반지름을 비교해 구할 수 있습니다. 두 점 사이의 거리는 $\sqrt{(a_x-b_x)^2+(a_y+b_y)^2}$로 구할 수 있습니다.

이 문제도 모든 상수가 정수로 주어지는데, 굳이 루트를 씌우지 않고 $(a_x-b_x)^2+(a_y+b_y)^2$과 r^2를 비교해 실수 연산을 피할 수 있습니다.

정답 코드

실행 시간 12ms, 메모리 1,988KB

```cpp
#include <iostream>

using namespace std;

int dist2(int x1, int y1, int x2, int y2) {
    // (x1 - x2)^2 + (y1 - y2)^2
    return (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
}

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

    while (t--) {
        int x1, y1, x2, y2, n;
        cin >> x1 >> y1 >> x2 >> y2 >> n;

        int s = 0;
        while (n--) {
            int x, y, r;
            cin >> x >> y >> r;

            int r2 = r * r;
            int sdist = dist2(x1, y1, x, y); // distance from start
            int edist = dist2(x2, y2, x, y); // distance from end

            if ((sdist > r2) != (edist > r2)) s++;
        }
        cout << s << '\n';
    }
    return 0;
}
```

6번째 줄을 보시면 제가 주석에 수식을 적을 수 있다는 것을 알 수 있습니다. 부럽죠???

1.8 두 용액 BOJ #2470

용액의 특성 값들이 주어집니다. 이 중 두 용액을 골랐을 때 특성값의 합이 0에 가장 가까운 두 용액을 골라야 합니다. 그런데 용약의 개수가 최대 10^5 개입니다. 모든 경우의 수를 다 해본다면 아마도 TLE를 받게 될 거 같습니다.

하지만 용액들을 정렬한다면 어떨까요? 가장 빠른 정렬 방법은 $\mathcal{O}(n \log n)$ 이고, 10^5 개쯤이야 시간 안에 정렬할 수 있습니다. 정렬을 한 후, 맨 왼쪽 원소 l 과 맨 오른쪽 원소 r 을 시작으로 다음과 같이 진행합니다.

- 만약 $l + r = 0$ 이라면, 0에 더 가까워질 수 없으니 알고리즘을 종료합니다.
- 만약 $l + r > 0$ 이라면, 합을 줄이면 0에 가까워질 수 있으니 r 을 왼쪽으로 한 칸 옮겨서 확인합니다.
- 만약 $l + r < 0$ 이라면, 합을 키우면 0에 가까워질 수 있으니 l 을 오른쪽으로 한 칸 옮겨서 확인합니다.

이렇게 하면서 얻은 합들의 최솟값을 출력하면 됩니다. 왼쪽과 오른쪽에서 시작했지만 N 개의 원소를 전부 한 번씩 확인하므로 합을 계산하는 부분은 $\mathcal{O}(n)$ 이고, 정렬이 $\mathcal{O}(n \log n)$ 이었으므로 상대적으로 의미없는 시간 복잡도가 됩니다. 이 방법으로 충분히 AC를 받을 수 있습니다.

정답 코드

실행 시간 52ms, 메모리 2,380KB

```cpp
#include <iostream>
#include <algorithm> // sort()

using namespace std;

int a[100000];

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

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

```

```
16      sort(a, a + n);
17
18      int mn = abs(a[0] + a[n - 1]), mnl = a[0], mnr = a[n - 1];
19
20      int s = 0, e = n - 1;
21      while (s < e) {
22          int l = a[s], r = a[e];
23          int sum = l + r;
24
25          if (mn > abs(sum))  {
26              mn = abs(sum);
27              mnl = l;
28              mnr = r;
29          }
30
31          if (sum < 0) {
32              s++;
33          } else if (sum > 0) {
34              e--;
35          } else {
36              break;
37          }
38      }
39
40      cout << mnl << ' ' << mnr;
41
42      return 0;
43  }
```

이 문제에서 들어오는 수의 개수는 10^5 개에 육박합니다. 입력이 이렇게 큰 경우 입력 속도도 실행 시간에 영향을 미칠 수 있습니다. `cin`을 쓰는데 입력이 큰 경우 `int main()` 바로 아래에, 아래와 같이 두 줄을 추가하면 됩니다.

```
8   int main() {
9       ios_base::sync_with_stdio(false);
10      cin.tie(nullptr);
```

이 두 줄을 추가하는 것만으로 실행 시간이 16ms가 됩니다. 대신 `scanf` 등과 `cin` 등을 섞어 쓸 수 없게 됩니다.

1.9 수열의 합 BOJ #1024

지문이 상당히 간단합니다. 문제는 간단해 보이지 않습니다. N과 L이 주어질 때 길이가 적어도 L인 가장 짧은 음이 아닌 연속된 정수 리스트를 구해야 합니다. 첫 번째 문제였던 **시그마**의 강화판인 듯해 보입니다.

하지만 리스트의 길이에 제한이 있습니다. 길이가 100보다 긴 리스트는 체크해

볼 필요가 없습니다. 따라서 길이가 L인 리스트부터 100인 리스트까지 차례로 만드는 시도를 해 보면 될 것 같습니다.

길이 i인 리스트가 만들어지려면

$$n = k + (k+1) + \cdots + (k+i-1)$$

이어야 합니다. 이 말은 곧

$$n - 1 - 2 - \cdots - (i-1) = k + \cdots + k = ik$$

라는 뜻인데, 왼쪽 식은 시그마에서 했던 것처럼 간단히 표현할 수 있습니다. 정리하면

$$n - \frac{i(i-1)}{2} = ik$$

가 됩니다. k가 리스트의 시작임에 주의합시다.

계산의 편의를 위해 좌변을 x로 둡시다. 여기서 i를 고정한다고 생각하면, 좌변을 계산했을 때 i로 나누어떨어져야 합니다. 또한

$$\frac{x}{i} = k$$

인데, $k < 0$이면 안 되고 i는 양수이므로 $x < 0$이면 안 됩니다. 이 조건들을 만족하는 가장 작은 i가 곧 문제의 답임을 알 수 있습니다.

정답 코드

실행 시간 0ms, 메모리 1,984KB

```cpp
#include <iostream>

using namespace std;

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

    bool flag = false;

    // i = length;
    // 0+1+⋯+i≤n
    for (int i = l; i * (i - 1) / 2 ⩽ n && i ⩽ 100; i++) {
        int x = n - i * (i - 1) / 2;
        if (x % i ≠ 0) continue;
        if (x < 0) continue;
        flag = true;
        int k = x / i; // starting index
        for (int j = k; j < k + i; j++) {
                cout << j << ' ';
        }
        break;
    }
    if (!flag) {
        cout << -1;
```

```
    }
    return 0;
}
```

1.10 소수 구하기 BOJ #1929

두 정수 $M \leq N$ 사이에 존재하는 모든 소수를 구해야 합니다. 백만 개의 수에 대해 하나하나 약수의 개수를 구하는 건 굉장히 오랜 시간이 걸리기 때문에 소수들을 구하는 잘 알려진 알고리즘인 **에라토스테네스의 체**Sieve of Eratosthenes 알고리즘을 활용합시다. 에라토스테네스의 체는 이런 식으로 동작합니다.

- 일단 2부터 시작합니다. 2는 첫번째 소수입니다. 2를 제외한 2의 배수들을 모조리 지웁니다.
- 3은 지워지지 않았습니다. 그러면 3은 소수이므로, 3을 제외한 3의 배수들을 모두 지웁니다.
- 4는 지워졌습니다. 그러면 4는 어떤 소수의 배수라는 뜻이고, 이는 4가 소수가 아니라는 뜻이 됩니다. 스킵합니다.
- 5는 지워지지 않았습니다. 그러면 5은 소수이므로, 5를 제외한 5의 배수들을 모두 지웁니다.
- 이 과정을 계속 반복합니다.

이는 `bool` 배열로 비교적 간단하게 구현할 수 있는 알고리즘입니다. 이 알고리즘의 시간 복잡도는 $\mathcal{O}(n \log \log n)$ 입니다.

정답 코드

실행 시간 16ms, 메모리 2,964KB

```
#include <iostream>

using namespace std;

bool sieve[1000001]; // FALSE if prime

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

    sieve[0] = sieve[1] = true;

    for (int p = 2; p <= 1000000; p++) {
        if (sieve[p]) continue;
```

```
        if (m ⩽ p && p ⩽ n) {
            cout << p << '\n';
        }
        for (int x = p * 2; x ⩽ 1000000; x += p) {
            sieve[x] = true;
        }
    }

    return 0;
}
```