1. A certain string processing language allows the programmer to break a string into two pieces. It costs n units of time to break a string of n characters into two pieces, since this involves copying the old string. A programmer wants to break a string into many pieces, and the order in which the breaks are made can affect the total amount of time used. For example, suppose we wish to break a 20-character string after characters 3, 8, and 10. If the breaks are made in left-right order, then the first break costs 20 units of time, the second break costs 17 units of time, and the third break costs 12 units of time, for a total of 49 steps. If the breaks are made in right-left order, the first break costs 20 units of time, the second break costs 10 units of time, and the third break costs 8 units of time, for a total of only 38 steps. Give a dynamic programming algorithm that takes a list of character positions after which to break and determines the cheapest break cost in O(n3) time.

در ابتدا اقدام به تعریف چند متغیر میکنیم.

K: برابر با تعداد break هاست و آنها تکراری نیستند. (اگر طول رشته n باشد، $k < n$)

شکست[1] i بعد از کاراکتر s_i اتفاق میافتد ($1 \le i \le k$). ما دو مورد اضافه s_0 و s_{k+1} را برای سادگی در اول و آخر اضافه میکنیم.

تابع مورد نظر را اینگونه تعریف میکنیم:

$mino(i,j)$
$=$ the optimal cost of breaking a string segment consisting of characters si + 1 to sj

که مقدار بازگشتی آن برابر میشود با:

$$mino(i,j) = (s_i - s_j) + \min_{i<\alpha<j}\big(mino(i,\alpha) + mino(\alpha,j)\big) \rightarrow when\ j - i \ge 2.$$

ما اقدام به محاسبه $mino(0, k+1)$ میکنیم. بهترین حالت زمان j-i=1 است.

[1] break

برای اینکار ما احتیاج به جدول $(k+2)(k+2)$ نیاز داریم که سطر و ستونش از ۰ تا k+1 ایندکس شده باشند. البته ما فقط از بالا مثلثی آن استفاده میکنیم. قطعه کد زیر پیاده سازی این الگوریتم را نشان میدهد.

```
for i = 0 to k do: table[i][i+1] = 0; \\ base condition
for g = 2 to k+1 do:
    for i = 0 to k+1-j do:
        j = i + g;
        min = inf;
        minAlpha = NULL;
        for alpha = i+1 to j-1 do:
            cost = sj-si + table[i][a] + table[a][j];
            if (cost < min)
                min = cost;
                minAlpha = alpha;
        table[i][j] = min;
        back_track[i][j] = minAlpha;
```

برای پیدا کردن راه باید از متد زیر استفاده کنیم:

```
backtrack(i, j){
    if (i+1 == j) return;
    print(back_track[i][j]);
    backtrack(i, back_track[i][j]);
    backtrack(back_track[i][j], j);
}
```

ما اقدام به محاسبه backtrack(0, k+1) میکنیم.

زمان الگوریتم به دلیل داشتن سه حلقه تو در تو $O(n^3)$ است.

2. The traditional world chess championship is a match of 24 games. The current champion retains the title in case the match is a tie. Each game ends in a win, loss, or draw (tie) where wins count as 1, losses as 0, and draws as 1/2. The players take turns playing white and black. White has an advantage, because he moves first.
The champion plays white in the first game. He has probabilities w_w, w_d, and wl of winning, drawing, and losing playing white, and has probabilities b_w, b_d, and b_l of winning, drawing, and losing playing black.
(a) Write a recurrence for the probability that the champion retains the title. Assume that there are g games left to play in the match and that the champion needs to win i games (which may end in a 1/2).
(b) Based on your recurrence, give a dynamic programming to calculate the champions probability of retaining the title.
(c) Analyze its running time for an n game match.

آ)

```
void func(int g, int i){
    // base conditions
    if (g>=0 && i<=0) return 1;
    if (g == 0  && i > 0) return 0;

    // g is odd, plays Black
    if (g % 2) return bw * func(g-1, i-1) + bd * func(g-1, i-0.5) + bl * func(g-
1 , i);
    else return ww * func(g-1, i-1) + wd * func(g-1, i-0.5) + wl * func(g-1 , i);
}
```

ب) ما به یک جدول دوبعدی نیاز داریم تا تمام ترکیبات i و g را نمایش دهیم. ۲۴ مسابقه وجود دارد، بنابراین تعداد بازی های باقی مانده $0 \leq g \leq 24$ است. چون قهرمان با تساوی هم قهرمان میشود حداکثر به ۱۲ امتیاز نیاز دارد. $0 \leq i \leq 12$. چون گام های i ۰,۵ است پس اندازه جدول 25×25 میشود.

```
for g in range (24):
    table[g][0] = 1.0 # g>=0 and i=0 champ retains

for i in range(0.5, 12, 0.5):
    table[i][0] = 0.0 # g==0 and i!=0

for g in range(1,24):
    for i in range(0.5, 12, 0.5):
        if g is even :
            table[g][i] = ww*table[g-1, i-1]+ wd*table[g-1, i-
0.5] + wl * table[g-1, i]
        else :
            table[g][i] = bw*table[g-1, i-1]+ bd*table[g-1, i-
0.5] + bl * table[g-1, i]
```

البته در کد بالا ما واحدهای i را نیم درنظرگرفتیم که این در پیاده سازی باید مورد توجه قرار گیرد.

پ) اگر n بازی در جریان باشد، قهرمان به نصف تعداد بازی امتیاز نیاز دارد (مساوی حکم برد دارد). و سایز جدول $\theta(n^2)$ میشود که برای هر دسترسی به خانه ای از آن نیاز به $\theta(1)$ است. در کل به دلیل دو حلقه تو در تو به $\theta(n^2)$ نیاز داریم.