1	2	3	4	5	6
[0]		∞			∞
	[11]	∞	[12]		∞
		26			∞
		26		[25]	∞

5. На следующем этапе, вычисляя расстояния от вершины 5 с постоянной меткой 25, приходим к конечной вершине B. Но ее метка 25+14=39 не становится постоянной, так как она не является минимальной. От вершины 5 до вершины 3 расстояние ∞ (они не соединены). Прежнее значение временной метки вершины 3 меньше ∞. Поэтому метка вершины 3 не меняется. Метка вершины 3 со значением 26 меньше 39 становится постоянной и от нее на следующем этапе ищем расстояния.

1	2	3	4	5	6
[0]					∞
	[11]		[12]		∞
					∞
				[25]	∞
		[26]			39

6. От вершины 3 до вершины 6 расстояние 20, так как 26+20>39, то значение метки 6 не меняем. На этом шаге она остается прежней и единственной временной меткой. Временная метка вершины 6 становится постоянной, что означает конец процесса. Минимальное расстояние от A до B равно 39.

Две Maple-программы для определения кратчайшего пути в орграфе приведены на с. 81 и с. 83.

4.2. Поток в сети

Сетью называют взвешенный орграф с двумя выделенными вершинами: истоком и стоком. Исток имеет нулевую полустепень захода, а сток нулевую степень исхода. Вес дуг означает как правило пропускную способность дуги. Задача о наибольшем потоке в сети не единственная, но вероятно, основная задача для потоков в сети. Очевидно практическое применение решения этой задачи для решения транспортных

проблем (пробки на дорогах это и есть насыщение сети или отдельной ее дуги), проблем транспортировки нефтепродуктов или электричества.

Задача. *Задана пропускная способность дуг транспортной сети (рис. 35) с началом в вершине* 1 *и концом в вершине* 8. *Используя алгоритм Форда-Фалкерсона[13], найти максимальный поток по сети.*

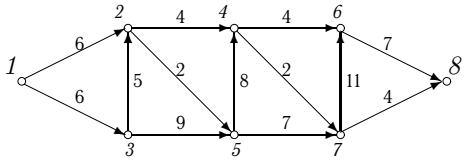


Рис. 35

Решение

Алгоритм из двух частей — насыщение потока и его перераспределение. В первой части задача ставится локально. Дуги рассматриваются отдельно (возможно хаотично) и каждой дуге приписывается возможно больший поток, согласованный только с условием сохранения в узлах (вершинах). Во второй части перераспределения потока выполняется из условия достижения общего по сети максимума потока.

1. **Насыщение потока**. Поток называется насыщенным, если любой путь из истока (1) в сток (8) содержит насыщенную дугу.

Рассмотрим путь 1-2-4-6-8. Пропустим через этот путь поток равный 4. При этом дуга 2-4 и 4-6 будут насыщенными. Аналогично, путь 1-3-5-7-8 насытим потоком 4. Распределение потока отметим на графе. В числителе ставим пропускную способность, в знаменателе — поток. Числитель всегда больше знаменателя, знаменатель может быть и нулем.

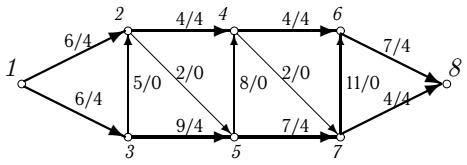


Рис. 36

Заметим, что из 1 в 8 есть еще ненасыщенный путь 1-3-2-5-4-7-6-8, в котором можно увеличить поток на 2. При этом насытятся дуги 1-3, 2-5, 4-7.

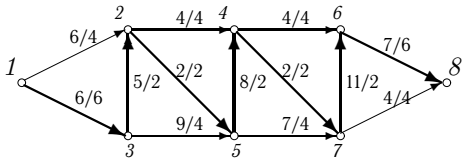


Рис. 37

Из 1 в 8 больше нет ненасыщенных путей. По дуге 1-3 двигаться нельзя (она уже насыщена), а движение по дуге 1-2 заканчивается в вершине 2, так как обе выходящие из нее дуги насыщены.

2. **Перераспределение потока.** Найдем последовательность вершин из 1 в 8, такую, что дуги, соединяющие соседние вершины, направленные из 1 в 8 ненасыщены, а дуги, направленные в обратном направлении не пусты. Имеем единственную последовательность 1-2-3-5-7-6-8. Перераспределяем поток. Поток в дугах прямого направления увеличиваем на 1, а поток в дугах обратного направления уменьшаем на 1. Процесс продолжаем до тех пор, пока одна из прямых дуг будет насыщена или какая-нибудь обратная дуга будет пуста.

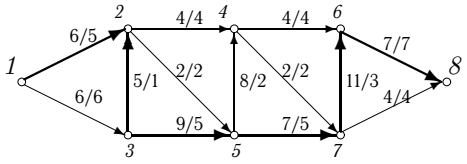


Рис. 38

Поток в насыщенной сети можно посчитать по потоку выходящему из истока 1, или по входящему в 8. Очевидно, эти числа должны быть равны. Кроме этого для проверки решения следует проверить условие сохранения потока по узлам. В каждый узел суммарный входящий поток должен быть равен выходящему. В рассматриваемом примере поток равен 11. Распределение потока по дугам при одном и том же суммарном минимальном потоке по сети неединственно.

Maple-программа для определения максимального потока в сети приведена на с. 94.

4.3. Топологическая сортировка сети

Задача. *Отсортировать топологически сеть на рис. 39.*

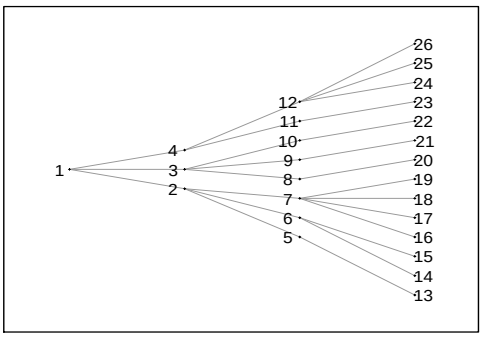
```
> c[1]:=1:# Одна вершина в ярусе 1
> t:=0: h:=1:
> for z while n1-t>0 do
>   t:=t+c[z]:
```

Число вершин яруса

```
>   c[z+1]:=add(kod[n1-t+i],i=1..c[z]);
>   h:=h+c[z+1]; vr:=vr,[seq(j,j=h+1-c[z+1]..h)];
> od:
> vr;       # Ответ. Список вершин
```

[1], [2, 3, 4], [5, 6, 7, 8, 9, 10, 11, 12],
[13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]

```
> addedge([rb],G):   # Добавление к графу ребер
> draw(Linear(vr),G);# Рисунок графа
```



5.19. Поток в сети

В программе даны одновременно два способа вычисления максимального потока в сети (рис. 35, с. 43). Оператор `flow(G,v1,v2,edgsatur)` из пакета `networks` возвращает наибольший поток по сети от источника `v1` в сток `v2`, а в переменную `edgsatur` помещает список насыщенных ребер [1].

Программа 24

```
> restart: with(networks):
> new(G):V:=$1..8: addvertex([V],G):
> v1:=1:# Источник
```

[1] Для орграфов оператор `flow` путает ребра и дуги дает в ответе лишние ребра. Например, для сети с дугами

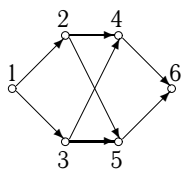
```
> v2:=8:# Сток
> E:=[[1,3],[3,5],[5,7],[7,8],[1,2],[2,4],[4,6],
> [6,8],[3,2],[2,5],[5,4],[4,7],[7,6]]:
> w:=[6,9,7,4,6,4,4,7,5,2,8,2,11]:#Пропускн.способн.
> addedge(E,weights=w,G):
> draw(Linear([1],[3,2],[5,4],[7,6],[8]),G):
> Поток=flow(G,v1,v2,edgsatur);
> edgsatur;                # Насыщенные дуги
```

$$\text{Поток} = 11$$

$$\{\{1, 2\}, \{2, 5\}, \{4, 7\}, \{7, 8\}, \{2, 4\}, \{4, 6\}, \{6, 8\}\}$$

```
> m:=nops(edges(G)):
> H:=duplicate(G):
> potok1:=table([seq(e||i=0,i=1..m)]):# Нач. поток
> while (v1 in vertices(G)) do
> s:=[]: d:={v1}: d2:=v1:
> c1:={v1}: ndep1:=v1:
> while d2<>v2 and ndep1<>0 do
>  d1:=d[1]:            # Начало следующей дуги
>  d:=departures(d1,G):# Множество возможных концов
>  ndep1:=nops(d);
>  if ndep1=0 then delete(d1,G); else
>    d:=d minus c1; # Исключаем пройденные вершины
>    d2:=d[1]:      # Конец дуги
>    nd:=op(edges([d1,d2],G));
>    c1:= c1 union {d2}; # Пополняем список
>    s:=[op(s),nd]:      # Список пройденных дуг
>  fi;
>  od:
> if v2 in c1 then  # Если образовалась цепь
>    n1:=nops(s);   # Длина цепи
>    pt:=[potok1[s[j]]$j=1..n1];
```

`[[1,3],[3,5],[5,6],[1,2],[2,4],[4,6],[2,5],[3,4]]`, имеющими одинаковую пропускную способность равную по умолчанию 1, истоком в вершине 1, стоком в 6 оператор дает правильный ответ: максимальный поток равен 2. Однако при этом оператор возвращает семь насыщенных ребер (не дуг) $\{\{1, 3\}, \{2, 4\}, \{4, 6\}, \{1, 2\}, \{2, 5\}, \{5, 6\}, \{3, 4\}$, где ребро $\{2, 4\}$ явно лишнее.



```
>      sp:=[op(eweight(s,H))];
>      # насыщаем цепь
>      potok2:=map(`+`,pt,min(op(sp-pt)));
>      for i to n1 do potok1[s[i]]:=potok2[i];
>       if potok1[s[i]]=eweight(s[i],H) then
>       delete(s[i],G); end;# Удаляем насыщенные дуги
>      od:
>  fi:
>  end:#while
```

Перераспределение

```
>  H2:=duplicate(H):
>  while (v1 in vertices(H2))  do
>  c1:={}:   # Множество пройденных вершин
>  in2:={}:  # Множество входящих дуг
>  out2:={}: # Множество выходящих дуг
>  d1:=v1: # Первая вершина
>  notupik1:=true;
>  while d1<>v2 and notupik1 do
>  out0:=departures(d1,H2) minus c1;
>  out1:={};
```

Не рассматриваем полные выходящие дуги

```
>  for i in  out0 do
>  nd:=op(edges([d1,i],H2));
>  if eweight(nd,H2)<>potok1[nd] then
>  d2:=i; out1:=edges([d1,i],H2) end;
>  od;
>  out2:=out2 union out1;#Множество прямых дуг цепи
>  in0:=arrivals(d1,H2) minus c1;
>  in1:={};
```

Не рассматриваем пустые входящие дуги

```
>  for i in  in0  do
>    nd:=op(edges([i,d1],H2));
>    if potok1[nd]<>0 then
>    d2:=i; in1:=edges([i,d1],H2); end;
>  od;
>  in2:=in2 union in1;#Множество обратных дуг в цепи
>  if nops(in1 union out1)=0 then # Если d1 тупик
>    delete(d1,H2);   # Удаляем вершину d1
```

```
>     notupik1:=false; # Начинаем поиск заново
>     else
>     c1:=c1 union {d1}; # Присоединяем d1 к пройденным
      Конец(начало) последней дуги - новая вершина для поиска
>     d1:=d2;
>   fi;
>   od;
>   pr1:=(x)->eweight(x,H)-potok1[x]; # Процедура 1
>   pr2:=(x)->potok1[x]:              # Процедура 2
>   if notupik1 then# Перераспределяем поток
>     m1:=min(op(map(pr1,out2)));
>     m2:=min(op(map(pr2,in2)));
>     ptk:=min(m1,m2);
>     for i in in2 do potok1[i]:=potok1[i]-ptk;od:
>     for i in out2 do potok1[i]:=potok1[i]+ptk;od:
>   fi;
>   od:#while
>   edg2:=incident(v2,H,'In'):#Дуги, входящие в сток
>   Поток=map('+',op(eweight([op(edg2)],H)));
>   satur1:=[]:
>   for x in edges(H) do
>     if pr1(x)=0 then satur1:=[op(satur1),x];fi;
>   od;
>   satur1;  # Насыщенные дуги
```

$$\text{Поток} = 11$$

$$[e4,\ e5,\ e6,\ e7,\ e10,\ e12,\ e8]$$

5.20. Топологическая сортировка сети

Вводятся данные сети на рис. 39, с. 45. В алгоритме используется удобная функция `delete`, позволяющая удалять список вершин и соответствующие дуги. Каждый уровень организуется в виде списка для того, чтобы использовать результаты в операторе `Linear` топологически упорядоченной сети. После формирования списка вершин одного уровня эти вершины удаляются из графа, и работа цикла `while NV<>0 do` продолжается до тех пор, пока множество вершин не станет пустым. Счетчик числа вершин - `NV`. Копия графа `H` выводится на экран по полученным уровням.

Если граф содержит цикл, то на очередном этапе поиска вершины с нулевой полустепенью захода список уровня окажется пустым и