سوال زیرانداز

ابتدا مسئله partial sum دو بعدی را حل می کنیم. با اجرای کد زیر در نهایت ps[i][j] برابر مجموع زیر مستطیل با سطر های [0, i) و ستون های [0, j) می شود.

```
for (int i = 0; i < n; i++) {
       for (int j = 0; j < m; j++) {
              ps[i+1][j+1] = ps[i][j+1] + ps[i+1][j] - ps[i][j] + a[i][j];
        }
}
```

حال برای به دست آوردن مجموع اعداد یک زیرمستطیل دلخواه با سطر های [x1, x2) و ستون های [y1, y2) از تابع زیر استفاده می کنیم:

```
int get_sum(int x1, int y1, int x2, int y2) {
       return ps[x2][y2] - ps[x2][y1] - ps[x1][y2] + ps[x1][y1];
}
```

در نهایت برای همه ی زیرمسطیل های با حداقل مساحت k، مجموع اعداد آن را به دست می آوریم و کمترین را چاپ می کنیم. در کد زیر length را طوری انتخاب کرده ایم که مساحت زیرمستطیل حداقل k بشود.

```
    for (int x1 = 0; x1 < n; x1++) {
        for (int y1 = 0; y1 < m; y1++) {
            for (int x2 = x1+1; x2 <= n; x2++) {
                int length = ceil(1.0 * k / (x2 - x1));
                int y2 = y1 + length;
                if (y2 <= m) {
                    ans = min(ans, get_sum(x1, y1, x2, y2));
                }
            }
        }
    }
```

سوال کلمه

تعریف dp[a][b] : حداقل هزینه برای نوشتن کاراکتر های باقی مانده از t در صورتی که اشاره گر در اندیس a ام از کلمه s است و ما b حرف را از کلمه t نوشته ایم.

پیاده سازی را به صورت memoize انجام می دهیم.

```
int compute_dp(int a, int b) {
    if (b == m)
        return 0;
    if (dp[a][b] != -1)
        return dp[a][b];
    int ret = INF;
    for (int i = 0; i < n; i++) {
        if (s[i] == s[a]) {
            for (int j = -1; j <= 1; j++) {
                if (!j || i + j < 0 || i + j >= n)
                    continue;
                if (s[i + j] != t[b])
                    continue;
                ret = min(ret, compute_dp(i + j, b + 1) + abs(i - a) + 1);
            }
        }
    }
    return dp[a][b] = ret;
}
```

اگر b == m بود یعنی همه ی کاراکتر های t را نوشته ایم و هزینه باقی مانده صفر است.

در ابتدا همه ی خانه های آرایه dp را 1- می کنیم و چک می کنیم که اگر dp[a][b] == -1 بود، یعنی حساب نشده. وگرنه حساب شده و مقدارش را همان ابتدا خروجی می دهیم.

سپس روی این که در این گام اشاره گر را کجا ببریم for می زنیم. اشاره گر را باید به جایی ببریم که در آن کاراکتر s[a] نوشته شده باشد و در حداکثر فاصله ۱ از آن کارکتر t[b] نوشته شده باشد. در واقع i اندیسی است که ابتدا اشاره گر را به آن می بریم و i+j اندیسی است که کاراکتر بعدی را می نویسیم.

سپس بین همه ی حالات مینیمم می گیریم و از dp[i+1][b+1] که برای گام بعدی است استفاده می کنیم.

سوال چگالی

برای هر وسیله یک مقدار balance = a – b*k تعریف می کنیم. a جرم و b حجم است. پیدا کردن چگالی k معادل این است که مجموع balance وسیله هایی که بر می داریم برابر صفر باشد.

تعریف dp[i][j] : برای i وسیله ی اول، وزن سنگین ترین زیرمجموعه با balance = j چقدر است؟

در این جا j می تواند منفی باشد. برای همین از آرایه که نمی تواند اندیس منفی داشته باشد استفاده نمی کنیم، بلکه از map استفاده می کنیم.

در is[i][j] هم یک bool نگه می داریم که نشان می دهد dp[i][j] وجود دارد یا ندارد(همچین زیرمجموعه ای نیست).

dp را به صورت زیر آپدیت می کنیم:

```
is[0][0] = 1;
for(int i = 0 ; i <= n ; i++){
       for(int j = -MA ; j < MA ; j++){
              if(!is[ i%2 ][j])
          continue;

      int balance = a[i]-b[i]*k;
      smax(dp[ (i+1)%2 ][ j+balance ] , dp[ i%2 ][j] + a[i]);
              is[ (i+1)%2 ][ j+balance ] = 1;

              smax(dp[ (i+1)%2 ][j] , dp[ i%2 ][j]);
              is[ (i+1)%2 ][j] = 1;
       }
}
```

دلیل گذاشتن i%2 ها بهینه کردن حافظه است. از آن جایی که هر سطر از آرایه dp فقط به سطر قبلی نیاز دارد پس می توان باقی مانده به ۲ شماره سطر را گذاشت.