
بسم الله الرحمن الرحیم

مؤسسه آموزش عالی بهار
غیرانتفاعی - غیردولتی

وزارت علوم، تحقیقات و فناوری

ساختمان داده‌ها

تهیه کننده: فرزاد زندی

zandi_farzad@yahoo.com

zandi8farzad@gmail.com

آنچه در این درس فرا می‌گیرید:

۱- مقدمه‌ای بر ساختمان داده‌ها
۲- روشهای تحلیل الگوریتمها
۳- صف, پشته, لیست و انواع آنها
۴- درختها
۵- روشهای مرتب‌سازی و درهم‌سازی

آنچه از شما انتظار می‌رود:

۱- درک مفاهیم ساختمان داده‌ها و الگوریتمها
۲- درک مفاهیم روشهای تحلیل الگوریتمها و توانایی تحلیل آنها
۳- آشنایی و درک مفاهیم پشته, صف, لیست و توانایی کار با آنها
۴- آشنایی با ساختار درختها و توانایی تحلیل آنها
۵- شناسایی و درک روشهای مرتب‌سازی و درهم‌سازی
۶- ارائه در کلاس درباره یکی از موضوعات مطرح شده

نحوه ارزیابی نمره:

۱- حضور مستمر
۲- فعالیت کلاسی
۳- انجام تمرینات محول شده
۴- ارائه در کلاس

۱۰ نمره

۵- امتحان میان ترم ۵ نمره
۶- امتحان پایان ترم ۵ نمره

جلسه دوم:

مقدمه‌ای بر ساختمان داده‌ها

آنچه در این جلسه می‌خوانید:

۱- حل مسأله بوسیله کامپیوتر
۲- نکاتی راجع به الگوریتمها
۳- کارایی الگوریتم
۴- محاسبه زمان اجرای الگوریتم
۵- آرایه‌ها
۶- پیاده سازی آرایه‌های یک بُعدی و دو بُعدی و کاربردهای آنها
۷- الگوریتم مرتب‌سازی انتخابی و محاسبه زمان اجرای آن

حل مسأله بوسیله کامپیوتر:

حل مسأله توسط کامپیوتر نیازمند استفاده از نرم‌افزار و سخت‌افزار است.

توسعه نرم‌افزار در چند مرحله انجام می‌شود که چرخه زندگی نرم‌افزار را تشکیل می‌دهد:

- تحلیل مسأله و مشخصات
- طراحی
- کُدنویسی
- تست , اجرا و اشکال‌زدایی
- نگهداری

نکاتی راجع به الگوریتمها:

الگوریتم مجموعه‌ای از دستورالعملها است که توسط کامپیوتر اجرا می‌شود.

۱- باید بدون ابهام باشد یعنی هر مرحله روشن, واضح, معین و قطعی باشد.
۲- باید به قدری ساده باشد که توسط کامپیوتر قابل اجرا باشد.
۳- باید خاتمه‌پذیر باشد.

کارایی الگوریتم:

برای حل یک مسأله ممکن است الگوریتمهای مختلفی وجود داشته باشدکه هر کدام دارای مزایا و معایب خود هستند.

کارایی الگوریتم بر اساس دو معیار بهره‌وری از فضا و کارایی زمان اندازه‌گیری می‌شود.

بهره‌وری فضا:

حافظه مورد نیاز برای ذخیره داده‌ها است.

کارایی زمان:

زمان لازم برای پردازش داده‌ها است.

محاسبه زمان اجرای الگوریتم:

زمان اجرای الگوریتم تحت تأثیر عوامل متعددی قرار دارد:

- اندازه ورودی:

تعداد مقادیر ورودی, در زمان پردازش آنها مؤثر استو اگر زمان اجرا را T در نظر بگیریم, باید به صورت تابع T(n) بیان شود که n اندازه ورودی است.

- نوع دستورالعمل‌ها
- سرعت اجرای ماشین

نکته:

T(n) را نمی‌توان برحسب واحدهای واقعی زمان مثل ثانیه بیان کرد, لذا T(n) تعداد تقریبیِ دفعاتی دستورالعملهایی است که باید اجرا شود که آنرا پیچیدگی زمانی الگوریتم می‌گویند.

آرایه‌ها:

آرایهٔ ساده‌ترین ساختمان داده‌ای است که وجود دارد. ساده‌ترین شکل آن, آرایه یک‌بُعدی است که بردار نیز نامیده می‌شود.

آرایه دو بُعدی شکل دیگری از آرایه‌ها است که ماتریس هم نامیده می‌شود.

نکته:

- آرایه مجموعه مرتب و محدودی از عناصر همگن است.
- طول و نوع آرایه در هنگام تعریف آن باید نعیین شود و در طول اجرای برنامه قابل تغییر نیست.
- در آرایه برای دستیابی به عنصری از آرایه از اندیس استفاده می‌شود.

آرایه یک بُعدی:

شکل کلی تعریف آرایه یک بُعدی:

نوع name[ظرفیت];

نوع name[ظرفیت] = {مقادیر اولیه};

مثال:

Int a[10] = {5,10,9,0,7,6,2,44,56,52};

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
a	5	10	9	0	7	6	2	44	56	52

آرایه دو بُعدی:

شکل کلی تعریف آرایه دو بُعدی:

نوع name[ظرفیت][ظرفیت];

نوع name[ظرفیت][ظرفیت] = {مقادیر اولیه};

مثال:

```
Int  a[3][3];
a[1][1] = 6;
a[1][2] = 8;
a[1][3] = 2;
...
```

6	8	2
11	18	4
3	20	9

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int  a[10] = {5,10,9,0,7,6,2,44,56,52};
    int i,j,temp;
    for (i=0;i<9;i++)
    {
        for (j=i+1;j<=9;j++)
        {
            if (a[j] < a[i])
            {
                temp = a[j];
                a[j] = a[i];
                a[i] = temp;
            }
        }
    }
}
```

الگوریتم مرتب‌سازی انتخابی

زمان اجرا:

حلقه اول n بار (۱۰ بار)

حلقه دوم n بار (۱۰ بار)

زمان کل:

$T(n) = n \times n = n^2$

جلسه سوم:

مقدمه‌ای بر ساختمان داده‌ها

آنچه در این جلسه می‌خوانید:

۱- الگوریتم جستجوی خطی و مرتبه آن
۲- الگوریتم جستجوی دودویی و مرتبه آن

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int  a[10] = {5,10,9,0,7,6,2,44,56,52};
    int  num,i;
    char  found =‘n’;

    cin>>num;
    for (i=0;i<=10;i++)
    {
        if   (num = a[i])  found = ‘y’;
    }
    if (found=‘y’)  cout<<“Item exist”;
    else  cout<<“Item does not exist”;
}
```

الگوریتم جستجوی خطی

زمان اجرا:
$T(n) = n$

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int  a[10] = {5,7,9,10,16,21,27,44,56,82};
    int  num,first,last,loc;
    char  found ='n';
    first = 0;
    last = 9;
    cin>>num;
    while (first<=last && found = 'n')
    {
        loc = (first+last)/2;
        if (num<a[loc])
            last = loc-1;
        else  if (num>a[loc])
            first = loc+1;
        else  found = 'y';
    }
    if (found='y')  cout<<"Item exist";
    else  cout<<"Item does not exist";
}
```

الگوریتم جستجوی دودویی در لیست مرتب صعودی

زمان اجرا:

$$T(n) = \log_2 n$$

نکته:

$$n = 2^k$$

$$k = \log_2 n$$

تمرین:

برنامه‌ای بنویسید که حاصل دو ماتریس ۲در۲ را محاسبه کند.

(هدف: استفاده از آرایه‌ها)

$$\begin{bmatrix} 5 & 3 \\ 8 & 9 \end{bmatrix} \times \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} (5*2)+(3*4) & (5*3)+(3*5) \\ (8*2)+(9*4) & (8*3)+(9*5) \end{bmatrix}$$

یادآوری:
حاصل ضرب دو ماتریس ۲در۲ ماتریسی ۲در۲ است.

جلسه چهارم:

پشته‌ها

آنچه در این جلسه می‌خوانید:

۱- پشته چیست؟

۲- تعریف نوع داده پشته

۳- روش‌های پیاده‌سازی پشته

۴- پیاده‌سازی پشته با آرایه

پشته چیست؟

پشته ساختمان داده‌ای است که در کامپیوتر کاربرد زیادی دارد.

در پشته عمل حذف و اضافه کردن عناصر, در یک طرف انجام می‌شود, به همین دلیل آنرا ساختمان داده LIFO (Last Input First Output) می‌نامند.

در ساختمان داده پشته, عنصری که دیرتر از همه وارد پشته شود, زودتر از همه خارج می‌شود.

مثال:

- چیدن چندین کتاب روی هم
- قرار دادن چندین بشقاب روی هم

تعریف نوع داده پشته:

با توجه به عملکرد پشته می‌توان آنرا به صورت یک نوع داده تعریف کرد:

مجموعه‌ای از عناصر داده‌ها:

مجموعه‌ای از عناصر مرتب که از یک طرف قابل دستیابی هستند و این طرف, بالای پشته نام دارد.

عملیات اصلی:

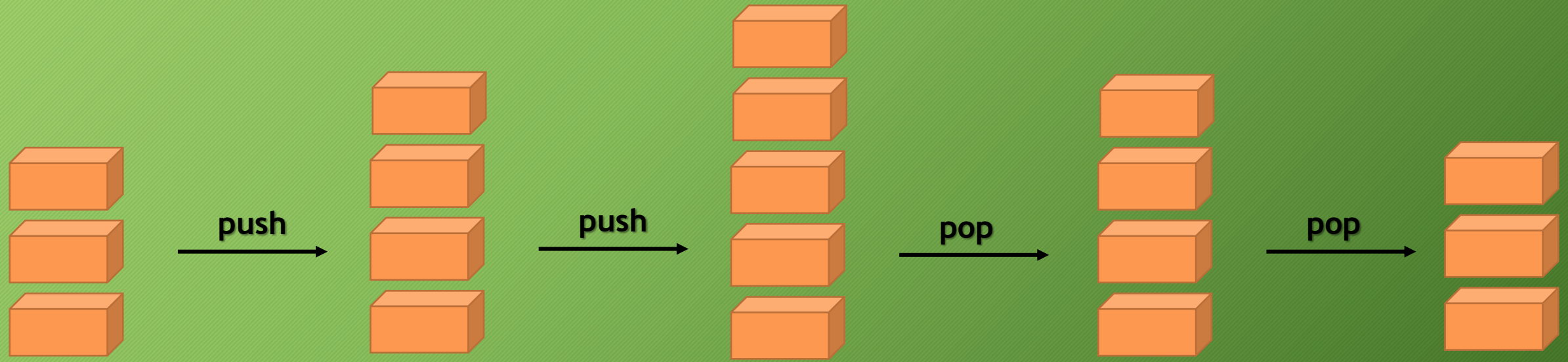
- Creator: ایجاد کننده پشته (پشته خالی ایجاد می‌کند).
- Empty: خالی بودن پشته را تست می‌کند.
- Push: عنصری به بالای پشته اضافه می‌کند.
- Pop: عنصری از بالای پشته حذف می‌کند.
- Top: عنصرِ بالای پشته را بازیابی می‌کند.

روش‌های پیاده‌سازی پشته:

پشته را می‌توان به دو شکل پیاده‌سازی کرد:

۱- پیاده‌سازی خطی با استفاده از آرایه

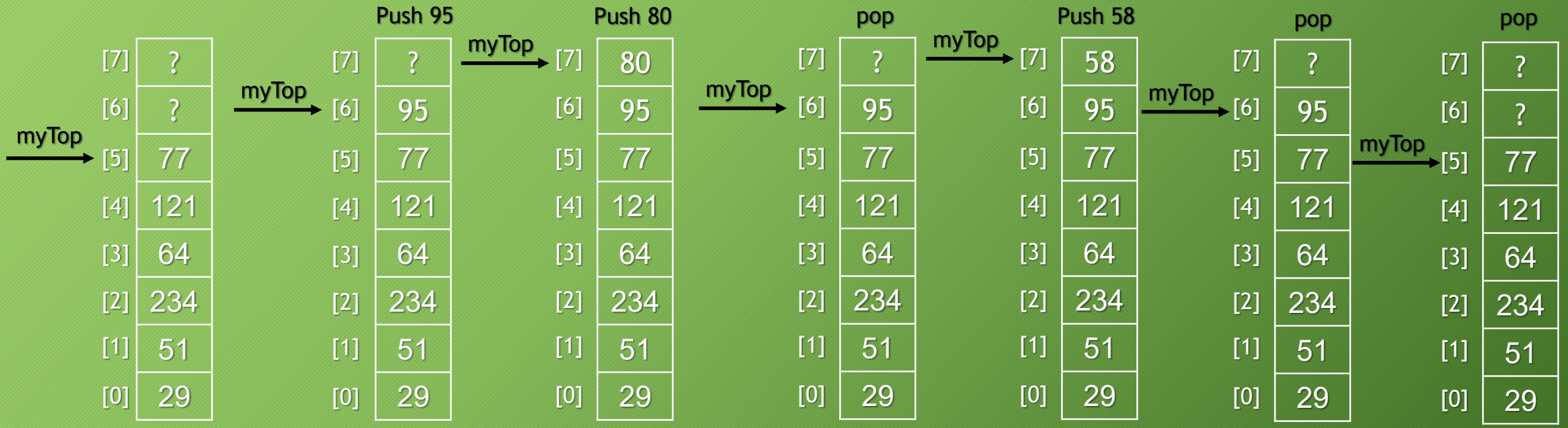
۲- پیاده‌سازی پیوندی با استفاده از لیست پیوندی



پیاده‌سازی پشته با آرایه:

در این روش برای ذخیره عناصر پشته از آرایه استفاده می‌کنیم. داده‌های مورد نیاز برای پشته در این روش عبارتند از:

- آرایه‌ای که عناصر پشته را نگهداری کند.
- یک متغییر صحیح که بالای پشته را مشخص کند.



جلسه پنجم:

پیاده‌سازی پشته با آرایه

آنچه در این جلسه می‌خوانید:

۱- طراحی و ساخت کلاس پشته
۲- پیاده‌سازی تابع اصلی و ایجاد کننده پشته
۳- پیاده‌سازی تابع حذف از پشته
۴- پیاده‌سازی تابع درج در پشته
۵- پیاده‌سازی تابع نمایش عنصر بالایی پشته
۶- پیاده‌سازی تابع نمایش همه عناصر پشته

طراحی و ساخت پشته:

برای طراحی و ساخت کلاس پشته, نیاز به توابعی برای پیاده‌سازی عملیات ایجاد و درج و حذف و خالی بودن و نمایش بالای پشته است:

- تابعی برای ایجاد پشته خالی: stack()
- تابعی برای درج عنصر در پشته: push()
- تابعی برای حذف عنصر از پشته: pop()
- تابعی برای نمایش عنصر بالای پشته: top()
- تابعی برای نمایش عناصر پشته: display()

پیاده‌سازی تابع اصلی پشته:

```
void  main()
{
      int  items[5];
      int  mytop=-1;
      int  num,s,i;
      clrscr();
      cout<<“1. Add an item to stack.\n” ;
      cout<<“2. Delete an item from stack.\n” ;
      cout<<“3. Display stack items.\n” ;
      cout<<“4. Display stack top item.\n” ;
      cout<<“5. Exit.\n\n”;
      cout<<“Enter your select:”;
      cin>>s;
      switch  (s)
      {
            case 1:
                  cout<<“Enter a number:”;
                  cin>>num;
                  push(num);
            case 2:
                  pop();
            case 3:
                  display();
            case 4:
                  top();
            case 5:
                  exit(1);
      }
}
```

ایجاد یک آرایه با ۵ عنصر برای نگهداری پشته

تعریف متغییر بالای پشته

ساخت منو برای تعیین عملیات روی پشته

فراخوانی توابع مرتبط

پیاده‌سازی تابع حذف عنصر از پشته:

```
int  pop()
{
        if  (mytop==-1)
                cout<<“Stack is empty. Can’t pop.”;
        else
                mytop--;
        return 0;
}
```

پیاده‌سازی تابع افزودن عنصر به پشته:

```
int  push(int num)
{
        if  (mytop==4)
                cout<<“Stack is full. Can’t push.”
        else
        {
                mytop++;
                item[mytop] = num;
        }
        return 0;
}
```

پیاده‌سازی تابع نمایش عنصر بالای پشته:

```
int  top()
{
        if  (mytop==-1)
                cout<<“Stack is empty. Can’t display.”
        else
                cout<<item[mytop];
        return 0;
}
```

پیاده‌سازی تابع نمایش عناصر پشته:

```
int  display()
{
        if  (mytop==-1)
                cout<<“Stack is empty. Can’t display.”
        else
        {
                for (i=0;i<=mytop;i++)
                        cout<<item[i];
        }
        return 0;
}
```

تمرین:

توابع فوق را مجتمع کرده و به صورت یک برنامه واحد بنویسید.
منو تا هنگامی‌که گزینه exit را نزده‌اید تکرار گردد.
برنامه را تست کرده و کُد آنرا به همراه خروجی تحویل دهید.

جلسه ششم:

انواع عبارات

آنچه در این جلسه می‌خوانید:

۱- انواع عبارات

۲- تبدیل عبارات میانوند به پسوند

۳- تبدیل عبارات پسوند به میانوند

انواع عبارات:

- **عبارات میانوند:**

در این عبارات, عملگر بین عملوندها نوشته می‌شود:

A+B

- **عبارات پیشوند:**

در این عبارات, عملگر قبل از عملوندها نوشته می‌شود:

+AB

- **عبارات پسوند:**

در این عبارات, عملگر بعد از عملوندها نوشته می‌شود:

AB+

انواع عبارات:

عبارات پیشوند و پسوند, برخلاف ظاهرشان, به آسانی مورد استفاده قرار می‌گیرند.

به عنوان مثال تابعی که برای جمع کردن دو مقدار A و B به کار می‌رود, به صورت add(A,B) فراخوانی می‌شود که عملگر add قبل از دو عملوند A و B قرار می‌گیرد.

اغلب کامپایلرها, عبارت میانوند را به صورت عبارت پسوند در می‌آورند و سپس دستورات زبان ماشین را برای ارزیابی آنها تولید می‌کنند (به علت ارزیابی آسان آنها).

عبارات میانوند می‌توانند حاوی پرانتز باشند تا تقدم عملگرها را مشخص کنند, ولی عبارات پسوندی فاقد پرانتز هستند.

تبدیل عبارات میانوند به پسوند:

برای تبدیل یک عبارت میانوند به پسوند, ابتدا قسمت‌های با تقدم بالا به عبارت پسوند تبدیل می‌شود و عبارت حاصل به عنوان یک عملوند محسوب شده و با سایر عملوندها به عبارت پسوند تبدیل می‌شود.

مثال:

عبارت میانوند A+B*C را به عبارت پسوندی تبدیل کنید.

A+B*C	عبارت میانوند
A+(B*C)	مرحله ۱
A+(BC*)	مرحله ۲
A(BC*)+	مرحله ۳
ABC*+	مرحله ۴ (عبارت پسوند نهایی)

تبدیل عبارات میانوند به پسوند:

مثال:

عبارت میانوند (A+B)*C را به عبارت پسوندی تبدیل کنید.

عبارت میانوند (A+B)*C

مرحله ۱ (AB+)*C

مرحله ۲ (AB+)C*

مرحله ۳ (عبارت پسوند نهایی) AB+C*

تبدیل عبارات میانوند به پسوند:

مثال:

عبارت میانوند A+B-C را به عبارت پسوندی تبدیل کنید.

عبارت میانوند A+B-C

مرحله ۱ (AB+)-C

مرحله ۲ (AB+)C-

مرحله ۳ (عبارت پسوند نهایی) AB+C-

تبدیل عبارات میانوند به پسوند:

مثال:

عبارت میانوند (A+B)*(C-D) را به عبارت پسوندی تبدیل کنید.

عبارت میانوند	(A+B)*(C-D)
مرحله ۱	(AB+)*(C-D)
مرحله ۲	(AB+)*(CD-)
مرحله ۳	(AB+)(CD-)*
مرحله ۴ (عبارت پسوند نهایی)	AB+CD-*

تبدیل عبارات میانوند به پسوند:

مثال: عبارت میانوند A^B*C-D+E/F/(G+H) را به عبارت پسوندی تبدیل کنید.

مرحله ۱	A^B*C-D+E/F/(GH+)
مرحله ۲	(AB^)*C-D+E/F/(GH+)
مرحله ۳	((AB^)C*)-D+E/F/(GH+)
مرحله ۴	((AB^)C*)-D+(EF/)/(GH+)
مرحله ۵	((AB^)C*)-D+(EF/)(GH+)/
مرحله ۶	(((AB^)C*)D-)+(EF/)(GH+)/
مرحله ۷	((AB^)C*)D-)(EF/)(GH+)/+
مرحله ۴ (عبارت پسوند نهایی)	AB^C*D-EF/GH+/+

تبدیل عبارات میانوند به پسوند:

مثال: عبارت میانوند ((A+B)*C-(D-E))^(F+G) را به عبارت پسوند تبدیل کنید.

مرحله ۱ ((AB+)*C-(D-E))^(F+G)

مرحله ۲ ((AB+)*C-(DE-))^(F+G)

مرحله ۳ ((AB+)*C-(DE-))^(FG+)

مرحله ۴ (((AB+)C*)-(DE-))^(FG+)

مرحله ۵ ((AB+)C*)(DE-)-)^(FG+)

مرحله ۶ ((AB+C*)(DE-)-)(FG+)^

مرحله ۷ (عبارت پسوندی نهایی) AB+C*DE--FG+^

تبدیل عبارات میانوند به پسوند:

مثال: عبارت میانوند A-B/(C*D^E) را به عبارت پسوند تبدیل کنید.

مرحله ۱ A-B/(C*(DE^))

مرحله ۲ A-B/(C(DE^)*)

مرحله ۳ A-(B(C(DE^)*)/)

مرحله ۴ A(B(C(DE^)*)/)-

مرحله ۵ (عبارت پسوندی نهایی) ABCDE^*/-

تمرین:

عبارات میانوندی زیر را به پسوندی تبدیل کنید.

1. (A-(B+C)) / (E+F)
2. a*b+c-d
3. a+b/c+d
4. (((a-b)-c)-d)-e
5. (A+B) / (C+D)
6. (A+B)/C+D
7. (a-b)*(c-(d+e))

تبدیل عبارات پسوند به میانوند:

برای تبدیل یک عبارت پسوند به میانوند, از سمت چپ شروع کرده و به هر عملگر که رسیده شود, آنرا بین دو عملوند قبلش قرار داده و تا پایان عبارت این عمل تکرار می‌شود.

عبارت پسوند ABC*+ را به عبارت میانوندی تبدیل کنید.

ABC*+ عبارت پسوند

A(B*C)+ مرحله ۱

A+(B*C) مرحله ۲

A+B*C مرحله ۳ (عبارت میانوند نهایی)

تبدیل عبارات پسوند به میانوند:

مثال:

عبارت پسوند AB+C* را به عبارت میانوندی تبدیل کنید.

AB+C* عبارت پسوند

(A+B)C* مرحله ۱

(A+B)*C مرحله ۲ (عبارت میانوند نهایی)

تمرین:

عبارات پسوندی زیر را به میانوندی تبدیل کنید.

1. ABC+-EF+/
2. ab*c*d-
3. abc/+d+
4. ab-c-d-e-
5. AB+CD+/
6. AB+C/D+
7. ab-cde+-*

جلسه هفتم:

صف‌ها

آنچه در این جلسه می‌خوانید:

۱- مقدمه‌ای بر صف

۲- روشهای پیاده‌سازی صف

۳- طراحی و ساخت صف با آرایه

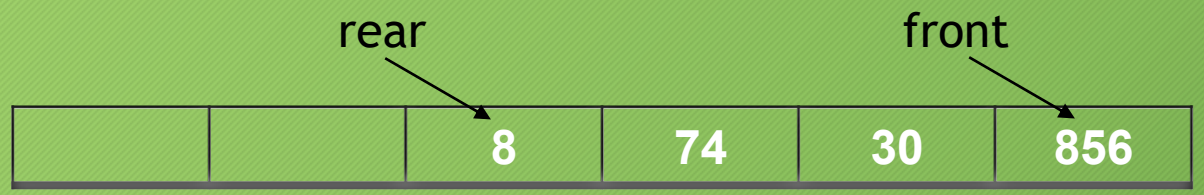
۴- پیاده‌سازی صف

مقدمه‌ای بر صف:

صف, مجموعه‌ای از عناصر مرتب است که هر عنصر از یک طرف به‌نام جلوی صف از آن حذف می‌شود و از طرف دیگر به‌نام انتهای صف در صف قرار می‌گیرد. به صف ساختمان داده FIFO (First In First Out) (خروج ترتیب ورود) نیز می‌گویند.

عملیات اصلی:

- ایجاد صف
- تست خالی‌بودن صف
- افزودن عنصر به آخر صف
- بازیابی عنصری از جلوی صف
- حذف عنصری از جلوی صف



روشهای پیاده‌سازی صف:

همانند پشته، صف را نیز می‌توان به دو صورت خطی یا پیوندی نمایش داد.

در روش خطی، برای نمایش صف می‌توان از آرایه استفاده کرد.

در روش پیوندی، برای نمایش صف می‌توان از لیست پیوندی استفاده کرد.

طراحی و ساخت صف با آرایه:

برای طراحی و ساخت صف, نیاز به توابعی برای پیاده‌سازی عملیات ایجاد و درج و حذف و خالی بودن و نمایش بالای صف است:

- تابعی برای ایجاد صف خالی: queue()
- تابعی برای درج عنصر در صف: add()
- تابعی برای حذف عنصر از صف: remove()
- تابعی برای نمایش عنصر جلوی صف: retrive()
- تابعی برای نمایش عناصر صف: display()

پیاده‌سازی تابع اصلی صف:

```
void  main()
{
      int  items[5];
      int  front  = 0;
      int  rear = -1;
      int  num;
      clrscr();
      cout<<“1. Add an item to queue.\n” ;
      cout<<“2. Delete an item from queue.\n” ;
      cout<<“3. Display queue items.\n” ;
      cout<<“4. Display queue front item.\n” ;
      cout<<“5. Exit.\n\n”;
      cout<<“Enter your select:”;
      cin>>s;
      switch  (s)
      {
            case 1:
                  cout<<“Enter a number:”;
                  cin>>num;
                  add(num);
            case 2:
                  remove();
            case 3:
                  display();
            case 4:
                  retrive();
            case 5:
                  exit(1);
      }
}
```

ایجاد یک آرایه با ۵ عنصر برای نگهداری صف

تعریف متغییر ابتدای صف

تعریف متغییر انتهای صف

ساخت منو برای تعیین عملیات روی صف

فراخوانی توابع مرتبط

پیاده‌سازی تابع حذف عنصر از صف:

```
int  remove()
{
        if  (rear<front)
                cout<<“Queue is empty. Can’t remove.”;
        else
        {
                for(i=0;i<rear-1;i++)
                        item[i] = item[i+1];
                rear--;
        }
        return 0;
}
```

پیاده‌سازی تابع افزودن عنصر به صف:

```
int  add(int num)
{
        if  (rear==4)
                cout<<“Queue is full. Can’t add.”
        else
        {
                rear++;
                item[rear] = num;
        }
        return 0;
}
```

پیاده‌سازی تابع نمایش عنصر ابتدای صف:

```
int  retrive()
{
        if  (rear<front)
                cout<<“Queue is empty. Can’t display.”
        else
                cout<<item[front];
        return 0;
}
```

پیاده‌سازی تابع نمایش عناصر صف:

```
int  display()
{
        if  (rear<front)
                cout<<“Queue is empty. Can’t display.”
        else
        {
                for (i=0;i<=rear;i++)
                        cout<<item[i];
        }
        return 0;
}
```

تمرین:

توابع فوق را مجتمع کرده و به صورت یک برنامه واحد بنویسید.
منو تا هنگامی‌که گزینه exit را نزده‌اید تکرار گردد.
برنامه را تست کرده و کُد آنرا به همراه خروجی تحویل دهید.

جلسه هشتم:

لیست‌های پیوندی

آنچه در این جلسه می‌خوانید:

۱- مقدمه‌ای بر لیست پیوندی

۲- ساختار لیست پیوندی یک‌طرفه

۳- روشهای پیاده‌سازی لیست پیوندی

۴- تعریف گره لیست پیوندی با استفاده از اشاره‌گرها

۵- ایجاد, دسترسی, حذف و پیوند گره

۶- تعریف اشاره‌گرها

۷- پیاده‌سازی توابع لیست پیوندی

مقدمه‌ای بر لیست پیوندی:

لیست پیوندی, ساختمان داده است که عناصر آن در زمان اجرا اضافه یا حذف می‌شوند.
لیست پیوندی می‌تواند یک پیوندی یا دو پیوندی یا حلقوی باشد.

عملیات اصلی:

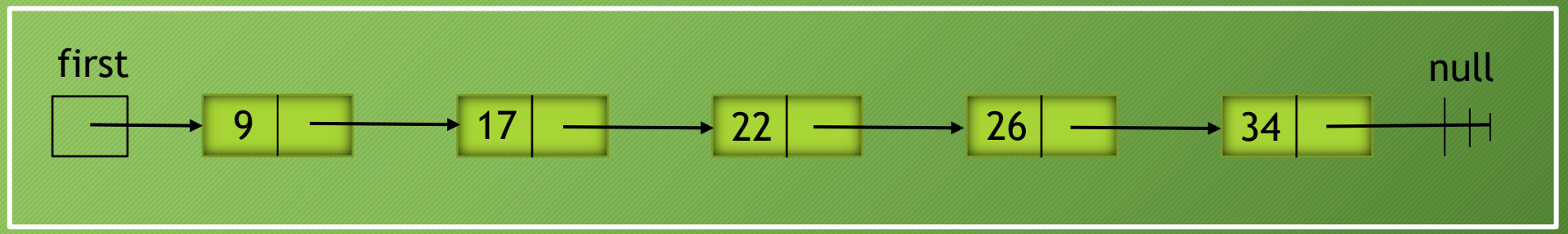
- ایجاد لیست
- تست خالی بودن
- پیمایش لیست
- درج
- حذف

ساختار لیست پیوندی یک‌طرفه:

لیست پیوندی همانند پشته و صف از مجموعه‌ای از عناصر تشکیل شده است. به هر یک از عناصر لیست یک **گره** (node) گفته می‌شود. هر گره شامل دو فیلد است: فیلد اطلاعات و فیلد آدرس گره بعدی.

فیلد اطلاعات داده‌هایی را ذخیره می‌کند که باید در لیست پیوندی قرار گیرند.

فیلد آدرس حاوی آدرس گره بعدی است. فیلد آدرس را اشاره‌گر نیز می‌گویند.



نکته:

چون هر گره لیست پیوندی، آدرس گره بعدی را دارد، لازم نیست عناصر لیست در کنار هم باشند.

روشهای پیاده‌سازی لیست پیوندی:

دو روش برای پیاده‌سازی لیست پیوندی وجود دارد:

- پیاده‌سازی با استفاده از آرایه.
- پیاده‌سازی با استفاده از اشاره‌گرها.

تعریف گره لیست پیوندی با استفاده از اشاره‌گرها:

گره لیست پیوندی را می‌توان با استفاده از دستور struct یا پیاده‌سازی کلاس ایجاد کرد.
هر گره لیست پیوندی یک ساختار است.

- مثال:

برای تعریف گره‌ای از لیست پیوندی که شامل نام و شماره دانشجویی است, به صورت زیر عمل می‌شود:

```
struct  student
{
        int  stID;
        string  name;
        student  *next;
};
```

ایجاد, دسترسی, حذف و پیوند گره:

```
node = new  student;
```


```
node->stID = 1254;
node->name = 'ghh';
```


```
delete   node;
```

```
d = new student;
d->stID = 5465;
d->name = 'kkk';
d->next = NULL;
node->next = d;
```

تعریف اشاره‌گرها:

اشاره‌گرهایی که به عنوان بخشی از ساختمان گره لیست می‌باشند, اشاره‌گر داخلی نام دارند. برای ایجاد لیست پیوندی به اشاره‌گرهای دیگری نیاز است که به آنها اشاره‌گرهای خارجی می‌گویند. اشاره‌گرهای خارجی از نوع گره لیست بوده و به گره‌ها اشاره می‌کنند.

یادآوری:
اشاره‌گر یک آدرس حافظه است

- **مثال:**

دستور زیر دو اشاره‌گر first و last را از نوع node تعریف می‌کند.

```
struct  node  *first, *last;
```

```
class  student
{
      friend  class  linkedList;
      int  stID;
      string  name;
      student  *next;
};

class  linkedList
{
      student  *first;
      student  *last;
      public:
            linkedList();
            void  add();
            void  remove();
            void display();
};

linkedlist::linkedlist()
{
      first = last = NULL;
}
```

```
void  main()
{
      linkedlist  list;
      int  s;
      do
      {
            clrscr();
            cout<<“1. Add an item to list.\n” ;
            cout<<“2. Delete an item from list.\n” ;
            cout<<“3. Display list items.\n” ;
            cout<<“4. Exit.\n\n”;
            cout<<“Enter your select(1-4): ”;
            cin>>s;
            switch  (s)
            {
                  case 1:
                        list.add();
                        break;
                  case 2:
                        list.remove();
                        break;
                  case 3:
                        list.display();
                        break;
                  case 4:
                        exit;
            }
      }while (s != 4);
}
```

پیاده‌سازی لیست پیوندی و منوی اصلی

پیاده‌سازی عمل
اضافه کردن به
لیست پیوندی

```
void  linkedList :: add()
{
     clrscr();
     student  *node;
     node = new  student;
     node->next = NULL;
     if (first == NULL)
          first = last = node;
     else
     {
          last->next = node;
          last = node;
     }
     cout<<“Enter student number: “;
     cin>>last->stID;

     cout<<“Enter student name: “;
     cin>>last->name;
}
```

پیاده‌سازی عمل
حذف از لیست
پیوندی

```
void  linkedList :: remove()
{
        clrscr();
        student  *p,*h;
        int  n;
        if (!first)
        {
                cout<<“List is empty…”;
                getch();
                return;
        }
        cout<<“Enter student number for delete: “;
        cin>>n;
        h = p = first;
        while ( h != NULL)
        {
                if ( h->stID != n )
                {
                        p = h;
                        h = h->next;
                        continue;
                }
                else
                {
                        if (h == first)
                        {
                                first = h->next;
                                delete  h;
                                delete  p;
                                cout<<“Student deleted…”;
                                break;
                        }
                        else
                        {
                                if (h == last)
                                        last = p;
                                p->next = h->next;
                                delete  h;
                                cout<<“Student deleted…”;
                                break;
                        }
                }
        }
}
```

پیاده‌سازی نمایش
عناصر لیست
پیوندی

```
void  linkedList :: display()
{
    student  *helpNode;
    if (first == NULL)
    {
        cout>>”List is empty”;
        getch();
        return;
    }
    helpNode = first;
    do
    {
        cout>>helpNode->stID;
        cout>>”--”;
        cout>>helpNode->name;
        cout>>”\n”
        helpNode = helpNode->next;
    }while(helpNode != NULL);
    getch();
}
```

تمرین:

توابع فوق را مجتمع کرده و به صورت یک برنامه واحد بنویسید.
منو تا هنگامی‌که گزینه exit را نزده‌اید تکرار گردد.
برنامه را تست کرده و کُد آنرا به همراه خروجی تحویل دهید.

جلسه نهم:

درخت‌ها

آنچه در این جلسه می‌خوانید:

۱- مقدمه‌ای بر درخت‌ها
۲- اصطلاحات مربوط به درختها
۳- درخت‌های دودویی
۴- درخت دودویی پُر
۵- درخت دودویی کامل
۶- پیاده‌سازی درخت دودویی
۷- پیاده‌سازی درخت دودویی با استفاده از آرایه

مقدمه‌ای بر درخت‌ها:

درخت یکی از ساختمان‌داده‌های مهم است که کاربردهای فراوانی دارد.

درخت از مجموعه‌ای از عناصر به نام گره تشکیل شده است که یکی از گره‌ها ریشه نام دارد. برخلاف درخت‌های طبیعی که ریشه آنها در پایین و برگ‌ها در بالا قرار دارند, در درخت‌های کامپیوتری, ریشه در بالا و برگ‌ها در پایین قرار دارند.

هر گره شامل فیلدی برای داده‌ها است و تعدادی پیوند دارد که به گره‌های دیگر وصل می‌شود. این پیوندها را انشعاب یا اتصال نیز می‌گویند. گره‌ای که هیچ انشعابی از آن خارج نشود, برگ نام دارد.

درخت‌ها به طور کلی بر دو دسته‌اند:

- درخت‌های عمومی: در این نوع از درخت‌ها هر گره می‌تواند هر تعداد پیوند داشته باشد.
- درختهای دودویی: در این نوع از درخت‌ها از هر گره حداکثر دو پیوند خارج می‌شود.

اصطلاحات مربوط به درخت‌ها:

- **گره:**

عناصر درخت را گره می‌گویند. هر گره دارای یک مسیر منحصر به فرد است که آنرا به ریشه درخت وصل می‌کند.

- **مسیر:**

دنباله‌ای از گره‌ها همجوار است.

- **طول مسیر:**

تعداد اتصال‌های همجوار می‌باشد که یکی کمتر از تعداد گره‌های موجود در آن مسیر است.

- **عمق گره:**

طول مسیر یک گره تا ریشه است.

- **سطح درخت:**

هر گره موجود در درخت دودویی دارای سطح است. سطح گره ریشه صفر در نظر گرفته می‌شود.سطوح بقیه گره‌ها یک واحد بیشتر از گره بالایی است.

اصطلاحات مربوط به درخت‌ها:

- عمق درخت:

عمق درخت که نام دیگر آن ارتفاع درخت است, بزرگترین سطح برگ‌های درخت (طول بزرگترین مسیر از برگ‌ها به ریشه) است.

- درخت یگانه:

درختی که فقط دارای گره ریشه است که ارتفاع یا عمق آن صفر است.

- درخت خالی:

درختی که فاقد هرگونه گره‌ای باشد و عمق آن ۱- است.

- اجداد گره:

گره‌های بالاتر مسیر یک گره تا ریشه را اجداد گره می‌گویند. ریشه جدّ همه گره‌ها است و اجدادی ندارد.

- نسل‌های گره:

اگر گره y جدّ گره x باشد, x نسل y است.

اصطلاحات مربوط به درخت‌ها:

- والد (پدر) گره:

جدّ بلافصل یک گره را والد آن گره می‌گویند.

- فرزندان گره:

نسل‌های بلافصل یک گره را فرزندان آن گره می‌گویند.

- گره‌های برگ:

گره‌هایی که هیچ فرزندی ندارند, برگ نامیده می‌شوند.

- گره‌های داخلی:

گره‌های غیر برگ را گره‌های داخلی می‌نامند.

- اندازه درخت:

تعداد گره‌های موجود در درخت را اندازه درخت می‌گویند.

اصطلاحات مربوط به درخت‌ها:

- زیر درخت:

برای هر گره y موجود در درخت, مجموعه‌ای گره‌ها که حاوی y و تمام نسل‌های آنها هستند, زیر درختی را تشکیل می‌دهند که ریشه آن گره y است.

- طول مسیر درخت:

مجموع طول‌های تمام مسیرها از ریشه است.

- درجه گره:

درجه گره برابر با تعداد فرزندان آن است.

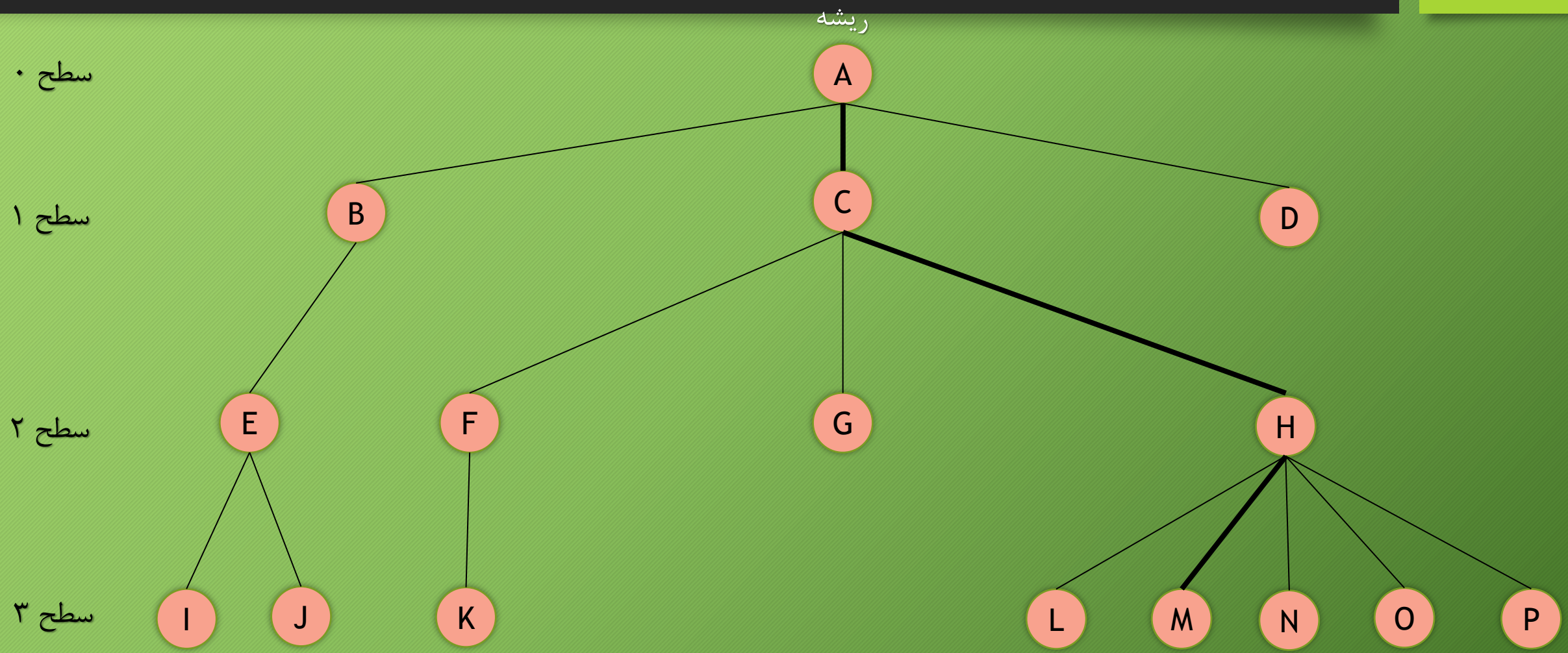
- درخت پُر:

درختی است که درجه تمام گره‌های داخلی آن یکسان باشد و تمام برگ‌های آن در یک سطح قرار داشته باشند

- تعداد گره‌های درخت پُر:

تعداد گره‌های درخت پُر با درجه d و ارتفاع h برابر است با $\frac{d^{h+1}-1}{d-1}$

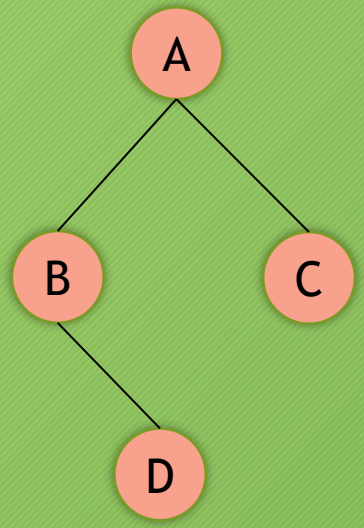
مثال:

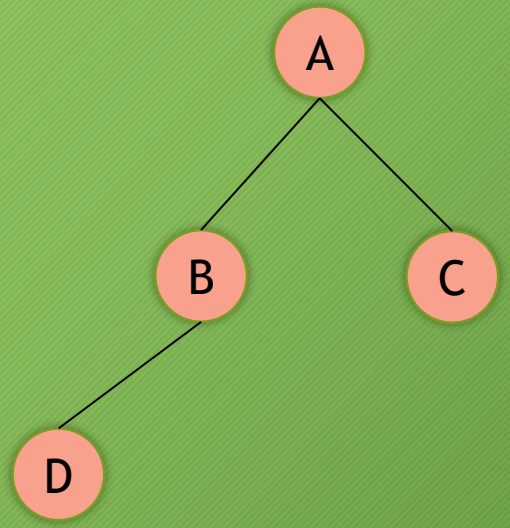


درخت‌های دودیی:

درخت‌های دودویی, درخت‌هایی هستند که هر گره آن حداکثر دو فرزند دارد. این درخت‌ها کاربردهای فراوانی در کامپیوتر دارند.

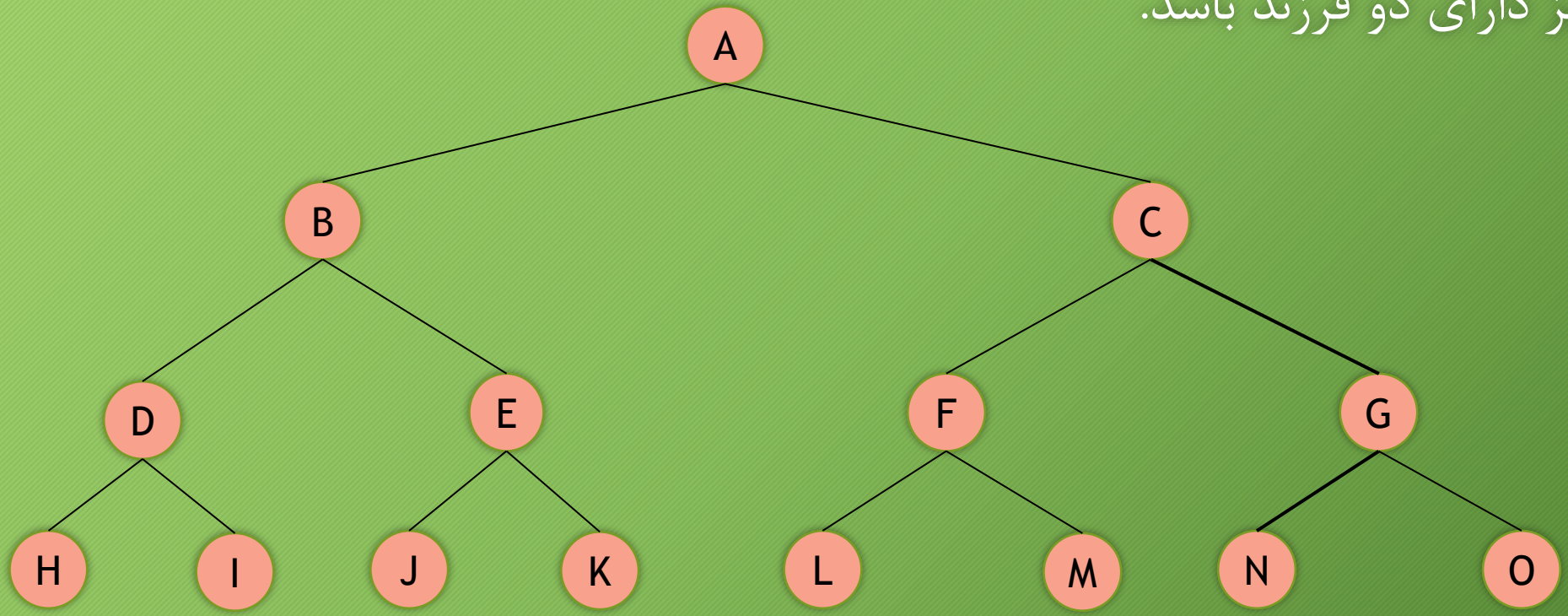
درخت‌های دودویی را می‌توان درختی با درجه ۲ در نظر گرفت, اما ترتیب فرزندان در درخت دودویی مهم است.





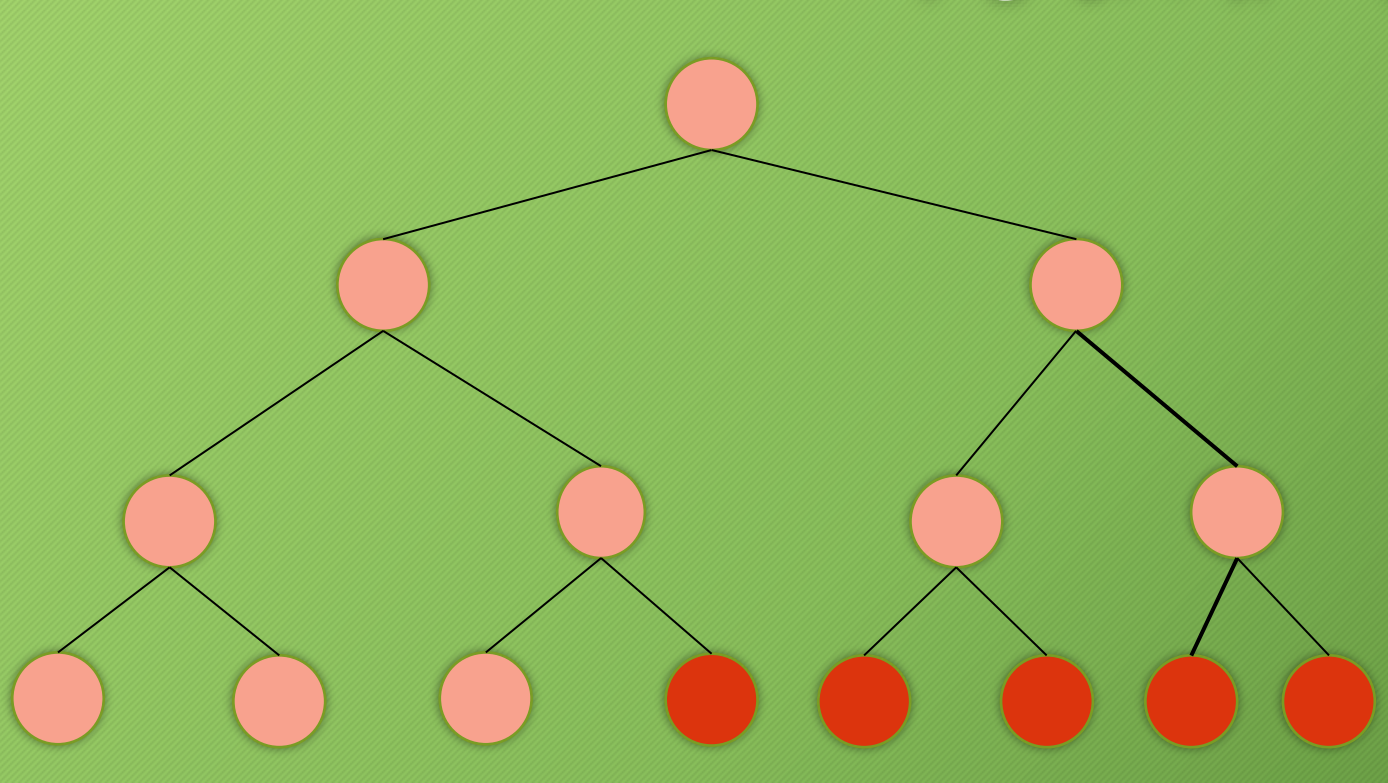
درخت دودیی پُر:

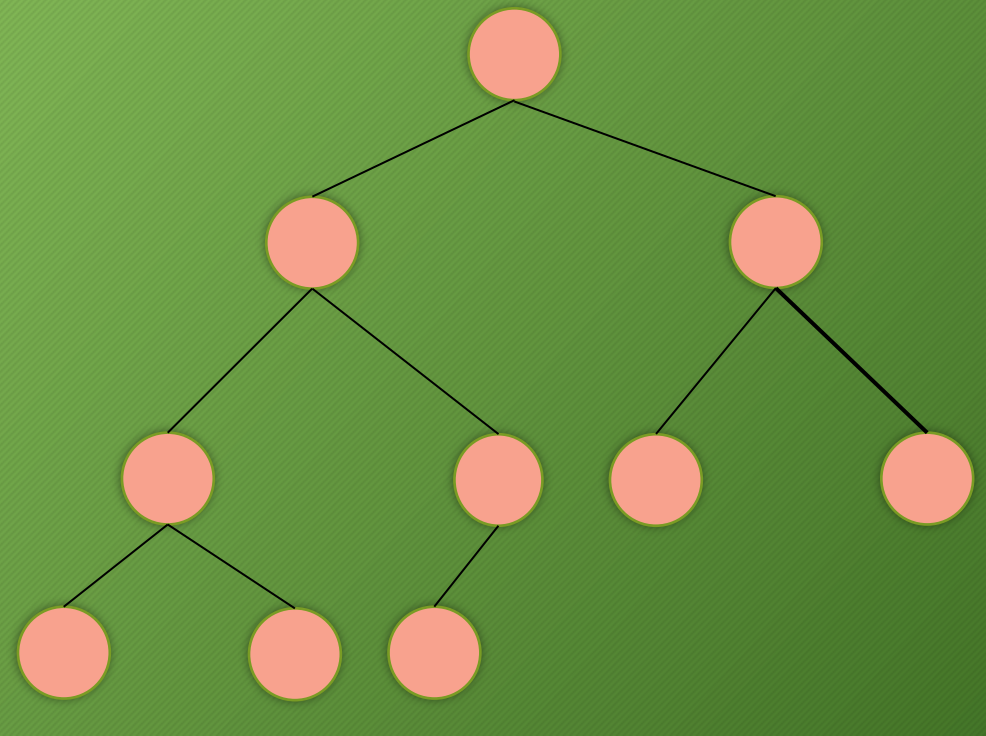
درخت دودویی پُر, درختی است که تمام برگ‌های آن در یک سطح باشند و هر گره داخلی نیز دارای دو فرزند باشد.



درخت دودیی کامل:

درخت دودویی کامل، درختی است که یا پُر است یا با افزودن گره‌های پشت سر هم به سمت راستِ سطح پایینی، به درخت پُر تبدیل می‌شود.





پیاده‌سازی درخت دودیی:

درخت دودویی را می‌توان به دو روش نمایش داد و پیاده‌سازی کرد:

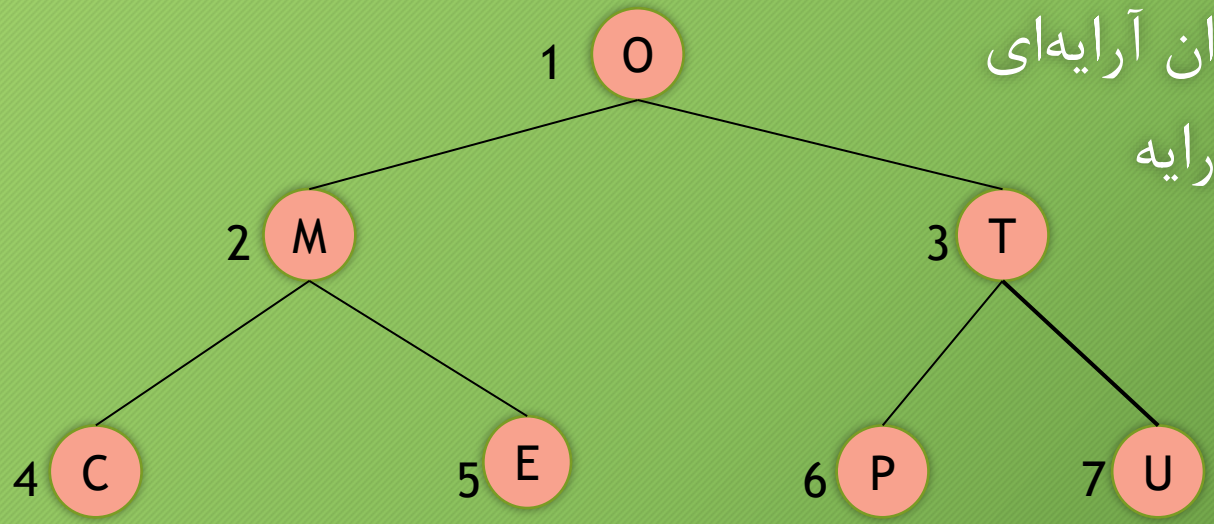
۱- با استفاده از آرایه.
۲- با استفاده از اشاره‌گرها.

پیاده‌سازی درخت دودیی با استفاده از آرایه:

ایدهٔ درخت دودویی پر یا کامل آنرا برای پیاده‌سازی به کمک آرایه مناسب می‌کند.

گره‌های یک درخت دودودیی از ریشه شروع و با حرکت از چپ به راست شماره‌گذاری می‌شوند. درخت‌های کامل را نیز شماره‌گذاری کرد.

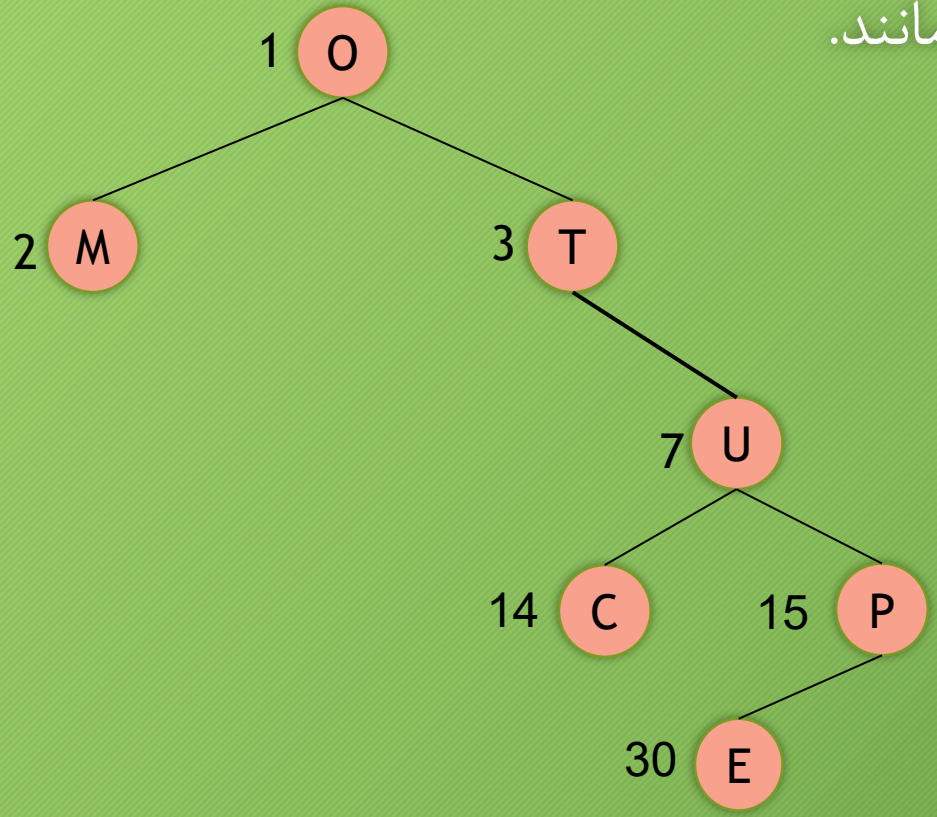
با توجه به شماره‌گذاری درخت‌های پر و کامل, اگر این درختها دارای n گره باشند, می‌توان آرایه‌ای به طول n تعریف کرد و در هر عنصر آرایه یک گره درخت را قرار داد.



1	2	3	4	5	6	7
O	M	T	C	E	P	U

پیاده‌سازی درخت دودیی با استفاده از آرایه:

اگر درخت پر یا کامل نباشد, نمایش درخت دودویی با استفاده از آرایه چندان جالب نخواهد بود, زیرا عناصر زیادی از آرایه خالی می‌مانند.



جلسه دهم:

پیاده‌سازی درخت دودویی با اشاره‌گر

آنچه در این جلسه می‌خوانید:

۱- پیاده‌سازی درخت دودویی با اشاره‌گر
۲- پیمایش درخت دودویی
۳- روش پیمایش preorder
۴- روش پیمایش postorder
۵- روش پیمایش inorder
۶- درخت عبارت دودویی

پیاده‌سازی درخت دودویی با اشاره‌گر:

در پیاده‌سازی درخت‌ها توسط آرایه, مشاهده شد آرایه برای نمایش درخت‌های دودویی کامل مناسب است, ولی برای نمایش سایر درخت‌های دودویی با اتلاف حافظه مواجه است.

علاوه بر این, نمایش درخت‌ها به صورت آرایه, مشکلات مربوط به آرایه را دارد. به عنوان مثال, درج و حذف گره‌ها مستلزم جابجایی عناصر آرایه است و شماره سطح نیز تغییر می‌کند که این مشکلات از طریق پیاده‌سازی درخت‌ها با استفاده از اشاره‌گرها حل می‌شود.

برای استفاده از اشاره‌گرها, هر گره در درخت دارای فیلد داده, فیلد اشاره‌گر به چپ (برای نمایش فرزند چپ) و فیلد اشاره‌گر به راست (برای نمایش فرزند راست) است.

پیمایش درخت‌های دودویی:

منظور از پیمایش درخت‌های دودویی, حرکت در سراسر درخت دودویی و ملاقات همهٔ گره‌های آن است.

پیمایش درخت, موجب دستیابی به تمام گره‌های درخت می‌شود.

دستیابی به گره ممکن است برای اهداف خاصی صورت گیرد, مانند چاپ محتویات گره یا انجام هر نوع محاسبات بر روی آنها.

- سه روش متداول برای پیمایش درخت‌ها وجود دارد:

۱- روش پیشوندی یا preorder

۲- روش پسوندی یا postorder

۳- روش میانوندی یا inorder

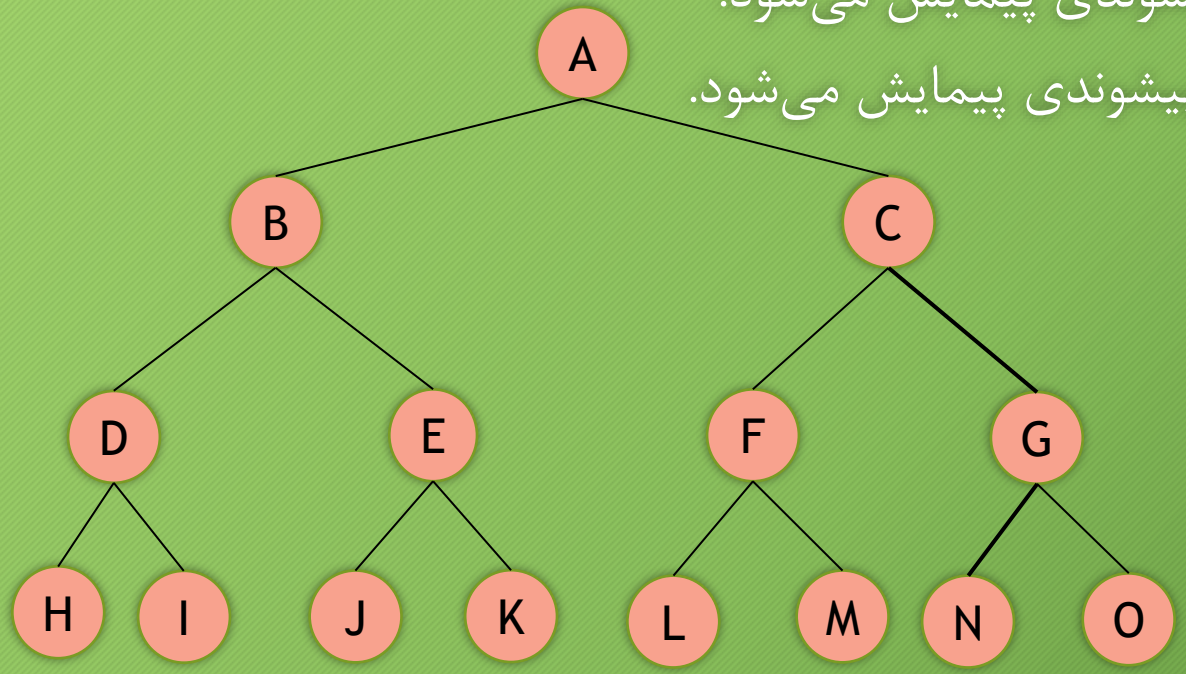
روش پیمایش preorder:

در روش پیمایش پیشوندی, مراحل زیر انجام می‌شود:

۱- ریشه درخت ملاقات می‌شود.

۲- اگر زیر درخت چپ خالی نباشد, به روش پیشوندی پیمایش می‌شود.

۳- اگر زیر درخت راست خالی نباشد, به روش پیشوندی پیمایش می‌شود.



ABDHIEJKCFLMGNO

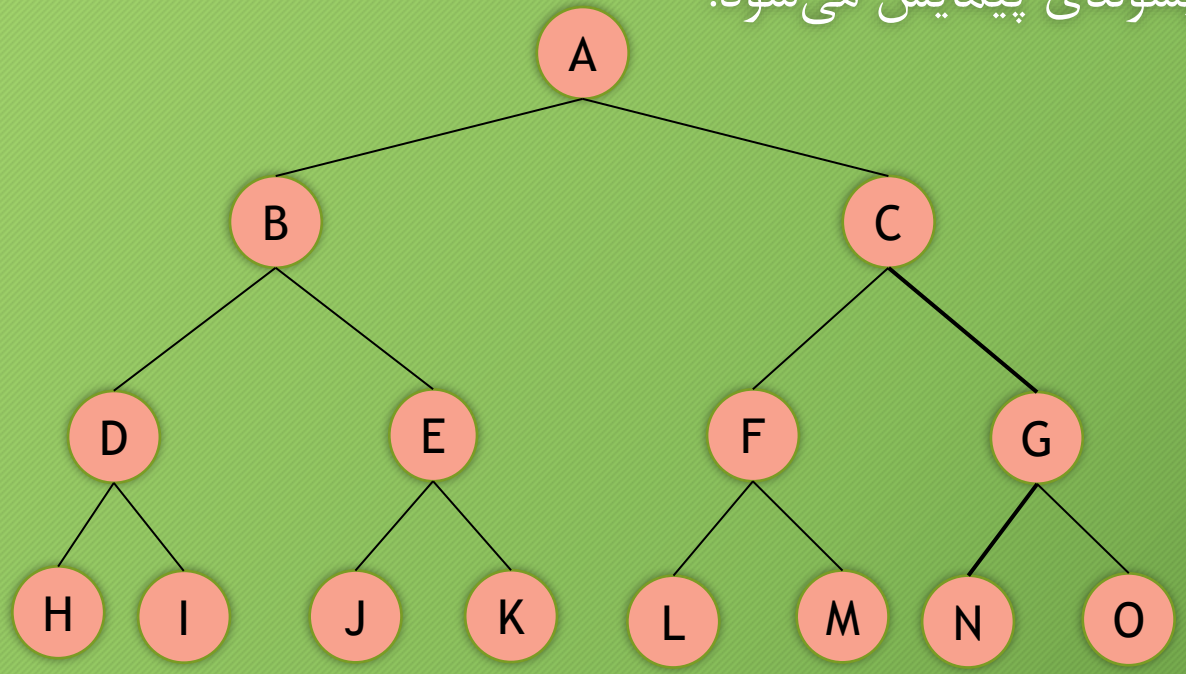
روش پیمایش postorder:

در روش پیمایش پسوندی, مراحل زیر انجام می‌شود:

۱- اگر زیر درخت چپ خالی نباشد, به روش پسوندی پیمایش می‌شود.

۲- اگر زیر درخت راست خالی نباشد, به روش پسوندی پیمایش می‌شود.

۳- ریشه درخت ملاقات می‌شود.



HIDJKEBLMFNOGCA

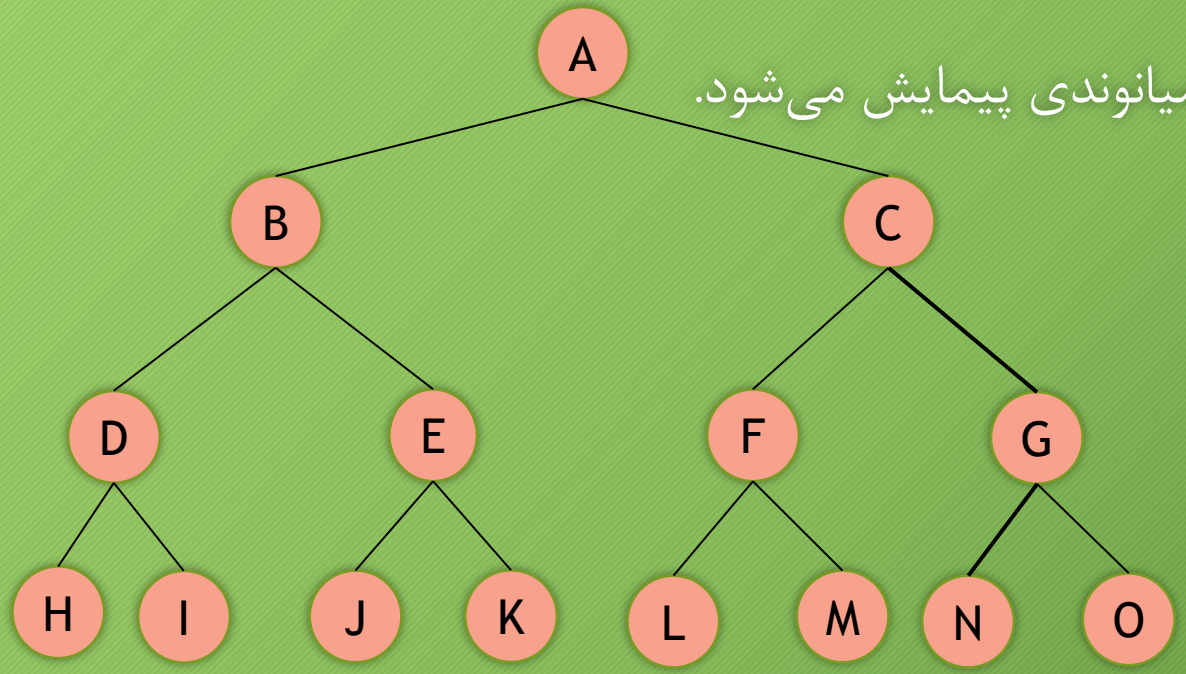
روش پیمایش inorder:

در روش پیمایش میانوندی, مراحل زیر انجام می‌شود:

۱- اگر زیر درخت چپ خالی نباشد, به روش میانوندی پیمایش می‌شود.

۲- ریشه درخت ملاقات می‌شود.

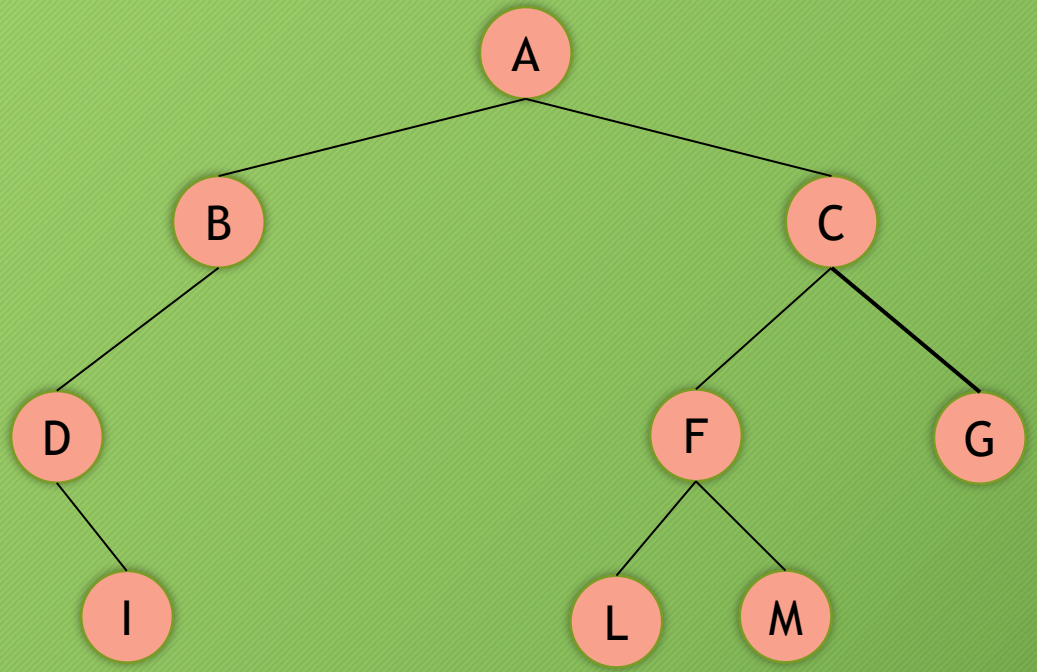
۳- اگر زیر درخت راست خالی نباشد, به روش میانوندی پیمایش می‌شود.

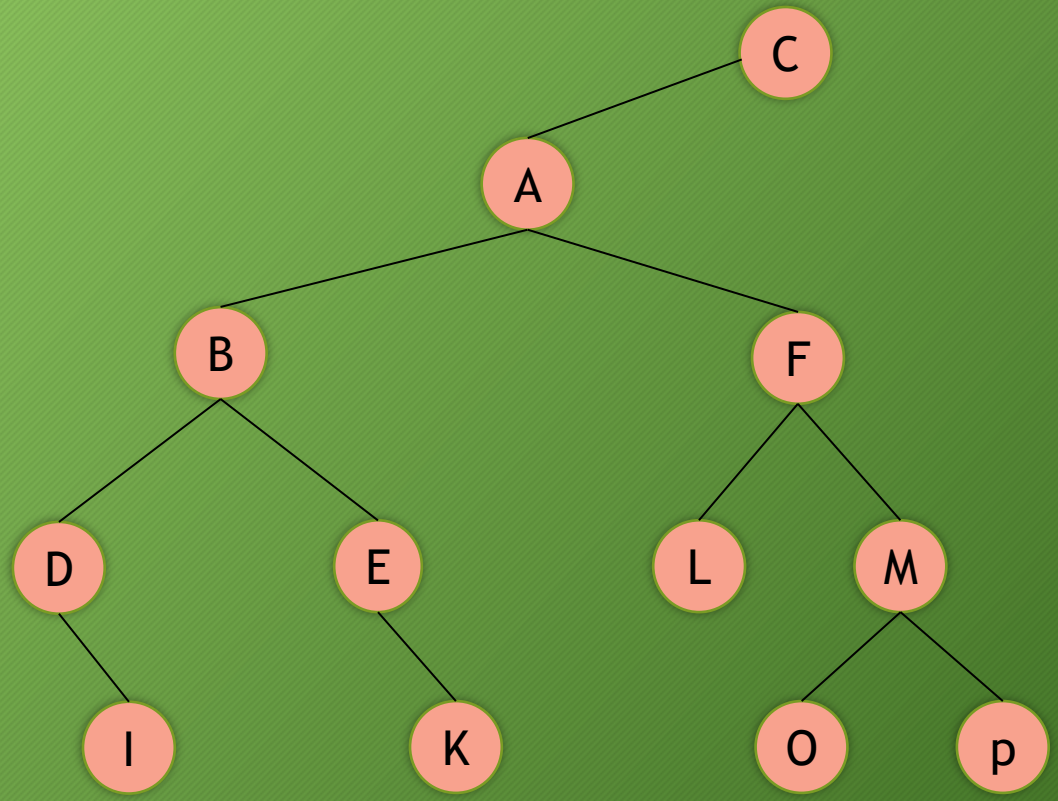


HDIBJEKALFMCNGO

تمرین:

درخت‌های دودویی زیر را به سه روش پیمایش کنید.





پیاده‌سازی
درخت

```
class  treeNode
{
     friend  class  tree;
     private:
          treeNode  *left;
          int  info;
          treeNode  *right;
};

class  tree
{
     public:
          tree();
          void  add();
          void  remove();
          void display(treeNode  *node);
     private:
          treeNode  *root;
};

tree::tree()
{
     left = right = NULL;
}
```

```
void  main()
{
     tree  t;
     int  s;
     do
     {
          clrscr();
          cout<<“1. Insert an item to tree.\n” ;
          cout<<“2. Delete an item from tree.\n” ;
          cout<<“3. Display tree items.\n” ;
          cout<<“4. Exit.\n\n”;
          cout<<“Enter your select(1-4): ”;
          cin>>s;
          switch  (s)
          {
               case 1:
                    t.insert();
                    break;
               case 2:
                    t.remove();
                    break;
               case 3:
                    t.display(treeNode  *root);
                    break;
               case 4:
                    exit;
          }
     }while (s != 4);
}
```

```
void  tree :: display(treeNode  *node)
{
    if (node)
    {
        cout<<node->info<<“ “;
        display(node->left);
        display(node->right);
    }
}
```

پیمایش درخت دودویی
به روش پیشوندی

```
void  tree :: display(treeNode  *node)
{
    if (node)
    {
        display(node->left);
        display(node->right);
        cout<<node->info<<“ “;
    }
}
```

پیمایش درخت دودویی
به روش پسوندی

پیمایش درخت دودویی
به روش میانوندی

```
void  tree :: display(treeNode  *node)
{
    if (node)
    {
        display(node->left);
        cout<<node->info<<“ “;
        display(node->right);
    }
}
```

درخت عبارت دودویی:

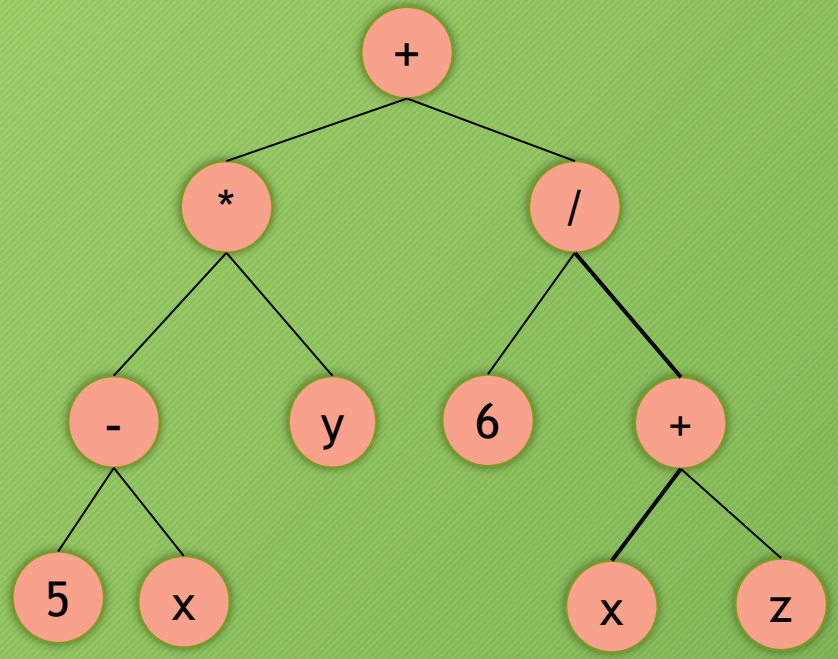
درخت عبارت دودویی نوعی درخت است که برای نمایش عبارات ریاضی دودویی به کار می‌رود. عبارات دودویی آنهایی هستند که هر عملگر آن دارای دو عملوند است. یعنی هر عبارت دودویی را می‌توان توسط یک درخت دودویی منحصر به فرد نمایش داد که ساختار آن به وسیله تقدم عملگرها در عبارت تعیین می‌شود.

این درخت را درخت عبارت دودویی می‌گویند.

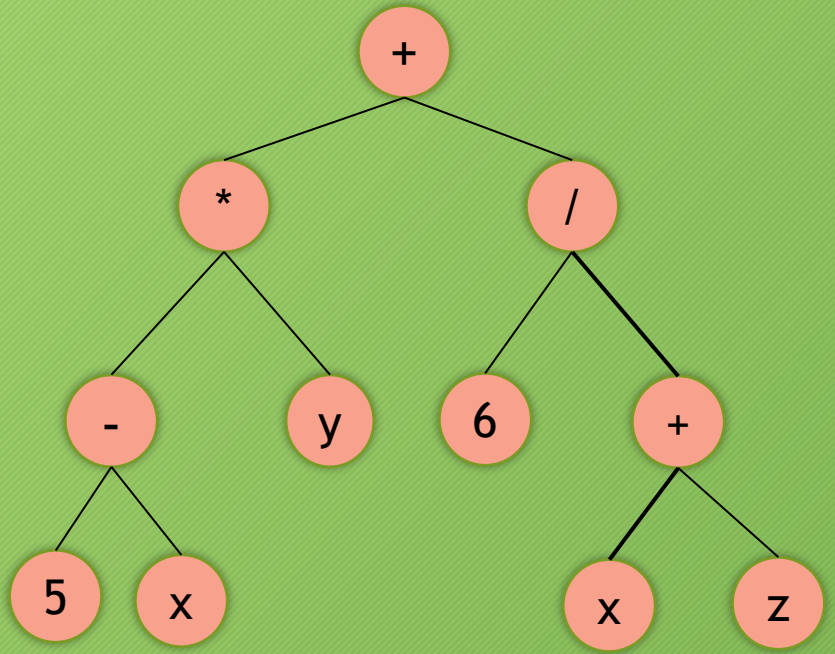
- **نکته:**

در درخت عبارت ,عملوندها در برگ‌ها

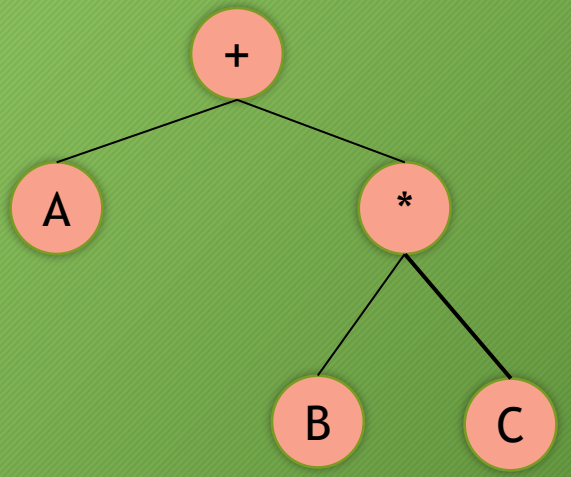
و عملگرها در گره‌های غیر برگ قرار دارند.



درخت عبارت دودویی:



Preorder: +*-5xy/6+xz
Postorder: 5x-y*6xz+/+



Preorder: +A*BC
Postorder: ABC*+

جلسه یازدهم:

درخت‌های جستجو

آنچه در این جلسه می‌خوانید:

۱- درخت جستجو

۲- درخت‌های جستجوی دودویی

۳- درج گره در BST

۴- جستجوی گره در BST

۵- حذف گره از BST

درخت جستجو:

ساختارهای درختی بدلیل کارآمدی سازمان آنها برای دستیابی به داده‌ها, برای ذخیرهٔ داده‌ها مورد استفاده قرار می‌گیرند.

درخت جستجو, درختی است که داده‌ها در آن به ترتیب خاصی وجود دارند.

درخت جستجو دودویی:

درخت جستجوی دودویی (Binary Search Tree) درختی دودویی است که گره‌های آن حاوی فیلد کلید است.

این نوع درخت‌ها, دارای ویژگی زیر است:

اگر k مقدار کلید در هر گره باشد, آنگاه برای هر کلید x که در زیر درخت چپ این گره وجود دارد, x<k و برای هر کلید y که در زیر درخت راست آن گره قرار دارد, y>k.

این ویژگی را BST می‌گویند و تضمین می‌کند که پیمایش inorder درخت BST , مقدار کلید موجود در گره‌های درخت را به ترتیب صعودی نمایش دهد.

درج گره در BST:

برای درج گره‌ای با مقدار کلید k در BST مراحل زیر باید دنبال شود:

۱- اگر درخت خالی است, گره جدید در ریشه قرار داده شود. پایان

۲- در غیر این‌صورت اشاره‌گر p برابر با اشاره‌گر ریشه قرار گیرد.

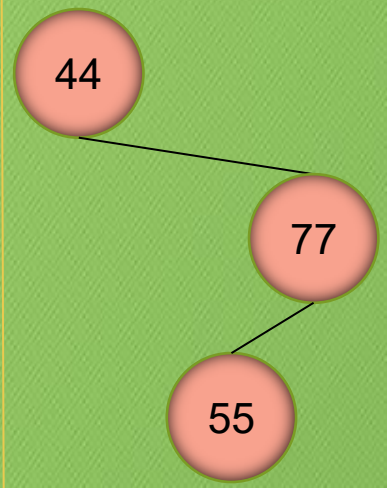
۳- اگر k کوچک‌تر از مقداری است که در p قرار دارد و گره p فاقد فرزند چپ است, گره جدید به عنوان فرزند چپ p قرار گیرد. پایان

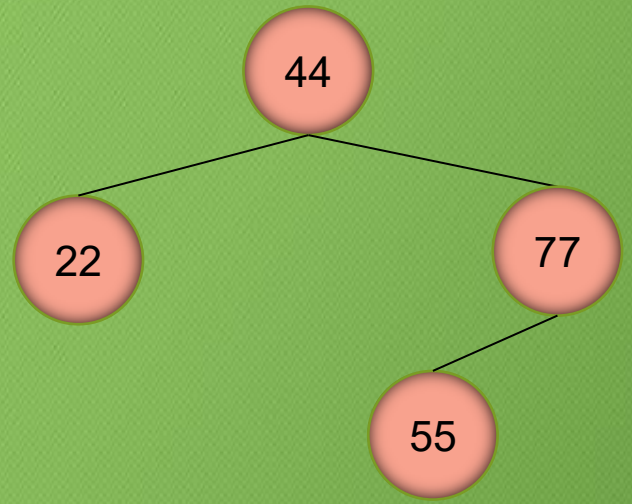
۴- اگر k بزرگ‌تر از مقداری است که در p قرار دارد و گره p دارای فرزند چپ است, p برابر فرزند چپ قرار گیرد و به مرحله ۳ برود.

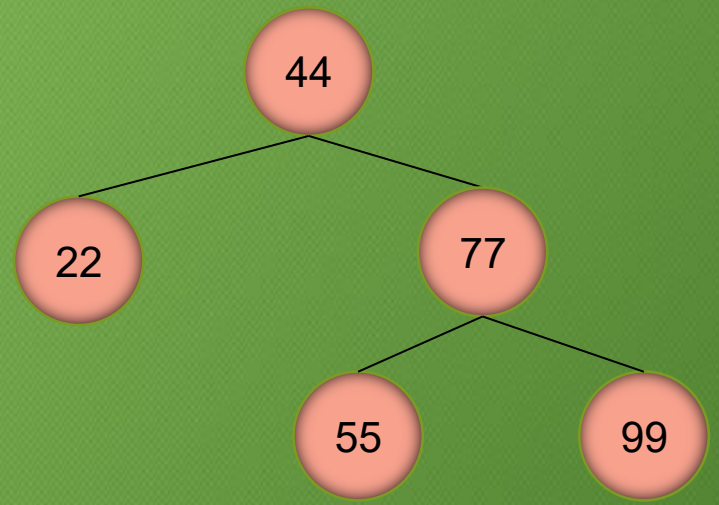
۵- اگر گره‌ای که p به آن اشاره می‌کند فرزند راست ندارد, گره جدید به عنوان فرزند راست p انتخاب شود. پایان

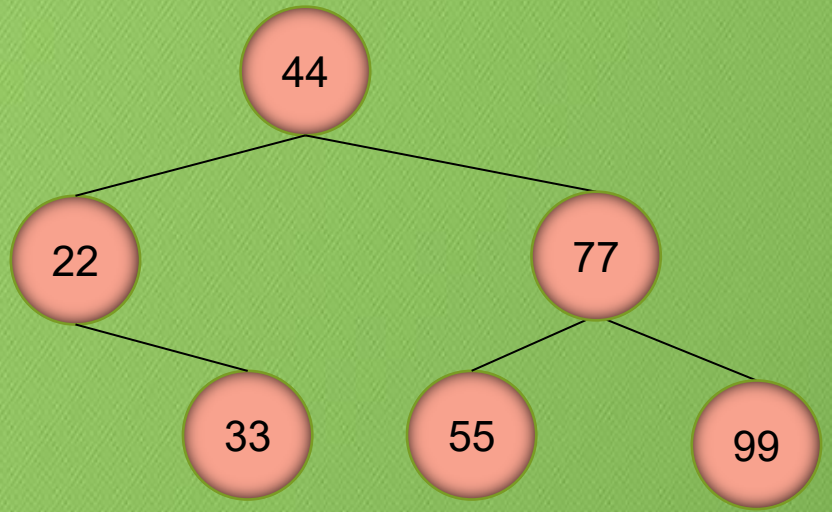
۶- گره p برابر با فرزند راست p قرار گیرد و به مرحله ۳ برود.

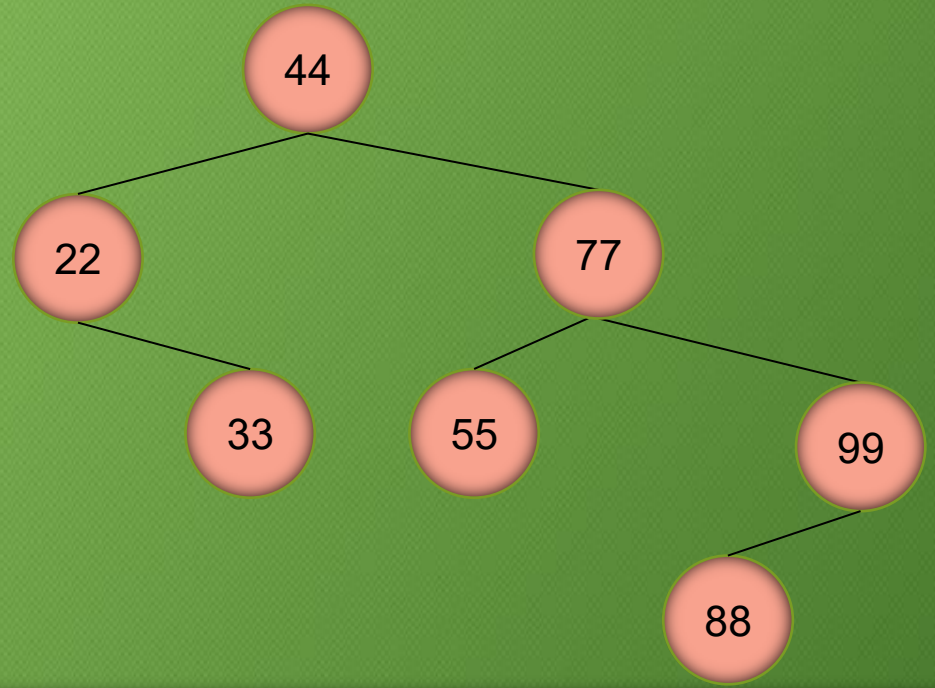
ساخت درخت جستجوی دودویی با عناصر 44, 77, 55, 22, 99, 33, 88











درج در BST

```
void  tree::insert(int  num)
{
	treeNode  *node, *help;
	help = root;
	node = new treeNode;
	node->left = node->right = NULL;
	node->info = num;
	if (root == NULL)
		root = node;
	else
	{
		while (help != NULL)
		{
			if (node->info > help->info)
			{
				if (help->right != NULL)
					help = help->right;
				else
				{
					help->right = node;
					break;
				}
			}
			else
			{
				if (help->left != NULL)
					help = help->left;
				else
				{
					help->left = node;
					break;
				}
			}
		}
	}
}
```

جستجوی گره در BST:

برای جستجوی گره‌ای با مقدار k در یک درخت جستجوی دودویی مراحل زیر دنبال می‌شود:

از ریشه شروع می‌شود.

۱- اگر مقدار k برابر با مقدار p بود جستجو پایان می‌یابد.

۲- اگر مقدار k کمتر از گره p بود, مقدار k در زیر درخت چپ قرار دارد و زیر درخت چپ باید جستجو شود, اشاره‌گر p به زیر درخت چپ اشاره کرده و به مرحله ۱ می‌رود. پایان.

۳- اگر مقدار k بیشتر از گره p بود, مقدار k در زیر درخت راست قرار دارد و زیر درخت راست باید جستجو شود, اشاره‌گر p به زیر درخت راست اشاره کرده و به مرحله ۱ می‌رود. پایان.

تابع جستجوی
BST

```
int  tree :: search(tree *s, int  num)
{
    if (!s)
        return 0;
    if (num == s->info)
        return 1;
    if (num < s->key)
        return search(s->left, num);
    else
        return search(s->right, num);
}
```

حذف گره از BST:

برای حذف گره‌ای با مقدار k در یک درخت جستجوی دودویی سه حالت در نظر گرفته می‌شود:

- k برگ است.

اگر k فرزند چپ باشد, اشاره‌گر چپ والد و اگر k فرزند راست باشد, اشاره‌گر راست والد باید تهی شود.

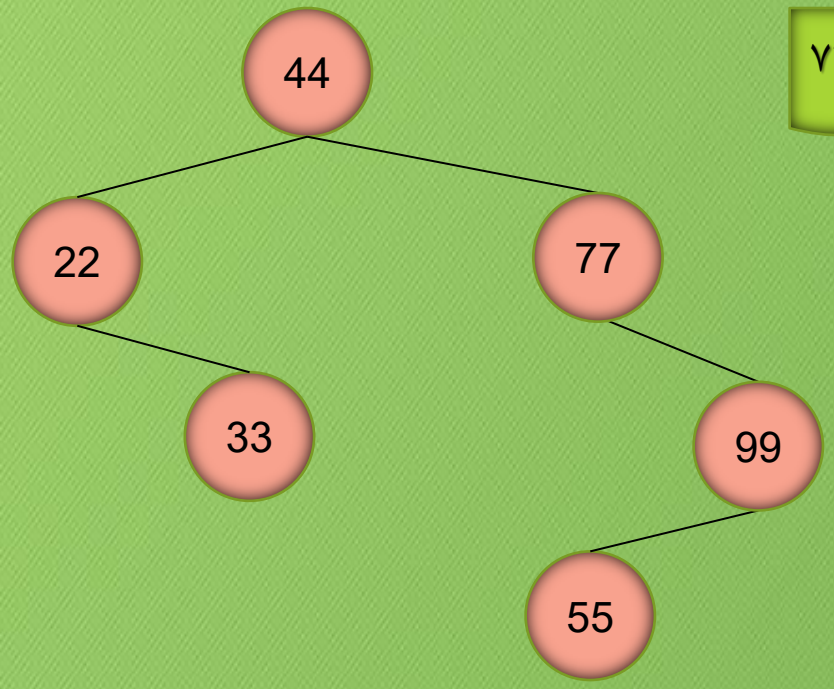
- k یک فرزند دارد.

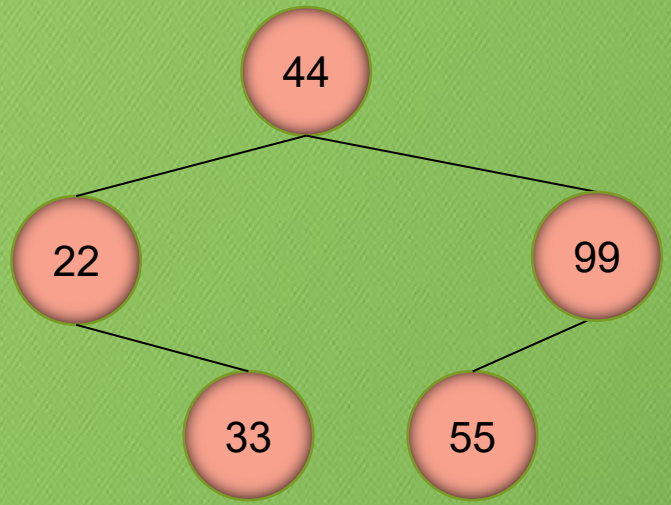
در این حالت باید اشاره‌گر مناسبی از گره والد k به فرزند k اشاره نماید.

- k دو فرزند دارد.

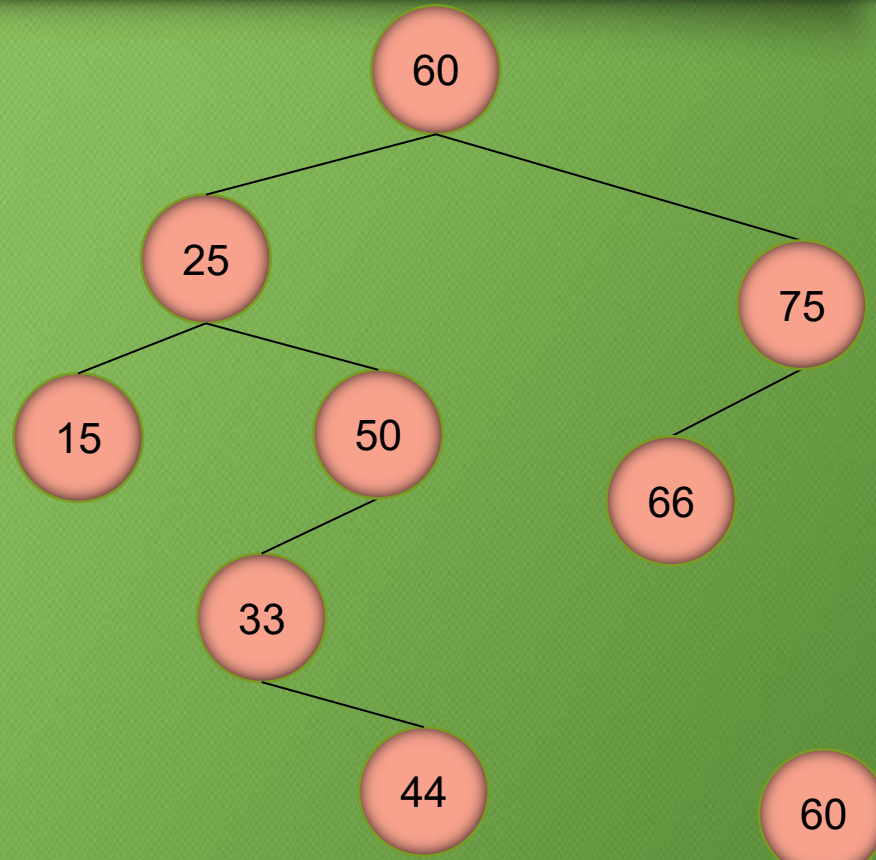
در این حالت باید گره‌ای پیدا شود که در پیمایش inorder بعد از گره k قرار دارد و این گره y نامیده می‌شود. گره y با این ویژگی فاقد فرزند چپ است. ابتدا گره y با استفاده از یکی از دو حالت فوق حذف شده و سپس گره y به جای گره k قرار می‌گیرد.

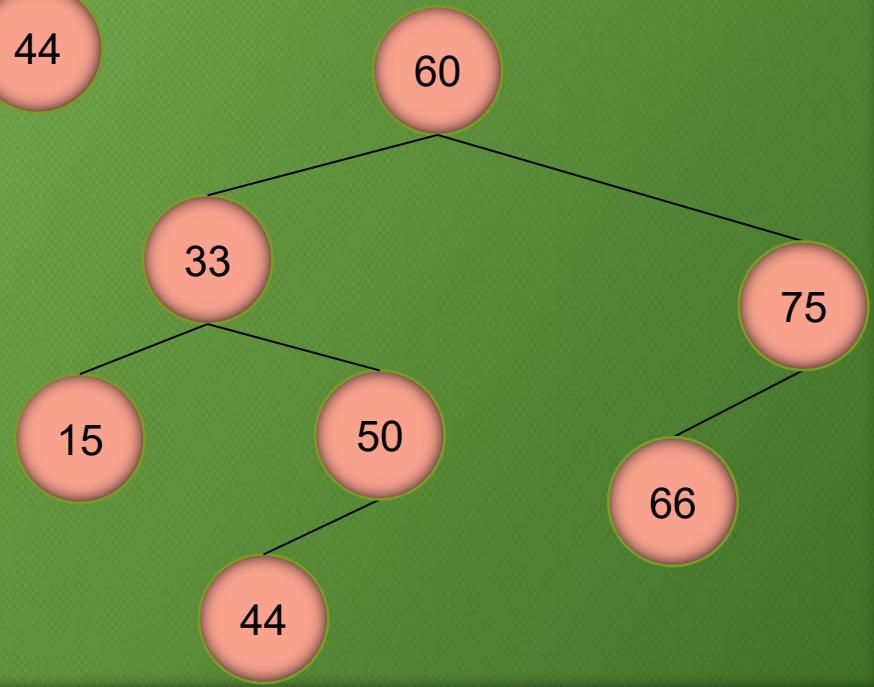
حذف ۷۷


44
22
77
33
99
55


44
22
99
33
55

حذف ۲۵


60
25
75
15
50
66
33
44


60
33
75
15
50
66
44

جلسه دوازدهم:

درخت‌ها

آنچه در این جلسه می‌خوانید:

۱- درخت متوازن

۲- درج گره در درخت AVL

۳- انواع چرخش

درخت متوازن:

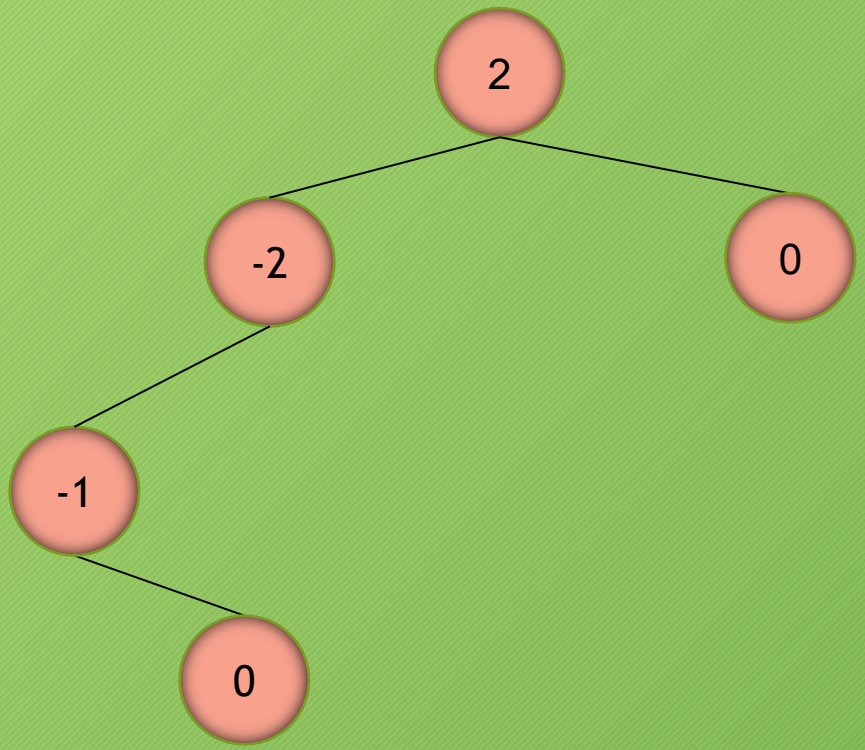
درخت متوازن که نام دیگر آن درخت AVL است, نوعی درخت جستجوی دودویی است. AVL حروف اول نام مخترعین آن است.

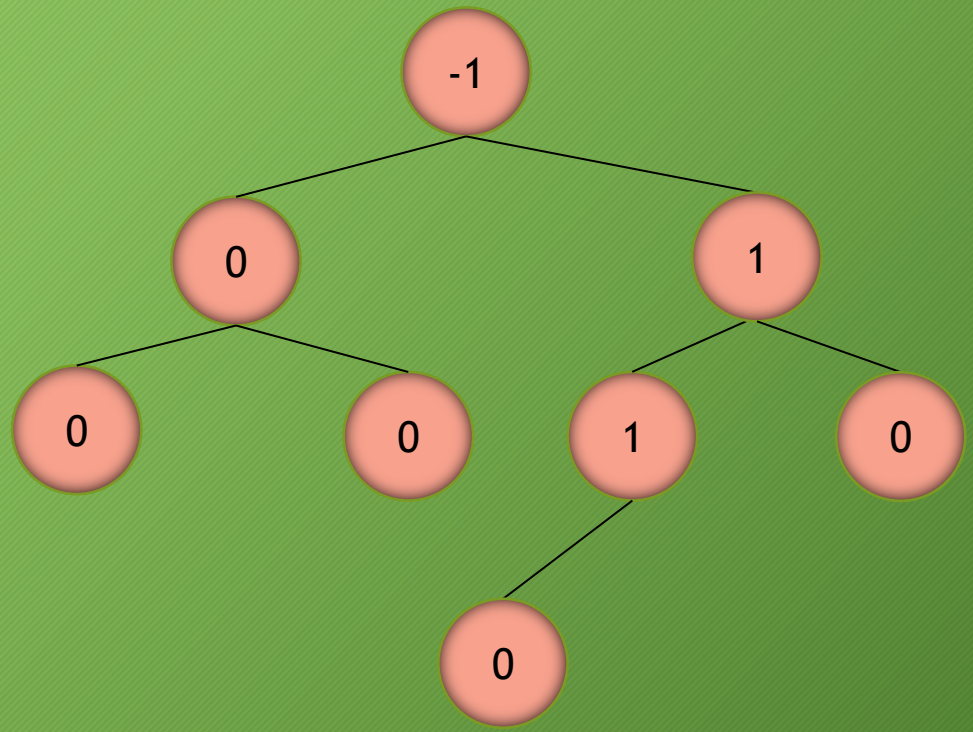
- **درجه توازن:**

درجه توازن هر گره, اختلاف ارتفاع زیر درخت راست و زیر درخت چپ آن گره است.

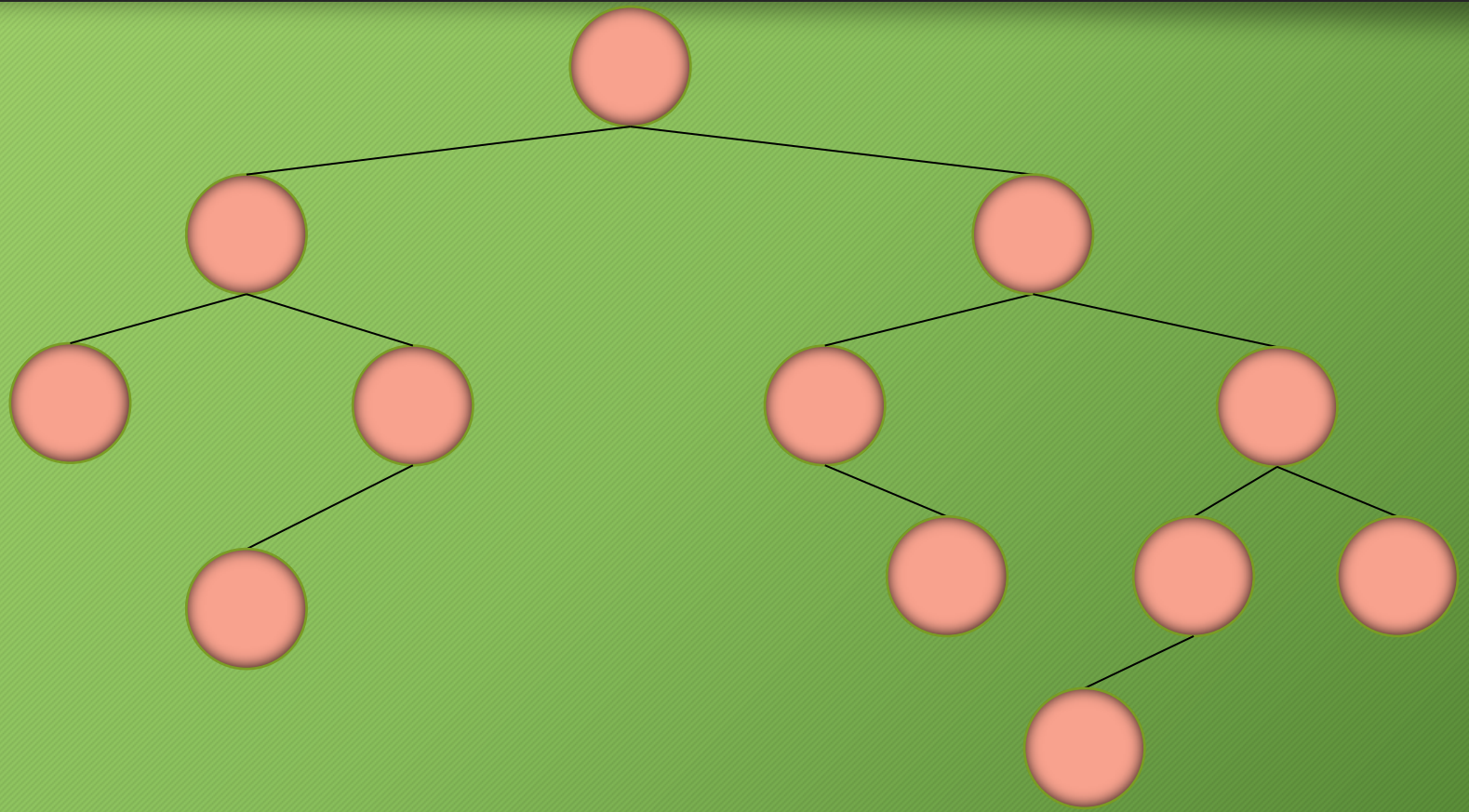
درخت جستجوی دودویی را وقتی متوازن می‌گوییم که درجه توازن تمام گره‌های آن ۰, ۱ یا ۱- باشد.

درخت متوازن؟





درخت متوازن؟



درج گره در AVL:

درج در AVL همانند درج در BST است اما درج گره‌ای در BST ممکن است آن را از حالت توازن خارج کند و درجه توازن را به ۲ یا ۲- تغییر دهد.

گره‌ای که درجه توازن آن به ۲- یا ۲ تبدیل شده است, گره محور نامیده می‌شود.

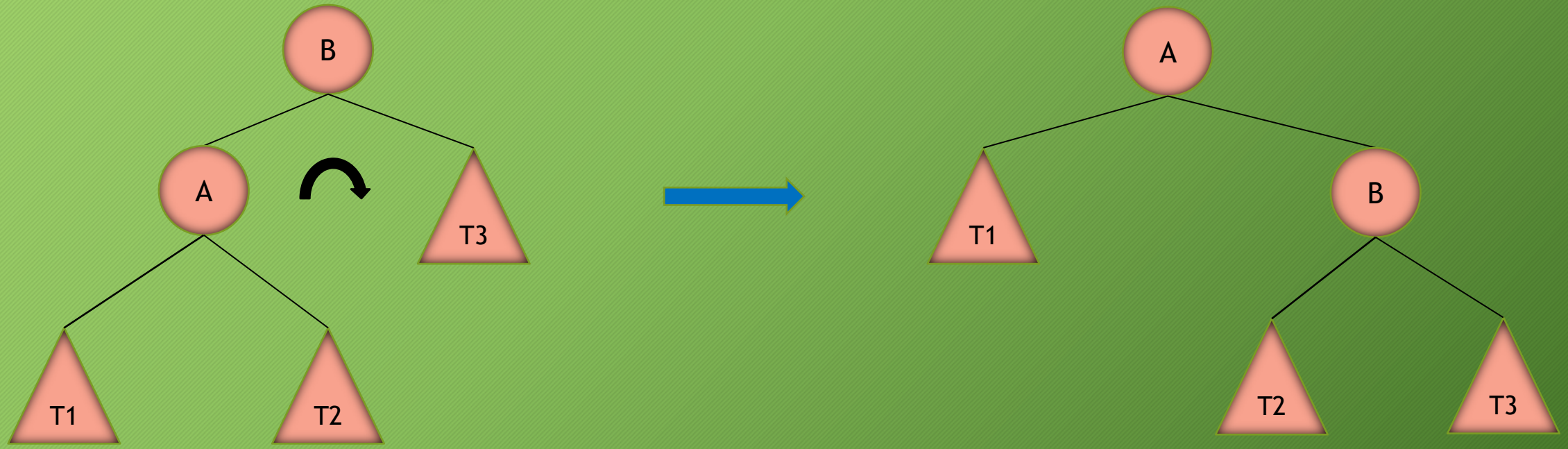
یکی از کارهای عمل درج در AVL این است که درخت متوازن نگه داشته شود.

برای تبدیل درخت نامتوازن به درخت متوازن از چهار نوع چرخش استفاده می‌شود.

انواع چرخش:

- چرخش راست:

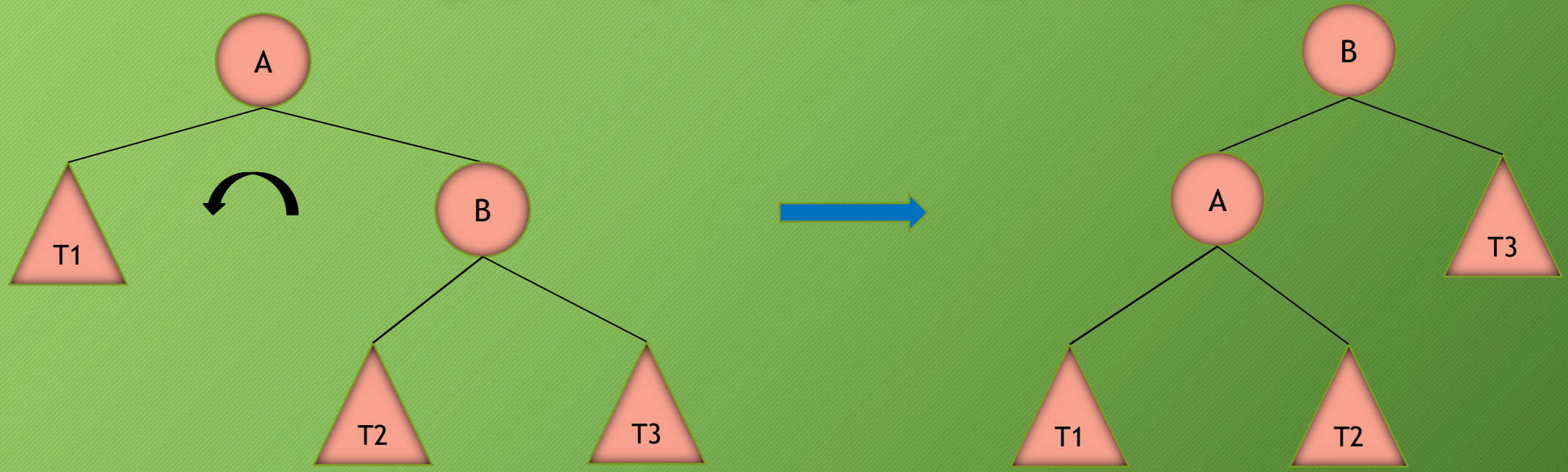
وقتی انجام می‌شود که گره جدید به زیر درخت چپِ (L), مربوط به زیر درخت چپِ (L) گره محور اضافه شود. این چرخش را, چرخش LL نیز می‌نامند.



انواع چرخش:

- چرخش چپ:

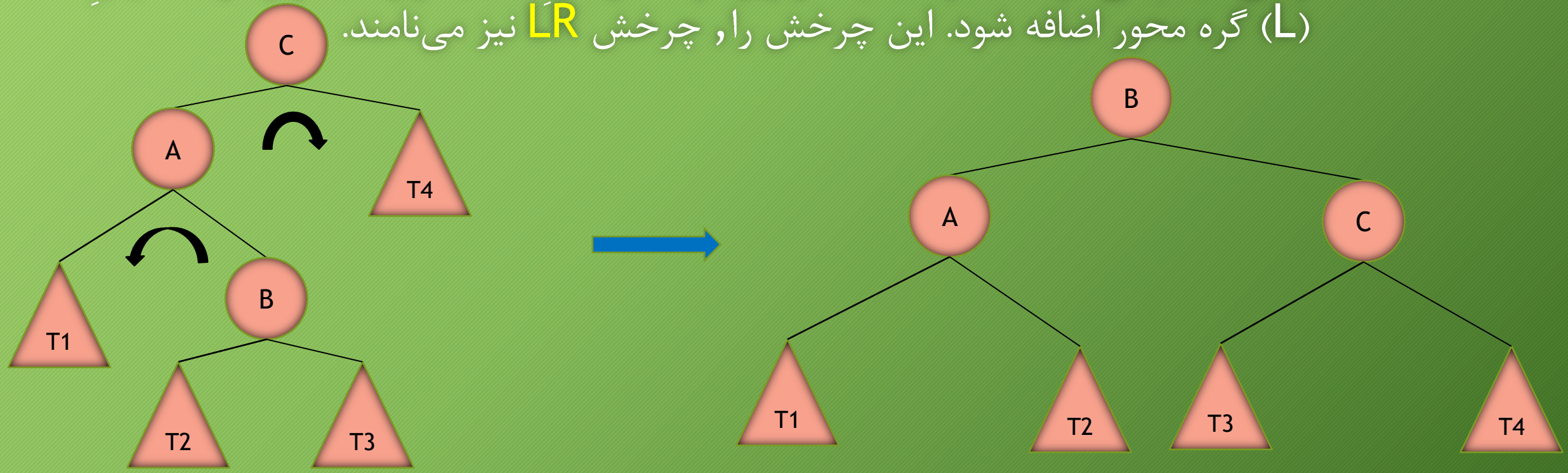
وقتی انجام می‌شود که گره جدید به زیر درخت راست (R), مربوط به زیر درخت راستِ (R) گره محور اضافه شود. این چرخش را, چرخش RR نیز می‌نامند.



انواع چرخش:

- چرخش مضاعف چپ-راست:

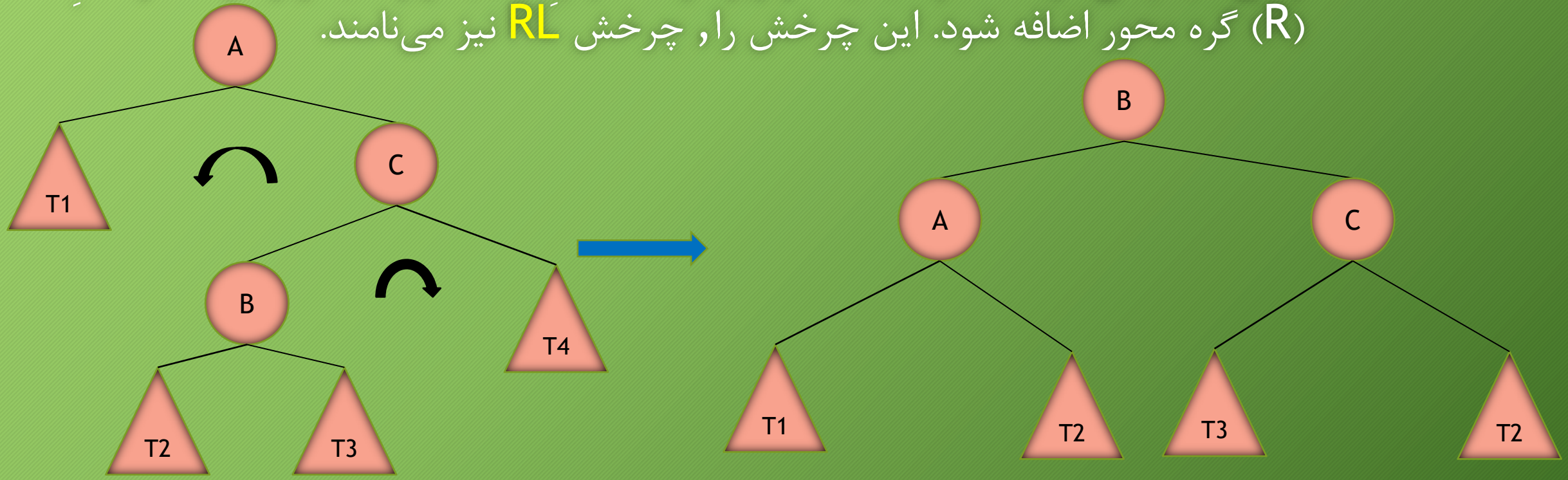
وقتی انجام می‌شود که گره جدید در زیر درخت راست (R) , مربوط به زیر درخت چپِ (L) گره محور اضافه شود. این چرخش را, چرخش LَR نیز می‌نامند.



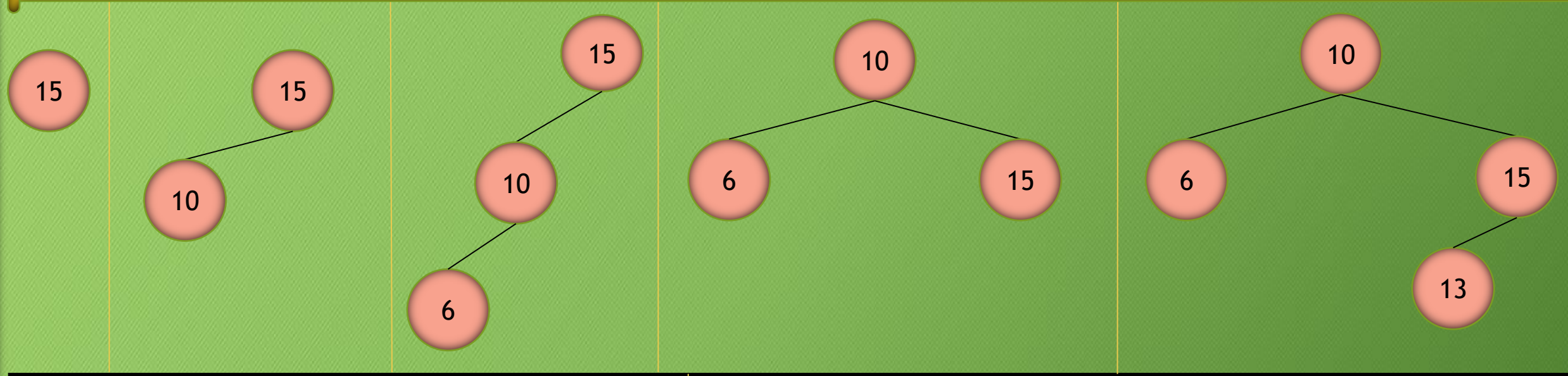
انواع چرخش:

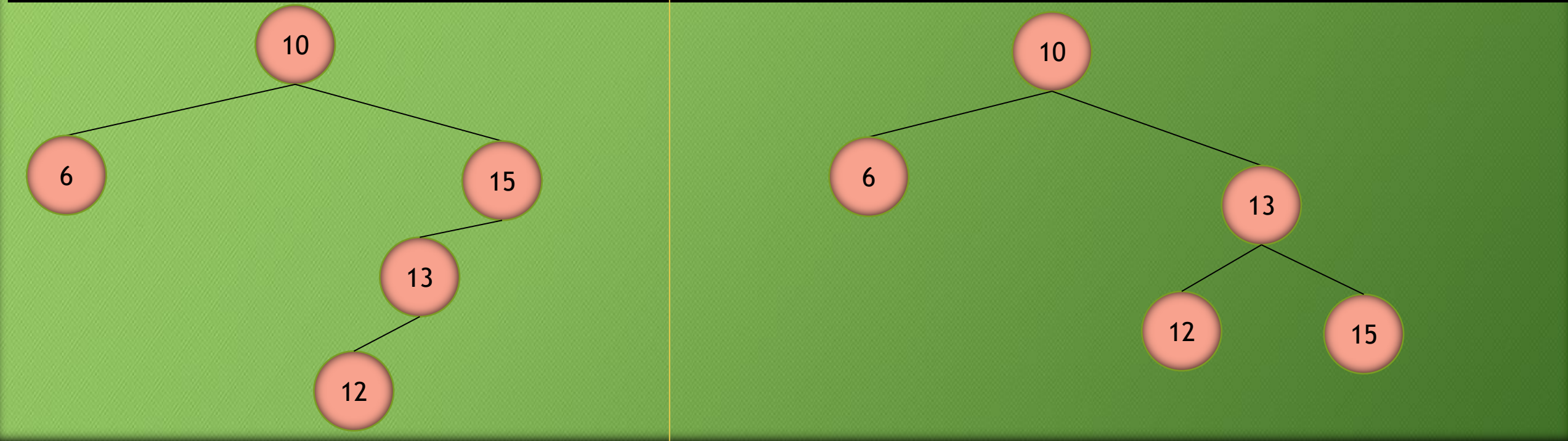
- چرخش مضاعف راست–چپ:

وقتی انجام می‌شود که گره جدید در زیر درخت چپ (L), مربوط به زیر درخت راستِ (R) گره محور اضافه شود. این چرخش را, چرخش RL نیز می‌نامند.



15, 10, 6, 13, 12





تمرین:

18, 56, 43, 60, 15, 4, 29, 33, 66, 65

۱- با اعداد فوق درخت BST بسازید.

۲- با اعداد فوق درخت AVL بسازید.

جلسه سیزدهم:

کاربرد درخت‌های دودویی

آنچه در این جلسه می‌خوانید:

۱- رمزگذاری هافمن

رمزگذاری هافمن:

فرض کنید الفبایی با n نماد موجود است و قرار است یک پیام طولانی متشکل از نمادهای این الفبا ارسال شود (به عنوان مثال, الفبای انگلیسی دارای ۲۶ نماد است).

این پیام قبل از ارسال باید به صورت رشته طولانی از بیت‌ها در آید که هر بیت ۰ یا ۱ است.

برای رمزگذاری پیام, به هر نماد یک کد نسبت داده می‌شود و سپس کد نمادهای تشکیل دهنده پیام با هم ترکیب می‌شوند.

در پیام‌های خیلی طولانی که نمادها به ندرت تکرار می‌شوند, صرفه‌جویی‌ها قابل توجه هستند.

معمولاً کدها بر اساس تکرار آنها در یک پیام تولید نمی‌شوند, بلکه بر اساس تکرار آنها در تعداد زیادی از پیام‌ها ساخته می‌شوند. سپس این کدها برای تمام پیام‌ها مورد استفاده قرار می‌گیرند.

درخت‌هایی دودویی برای یافتن کدهای با حداقل طول, به کار می‌روند.

برای این منظور از نوعی درخت دودویی به‌نام درخت رمزگذاری هافمن استفاده می‌شود.

نماد	A	B	C	D
کد	00	01	10	11

ABACCDA
00010010101100

رمزگذاری هافمن:

با توجه به جدول فراوانی زیر درخت هافمن بسازید.

نماد	N	T	E	S	I
فراوانی	9	10	29	4	5

- مرحله ۱:

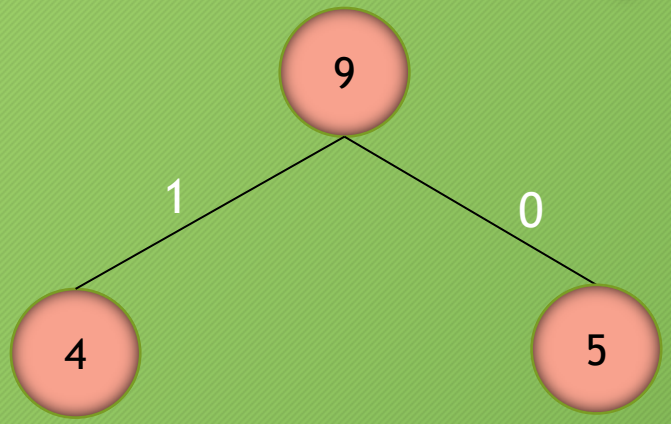
ابتدا باید فراوانی‌ها مرتب شوند.

4, 5, 9, 10, 29

رمزگذاری هافمن:

- مرحله ۲:

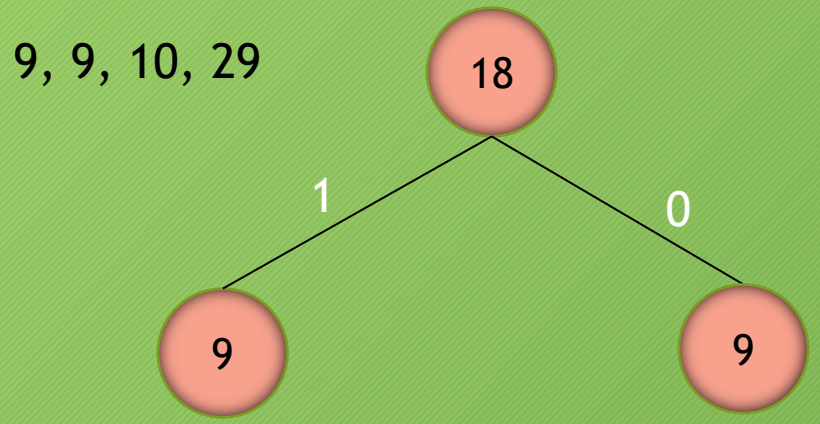
دو مقدار کوچک یعنی ۴ و ۵ را انتخاب کرده و درختی دودویی ایجاد می‌شود که ریشه آن جمع دو عدد یعنی ۹ و برگهای آن دارای مقادیر ۴ و ۵ هستند. به پیوند سمت چپ مقدار ۱ و به پیوند سمت راست مقدار ۰ نسبت داده می‌شود.

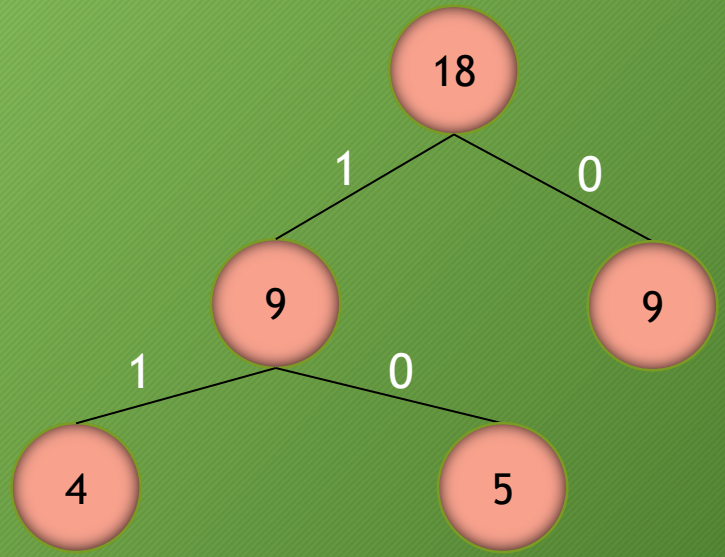


رمزگذاری هافمن:

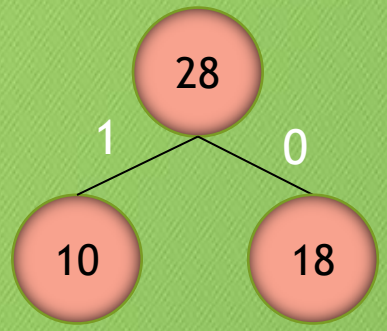
- مرحله ۳:

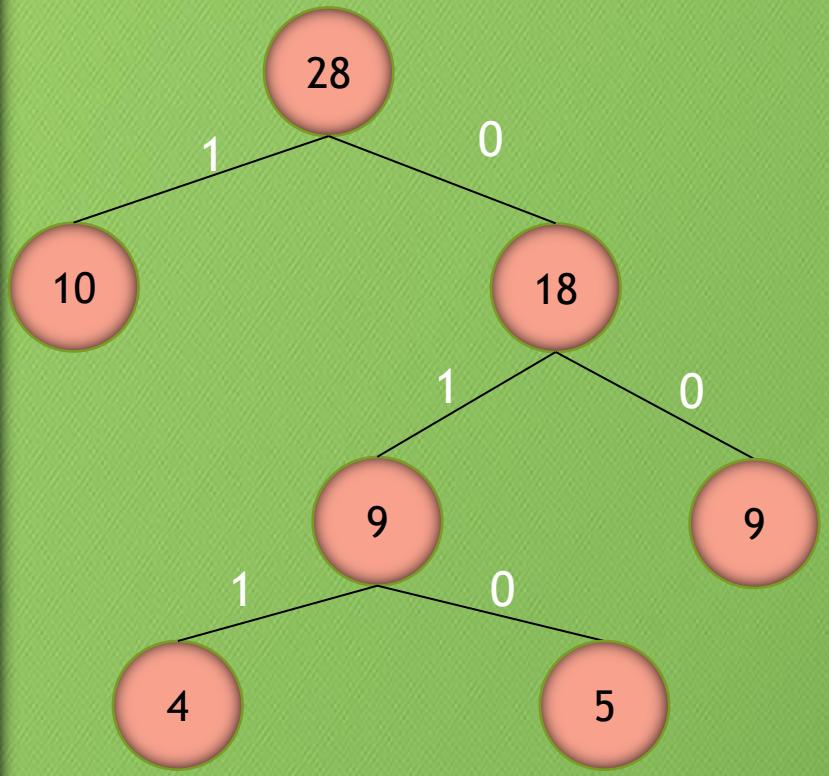
در دنباله قبل به‌جای ۴ و ۵, ۹ قرار داده, اگر نیاز به مرتب کردن باشد مرتب شده و سپس مانند مرحله ۲ عمل می‌شود. سپس درخت قبلی با درخت جدید ترکیب می‌شود.



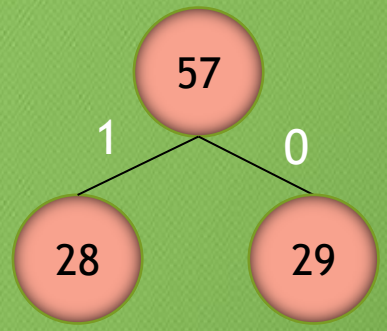


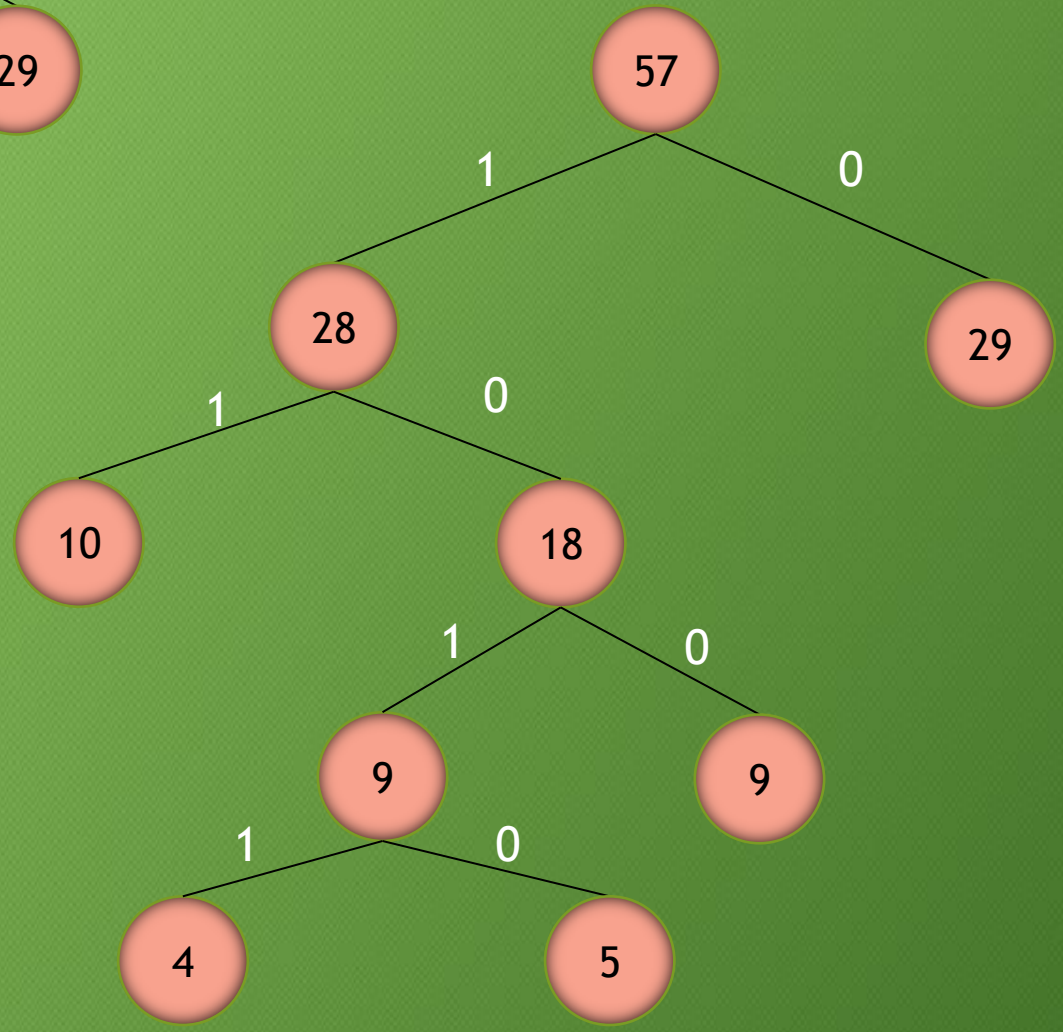
18, 10, 29 ⟹ 10, 18, 29



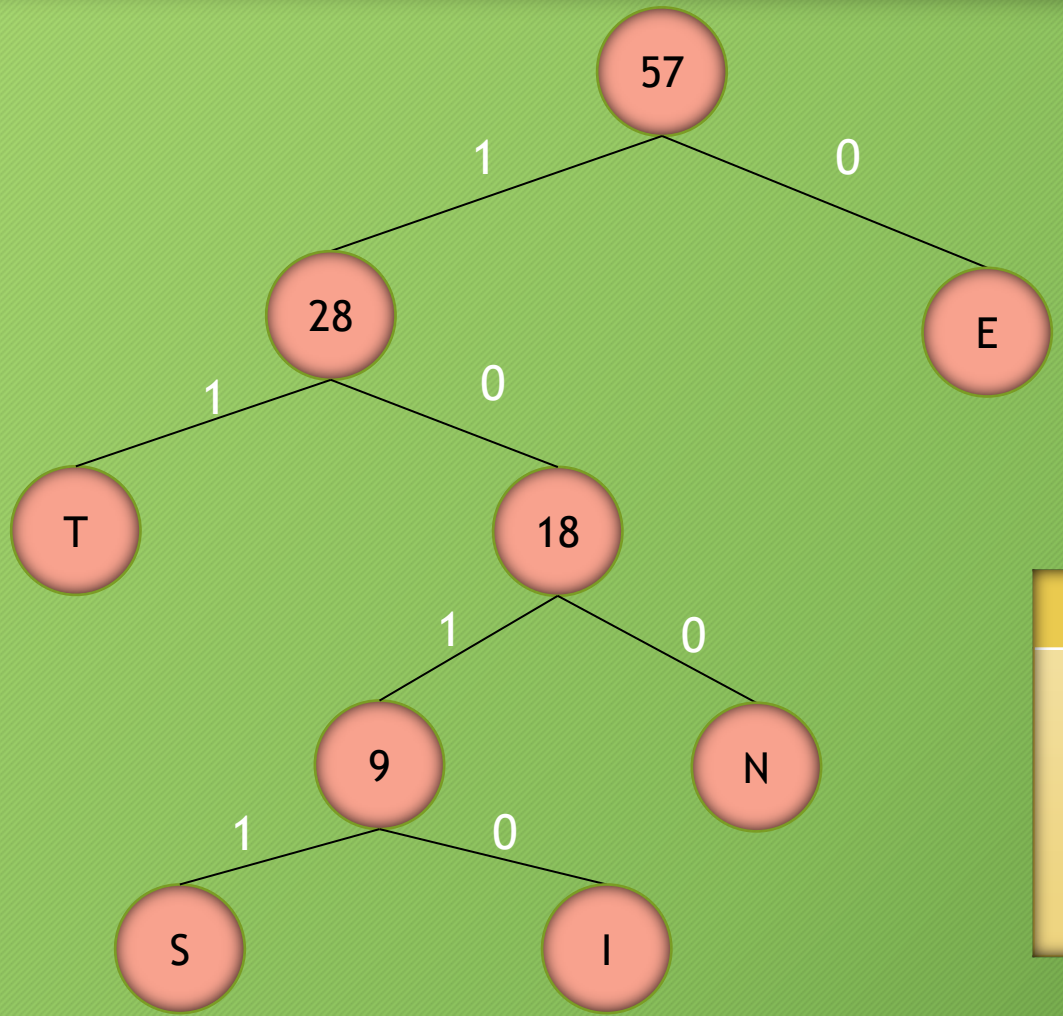


28, 29





رمزگذاری هافمن:



- مرحله ۴:

به جای مقادیر گره‌های برگ, حروف قرار داده می‌شود.

نماد	کد	فراوانی
E	0	29
T	11	10
N	100	9
I	1010	5
S	1011	4

پیام	رمز
SENT	1011010011
TENNIS	1101001001010101011
NEST	1000101111
SIT	1011101011

تمرین:

با توجه به جدول فراوانی زیر درخت هافمن بسازید.

نماد	G	R	E	N	D
فراوانی	18	12	22	4	7

جلسه چهاردهم:

گراف‌ها

آنچه در این جلسه می‌خوانید:

۱- تعریف گراف

۲- تعاریف مربوط به گراف

گراف:

گراف یک ساختار کلی است که درخت حالت خاصی از این ساختار است.

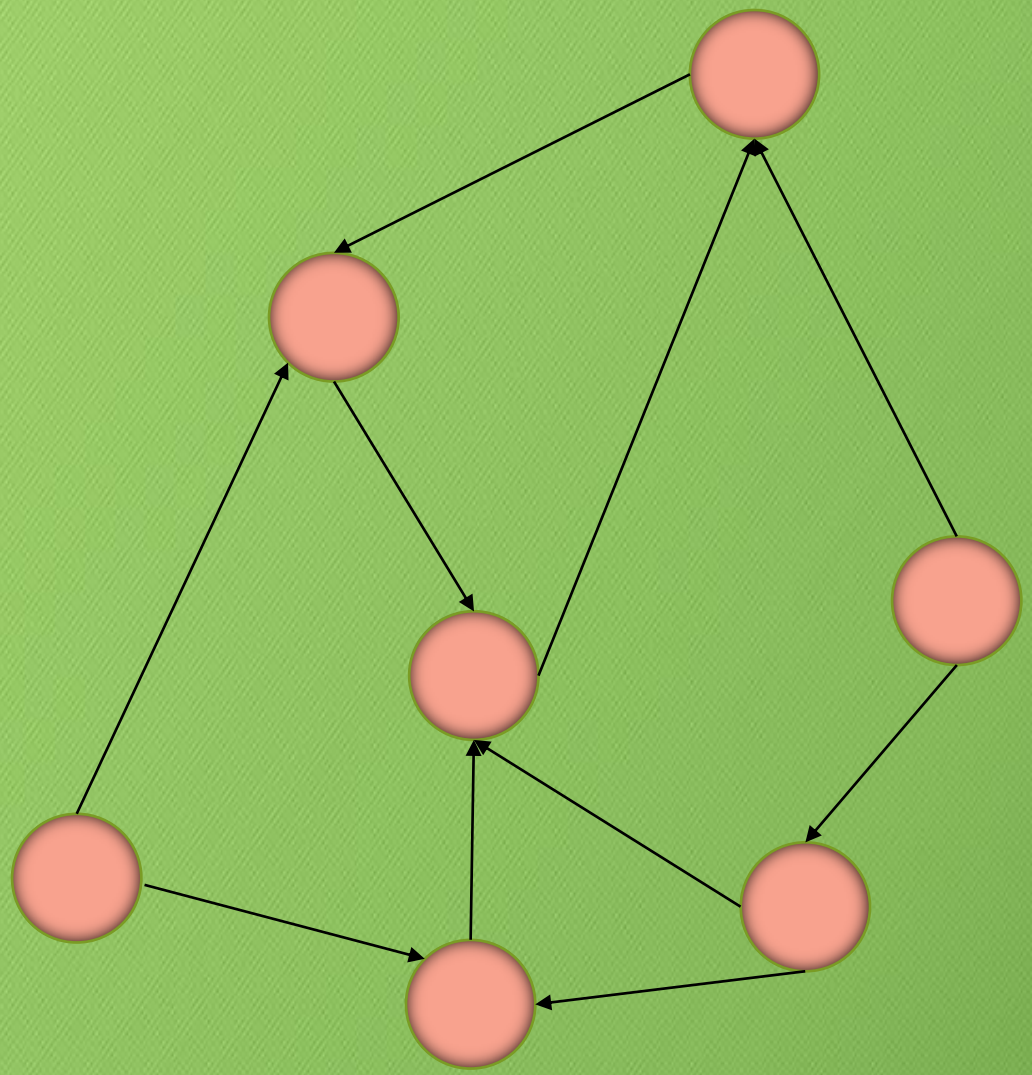
گراف‌ها برای مدل‌سازی شبکه‌های کامپیوتری و سایر شبکه‌هایی مفید است که در آنها سیگنال‌ها, پالس‌های الکتریکی هستند و مانند اینها در مسیرهای گوناگونی از یک گره به گره دیگر جریان می‌یابند.

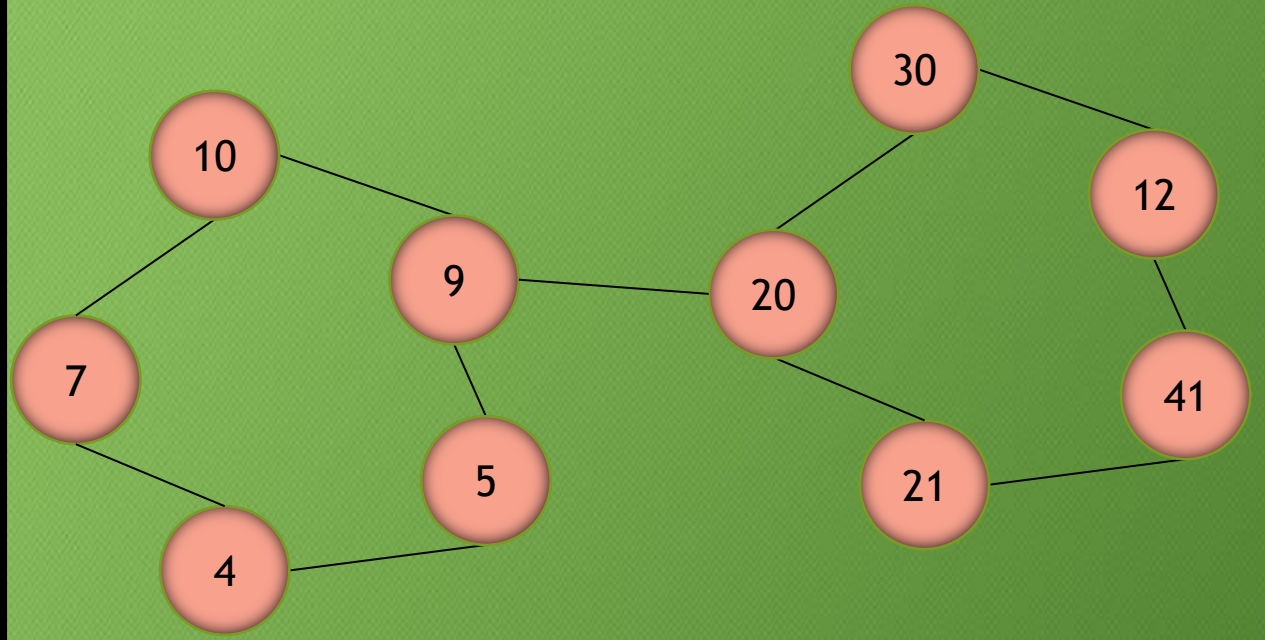
- گراف:

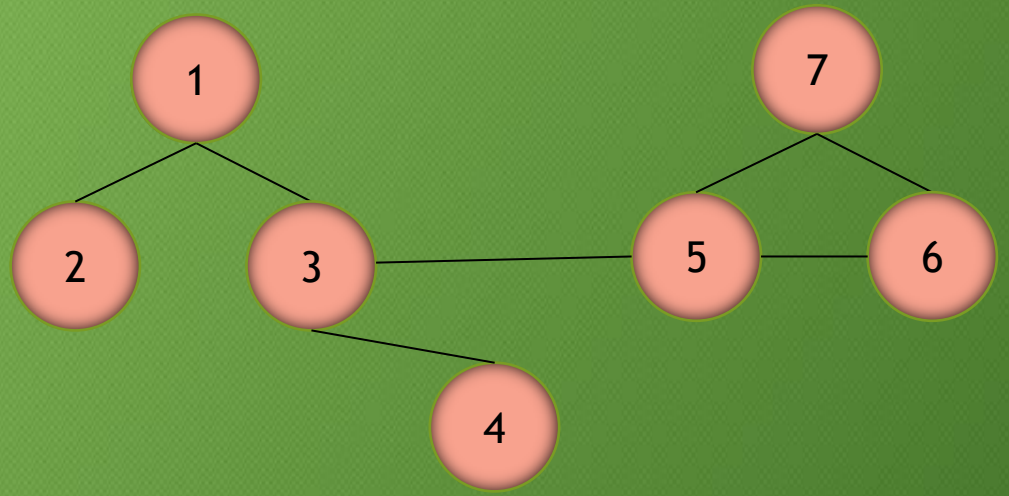
گراف مجموعه‌ای از نقاط است که در آن, بعضی از نقاط توسط خطوطی به هم متصل شده‌اند. نقاط را گره و خطوط را یال می‌نامند. اگر یال به صورت فلش باشد, آن گراف را جهت‌دار می‌گویند. اگر یال‌ها فاقد فلش باشند, گراف را بدون جهت می‌گویند.

- گره‌های همجوار:

در هر دو نوع گراف فوق, دو گره x و y را در صورتی همجوار گویند که یالی از x به y وجود داشته باشد.




30
10
12
9
20
7
41
5
21
4


1
7
2
3
5
6
4

گراف:

- **مسیر:**

مسیر دنباله‌ای از گره‌ها است که در آن هر گره, همجوار گره دیگری است.

- **طول مسیر:**

طول مسیر برابر با تعداد یال‌های موجود در آن است.

- **چرخه:**

مسیری با طول بیش از یک است که از یک گره شروع و به همان گره ختم می‌شود.

- **چرخه ساده:**

در گراف بدون جهت, چرخهٔ ساده چرخه‌ای است که از یک یا چند گره تشکیل شده است که در آن, در مسیر چرخه ساده, هیچ گره‌ای بیش از یک بار ملاقات نشود, البته بجز گرهٔ شروع و پایان که باید یکی باشد تا چرخه بسته شود و حلقه‌ای بوجود آید.

گراف:

- گره‌ها و مؤلفه‌های متصل:

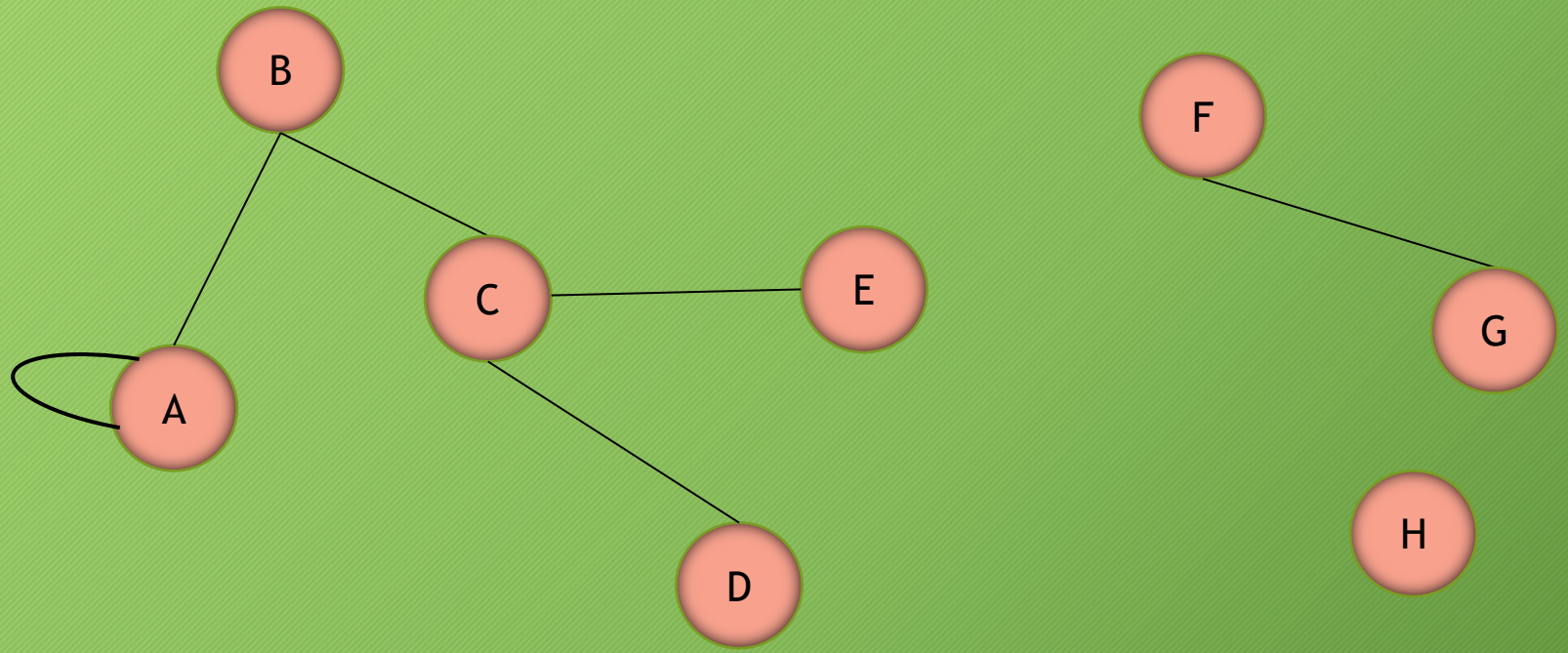
در گراف بدون جهت, دو گره x و y را در صورتی متصل گویند که مسیری بین آنها وجود داشته باشد. علاوه بر این, مجموعه‌ای از گره‌ها مثل S را متصل گویند اگر از هر گره‌ای به گره دیگر مسیری وجود داشته باشد.

گراف بدون جهت را می‌توان به چند مؤلفه متصل تقسیم کرد. هر مؤلفه متصل, از زیر مجموعه‌ای از گره‌ها تشکیل شده است که همگی به هم متصل هستند. اما اگر گراف بدون جهت از یک یا چند مؤلفه مجزا تشکیل شده باشد, هیچ‌گاه مسیری از یک گره در مؤلفه‌ای به گره دیگری در مؤلفه دیگر وجود ندارد.

گراف:

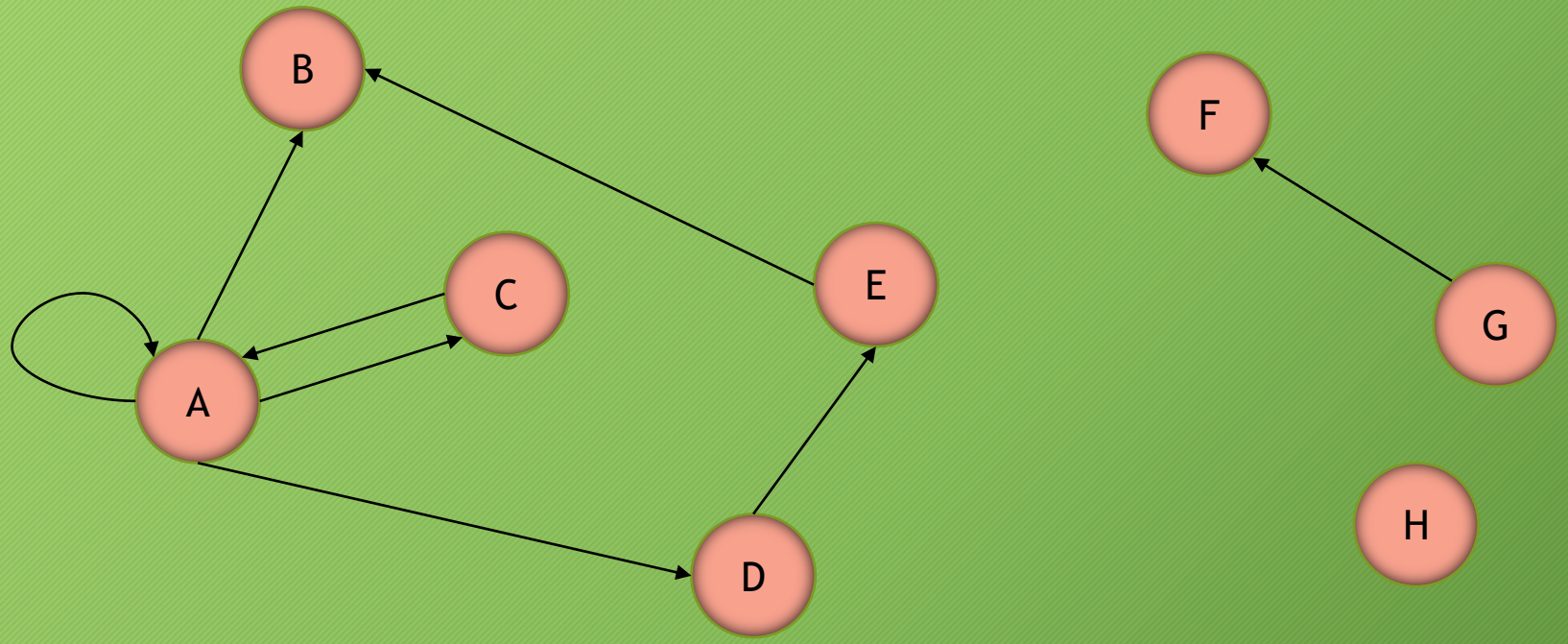
- گراف G=(V,E) از مجموعه‌ای از گره‌ها به نام V و مجموعه‌ای از یال‌ها به نام E تشکیل شده است, به طوری که یال‌های موجود در E از زوج‌های موجود در V بوجود می‌آید.
- در گراف بدون جهت, هر یال e=(v1,v2) زوج نامرتبی از گره‌ها است که گره‌های v1 و v2 را به هم متصل می‌کند, به طوری که جهتی از v1 به v2 و برعکس تعریف نمی‌شود.
- در گراف جهت‌دار, هر یال e=(v1,v2) زوج مرتبی از گره‌ها است که v1 را به v2 متصل می‌کند و جهت اتصال آن از v1 به v2 است که در این حالت v1 را مبدأ و v2 پایانه یال e نام دارند.

گراف:



V = {A, B, C, D, E, F, G, H}
E = {<A, B>, <B, C>, <C, D>, <C, E>, <F, G>, <A, A>}

گراف:



V = {A, B, C, D, E, F, G, H}
E = {<A, B>, <A, C>, <C, A>, <A, D>, <D, E>, <E, B>, <G, F>, <A, A>}

گراف:

- **همجواری:**

دو گره v_i و v_j در گراف $G=(V,E)$ را همجوار گویند, اگر یالی مثل $e \in E$ وجود داشته باشد به طوری که $e=(v_i,v_j)$.

- **مسیر:**

مسیری مثل p در $G=(V,E)$ دنباله‌ای از گره‌ها به شکل $p=v_1 , v_2 , ..., v_n$ است, به طوری که هر v_i همجوار v_{i+1} است $(n\geq 2, 1\leq i\leq n-1)$.

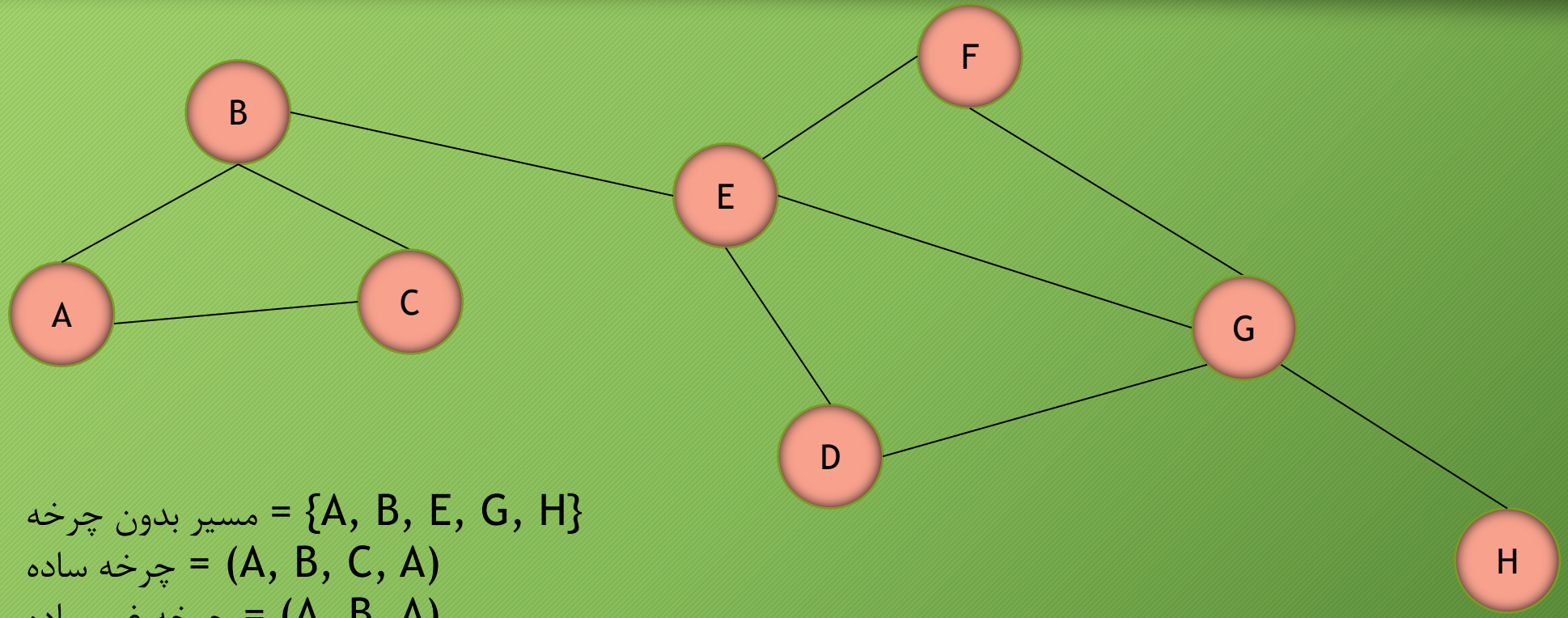
- **چرخه مسیر:**

چرخه مسیر مثل v_n ,..., v_2 , $v_1=p$ است, به طوری که هر $v_1=v_n$ است.

- **چرخه ساده:**

مسیری است که از سه یا بیشتر از سه رأس عبور می‌کند و آنها را در یک حلقه به هم وصل می‌نماید. به عبارت دیگر, وقتی در مسیر چرخه ساده حرکت می‌شود, حداقل باید سه گره ملاقات شود و از هر گره, بیش از یکبار نمی‌توان عبور کرد.

گراف:



مسیر بدون چرخه = {A, B, E, G, H}
چرخه ساده = (A, B, C, A)
چرخه غیر ساده = (A, B, A)
چرخه غیر ساده = (E, F, G, E, D, G, E)

گراف:

- **گره‌های متصل:**

دو گره v_i و v_j در گراف $G=(V,E)$ را همجوار گویند, اگر یالی مثل $e \in E$ وجود داشته باشد به طوری که $e=(v_i,v_j)$.

- **مسیر:**

مسیری مثل p در $G=(V,E)$ دنباله‌ای از گره‌ها به شکل $p=v_1 , v_2 , ..., v_n$ است, به طوری که هر v_i همجوار v_{i+1} است $(n\geq 2, 1\leq i\leq n-1)$.

- **چرخه مسیر:**

چرخه مسیر مثل v_n ,..., v_2 , $v_1=p$ است, به طوری که هر $v_1=v_n$ است.

- **چرخه ساده:**

مسیری است که از سه یا بیشتر از سه رأس عبور می‌کند و آنها را در یک حلقه به هم وصل می‌نماید. به عبارت دیگر, وقتی در مسیر چرخه ساده حرکت می‌شود, حداقل باید سه گره ملاقات شود و از هر گره, بیش از یکبار نمی‌توان عبور کرد.

گراف:

- **درجه گره:**

در گراف بدون جهت G درجهٔ گره‌ای مثل x برابر تعداد یال‌هایی است که در آن گره تلاقی می‌کنند.

در گراف جهت‌دار, درجه ورودی و درجه خروجی وجود دارد.

جلسه چهاردهم:

گراف‌ها

آنچه در این جلسه می‌خوانید:

۱- پیاده‌سازی یا نمایش گراف‌ها
۲- پیاده‌سازی گراف با ماتریس همجواری
۳- درجه گره
۴- گراف وزن‌دار
۵- الگوریتم کوتاه‌ترین مسیر
۶- الگوریتم وارشال برای تعیین ماتریس کوتاه‌ترین مسیر
۷- الگوریتم دایکسترا برای تعیین ماتریس کوتاه‌ترین مسیر

پیاده‌سازی یا نمایش گراف‌ها:

دو روش متداول برای نمایش گراف‌ها وجود دارد.

این دو روش عبارتند از ماتریس همجواری و لیست پیوندی.

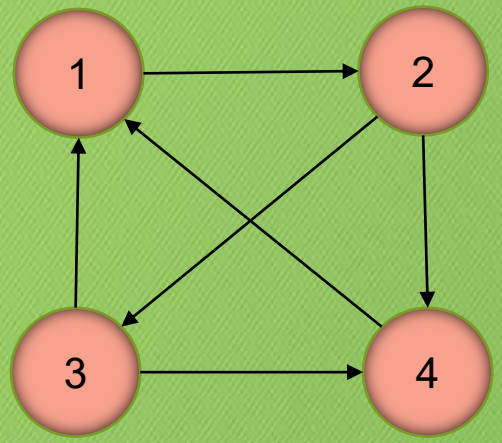
پیاده‌سازی گراف با ماتریس همجواری:

در گرافی مانند $G=(V,E)$ با n گره, ماتریس همجواری این گراف یک آرایه دو بُعدی $n\times n$ است که T نامیده می‌شود.

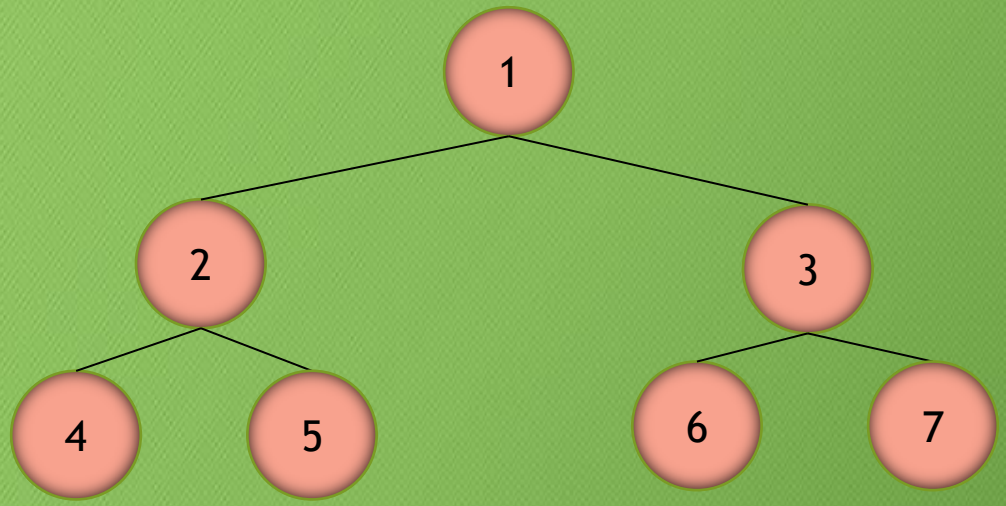
اگر (v_i,v_j) یالی در گراف باشد, آنگاه $T[i][j]=1$ خواهد بود و اگر چنین یالی در گراف وجود نداشته باشد, آنگاه $T[i][j]=0$ خواهد بود.

- **نکته:**

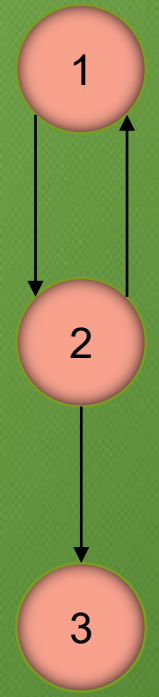
ماتریس همجواری گراف بدون جهت, متقارن است.



$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$



$$T = \begin{bmatrix} 0 & & & & & & \\ 1 & 0 & & & & & \\ 1 & 0 & 0 & & & & \\ 0 & 1 & 0 & 0 & & & \\ 0 & 1 & 0 & 0 & 0 & & \\ 0 & 0 & 1 & 0 & 0 & 0 & \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$



$$T = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

درجه گره:

با استفاده از ماتریس همجواری به سادگی می‌توان تعیین کرد که آیا بین دو گره یالی وجود دارد یا خیر.

- در گراف بدون جهت, درجهٔ گره‌ای مثل i به صورت زیر محاسبه می‌شود:

درجهٔ گره i= $\sum_{j=1}^{n} T[i][j]$

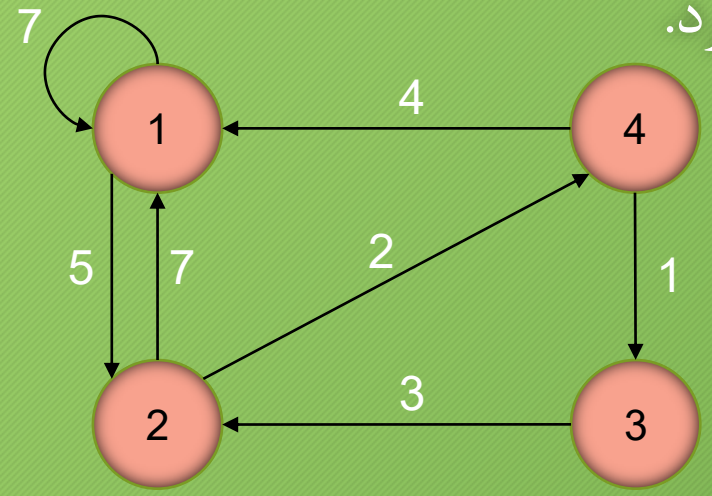
- در گراف جهت‌دار, درجهٔ گره‌ای مثل i به صورت زیر محاسبه می‌شود:

درجهٔ ورودی گره i= $\sum_{j=1}^{n} T[j][i]$

درجهٔ خروجی گره i= $\sum_{j=1}^{n} T[i][j]$

گراف وزن‌دار:

گاهی به یال‌های گراف وزن داده می‌شود و به این گراف، گراف وزن‌دار یا شبکه گفته می‌شود. این وزن می‌تواند فاصله یک گره تا گره دیگر، یا هزینه رفتن از یک گره به گره مجاور باشد. بنابراین در هنگام نمایش گراف وزن‌دار با استفاده از ماتریس همجواری، باید این وزن‌ها نگهداری شوند. در این صورت ماتریس همجواری را ماتریس وزن می‌گویند و آنرا با W نشان می‌دهند. در ماتریس وزن، وقتی بین دو گره i و j یک یال وجود دارد، در عنصر [i][j] به جای مقدار ۱، وزن آن یال قرار می‌گیرد.



$$T = \begin{bmatrix} 7 & 5 & 0 & 0 \\ 7 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{bmatrix}$$

الگوریتم کوتاه‌ترین مسیر:

برای تعیین کوتاه‌ترین مسیر بین دو گره، از دو الگوریتم زیر می‌توان استفاده کرد:

۱- الگوریتم وارشال (Warshall)
۲- الگوریتم دایکسترا (Dijkstra)

الگوریتم وارشال برای تعیین ماتریس کوتاه‌ترین مسیر:

برای یافتن کوتاه‌ترین مسیر بین دو گره با الگوریتم وارشال می‌توان به صورت زیر عمل کرد:

$$q_k[i][j] = \min(q_{k-1}[i]]j] \ , \ q_{k-1}[i][k] + q_{k-1}[k][j])$$

ماتریس اولیه q_0 برابر با ماتریس وزن است, با این تفاوت که در آن, به جای صفر, مقدار ∞ قرار می‌گیرد.

آخرین ماتریس بدست‌آمده, ماتریس کوتاه‌ترین مسیر خواهد بود.

```
int i, j, k;
int q[M][M];
int spath[M][M];

for (i=0;i<M;i++)
    for (j=0;j<M;j++)
        spath[i][j] = w[i][j];

for (k=0;k<M;k++)
{
    for (i=0;i<M;i++)
        for (j=0;j<M;j++)
            q[i][j] = min(spath[i][j] , spath[i][k] + spath[k][j]);

    for (i=0;i<M;i++)
        for (j=0;j<M;j++)
            spath[i][j] = q[i][j];
}
```

الگوریتم دایکسترا برای تعیین ماتریس کوتاه‌ترین مسیر:

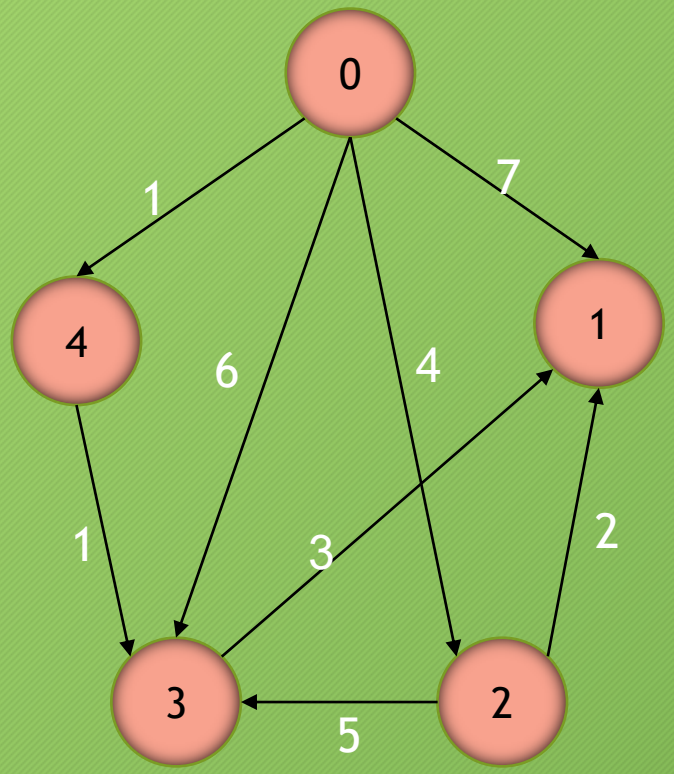
برای محاسبه کوتاه‌ترین مسیر با الگوریتم دایکسترا به صورت زیر عمل می‌شود:

W: ماتریس وزن

آرایه distance

S: مجموعه‌ای از گره‌ها که کوتاه‌ترین مسیر آنها مشخص است.

Y: مجموعه‌ای از گره‌ها غیر از آنهایی که در S هستند.



طول	مسیر	مقصد	مبدأ
1	0,4	4	0
2	0,4,3	3	0
4	0,2	2	0
5	0,4,3,1	1	0

	0	1	2	3	4
distance:	0				

$S = \{0\}$
$Y = \{1,2,3,4\}$
$\min \{d_0+w_{01} , d_0+w_{02} , d_0+w_{03} , d_0+w_{04}\} =$
$\min \{0+7 , 0+4 , 0+6 , 0+1\} = \min \{7 , 4 , 6 , 1\} = 1$

1

	0	1	2	3	4
distance:	0		4	2	1

$S = \{0,4,3,2\}$
$Y = \{1\}$
$\min \{d_0+w_{01} , d_2+w_{21} , d_3+w_{31}\} =$
$\min \{0+7 , 4+2 , 2+3\} = \min \{7 , 6 , 5\} = 5$

4

	0	1	2	3	4
distance:	0				1

$S = \{0,4\}$
$Y = \{1,2,3\}$
$\min \{d_0+w_{01} , d_0+w_{02} , d_0+w_{03} , d_{04}+w_{43}\} =$
$\min \{0+7 , 0+4 , 0+6 , 1+1\} = \min \{7 , 4 , 6 , 2\} = 2$

2

	0	1	2	3	4
distance:	0	5	4	2	1

$S = \{0,1,2,3,4\}$
$Y = \{\}$

5

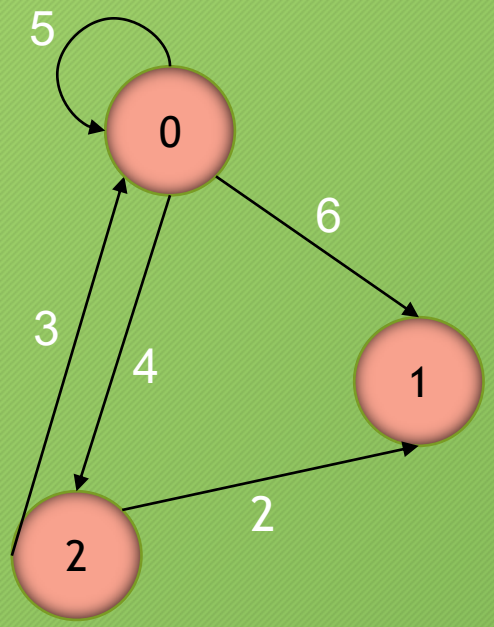
	0	1	2	3	4
distance:	0			2	1

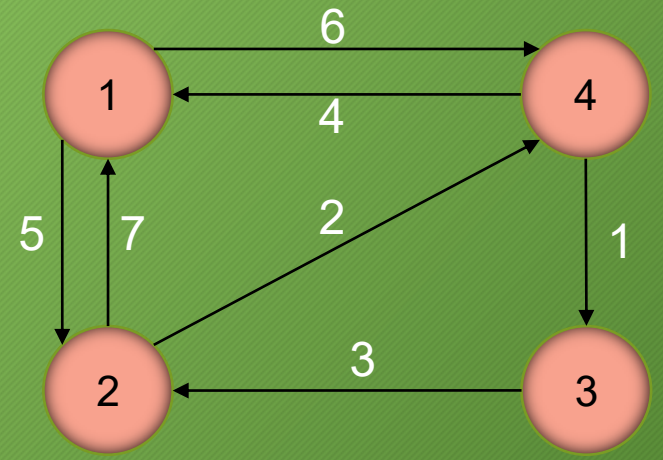
$S = \{0,4,3\}$
$Y = \{1,2\}$
$\min \{d_0+w_{01} , d_0+w_{02} , d_3+w_{31}\} =$
$\min \{0+7 , 0+4 , 2+3\} = \min \{7 , 4 , 5\} = 4$

3

تمرین:

ماتریس همجواری را برای دو گراف زیر نوشته و کوتاه‌ترین مسیر را با دو الگوریتم وارشال و دایکسترا محاسبه کنید.





جلسه پانزدهم:

گراف‌ها

آنچه در این جلسه می‌خوانید:

۱- گراف‌های متصل و غیر متصل

۲- گراف چرخه‌دار و بی‌چرخه

۳- درخت‌های پوشا

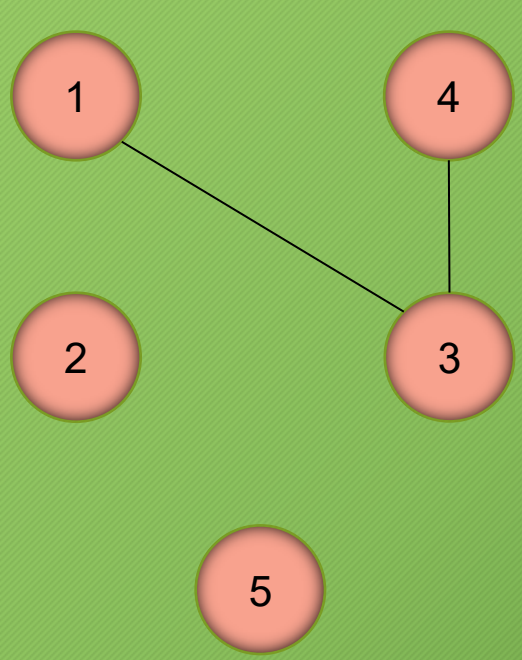
۴- درخت پوشای کمینه

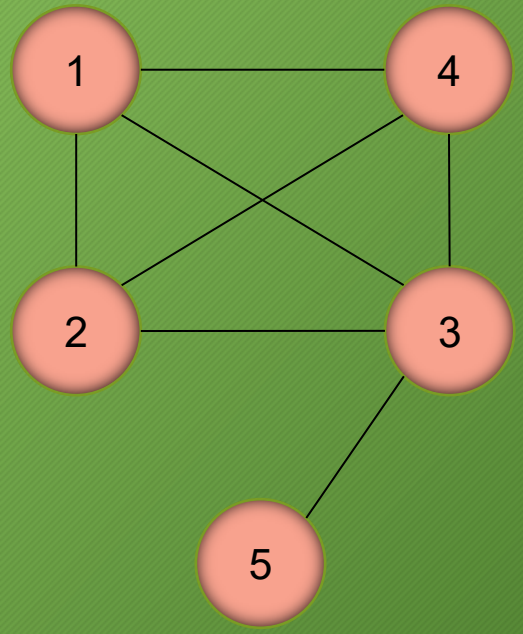
۵- الگوریتم وارشال برای ساخت درخت پوشای کمینه

۶- الگوریتم پریم برای ساخت درخت پوشای کمینه

گراف‌های متصل و غیر متصل:

گراف متصل گرافی است که در آن می‌توان از هر گره‌ای به گره دیگر رسید.

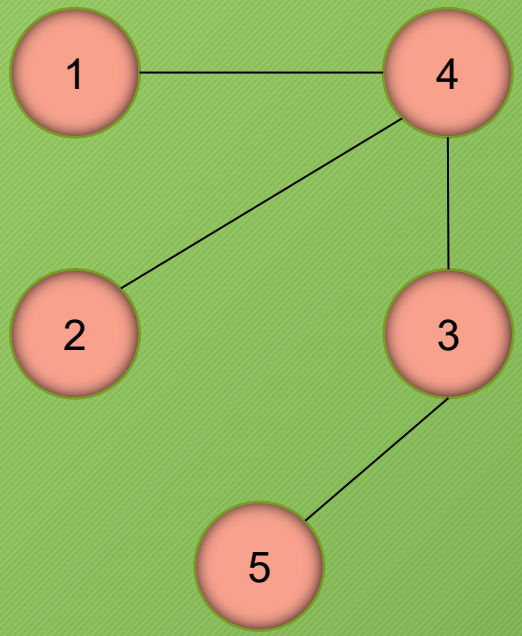


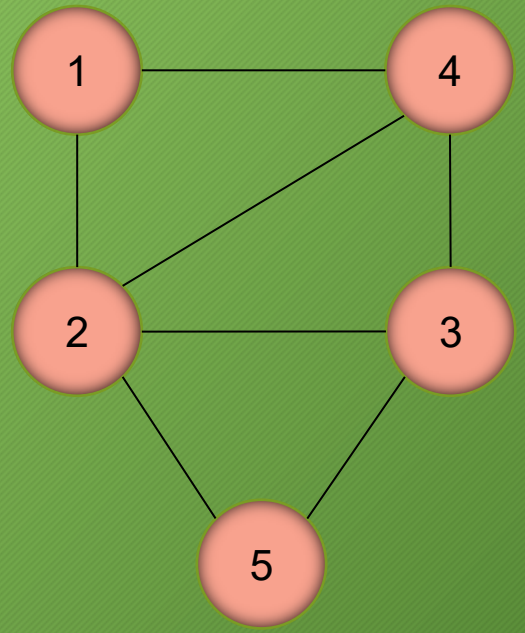


گراف چرخه‌دار و بی‌چرخه:

اگر در گرافی (بدون جهت یا جهت‌دار) مسیری گره‌ای را به خودش وصل کند, آن گراف را چرخه‌دار می‌گویند و اگر چنین مسیری وجود نداشته باشد, آنرا بی‌چرخه می‌گویند.

درخت, یک گراف بدون جهت, متصل و بی‌چرخه است.



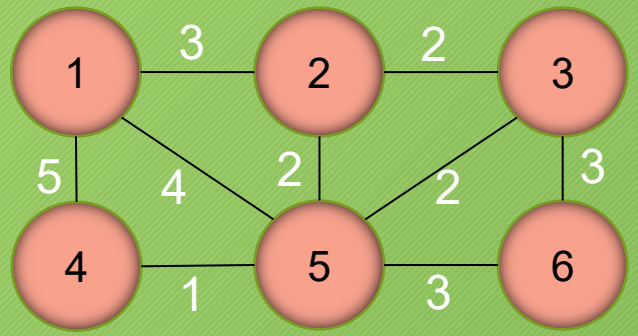


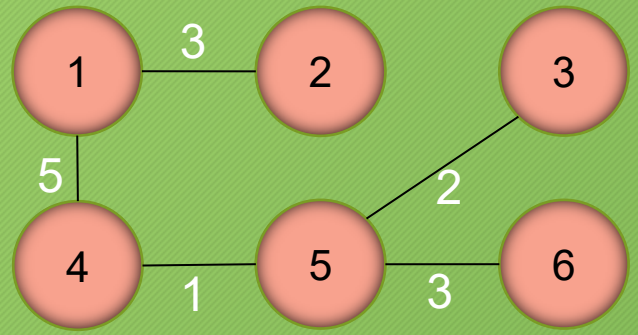
درخت‌های پوشا:

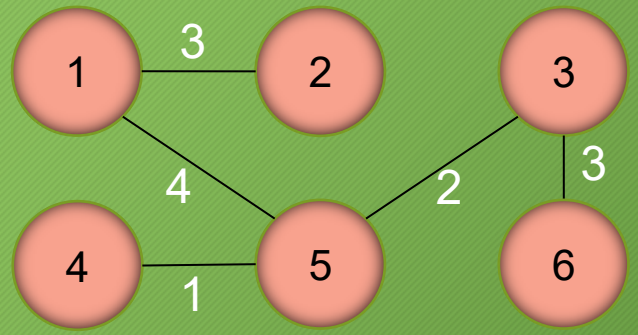
درخت پوشا، زیر گراف متصلی است که حاوی تمام گره‌های آن گراف بوده و درخت باشد (فاقد چرخه باشد).

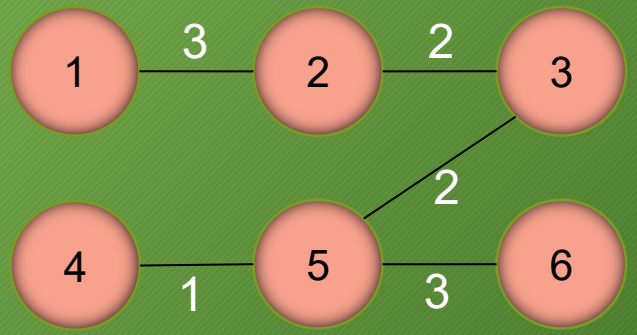
درخت پوشای کمینه، درخت پوشایی است که حداقل وزن را داشته باشد.

نکته:
یک گراف ممکن است بیش از یک درخت پوشای کمینه داشته باشد.









الگوریتم وارشال برای ساخت درخت پوشای کمینه:

در این الگوریتم, درخت پوشای کمینه, یال به یال ساخته می‌شود. یال‌های گراف بر حسب وزن به ترتیب صعودی مرتب می‌شوند. یال جدید وقتی به درخت T اضافه می‌شود که با یال‌های موجود در T چرخه ایجاد نکند.

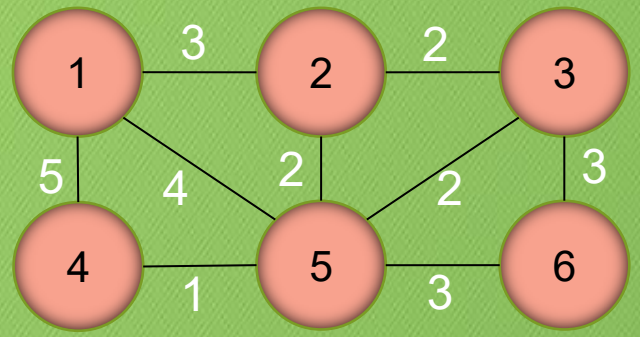
- **الگوریتم وارشال:**

۱- یال‌ها به ترتیب صعودی, از کمترین وزن به بیشترین وزن مرتب می‌شوند.

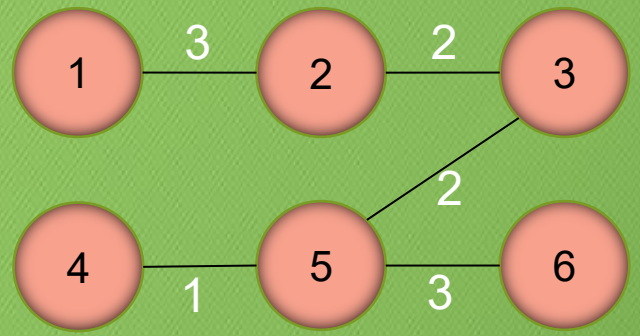
۲- یال‌های مرتب شده به ترتیب در یک مجموعه (درخت T) اضافه می‌شوند. اگر با افزودن این یال, چرخه ایجاد شود, از آن یال صرفنظر شده و یال بعدی بررسی می‌شود.

۳- مرحله ۲ آنقدر تکرار می‌گردد تا به انتهای لیست یال‌های مرتب شده برسد.

مثال



	وزن	یال
اضافه می‌شود	۱	(4,5)
اضافه می‌شود	۲	(5,3)
اضافه می‌شود	۲	(2,3)
اضافه نمی‌شود (چرخه)	۲	(5,2)
اضافه می‌شود	۳	(1,2)
اضافه می‌شود	۳	(5,6)
اضافه نمی‌شود (چرخه)	۳	(3,6)
اضافه نمی‌شود (چرخه)	۴	(1,5)
اضافه نمی‌شود (چرخه)	۵	(1,4)

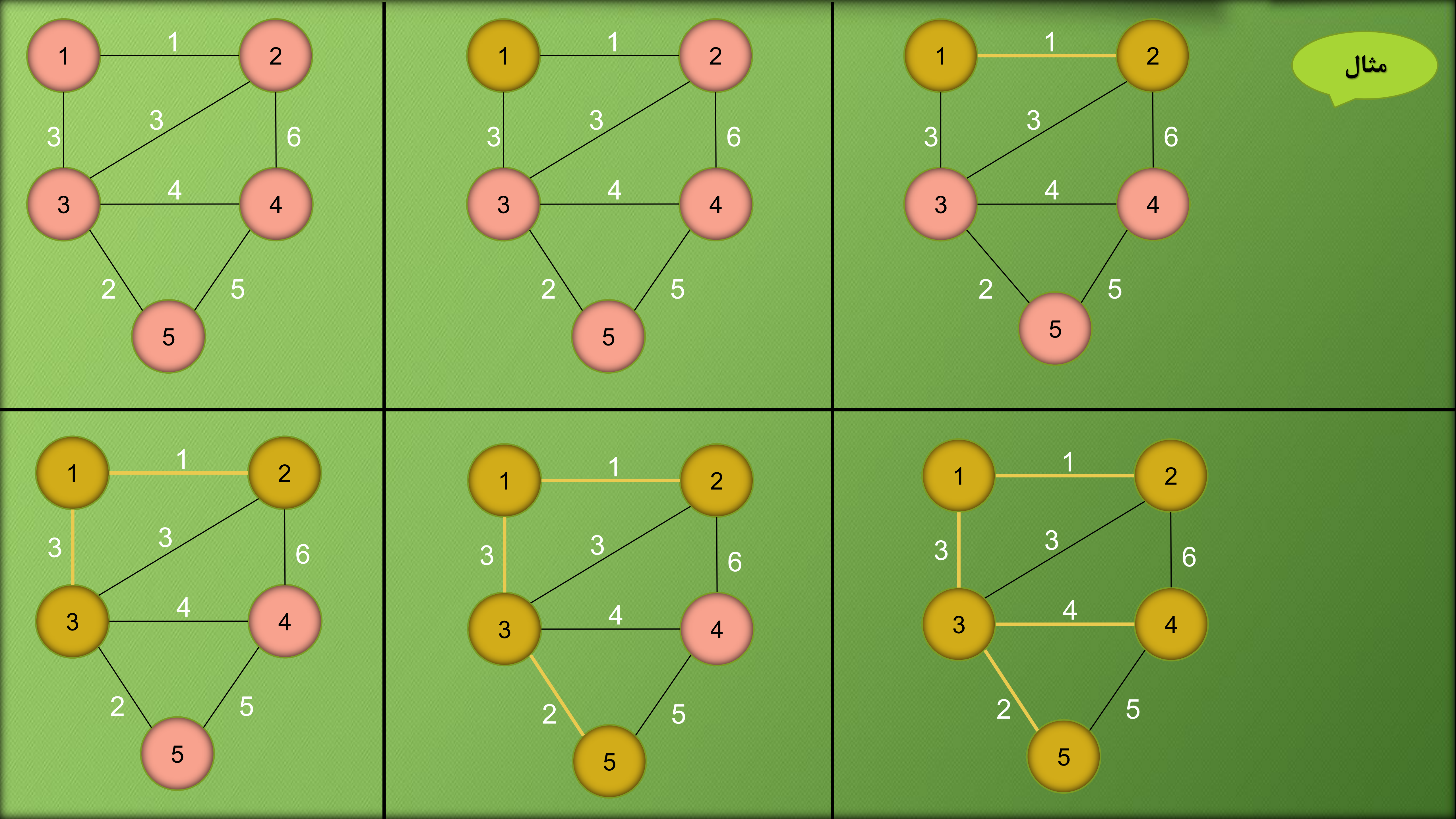


الگوریتم پریم برای ساخت درخت پوشای کمینه:

در این الگوریتم, V مجموعه‌ای از رأس‌های گراف است.

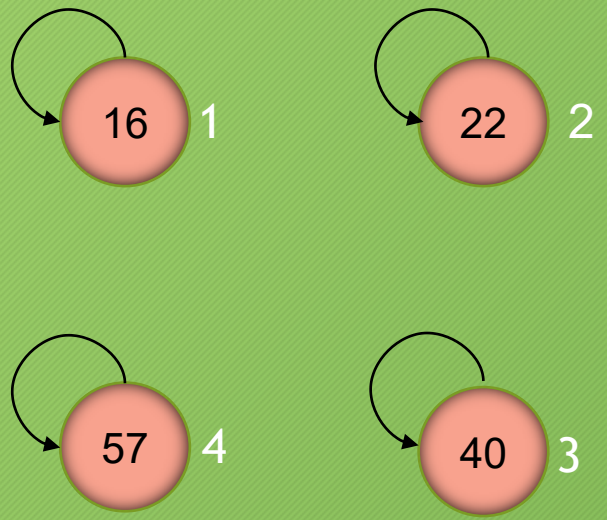
این الگوریتم با مجموعه‌ای تهی از یال‌ها به نام F و زیرمجموعه‌ای از رئوس به نام T آغاز می‌شود. زیرمجموعه T حاوی یک رأس دلخواه است.

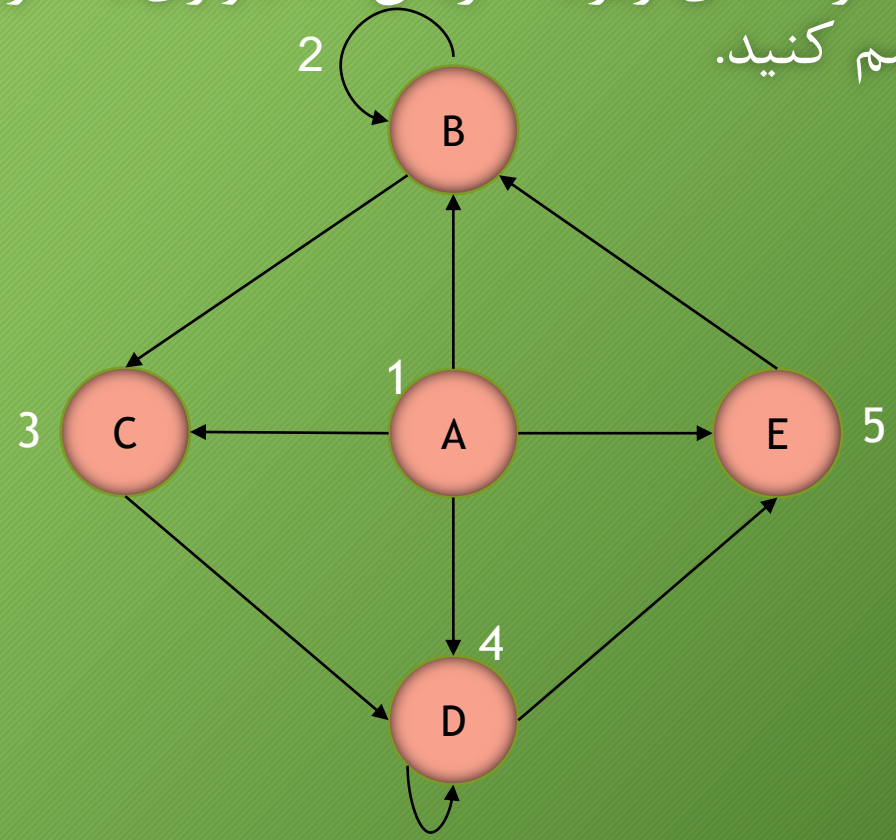
به عنوان مقدار اولیه, یک رأس در نظر گرفته می‌شود. نزدیک‌ترین رأس به T رأسی در V-T است که توسط یالی با وزن کمینه در T متصل است.


مثال

تمرین:

برای گراف‌های زیر، ماتریس همجواری، ماتریس داده‌ها،لیست همجواری و لیست یال‌ها را رسم کنید.





تمرین:

با توجه به ماتریس همجواری زیر، گراف جهت‌دار متناظر با آن را رسم کنید.

$$T = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

پایان