Тема 10 Сетевые модели

Введение

Сетевые модели могут строиться:

- в терминах событий: вершины- события, дуги - взаимосвязь событий;
- в терминах работ: вершины- работы, дуги - взаимосвязь работ;
- в терминах работ и событий: вершины - события результата работ (начало либо завершение), дуги - сами работы.

С помощью сетевых моделей можно решить множество задач. Рассмотрим следующие конкретные примеры.

1. Проектирование газопровода, соединяющего буровые скважины в Мексиканском заливе с находящейся на берегу приемной станцией. Следует выбрать проект, в котором строительство газопровода имеет минимальную стоимость.

2. Определение кратчайшего пути между двумя городами, проходящего по существующей сети шоссейных дорог.

3. Определение максимальной пропускной способности (в тоннах/год) трубопровода для транспортировки угольной пульпы с шахт Вайоминга на тепловые электростанции в Хьюстоне. (Уголь под напором воды поступает в специально спроектированный трубопровод и перегоняется с шахт в пункты назначения.)

4. Определение наиболее экономичной (имеющей минимальную стоимость) схемы транспортировки нефти из пунктов нефтедобычи на перерабатывающие заводы и далее в центры распределения. Сырую нефть и нефтепродукты можно транспортировать на танкерах, грузовиках или по нефтепроводу. Помимо условий, определяющих верхний предел для добычи нефти и нижний предел для удовлетворения спроса на нее, необходимо принимать во внимание ограничения мощности нефтеперерабатывающих заводов и пропускных способностей соответствующих транспортных коммуникаций.

Анализ указанных примеров показывает, что оптимизационные задачи на сети можно описать следующими четырьмя типами моделей:

1) минимизации сети;
2) нахождения кратчайшего маршрута;
3) определения максимального потока;
4) минимизации стоимости потока в сети с ограниченными пропускными способностями коммуникаций.

Рассмотренные примеры связаны с определением расстояний и материальных потоков. Однако во многих приложениях переменные могут представлять величины иной природы, как, например, потоки запасов или денег.

Перечисленные выше сетевые задачи можно сформулировать и решить как задачи линейного программирования. Однако специальная структура этих задач позволяет разработать более эффективные алгоритмы, которые в большинстве случаев основываются на теории линейного программирования.

Модель определения потока минимальной стоимости в сети с ограниченными пропускными способностями имеет широкий круг практических приложений. Задачу о кратчайшем пути и задачу о максимальном потоке можно рассматривать как частные случаи транспортной задачи с ограниченными пропускными способностями. Однако метод решения задачи с ограниченными пропускными способностями неудобен, т.к. включает большой объём вычислительных процедур.

Необходимо подчеркнуть, что принцип оптимальности является основой поэтапного решения задачи, однако он не содержит информации о способах решения подзадач, возникающих на данном этапе. В связи с этим принцип оптимальности иногда рассматривается как слишком общий для того, чтобы быть полезным в практических исследованиях.

1. Определение кратчайшего маршрута для сети без циклов

Наиболее просто решается задача определения пути для сети без циклов, т.е. не имеющей путей возврата. Пусть сеть упорядочена, так что номера пунктов предшественников всегда меньше номеров

последующих пунктов. Для упрощения анализа упорядоченная сеть разбивается на ряд последовательных этапов.

Введем следующие обозначения:

d_{ij} – расстояние на сети между *смежными* узлами i и j.

u_j – кратчайшее расстояние между узлами 1 и j, $u_1=0$.

Общая формула для вычисления u_j имеет вид: $u_j= \min_i \{u_j+d_{ij}\}$

(кратчайшее расстояние до предыдущего узла i плюс расстояние между текущим узлом j и предыдущим узлом i).

Из этой формулы следует, что кратчайшее расстояние u_j до узла j можно вычислить лишь после того, как определено кратчайшее расстояние до каждого предыдущего узла i, соединенного дугой с узлом j.

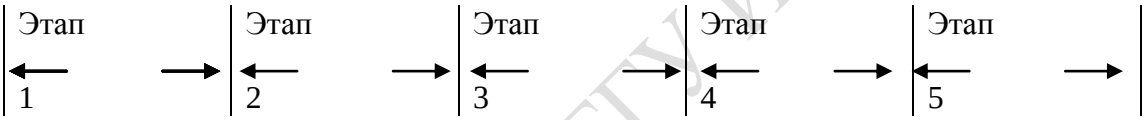
Минимальное расстояние между начальным и конечным узлами находится в конце прямого хода вычислений. Оптимальный маршрут определяется при обратном проходе с использованием условия $u_i= u_j-d_{ij}$.

Заметим также, что полученное решение дает кратчайшее расстояние между узлом 1 и любым из других узлов сети.

Представленный тип вычислений интересен тем, что он имеет **рекурсивный характер.** Вычисления выполняются с использованием информации обо всех кратчайших расстояниях до *непосредственно предшествующего узла.*

Рекурсивные вычисления представляют собой основу вычислительной схемы динамического программирования.

Рассмотрим алгоритм на примере сети, представленной на рис.1.1. Предварительно сеть упорядочена и разбита на последовательность этапов, определяющих ход вычислений.



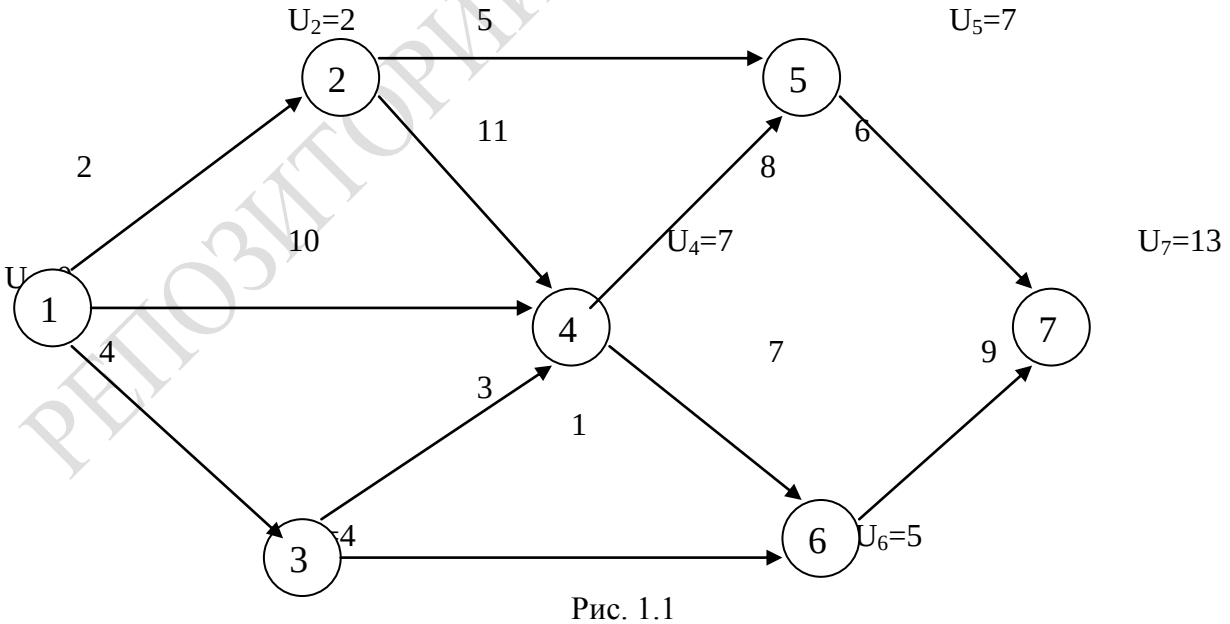


Рис. 1.1

Узел 1 представляет начальную точку (исходный пункт), а узел 7- конечную точку (пункт назначения). Заметим, что сеть не имеет циклов, поскольку нет ни одной цепи, связывающей узел с самим

собой. Для узла 1 можно вычислить лишь u_2 и u_3. (Заметим, что, хотя узел 4 соединен узлом 1 дугой, соответствующее значение u_4 вычислить нельзя, пока не будут определены u_2 и u_3).
Процедура завершается, когда получено значение u_7.

Вычислительная схема состоит из следующих этапов:

- ***Этап 1:*** $u_1=0$.
- ***Этап 2:*** $u_2= u_1+d_{12}=0+2=2$).
  $u_3= u_1+d_{13}=0+4=4$.
- ***Этап 3:*** $u_4= \min \{u_1+d_{14}, u_1+d_{24}, u_1+d_{34} \} = \min \{0+10, 2+11, 4+3 \} =7$.
- ***Этап 4:*** $u_5= \min \{u_2+d_{25}, u_4+d_{45}\}= \min \{2+5, 7+8 \} =7$.
  $u_6= \min \{u_3+d_{36}, u_4+d_{46} \}= \min \{4+1, 7+7 \}=5$.
- ***Этап 5:*** $u_7= \min \{ u_5+d_{57}, u_6+d_{67} \} = \min \{7+6, 5+9 \} =13$.

Минимальное расстояние между узлами 1 и 7 равно 13, а соответствующий маршрут-$1\rightarrow2\rightarrow5\rightarrow7$, который находится на обратном проходе с использованием условия : $u_i= u_j-d_{ij}$.

Заметим также, что полученное решение дает кратчайшее расстояние между узлом 1 и любым из других узлов сети.

Вычисления выполняются с использованием информации обо всех кратчайших расстояниях до *непосредственно предшествующего узла.* Например, в узле 5 величина u_5 вычисляется по кратчайшим расстояниям между узлом 1 и узлами 2 и 4, т.е. u_2 и u_4. Заметим, что не обязательно знать конкретный маршрут, дающий кратчайшее расстояние между узлами 1 и 4. Величина u_4 включает всю информацию, необходимую для узла 4. Именно такая информация позволяет использовать рекурсивные вычисления.

Все вычисления можно провести непосредственно на сети (рис. 1.1).
Для решения таких задач используется теория графов и алгоритмы, разработанные на основе теории графов.

2. Теоретические основы теории графов

2.1. Основные определения графа.

В настоящее время теория графов привлекает все большее внимание специалистов самых разных областей науки и техники, являясь эффективным аппаратом формализации современных инженерных задач, связанных с дискретными объектами. Такие задачи возникают при проектировании интегральных схем и схем управления, при исследовании автоматов и логических цепей, системном анализе, автоматизированном управлении производством, в теории расписаний и дискретной оптимизации.

Родившись при решении головоломок и занимательных задач (задача о кенигсбергских мостах, о шахматном коне, о ферзях, «кругосветное путешествие» и т.п.), теория графов стала в настоящее время простым, доступным и мощным средством решения вопросов в различных областях знаний. Она является очень наглядной для анализа путей и маршрутов.

Чтобы составить наглядное представление о графе, достаточно вообразить некоторое множество точек плоскости или пространства (вершины) и множество отрезков прямых или кривых линий (ребра или дуги), соединяющих все или некоторые из этих точек. Если в паре вершин x_i и x_j указанно направление связи, то соединяющий их отрезок называется *дугой*, если же ориентация не указана, - *ребром*.

Если в графе все ребра являются дугами, то он называется *ориентированным* (орграфом), если ни одно не является дугой, - то *неориентированным.* Иногда рассматривают *смешанные* графы, состоящие из дуг и ребер.

Путем в орграфе называется последовательность дуг, в которой конец каждой предыдущей дуги совпадает с началом следующей. Путь, проходящий через все вершины, и притом только по одному разу, называется *гамельтоновым*. Путь, содержащий все дуги графа и притом только по одному разу, называется *эйлеровым*. Конечный путь, у которого начальная вершина совпадает с конечной, называется *контуром.*

В неориентированном графе последовательность ребер, в которой каждые два соседних ребра смежны, называется *цепью,* а конечный путь, у которого начальная и конечная вершины совпадают, - циклом.

В различных приложениях теории графов дугам (ребрам) графов, моделирующим реальные процессы, обычно сопоставляют какие-либо числовые характеристики. Например, если дугами изображаются транспортные магистрали, то числовой характеристикой дуги может быть пропускная способность соответствующей магистрали и т. д. В подобных случаях говорят, что дугам графа приписаны определенные *веса*, а сам граф с весами на дугах называют *взвешенным*.

Граф часто обозначают символом G (V,E), G от английского Graph, V от Vertices – вершины, E от Edges – ребра. Взвешенные графы также называют *сетями*, их часто обозначают N (V,E,W), где N от английского Network – сеть, а W от Weight – вес.

Циклом называется цепь из V в V.

Часто в задачах встречается следующая конструкция - есть дома и дороги, их соединяющие; у каждой дороги есть длина. В терминах же теории графов, дома называются "вершинами", дороги - "ребрами" или "дугами", а длина дороги "весом ребра" или "весом дуги". Таким образом фраза 'Найти минимальный вес пути между вершинами s и k в графе' может быть переведена так: 'Есть дома и дороги их соединяющие. Также заданы длины дорог. Найти кратчайшую длину пути от города s до города k, если двигаться можно только по дорогам.

2.2. Задача о кратчайшем пути.

Задача о кратчайшем пути состоит в нахождении *связанных* между собой дорог на транспортной сети, которые в совокупности имеют минимальную длину от исходного пункта до пункта назначения.

Пусть дан граф G = (V,E) дугам которого приписаны веса (стоимости), задаваемые матрицей C = $[c_{i,j}]$. Задача о кратчайшем пути состоит в нахождении самого короткого пути от заданной начальной вершины $s \in V$ до заданной конечной вершины $t \in V$, при условии, что такой путь существует, т.е. при условии, что t принадлежит множеству, достижимому из вершины s. Элементы $c_{i,j}$ матрицы весов C могут быть положительными, отрицательными или нулями, что в большей степени и определяет выбор алгоритма. Единственное ограничение состоит в том, чтобы в G не было циклов с отрицательным суммарным весом.

Существует ряд алгоритмов решающих задачу нахождения оптимального варианта пути между заданными узлами:

- *Волновой алгоритм*

Позволяет найти один или все возможные пути между двумя заданными вершинами графа (орграфа), количество промежуточных вершин в которых минимально.

- *Алгоритм Дейкстры*

Позволяет найти один или все возможные пути минимальной суммарной длины между двумя заданными вершинам взвешенного графа(орграфа) с неотрицательными весами ребер (дуг).

- *Алгоритм Форда*

Позволяет найти один или все возможные пути минимальной суммарной длины между двумя заданными вершинами взвешенного графа (орграфа) с произвольными весами ребер (дуг).

- *Алгоритм Флойда*

Используется, если надо найти кратчайшие пути между всеми парами вершин взвешенного графа (орграфа) с произвольными весами ребер (дуг).

- *Алгоритм Форда - Беллмана*

Для случая, когда не существует замкнутых маршрутов, для которых сумма цен отрицательна. Найти в этом случае маршрут с наименьшей стоимостью.

- *Алгоритм Йена*

Позволяет находить k-кратчайшие пути без циклов последовательно

- *Алгоритм Флойда-Уоршелла*

Представляет последовательность операции над матрицей, в которой хранятся расстояния из одного города в другой

- *Алгоритм Шимбелла*

Находит по матрице кратчайшие расстояния между всеми парами вершин.

Можно дать много практических интерпретаций задачи о кратчайших путях.

Например, вершины могут соответствовать городам, а каждая дуга – некоторому пути, длина которого представлена весом дуги. Мы ищем затем кратчайшие пути между городами.

Вес дуги также может соответствовать стоимости (или времени) передачи информации между вершинами. В таком случае мы ищем самый дешевый (или самый скорый) путь передачи информации.

Ещё одну ситуацию мы получаем, когда вес дуги $<x_i,x_j>$ равен вероятности $p(x_i,x_j)$ безаварийной работы канала передачи информации. Если предположить, что аварии каналов не зависят друг от друга, то вероятность исправности пути передачи информации равна произведению вероятностей составляющих его дуг. Задачу нахождения наиболее надёжного пути легко можно свести к задаче о кратчайшем пути.

Наиболее простой случай задачи о кратчайших путях - если все цены равны 0 или бесконечны. Другими словами, мы интересуемся возможностью попасть из i в j, но за ценой не постоим. В других терминах: мы имеем ориентированный граф и нас интересуют вершины, доступные из данной.

Сейчас нас интересует такая задача: не просто перечислить все вершины, доступные из данной, но перечислить их в определенном порядке. Два популярных случая - поиск в ширину и в глубину.

Поиск в ширину:

Надо перечислить все вершины ориентированного графа, доступные из данной, в порядке увеличения длины пути от нее.

Начиная с произвольной вершины приписываем ей номер 0. Каждой вершине из окружения приписываем номер 1, и т. д. Если исходный граф *связен*, т. е. существует путь между любыми двумя вершинами, то поиск в ширину занумерует все его вершины. Таким образом можно найти кратчайшую цепь и множество всех вершин, достижимых из заданной.

Поиск в глубину:

Рассматривая поиск в глубину, удобно представлять себе ориентированный граф как образ дерева. Более точно, пусть есть ориентированный граф, одна из вершин которого выделена. Будем полагать, что все вершины доступны из выделенной по ориентированным путям. Построим дерево, которое можно было бы назвать "универсальным накрытием" нашего графа. Его корнем будет выделенная вершина графа. Из корня выходят те же стрелки, что и в графе - их концы будут сыновьями корня. Из них в дереве выходят те же стрелки, что и в графе и так далее. Разница между графом и деревом в том, что пути в графе, ведущие в одну и ту же вершину, в дереве "расклеены". В других терминах: вершина дерева - это путь в графе, выходящий из корня. Ее сыновья - это пути, продолженные на одно ребро. Заметим, что дерево бесконечно, если в графе есть ориентированные циклы. Имеется естественное отображение дерева в граф (вершин в вершины). При этом каждая вершина графа имеет столько прообразов, сколько путей в нее ведет. Поэтому обход дерева (посещение его вершин в том или ином порядке) одновременно является и обходом графа - только каждая вершина посещается многократно. Будем предполагать, что для каждой вершины графа выходящие из нее ребра упорядочены (например, пронумерованы). Тем самым для каждой вершины дерева выходящие из нее ребра также упорядочены. Будем обходить дерево так: сначала корень, а потом поддеревья (в порядке ведущих в них ребер). Если выкинуть из этого обхода повторные посещения уже посещенных вершин, то получится то, что называется "поиск в глубину".

Отыскание кратчайшего пути имеет естественную интерпретацию **в терминах матриц**. Пусть A - матрица цен одной авиакомпании, а B - матрица цен другой. (Мы считаем, что диагональные элементы матриц равны 0.) Пусть мы хотим лететь с одной пересадкой, причем сначала самолетом компании A, а затем - компании B. Сколько нам придется заплатить, чтобы попасть из города i в город j?

Можно доказать, что эта матрица вычисляется по обычной формуле для произведения матриц, только вместо суммы надо брать минимум, а вместо умножения - сумму.

Обычное (не модифицированное) умножение матриц тоже может оказаться полезным, только матрицы должны быть другие. Пусть есть не все рейсы, а только некоторые, a[i,j] равно 1, если рейс есть, и 0, если рейса нет. Возведем матрицу a (обычным образом) в степень k и посмотрим на ее i-j-ый элемент. **О**н равен числу различных способов попасть из i в j за k рейсов.

Случай, когда есть не все рейсы, можно свести к исходному, введя фиктивные рейсы с бесконечно большой (или достаточно большой) стоимостью.

3. Нахождение кратчайшего пути в сети с циклами

3.1. Схема алгоритма Дейкстры*:*

Для решения данной задачи рассмотрим ***алгоритм Дейкстры***, как классический пример нахождения кратчайшего пути в сети с циклами.

Граф может быть как ориентированным, так и нет. Требуется найти самый короткий простой путь для двух выделенных вершин графа.

Для нахождения пути каждой вершине будем приписывать следующую информацию:

1. длину кратчайшего пути от начальной вершины до данной вершины;
2. признак, говорящий о том, что
   - мы не были в данной вершине;
   - длину пути до этой вершины можно уменьшить;
   - длину пути до этой вершины уменьшить нельзя;
3. Ссылку на метку предыдущей вершины в пути.

Нахождение пути проводится в три этапа:

- на первом этапе создается начальная разметка;
- на втором этапе разметка изменяется;
- а на последнем этапе по разметке восстанавливается путь.

Разметка вершин делается так:

1. Для всех вершин, кроме начальной, устанавливаем равным бесконечности признак того, что мы здесь еще не были; оставляем неопределенными длину пути и ссылку на предыдущую вершину.
2. Для начальной вершины длину пути устанавливаем равной нулю, признак того, что длину пути уменьшить нельзя, номер предыдущей вершины оставляем неопределенным.

До тех пор, пока можно уменьшить длину до конечной точки пути, делаем следующее:

1. уменьшаем длины путей до вершин

Для этого смотрим, к каким вершинам ведут ребра от вершины, признак которой был установлен в значение 'длину пути уменьшить нельзя' последним.

Для каждой из этих вершин выясняем, что меньше: длина пути, приписанная вершине ранее или длина пути полученного при добавлении ребра. При необходимости корректируем длину пути до вершины и номер предыдущей вершины в пути.

2. из всех вершин, признак которых говорит о возможности уменьшения длины пути, выбираем вершину, длина пути до которой минимальна. Говорим, что длину пути до этой вершины уменьшить нельзя.

3.Наконец, мы найдем длину неуменьшаемого пути до конечной вершины.

Теперь осталось восстановить сам путь. Так как мы в каждой вершине запоминали, откуда мы пришли, то это делается просто: узнаем из какой вершины мы пришли в конечную, затем узнаем, из ка-

кой вершины мы пришли в вершину перед конечной и так далее, пока не окажемся в начальной вершине.

Алгоритм использует три массива из N чисел каждый.

- Первый массив S содержит метки с двумя значения:
  0 (вершина еще не рассмотрена) и 1 (если уже рассмотрена);
- второй массив B содержит расстояния - текущие кратчайшие расстояния от и до соответствующей вершины;
- третий массив C содержит номера вершин - k-й элемент C[k] есть номер предпоследней вершины на текущем кратчайшем пути из V_i в V_k.

Матрица расстояний A[i,k] задает длины дуге (i,k); если такой дуги нет, то A[i,k] присваивается большое число M, равное "машинной бесконечности".

```
1. (Инициализация). В цикле от 1 до N заполнить нулями массив S; заполнить
числом i массив C; перенести i-ю строку матрицы A в массив B, S[i]:=1;
C[i]:=0 (i - номер стартовой вершины).
```

2. (Общий шаг). Найти минимум среди неотмеченных (т. е. тех k, для которых S[k]=0). Пусть минимум достигается на индексе j, т. е. B [j] ≤B[k].

Затем выполняются следующие операции:

S[j]:=1;

если B[k] > B[j]+A[j,k], то B[k]:=B[j]+A[j,k]; C[k]:=j

Условие означает, что путь V_i ... V_k длиннее, чем путь V_i...V_j V_k.

Если все S[k] отмечены, то длина пути от V_i до V_k равна B[k]. Теперь надо перечислить вершины, входящие в кратчайший путь.

3. (Выдача ответа). Путь от V_i до V_k выдается в обратном порядке следующей процедурой:

3.1. z:=C[k];

3.2. Выдать z;

3.3. z:=C[z]. Если z = O, то конец, иначе перейти к 3.2.

Для выполнения алгоритма нужно N раз просмотреть массив B из N элементов, т. е. алгоритм Дейкстры имеет квадратичную сложность. Алгоритм Дейкстры не может быть использован, если веса некоторых дуг отрицательны. В этом случае для поиска кратчайшего пути в графе с отрицательным весом применяется Алгоритм Форда, являющийся модификацией алгоритма Дейкстры.

3.2. Определение кратчайшего пути на сети с циклами с использованием матрицы

Введем вспомогательные величины:

$$\mathbf{V_j = \min_i \{U_i + d_{ij}\}},$$

где: U_i – кратчайшее растояние между узлами 1 и j,

d_{ij} – длина дуги между смежными узлами.

Процесс начинается с $\mathbf{i = 1}$ и $\mathbf{V_1 = U_1 = 0}$. После определения $\mathbf{V_j}$ полагаем $\mathbf{U_i = V_j}$. Заметим, что $\mathbf{U_i}$ включает расстояние до узла **i**, которое затем используется для определения расстояния до ближайшего узла.

Формула рекурсивная и для сети без циклов решение получается за один проход. При наличии циклов (имеется путь меньшей длины от узлов с большими номерами к меньшим номерам) необходим итерационный процесс, по сути напоминающий метод потенциалов в транспортной задаче.

Конец итерационного процесса определяется по условию

$$\mathbf{d_{ij} \geq V_j - U_i}$$

для всех строк матрицы. При обнаружении несоответствия в строке вычислить новые $\mathbf{V_j = U_i + d_{ij}}$ для j не удовлетворяюших условию, положить $\mathbf{U_j = V_j}$ и продолжить расчет.

3.3. Пример нахождения кратчайшего пути на сети с циклами

Рассмотрим задачу нахождения кратчайшего пути в сети с циклами на конкретном примере.

Дана следующая сеть (рис. 1):

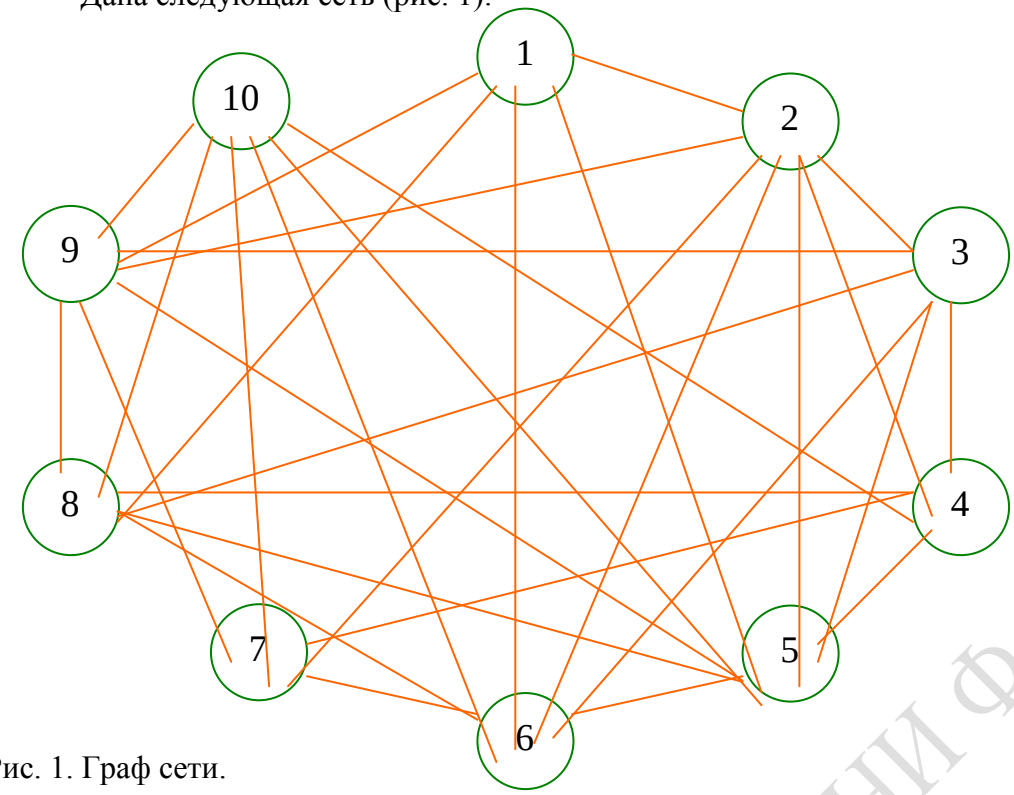


Рис. 1. Граф сети.

Необходимо найти кратчайший путь из пункта 1 в пункт 10 и обратно.

Определим кратчайший путь на сети с циклами с использованием матрицы.

Пусть расстояния между узлами приведены в виде следующей табл. 1:

Таблица 1. Матричное представление графа сети

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**U_i**
1		3			8	11					
2			7	9	2		2				
3		4		15		7		9	3		
4					4		5			4	
5		8	2			3		4	1		
6	4	2	1				7	6		7	
7		12		5		2			4	8	
8	7		6	8		3					
9	4	1	3		7		13	2		6	
10				1	4		1	8	5		
V_j											

Применяя рекурсивную формулу для расчета расстояний, найдем опорное решение (табл. 2).

Таблица 2. Опорное решение

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**U_i**
1		3			8	11					0
2			7	9	2		2				3
3		4		15		7		9	3		10
4					4		5			4	12
5		8	2			3		4	1		5
6	4	2	1				7	6		7	8
7		12		5		2			4	8	5

8	7		6	8		3					9
9	4	1	3		7		13	2		6	6
10				1	4		1	8	5		12
V_j	0	3	10	12	5	8	5	9	6	12	

Проверяем условие: $d_{ij} \geq V_j - U_i$.

В строке 5 условие не выполнено для d_{53}, проводим перерасчет с учетом найденных элементов $V_3 = U_5 + d_{53} = 7$, полагаем $U_3 = V_3 = 7$ и продолжаем проверку. Следующие несоответствующие элементы d_{74}, d_{76}, d_{98}. Результаты приведены в дополнительной строке и столбце таблицы 3.

Таблица 3. Окончательное решение

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**U_i**	
1		3			8	11					0	
2			7	9	2		2				3	
3		4		15		7		9	3		10	7
4					4		5			4	12	10
5		8	2			3		4	1		5	
6	4	2	1				7	6		7	8	7
7		12		5		2			4	8	5	
8	7		6	8		3					9	8
9	4	1	3		7		13	2		6	6	
10				1	4		1	8	5		12	
V_j	0	3	10	12	5	8	5	9	6	12		
			7	10		7		8				

В результате получили матрицу расстояний от пункта 1 к любому другому.

Найдем кратчайший путь к пункту 1, используя условие из рекуррентной формулы: $U_i = V_j - d_{ij}$ (табл.4).

Таблица 4. Решение для поиска пути из первого пункта в последний.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**U_i**	
1		3			8	11					0	√
2			7	9	2		2				3	√
3		4		15		7		9	3		7	
4					4		5			4	10	
5		8	2			3		4	1		5	√
6	4	2	1				7	6		7	7	
7		12		5		2			4	8	5	
8	7		6	8		3					8	
9	4	1	3		7		13	2		6	6	√
10				1	4		1	8	5		12	√
V_j	0	3	7	10	5	7	5	8	6	12		

Кратчайший путь: 1→2→5→9→10 с расстоянием в 12 единиц.

Для нахождения обратного пути матрица просчитывается от конечного пункта с учетом цикличности сети.

В результате получим опорное решение (табл.5).

Таблица 5. Решение для поиска пути из последнего пункта в первый.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**U_i**	

1		3			8	11					7	√
2			7	9	2		2				5	
3		4		15		7		9	3		4	
4					4		5			4	1	
5		8	2			3		4	1		4	
6	4	2	1				7	6		7	3	√
7		12		5		2			4	8	1	√
8	7		6	8		3					7	
9	4	1	3		7		13	2		6	5	
10				1	4		1	8	5		0	√
$\mathbf{V_j}$	7	5	4	1	4	3	1	7	5	0		

Т. к. условие $\mathbf{d_{ij} \geq V_j - U_i}$ выполняется везде, то по таблице начиная с левой верхней ячейки (с конца) находим обратный путь:
в первой колонке для $\mathbf{U_i}$ =7 1- 6, в шестой колонке для $\mathbf{U_i}$ =3 6-7, в седьмой колонке для $\mathbf{U_i}$ =1 7-10, так что окончательно путь длины 7 определяет цепь 10→7→6→1.

Определим кратчайший путь на сети с использованием алгоритма *Дейкстры*.

Построим дерево, соответствующее заданной сети. Дерево строится не до всех конечных узлов путей возможных для данной вершины графа, а лишь для тех, которые ведут к заданной конечной вершине. Причем вершины, для которых не имеет смысла продолжать путь, оставляются без дальнейшего внимания.

Корнем дерева возьмем соответствующий начальный узел сети (вершину графа) - 1 (рис.2).

Помечаем вершину 1.

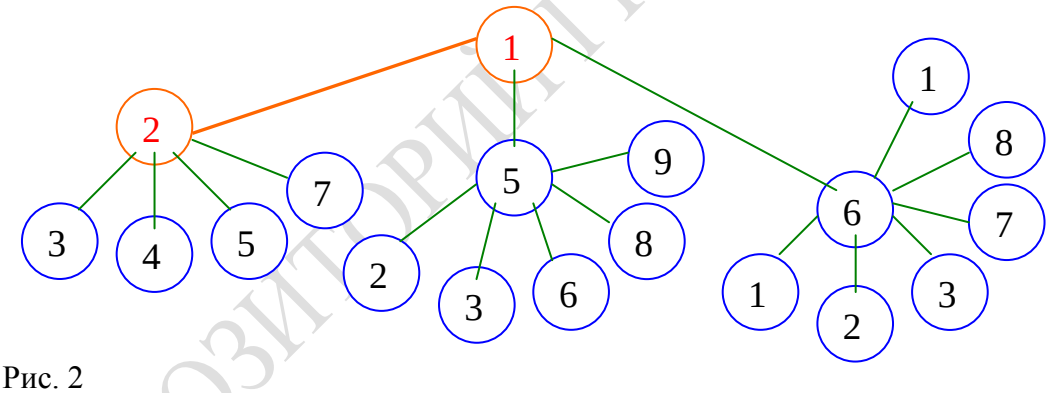


Рис. 2

Сначала пойдем по левой ветви, т. к. до вершины 2 путь наименьший. Помечаем вершину 2 (рис.3).

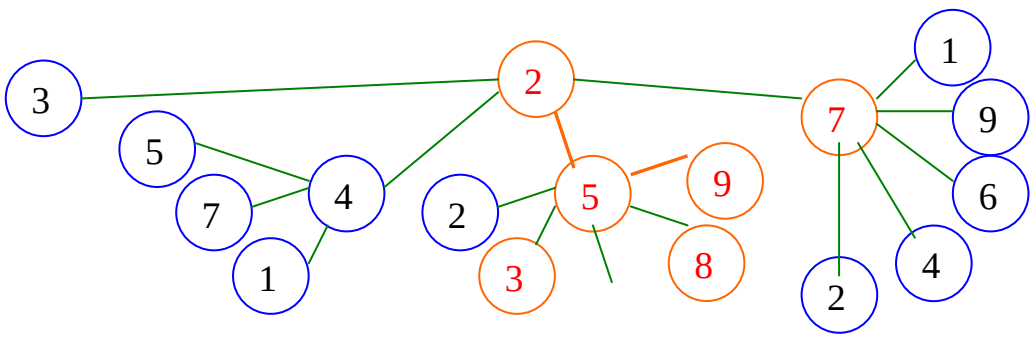


Рис. 3.

Далее опять идем по меньшему пути , т. е. к вершинам 5 и 7. При этом, т. к. пути 2→3, 1→5, 1→6 оказываются длиннее, мы их не будем рассматривать. В результате подобных рассуждений мы получим три пути:

1. 1→2→4→10 (длина = 16)
2. 1→2→7→4→10 (длина = 14)
3. 1→2→7→10 (длина = 13)

Осталось рассмотреть вершины 3, 8 и 9, исходящие из пятой. Результаты представлены на рис.4.

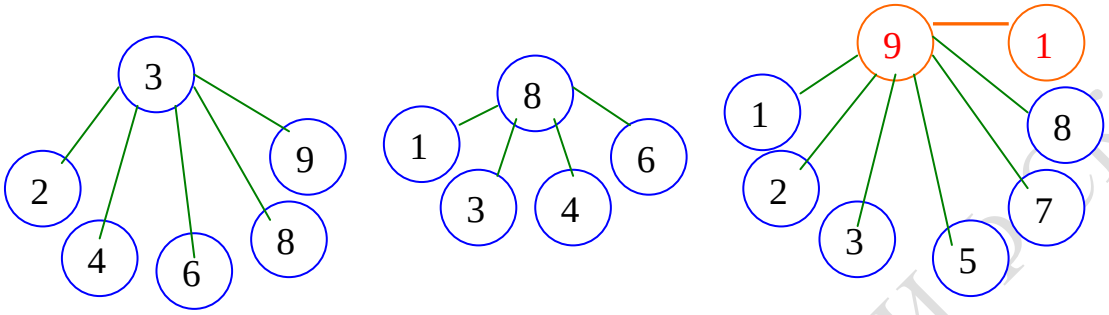


Рис. 4.

Таким образом нашли еще один путь
1→2→5→9→10 (длина = 12),
причем все остальные пути оказались заведомо большими.
Сравнивая полученные результаты, выбираем кратчайший:
1→2→5→9→10 (длина = 12).

4. Максимальный поток

4.1. Алгоритм нахождения максимального потока

Рассматривается задача определения максимального потока между двумя выделенными узлами связной сети. Каждая дуга сети обладает пропускными способностями в обоих направлениях, которые определяют максимальное количество потока, проходящего по данной дуге. Ориентированная (односторонняя) дуга соответствует нулевой пропускной способности в запрещенном направлении.

Пропускные способности c_{ij} сети можно представить в матричной форме. Для определения максимального потока из источника s в сток t используются следующие шаги.

Шаг 1. Найти цепь, соединяющую s с t, по которой поток принимает положительное значение в направлении s→*t.* Если такой цепи не существует, перейти к шагу 3. В противном случае перейти к шагу 2.

Шаг 2. Пусть c^-_{ij} — пропускные способности дуг цепи (s, t) в направлении s→t(t→s) и

$\theta=\min\{c^-_{ij}\}>0$

Матрицу пропускных способностей C_{ij} можно изменить следующим образом:

(а) вычесть θ из всех c^-_{ij};

(б) прибавить θ ко всем c^+_{ij}. При этом общая пропускная способность сети не изменится. Перейти к шагу 1.

Операция (а) дает возможность использовать остатки пропускных способностей дуг выбранной цепи в направлении s→t. Операция (б) восстанавливает исходные пропускные способности сети, поскольку уменьшение пропускной способности дуги в одном направлении можно рассматривать как увеличение ее пропускной способности в противоположном направлении.

Шаг 3. Найти максимальный поток в сети. Пусть $C=\|c_{ij}\|$ — исходная матрица пропускных способностей, и пусть $C^*=\|c^*_{ij}\|$ — последняя матрица, получившаяся в результате модификации исходной матрицы (шаги 1 и 2).

Оптимальный поток $X=\|x_{ij}\|$ в дугах задается как

$$x_{ij} = \begin{cases} c_{ij} - c^*_{ij}, c_{ij} > c^*_{ij} \\ 0, c_{ij} \le c^*_{ij} \end{cases}.$$

Максимальный поток из s в t равен

$$z = \sum_i x_{si} = \sum_j x_{jt}.$$

4.2. Пример нахождения максимального потока

Рассмотрим сеть на рис.5 с данными пропускными способностями.

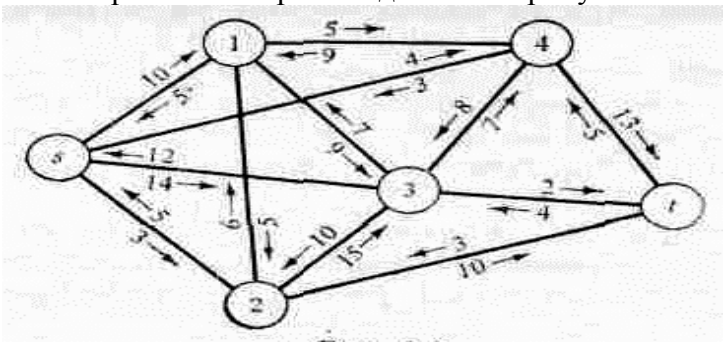


Рис. 5. Граф сети.

Матрица пропускных способностей C приведена в табл. 6.

Табл. 6 . Цепь (s→1→4→t), θ=5

	s	1	2	3	4	t
s		10	3	14	4	
1	5		5	9	5	
2	5	6		15		10
3	12	7	10		7	2
4	3	9		8		13
t			3	4	5	

Табл. 7 . Цепь (s→3→2→t), θ=10

	s	1	2	3	4	t
s		5	3	14	4	
1	10		5	9	0	
2	5	6		15		10
3	12	7	10		7	2
4	3	14		8		8
t			3	4	10	

В качестве исходной цепи можно выбрать s→1→4→t. Таким образом, ячейки (s, 1), (1, 4) и (4, t) помечаются знаком (-), а ячейки (1, s), (4, 1) и (t, 4) - знаком (+). Для данной цепи максимальный поток определяется как

$\theta=\min\{c_{s1}, c_{14}, c_{4t}\}=\min\{10, 5,13\}=5.$

Матрица C в табл. 6 корректируется путем вычитания θ=5 из всех элементов, помеченных знаком (-), и сложения со всеми элементами, имеющими знак (+). Результаты приведены в табл. 7.

Результаты последующих итераций приведены в табл. 8-11.

Табл. 8. Цепь (s→1→3→4→t), θ=5. Табл. 9. Цепь (s→1→3→4→t), θ=3.

	s	1	2	3	4	t
s		5	3	4	4	
1	10		5	9	0	
2	5	6		25		0
3	22	7	0		7	2
4	3	14		8		8
t			13	4	10	

	s	1	2	3	4	t
s		0	3	4	4	
1	15		5	4	0	
2	5	6		25		0
3	22	12	0		2	2
4	3	14		13		3
t			13	4	15	

Табл. 10. Цепь (s→3→t), θ=2 .

Табл. 11. Нет стока.

	s	1	2	3	4	t
s		0	3	2	1	
1	15		5	4	0	
2	5	6		25		0
3	24	12	0		2	0
4	6	14		13		0
t			13	4	20	

	s	1	2	3	4	t
s		0	3	4	1	
1	15		5	4	0	
2	5	6		25		0
3	22	12	0		2	2
4	6	14		13		0
t			13	4	18	

Из табл. 11 следует, что между s и t нельзя построить цепей с положительным потоком, поскольку все элементы в столбце t равны нулю. Таким образом, табл. 4.6 дает матрицу $\mathbf{C}^*$. Из табл.6 (матрица **C**) и табл.11 (матрица $\mathbf{C}^*$) вычисляем матрицу оптимального потока, элементы которой $\mathbf{X=C-C^*}$ с заменой отрицательных величин нулями. В табл.12 приведена матрица **X**, а в табл.13 показана исходная матрица с отметкой узлов, через которые происходит сток

Таблица 12. Матрица оптимального потока

	s	1	2	3	4	t
s		10		12	3	
1				5	5	
2						10
3			10		5	2
4						13
t						

.

Таблица 13. Исходная матрица

	s	1	2	3	4	t
s		10	3	14	4	
1	5		5	9	5	
2	5	6		15		10
3	12	7	10		7	2
4	3	9		8		13
t			3	4	5	

Из табл.12 видно, что z=10+12+3=25. Сумма всех θ(=5+10+5+3+2=25) также дает максимальный поток.

5. Календарное планирование

5.1. Историческая справка

До появления сетевых методов календарное планирование программ (планирование во времени) осуществлялось в небольшом объеме. Наиболее известным средством такого планирования был ленточный (линейный) *график Ганта*, задававший сроки начала и окончания каждой операции на горизонтальной шкале времени. Его недостаток заключается в том, что он не позволяет установить зависимости между различными операциями (определяющие в значительной мере темпы реализации программы). В связи с повышением сложности современных программ потребовалась разработка более четких и эффективных методов планирования, обеспечивающих оптимизацию всего процесса осуществления программ. При этом эффективность интерпретируется как минимизация продолжительности выполнения программы с учетом экономических факторов использования имеющихся ресурсов.

Организационное управление программами стало новой областью теоретических и прикладных исследований благодаря разработке двух аналитических методов структурного и календарного планирования, а также оперативного управления программами. Эти методы, разработанные почти одновременно в 1956— 1958 гг. двумя различными группами, получили названия **метод критического пути**

(МКП) и **метод оценки и пересмотра программ (ПЕРТ).** Метод критического пути был предложен фирмой E. I. du Font de Nemours & Company для управления программами строительства, а затем был развит и обобщен фирмой Mauchly Associates. Метод ПЕРТ разработан консультативной фирмой по заказу военно-морского министерства США для календарного планирования научно-исследовательских и опытно-конструкторских работ программы создания ракет “Поларис”.

В методах ПЕРТ и МКП основное внимание уделяется временному аспекту планов в том смысле, что оба метода в конечном счете определяют календарный план программы. Хотя эти методы были разработаны независимо, они отличаются поразительным сходством. Пожалуй, самым существенным различием первоначально было то, что в методе МКП оценки продолжительности операций предполагались детерминированными величинами, а в методе ПЕРТ — случайными. В настоящее время оба метода составляют единый **метод сетевого планирования и управления (СПУ) программами.**
Сетевое планирование и управление (СПУ) программами включает три основных этапа: *структурное планирование, календарное планирование* и *оперативное управление***.**

Этап структурного планирования начинается с разбиения программы на четко определенные операции. Затем определяются оценки продолжительности операций и строится сетевая модель (сетевой график, стрелочная диаграмма), каждая дуга (стрелка) которой отображает работу. Вся сетевая модель в целом является графическим представлением взаимосвязей операций программы. Построение сетевой модели на этапе структурного планирования позволяет детально проанализировать все операции и внести улучшения в структуру программы еще до начала ее реализации. Однако еще более существенную роль играет использование сетевой модели для разработки календарного плана выполнения программы.

Конечной целью этапа календарного планирования является построение календарного графика, определяющего моменты начала и окончания каждой операции, а также ее взаимосвязи с другими операциями программы. Кроме того, календарный график должен давать возможность выявлять критические операции (с точки зрения времени), которым необходимо уделять особое внимание, чтобы закончить программу в директивный срок. Что касается некритических операций, то календарный план должен позволять определять их резервы времени, которые можно выгодно использовать при задержке выполнения таких операций или с позиций эффективного использования ресурсов.

Заключительным этапом является оперативное управление процессом реализации программы. Этот этап включает использование сетевой модели и календарного графика для составления периодических отчетов о ходе выполнения программы. Сетевая модель подвергается анализу и в случае необходимости корректируется. В этом случае разрабатывается новый календарный план выполнения остальной части программы.

5.2. Сетевое представление программы (сетевая модель)

5.2.1. Представление операций

Сетевая модель отображает взаимосвязи между операциями и порядок их выполнения (отношение упорядочения или следования). Как правило, для представления операции используется **стрелка** (ориентированная дуга), направление которой соответствует процессу реализации программы во времени. Отношение упорядочения между операциями задается с помощью событий. **Событие** определяется как момент времени, когда завершаются одни операции и начинаются другие. Начальная и конечная точки любой операции описываются, таким образом, парой событий, которые обычно называют *начальным событием* и *конечным событием.* Операции, выходящие из некоторого события, не могут начаться, пока не будут завершены все операции, входящие в это событие. По принятой в СПУ терминологии каждая операция представляется ориентированной дугой, а каждое событие — узлом (вершиной). Не требуется, чтобы длина дуги была пропорциональна продолжительности операции, а графическое изображение дуг не обязательно должно представлять прямолинейный отрезок.

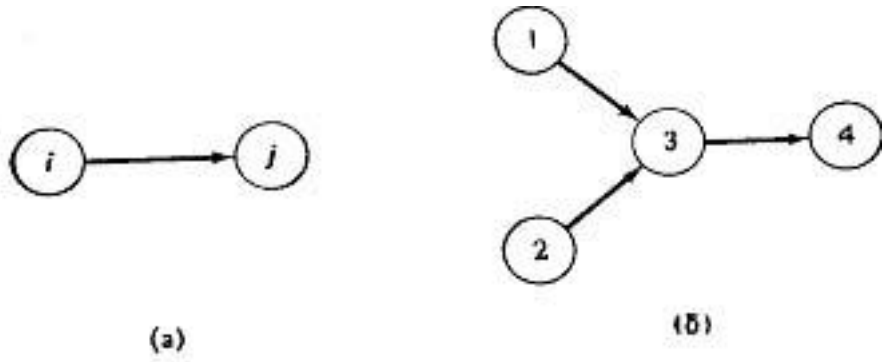


Рис.6. Представление операций

На рис.6(*а)* приведен типичный пример графического изображения операции i, j с начальным событием i и конечным событием j. На рис.6(*б*) показан другой пример, из которого видно, что для возможности начала операции (3, 4) требуется завершение операций (1, 3) и (2,3). Протекание операций во времени задается путем нумерации событий, причем номер начального события всегда меньше номера конечного. Такой способ нумерации особенно удобен при выполнении вычислений на ЭВМ

5.2.2. Правила построения сетевой модели

Правило 1. *Каждая операция в сети представляется одной , только одной дугой (стрелкой).* Ни одна из операций не должна появляться в модели дважды. При этом следует различать случай, когда какая-либо операция разбивается на части; тогда каждая часть изображается отдельной дугой. Так, например, прокладку трубопровода можно расчленить на прокладку отдельных секций и рассматривать прокладку каждой секции как самостоятельную операцию.

Правило 2. *Ни одна пара операций не должна определяться одинаковыми начальным и конечным событиями.* Возможность неоднозначного определения операций через события появляется в случае, когда две или большее число операций допустимо выполнять, одновременно. Чтобы исключить такую “ошибку” между А и конечным (начальным) событием или между *В* и конечным (начальным)

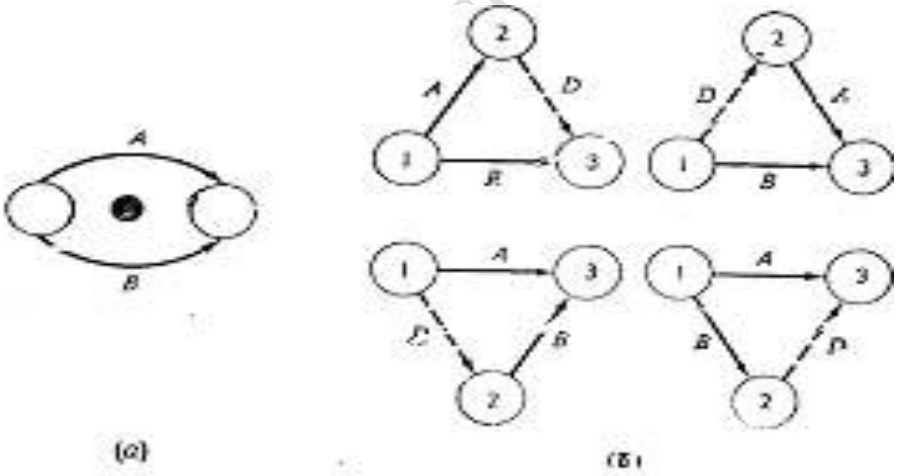


Рис.7. Введение фиктивной операции

событием вводится фиктивная операция. Рис.7(*б)* иллюстрирует различные варианты введения такой фиктивной операции *D.* В результате операции ***А*** и ***В*** определяются теперь однозначно парой событий, отличающихся либо номером начального, либо номером конечного события. Следует обратить внимание на то, что фиктивные операции не требуют затрат ни времени, ни ресурсов.

Правило 3. *При включении каждой операции в сетевую модель для обеспечения правильного упорядочения необходимо дать ответы на следующие вопросы.*

а) *Какие операции необходимо завершить непосредственно перед началом рассматриваемой*

операции?

б) *Какие операции должны непосредственно следовать после завершения данной операции?*

в) *Какие операции могут выполняться одновременно с рассматриваемой?*

Это правило не требует пояснений. Оно позволяет проверять (перепроверять) отношения упорядочения в процессе построения сети.

5.2.3. Расчет сетевой модели

Применение методов СПУ в конечном счете должно обеспечить получение календарного плана, определяющего сроки начала и окончания каждой операции. Построение сети является лишь первым шагом на пути к достижению этой цели. Вследствие наличия взаимосвязей между различными операциями для определения сроков их начала и окончания необходимо проведение специальных расчетов. Эти расчеты можно выполнять непосредственно на сети, пользуясь простыми правилами. В результате вычислений определяются *критические* и *некритические* операции программы.

Операция считается **критической**, если задержка ее начала приводит к увеличению срока окончания всей программы. **Некритическая** операция отличается тем, что промежуток времени между ее *ранним началом* и *поздним окончанием* (в рамках рассматриваемой программы) больше ее фактической продолжительности. В таком случае говорят, что некритическая операция имеет резерв, или запас, времени.

5.2.3.1. Определение критического пути

Критический путь определяет непрерывную последовательность критических операций, связывающих исходное и завершающее события сети. Другими словами, критический путь задает все критические операции программы. Метод определения такого пути иллюстрируется на численном примере.

Пример. Рассмотрим сетевую модель, показанную на рис.8, с исходным событием 0 и завершающим событием 6.

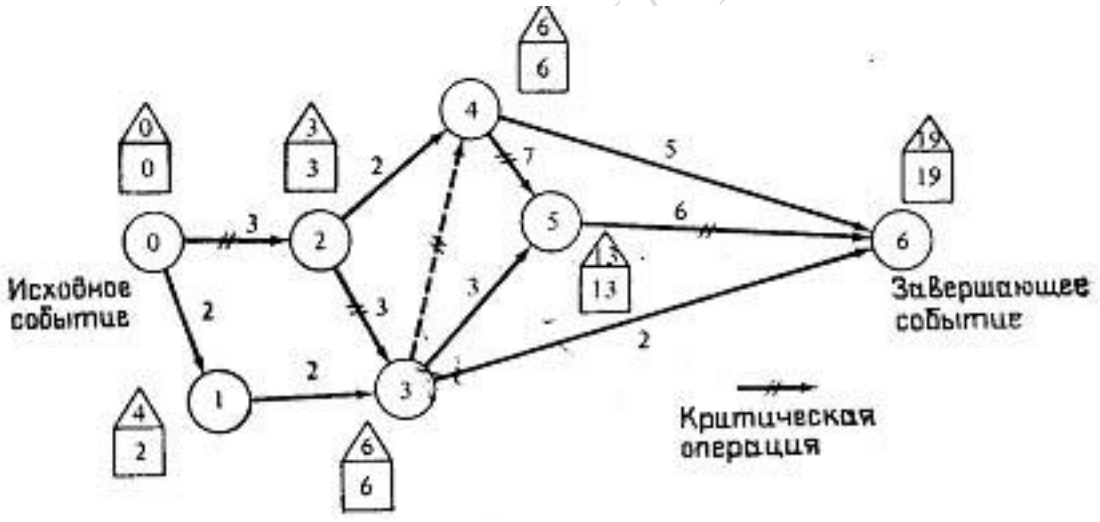


Рис.8. Сетевая модель календарного плана.

Оценки времени, необходимого для выполнения каждой операции, даны у стрелок.

Расчет критического пути включает два этапа. Первый этап называется *прямым проходом***.** Вычисления начинаются с исходного события и продолжаются до тех пор, пока не будет достигнуто завершающее событие всей сети. Для каждого события вычисляется одно число, представляющее ранний срок его наступления. Эти числа указаны на рис.8 в квадратах. На втором этапе, называемом *обратным проходом***,** вычисления начинаются с завершающего события сети и продолжаются, пока не будет достигнуто исходное событие. Для каждого события вычисляется число, представляющее поздний срок его наступления. Эти числа даны в треугольниках.

Рассмотрим теперь **прямой проход**.

Пусть $\mathbf{tрн_i}$ — *ранний срок начала* всех операций, выходящих из события i. Таким образом, $\mathbf{tрн_i}$, является также ранним сроком наступления события i. Если принять i=0, т. е. считать, что номер исходного события сети равен нулю, то при расчете сети $\mathbf{tрн_o}$=0. Обозначим символом $\boldsymbol{\tau_{ij}}$ продолжительность операции (i, j). Тогда вычисления при прямом проходе выполняются по формуле

$\mathbf{tрн_j=max\{tрн_i+\tau_{ij}\}}$ для всех операций (i,j),

где $tрн_o$==0. Следовательно, чтобы вычислить **tрнj** для события j, нужно сначала определить **tрнi** начальных событий *всех* операций (i, j), входящих в событие j.

Применительно к рис.8 вычисления при прямом проходе начинаются с $tрн_o$=0, как показано в квадрате над событием 0. Поскольку в событие 1 входит только одна операция (0, 1) продолжительностью τ_{o1}=2,

$$tрн_i=tрн_o+\tau_{o1}==0+2=2.$$

Этот результат записан в квадрате у события 1. Рассмотрим далее событие 2. [Заметим, что событие 3 пока рассматривать нельзя, так как срок $tрн_2$ (событие 2) еще неизвестен.] Таким образом,

$tрн_2=tрн_0+\tau_{02}=0+3=3$.

Поместим этот результат в квадрат у события 2. Перейдем теперь к событию 3. Поскольку в него входят две операции (1, 3) и (2, 3),

$tрн_3 = max\ \{ tрн_i + \tau_{i3}\} = max\ \{2 + 2,3 + 3\} = 6,\ i=1,2$

Этот результат также записан в квадрате у события 3.

Вычисления продолжаются аналогичным образом, пока не будут определены значения $tрн_j$, для всех j. Имеем

$tрн_4 = max\ \{tрн_i + \tau_{i4}\} = max\ \{3 + 2,6 + 0\} == 6,\ i=2, 3$

$tрн_5 = max\ \{tрн_I + \tau_{i5}\} = max\ \{6 + 3,6 + 7\} == 13,\ i=3,4$

$tрн_6 = max\ \{tрн_i + \tau_{i6}\} = max\ \{б + 2,6 + 5,13 + 6\} = 19,\ i=3, 4,6$

На этом вычисления прямого прохода заканчиваются.

Обратный проход начинается с завершающего события сети. При этом целью является определение $tпо_i$; — *поздних сроков окончания* всех операций, входящих в событие i. Если принять i=n, где n — завершающее событие сети, то $tпо_n=tрн_n$ является отправной точкой обратного прохода. В общем виде для любого события i

$tпо_i=min\{tпо_j—\tau_{ij}\}$ для всех операций (i, j).

Значения tпо (указанные в треугольниках) вычисляются следующим образом:

$tпо_6=tрн_6==19$,

$tпо_5=tпо_6 –\tau_{56} = 19-3=13$,

$tпо_4=min\ \{tпо_j—\tau_{4j}\}=min=\{13—7, 19—5\}=6,\ i = 5,6$

$tпо_3=min\ \{tпо_j—\tau_{3j}\}=min\{6—0, 13—3, 19—2\} =6,\ i=4, 5, 6$

$tпо_2=min\ \{tпо_j—\tau_{2j}\}=min\{6—3, 6—2\}=3,\ i=3,4$

$tпо_1 = tпо_3—\tau_{13} == 6 — 2 = 4$,

$tпо_0=min\ \{tпо_j—\tau_{0j}\}==min\{4—2, 3—3\}==0,\ i=1,2$

Вычисления при обратном проходе закончены.

Используя результаты вычислений при прямом и обратном проходах, определяем операции критического пути. Операция (i,j) принадлежит критическому пути**,** если она удовлетворяет следующим трем условиям:

$$\mathbf{tрн_i=tпо_i} \qquad (1)$$

$$\mathbf{tрн_j=tпо_j} \qquad \textit{(2)}$$

$$\mathbf{tрн_j—tрн_i=tпо_j—tпо_i=\tau_{ij}} \qquad (3)$$

По существу, эти условия означают, что между ранним сроком начала (окончания) и поздним сроком начала (окончания) критической операции запас времени отсутствует. В сетевой модели это отражается в том, что для критических операций числа, проставленные в квадратах и треугольниках у начальных и конечных событий, совпадают, а разность между числом в квадрате (или треугольнике) у конечного события и числом у начального события равна продолжительности соответствующей операции в квадрате (или треугольнике).

На рис.8 критический путь включает операции (0, 2), (2, 3), (3, 4), (4, 5) и (5, 6). Критический путь определяет кратчайшую возможную продолжительность всей программы в целом. Заметим, что операции (2, 4), (3, 5), (3, 6) и (4, 6) удовлетворяют условиям (1) и (2), но не условию (3). Поэтому они не являются критическими.

Критический путь представляет собой непрерывную цепочку операций, соединяющую исходные события сети с завершающим.

5.2.3.2. Определение резервов времени

При определении критического пути необходимо вычислить резервы времени для некритических операций. Очевидно, что резерв времени критической операции должен быть равен нулю. Поэтому она и называется критической.

Прежде чем приступить к вычислению резервов времени, нужно ввести определения еще двух сроков, связанных с каждой операцией. Это срок **позднего начала** (**tпн**) и срок **раннего окончания** (**tро**), которые для любой операции (i, j) задаются соотношениями

$$t_{пн_{ij}} = t_{по_j} - \tau_{ij},$$

$$t_{ро_{ij}} = t_{рн_i} + \tau_{ij}.$$

Различают два основных вида резервов времени: **полный резерв** (**r**) и **свободный резерв** (**ρ**). Полный резерв времени операции (i,j) представляет собой разность между максимальным отрезком времени, в течение которого может быть выполнена операция ($t_{по_j} - t_{рн_i}$), и ее продолжительностью τ_{ij}, т. е.

$$r_{ij} = t_{по_j} - t_{рн_j} - \tau_{ij} = t_{по_j} - t_{ро_{ij}} = t_{пн_{ij}} - t_{рн_i}.$$

Свободный резерв времени определяется в предположении, что все операции в сети начинаются в ранние сроки. При этом условии величина ρ_{ij} для операции {i, j} представляет собой превышение допустимого отрезка времени ($t_{рн_j} - t_{рн_i}$) над продолжительностью операции (τ_{ij}}, т. е.

$$\rho_{ij} = t_{рн_j} + t_{рн_i} - \tau_{ij}$$

Результаты расчета критического пути и резервов времени некритических операций можно свести в удобную для пользования таблицу. В столбцах (1),(2),(3) и (6) приведены результаты расчета сети, рассмотренной в примере 1. Остальные данные легко вычислить по приведенным выше формулам.

В таблице 14 приведены результаты типичного расчета сетевой модели. Она содержит всю необходимую для построения календарного плана (графика) информацию. Заметим, что только критические операции должны иметь нулевой полный резерв времени.

Таблица 14. Резервы времени.

Операция (i, j)	Продолжительность	Раннее		Позднее		Полный резерв	Свободный резерв
		начало	окончание	начало	окончание		
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
(0, 1)	2	0	2	2	4	2	0
(0, 2)	3	0	3	0	3	0 [a]	0
(1, 3)	2	2	4	4	6	2	2
(2, 3)	3	3	6	3	6	0 [a]	0
(2, 4)	2	3	5	4	6	1	I
(3, 4)	0	6	6	6	6	0 [a]	0
(3, 5)	3	6	9	10	13	4	4
(3, 6)	2	6	8	17	19	11	11
(4, 5)	7	6	13	6	13	0 [a]	0
(4, 6)	5	6	11	14	19	8	8
(5, 6)	6	13	19	13	19	0 [a]	0

[a] Критическая операция.

В таблице приведены результаты типичного расчета сетевой модели. Она содержит всю необходимую для построения календарного плана (графика) информацию. Заметим, что только критические операции должны иметь нулевой *полный* резерв времени. Когда полный резерв равен нулю, свободный резерв также должен быть равен нулю. Однако обратное неверно, поскольку свободный, резерв некритической операции также может быть нулевым. Так, например, в таблице свободный резерв времени некритической операции (0, 1) равен нулю.

5.3. Оптимизация плана по условию равномерной загрузки

Конечным результатом выполняемых на сетевой модели расчетов является календарный график (план). Этот график легко преобразуется в реальную шкалу времени, удобную для реализации процесса выполнения программы.

При построении календарного графика необходимо учитывать наличие ресурсов, так как одновременное (параллельное) выполнение некоторых операций из-за ограничений, связанных с рабочей силой, оборудованием и другими видами ресурсов, может оказаться невозможным. Именно в этом отношении представляют ценность полные резервы времени некритических операций. Сдвигая некритическую операцию в том или ином направлении, но в пределах ее полного резерва времени, можно добиться снижения максимальной потребности в ресурсах. Однако даже при отсутствии ограничений на ресурсы полные резервы времени обычно используются для выравнивания потребностей в ресурсах на протяжении всего срока реализации программы. По существу, это означает, что программу удается выполнить более или менее постоянным составом рабочей силы по сравнению со случаем, когда потребности в рабочей силе (и других ресурсах) резко меняются при переходе от одного интервала времени к другому.

Покажем, как можно выровнять потребности в ресурсах на примере сети, представленной на рис.8.

Данные, необходимые для построения календарного графика, приведены в итоговой таблице 14 результатов расчетов календарного плана.

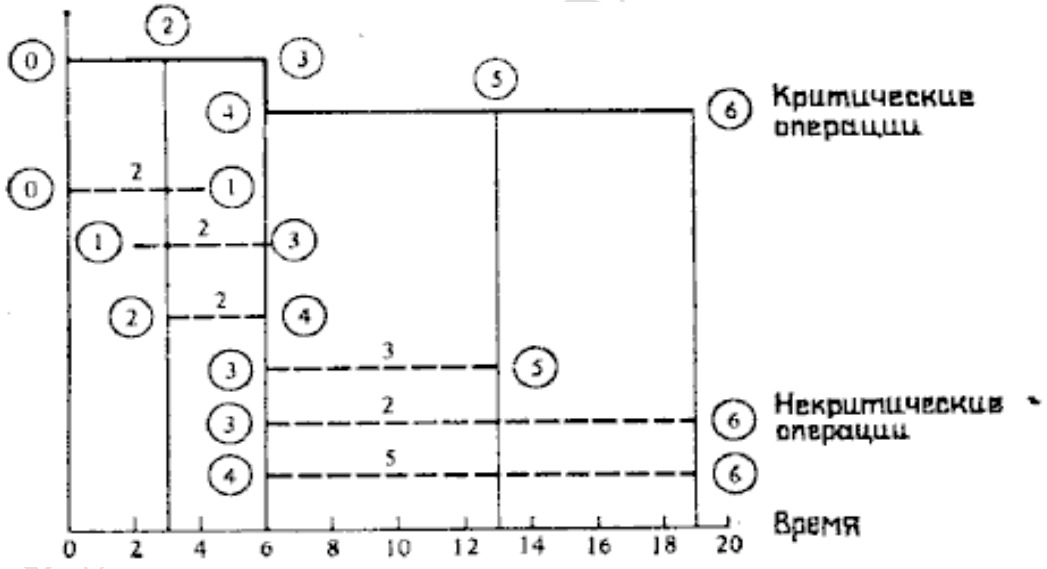


Рис. 9. Гант - карта календарного плана.

Прежде всего, определяются календарные сроки выполнения критических операций. Далее рассматриваются некритические операции и указываются их ранние сроки начала tpн и поздние сроки окончания tпо. Критические операции изображаются сплошными линиями.

Отрезки времени, в пределах которых могут выполняться некритические операции, наносятся пунктирными линиями, показывающими, что календарные сроки этих операций можно выбрать в указанных пределах при условии сохранения отношений следования.

На рис. 9 показан календарный график, соответствующий рассматриваемй сетевой модели. Фиктивная операция (3, 4) не требует затрат времени и поэтому показана на графике вертикальным отрезком. Числа, проставленные над некритическими операциями, соответствуют их продолжительностям.

Роль *полных* и *свободных* резервов времени при выборе календарных сроков выполнения некритических операций объясняется двумя общими правилами.

1. Если полный резерв *равен* свободному, то календарные сроки некритической операции можно выбрать в любой точке между ее ранним началом и поздним окончанием (пунктирные отрезки на рис. 9).

2. Если свободный резерв *меньше* полного, то срок начала некритической операции можно сдвинуть по отношению к ее раннему сроку начала не более чем на величину свободного резерва, не влияя при этом на выбор календарных сроков *непосредственно* следующих операций.

В рассматриваемом примере правило 2 применимо только к операции (0, 1), а календарные сроки всех остальных операций выбираются по правилу 1. Это объясняется тем, что у операции (0, 1) свободный резерв времени равен *нулю*. Таким образом, если срок начала операции (0, 1) совпадает с ее ранним сроком (t=0), то календарные сроки непосредственно следующей операции (1, 3) можно выбрать любыми между ранним началом (t=2) и поздним окончанием (t=6) этой операции. Если же срок начала операции (0, 1) сдвинут по отношению к t=0, то раннее начало операции (1, 3) должно быть сдвинуто, по крайней мере, на ту же величину. Так, например, в случае, когда операция (0, 1) начинается в момент t=1, она закончится в момент t=3, а календарные сроки операции (3, 1) можно выбрать так, чтобы она начиналась в любой момент между t=3 и t=6. Это ограничение не относится к остальным некритическим операциям, так как их полный и свободный резервы времени совпадают. Этот вывод легко сделать из рассмотрения рис. 5.5, так как операции (0, 1) и (1, 2) единственные, допустимые интервалы времени которых накладываются друг на друга.

Таким образом, если свободный резерв времени операции меньше полного, то это служит признаком того, что окончательные календарные сроки такой операции нельзя фиксировать, не проверив сначала, как это повлияет на сроки начала непосредственно следующих операций. Столь ценную информацию можно получить только на основе расчетов сетевой модели.

Предположим, что для выполнения различных операций требуются указанные ниже в таблице 15 ресурсы рабочей силы.

Таблица 15. Требуемые ресурсы исполнителей

Операция	К-во испонителей	Операция	К-во испонителей
(0,1)	0	(3,5)	2
(0,2)	5	(3,6)	1
(1,3)	0	(4,5)	2
(2,3)	7	(4,6)	5
(2,4)	3	(5,6)	6

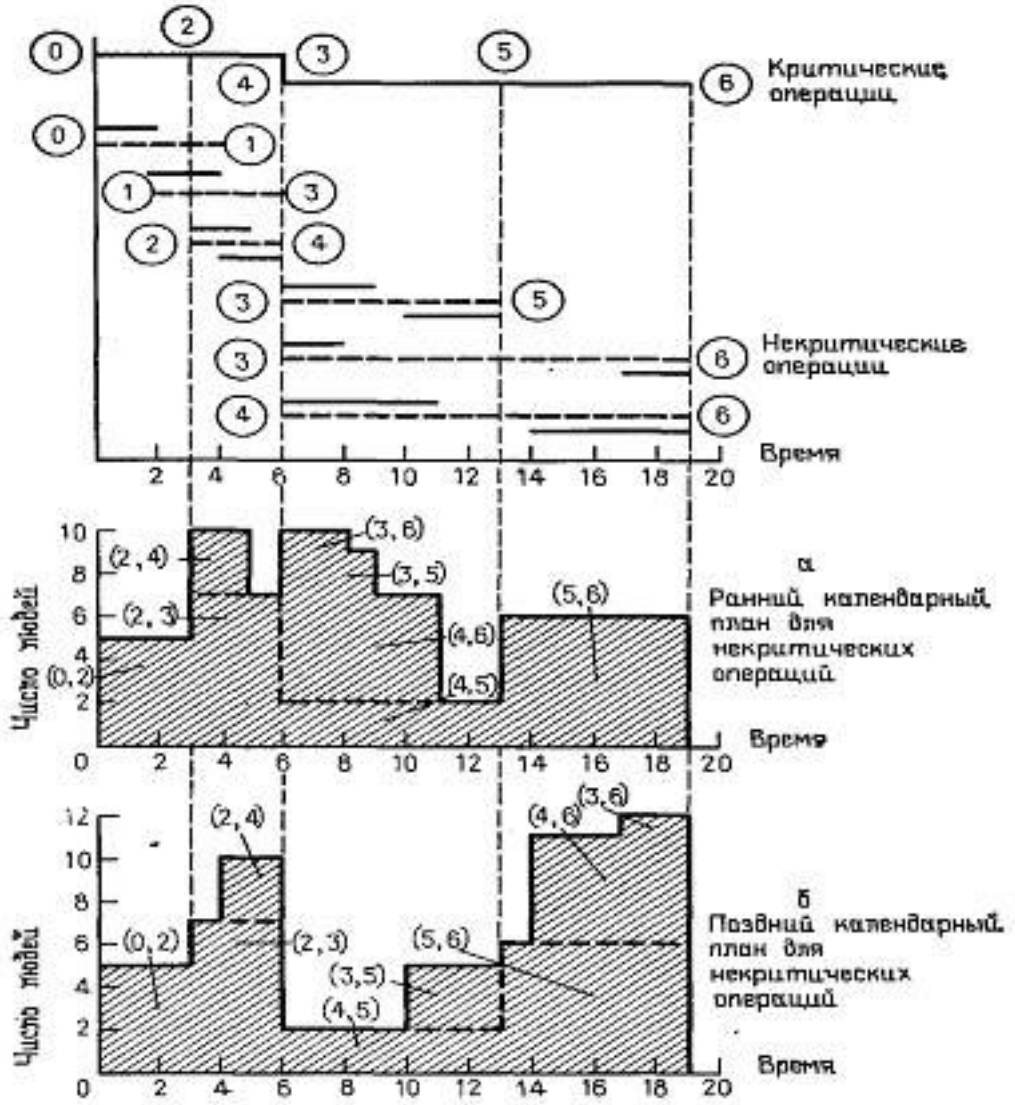


Рис. 10; а – ранний календарный план для некритических операций; б – поздний календарный план для некритических операций.

Чтобы оценить распределение работ найдём среднюю оценку потребности

$\mathbf{n_{cp}} = \sum \boldsymbol{\tau_i} * \mathbf{n_i} / \mathbf{T_0} \approx 7$ человек.

Задача сводится к определению плана, доставляющего минимум отклонения от среднего

$\mathbf{min\ z = min\ max_t \{ | n_{cp} - n_t | \}}$,

т.е. требуется построить такой календарный план (график) реализации программы, при котором потребности в рабочей силе будут наиболее равномерными на протяжении всего срока осуществления программы.

Заметим, что для выполнения операций (0, 1) и (1,3) рабочая сила не требуется, на что указывает, нулевое количество человек для каждой из этих операций. Вследствие этого календарные сроки операций (0,1) и и (1,3) можно выбирать независимо от процедуры выравнивания потребностей в ресурсах.)

На рис. 5.6 а, показана потребность в рабочей силе при условии выбора в качестве календарных сроков некритических операций начала их ранних сроков (так называемый ранний, или левый,

календарный план), а на рис. 5.6 б — потребность в рабочей силе при выборе наиболее поздних сроков (так называемый поздний, или правый, календарный план). Пунктирной линией представлена потребность критических операций, которая должна быть обязательно удовлетворена, если нужно выполнить программу в минимально возможный срок. (Отметим, что для операций (0, 1) и (1, 3) ресурсы рабочей силы не требуются.)

Как показывают потребности в ресурсах критической операции (2, 3), для реализации программы необходимо, по крайней мере, 7 человек. При раннем календарном плане некритических операций максимальная потребность в ресурсах составляет 10 человек, а при позднем — 12. Этот пример наглядно показывает, что максимальные потребности в ресурсах зависят от использования резервов времени некритических операций. Однако, как видно из рис. 5.6, независимо от распределения этих резервов максимальная потребность в рабочей силе для рассматриваемой программы не может быть меньше 10 человек, так как интервал времени, в пределах которого можно выполнять операцию (2, 4), совпадает с интервалом критической операции (2, 3). График потребности в рабочей силе при раннем календарном плане можно улучшить, выбрав поздние календарные сроки для операции (3, 5) и назначив выполнение операции (3, 6) непосредственно после завершения операции (4, 6).

Новый график потребности в рабочей силе, приведенный на рис. 11, обеспечивает более равномерное распределение ресурсов с максимальным отклонением от среднего на 3 человека.

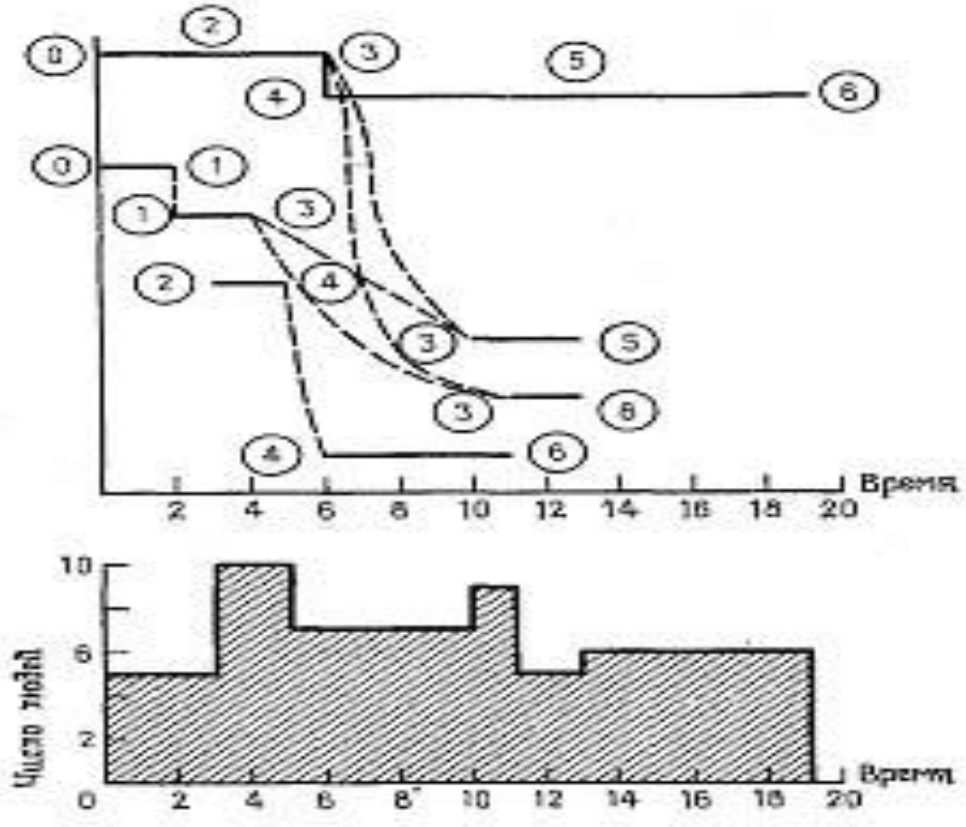


Рис. 11. Оптимальное распределение ресурсов.

При реализации некоторых программ может ставиться цель не просто обеспечения равномерного использования ресурсов, а ограничения максимальной потребности в них определенным пределом. Если этой цели не удается достичь путем перепланирования календарных сроков некритических операций, то, чтобы снизить потребность в ресурсах, приходится увеличивать продолжительность некоторых критических операций.

Из-за математических трудностей пока что не разработан метод, обеспечивающий *оптимальное* решение задачи равномерного использования ресурсов, т. е. задачи минимизации максимальной потребности в ресурсах в любой момент процесса выполнения программы. Поэтому приходится

пользоваться эвристическими алгоритмами, подобными приведенному выше. Все эти алгоритмы построены на правилах использования резервов времени некритических операций.

5.4. Оптимизация плана с учетом стоимости.

5.4.1. Стоимостные факторы, учитываемые при календарном планировании программ

Стоимостный аспект вводится в схему календарного планирования программ путем определения зависимости “затраты (стоимость) — продолжительность” для каждой операции программы. При этом рассматриваются только элементы так называемых прямых затрат, а косвенные затраты типа административно-управленческих расходов не принимаются во внимание. Однако их влияние учитывается при выборе окончательного календарного плана программы.

Рис.12. Линейная аппроксимация стоимости операции.

На рис.12 показана типичная линейная зависимость стоимости операции от ее продолжительности, используемая для большинства программ. Точка (tn, Cn), где tn — продолжительность операции, а Cn — ее стоимость, соответствует так называемому *нормальному режиму* выполнения операции. Продолжительность операции t_n можно уменьшить (“сжать”), увеличив интенсивность использования ресурсов (т. е. количество ресурсов, затрачиваемых на выполнение операции в единицу времени), а следовательно, увеличив и стоимость операции. Однако существует предел, называемый *минимальной продолжительностью операции***.** За точкой, соответствующей этому пределу (точкой максимально интенсивного режима), дальнейшее увеличение интенсивности использования ресурсов ведет лишь к увеличению затрат без сокращения продолжительности операции. Этот предел обозначен на рис.12 точкой с координатами (t_c, C_c)**.**

Линейная зависимость “затраты — продолжительность” принимается прежде всего из соображений удобства, так как ее можно определить для любой операции всего по двум точкам нормального и максимально интенсивного режимов, т. е. по точкам (t_n, C_n) и (t_c,C_c)**.**

Использование нелинейной зависимости существенно усложняет вычисления. Однако иногда нелинейную зависимость можно аппроксимировать *кусочно-линейной***.** При таких условиях операция разбивается на части, каждая из которых соответствует одному линейному отрезку. Отметим, что наклоны этих отрезков при переходе от точки нормального режима к точке максимально интенсивного режима возрастают. Если это условие не выполняется, то аппроксимация не имеет смысла.

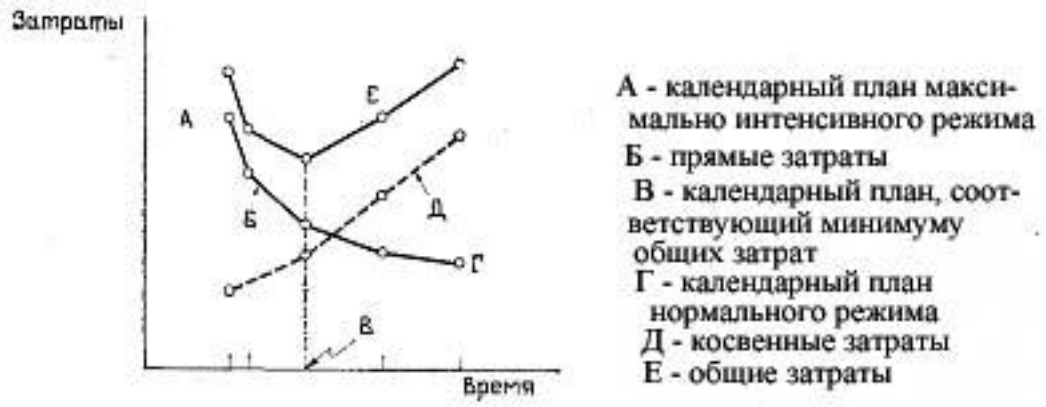
Определив зависимость “затраты — продолжительность”, для всех операций программы принимают нормальную продолжительность. Далее производится полный расчет сети и фиксируется сумма прямых затрат на программу при этой продолжительности операций. На следующем шаге рассматриваются возможности сокращения продолжительности программы. Поскольку этого можно достичь за счет уменьшения продолжительности какой-либо критической операции, только такие операции и подвергаются анализу. Чтобы добиться сокращения продолжительности выполнения программы при минимально возможных затратах, необходимо в максимально допустимой степени сжать ту критическую операцию, у которой наклон кривой “затраты — продолжительность” наименьший.

Отрезок, на который можно “сжать” продолжительность операции, ограничен точкой максимально интенсивного режима. Однако, чтобы точно определить, насколько следует сжимать продол-

жительность выбранной таким образом критической операции, нужно учесть и другие ограничения, которые подробно рассматриваются в приведенном ниже примере.

В результате "сжатия" критической операции получают новый календарный план, возможно, с новым критическим путем. Стоимость программы при новом календарном плане должна быть обязательно выше стоимости при непосредственно предшествующем календарном плане.

Рис 13. Графики затраты - продолжительность.



Далее этот новый план вновь подвергается сжатию за счет следующей критической операции с минимальным наклоном кривой "затраты — продолжительность", при условии что продолжительность этой операции не достигла минимального значения. Описанная процедура повторяется, пока все *критические* операции не будут выведены в режим максимальной интенсивности, т. е. не окажутся сжатыми до минимума". В результате расчетов получаются кривые "затраты — продолжительность". Для всех допустимых календарных планов программ и оцениваются затраты, соответствующие каждому из этих планов. Типичная кривая такого рода показана на рис.13 нижней сплошной линией. Как уже отмечалось ранее, эта кривая определяет только прямые затраты.

Естественно считать, что с увеличением продолжительности выполнения программы *косвенные* затраты должны возрастать, как показано на рис.6 штриховой кривой. Сумма прямых и косвенных затрат определяет общие затраты на программу (верхняя сплошная кривая на рис.13). Оптимальный календарный план соответствует минимуму общих затрат.

5.4.3. Пример оптимизации календарого плана по стоимости

Рассмотрим сетевую модель, приведенную на рис.14.

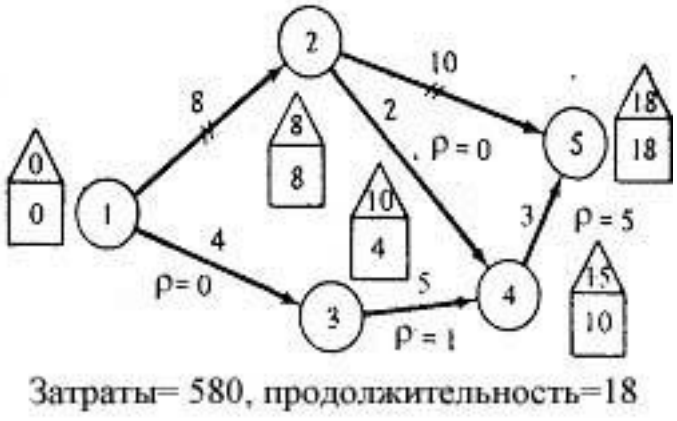


Рис.14. Сетевая модель в нормальном режиме затрат.

Таблица 16 . Затраты на операции.

Операция (*i*, *j*)	Нормальный режим		Максимально интенсивный режим	
	продолжительность	затраты	продолжительность	затраты
(1, 2)	8	100	6	200
(1, 3)	4	150	2	350
(2, 4)	2	50	1	90
(2, 5)	10	100	5	400
(3, 4)	5	100	1	200
(4, 5)	3	80	1	100

В таблице 16 указаны продолжительность и затраты на каждую операцию, соответствующие нормальному и максимально интенсивному режимам ее выполнения. Требуется определить календарные планы минимальной стоимости, которые можно реализовать в интервале между точками нормального и максимально интенсивного режимов.

Решение рассматриваемой задачи основано главным образом на учете наклона кривых “затраты — продолжительность” для различных операций.

Таблица 17. Аппроксимация функций затрат.

Операция	Наклон прямой	Операция	Наклон прямой
(1,2)	50	(2,5)	60
(1,3)	100	(3,4)	25
(2,4)	40	(4,5)	10

Наклон аппроксимирующей прямой вычисляется по формуле:

наклон = $(C_c — C_n)/(t_n — t_c)$. Вычисленные наклоны прямых для операций сети приведены в таблице.

На первом шаге вычислений предполагается, что все операции программы имеют нормальную продолжительность. На рис.14 приведены результаты расчета сети при этих условиях. Критический путь состоит из операций (1,2) и (2,5). Продолжительность выполнения программы равна 18 единицам времени, а соответствующие затраты (нормальные) составляют 580.

Второй шаг состоит в сокращении продолжительности программы за счет “сжатия” (максимально возможного) критической операции с минимальным наклоном кривой “затраты — продолжительность”. В сети рис.7 всего две критические операции (1, 2) и (2, 5), Поскольку у операции (1, 2) наклон этой кривой меньше, она и выбирается для сжатия. В соответствии с кривой “затраты — продолжительность” эту операцию можно сжать на две единицы времени, т. е. до предела, определяемого точкой максимально интенсивного режима, которая в дальнейшем называется *пределом интенсивности*. Однако уменьшение продолжительности критической операции до этого предела не обязательно приводит к сокращению продолжительности всей программы на ту же величину. Это объясняется тем, что при сжатии критической операции может возникнуть *новый* критический путь. В таком случае необходимо перейти к рассмотрению операций нового критического пути.

Один из способов оценки появления нового критического пути до момента достижения предела интенсивности заключается в анализе свободных резервов времени некритических операций. По определению свободный резерв времени некоторой операции не зависит от сроков начала других некритических операций. Таким образом, если при сжатии какой-либо критической операции *положительный* свободный резерв времени некоторой некритической операции становится равным нулю, то следует прекратить сжатие этой критической операции и проверить, не стала ли некритическая опе-

рация с нулевым свободным резервом времени критической. Следовательно, помимо предела интенсивности необходимо учитывать и предел свободного резерва времени.

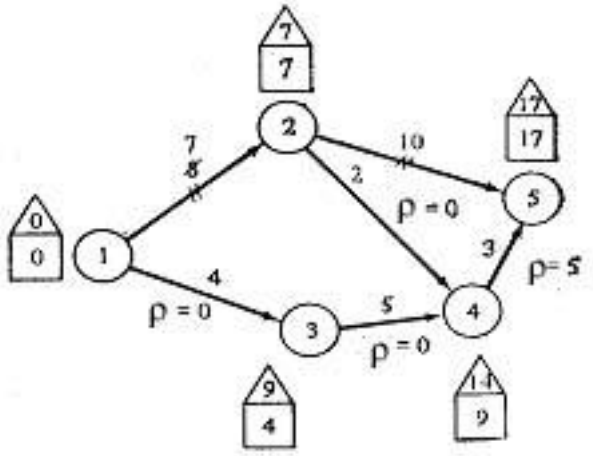
Чтобы определить предел свободного резерва времени, нужно сократить продолжительность выбранной для сжатия критической операции на *одну* единицу времени, пересчитать свободные резервы времени всех некритических операций и определить, у каких из них положительный свободный резерв уменьшился также *на одну* единицу времени. Наименьший свободный резерв времени всех таких операций (до сокращения) определяет искомый предел свободного резерва.

Применение этого правила к сети рис.14 дает свободные резервы времени (ρ), проставленные у соответствующих операций. Сокращение продолжительности операции (1,2) на одну единицу времени приводит к уменьшению свободного резерва времени операции (3, 4) на величину от единицы до нуля. Свободный резерв времени операции (4, 5) при этом не меняется, оставаясь равным 5. Таким образом, предел свободного резерва времени равен единице. Поскольку предел интенсивности для операции (1, 2) составляет 2 единицы, ее предел сжатия равен минимуму из пределов интенсивности и свободного резерва времени, т. е. min{2,1}=1. Новый календарный план показан на рис.15.

Рис.15. Сетевая модель после первого шага оптимизации.

Продолжительность программы составляет теперь 17 единиц времени, а ее стоимость равна сумме стоимости предыдущего плана и дополнительным затратам, обусловленным сокращением продолжительности операции (1, 2) на единицу времени, т. е. она составляет 580+(18—17)*50=630. Хотя свободный резерв времени определяет предел сжатия, критический путь не изменился. Следовательно, при задании предела сжатия с помощью предела свободного резерва времени не всегда справедливо утверждение о возникновении нового критического пути.

Поскольку операция (1, 2) все еще наиболее выгодна для сжатия, вычисляются соответствующие ей



Затраты =630, продолжительность = 17

пределы интенсивности и свободного резерва времени. Однако в связи с тем, что предел интенсивности операции (1, 2) равен единице, в данном случае нет необходимости определять ее предел свободного резерва времени, так как любой положительный свободный резерв не может быть меньше 1. Поэтому операция (1, 2) сжимается еще на единицу времени и тем самым достигает своего предела интенсивности.

Результаты расчетов приведены на рис.16, из которого видно, что критический путь не изменился. Продолжительность программы составляет теперь 16 единиц времени, а ее стоимость равна 630+(17—16)*50=680.

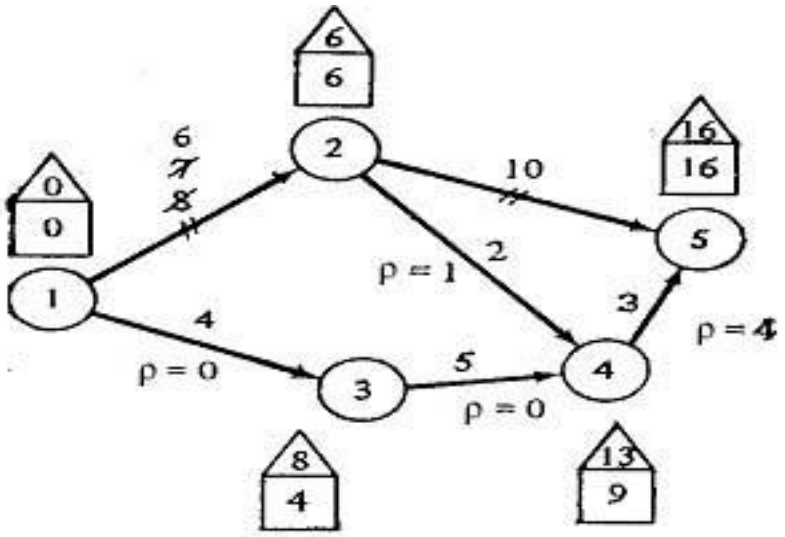


Рис.16. Сетевая модель после второго шага оптимизации.

Операцию (1, 2) теперь уже больше сжать невозможно, так как для нее достигнут максимально интенсивный режим. Поэтому для дальнейшего сокращения продолжительности программы выбирается операция (2, 5). Для этой операции имеем: предел интенсивности (10—5=5); предел свободного резерва времени 4, соответствующего операции (4, 5); предел сжатия min{5, 4} =4. Результаты выполненных вычислений приведены на рис.17.

Теперь в сети получилось два критических пути: (1, 2, 5) и (1, 3, 4, 5). Продолжительность нового календарного плана составляет 12 единиц времени, его стоимость: 680+(16—12)*60=920.

Появление двух критических путей свидетельствует о том, что для дальнейшего сокращения продолжительности программы необходимо уменьшить длину двух критических путей одновременно. Приведенное выше правило выбора критических операций для сжатия справедливо и в этом случае. В пути (1, 2, 5) операцию (2, 5) можно сжать на одну единицу времени, в пути (1, 3, 4, 5) наименьший наклон кривой “затраты — продолжительности” у операции (4, 5), а ее предел интенсивности равен двум единицам времени. Поэтому предел интенсивности двух путей равен min{l, 2} =1. Предел свободного резерва времени определяется как минимум этих пределов для каждого пути в отдельности. Однако, поскольку предел интенсивности равен единице, предел свободного резерва времени вычислять не требуется.

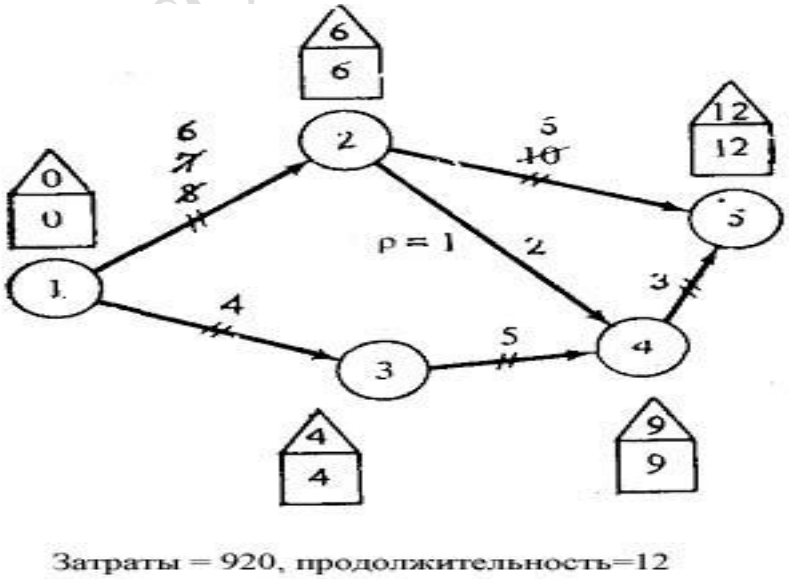


Рис.17. Сетевая модель после третьего шага оптимизации.

Новый календарный план показан на рис.18.

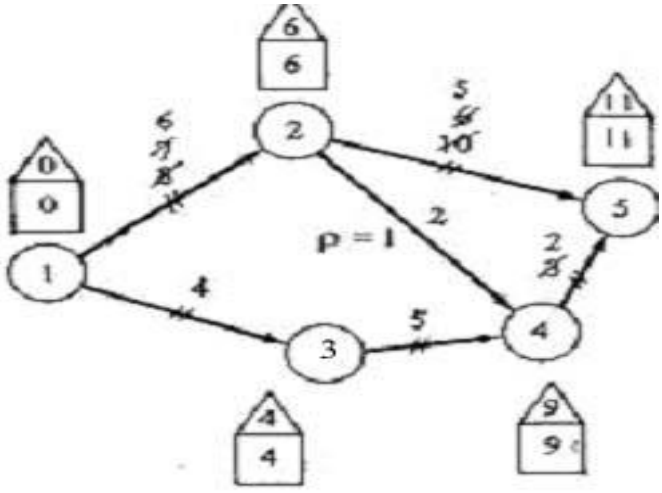


Рис 18. Сетевая модель после четвертого шага оптимизации.

Продолжительность плана составляет 11 единиц времени, стоимость: 920+(12—11)*(10+60)=990.

Два критических пути больше не меняются. Так как все операции критического пути (1, 2, 5) сжаты до предела интенсивности, дальнейшее сокращение продолжительности программы невозможно. Следовательно, календарный план является планом максимальной интенсивности.

Окончательные результаты выполненных расчетов иллюстрирует рис.19, на котором приведена кривая прямых затрат по рассмотренной программе.

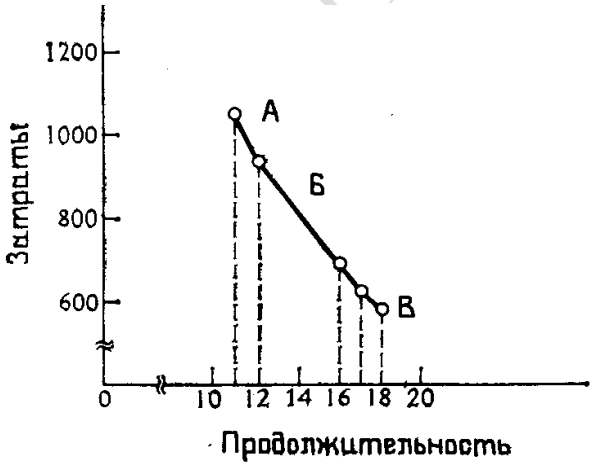


Рис. 19. График изменения прямых затрат. ***А*** — точка режима максимальной интенсивности, ***В*** - точка нормального режима.

С учетом косвенных затрат, соответствующих каждому из возможных календарных планов, можно найти план, минимизирующий общие затраты, т. е. оптимальный календарный план программы.

Например, если функция косвенных затрат имеет вид

100+60 * t ,

то последовательные оценки полученных ранее планов следующие:

1760, 1750, 1740, 1740,1750.

План минимальной стоимости представлен на рис.17 и имеет продолжительность 12 единиц.

В примере применены все правила сжатия операций в соответствующих рассмотренному случаю условиях.

Однако встречаются ситуации, когда для сокращения продолжительности выполнения программы приходится увеличивать уже сжатые операции. Рис. 5.16 соответствует одной из таких типичных ситуаций. В сети три критических пути: (1, 2, 3, 4), (1, 2, 4) и (1, 3, 4). Продолжительность операции (3, 4) была сокращена от нормальной, составляющей 8 единиц времени, до 5. Продолжительность программы можно уменьшить, одновременно сжимая одну из операций критических путей (1, 2, 4) и (1, 3, 4), либо сжимая операции (1, 2) и (3, 4) и в то же время увеличивая продолжительность операции (2, 3).

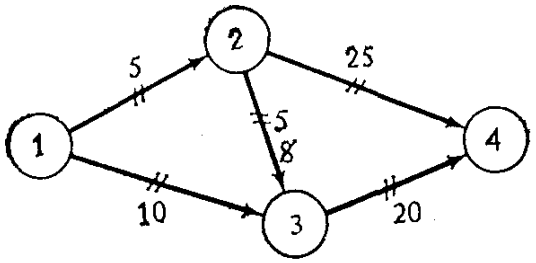


Рис.20. План, в котором все операции критические.

Выбирается вариант с наименьшей суммой наклонов. Отметим, что при сжатии операций (1, 2) и (3, 4) и увеличении продолжительности операции (2, 3) сумма наклонов представляет собой сумму наклонов для операций (1, 2) и (3, 4) за вычетом наклона для операций (2, 3). Во всех остальных случаях, когда отсутствует возможность увеличения продолжительности операций, сумма наклонов равна сумме наклонов сжатых операций.

Если требуется увеличить продолжительность, то помимо пределов интенсивности и свободного резерва времени необходимо учесть предел расширения. Этот предел равен разности между нормальной продолжительностью и продолжительностью операции, сжатой до некоторого уровня. Таким образом, предел сжатия представляет собой минимум трех величин: *предела интенсивности, предела свободного резерва времени* и *предела расширения*.

5.4.4. Упрощенный метод выявления новых критических путей

Выше для определения возможности появления новых критических путей использовался предел свободного резерва времени. Если этот предел велик и равен пределу сжатия, то продолжительность программы можно сокращать большими шагами. По существу, преимущество такого метода заключается в том, что он позволяет свести к минимуму число календарных планов, рассчитываемых в интервале от точки нормального режима до точки максимально интенсивного режима. Это в свою очередь означает, что минимизируется число циклов полного расчета календарного плана программы. Однако для определения пределов свободного резерва времени необходимы дополнительные вычисления, объем которых увеличивается с ростом числа критических путей в сети программы. Таким образом, применение метода определения предела свободного резерва времени не гарантирует минимизации объема вычислений.

Разработан другой метод, полностью исключающий необходимость определения предела свободного резерва времени. При рассмотрении примера указывалось, что если предел интенсивности равен единице, то вычислять предел свободного резерва времени не требуется, так как любой положительный свободный резерв времени по крайней мере равен 1.

В связи с этим новый метод предусматривает сокращение продолжительности программы в точности на одну единицу времени на каждом цикле вычислений, что, как и прежде, реализуется путем сжатия критической операции с наименьшим наклоном. Эту процедуру повторяют для нового календарного плана [и нового критического пути (путей), если он появляется], пока не получают ка-

лендарный план для режима максимальной интенсивности. Отметим, что при использовании такого метода продолжительность программы сокращается на каждом цикле вычислений на одну единицу времени. Следовательно, если интервал между нормальной продолжительностью программы и продолжительностью при максимально интенсивном режиме содержит *n* единиц времени, то требуется выполнение *n* циклов вычислений.

Убедительных доказательств большей эффективности одного из описанных методов по сравнению с другим нет.

Литература

1. В.М.Бондарев, В.И.Рублинецкий, Е.Г.Качко. Основы программирования. Харьков/Ростов-на-Дону, 1997
2. Кристофидес Н. “Теория графов. Алгоритмический подход”. Москва. Мир, 1978.
3. Ахо А. , Хопкрофт Дж.. Ульман Дж. “Построение и анализ вычислительных алгоритмов” Москва. Мир, 1979.
4. Фролов А.Б. и др. “Прикладные задачи дискретной математики и сложность алгоритмов”. Москва. Издательство МЭИ, 1997.
5. Емеличев В. А. «Лекции по теории графов». Москва. Наука. 1990
6. Таха Х. Введение в исследование операций. Т. 1, М., Мир, 1985.
7. Дягтерев Ю. И. Методы оптимизации. Сов. радио, 1980.
8. Габасов Р., Кириллова Ф.М. Методы оптимизации.
9. Вентцель Е.С. Исследование операций. Задачи, принципы, методология. М., Наука, 1980.
10. Таха Х. Введение в исследование операций. Т. 1, М., Мир, 1985.