ՀԱՅԱՍՏԱՆԻ ՀԱՆՐԱՊԵՏՈՒԹՅԱՆ
ԳԻՏՈՒԹՅԱՆ ԵՎ ԿՐԹՈՒԹՅԱՆ ՆԱԽԱՐԱՐՈՒԹՅՈՒՆ

Հայաստանի Պետական ճարտարագիտական Համալսարան
(Պոլիտեխնիկ)

Տեղեկատվական անվտանգության և
ծրագրային ապահովման ամբիոն

ՀԱՇՎՈՂԱԿԱՆ ՏԵԽՆԻԿԱՅԻ ԵՎ ԻՆՖՈՐՄԱՏԻԿԱՅԻ ՄԱԹԵՄԱՏԻԿԱԿԱՆ ՀԻՄՈՒՆՔՆԵՐ

Լաբորատոր աշխատանքների
մեթոդական ցուցումներ

ԵՐԵՎԱՆ 2014

ՀՏԴ

Կազմող՝ Ե.Ո.Ալավերդյան

Հաշվողական տեխնիկայի և ինֆորմատիկայի մաթեմատիկական հիմունքներ:
Լաբորատոր աշխատանքների մեթոդական ցուցումներ:
Հայաստանի պետական ճարտարագիտական համալսարան (Պոլիտեխնիկ): Երևան, 2014 թ: 40 էջ:

Լաբորատոր աշխատանքների նպատակն է՝ խորացնել «Հաշվողական տեխնիկայի և ինֆորմացիայի մաթեմատիկական հիմունքներ» դասընթացից ուսանողի ստացած տեսական գիտելիքները և գործնական հմտություններ ձևավորել դիսկրետ հանրահաշվական կառուցվածքների ստեղծման և մշակման ալգորիթմական տեխնիկայի ոլորտում՝ օգտագործելով C/C++ լեզուների ծրագրավորման միջավայրը:

Գրախոս՝ տ.գ.թ.,դոցենտ Ռ.Գ.Հակոբյան
Խմբագիր՝ Ն.Խաչատրյան

Ընդհանուր տեղեկություններ

«Հաշվողական տեխնիկայի և ինֆորմատիկայի մաթեմատիկական հիմունքներ» առարկայի լաբորատոր աշխատանքների շրջանակներում ուսումնասիրվում են դիսկրետ հանրահաշվական կառուցվածքների առաջադրման և դրանց համակարգչային ներկայացման եղանակները: Վերջինս լայն հնարավորություններ է ընձեռում ուսանողին զարգացնել դիտարկվող կառուցվածքներին վերաբերող խնդիրների ծրագրավորման հմտությունները:

Մեթոդական ցուցումներն արված են C++ լեզվով ծրագրերի ստեղծման միջոցով՝ օգտագործելով C++ ստանդարտ գրադարանը: Որոշ առաջադրանքների յուրացման և ինքնուրույն մշակման նպատակով բերված են փսեվդոկոդեր: Մեթոդական ցուցումներում տրված են բավարար քանակի խնդիրներ, իսկ յուրաքանչյուր նոր թեման զուգորդված է ներածական բացատրություններով:

Առաջադրված է տասնչորս լաբորատոր աշխատանք, որոնք ներառում են տվյալ դասընթացի բոլոր թեմաները:

Լաբորատոր աշխատանք N 1

Լաբորատոր աշխատանքի նպատակն է՝ ուսումնասիրել բազմության տարրերի համակարգչային առաջադրման եղանակները՝ ներառյալ պատահական տարրերի գեներացումը տրված միջակայքում: Քանի որ ըստ սահմանման, բազմությունը կրկնվող տարրեր չունի, ապա վերջիններիս հետ գործողությունների իրականացման պարզեցման նպատակով առավել հարմար է կիրառել այսպես կոչված՝ «բիթային տողերով կոդավորման եղանակը», որը ներկայացված է ստորև:

Քայլ 1. Տրված բազմության տարրերը նախապես կարգավորվում են աճման կարգով, որի արդյունքում ակնհայտորեն պարզվում է բնական թվերի այն միջակայքը, որին այդ տարրերը պատկանում են: Նշենք, որ առաջադրված բազմության տարրերի որևէ լեզվի այբուբենին պատկանելությունը ոչնչով չի դժվարացնում ներկայացումը, քանի որ ամեն մի տառին համապատասխանության մեջ կարելի է դնել որևէ բնական թիվ՝ այբբենական կարգը փոխարինելով թվային առանցքի կարգավորված հաջորդականությամբ:

Քայլ 2. Վերոհիշյալ միջակայքի բոլոր տարրերը հայտարարվում են որպես համընդհանուր (ունիվերսալ) բազմության տարրեր:

Քայլ 3. Առաջադրված բազմության համար գեներացվում է բիթային տող՝ հետևյալ սկզբունքով.

- հայտարարվում են միաչափ զանգվածներ, որոնց չափը նույնն է, ինչ որ տրված ենթատեքստում համընդհանուր բազմության տարրերի քանակը, իսկ արժեքավորումը կատարվում է 1-երով կամ 0-ներով՝ կախված տրված բազմությանը տվյալ տարրի պատկանելությունից: Օրինակ, եթե համընդհանուր բազմությունը ներկայացնում է տասական թվանիշերի բազմությունը, ապա $A=\{2,4,7,9\}$ բազմությանը համապատասխանող բիթային տողը կունենա հետևյալ տեսքը՝ 0010100101, որի երկարությունը 10 է, քանի որ տասական թվանիշերի քանակը նույնպես 10 է:

Քայլ 4. Առաջադրված բազմությունների բիթային տողերի նկատմամբ կիրառվում են տրամաբանական գործողություններ՝ ըստ խնդրի պահանջի, որոնք պարզապես մեկ գործողության շնորհիվ արտածում կամ գրանցում են արդյունարար բազմութան տարրերը՝ խուսափելով տրված բազմությունների տարրերի շրջափուլային բազմակի դիտարկումներից:

Ստորև բերված ծրագրային մարմինը ներկայացնում է տրված երեք բազմությունների հատումը և միավորումը՝ շրջանցելով տրված բազմությունների տարրերը միայն մեկ անգամ և այն էլ՝ միաժամանակ:

```
void operate(int* Univ, int* set1, int* set2, int* set3, int& size)
{
     cout << ”The given sets intersection elements are:\n ”;
      for ( int i = 0 ; i < size ; i++ )
      {
         if ( *( set1 + i ) && *(set2 + i ) && *( set3 + i ) )
         cout << *( Univ + i ) <<’ ‘;
       }
       cout << endl << endl ;
      cout << ” The given sets union elements are: \n ” ;
      for ( int i = 0 ; i < size ; i ++ )
      {
        if ( *( set1 + i ) || * (set2 + i ) || * (set3 + i ) )
        cout << *( Univ + i ) <<’ ‘;
       }
      cout << endl <<endl;
}
```

Վերոհիշյալ ծրագրային մարմնում Univ պարամետրը ներկայացնում է համընդհանուր բազմությունը:

Առաջադրանքներ

1.1. Ստեղծել հիշողության դինամիկ տեղաբաշխմամբ միաչափ զանգված, արժեքավորել այն [-10;67] միջակայքի 25 պատահական ամբողջ թվերով և արտածել այդ բազմությունների բիթային տողերը

համապատասխան երկարությամբ: Կրկնել խնդիրը որևէ լեզվի այբուբենի տառերի առաջադրման համար: Կիրառել պատահական թվերի կրկնությունների բացառման որևէ ստանդարտ ալգորիթմ:

1.2. Գրանցել տրված երկու բազմությունների տարբերությունը, համաչափ տարբերությունը և լրացում բազմությունները:
1.3. Պարզել տրված բազմությունների մեկը մյուսի ենթաբազմություն լինելը:
1.4. Արտածել տրված բազմությունը և դրա բոլոր ենթաբազմությունները:
1.5. Արտածել տրված երկու և ավելի բազմությունների դեկարտյան արտադրյալի տարրերը՝ համապատասխանաբար զույգերի, եռյակների, քառյակների և այլն տեսքով:

Լաբորատոր աշխատանք N 2

Լաբորատոր աշխատանքի նպատակն է՝ տրված բազմությունների հիման վրա ստեղծել բինար հարաբերությունների մոդելներ, որոնք, մասնավորապես, ընկած են տվյալների հենքերի նախագծման հիմքում: Օրինակ՝ հարաբերություններ ծառայողների և անհատական հեռախոսահամարների միջև, ամբողջ թվերի և իրենց բաժանարարների միջև, բաղդատելի թվերի միջև, ամբողջ թվի և նրան կոնգրուենտ modո թվերի միջև և այլն:

Բինար հարաբերությունների մոդելը կարելի է կիրառել՝ պարզելու համար, օրինակ, հեռահաղորդակցությունների առկայությունը տրված քաղաքների զույգերի միջև. օրինակ՝ թռիչքուղիներ, բանկային գործառույթներ, համացանց և այլն:

Տրված բազմությունների տարրերի միջև բինար հարաբերությունների ստեղծման եղանակը բազմության տարրերի կարգավոր զույգերի ձևավորումն ու թվարկումն է, որը վերլուծությունների համար որոշակի բարդություններ

ու անհարմարություններ է ստեղծում: Այս լաբորատոր աշխատանքի շրջանակում առաջարկվում է հարաբերութուններ համակարգչային ներկայացման այսպես կոչված՝ «բինար մատրիցների» մեթոդը, որը բերված է ստորև:

Ենթադրենք՝ R-ը հարաբերություն է $A=\{a_1, a_2, \dots, a_m\}$ և $B=\{b_1, b_2, \dots, b_n\}$ բազմությունների միջև: Այդ դեպքում R հարաբերությունը կարելի է ներկայացնել $M=m_{ij}$ բինար մատրիցի տեսքով, որտեղ

$$m_{ij}=\begin{cases}1, & \text{եթե } (a_i,b_j)\in R,\\ 0, & \text{եթե } (a_i,b_j)\notin R\end{cases}$$

Առաջադրանքներ

2.1.Գեներացնել համապատասխան չափերի պատահական բինար մատրից և դիտարկելով այն որպես տողերի և սյուների բինար հարաբերություն՝ որակել վերջինս ըստ հետևյալ չափանիշների.

- անդրադարձ, ոչ անդրադարձ, հակաանդրադարձ
- համաչափ, ոչ համաչափ, հակահամաչափ
- փոխանցելի, ոչ փոխանցելի

2.2.Մուտքագրել $\{1,2,3,4,5\}$ բազմության վրա սահմանված բինար հարաբերության կարգավոր զույգերը և պատկերել դրանք բինար մատրիցի տեսքով:

ա) { (1,1),(3,4),(5,3),(4,3),(5,5)}

բ) {(1,2),(2,2),(3,3),(3,1),(5,1),(3,4),(5,2)}

գ) օգտագործողին հնարավորություն տալ մուտքագրել հարկավոր քանակի զույգեր՝ ընդսմին բացառելով կրկնությունները

2.3.Թվարկել ստորև բերված մատրիցներով ներկայացված $\{1,2,3\}$ բազմության վրա սահմանված բինար հարաբերությունների կարգավոր զույգերը:

ա) $\begin{matrix}1&0&0\\0&1&1\\1&1&1\end{matrix}$ բ) $\begin{matrix}1&0&1\\0&0&1\\1&0&1\end{matrix}$ գ) $\begin{matrix}1&0&1\\1&1&1\\1&1&0\end{matrix}$

2.4. Ստեղծել պատահականորեն գեներացված երկու բինար մատրից և դիտարկելով դրանք որպես հարաբերություններ սահմանված միևնույն բազմության վրա՝ արտածել երրորդ բինար մատրիցը, որը համապատասխանում է տրված երկու հարաբերությունների՝

- միավորմանը և հատմանը,
- կոմպոզիցիային՝ նաև փոխատեղմամբ,
- համաչափ տարբերությանը:

2.5.Տրված բինար մատրիցը դիտարկելով որպես հարաբերություն՝ արտածել հարաբերությունների տարբեր աստիճաններն ըստ օգտագործողի նախասիրության:

Լաբորատոր աշխատանք N 3

Լաբորատոր աշխատանքի նպատակն է՝ ուսանողի մոտ զարգացնել տրված բազմության տարրերի այսպես կոչված «համարակալման» ծրագրավորման տեխնիկան: Տարրերի համարակալումը կամ որ նույնն է՝ տարրերի քանակի հաշվումը, կոմբինատորիկայի կարևոր մասն է, որի գործնական կիրառությունն ուղղակի առօրեական է: Օրինակ, այդպիսի հաշվարկը թույլ է տալիս պարզել որևէ ալգորիթմի բարդության աստիճանը կամ պարզել՝ արդյոք հեռախոսահամարների կամ անհատական քարտերի նշված քանակները բավարարում են պահանջարկը և այլն:

Նշված լաբորատոր աշխատանքի սահմաններում արտածվում են տրված բազմության տարրերից կարգավորված/չկարգավորված ընտրություններ, բառեր, կարգավորություններ/տեղափոխություններ և զուգորդություններ՝ կրկնություններով կամ առանց կրկնությունների:

Տեղափոխությունների գեներացման սկզբունքը հետևյալն է. տրված է բազմության տարրերի նախնական դասավորությունը, ամեն մի հաջորդ տեղափոխությունը ձևավորվում է որպես այբբենական կարգով ավելի աջ (թվերի պարագայում՝ հաջորդ տեղափոխության ավելի մեծ “թվային” արժեքով)՝ միաժամանակ համոզվելով, որ

ձևավորված հարևան երկու նմուշների միջև բացակա նմուշ չկա: Ծրագրի փսեվդոկոդը ներկայացված է ստորև:

Ալգորիթմ 3.1. Տեղափոխությունների գեներացիա

Next_Perm($a_{1,}a_2 \dots a_{n,}$: permutation of {1,2,…,n} not equal to $n, n-1, \dots, 2, 1$)

{

$j \coloneqq n-1$

while $\boldsymbol{a_j > a_{j+1}}$

$j \coloneqq j-1$

(այստեղ j –ն ամենաաջ ինդեքսն է այնպես, որ $\boldsymbol{a_j < a_{j+1}}$)

$k \coloneqq n$

while $\boldsymbol{a_j > a_k}$

$k \coloneqq k-1$

(այստեղ $\boldsymbol{a_k}$ –ն $\boldsymbol{a_j}$-ից աջ և նրանից մեծ նվազագույն թիվն է)

փոխատեղել $\boldsymbol{a_j}$-ն և $\boldsymbol{a_k}$ -ն:

$k \coloneqq n$

s $\coloneqq j+1$

while $\boldsymbol{r > s}$

{

փոխատեղել $\boldsymbol{a_r}$-ը և $\boldsymbol{a_s}$ –ը

r $\coloneqq r-1$

s $\coloneqq s+1$

}

}

Ջուգորդություններն առանց կրկնությունների գեներացնելու և արտածելու սկզբունքը հետևյալն է. բազմության տարրերի հավաքածուն ներկայացվում է բիթային տողի տեսքով, որում 1-երի առկայությունը ցույց է տալիս բազմության համապատասխան տարրի առկայությունն ընտրույթի կազմում, որից հետո կատարելով երկուական գումարում կամ հանում, գեներացվում է հաջորդ կամ

նախորդ նմուշին համապատասխանող երկուական կոդը, ըստ որի արտածվում է տարրերի համապատասխան ընտրույթը:

Ալգորիթմ 3.2. Զուգորդությունների գեներացիա

Comb_next_Selection($b_{n-1}b_{n-2}b_1b_0$: $bitString \neq 11\ldots11$)
{
    i:=0
    while $b_i = 1$ { $b_i := 0$ i :=i+1 } $b_i = 1$
}

Առաջադրանքներ

3.1.Արտածել {1,2,3,4} բազմության տարրերի բոլոր տեղափոխությունները և դրանց քանակը:
3.2.Արտածել {1,2,3,4} բազմության տարրերի բոլոր (5,3) կարգավորությունները և դրանց քանակը:
3.3.Հաշվել հետևյալ բանաձևերի արդյունքները և արտածել դրանց համապատասխանող ընտրույթները.

- P(6,3), P(6,5), P(8,8)
- C(6,3), C(6,5), C(8,7)

3.4.Ստեղծել և արտածել Պասկալի եռանկյան տարրերն ըստ օգտագործողի նախասիրության:
3.5.Արտածել չորս տառանոց այբուբենի բոլոր երեք տառանոց բառերը և դրանց քանակը:
3.6.Արտածել 9 բիթերից բաղկացած և ճիշտ 5 հատ 1 պարունակող բիթային տողերը և դրանց քանակը:
3.7.Արտածել կանոնական և կապույտ ճանապարհներ տրված ուղղանկյուն ցանցում:
3.8.Տրված դրամանիշերի հավաքածուի համար մուտքագրել գումարի չափ և արտածել այդ գումարն ապահովող բոլոր հնարավոր զուգորդությունները և դրանց քանակը:
3.9. Մուտքագրելով m և n պարամետրերի արժեքները՝ արտածել n-ից m-ական կոդը:

Լաբորատոր աշխատանք N 4

Լաբորատոր աշխատանքի նպատակն է՝ ծանոթանալ գրաֆների համակարգչային ներկայացմանը և նախագծված տվյալների կառուցվածքներում տարատեսակ վերլուծություններ իրականացմանը:

Պարզ գրաֆների ներկայացման եղանակներից մեկը կողերի թվարկումն է (գրաֆի բինար հարաբերության կարգավոր զույգերի ցանկ): Մյուս՝ ավելի նպատակահարմար եղանակը, **հարևանության** ցուցակի կիրառությունն է, որը ներկայացնում է իրար հարևան գագաթների զույգերը: Տրված $G=(V,E)$ պարզ գրաֆի համար, որում գագաթների քանակը՝ $|V|=n$, կառուցվում է $n\times n$ չափի հարևանության A_{ij} բինար մատրից այնպես, որ մատրիցի (i,j) տարրն ընդունում է 1 արժեք, եթե v_i և v_j գագաթները հարևան են: Այլ կերպ ասած, եթե $A=[a_{ij}]$-ն G գրաֆի հարևանության մատրիցն է, ապա՝

$$a_{ij}=\begin{cases}1, & \textit{եթե } (v_i,v_j)\in G\\ 0, & \textit{հակառակ դեպքում}\end{cases}$$

Պարզ գրաֆների ներկայացման մյուս եղանակը **կցության** մատրիցի կառուցումն է: Տրված $G=(V,E):|V|=n,|E|=m$ ոչ ուղղորդված գրաֆի համար կցության $n\times m$ չափի բինար $M=[m_{ij}]$-մատրիցը կառուցվում է հետևյալ սկզբունքով.

$$m_{ij}=\begin{cases}1, & \textit{եթե } e_j-\textit{ն կից է } v_i-\textit{ին}\\ 0, & \textit{հակառակ դեպքում}\end{cases}$$

Օրինակ՝

```
class Graph {
private:
    bool** adjacencyMatrix;
    int vertexCount;
public:
```

```
    Graph ( int vertexCount )
    {
        this -> vertexCount = vertexCount;
        adjacencyMatrix = new bool* [ vertexCount ] ;

        for ( int i = 0 ;  i < vertexCount ;  i  ++)
        {
            adjacencyMatrix [ i ] = new bool [ vertexCount ] ;

            for ( int j =0 ;  j < vertexCount ; j ++)
            {
                adjacencyMatrix [  i  ] [ j ] = false;
            }
        }
    }
 void addEdge ( int i, int j)
{
      if ( i >= 0 && i < vertexCount && j > 0 && j < vertexCount )
     {
        adjacencyMatrix [ i ] [ j ] = true;
        adjacencyMatrix [ j ] [ i ] = true;
      }
 }
 void removeEdge ( int i, int j)
{
      if ( i >= 0 && i < vertexCount && j > 0 && j < vertexCount )
     {
          adjacencyMatrix [ i ] [ j ] = false;
          adjacencyMatrix [ j ][ i ] = false;
      }
 }
  bool isEdge (int i, int j)
 {
      if (i >= 0 && i < vertexCount && j > 0 && j < vertexCount)
       {
          return adjacencyMatrix[ i ] [ j ] ;
       }
       return false;
 }
 ~Graph()
{
```

```
        for (int i = 0 ;  i < vertexCount ;  i ++)
        {
            delete [ ] adjacencyMatrix [ i ];
        }
            delete [ ] adjacencyMatrix;
    }
};
```

Առաջադրանքներ

4.1 Դիտարկելով պատահականորեն գեներացված քառակուսային բինար մատրիցը որպես ինչ-որ պարզ գրաֆի հարևանության մատրից՝ արտածել այդ գրաֆի գագաթների աստիճանների արժեքները:
4.2. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ ստուգել՝ արդյոք տրված գրաֆը կապակցված է:
4.3. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ ստուգել՝ արդյոք տրված գրաֆը լրիվ է:
4.4. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ արտածել գրաֆում տրված երկարության ճանապարհներ:
4.5. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ պարզել՝ արդյոք գագաթների մուտքագրված հաջորդականությունը ճանապարհ է այդ գրաֆում:
4.6. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ պարզել՝ արդյոք այն փսեվդոգրաֆ է:
4.7. Գեներացնել պատահական քառակուսային բինար մատրից և ստուգել՝ արդյոք այն կարող էր պարզ գրաֆ ներկայացնել:

Լաբորատոր աշխատանք N 5

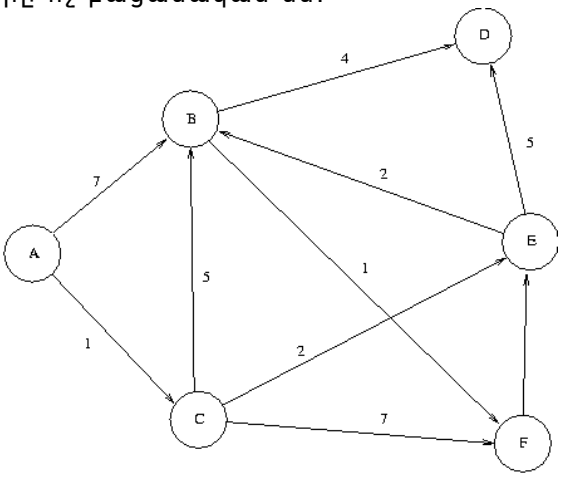
Լաբորատոր աշխատանքի նպատակն է՝ ծանոթանալ ուղղորդված գրաֆների համակարգչային ներկայացման առանձնահատկություններին:

Ուղղորդված գրաֆների պարագայում կողերը կարող են կշռավորված լինել: Այս դեպքում ամենատարածված խնդիրներից են՝

- պարզել՝ արդյոք հնարավոր է այցելել բոլոր կողերը՝ առանց կրկնելու դրանք (էյլերյան ճանապարհ),
- շրջիկ գործակալի խնդիր. ինչպես այցելել բոլոր հանգույցները նվազագույն կշռով,
- գտնել կարճագույն՝ ամենաէժան ճանապարհը տրված երկու գագաթների միջև,
- գտնել գրաֆի կմախքային ծառը, որն ունի նվազագույն կշիռ և ներառում է բոլոր գագաթները:

Նշենք, որ ուղղորդված գրաֆների գագաթների այցելությունը ակնհայտ չէ այն պարզ պատճառով, որ տրված երկու գագաթների միջև կարող են բազմաթիվ կողեր լինել՝ ի տարբերություն ծառերի, որոնցում ցանկացած երկու հանգույցների միջև կա միակ ճանապարհ:

Ստորև բերված ուղղորդված գրաֆն ունի 6 գագաթ: Դիքստրայի ալգորիթմը կարող է կիրառվել՝ հայտնաբերելու համար կարճագույն ճանապարհը տրված գագաթից դեպի մյուս գագաթներից յուրաքանչյուրը՝ քանի դեռ կողերի կշիռները ոչ բացասական են:



Լայնական շարժն առաջին հանգույցից տարածվում է գրաֆի տիրույթով: Երբ այն հասնում է որևէ չայցելած գագաթ, սկզբնական գագաթից մինչև տվյալ գագաթի միջև հեռավորությունների արժեքները գումարվում են և

պահպանվում: Եթե պարզվում է, որ հերթական գագաթն արդեն այցելվել է, ստուգվում է սկզբնական գագաթից նրա ունեցած հեռավորության արժեքը և, եթե պարզվում է, որ վերջինս նվազագույնը չէ, այսինքն՝ ընթացիկ ճանապարհն ավելի կարճ է, տվյալ գագաթի և սկզբնական գագաթի միջև հեռավորության արժեքը փոխարինվում է այդ ավելի փոքր գումարով:

Սույնով՝ A գագաթն ընդունելով որպես մեկնարկային՝ առաջին քայլում B գագաթի հեռավորությունը կարելի է համարել 7, իսկ C գագաթինը՝ 1: Երկրորդ քայլում B և C գագաթներից ելնող ճանապարհները ենթակա են հետազոտման. երբ B գագաթն այցելվում է C գագաթից, հեռավորության արժեքն ստացվում է 1+5, այնպես որ B-ի հեռավորությունն A-ից սկզբնական 7 արժեքից փոխարինվում է 6–ով: Հետագա քայլերի արդյունքում B-ի հեռավորությունը 6–ից կդառնա 5, որն էլ կներկայացնի B-ի նվազագույն վերջնական հեռավորությունը:

Տրված գրաֆի համար հետազոտությունը կավարտվի 5 քայլով, քանի որ այդ ընթացքում բոլոր ճանապարհներն անցած կլինեն: Նշենք, որ ճանապարհները չեն կարող ունենալ ավելի քան ***գագաթների քանակ-1*** կողեր:

Ծրագրային ապահովման իրականացման համար նպատակահարմար է գրաֆի կառուցվածքը ներմուծել տեղեկատու ֆայլից հետևյալ տեսքով.

```
A B 7 anon
A C 1 anon
B D 4 anon
B F 1 anon
C B 5 anon
C E 2 anon
C F 7 anon
E B 2 anon
E D 5 anon
F E 3 anon
SEARCHMODE shortest
STARTNODE  A
```

Այս գրառման մեջ “anon” պիտակը նշանակում է, որ այս դեպքում կողերը դեռևս անուններ չունեն: Ծրագրի գործարկման արդյունքում էկրանին պետք է արտածվի հետևյալը.

```
B is 5 from the source.
Shortest path is A to C to E to B
C is 1 from the source.
Shortest path is A to C
D is 8 from the source.
Shortest path is A to C to E to D
F is 6 from the source.
Shortest path is A to C to E to B to F
E is 3 from the source.
Shortest path is A to C to E
```

Գրաֆի գագաթների կարգավորությունների և այցելությունների տեսակները կարելի է գրանցել թվարկվող հաստատունների միջոցով, որոնք նպատակահարմար է հայտարարել օգտագործողի գրադարանային ֆայլի գլխամասում՝ գրաֆի դասի հայտարարման հետ համատեղ:

Ստորև ներկայացված է համապատասխան ծրագրային մարմինը:

```
enum ordering {preorder, inorder, postorder};
enum search_mode {breadth_first, depth_first, hill_climbing, best_first,
Astar, shortest_paths, min_spanning_tree};

class Search_Description
{
  public:

     ordering order;
     search_mode searching;
     string startnode;
     string goalnode;
     bool printnode;
};
```

Առաջադրանքներ

5.1.Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ պարզել՝ արդյոք տրված երկու գագաթներն իրար հարևան են:

5.2.Արտածել գրաֆի տրված երկու գագաթների միջև որևէ պարզ ճանապարհ:
5.3.Պարզել՝ արդյոք տրված գրաֆը շրջափուլ պարունակում է, թե ոչ:
5.4.Արտածել տրված գրաֆի գագաթների աստիճանների արժեքները նվազման կարգով:
5.5.Պարզել՝ արդյոք տրված գրաֆն էյլերյան է:
5.6.Պարզել՝ արդյոք տրված գրաֆը մեկուսացած գագաթ ունի, և եթե այո, ապա այդ գագաթի համար ստեղծել հարևանություն և արտածել այդ գագաթից սկիզբ առնող երկու երկարության որևէ ճանապարհ:
5.7.Պարզել՝ արդյոք տրված գրաֆը «կախազարդ» գագաթ ունի, և եթե այո, ապա արտածել այդ գագաթից սկիզբ առնող երեք երկարության որևէ ճանապարհ:
5.8.Պարզել՝ արդյոք տրված գրաֆը տարանջատ բաղադրամասեր ունի և եթե այո, ապա ներմուծել կամրջակ՝ ամբողջ գրաֆը կապակցված դարձնելու համար: Արտածել ստացված գրաֆի հարևանության մատրիցը:

Լաբորատոր աշխատանք N 6

Լաբորատոր աշխատանքի նպատակն է՝ ծանոթանալ ուղղորդված գրաֆների համակարգչային ներկայացման այլընտրանքային եղանակներին:

Գրաֆները կարող են հայտարարվել նաև ծրագրային եղանակով՝ ամեն հաջորդ քայլում ավելացնելով պահանջվող գագաթը: Ստորև ներկայացված ծրագրային մարմինը նախատեսված է 6 գագաթանոց գրաֆի ստեղծման և նրանում B գագաթից ելնող կարճագույն ճանապարհի փնտրման համար: Ծրագրի լրամշակման համար կարելի է օգտվել նախորդ լաբորատոր աշխատանքներում ձեռք բերված հմտություններից:

```
void  main ( int argc, char *argv[ ] )
{
    Graph g;
    Search_Description s;
    vector  < string > nodenames;
```

```
    nodenames . push_back ( " A " ) ;
    nodenames . push_back ( " B " ) ;
    nodenames . push_back ( " C " ) ;
    nodenames . push_back ( " D " ) ;
    nodenames . push_back ( " E " ) ;
    nodenames . push_back ( " F " ) ;

    for ( int i = 0 ; i < 6 ; i ++ )
    {
       for ( int j = i + 1 ; j < 6 ; j ++ )
       {
          g . addEdge ( nodenames [ I ] , nodenames [  j ] , i + j , " anon "  ;
       }
    }
        g . shortestpath ( " B " ) ;
}
```

Առաջադրանքներ

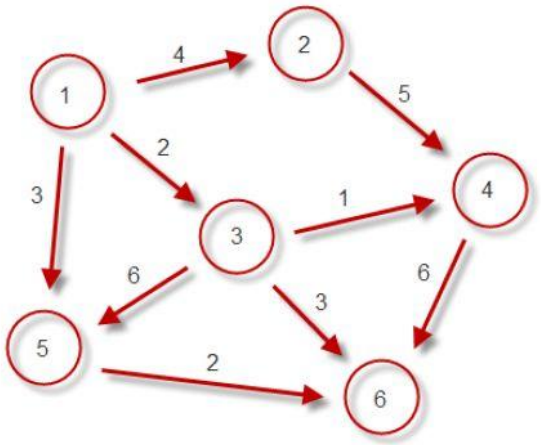
6.1. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ պարզել՝ արդյոք այն Q_3 է:
6.2. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ պարզել՝ արդյոք այն պարունակում է C_3:
6.3. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ պարզել՝ արդյոք այն պարունակում է W_3:
6.4. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ պարզել՝ արդյոք այն երկշերտ է:

Լաբորատոր աշխատանք N 7

Լաբորատոր աշխատանքի նպատակն է՝ խորացնել գրաֆների համակարգչային ներկայացման և մշակման եղանակները:

Այդ նպատակով ուսանողին առաջարկվում է դիտարկել գրաֆների վերաբերյալ ստորև ներկայացված մեկ այլ ծրագրային հավելված, որը ներկայացնում է

Կրուսկալի ալգորիթմի ծրագրային իրականացումը: Կրուսկալի ալգորիթմը տրված կապակցված գրաֆի նվազագույն կմախքային ծառի որոնման ալգորիթմն է: Արդյունքում փնտրվում է գրաֆի կողերի այնպիսի ենթաբազմություն, որը ներկայացնում է բոլոր գագաթները ներառող և բոլոր կողերի նվազագույն կշռով ծառ: Որպես օրինակ՝ դիտարկենք հետևյալ գրաֆը.



Ստորև բերված է ներկայացված գրաֆի Կրուսկալի ալգորիթմի ծրագրային իրականացումը:

```
#include<stdio.h>
#include<stdlib.h>
void printArray ( int a[ ][100] , int  n );
void AdjacencyMatrix (int  a[ ][ 100 ] ,  int n ) {
   int i , j ;
   for ( i = 0; i  <  n ;  i ++)
   {
      for ( j = 0 ; j <  i ;  j ++ )
      {
         a [ i ] [ j ] = a [ j ] [ I ] = rand ( ) % 50 ;
         if ( a [ i ] [ j ]  > 40 )
         {
            a [ i ] [ j ] = a [ j ] [ i ] = 999;
         }
```

```
        }
        a [ i ] [ i ] = 999;
    }
    printArray ( a, n ) ;
}
void printArray ( int a[ ] [ 100 ] , int n )
{
    int i , j ;
    for( i = 0 ; i < n ; i ++ )
    {
        for ( j = 0 ; j < n ; j++ )
        {
            Printf ( " % d \ t " , a [ i ] [ j ] );
        }
        printf( " \ n " );
    }
}
int root ( int v , int p[ ] )
{

     while ( p [ v ] ! =  v )
     {
         v  =  p [ v ] ;
     }

     return v;
}
 void union_ij ( int  i , int j , int p[ ] )
{
    if ( j  >  i)
    {
        p [ j ] = i;
    }
    else
    {
        p [ i ] = j;
    }
}
void kruskal ( int a[ ][ 100 ], int  n )
{
    int count,  i,  p [ 100 ],  min, j, u, v, k, t [ 100 ] [ 100 ], sum;
```

```
count = k = sum = 0;
for ( i = 0; i < n; i++)
{
   p [ i ] = i;
}

while (count < n)
{
   min = 999;
   for ( i = 0 ;  i < n ;  i ++)
   {
      for ( j = 0 ; j < n ;  j ++)
      {
         if ( a [ i ] [ j ] <  min )
         {
            min = a [ i ] [ j ] ;
            u = i;
            v = j;
         }
      }
   }

   if ( min != 999)
   {
      i = root ( u,  p );
      j = root (v,  p);
      if (i ! = j)
      {
         t [ k ] [ 0 ] = u;
         t [ k ] [ 1 ] = v;

         k ++;
         sum += min;
         union_ij ( i , j , p);
      }
   a [ u ] [ v ] = a [ v ] [ u ] = 999;

   } count +=1;
}

if ( count ! = n)
{
```

```
        printf (" spanning tree not exist \ n " ) ;
    }
    if ( count == n )
    {
        printf (" Spanning tree is \ n " ) ;
        for ( k = 0; k < n-1 ; k ++ )
        {
            printf ("  % d  ->  % d " , t [ k ] [ 0 ] , t [ k ] [ 1 ] ) ;
        }
        printf (" \ n cost =  % d  \ n " , sum ) ;
    }
}

int main()
{
    int a [ 100 ] [ 100 ], n ;
    printf ( "enter the number of vertices \ n " ) ;
    scanf ( " % d " , &n ) ;
    AdjacencyMatrix ( a,  n ) ;
    kruskal ( a,   n ) ;
    return 0;
}
```

Առաջադրանքներ

7.1. Արտածել կապակցված գրաֆի տրված երկու գագաթների միջև որևէ պարզ ճանապարհ և դրա երկարությունը:
7.2. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ պարզել՝ արդյոք գրաֆն աստղածև ցանց է ներկայացնում/պարունակում:
7.3. Օգտվելով տրված գրաֆի հարևանության մատրիցից՝ պարզել՝ արդյոք գրաֆն օղակածև ցանց է ներկայացնում/պարունակում:
7.4. Տրված գրաֆի հարևանության մատրիցի հիման վրա ստեղծել ևս երկու բինար մատրից այնպես, որ վերջիններս ներկայացնեն տրված գրաֆի երկու տարբեր ենթագրաֆներ:

Լաբորատոր աշխատանք N 8

Լաբորատոր աշխատանքի նպատակն է՝ դիտարկել և ըմբռնել ծրագրային հավելվածի իմաստը և կատարել հարկավոր եզրահանգումներ:

Տրված են գրաֆի հետևյալ պարամետրերը.

- int size=10, որը գրաֆի կառուցվածքը ներկայացնող միաչափ զանգվածի չափն է:
- adj_matrix[s][u] –ը ներկայացնում է գրաֆի հարևանության մատրիցը,
- distance[s][u] –ն ներկայացնում է գրաֆի կողերի կշիռները,
- precede[s][u] –ն ներկայացնում է քայլ դեպի նախորդ գագաթ,
- all() ֆունկցիան ստուգում է գրաֆը ներկայացնող զանգվածի տարրերի տրամաբանական արժեքները. տվյալ դեպքում դրանք կեղծ են:
- INT_MAX –ը տրված գրաֆի կողերի առավելագույն կշիռն է:

Ստորև ներկայացված ծրագրային մարմինը ներկայացնում է տրված գրաֆի երկու գագաթների միջև կարճագույն ճանապարհի փնտրման ալգորիթմի իրականացումը:

```
void short_path_graph( int s, int size)
{
    bool permanent [ size ] ;
    for ( int x = 0; x < size; x ++)
    {
      permanent [ x ] = false;
    }

    for (int i = 1; i < size; i ++)
    {
        distance [ s ] [ i ] = INT_MAX;
```

```
}
distance [ s ] [ s ] = 0 ;

while ( ! all ( permanent ) )
{
     int u = 0;
     int v = 0;

     for ( int i = 1; i < size; ++i)
     {
          for ( int y = 0; y < size; y++)
          {
            if ( adj_matrix [ s ][ i ] && adj_matrix[ s ][ y])
               {
                if ( distance[ s ] [ I ] < distance [ s ] [y]
                    &&
                   ! permanent [ s ] )
                   {
                        v = distance [ s ] [ i ] ;
                        y ++ ;
                   }
               }
          }
     }

     permanent [ v ] = true ;

     for ( u = 0 ; u < size ; ++ u )
     {
          if ( adj_matrix [ v ] [ u ] )
          {
               if ( distance [ s ] [ u ] > distance [ s ] [ v ] +
               distance [ v ] [ u ] )
               {
                    distance [ s ] [ u ] = distance [ s ] [ v ] +
                    distance [ v ] [ u ] ;
                     precede [ s ] [ u ] = v ;
               }
          }
      }
}
```

Առաջադրանքներ

8.1.Արտածել տրված ուղղորդված գրաֆի բիներ հարաբերության կարգավոր զույգերը և գագաթների մտից ու ելից աստիճանները:
8.2. Տրված է ինչ-որ հակաանդրադարձ և համաչափ բիներ հարաբերություն: Կառուցել այդ հարաբերությանը համապատասխան պարզ գրաֆի հարևանության և կցության մատրիցները:
8.3.Պարզել՝ արդյո՞ք օգտագործողի կողմից մուտքագրված գագաթների հաջորդականությունը ճանապարհ է տրված ուղղորդված գրաֆում:
8.4.Պարզել՝ արդյո՞ք տրված ուղղորդված գրաֆը ուժեղ կապակցված է:
8.5.Հայտնաբերել պահանջված երկարության ճանապարհներ ուղղորդված գրաֆի տրված գագաթների զույգերի միջև:
8.6.Իրականացնել Դիքստրայի ալգորիթմը:
8.7.Արտածել տրված ուղղորդված գրաֆի «բետա» ինդեքսի արժեքը:
8.8. Պարզել՝ արդյո՞ք տրված ուղղորդված գրաֆն օղակներ ունի:

Լաբորատոր աշխատանք N 9

Լաբորատոր աշխատանքի նպատակն է՝ ծանոթանալ ծառերի համակարգչային ներկայացմանը և ստեղծված տվյալների կառուցվածքներում տարատեսակ վերլուծություններ իրականացմանը:

Ծառերը տվյալների ստորակարգային կառուցվածքներ են և, չնայած այն հանգամանքին, որ դրանք գրաֆների մասնավոր դեպքն են՝ կապակցված և շրջափուլերից զուրկ, այնուամենայնիվ ունեն համակարգչային ներկայացման և մշակման յուրահատուկ եղանակ:

Ծառերի պարագայում ներմուծվում են սկզբունքորեն այլ գաղափարներ, քան միայն գագաթները և կողերը գրաֆների տեսության շրջանակներում:

Հիշենք, որ ի տարբերություն գրաֆների, ծառերի կառուցվածքում ցանկացած երկու հանգույցների միջև կա միակ ճանապարհ, որը նման է մարդկանց խմբերի միջև ժառանգությանը տոհմածառերի կառուցվածքում:

Ծառն իրականացվում է *հանգույցների* հավաքածուի (որում հանգույցներից մեկը կանխորոշվում է որպես *արմատ*) և «ծնողական» հարաբերությունների տեսքով, որոնք էլ ներկայացնում են առաջադրված ստորակարգային կառուցվածքը:

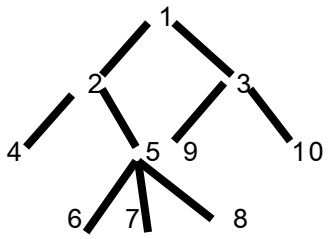
Ծառի հանգույցները կարող են լինել ինչպես հիմնային տեսակների, ինչպիսիք են՝ թվերը, նիշերը, կոտորակները, այնպես էլ՝ օգտագործողի կողմից նախագծված դասի նմուշներ: Ընդունված է ասել, որ հանգույցները ծառայում են տվյալները կուտակելու, մշակելու և հիշելու համար, իսկ կողերը՝ հաշվարկի կամ մշակման արդյունքները նպատակային գագաթներին փոխանցելու համար: Այդ փոխանցումը կարող է լինել կամ ուղղակի, կամ միջնորդավորված՝ կախված նախագծի պահանջներից:

Ծառերը կառուցվում են անդրադարձ սկզբունքով՝ սկսելով ինչ-որ տրված հանգույցից, որն էլ հենց արմատն է, այնուհետև շարունակելով կցել մնացած հանգույցներն ըստ առաջադրված ստորակարգի, հասնում են ծառերի վերջնակետերին՝ տերևներին, որոնք տվյալ պահին եզրափակում են նախագիծը: Դինամիկ ծառերի պարագայում ծառը կարող է ձևափոխվել՝ ավելացնելով կամ հեռացնելով նշված հանգույցները:

Այս լաբորատոր աշխատանքի սահմաններում դիտարկվում է ծառերի համակարգչային ներկայացման վեկտորային եղանակը, որն ամենապարզն է և որի սկզբունքը հետևյալն է. առաջադրվում է միաչափ զանգված, որի չափն ընտրվում է ծառի հանգույցների քանակին հավասար: Այդ դեպքում արդյունարար միաչափ զանգվածի ինդեքսները դառնում են ծառի հանգույցներ, իսկ տարրերը՝

այդ հանգույցների ծնողներ: Ակնհայտ է, որ ծառի արմատի կարգային համարին՝ ինդեքսին, համատասխանող վեկտորի արժեքը 0 է, քանի որ, ըստ սահմանման, ծառի արմատը ծնող չունի:

Ստորև բերված է ինչ-որ ստորակարգային տվյալներին համապատասխանող ծառ և նրա համակարգչային ներկայացումը:



Բերված ծառի կառուցվածքին համապատասխանող միաչափ զանգվածը կունենա հետևյալ տեսքը և պարունակությունը:

0	1	1	2	2	5	5	5	3	3
1	2	3	4	5	6	7	8	9	10

Առաջադրանքներ

9.1.Արտածել ծառի բոլոր հանգույցների ծնողները՝ բացի արմատից, և հետնորդները:
9.2.Պարզել ծառի բարձրությունը և արտածել վերջինիս թվային արժեքը:
9.3.Պարզել՝ արդյո՞ք տրված ծառը հավասարակշռված է:
9.4.Պարզել տրված ծառի ներքին հանգույցների քանակը:
9.5.Պարզել տրված ծառի տերևների քանակը:
9.6.Արտածել տրված ծառի գագաթից մինչև տրված տերև առկա ճանապարհը:

Լաբորատոր աշխատանք N 10

Լաբորատոր աշխատանքի նպատակն է՝ ուսումնասիրել դինամիկ ծառերի համակարգչային ներկայացման եղանակները, քանի որ վերոհիշյալ եղանակն ավելի կիրառելի է սևեռված ստորակարգային տվյալների կառուցման դեպքում:

Դինամիկ ծառերը տվյալների հարափոփոխ կառուցածքներ են, որոնք կարող են աճել դեպի աջ, դեպի ձախ, վերև, ներքև և այլ ուղղություններով՝ ըստ օգտագործողի նախասիրության, որն էլ տվյալների ավանդական զանգվածների հետ դրանց ունեցած հիմնական տարբերությունն է: Այսպիսով, ծառերը որպես տվյալների կառուցվածքներ ավելի ճկուն են, քանի որ տարրեր ավելացնելու և հեռացնելու բարդություն չունեն. պարզապես ներմուծվող տարրի ցուցիչը կապվում է հարևանների ցուցիչների հետ, իսկ հեռացվող տարրի հարևան հանգույցներն ուղղակի թարմացնում են կապերը: Բերված տեխնոլոգիան ծրագրավորման լեզուներում իրականացվում է այսպես կոչված «ինքնահասցեավորվող» կամ «ինքնահղմամբ» դասերի միջոցով:

Հետևյալ ծրագիրը ներկայացնում է մեկ ուղղությամբ դինամիկ ծառի կառուցման եղանակը, որը համապատասխան փոփոխության ենթարկելուց հետո կարող է ավելի բարդ տվյալների կառուցվածքի մշակման հիմք ծառայել: Նախատեսված է ծրագրավորման մեջ ավելի հմուտ ուսանողների համար, որպես լրացուցիչ հետազոտության նյութ:

Մաս1. Ծառի հանգույց արտադրող դաս:

```
class Node {
public:
        Node ( ) ;
        int el ;
        Node*  Next ;
};
Node ::  Node ( )
```

```
{
        el = 0 ;
        Next = NULL ;
}
```

Մաս2. Ծառի կառուցում և նրանում տվյալների փնտրում

```
void main()
{
   Node *top = new Node ( ) ;
   top - > el = 10;
   top - > Next = new Node ( ) ;

   Node *step1 = top - > Next ;
   step1- > Next = new Node ( ) ;
   step1 - > el = 20 ;

   Node * step2 = step1- > Next ;
   step2 - > el = 30 ;
   int count = 0 ;
   Node * temp = top ;
   int elem = 0 ;
   cout << " \ n ???  "; cin >> elem ;
   int flag = 0 ;
   while ( temp )
   {
          count  ++ ;
          if ( temp -  > el = = elem )
          {
            flag  ++ ;
            break ;
          }
          temp = temp -> Next ;
    }
     if ( flag )
     {
       cout<<"\nFound= "<< count <<endl ;
     }
     else
     {
        cout << " \ n No such element in that tree \ n " ;
      }
}
```

Բերված ծրագրային մարմնի հիման վրա առաջարկվում է իրականացնել այլ վերլուծություններ՝ կատարելով ծրագրային համապատասխան փոփոխություններ կամ ավելացումներ:

Առաջադրանքներ

10.1.Կառուցել ծառ, որն ունի առնվազն չորս ուղղությամբ ընդարձակվելու հնարավորություն:
10.2.Իրականացնել որոնման որևէ բինար ծառի ծրագրային ապահովում:
10.3.Իրականացնել որևէ ձեռնարկության ստորակարգային կառուցվածք և արտածել տարբեր նշանակության բաժինների ծառայողների ցուցակը և աշխատավարձերը:
10.4.Ծառի կառուցվածքում ներմուծել հանգույց՝ պահանջվող դիրքում:
10.5.Ծառի կառուցվածքից հեռացնել պահանջվող հանգույցը:
10.6. Կառուցել կամայական բինար ծառ և, վերագրելով բոլոր ձախ հանգույցներին 0, իսկ բոլոր աջ հանգույցներին 1 արժեքներ, արտածել արմատից մինչև յուրաքանչյուր տերև կամ տերևից դեպի արմատ ձգվող բոլոր երկուական կոդերը: Պարզել նաև, թե այդպիսով, այբուբենի քանի տառ է կոդավորվել տվյալ ծառի կառուցվածքում:

Լաբորատոր աշխատանք N 11

Լաբորատոր աշխատանքի նպատակն է՝ ուսումնասիրել արմատավորված ուղղորդված ծառերը որպես տեղեկույթի պահպանման կառուցվածքներ: Ընդսմին, հարկավոր է մշակել այնպիսի ընթացակարգեր, որոնք թույլ կտան այցելել յուրաքանչյուր հանգույց՝ հարկավոր տվյալների մատչելիությունն ապահովելու համար:

Ծառերի հանգույցների շրջանցումը կատարվում է *ուղիղ*, *հակադարձ* և *համաչափ* եղանակներով, որոնք

համապատասխանաբար կարգավորում են ծառի հանգույցներն ըստ տրված չափանիշի:

Ստորև ներկայացված է ծառի ուղիղ շրջանցման իրականացման ծրագրային ապահովման օրինակ:

Մաս1.Օգտագործողի գրադարանային ֆայլի իրականացում

```
//tree.h
class element
{
        char  a;  element *parent;   element *left;  element *right;
         friend void print( element &x ) ;
public:
        element( char b, element *pa ) ;
        element *get_left( ) ;
        element *get_right( ) ;
        ~element( ) ;
        bool child ( char b ) ;
};
```

Մաս2. “element” դասի անդամների իրականացում

```
//tree. cpp
element::element ( char  b, element *pa )
{
        a = b;                parent = pa;
        left = NULL;       right = NULL;
}
bool element:: child ( char  )
{
        if ( b > a)
        {
                if ( right )
                {
                   return right -> child( b ) ;
                }
                else
                {
                    right = new element (b , this );  return true ;
                }
```

```
        }
        else if ( b < a )
        {
                if ( left )
                {
                   return left -> child( b ) ;
                }
                else
                {
                   left = new element( b , this );
                   return true ;
                }
        }
        else
        {
           return false;
        }
}
void print ( element &x )
{
        cout << x.a << ' ' ;
        if ( x . left)          print(*x . left );
        if (x . right)          print(*x . right );
}
element :: ~element()
{
        if ( left )             delete left;
        if ( right )            delete right;
}
element *element :: get_left( )   { return left ;   }
element *element :: get_right( ) { return right ; }
```

Առաջադրանքներ

11.1.Իրականացնել տրված ծառի հանգույցների հակադարձ շրջանցումը:
11.2.Իրականացնել տրված ծառի հանգույցների համաչափ շրջանցումը:

11.3.Արտածել տրված ծառի բարձրության արժեքը և այդ բարձրությունն ապահովող համապատասխան հանգույցները:
11.4.Փնտրել մուտքագրված երկուական կոդը տրված ծառի կառուցվածքում և արտածել արմատից մինչև տերև հանգույցների անվանումները:
11.5.Արտածել ծառի միևնույն մակարդակի հանգույցները:
11.6. Չևավորել բինար ծառ, ապահովել նրա հանգույցների կոդավորումը և իրականացնել նրանում տվյալների երկուական փնտրում:

Լաբորատոր աշխատանք N 12

Լաբորատոր աշխատանքի նպատակն է՝ ուսումնասիրել Բուլյան ֆունկցիաների առաջադրման և վերլուծության համակարգչային եղանակները:

Ինչպես հայտնի է, համակարգչի կառուցվածքային բաղադրամասերի և էլեկտրոնային սարքավորումների մուտքային/ելքային ազդանշաններն ընդունում են 1 կամ 0 արժեքներ, ուստի գործարկում են այնպիսի տրամաբանական հիմնային տարրեր, որոնք ունեն երկու կայուն վիճակ: Այդպիսի տարրերի տրամաբանության հիմքում ընկած են Բուլյան հանրահաշվի դրույթները, իսկ տրամաբանական շղթայի գործողությունը որոշվում է համապատասխան Բուլյան ֆունկցիայի միջոցով, որը սահմանում է շղթայի ելքային արժեքը մուտքային հավաքներից յուրաքանչյուրի համար:

Բուլյան ֆունկցիաները լայնորեն կիրառվում են նաև կոդերի տեսության մեջ՝ շնորհիվ իրենց իրականացման պարզության և բարձր արագագործության: Երկհիմնային ավելցուկային կոդերը, ինչպես նաև սխալներ հայտնաբերող և ուղղող կոդերի գերակշիռ մասը հիմնված են Բուլյան ֆունկցիաների վրա:

Բուլյան ֆունկցիայի համակարգչային առաջադրման առենահարմար եղանակը աղյուսակային ներկայացումն է,

որտեղ Բուլյան ֆունկցիայի ընդունած արժեքները կարող են մուտքագրվել կամ առաջադրվել պատահական կերպով:

Ստորև ներկայացված է երեք փոփոխականի Բուլյան ֆունկցիայի մուտքային հավաքների առաջադրման ծրագրային ապահովման օրինակ:

```
//TruthTable. cpp
void InitTT( bool** , int&, int& );
void GenerateTT( bool** , int&,  int& );
void DemonstrTT( bool** , int&,  int& );
int _tmain(int argc, _TCHAR* argv[])
{
        int R = 8, C = 3;
        bool**  TruthTable = new bool * [R] ;
        for ( int  i = 0 ; i < R ; i ++)
        {
                TruthTable [ i] = new bool [ C ] ;
        }
        InitTT( TruthTable , R , C ) ;
        GenerateTT( TruthTable , R , C ) ;
        DemonstrTT( TruthTable , R , C ) ;
        for (int i = 0 ; i < R ; i + +)
        {
                delete [ ] TruthTable [ i ] ;
        }
        delete [ ] TruthTable ;
        return 0 ;
}
void InitTT(bool** TT, int& R, int& C)
{
    for ( int i = 0; i < R ; i + +)
    {
         for( int j = 0 ; j < C ; j + +)
         {
             * ( * ( TT + i ) + j ) = false;
         }
         cout<<endl;
    }
}
void GenerateTT(bool** TT , int& R ,  int& C )
{
```

```
    for ( int i = 0 ; i < R ; i + + )
    {
        for ( int j = 0 ; j < C ; j + + )
        {
                if ( ! j )
                {
                     for (int k = 4 ; k < 8 ; k + + )
                     {
                        TT [ k ] [ j ] = true;
                     }
                }
                 if ( j = = 1 )
                 {
                    TT  [ 2 ] [ j ] = true;  TT  [ 3 ] [ j ] = true;
                     TT [ 6 ] [ j ] = true;   TT [ 7 ] [ j ] = true;
                 }
                 if ( j = = 2 )
                 {
                     for ( int k = 1 ; k < 8 ; k + = 2 )
                     {
                         TT [ k ] [ j ] = true;
                     }
                 }
          }
     }
}
void DemonstrTT(bool** TT, int& R, int& C)
{
    for ( int i = 0 ; i < R ; i ++ )
    {
        for ( int j = 0 ; j < C ;  j + + )
        {
            cout < < TT [ I ] [ j ] < < '  ';
        }
            cout < < endl ;
    }
}
```

Նկատենք, որ տրված փոփոխականների համար բոլոր Բուլյան ֆունկցիաների մուտքային հավաքները նույնն են: Փոփոխականների մեծ քանակի դեպքում խորհուրդ է տրվում ներկայացման աղյուսակային եղանակը փոխարինել

վեկտորական եղանակով: Այս դեպքում վեկտորի դիրքաթիվը մուտքային համապատասխան հավաքի տասական արժեքն է, իսկ վեկտորի տարրը՝ այդ հավաքի վրա ֆունկցիայի ընդունած արժեքը:

Առաջադրանքներ

12.1.Արտածել ճշմարտության աղյուսակ՝ պահանջվող փոփոխականների և առնվազն երեք Բուլյան ֆունկցիաների արժեքներով:
12.2.Արտածել աղյուսակի տեսքով տրված Բուլյան ֆունկցիայի արժեքները և այդ արժեքների “բացառող կամ” գործողության արդյունքը:
12.3.Արտածել աղյուսակային տեսքով տրված Բուլյան ֆունկցիայի ԿԴՆՁ բազմանդամի տարրական կոնյունկցիաները:
12.4.Արտածել աղյուսակային տեսքով տրված Բուլյան ֆունկցիայի ԿԿՆՁ բազմանդամի տարրական դիզյունկցիաները:

Լաբորատոր աշխատանք N 13

Լաբորատոր աշխատանքի նպատակն է՝ ուսումնասիրել հավասարակշռված, կամ որ նույնն է՝ բալանսավորված Բուլյան ֆունկցիաների առաջադրման համակարգչային եղանակը:

Դիսկրետ մաթեմատիկայում և հաշվողական համակարգերում հավասարակշռված Բուլյան ֆունկցիաներն այնպիսի ֆունկցիաներ են, որոնց ելքային արժեքներում 1 և 0 նիշերի քանակը նույնն է: Սա նշանակում է, որ բիթերի կամայական մուտքային հաջորդականության համար ելքային նիշերի 1 արժեք ընդունելու հավանականությունը 1/2 է: Հավասարակշռված

Բուլյան ֆունկցիաները լայնորեն կիրառվում են հատկապես գաղտնագրային ընթացակարգերում այն հիմնավորմամբ, որ դրանց կիրառության շնորհիվ զգալիորեն բարդանում է գաղտնագրված տվյալների չլիազորված ընթերցման տարբերութային գաղտնավերլուծությունը:

Հավասարակշռված Բուլյան ֆունկցիաների հավաքածուների համակարգչային առաջադրման ամենահարմար եղանակը համապատասխան կարգի քվազիխմբի Քեյլիի աղյուսակի գեներացումն է, որի արդյունքում ստացվում են պահանջվող կարգի և բաշխվածության Բուլյան ֆունկցիաներ՝ շրջանցելով վերջիններիս կոմբինատոր առաջադրման ծախսատար եղանակները:

Ստորև ներկայացված է հավասարակշռված Բուլյան ֆունկցիայի մուտքային հավաքների առաջադրման ծրագրային ապահովման օրինակ:

```
// Cayley.cpp
bool BM_C[ R ] [ C ] ={false};
int balancedRows = 0 ;
int QuasigroupOrder = rand ( ) % 5 + 5 ;
cout < < ” \n Put the order of the Quasigroup ”;
cin>> QuasigroupOrder;
for ( int i = 0 ; i < R ; i + + )
{
        for ( int j = 0 ; j < R ; j + + )
        {
           BM_C [ j ] [ i ] = rand ( ) % 2 ;
        }
        for ( j = 0 ; j < R ; j + + )
        {
             if (BM_C [ j ] [ i ] ) bits + + ;
             if ( bits ! = R / 2 )
             {   i - - ; bits = 0 ; continue ;    }
              balancedRows + + ;
              if( balancedRows = = QuasigroupOrder ) break;
         }
}
```

Առաջադրանքներ

13.1.Արտածել տրված հավասարակշռված Բուլյան ֆունկցիաների արժեքների Հեմինգի հեռավորությունը:
13.2.Արտածել տրված հավասարակշռված Բուլյան ֆունկցիաների արժեքների Լիի հեռավորությունը:

Լաբորատոր աշխատանք N 14

Լաբորատոր աշխատանքի նպատակն է՝ ուսումնասիրել Բուլյան ֆունկցիաների ելքային արժեքների պահանջվող եղանակով ձևափոխությունների իրականացման արդյունավետ եղանակները:

Ստորև ներկայացված են Բուլյան ֆունկցիաների ելքային արժեքների հակադարձման և տեղափոխության համապատասխան ծրագրային ապահովման օրինակները:

```
//Bool-Reverse.cpp
x = (x & 0x55555555) <<  1 | (x & 0xAAAAAAAA) >>  1;
x = (x & 0x33333333) <<  2 | (x & 0xCCCCCCCC) >>  2;
x = (x & 0x0F0F0F0F) <<  4 | (x & 0xF0F0F0F0) >>  4;
x = (x & 0x00FF00FF) <<  8 | (x & 0xFF00FF00) >>  8;
x = (x & 0x0000FFFF) << 16 | (x & 0xFFFF0000) >> 16;
//Bool-Shuffle.cpp
x = (x & 0x0000FF00) << 8 | (x >> 8) & 0x0000FF00 |
     x & 0xFF0000FF;
x = (x & 0x00F000F0) << 4 | (x >> 4) & 0x00F000F0 |
     x & 0xF00FF00F;
x = (x & 0x0C0C0C0C) << 2 | (x >> 2) & 0x0C0C0C0C |
     x & 0xC3C3C3C3;
x = (x & 0x22222222) << 1 | (x >> 1) & 0x22222222 |
     x & 0x99999999;
```

Հետևյալ ծրագրային մարմնի ներդրման և գործարկման արդյունքում իրականացվում է Բուլյան ֆունկցիաների հավաքածուի ուղիղ և տրանսպոնացված մատրիցների ցուցադրում: Ծրագրի ընթեռնելիության ապահովման

Նպատակով Բուլյան ֆունկցիաների ելքային արժեքների բիթերը ներկայացված են տարբեր տառաթվային նիշերով:

```
//Bool-Transpose.cpp
a0 = 0123  4567       b0 = 08go  wEMU
a1 = 89ab  cdef       b1 = 19hp  xFNV
a2 = ghij  klmn       b2 = 2aiq  yGOW
a3 = opqr  stuv  ==>  b3 = 3bjr  zHPX
a4 = wxyz  ABCD       b4 = 4cks  AIQY
a5 = EFGH  IJKL       b5 = 5dlt  BJRZ
a6 = MNOP  QRST       b6 = 6emu  CKS$
a7 = UVWX  YZ$.       b7 = 7fnv  DLT.
b0 = (a0 & 128)   | (a1 & 128)/2  | (a2 & 128)/4  | (a3 & 128)/8 |
     (a4 & 128)/16 | (a5 & 128)/32 | (a6 & 128)/64 | (a7 )/128;
b1 =  (a0 & 64)*2  | (a1 &  64)   | (a2 &  64)/2  | (a3 &  64)/4 |
     (a4 & 64)/8  | (a5 &  64)/16 | (a6 &  64)/32 | (a7 & 64)/64;
b2 =  (a0 & 32)*4  | (a1 &  32)*2  | (a2 &  32)   | (a3 & 32)/2 |
     (a4 & 32)/4  | (a5 &  32)/8  | (a6 &  32)/16 | (a7 & 32)/32;
b3 =  (a0 & 16)*8  | (a1 &  16)*4  | (a2 &  16)*2  | (a3 & 16)   |
     (a4 & 16)/2  | (a5 &  16)/4  | (a6 &  16)/8  | (a7 & 16)/16;
b4 =  (a0 & 8)*16 | (a1 &   8)*8  | (a2 &   8)*4  | (a3 & 8)*2 |
     (a4 & 8)   | (a5 &   8)/2  | (a6 &   8)/4  | (a7 & 8)/8;
b5 =  (a0 & 4)*32 | (a1 &   4)*16 | (a2 &   4)*8  | (a3 & 4)*4 |
     (a4 & 4)*2 | (a5 &   4)    | (a6 &   4)/2  | (a7 & 4)/4;
b6 =  (a0 & 2)*64 | (a1 &   2)*32 | (a2 &   2)*16 | (a3 & 2)*8 |
     (a4 & 2)*4 | (a5 &   2)*2  | (a6 &   2)    | (a7 & 2)/2;
b7 =  (a0)*128| (a1 &   1)*64 | (a2 &   1)*32 | (a3 & 1)*16|
     (a4 & 1)*8 | (a5 &   1)*4  | (a6 &   1)*2  | (a7 & 1);
```

Առաջադրանքներ

14.1.Արտածել տրված Բուլյան ֆունկցիաների արժեքները ներկայացնող մատրիցի տողերի և սյուների կշիռները և զույգության բիթերը:
14.2.Արտածել տրված Բուլյան ֆունկցիաների կարգը և համապատասխան ավելցուկային կոդի բաղադրիչները, ներառյալ՝ նվազագույն հեռավորությունը:

Գրականություն

1. Տոնոյան Ռ.Ն.,Դիսկրետ մաթեմատիկայի դասընթաց (դասախոսություններ և առաջադրանքներ): Եր. ԵՊՀ.,1997
2. Джеймс Андерсон, Дискретная математика и комбинаторика: Пер. С англ., М. 2003.
3. Прата С. Язык программирования С++. СП6. ООО “ДиаСофтОП”, 2002.

Պատվեր՝ 1390 Տպաքանակ՝ 100

Տպագրված է Հայաստանի Պետական Ճարտարագիտական Համալսարանի (Պոլիտեխնիկ) տպարանում

Երևան, Տերյան 105