Дана таблица с произвольным числом строк и столбцов с произвольным содержимым.

Необходимо:

Найти все возможные пути из левого края таблицы в правый. В каждом варианте пути каждый столбец может быть занят только один раз. Вывести список всех этих путей в виде перечисленного через запятую содержимого этих строк.

Примеры путей:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Оглавление**

Постановка задачи……………………………………………………………………………….3

Глава 1: Основные теоретические сведения…………………………………………………...4

§1. Определение графа и способы его задания………………………….………………...4

§2. Остовное дерево…………………………………………………………………………5

Глава 2: Методы решения задачи……………………………………………………………….6

§1. Алгоритм Прима………………………………………………………………….……..6

§2 Алгоритм Крускала……………………………………………………………..……….7

Глава 3: Характеристики программного продукта…………………………………………….8

§1. Входная и выходная информация………………………………………………...……8

§2. Среда разработки программного обеспечения………………………………………..8

§3. Реализация алгоритмов……………………………………………………………...….8

§4. Тестирование……………………………………………………………………….…..13

§5. Возможные ошибки при работе программы………………………………………....14

§6. Сопровождение и эксплуатация………………………………………………………15

Список литературы……………………………………………………………………………..17

**Постановка задачи**

Для данного неориентированного графа

 1. реализовать *алгоритм Прима (ближайшего соседа)* нахождения остовного дерева наименьшей стоимости;

 2. реализовать *алгоритм Крускала (“жадный” алгоритм)* нахождения остовного дерева наименьшей стоимости.

 Сравнить время выполнения программ на множестве “случайных” графов.

 Снабдить программу графической иллюстрацией, выделяя цветом ребра, образующие остовное дерево.

 Входная информация: текстовый файл, содержащий в произвольном порядке список ребер данного графа с указанием веса каждого ребра.

**Глава 1: Основные теоретические сведения**

**§1. Определение графа и способы его задания**

Рассмотрим множество *V*={*v*1,*v*2,...,*vn*}, *n*>2, и множество *E*={*e*1,*e*2,...,*em*}, элементами *ek* которого являются двухэлементные подмножества {*v*i, *v*j} множества *V*. Пара множеств *V* и *E* называется *неориентированным* *графом* *F*(*V*,*E*) с *множеством вершин* *V* и *множеством ребер* *E.*

 Пусть теперь множество *E*={*e*1,*e*2,...,*em*} представляет собой некоторое бинарное отношение на множестве *V* ((*E**V**V*). Тогда пара множеств *V* и *E* называется *ориентированным* *графом* (*орграфом*) *F*(*V*,*E*) с *множеством вершин* *V* и *множеством ребер* *E*.

 Граф, по определению представляющий собой пару множеств, можно задать любым из способов задания множеств. Однако для графа существуют некоторые специфические способы задания, которые будут рассмотрены в этом параграфе.

 Пусть *v*1,*v*2,...,*vn* - вершины графа *F*; *e*1,*e*2,...,*em* - его рёбра. Отношение инцидентности можно определить *матрицей инцидентности Bm*x*n* , *n* столбцов которой соответствуют вершинам графа, а  *m* строк - его рёбрам. Для *неориентированного* графа: *bij* =1, если ребро *ei* инцидентно вершине *vj*, в противном случае *bij* =0.

 В матрице инцидентности *ориентированного* графа, если вершина *vj* - начало ребра *ei*, то *bij* = 1, если *vj* - конец *ei*, то *bij* =1, если *ei* - петля, а *vj* - инцидентная ей вершина, то *bij* = (где  *-* любое число, отличное от 1, 0, -1), в остальных случаях *bij* =0.

 В каждой строке матрицы инцидентности только два элемента отличны от нуля (или один, если ребро является петлёй), поэтому такой способ задания графа оказывается недостаточно экономным.

 Отношение инцидентности можно задать *списком рёбер R*. Каждая строка этого списка соответствует одному ребру; в ней записаны номера вершин, инцидентных ему. Для неориентированного графа порядок вершин в строке произволен, для ориентированного графа сначала указывается начальная вершина, а затем - конечная.

 *Матрицей смежности* графа является квадратная матрица *C n*x*n*, столбцам и строкам которой соответствуют вершины. Для *неориентированного* графа элемент матрицы смежности *cij* равен количеству рёбер, инцидентных *i*-й и *j*-й вершинам; для *ориентированного* графа этот элемент равен количеству рёбер с началом в *i*-й вершине и концом в *j*-й вершине.

 Таким образом, матрица смежности неориентированного графа симметрична (*cij*=*cji*) относительно главной диагонали, а для ориентированного графа она будет симметрична только в том случае, если для каждого ребра имеется ребро, соединяющее те же вершины, но идущее в противоположном направлении.

 В силу симметричности матрицы смежности для неориентированного графа все его рёбра будут определяться верхним правым треугольником вместе с главной диагональю этой матрицы.

**§2. Остовное дерево**

*Остовное дерево* — ациклический связный [подграф](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%B3%D1%80%D0%B0%D1%84) данного связного неориентированного графа, в который входят все его вершины. Неформально говоря, остовное дерево состоит из некоторого подмножества рёбер графа, таких, что из любой вершины графа можно попасть в любую другую вершину, двигаясь по этим рёбрам, и в нём нет циклов, то есть из любой вершины нельзя попасть в саму себя, не пройдя какое-то ребро дважды..

Остовное дерево также иногда называют *покрывающим деревом*, *остовом* или *скелетом* графа.

*Минимальное остовное дерево* (или *минимальное покрывающее дерево*) в связанном, взвешенном, неориентированном графе — это остовное дерево этого графа, имеющее минимальный возможный вес, где под весом дерева понимается сумма весов входящих в него рёбер.

**Глава 2: Методы решения задачи**

**§1. Алгоритм Прима**

Этот алгоритм назван в честь американского математика *Роберта Прима* (*Robert Prim*), который открыл этот алгоритм в 1957 г. Впрочем, ещё в 1930 г. этот алгоритм был открыт чешским математиком *Войтеком Ярником* (*Vojtěch Jarník*). Кроме того, *Эдгар Дейкстра* (*Edsger Dijkstra*) в 1959 г. также изобрёл этот алгоритм, независимо от них.

Алгоритм начинает работу с произвольной вершины графа *s*, выбираемой в качестве корня дерева, и в ходе последовательно выполняемых итераций расширяет дерево до МОД (*минимального остовного дерева*). Пусть $V\_{t}$ есть множество вершин, уже включенных алгоритмом в МОД, а величины $d\_{i}$, *1 ≤ i ≤ N*, характеризуют дуги минимальной длины от вершин, еще не включенных в дерево, до множества $V\_{t}$, т.е.

$$∀i\notin V\_{t}⇒d\_{i}=min\{w(i,u):u\in V\_{t},(i,u)\in R\}$$

(если для какой-то вершины $∀i\notin V\_{t}$не существует ни одной дуги в $V\_{t}$, значение $d\_{i}$ устанавливается в ∞). В начале работы алгоритма выбирается корневая вершина МОД s и полагается:

$$V\_{t}=\{s\},d\_{s}=0$$

Действия, выполняемые на каждой итерации алгоритма Прима, состоят в следующем:

* определяются значения величин $d\_{i}$ для всех вершин, еще не включенных в состав МОД;
* выбирается вершина t графа G, имеющая дугу минимального веса до множества $V\_{t}$: $t:d\_{t}=min(d\_{i}),i\notin V\_{t}$
* вершина t включается в$ V\_{t}$.

После выполнения *n-1* итерации методом МОД будет сформировано. Вес этого дерева может быть получен при помощи выражения

$$W\_{t}=\sum\_{i=1}^{n}d\_{i}$$

Трудоемкость нахождения МОД характеризуется квадратичной зависимостью от числа вершин графа *O(n2)* .

**§2. Алгоритм Крускала**

*Алгоритм Крускала* — алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Открыт *Джозефом Крускалом* в 1956 году.

Предположим, что есть связный граф *G* = *(V, Е)* с множеством вершин *V =* (1, 2, ..., *п*}и функцией стоимости с, определенной на множестве ребер E. В ал­горитме Крускала (Kruskal) построение остовного дерева минимальной стоимости для графа *G* начинается с графа *Т* = (V, $∅$), состоящего только из *п* вершин графа G и не имеющего ребер. Таким образом, каждая вершина является связной (с самой собой) компонентой. В процессе выполнения алгоритма мы имеем набор связных компонент, постепенно объединяя которые формируем остовное дерево.

При построении связных, постепенно возрастающих компонент поочередно прове­ряются ребра из множества *Е* впорядке возрастания их стоимости. Если очередное ребро связывает две вершины из разных компонент, тогда оно добавляется в граф *Т.* Если это ребро связывает две вершины из одной компоненты, то оно отбрасывается, так как его добавление в связную компоненту, являющуюся свободным деревом, приведет к образованию цикла. Когда все вершины графа *G* будут принадлежать одной компоненте, построение остовного дерева минимальной стоимости *Т* для этого графа заканчивается.

Временная сложность данного алгоритма:

* *T(L)=O(ElogE)*, при *EF >> ET*

**Глава 3: Характеристики программного продукта**

**§1. Входная и выходная информация**

*Входная информация*: граф может быть задан одним из двух способов:

* Путём считывания списка смежности из текстового файла (первая строка – число вершин, вторая – число рёбер, начиная с третьей строки – список смежности);
* Путём генерации псевдослучайного взвешенного графа самой программой, при этом пользователь задаёт число вершин;

после чего пользователь выбирает интересующий его метод (Крускала или Прима)

*Выходная информация*: в результате своей работы программа находит остовное дерево наименьшей стоимости одним из двух способов, строит исходный граф и выделяет в нём остов. Так же выводится список рёбер остова с указанием их веса, суммарный вес остова, а так же время выполнения алгоритма.

**§2. Среда разработки программного обеспечения**

Данная программа была разработана в среде Visual C++ 2010.

**§3. Реализация алгоритмов**

Алгоритм Прима:

void prim(edge \*E,int n,int m,int \*L,int \*\*R)

 //алгоритм Прима для нахождения кратчайшего остова

 //а - массив ребер

 //m - число ребер

 //n - число вершин

 //L - вес остова

 //R - массив списка ребер остова

 {

 int i,j,k,//счетчики

 min,//вес текущего минимального ребра

 kr=0;//счетчик добавленных в остов ребер

 int \*\*a;//матрица весов ребер графа

 bool \*v;//массив меток

 int \*d;//массив весов ребер остова

 int \*p;//массив номеров вершин

 int mst\_weight=0;//вес остова

 v=new bool[n+1];//инициализация массивов

 d=new int[n+1];

 p=new int[n+1];

 for (i=1;i<=n;i++)

 {

 v[i]=false;

 d[i]=0;

 p[i]=0;

 }

 a=new int\*[n+1];

 for (i=0;i<=n;i++)

 a[i]=new int[n+1];

 for (i=0;i<=n;i++)//заполняем матрицу смежности большими числами

 for (j=0;j<=n;j++)

 a[i][j]=inf;

 for (i=1;i<=m;i++)//формируем матрицу смежности из списка ребер

 {

 a[E[i].x][E[i].y]=E[i].w;

 a[E[i].y][E[i].x]=E[i].w;

 }

 k=1;//начальная вершина

 v[1]=true;//пометили

 for (i=2;i<=n;i++)//первый столбец

 {

 d[i]=a[i][1];//запоминаем веса

 p[i]=1;//номер начальной вершины

 }

 for (i=1;i<=n-1;i++)//основной цикл

 {

 min=inf;//присвоили большое число

 for (j=1;j<=n;j++)//ищем ребро с минимальным весом

 if (! v[j] && d[j]<min)//если вершина не помечена и вес меньше

 {

 min=d[j];//запомнили

 k=j;

 }

 mst\_weight=mst\_weight+a[k][p[k]];//добавили в вес остова

 kr++;//увеличили число ребер

 R[kr][1]=k;//запомнили ребро в массиве

 R[kr][2]=p[k];

 v[k]=true;//пометили вершину

 for (j=1;j<=n;j++)//цикл корретировки

 if (! v[j] && d[j]>a[k][j])//если не помечена и вес меньше

 {

 p[j]=k;//запоминаем

 d[j]=a[k][j];

 }

 }

 \*L=mst\_weight;//передали в выходной параметр

 }

Алгоритм Крускала:

void kruskal(edge \*a,int mreb,int ngr,int \*L,int \*\*R)

 //построение остова (алгоритм Краскала)

 //а - массив ребер

 //mreb - число ребер

 //ngr - число вершин

 //L - вес остова

 //R - массив списка ребер остова

 {

 int k,i,//счетчики

 p,q,//номер вершин корней

 kr=0;//счетчик добавленных в остов ребер

 int \*r;//массив компонент связности

 int \*s;

 int mst\_weight=0;//вес остова

 qsort(a,1,mreb); //сортируем список ребер по неубыванию

 //инициализация массивов

 r=new int[ngr+1];

 s=new int[ngr+1];

 for (i=1;i<=ngr;i++) //цикл по вершинам

 {

 r[i] = i; //у вершины своя компонента связности

 s[i] = 1; //размер компоненты связности

 }

 k=0; //номер первого ребра + 1

 for (i=1;i<=ngr-1;i++) //цикл по ребрам mst

 {

 do // ищем ребра из разных

 {

 k++; //компонент связности

 p=a[k].x;

 q=a[k].y;

 while (r[p] != p) //ищем корень для p

 p = r[p];

 while (r[q] != q) //ищем корень для q

 q = r[q];

 }

 while (p == q);

 kr++;//запоминаем ребро

 R[kr][1]=a[k].x;

 R[kr][2]=a[k].y;

 mst\_weight = mst\_weight + a[k].w;//считаем вес

 if (s[p] < s[q]) // взвешенное объединение

 { //компоненты связности

 r[p]=q;

 s[q]=s[q]+s[p];

 }

 else

 {

 r[q] = p;

 s[p] = s[p] + s[q];

 }

 }

 \*L=mst\_weight;

 }

Генератор псевдослучайного взвешенного графа:

private: System::Void btnRandom\_Click(System::Object^ sender, System::EventArgs^ e)

 //генератор псевдослучайного взвешенного графа

 {

 Random^ rnd=gcnew Random;//описываем генератор случайных чисел

 int i,j,k,//счетчики

 s;//показатель полноты графа

 int RR[101];//массив меток сформированных вершин

 int \*\*A;

 int n;//количество вершин

 int m;//количество ребер

 n = Convert::ToInt32(numericUpDown1->Value);//читаем из строки ввода

 s=0;//обнулили

 A=new int\*[n+1];//инициализация массива

 for (i=0;i<=n;i++)

 A[i]=new int[n+1];

 for (i=1;i<=n;i++)

 {

 RR[i]=0;

 for (j=1;j<=n;j++)

 A[i][j]=0;

 }

 for (i=1;i<=n ;i++)//цикл начального заполнения

 if (RR[i]==0)

 {

 do

 {

 k=rnd->Next(n)+1;

 }

 while (k==i);

 A[i][k]=rnd->Next(100)+1;

 A[k][i]=A[i][k];

 RR[k]=1;

 s=s+2;

 }

 s=int((n\*n-n)\*70/100-s);//определяем полноту

 while (s > 1)//цикл окончательного формирования матрицы

 {

 i=rnd->Next(n\*n);

 k=i % n+1;

 i=i / n+1;

 if (i != k && A[i][k]==0)

 {

 A[i][k]=rnd->Next(100)+1;

 A[k][i]=A[i][k];

 s=s-2;

 }

 }

 m=0;//обнулили счетчик ребер

 for (i=1;i<=n;i++) //цикл подсчет числа ребер

 for (j=i+1;j<=n;j++)

 if (A[i][j]!=0)

 m++;

 if (m<3)//число ребер меньше 3

 {

 MessageBox::Show("Число ребер меньше 3!","Информация",

 MessageBoxButtons::OK,MessageBoxIcon::Information);//сообщение

 return;//выход

 }

 numericUpDown2->Value=m;//передаем в строки ввода

 numericUpDown1->Value=n;

 m=0;//счетчик номеров строк таблицы

 for (i=1;i<=n;i++)

 for (j=i+1;j<=n;j++)

 if (A[i][j]!=0)//если есть ребро

 {

 m++;//увеличили счетчик

 dataGridView1[1,m-1]->Value=Convert::ToString(i);//выводим в ячейки

 dataGridView1[2,m-1]->Value=Convert::ToString(j);

 dataGridView1[3,m-1]->Value=Convert::ToString(A[i][j]);

 }

 mnuKraskal->Enabled=true;//меню расчета доступны

 mnuPrim->Enabled=true;

 }

Визуализация:

private: System::Void pictureBox1\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e)

 //вывод остова

 {

 if (! mst) return;

 Graphics^ g=e->Graphics;//область рисования

 System::Drawing::Pen^ Pr = gcnew System::Drawing::Pen(System::Drawing::Color::Gray,1);//цвет ребер

 System::Drawing::Pen^ Pc = gcnew System::Drawing::Pen(System::Drawing::Color::Green,2);//цвет ребер цепи

 int i;//счетчик

 int n;//количество вершин

 int m;//количество ребер

 n = Convert::ToInt32(numericUpDown1->Value);//читаем из строки ввода

 m = Convert::ToInt32(numericUpDown2->Value);

 g->Clear(System::Drawing::Color::White);//очистка области построения

 for (i=1;i<=m;i++)

 g->DrawLine(Pr,xv[R[i].x],yv[R[i].x],xv[R[i].y],yv[R[i].y]);//рисуем линию

 System::Drawing::SolidBrush^ Pv =

 gcnew System::Drawing::SolidBrush(System::Drawing::Color::Red);//цвет вершины

 System::Drawing::SolidBrush^ Pf =

 gcnew System::Drawing::SolidBrush(System::Drawing::Color::Blue);//цвет шрифта

 System::Drawing::Font^ F = gcnew System::Drawing::Font("Arial", 8,

 FontStyle::Bold);//задаем параметры шрифта

 for (i=1;i<n;i++)//выводим ребра остова

 g->DrawLine(Pc,xv[Ost[i][1]],yv[Ost[i][1]],

 xv[Ost[i][2]],yv[Ost[i][2]]);

 for (i=1;i<=n;i++)

 {

 g->FillEllipse(Pv,xv[i]-5,yv[i]-5,10.0,10.0);//выводим вершины

 g->DrawString(Convert::ToString(i),F,Pf,xv[i]-15,yv[i]-15);//подписи вершин

 }

 }

private: System::Void Form1\_Resize(System::Object^ sender, System::EventArgs^ e)

 {

 if (! mst) return;

 int i;//счетчик

 int n;//количество вершин

 float R = (float)Math::Min((float)pictureBox1->Width,(float)pictureBox1->Height) / 2 - 30;

 float a = 0;

 n = Convert::ToInt32(numericUpDown1->Value);//читаем из строки ввода

 for (i=1;i<=n;i++)

 {

 xv[i]=(float)pictureBox1->Width/2+(float)(R\*Math::Cos(a));

 yv[i]=(float)pictureBox1->Height/2-(float)(R\*Math::Sin(a));

 a=a+(float)(2\*Math::PI/n);

 }

 pictureBox1->Refresh(); //обновляем область вывода

 pictureBox1->Invalidate();

 }

};

}

**§4. Тестирование**

Произведём тестирование нашей программы на некотором множестве псевдослучайных графов.







**§5. Возможные ошибки при работе программы**

Ошибки при работе программного обеспечения могут возникать при вводе информации пользователем.

* При считывании исходного графа из текстового файла должны быть соблюдены указанные в §1 требования к его структуре.
* Исходный граф должен содержать не менее трех, и не более тысячи рёбер

В случае невыполнения данных требований будет выдано сообщение об ошибке, предоставляющея на выбор пользователя возможность продолжить работу с программой, или же закрыть её.

**§6. Сопровождение и эксплуатация**

Для начала работы с программой требуется запустить исполняемый файл *ostov\_KR.exe*.

После запуска программы пользователь увидит следующее окно:



*Панель меню* имеет следующую структуру:

1. Файл
	1. Открыть – выбор текстового файла для чтения исходного графа
	2. Выход – завершение работы программы
2. Расчет
	1. Алгоритм Краскала – построение остовного дерева наименьшей стоимости по алгоритму Краскала
	2. Алгоритм Прима - построение остовного дерева наименьшей стоимости по алгоритму Прима

Под панелью меню расположены три области:

* *Исходные данные*. В поле «*Количество вершин*» пользователь задаёт число вершин генерируемого графа, далее при нажатии кнопки «*ГСЧ*» (*генератор случайных чисел*) в поле «*Количество рёбер*» будет выведено число рёбер сгенерированного графа, в *таблице* ниже будет приведён список этих рёбер с указанием каждого из них. Данная таблица выводится и в случае чтения исходного графа из текстового файла.
* *Остов* – содержит информацию о результате выполнения выбранного пользователем алгоритма, по завершении работы которого в данном окне будет указано название используемого алгоритма, список рёбер получившегося остова, вес остова, а так же время работы алгоритма.
* Правее всего расположена *область рисования* – в ней происходит построение исходного графа с выделением в нём найденного остова после выбора пользователем одного из двух алгоритмов. При этом вершины представлены кружками красного цвета, не входящие в остов рёбра – прямыми линиями серого цвета, рёбра остова – прямыми линиями зелёного цвета большей толщины, а номера вершин выделены синим цветом. При изменении размеров окна программы изображение графа так же массштабируется.

**Список литературы**

1. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. М.: Издательский дом “Вильямс”, 2001.- 384 с.
2. Зубов В.С. Справочник программиста. Базовые методы решения графовых задач и сортировки. М.: Информационно-издательский Дом “Филинъ”, 1999. - 256 с.
3. Иванов Б.Н. Дискретная математика. Алгоритмы и программы: Учеб. пособие. - М.: Лаборатория базовых знаний, 2001. - 288 с.
4. Стивенс Р. Delphi. Готовые алгоритмы. М.: ДМК Пресс, 2001. - 384 с.
5. Новиков Ф.А. Дискретная математика для программистов. СБП: Питер, 2000. – 304 с.
6. Конспект лекций по дисциплине САОД