طراحی الگوریتم- برنامه‌ریزی پویا

محسن هوشمند

برنامه‌ریزی پویا

الگوریتم بازگشتی فیبوناتچی برای مقدار n همراه با فراخوانی به تعداد نمایی از n است. در هر نمونه از مسئله تقسیم و غلبه تقسیم به زیرنمونه‌های کوچکتر که هر یک به صورت بی‌خبر از هم یکبار دیگر حل خواهند شد. پس می‌توان گفت تقسیم و غلبه روشی است که اصطلاحا بالا به پایین است. چنین روشی در مسائلی مانند ادغامی بهینه عمل می‌کند چون زیرنمونه‌ها مستقل از یکدیگرند. اما در فیبوناتچی زیرنمونه‌ها دارای وابستگی هستند. مثال فیبو ۵ به محاسبهٔ فیبو ۴ و فیبو ۳ نیاز دارد. در حالی که دو مقدار مذکور وابسته به یکدیگرند.

برنامه‌ریزی پویا روشی عکس را در دستور کار دارد. همانند تقسیم و غلبه نمونه را به زیرنمونه می‌شکند. اما پس از حل زیرنمونه، نتیجه آن را ذخیره می‌کند و در مواقع نیاز دوباره به مقدار زیرنمونه صرفا نتیجه را برمی‌گرداند و از محاسبهٔ دوباره اجتناب می‌شود. الگوریتم بهینه‌ای که مقادیر را در آرایه ذخیره نگه می‌دارد از نوع برنامه‌ریزی پویا است، زیرا مقادیر را در آرایه‌ای به طول $n+1$ ذخیره و در موعد مقرر استفاده می‌کند. پس، جهت تقریب ذهن روشی پائین به بالا است. همچنین می‌توان الگوریتم را به نوعی بهبود داد که از فضای کمتری از حافظه استفاده کند. سخن کوتاه، مراحل برنامه‌ریزی پویا عبارتند از

- ایجاد: استفاده از خاصیت بازگشتی جهت حل نمونه‌ای از مسئله
- حل نمونهٔ مسئله از روش پایین به بالا با حل مقدم نمونه‌های کوچکتر

ضریب دوجمله‌ای

در ترکیبیات تعداد ترکیب حاصل از انتخاب m عضو از مجموعه‌ای n عضوی به صورت زیر محاسبه می‌شود.

$$\binom{n}{m} = \frac{n!}{m!\,(n-m)!}, 0 \le m \le n$$

محاسبهٔ بالا را می‌توان به صورت بازگشتی زیر نیز حساب کرد.

$$\binom{n}{m} = \begin{cases} \binom{n-1}{m-1} + \binom{n-1}{m}, 0 < m < n \\ 1, m = 0 \text{ یا } m = n \end{cases}$$

در نتیجه، شبه‌کد تقسیم و غلبهٔ الگوریتم محاسبهٔ ترکیب m از n را می‌توان به صورت زیر نوشت.

```
Alg. Tarkib_ToQ(n,m):
   if m == 0 | n == m:
      return 1
   return Tarkib_ToQ(n-1,m-1) + Tarkib_ToQ(n-1,m)
```

کد سی آن به صورت زیر است.

```
#include <stdio.h>


int trkib_ToQ(int m, int n) {
   if (n == 0 || n == m) {
      return 1;
   }
   if (n > m) {
      return 0;
   }


   //Tqsim o qlbe: C(m, n) = C(m-1, n-1) + C(m-1, n)
   return trkib_ToQ(m - 1, n - 1) + trkib_ToQ(m - 1, n);
}


int main() {
   int m, n;
   printf("mqdare m: ");
   scanf("%d", &m);
   printf("mqdare n: ");
   scanf("%d", &n);
   int ntije = trkib_ToQ(m, n);
   printf("C(%d, %d) = %d\n", m, n, ntije);


   return 0;
}
```

تعداد فراخوانی برابر با $2\binom{n}{m} - 1$ است. در نتیجه برای مقادیر بزرگتر ناکارآمد است. الگوریتمی دیگر نیز طراحی می‌توان کرد که کاراتر است و از خاصیت بازگشتی اما به صورت دیگر استفاده می‌کند. روش مزبور از نوع پائین به بالاست و و ابتدا مقادیر کمتر را محاسبه و نگه می‌دارد و سپس در موعد لازم از آنها استفاده می‌کند. جهت روش الگوریتم مذکور ابتدا نحوه محاسبه را به ماتریس تحویل می‌دهیم.

$$B[i,j] = \begin{cases} B[i-1,j-1] + B[i-1,j], 0 < j < i \\ 1, j = 0 \text{ یا } j = i \end{cases}$$

حال با استفاده از تعریف بازگشتی بالا جدولی ایجاد که ردیف‌های آن معادل مقادیر i و ستون‌های آن معادل مقادیر j هستند. با قرار دادن مقادیر در جدول ستون و قطر اصلی برابر یک است (چرا؟). حال هر مدخل براساس دو مدخل بالا و شمال‌غربی به دست می‌آید.

	0	۱	۲	۳	۴	...	m
0	۱						
۱	۱	۱					
۲	۱↘	۲↓	۱				
۳	۱	۳	۳	۱			
۴	۱	۴	۶	۴	۱		
...				↘	↓		
n	۱				+		

در ردیف صفر ماتریس داریم.

$$B[0,0] = ۱$$

ردیف یک ماتریس مقادیر زیر قرار می‌گیرند.

$$B[۱,0] = ۱$$

$$B[۱,۱] = ۱$$

ردیف دو ماتریس مقادیر زیر قرار می‌گیرند.

$$B[۲,0] = ۱$$

$$B[۲,۱] = B[۱,0] + B[۱,۱] = ۱ + ۱ = ۲$$

$$B[۲,۲] = ۱$$

درایه‌های ردیف سوم ماتریس برابر با مقادیر زیر هستند.

$$B[۳,0] = ۱$$

$$B[۳,۱] = B[۲,0] + B[۲,۱] = ۱ + ۲ = ۳$$

$$B[۳,۲] = B[۲,۱] + B[۲,۲] = ۲ + ۱ = ۳$$

$$B[۳,۳] = ۱$$

مدخل‌های ردیف چهارم عبارتند از

$$B[۴,0] = ۱$$

$$B[۴,۱] = B[۳,0] + B[۳,۱] = ۱ + ۳ = ۴$$

$$B[۴,۲] = B[۳,۱] + B[۳,۲] = ۳ + ۳ = ۶$$

$$B[۴,۳] = B[۳,۲] + B[۳,۳] = ۳ + ۱ = ۴$$

$$B[۴,۴] = ۱$$

به بیان دیگر، شبه‌کد آن به صورت زیر است.

```
def Tarkib_BP(n,m):
   C = [0]_{n+1 × m+1}
   for i  = 0 : n
      for j = 0: min(i, m):
         # Paye
         if j == 0 | j == i:
            C[i][j] = 1
         # estefade as mqadire Zakhire
         C[i][j] = C[i-1][j-1] + C[i-1][j]
   return C[n][m]
```

کد پیاده‌سازی در زبان سی آن به صورت زیر است.

```
#include <stdio.h>
#include <stdlib.h>


int trkib_BP(int m, int n) {
   // Create a 2D table to store intermediate ntijes
   int **C = (int **)malloc((m + 1) * sizeof(int *));
   for (int i = 0; i <= m; i++) {
      C[i] = (int *)malloc((n + 1) * sizeof(int));
   }


   // BP paeen be bala
   for (int i = 0; i <= m; i++) {
      for (int j = 0; j <= n; j++) {
         if (j == 0 | | j == i) {
            C[i][j] = 1;
         } else if (j > i) {
            C[i][j] = 0;
         } else {
            C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
         }
      }
   }
```

```
    int ntije = C[m][n];

    for (int i = 0; i <= m; i++) {
        free(C[i]);
    }
    free(C);

    return ntije;
}

int main() {
    int m, n;
    printf("mqdare m: ");
    scanf("%d", &m);
    printf("mqdare n: ");
    scanf("%d", &n);

    int ntije = trkib_BP(m, n);
    printf("C(%d, %d) = %d\n", m, n, ntije);

    return 0;
}
```

جهت تعیین پیچیدگی زمانی، تعداد تکرارهای حلقه داخلی را در جدول زیر می‌بینیم.

مقدار i	0	۱	۲	۳		m	$m+۱$	...	n
تعداد تکرار حلقه داخلی متناظر	0	۱	۲	۳		m	$m+۱$	...	$m+۱$

تعداد تکرارها برابر جمع مقادیر بالاست. به بیان دیگر،

$$۱+۲+۳+۴+\cdots+m + \overbrace{(m+۱)+(m+۱)+\cdots+(m+۱)}^{n-m+۱} = \frac{(۲n-m+۲)(m+۱)}{۲} \in \theta(nm).$$

تمرین- فضای اشغالی الگوریتم معرفی شده چقدر است؟ الگوریتم را صرفا فهرستی یک بعدی با اندازه m + ۱ بازنویسی کنید.

کوتاهترین تمامی مسیرها-- الگوریتم فلوید-وارشال

الگوریتم‌های دایکسترا و بلمن-فورد کوتاهترین مسیر تمامی رأس‌ها از گرهی مبدأ را محاسبه می‌کنند. اما گاهی اوقات کوتاهترین مسیر بین هر جفت از گره‌ها لازم است. الگوریتم فلوید-وارشال مآموریت حل چنین مسئله‌ای است. الگوریتم مذکور هم در گراف‌های جهت‌دار و هم بی‌جهت کاربرد دارد و به نوعی مسئله‌ای بهینه‌سازی است (چرا؟). به سخن دیگر، بین دو گره ممکن است مسیرهای گوناگونی موجود باشد. هدف مسئله کوتاهترین آنها است. در صورتی که از الگوریتم جستجوی کامل از هر راس استفاده شود مرتبهٔ آن بدتر از نمایی و از مرتبهٔ فاکتوریل است. اما با استفاده از الگوریتم ب‌پ مرتبه زمان اجرای درجه سه خواهد شد.

در ابتدا نمایش ماتریس فاصله را تعریف می‌کنیم. ماتریس مجاورت در قطر اصلی و مواردی که یالی وجود ندارد مقدار صفر و در صورت وجود یال مدخل متناظر برابر مقدار وزن است. مثلا

	0	1	2	3	4
0	0	3	0	7	0
1	8	0	2	0	5
2	5	0	0	1	0
3	2	0	0	0	4
4	0	6	0	3	0

یافتن کمترین مسافت بین هر دو گره i و j

$D^{(k)}[i,j] =$ طول کوتاهترین مسیر از رأس i به j با استفاده از راس‌های ۱، ۲،...،k به عنوان گره‌های واسط

مقادیر زیر را جهت دیدن مثال ببینید.

$D^{(0)}[۲,۴] =$	$Fasele[۲,۴] = \infty$
$D^{(۱)}[۲,۴] =$	$\text{کم}(Fasele[۲,۴], Fasele[۲,۱,۴] = \text{کم}(\infty, ۱۵) = ۱۵$
$D^{(۲)}[۲,۴] =$	با شروع از راس ۲ از راس ۲ نمی‌گذرد. $D^{(۱)}[۲,۴] = ۱۵$
$D^{(۳)}[۲,۴] =$	$\text{کم}\left(D^{(۲)}[۲,۴], D^{(۲)}[۲,۱,۳] + D^{(۲)}[۳,۴], D^{(۲)}[۲,۳] + D^{(۲)}[۳,۱,۴]\right)$

$$fasele[i,j] = \begin{cases} \text{وزن یال در صورت وجود یالی جفت گره} \\ \text{در صورت عدم وجود یال جفت گره}, \infty \\ 0, j = i \end{cases}$$

دو ماتریس در هر مرحله لازم است. ماتریس بروز کردن مقادیر فاصله بین راس‌ها با استفاده از افزودن راس k-ام به عنوان راس واسط و ماتریسی جهت نگه‌داری رد راس‌های واسط به ازای کوتاهترین مسیر یافت شده بین راس‌ها. در ماتریس واسط در صورت وجود یال بین هر دو راس راس شروع را در جدول واسط اضفه می‌کنیم، وگرنه تهی قرار می‌دهیم. پس در ابتدا هر ردیفی که دارای یالی باشد برابر با مقدار اندیس ردیف متناظر می‌شود(می‌توان از اندیس ستون در هر ستون بهره برد، صرفا نحوهٔ خواندن تغییر می‌کند). در ابتدا ماتریس‌ها به صورت زیر مقداردهی می‌شوند

$F^{(0)} = 3$

	1	2	3	4	5
1	0	3	∞	7	∞
2	8	0	2	∞	5
3	5	∞	0	1	∞
4	2	∞	∞	0	4
5	∞	6	∞	3	0

$vaset^{(0)} =$

	1	2	3	4	5
1	ϕ	۱	ϕ	۱	ϕ
2	۲	ϕ	۲	ϕ	۲
3	۳	ϕ	ϕ	۳	ϕ
4	۴	ϕ	ϕ	ϕ	۴
5	ϕ	۵	ϕ	۵	ϕ

در مرحله بعد راس نخست به عنوان واسط لحاظ می‌شود. روشن است که سطر و ستون نخست و به همراه قطر اصلی دستخوش تغییر نمی‌شود.

$D^{(1)} = 3$

	1	2	3	4	5
1	0	3	∞	7	∞
2	8	0	2	۱۵	5
3	5	∞	0	1	∞
4	2	∞	∞	0	4
5	∞	6	∞	3	0

$$vaset^{(۱)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \phi & ۱ & \phi & ۱ & \phi \\ 2 & ۲ & \phi & ۲ & ۱ & ۲ \\ 3 & ۳ & \phi & \phi & ۳ & \phi \\ 4 & ۴ & \phi & \phi & \phi & ۴ \\ 5 & \phi & ۵ & \phi & ۵ & \phi \end{array}$$

در حالت استفاده از راس یک به عنان راس واسط مسیری به ۲ و ۴ یافت شد. ر نتیجه مقدار در مدخل متناظر (2,4) با مقدار جدید تعویض و در جدول واسط نیز با مقدار تهی با راس یک تعویض شد. با استفاده از مقادیر بدست آمده و جدول‌های جدید، راس دوم امتحان می‌شود. نتایج در جدول‌های زیر آمده است.

$$D^{(۲)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & ۵ & 7 & ۸ \\ 2 & 8 & 0 & 2 & ۱۵ & 5 \\ 3 & 5 & \infty & 0 & 1 & \infty \\ 4 & 2 & \infty & \infty & 0 & 4 \\ 5 & ۱۱ & 6 & ۸ & 3 & 0 \end{array}$$

$$vaset^{(۲)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \phi & ۱ & ۲ & ۱ & ۲ \\ 2 & ۲ & \phi & ۲ & ۱ & ۲ \\ 3 & ۳ & \phi & \phi & ۳ & \phi \\ 4 & ۴ & \phi & \phi & \phi & ۴ \\ 5 & ۲ & ۵ & ۲ & ۵ & \phi \end{array}$$

با بدست آمدن ماتریس‌ها با استفاده از راس های یک و دو به عنوان واسط، از راس سه به عنوان واسط استفاده می‌شود. جداول زیر نتایج حاصل را برای مثال نشان می‌دهند.

$$D^{(۳)} = 3 \quad \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & ۵ & ۶ & ۸ \\ 2 & ۷ & 0 & 2 & ۳ & 5 \\ 3 & 5 & \infty & 0 & 1 & \infty \\ 4 & 2 & \infty & \infty & 0 & 4 \\ 5 & ۱۱ & 6 & ۸ & 3 & 0 \end{array}$$

$$vaset^{(۳)} = 3$$

	1	2	3	4	5
1	ϕ	۱	۲	۳	۲
2	۳	ϕ	۲	۳	۲
	۳	ϕ	ϕ	۳	ϕ
4	۴	ϕ	ϕ	ϕ	۴
5	۲	۵	۲	۵	ϕ

در مرحلهٔ بعد راس چهارم به عنوان راس واسط افزوده می شود که منحر به نتایج زیر می‌شود.

$$D^{(۴)} =$$

	1	2	3	4	5
1	0	3	۵	۶	۸
2	۵	0	2	۳	5
3	5	∞	0	1	۵
4	2	∞	∞	0	4
5	۵	6	۴	3	0

$$vaset^{(۴)} =$$

	1	2	3	4	5
1	ϕ	۱	۲	۳	۲
2	۴	ϕ	۲	۳	۲
3	۳	ϕ	ϕ	۳	۴
4	۴	ϕ	ϕ	ϕ	۴
5	۴	۵	۴	۵	ϕ

در نهایت راس پنجم به عنوان راس واسط انتخاب می‌شود که نتایج زیر به دست می‌آیند.

$$D^{(۵)} =$$

	1	2	3	4	5
1	0	3	۵	۶	۸
2	۵	0	2	۳	5
3	5	∞	0	1	۵
4	2	۱۰	۸	0	4
5	۵	6	۴	3	0

	1	2	3	4	5
1	ϕ	۱	۲	۳	۲

$$vaset^{(۵)} = \begin{array}{c|ccccc} 2 & ۴ & \phi & ۲ & ۳ & ۲ \\ 3 & ۳ & \phi & \phi & ۳ & ۴ \\ 4 & ۴ & ۵ & ۵ & \phi & ۴ \\ 5 & ۴ & ۵ & ۴ & ۵ & \phi \end{array}$$

سخن کوتاه، محاسبهٔ D^k از D^{k-1} از دو حالت حاصل می‌شود.

۱- مسیرکوتاهتر بدون واسطه بیشتر بدست آمده است. پس

$$D^k[i,j] = D^{k-1}[i,j]$$

۲- مسیر کوتاهتر با استفاده از واسطهٔ جدید

$$D^k[i,j] = D^{k-1}[i,k] + D^{k-1}[k,j]$$

پس داریم

$$D^k[i,j] = \text{کم}(D^{k-1}[i,j], D^{k-1}[i,k] + D^{k-1}[k,j])$$

شبه‌کد آن به صورت زیر است.

```
Alg. kotahtarin_tamami_masirha(mat, n):
   F = mat
   vaset[n,n] = {-1}
   for i = 0 : n-1
      for j 0 : n - 1
         if i !=j & mat[i,j] ==0
           F[i,j] = ∞
         elseif mat[i,j] != 0 & mat[i,j] < inf
           vaset[i,j] = i
   for k = 0 : n-1
      for i = 0 : n - 1
         for j = 0 : n-1
            if i!=j & k !=i & k!=j
               F[i,j] = min([F[i,j], F[i,k] + F[k,j]])
               vaset[i][j] = k; #vaset[k][j];
   return F, vaset
```

با نحوه مقداردهی ماتریس واسط، شبه‌کد چاپ مسیر را می‌توان به صورت زیر نوشت

```
Alg. chap (vaset, q, r):
   if vaset[q,r] != 0:
      chap (vaset, q, vaset[q,r])
      print(vaset[q,r])
```

```
    chap (vaset, vaset[q,r], r)
```

اجرای الگوریتم فلوید-وارشال در زبان سی به صورت زیر می‌تواند باشد.

```
#include <stdio.h>
#define INF 1000000
#define N 5 // tdad gereha


void NmayeshMsir(int vaset[N][N], int i, int j) {
   int v = vaset[i][j];
   if (v == -1){
      printf("bi masir\n");
      return;
   }
   if (i == j) {
      printf("%d ", i);
      return;
   }
  if (v == i || v == j) {
      printf("%d -> ", i);
      return;
   }
   NmayeshMsir(vaset, i, v);
   NmayeshMsir(vaset, v, j)
}


void floydWarshall(int graph[N][N]) {
   int fasele[N][N], vaset[N][N];


   // mqdardhi matrix fasele o matrix vaset
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         if (graph[i][j] == 0 && i != j) {
            fasele[i][j] = INF; //bdon yal
            vaset[i][j] = -1;
         } else {
            fasele[i][j] = graph[i][j];
            vaset[i][j] = (graph[i][j] != 0) ? i : -1;
         }
      }
   }


   //
   for (int k = 0; k < N; k++) {
      for (int i = 0; i < N; i++) {
         for (int j = 0; j < N; j++) {
            if (fasele[i][k] != INF && fasele[k][j] != INF) {
```

```
                if (fasele[i][j] > fasele[i][k] + fasele[k][j]) {
                    fasele[i][j] = fasele[i][k] + fasele[k][j];
                    vaset[i][j] = k;
                }
            }
        }
    }

    printf("kotahtarin faselehar joft gereh:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (fasele[i][j] == INF) {
                printf("INF ");
            } else {
                printf("%d ", fasele[i][j]);
            }
        }
        printf("\n");
    }

    // msirha
    printf("\nmsirhaye beine gereha:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (i != j) {
                printf("kotahtrin msir az %d be %d: ", i, j);
                NmayeshMsir(vaset, i, j);
                    printf("%d\n", j);//printf("\n");
            }
        }
    }
}

int main() {
    int graph[N][N] = {
        {0, 3, 0, 7, 0},
        {8, 0, 2, 0, 5},
        {5, 0, 0, 1, 0},
        {2, 0, 0, 0, 4},
        {0, 6, 0, 3, 0}
    };
    floydWarshall(graph);
    return 0;
}
```

زمان $T(n) = \theta(n^3)$

اصل بهینه‌سازی بلمن

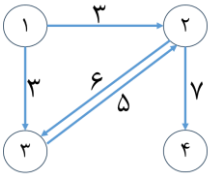
الگوریتم قبلی نه تنها وزن کوتاهترین مسیرها را مشخص می‌کند بلکه کوتاهترین مسیر را نیز ایجاد می‌کند. ایجاد راه‌حل بهینه گام سوم از روش برنامه‌ریزی پویا در مسئله‌ای بهینه‌سازی است. می‌توان یافتن بهترین مسیر و ایجاد راه‌حل بهینه را با هم پیش برد.

خواننده ممکن است گمان برد که هر مسئلهٔ بهینه‌سازی را بتوان با برنامه‌ریزی پویا حل کرد. اما چنین نیست. در واقع مسئله باید دارای خاصیت اصل بهینگی باشد.

اصل بهینگی بلمن: بر مسئله‌ای اعمال‌پذیر است اگر راه‌حل بهینهٔ نمونه‌ای از مسئله موجود باشد، آن‌گاه تمامی زیرنمونه‌های آن نیز دارای راه حل بهنیه باشند.

مثال- کوتاهترین مسیرها: در صورت وجود گره k در مسیر بهینه از i به j، آن‌گاه از i به k و از k به j نیز حتما بهینه هستند. در نتیجه پاسخ بهنیه به نمونه‌ای، به طریق اولی، موجب ایجاد پاسخ بهینهٔ تمامی زیرنمونه‌هایش خواهد شد. در صورت برقراری اصل بهینگی در مسئله‌ای خاصیت بازگشتی پاسخی بهینه برای نمونه‌ای را بدست خواهد داد که دارای پاسخ‌های بهینه برای تمامی زیرنمونه‌های آن باشد. دلیل ظریف و مهم امکان استفاده از برنامه‌ریزی پویا این است که ایجاد پاسخ بهینه برای نمونه‌ای آن‌گاه پاسخ‌های بهینه برای تمامی زیرنمونه‌ها می‌تواند هر پاسخ نمونه‌ای باشد.

بررسی اعمال‌پذیری اصل بهینگی بلمن کاری دم دستی و راحت نیست. مثلا، در یافتن طولانی‌ترین مسیر ساده درگراف جهت‌دار با وزن نامنفی لزوما پاسخ بهینه بین دو راس با پاسخ بهینه زیرنمونه‌ها یکی نیست. شکل زیر از راس ۱ به راس ۴ دارای [1,3,2,4] است. اما از ۱ به ۳ بجای [1,3] پاسخ برابر [1,2,3] است. در نتیجه پاسخ بهینه زیرنمونه در پاسخ نمونه اصلی قرار ندارد و در نتیجه اصل بهینگی برقرار نیست.



ضرب زنجیری ماتریس‌ها

در ضرب دو ماتریس عملیات اصلی را ضرب در نظر می‌گیریم. چرا؟ ضرب ماتریسی دو ماتریس نیاز به چند ضرب دارد؟ فرض ماتریس نخست دارای اندازهٔ $m \times n$ و ماتریس دوم $n \times l$ باشد. آن‌گاه تعداد ضرب‌های لازم برای ضرب دو ماتریس برابر $m \times n \times l$ است.

احتمال ضرب چند ماتریس را در نظر می‌گیریم.

$M_۱$	$M_۲$	$M_۳$	$M_۴$
۲۰×۳	۳×۳۰	۳۰×۱۵	۱۵×۸

ترتیب‌های متفاوت ضرب ماتریس‌ها نیاز به تعداد زیادی ضرب عددی دارد. همه حالت‌ها برای ضرب چهار ماتریس مثالی به صورت زیر است.

$M_۱(M_۲(M_۳M_۴))$	$۳۰ \times ۱۵ \times ۸ + ۳ \times ۳۰ \times ۸ + ۲۰ \times ۳ \times ۸$	$= ۴۸۰۰$
$(M_۱M_۲)(M_۳M_۴)$	$۲۰ \times ۳ \times ۳۰ + ۳۰ \times ۱۵ \times ۸ + ۲۰ \times ۳۰ \times ۸$	$= ۵۸۸۰$
$M_۱((M_۲M_۳)M_۴)$	$۳ \times ۳۰ \times ۱۵ + ۳ \times ۱۵ \times ۸ + ۲۰ \times ۳ \times ۸$	$= ۲۱۹۰$
$((M_۱M_۲)M_۳)M_۴$	$۲۰ \times ۳ \times ۳۰ + ۲۰ \times ۳۰ \times ۱۵ + ۲۰ \times ۱۵ \times ۸$	$= ۱۳۲۰۰$
$(M_۱(M_۲M_۳))M_۴$	$۳ \times ۳۰ \times ۱۵ + ۲۰ \times ۳ \times ۱۵ + ۲۰ \times ۱۵ \times ۸$	$= ۴۶۵۰$

حالت بهینه از ۲۱۹۰ عملیات ضرب استفاده می‌برد. به دنبال الگوریتمی هستیم که ترتیب بهینه ضرب چند ماتریس در هم را تعیین کند. جستجوی کامل انواع ترتیب‌ها از مرتبه زمانی خواهد بود.

$$M_۱\left(\underbrace{M_۲M_۳\cdots M_n}_{z_{n-۱}\text{ ترتیب مختلف}}\right)$$

از مرتبه نمایی است. اما اصل بهینگی برقرار است. مثلا اگر $M_۱\left(\left(\left((M_۲M_۳)M_۴\right)M_۵\right)M_۶\right)$ آن‌گاه ضرب $(M_۲M_۳)M_۴$ نیز بهینه است. پس می‌توان از برنامه‌ریزی پویا جهت ایجاد راه‌حل استفاده کرد.

برای $1 \le i \le j \le n$

$M[i,j]$ کمترین تعداد ضرب M_i تا M_j و $i < j$

$$M[0,0] = 0$$

مثال

$M_۱$	$M_۲$	$M_۳$	$M_۴$	$M_۵$	$M_۶$
۶×۳	۳×۴	۴×۵	۵×۷	۷×۸	۸×۹
$d_۰\ d_۱$	$d_۱\ d_۲$	$d_۲\ d_۳$	$d_۳\ d_۴$	$d_۴\ d_۵$	$d_۵\ d_۶$

تعداد ضرب‌های $(M_۴M_۵)M_۶$ $d_۳d_۴d_۵ + d_۳d_۵d_۶ = ۵ \times ۷ \times ۸ + ۵ \times ۸ \times ۹ = ۶۴۰$

تعداد ضرب‌های $M_۴(M_۵M_۶)$ $d_۴d_۵d_۶ + d_۳d_۴d_۶ = ۷ \times ۸ \times ۹ + ۵ \times ۷ \times ۹ = ۸۱۹$

ترتیب بهینه ضرب شش ماتریس حتما دارای یکی از موارد ترتیب زیر را رعایت می‌کند (چرا؟).

$$M[۱,۱]M[۲,۶]$$
$$M[۱,۲]M[۳,۶]$$
$$M[۱,۳]M[۴,۶]$$
$$M[۱,۴]M[۵,۶]$$
$$M[۱,۵]M[۶,۶]$$

به طوری که در هر $M[i,j]$ ضرب از طریق ترتیبی بهینه از ماتریس‌ها بدست می‌آید. به دیگر سخن، تعداد ضرب‌های کل برابر با تعداد $M[1,k]$ و $M[k+1,6]$ ضرب دو حاصلضرب در یکدیگر است. بنابرانی، تعداد ضرب‌ها برابر با مقدار زیر است.

$$M[1,k] + M[k+1,6] + d_0 d_k d_6$$

پس جهت محاسبهٔ تعداد ضرب کمینه داریم

$$M[1,6] = \min_{1 \le k \le 5} (M[1,k] + M[k+1,6] + d_0 d_k d_6)$$

هیچ دلیلی بر محدودیت بر اندیس‌های بالا نیست. ضرب ماتریس‌های i-ام تا j-ام نیز به طریق مشابه تعریف می‌شود.

$$M[i,j] = \min_{i \le k \le j-1} \left(M[i,k] + M[k+1,j] + d_i d_k d_j\right)$$

$$M[0,0] = 0$$

حل آن با تقسیم و غلبه برابر زمانی نمایی است. شبیه مثلث خیام پاسکال که استفاده شد را بکار می‌بریم و الگوریتمی بر مبنای برنامه‌ریزی پویا اجرا می‌کنیم.

- قطر اصلی برابر صفر
- محاسبه بالا قطر
- سپس بالای قطر بعدی
- مثال

$M_۱$ $M_۲$ $M_۳$ $M_۴$ $M_۵$ $M_۶$

۶×۳	۳×۴	۴×۵	۵×۷	۷×۸	۸×۹
$d_۰\ d_۱$	$d_۱\ d_۲$	$d_۲\ d_۳$	$d_۳\ d_۴$	$d_۴\ d_۵$	$d_۵\ d_۶$

تعداد ضرب‌های $(M_۴M_۵)M_۶$ $\quad d_۴d_۵d_۶ + d_۳d_۵d_۶ = ۵\times۷\times۸ + ۵\times۸\times۹ = ۶۴۰$

تعداد ضرب‌های $M_۴(M_۵M_۶)$ $\quad d_۴d_۵d_۶ + d_۳d_۴d_۶ = ۷\times۸\times۹ + ۵\times۷\times۹ = ۸۱۹$

مثال

قطر صفر:

$$M[i,i] = ۰, ۱ \le i \le ۶$$

قطر یک:

$$M[۱,۲] = \min_{۱\le k\le ۱}(M[۱,k] + M[k+۱,۲] + d_۰d_kd_۲)$$
$$= M[۱,۱] + M[۲,۲] + d_۰d_۱d_۲$$
$$= ۰ + ۰ + ۶\times۳\times۴ = ۷۲$$

$$M[۲,۳] = \min_{۲\le k\le ۲}(M[۲,k] + M[k+۱,۳] + d_۱d_kd_۳)$$
$$= M[۲,۲] + M[۳,۳] + d_۱d_۲d_۳$$
$$= ۰ + ۰ + ۳\times۴\times۵ = ۶۰$$

$$M[۳,۴] = \min_{۳\le k\le ۳}(M[۳,k] + M[k+۱,۴] + d_۲d_kd_۴)$$
$$= M[۳,۳] + M[۴,۴] + d_۲d_۳d_۴$$
$$= ۰ + ۰ + ۴\times۵\times۷ = ۱۴۰$$

$$M[۴,۵] = \min_{۴\le k\le ۴}(M[۴,k] + M[k+۱,۴] + d_۳d_kd_۵)$$
$$= M[۴,۴] + M[۵,۵] + d_۳d_۴d_۵$$
$$= ۰ + ۰ + ۵\times۷\times۸ = ۲۸۰$$

$$M[۵,۶] = \min_{۵\le k\le ۵}(M[۵,k] + M[k+۱,۶] + d_۴d_kd_۶)$$
$$= M[۵,۵] + M[۶,۶] + d_۴d_۵d_۶$$

$$= 0 + 0 + ۷ \times ۸ \times ۹ = ۵۰۴$$

قطر دوم

$$M[۱,۳] = \min_{۱ \le k \le ۲} (M[۱,k] + M[k+۱,۳] + d_0 d_k d_۳)$$

$$= \text{کم}(M[۱,۱] + M[۲,۳] + d_0 d_۱ d_۳, M[۱,۲] + M[۳,۳] + d_0 d_۲ d_۳)$$

$$= \text{کم}(0 + ۶۰ + ۶ \times ۳ \times ۵,۷۲ + 0 + ۶ \times ۴ \times ۵) = ۱۵۰$$

$$M[۲,۴] = \min_{۲ \le k \le ۳} (M[۲,k] + M[k+۱,۴] + d_۱ d_k d_۴)$$

$$= \text{کم}(M[۲,۲] + M[۳,۴] + d_۱ d_۲ d_۴, M[۲,۳] + M[۴,۴] + d_۱ d_۳ d_۴)$$

$$= \text{کم}(0 + ۱۴۰ + ۳ \times ۴ \times ۷,۶۰ + 0 + ۳ \times ۵ \times ۷) = ۱۷۵$$

$$M[۳,۵] = \min_{۳ \le k \le ۴} (M[۳,k] + M[k+۱,۵] + d_۲ d_k d_۵)$$

$$= \text{کم}(M[۳,۳] + M[۴,۵] + d_۲ d_۳ d_۵, M[۳,۴] + M[۵,۵] + d_۲ d_۴ d_۵)$$

$$= \text{کم}(0 + ۲۸۰ + ۴ \times ۵ \times ۸, ۱۴۰ + 0 + ۴ \times ۷ \times ۸) = ۳۶۴$$

$$M[۴,۶] = \min_{۴ \le k \le ۵} (M[۴,k] + M[k+۱,۶] + d_۳ d_k d_۶)$$

$$= \text{کم}(M[۴,۴] + M[۵,۶] + d_۳ d_۴ d_۶, M[۴,۵] + M[۶,۶] + d_۳ d_۵ d_۶)$$

$$= \text{کم}(0 + ۵۰۴ + ۵ \times ۷ \times ۹, ۲۸۰ + 0 + ۵ \times ۸ \times ۹) = ۶۴۰$$

قطر سوم

$$M[۱,۴] = \min_{۱ \le k \le ۳} (M[۱,k] + M[k+۱,۴] + d_0 d_k d_۴)$$

$$= \text{کم}(M[۱,۱] + M[۲,۴] + d_0 d_۱ d_۴, M[۱,۲] + M[۳,۴] + d_0 d_۲ d_۴, M[۱,۳] + M[۴,۴] + d_0 d_۳ d_۴)$$

به همرین طریق بقیه مقادیر در قطرهای چهارم و پنجم و ششم را محاسبه و مقادر $M[۱,۶]$ برابر تعداد ضرب‌های در حالت پرانتزگذاری بهینه ضرب زنجیری ماتریس‌هاست. جدول زیر مقادیر را تا مراحل حساب شده نشان می‌دهد. بقیه مدخل‌ها را پر کنید.

لاقطر۰	لاقطر۱	لاقطر۲	لاقطر۳	لاقطر۴	لاقطر۵	
	۱	۲	۳	۴	۵	۶
۱	۰	۷۲	۱۵۰			
۲		۰	۶۰	۱۷۵		
۳			۰	۱۴۰	۳۶۴	
۴				۰	۲۸۰	۶۴۰
۵					۰	۵۰۴
۶						۰

تمرین- چاپ ترتیب نمایش:

P[2,5] = 4 به معنای (A2A3A4)A5

P[1,6] = 1 به معنای A1(A2A3A4A5A6)

P[2,6] = 5 به معنای (A2A3A4A5)A6 و در نتیجه A1((A2A3A4A5)A6) و A1(((A2A3A4)A5)A6) و A1((((A2A3)A4)A5)A6)

شبه‌کد الگوریتم ضرب چند ماتریس به صورت زیر است.

یافتن ترتیب به علاوه مقدار کمینه

```
Alg zrbZnjireMatrix(d[n+1], n)
..... for i = 0 : n-1
```

```
…….. m[i,i] = 0
….. for q = 1 : n - 1
…….. for i = 1 : n - q
…….. j = i + q ;
…….. m[i,j] = ∞
…….. for k = i : j - 1
…….. hzine = m[i,k] + m[k + 1,j] + d[i - 1] * d[k] * d[j];
…….. if (hzine < m[i,j]) {
…….. m[i,j] = hzine;
…….. t[i,j] = k; // mhal tqsim
….. return m[1, n-1] , t
```

نحوه نمایش پرانتزگذاری به صورت زیر است.

```
Alg. chaptrtibBhine(t[,], i, j) {
….. if (i == j)
…….. print("M_i)
….. else
…….. print("(")
…….. chaptrtibBhine(t, i, t[i,j])
…….. chaptrtibBhine(t, t[i,j] + 1, j)
…….. printf(")")
```

پیاده‌سازی یافتن ترتیب ضرب چند ماتریس در سی به صورت زیر است.

```
#include <stdio.h>
#include <limits.h>


// Chap trtib
void chaptrtibBhine(int t[][100], int i, int j) {
   if (i == j) {
      printf("M%d", i);
   } else {
      printf("(");
      chaptrtibBhine(t, i, t[i][j]);
      chaptrtibBhine(t, t[i][j] + 1, j);
      printf(")");
   }
}


// Function to find the minimum number of multiplications and the optimal order
int trtibZnjireMatrix(int d[], int n) {
   // m[i][j] zkhir tdad kmineye zrbha az i ta j
   int m[n][n];


   // t[i][j] zkhireye mhal noqat tqsim
   int t[100][100];


   // hzine (tdad zrbhaye) tak matrix
```

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


    for (int q = 2; q < n; L++) {
        for (int i = 1; i < n - q + 1; i++) {
            int j = i + q - 1;
            m[i][j] = INT_MAX; // mqdare avaleye


            // Azmayesh kole noqat tqsim momken
            for (int k = i; k <= j - 1; k++) {
                int hzine = m[i][k] + m[k + 1][j] + d[i - 1] * d[k] * d[j];
                if (hzine < m[i][j]) {
                    m[i][j] = hzine;
                    t[i][j] = k; // mhal tqsim
                }
            }
        }
    }


    // chap tribe zrb
    printf("prantezgozari bhine: ");
    chaptrtibBhine(t, 1, n - 1);
    printf("\n");


    return m[1][n - 1];
}


int main() {
    int arr[] = {20, 3, 30, 15, 8};
    int n = sizeof(arr) / sizeof(arr[0]);


    printf("mqdare kmine tdada zrbhaye lazem:: %d\n", trtibZnjireMatrix(arr, n));


    return 0;
}
```

معادله $\sum_{q=1}^{n-1}\big((n-q)q\big)$ تعداد محاسبات در هر مرحله را نشان می‌دهد. در نتیجه، زمان اجرا از مرتبه $\theta(n^3)$ است.

طولانی‌ترین زیردنباله مشترک

مسئلهٔ بزرگترین زیردنبالهٔ مشترک (ب‌زم) از مسائل معروف و پراستفاده در علم رایانه و زی‌انفورماتیک است. مجموعه‌ای از رشته‌های متناهی روی الفبای متناهی فرض می‌شود. هدف ب‌زم یافتن بزرگترین

زیردنباله‌ای است که بین تمامی رشته‌ها مشترک باشد. توجه به معنای زیردنباله مهم است. زیردنباله رشته‌ای حاصل از رشتهٔ اصلی است که صرفا ترتیب مهم است و توالی پی در پی بر عکس زیررشتهٔ مشترک اهمیتی ندارد. از سال ۱۳۴۹ مسئلهٔ بزم موضوع تحقیقی بوده است که بسیاری از محققان سعی بر حل بهینه یا نزدیک بهینهٔ آن داشته و دارند. از روش‌های متفاوت مانند برنامه‌نویسی پویا، برنامه‌ریزی خطی می‌توان استفاده برد و حل دقیق بزم در حالتی که مجموعه دارای دقیقا دو رشته باشد را بدست آورد. در حالت دو رشته‌ای زمان اجرا از مرتبهٔ $O(n^2)$ است به طوری که n طول بلندترین رشته از بین دو رشته است. اما برای مجموعه رشته‌هایی با تعداد بیشتر از دو مسئله ان-پی سخت است.

- الفبا
- رشته
- مجموعه رشته
- تعداد رشته‌ها در هر مجموعه

$$S_1 = \text{ACGTTCAGTCAGTCAT}$$
$$S_2 = \text{ACGTCACGTTGTCAGT}$$
$$S_3 = \text{ACGTCACGTTCTGTCA}$$
$$\vdots$$
$$S_N = \text{ACGTACGTTCCTGTCAGT}$$

$$S = \{S_1, S_2, S_3, \dots, S_N\}$$

$$|S| = N$$

$$\Sigma = \{A, C, G, T\}$$

$$s_1 = 1011100010$$
$$s_2 = 0010111010$$
$$s_3 = 0101010111$$

$$S = \{s_1, s_2, s_3\}$$

$$|S| = 3$$

$$\Sigma = \{0, 1\}$$

در مسئلهٔ بلندترین زیردنبالهٔ مشترک ترتیب اهمیت دارد ولی توالی اهمیتی ندارد. به دنبال نویسه‌های تشکیل دهنده در تمام رشته‌ها هستیم و از آن‌ها زیردنباله‌ای تشکیل می‌دهیم که بلندترین دنباله ممکن باشد. در مقابل در بلندترین زیررشته مشترک توالی علاوه بر ترتیب در بلندترین زیررشته مشترک واجد اهمیت است. مثال $\Sigma = \{A, C, G, T\}$ و مجموعه $S=\{\text{AACACTTG}, GACGATG\}$، $|S| = 2$ آن‌گاه بلندترین زیردنبالهٔ مشترک ACATG است.

اولین راه‌حل ایجاد تمامی زیردنباله‌های ممکن یکی از رشته‌ها و بررسی وجود آنها در رشتهٔ دیگر است، و طولانی‌ترین را تعیین می‌کند. روشن است که مرتبه اجرا در اندازهٔ نمائی است و کاربردی نیست. اما می‌توان برای مسئلهٔ بزم خاصیت بهینگی زیرساختار را نشان داد. پیشوند یکی از رشته‌ها را در نظر می‌گیریم.

قضیه- زیرساختار بهینهٔ بزم: اگر $S = s_1 s_2 \dots s_m$ و $T = t_1 t_2 \dots t_n$ دو دنباله باشند. و $Z = z_1 z_2 \dots z_k$ بزم دو رشتهٔ مذکور باشند. داریم:

۱- اگر $s_m = t_n$ آن‌گاه $z_k = s_m = t_n$ و Z_{k-1} بزم S_{m-1} و T_{n-1} است.

۲- اگر $s_m \neq t_n$ آن‌گاه $z_k \neq s_m$ دلالت بر این دارد که Z بزم S_{m-1} و T است.

۳- اگر $s_m \neq t_n$ آن‌گاه $z_k \neq t_m$، دلالت بر این دارد که Z بزم S و T_{n-1} است.

اثبات- برای مورد اول اگر $z_k \neq s_m$ میتوان $s_m = t_n$ را به زیردنباله مشترک افزود، پس فرض خلف اشتباه است. جهت مورد دوم، اگر $z_k \neq s_m$ اگر زیردنباله‌ای با طول بزرگتر از k برای S_{m-1} و T داشته باشیم، آن‌گاه زیرنبالهٔ مذکور زیردنبالهٔ S و T نیز خواهد بود که با فرض در تناقض است. بخش سوم هم مانند مورد دوم اثبات می‌شود.

راه‌حل- بازگشتی

مسئله دارای خاصیت هم‌پوشانی زیرنمونه‌ها است. جهت یافتن بزم دو رشته نیاز به حل S و T_{n-1} و همچنین حل S_{m-1} و T است. هر چند نیاز به حل S_{m-1} و T_{n-1} است. بنابراین می‌توان همانند ضرب زنجیری ماتریس‌ها از خاصیتی بازگشتی بهره برد. A[i,j] طول زیردنباله مشترک S_i و T_j باشد. اگر i=0 یا j=0 آن‌گاه زیردنباله نیز دارای طول صفر خواهد بود. اگر $s_i = t_j$ باشد، آن‌گاه ادامهٔ زیردنباله را می‌توان از S_{i-1} و T_{j-1} بدست آورد. اگر $s_i \neq t_j$ باشد، آن‌گاه ادامه زیردنباله را می‌توان از بیش زیرنمونهٔ S_{i-1} و T_j و زیرنمونهٔ S_i و T_{j-1}بدست آورد. سخن کوتاه،

$$A[i,j] = \begin{cases} 0 & ,i = 0 \text{ یا } j = 0 \\ A[i-1, j-1] + 1, i, j > 0, s_i = t_j \\ \max(A[i, j-1], A[i-1, j]), i, j > 0, s_i \neq t_j \end{cases}$$

شبه‌کد آن به صورت زیر است. اما پیاده‌سازی بازگشتی (تقسیم و غلبهٔ روش) دارای زمان بالایی است.

```
Alg. BZM_ToQ(S[m], T[n])
..... ℓ.lcs = ''; ℓ.t=0
..... if m == 0 | n == 0
..... return ℓ
..... if S[m - 1] == T[ n – 1]
..... ℓ_b = BZM_ToQ(S[m - 1], T[n - 1])
..... ℓ.t += ℓ_b.t
..... ℓ.lcs = ℓ_b.lcs ⊕ ℓ.lcs
..... return ℓ
..... else
..... ℓ1 = BZM_ToQ(S[m], T[n - 1])
```

```
..... ℓ2 = BZM_ToQ(S[m - 1], T[n])
..... if ℓ1.t > ℓ2.t
..... ℓ.t += ℓ1.t
..... ℓ.lcs = ℓ1.lcs ⊕ ℓ.lcs
..... else
..... ℓ.t += ℓ2.t
..... ℓ.lcs = ℓ2.lcs ⊕ ℓ.lcs
```

اما از مرتبه نمایی است. تعداد زیرنمونه‌های مجزای مسئله با طول‌های m و n برابر $\theta(mn)$ است. پس روش پائین به بالا برنامه‌ریزی پویا می‌توانند به مرتبه‌ای بهتر مسئله را حل کند.

	j	0	۱	۲	۳	۴	۵	۶
i		yj	G	T	C	A	G	A
0	xi	0	0	0	0	0	0	0
۱	A	0	↑ 0	↑ 0	↑ 0	↖ ۱	← ۱	↖ ۱
۲	G	0	↖ ۱	← ۱	← ۱	↑ ۱	↖ ۲	← ۲
۳	C	0	↑ ۱	↑ ۱	↖ ۲	← ۲	↑ ۲	↑ ۲
۴	G	0	↖ ۱	↑ ۱	↑ ۲	↑ ۲	↖ ۳	← ۳
۵	T	0	↑ ۱	↖ ۲	↑ ۲	↑ ۲	↑ ۳	↑ ۳
۶	A	0	↑ ۱	↑ ۲	↑ ۲	↖ ۳	↑ ۳	↖ ۴
۷	G	0	↖ ۱	↑ ۲	↑ ۲	↑ ۳	↖ ۴	↑ ۴

شبه‌کد آن به صورت زیر است.

```
Alg. LCS_BP(char S[m], T[n] int m, int n, int bp[m + 1][n + 1]) {
  BP[m+1,n+1]={0}
  for i = 1 : m
    for j = 1 : n
      if (S[i] == Y[j])
        bp[i][j] = bp[i - 1][j - 1] + 1; // yaftan horof motabeq
      else
        bp[i][j] = max(bp[i - 1][j] , bp[i][j - 1])
  return bp[m][n]; // brgrdandan tool LCS
```

تمرین- شبه‌کد نمایش بزم را بنویسید.

برنامه سی الگوریتم برنامه‌ریزی پویا یافتن بزرگترین زیردنبالهٔ مشترک را در ادامه مشاهده می‌کنیم.

```
#include <stdio.h>
#include <string.h>


// Tabe BP jadval LCS o toll donbale
```

```
int LCS_BP(char *X, char *Y, int m, int n, int bp[m + 1][n + 1]) {
   // tkmil jadval BP
   for (int i = 0; i <= m; i++) {
      for (int j = 0; j <= n; j++) {
         if (i == 0 || j == 0) {
            bp[i][j] = 0; // reshte tohi
         } else if (X[i - 1] == Y[j - 1]) {
            bp[i][j] = bp[i - 1][j - 1] + 1; // yaftan horof motabeq
         } else {
            bp[i][j] = (bp[i - 1][j] > bp[i][j - 1]) ? bp[i - 1][j] : bp[i][j - 1]; // Yaftan bishine
         }
      }
   }
   return bp[m][n]; // brgrdandan tool LCS
}


// tabe nmayesh LCS
void nmayeshLcs(char *X, char *Y, int m, int n, int bp[m + 1][n + 1]) {
   int andis = bp[m][n]; // andis nhayee barabar tool lcs
   char lcs[andis + 1];
   lcs[andis] = '\0'; // entehaye reshte


   // harekat paeen be bala
   int i = m, j = n;
   while (i > 0 && j > 0) {
      if (X[i - 1] == Y[j - 1]) {
         lcs[andis - 1] = X[i - 1]; //afzodan harf
         i--;
         j--;
         andis--;
      } else if (bp[i - 1][j] > bp[i][j - 1]) {
         i--; // bala
      } else {
         j--; // chap
      }
   }
   printf("LCS: %s\n", lcs);
}


int main() {
   char X[] = "ABCBDBBAB";
   char Y[] = "BDCBBAB";


   int m = strlen(X);
   int n = strlen(Y);


   // jadval zakhireye tool
   int bp[m + 1][n + 1];
```

```
    int toolLcs = LCS_BP(X, Y, m, n, bp);
    printf("Toole LCS: %d\n", toolLcs);

    nmayeshLcs(X, Y, m, n, bp);

    return 0;
}
```

تمرین- الگوریتم نیدلمن وانش جهت هم‌ردیفی جفت توالی را بررسی کنید.

درخت جستجوی دودوئی (دجد)

درخت جستجوی دودوئی نوعی از درخت دودوئی از مقادیر (کلیدها) از مجموعه‌ای مرتب هستند که الف- هر گره دارای کلید یا مقدار است. ب- مقدار کلیدهای زیردرخت چپ هر گرهی کمتر از یا مساوی با مقدار کلید گره مذکور هستند. ج- مقدار کلیدهای زیردرخت راست هر گرهی بزرگتر از یا مساوی با مقدار کلید گره مذکور هستند. چنین درختی در اندیس‌گذاری چند سطحی در پایگاه داده multilevel indexing، مرتب‌سازی پویا، مدیریت بخش‌های حافظه مجازی در کرنل یونیکس کاربرد دارد.

عمق یا سطح گرهی در درخت تعداد یال‌هایی است که مسیری از ریشه به آن را برقرار می‌سازد. عمق درخت برابر مقدار عمق بیشینهٔ تمام گره‌های درخت است. درخت دودوئی متعادل است اگر عمق دو زیردرخت هر گره از درخت مزبور دارای تفاوت بیشتر از یک نباشند.

```
#include <stdio.h>
#include <stdlib.h>


// Ta'rife gereh
struct Gereh {
   int klid;
   struct Gereh* chap;
   struct Gereh* rast;
};


// Tabe ijad gereh jadid
struct Gereh* IjadeGereh(int klid) {
   struct Gereh* GerehJdid = (struct Gereh*)malloc(sizeof(struct Gereh));
   GerehJdid->klid = klid;
   GerehJdid->chap = GerehJdid->rast = NULL;
   return GerehJdid;
}


// Darje gereh dar derakht jotsojo dodoee (djd)
struct Gereh* darj(struct Gereh* rishe, int klid) {
   if (rishe == NULL) return IjadeGereh(klid);
   if (klid < rishe->klid)
      rishe->chap = darj(rishe->chap, klid);
   else if (klid > rishe->klid)
      rishe->rast = darj(rishe->rast, klid);
   return rishe;
}


// Jostojoye Gereh dar djd
struct Gereh* jostojo(struct Gereh* rishe, int mqdar) {
   if (rishe == NULL || rishe->klid == mqdar) return rishe;
```

```
    if (mqdar < rishe->klid)
        return jostojo(rishe->chap, mqdar);
    return jostojo(rishe->rast, mqdar);
}


// Yaftan gereh ba klid kamine
struct Gereh* GerehKlidKmine(struct Gereh* Gereh) {
    struct Gereh* jaari = Gereh;
    while (jaari && jaari->chap != NULL)
        jaari = jaari->chap;
    return jaari;
}


// Hazfe gereh az djd BST
struct Gereh* hazfGereh(struct Gereh* rishe, int mqdar) {
    if (rishe == NULL) return rishe;
    if (mqdar < rishe->klid)
        rishe->chap = hazfGereh(rishe->chap, mqdar);
    else if (mqdar > rishe->klid)
        rishe->rast = hazfGereh(rishe->rast, mqdar);
    else {
        if (rishe->chap == NULL) {
            struct Gereh* mvqt = rishe->rast;
            free(rishe);
            return mvqt;
        } else if (rishe->rast == NULL) {
            struct Gereh* mvqt = rishe->chap;
            free(rishe);
            return mvqt;
        }
        struct Gereh* mvqt = GerehKlidKmine(rishe->rast);
        rishe->klid = mvqt->klid;
        rishe->rast = hazfGereh(rishe->rast, mvqt->klid);
    }
    return rishe;
}


// Peymayesh miantrtib (chap, rishe, rast)
void miantrtib(struct Gereh* rishe) {
    if (rishe != NULL) {
        miantrtib(rishe->chap);
        printf("%d ", rishe->klid);
        miantrtib(rishe->rast);
    }
}


// Peymayesh pishtrtib (rishe, chap, rast)
void pishtrtib(struct Gereh* rishe) {
    if (rishe != NULL) {
```

```
        printf("%d ", rishe->klid);
                                pishtrtib(rishe->chap);
        pishtrtib(rishe->rast);
    }
}

// Peymayesh pastrtib (chap, rast, rishe)
void pastrtib(struct Gereh* rishe) {
    if (rishe != NULL) {
        pastrtib(rishe->chap);
        pastrtib(rishe->rast);
                                printf("%d ", rishe->klid);
    }
}

int main() {
    struct Gereh* rishe = NULL;
    rishe = darj(rishe, 50);
    darj(rishe, 30);
    darj(rishe, 20);
    darj(rishe, 40);
    darj(rishe, 70);
    darj(rishe, 60);
    darj(rishe, 80);
    darj(rishe, 10);
    darj(rishe, 90);
    darj(rishe, 12);
    darj(rishe, 8);


    printf("peymayesh miantrtib: ");
    miantrtib(rishe);
    printf("\n");


    printf("peymayesh pishtrtib: ");
    pishtrtib(rishe);
    printf("\n");


    printf("peymayesh pastrtib: ");
    pastrtib(rishe);
    printf("\n");


    rishe = hazfGereh(rishe, 20);
    printf("peymayesh miantrtib pas az hazf 20: ");
    miantrtib(rishe);
    printf("\n");


    rishe = hazfGereh(rishe, 30);
```

```
    printf("peymayesh miantrtib pas az hazf 30: ");
    miantrtib(rishe);
    printf("\n");


    rishe = hazfGereh(rishe, 50);
    printf("peymayesh miantrtib pas az hazf 50: ");
    miantrtib(rishe);
    printf("\n");


    darj(rishe, -4);
    printf("peymayesh miantrtib pas az darje -4: ");
    miantrtib(rishe);
    printf("\n");
    return 0;
}
```

میزان پیچیدگی زمانی هر یک از عملیات‌های روی درخت جستجو دودوئی در جدول زیر نشان داده شده است.

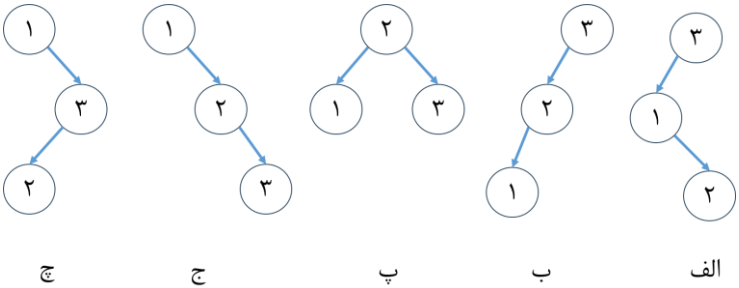
عملیات	بهترین زمان	زمان متوسط	بدترین زمان
درج	$O(۱)$	$O(\log n)$	$O(n)$
جستجو	$O(۱)$	$O(\log n)$	$O(n)$
حذف	$O(۱)$	$O(\log n)$	$O(n)$
پیمایش	$O(n)$	$O(n)$	$O(n)$

تمرین- بدترین در چه مواردی از درخت جستجو رخ می‌دهد. مرتب‌سازی با استفاده از درخت جستجو دارای چه زمانی است؟ جهت اجتناب از بدترین زمان می‌توان از درخت‌های خودمتعادل‌سازی مانند AVL و درخت سرخ-سیاه استفاده کرد که بدترین زمان را با $O(\log n)$ ضمانت می‌کنند. موارد مذکور را تحقیق کنید.

درخت جستجوی دودوئی (دجد) شامل مواردی است که بر اساس مقدار کلیدها بازیابی می‌شوند. هدف انجام بازیابی با میانگین زمان کمینه است. مسئلهٔ حاضر را «درخت جستجو دودویی بهینه» می‌خوانیم. بازیابی کلیدها داری توزیع احتمال یکنواخت نیست و میزان فراخوانی هر کلید متفاوت از کلید دیگر است. در بخش جاری صرفا جستجوهایی را در نظر می‌گیریم که کلید جستجو در درخت باشد. جهت کمینه‌سازی زمان جستجوی میانگین، نیاز به جستجوی تمامی درخت‌ها از مرتبه نمایی است. در عوض برنامه‌ریزی الگوریتمی کارآمد به دست می‌دهد. فرض کلید i تا کلید j در درخت طوری قرار یافته‌اند که $\sum_{m=i}^{j} p_m b_m$ را کمینه کند به طوری که b_m تعداد مقایسه‌های کلید m و p_m احتمال فراوانی متناظر کلید است. درخت را برای کلیدهای مذکور بهینه می‌خوانیم و A[i,j] را جهت نمایش مقدار بهینه منظور

می‌کنیم. داریم $A[i,i] = p_i$. چرا؟ چون جهت تعیین مقدار کلید در درخت با تک کلید به یک مقایسه نیاز داریم.

فرض می‌کنیم که $p_1 = 0.1$ $p_2 = 0.3, p_3 = 0.6$. جهت تعیین دجد بهینه $A[1,3]$ می‌توان تک تک دجد را به دست آورد و امید ریاضی جستجو را حساب کرد.



(الف) $0.1 \times (0 + 1) + 0.6 \times (1 + 1) + 0.3 \times (2 + 1) = 2.2$

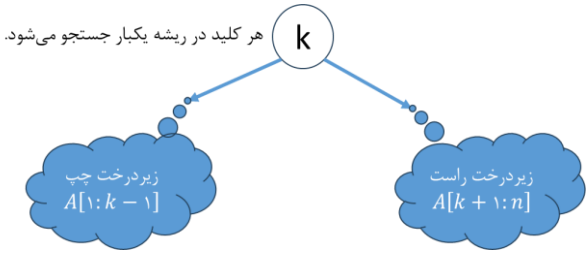
(ب) $0.1 \times (0 + 1) + 0.3 \times (1 + 1) + 0.6 \times (2 + 1) = 2.5$

(پ) $0.3 \times (0 + 1) + 0.6 \times (1 + 1) + 0.1 \times (1 + 1) = 1.7$

(ج) $0.6 \times (0 + 1) + 0.3 \times (1 + 1) + 0.1 \times (2 + 1) = 1.5$

(چ) $0.6 \times (0 + 1) + 0.1 \times (1 + 1) + 0.3 \times (2 + 1) = 1.7$

در نتیجه، دجد ج دارای کمترین امید ریاضی تعداد مقایسه است. اصل بهینگی در اینجا برقرار است. هر زیردرخت از درخت بهینه دارای بهینگی است. در صورتی که درخت ۱ با ریشه کلید ۱ بهینه و درخت دو با ریشهٔ کلید ۲ بهینه باشد، تا درخت n با کلید n بهینه باشد. به ازای $1 \le k \le n$ زیردرخت‌های درخت k باید بهینه باشند و در نتیجه زمان‌های جستجوی میانگین در چنین زیردرخت‌هایی را می‌توان با شکل زیر نمایش داد.



در تصویر بالا $A[1,k-1]$ و $A[k+1:n]$ به ترتیب برابر با میانگین زمان جستجو برای کلید در زیر درخت چپ و میانگین زمان جستجو برای کلید در زیردرخت راست است. هر گره دیگر غیر از ریشه در شکل مزبور دقیقا یک مقایسه در ریشه لازم خواهد بود. بنابراین زمان میانگین جستجو در درخت k برابر است با

$$A[1,k-1]+A[k+1,n]+\sum_{m=1}^{n} p_m$$

چون یکی از درخت‌ها بهینه است. پس زمان جستجوی میانگین درخت بهینه برابر است با

$$A[1,n]=\min_{1\le k\le n}(A[1,k-1]+A[k+1,n])+\sum_{m=1}^{n} p_m \ , A[1,0]=0, A[n+1,n]=0$$

$$\sum_{m=1}^{n} p_m = 1$$

اما برای کلید i تا کلید j

$$A[i,j]=\begin{cases}\min_{i\le k\le j}(A[i,k-1]+A[k+1,j])+\sum_{m=i}^{j} p_m \ , i<j \\ p_i, \quad j=i \\ 0, \quad i-j=1\end{cases}$$

تمرین- با کلیدهای 1,2,3,4 با احتمال‌های متناظر p1=3/8, p2=3/8, p3=1/8, p4=1/8 دجد کمینه را به دست آورید.

شبه‌کد درخت جستجوی دودویی بهینه به صورت زیر است.

```
Alg djdb(p[n], n)
..... for i = 1 : n
.......... A[i,i-1] = 0
.......... A[i,i] = p[i]
.......... R[i,i] = i
.......... R[i,i-1] = 0
..... A[n+1][n] = 0
..... R[n+1][n] = 0
..... for q = 1 : n - 1
.......... for i = 1 : n - q
.............. j = i + q ;
```

```
…… A[i,j] = ∞
…… for k = i : j - 1
…… hzine = A[i,k] + A[k + 1,j] + ∑_{m=i}^{j} p_m
…… if (hzine < A[i,j]) {
…… A[i,j] = hzine;
…… R[i,j] = k; // mhal kmine jahat rishe zirdrakht
….. return A[1, n] , R
```

نحوه نمایش پرانتزگذاری به صورت زیر است.

```
Alg. ijadrkht(R[,], i, j) {
….. k = R[i,j]
….. if (k == 0)
……. return ϕ
….. else
……. g = gereh jdid
……. g.klid = klid[k]
……. g.chp = ijadrkht(R[,], i, k - 1)
……. g.rast = ijadrkht(R[,], k + 1, j)
```

پیاده‌سازی آن در سی به صورت زیر است.

```
#include <stdio.h>
#include <limits.h>


// Mohasebeye fravani
int jame(int fravani[], int i, int j) {
   int jm = 0;
   for (int k = i; k <= j; k++)
      jm += fravani[k];
   return jm;
}


// Ijade djd bhine
int djdBhine(int klidha[], int fravani[], int n) {
   int hzine[n][n];


   // hzine avaleye tak klidha (halat paye)
   for (int i = 0; i < n; i++)
      hzine[i][i] = fravani[i];


   // Ijade jdvale paeen be bala
   for (int L = 2; L <= n; L++) { // L tole znjire
      for (int i = 0; i <= n - L; i++) {
         int j = i + L - 1;
         hzine[i][j] = INT_MAX;


         // Barressi tamami klidha dar bazeye (i, j)
         for (int r = i; r <= j; r++) {
```

```
            int hzinechap = (r > i) ? hzine[i][r - 1] : 0;
            int hzinerast = (r < j) ? hzine[r + 1][j] : 0;
            int hzinekol = hzinechap + hzinerast + jame(fravani, i, j);


            if (hzinekol < hzine[i][j])
               hzine[i][j] = hzinekol;
         }
      }
   }
   return hzine[0][n - 1];
}


int main() {
   int klidha[] = {10, 20, 30, 40};
   int fravani[] = {3, 3, 1, 1};
   int n = sizeof(klidha) / sizeof(klidha[0]);


   printf("hzine kmineye djdb: %d\n", djdBhine(klidha, fravani, n));
   return 0;
}
```

مرتبه زمانی برابر $O(n^3)$ است. البته با استفاده از بهینه‌سازی کنوث می‌توان زمان را به $O(n^2)$ کاهش می‌یابد. تمرین- روش کنوث چگونه است؟