**Содержание**

**Введение** 3

**Пример** 3

**Листинг программы** 8

**Результаты работы программы** 12

**Введение**

Метод критического пути (КМП)**-**является основным математическим средством для вычисления ранних и поздних начал и окончаний работ и резервов времени.

Сетевой график – необходимый элемент сложного производства, состоящего из нескольких связанных и зависящих друг от друга этапов. Выявление критического пути и временных резервов производства – основная задача, решаемая построением сетевого графика. Такие задачи могут быть представлены в виде графа и в виде отображающей его таблицы. Для нахождения критического пути (последовательности этапов работы, определяющих длительность всего проекта и не имеющих резерва по времени) применяются вычислительные методы. Одним из таких методов является табличный метод и применяется для данных, представленных в виде таблицы.

Проблема автоматизации расчёта сетевого графика является достаточно актуальной и важной. Вычисление критического пути с помощью ЭВМ поможет в несколько раз ускорить этот процесс, а при больших графиках – во много раз. Поэтому автоматизация расчёта сетевого графика может иметь большую практическую пользу.

Алгоритм нахождения длиннейшего пути в графе похож на алгоритм нахождения кротчайшего пути в графе (алгоритм Дикстры). Вместо кротчайших ребер в алгоритме нахождения кратчайшего пути выбираются длиннейшие.

**Пример**

Требуется найти такой путь на графе, который бы имел максимальную длину по сравнению со всеми возможными путями для данного графа.

Пусть существует ориентированный граф без циклов G.

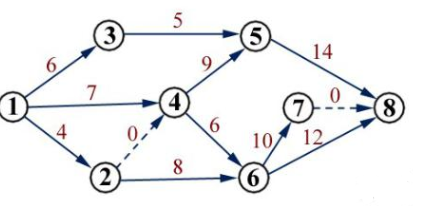
Рёбра графа — выполнение работы. Рёбра имеют длину, обозначающую продолжительность работы и направление, обозначающее последовательность выполнение работы. Вершины — это факт начала и окончания работ. (1, 3) — ребро с весом (длительностью работы) 6. 1 — начало работы.

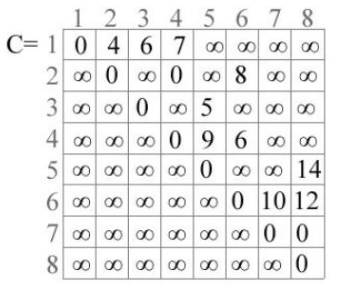
3 — конец работы. Ребро (2, 4) — фиктивная работа нулевой длительности.

t’j – наиболее ранний срок начала работы;

t’’j – наиболее ранний срок окончания работы;

tj= t’’j - t’j – максимальное время задержки между выполнениями работ;





## Найдем t’j для вершин 5 и 6. Путь вычисляем в направлении дуг графа.

Ввершину 5 можно попасть 3 путями:

m1={1, 3, 5} → l = 11 (I — общая длина);

m 2={1, 2, 4, 5} → l = 13;

m 3={1, 4, 5} → l = 16;

t’5 =maxli=max(11, 13, 16) = 16.

Ввершину 6 можно попасть 3 путями:

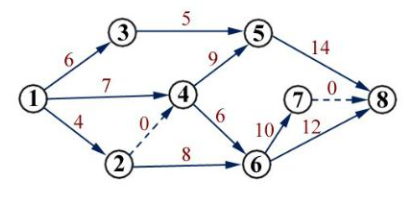
m 1={1, 4, 6} → l = 13;

m 2={1, 2, 4, 6} → l = 10;

m 3={1, 7, 6} → l = 12;

t’6=maxli=max(13, 10 ,12) = 13.

t’5 t’6 - являются оценками максимального пути для соответствующих вершин. Опираясь на уже полученные оценки, последовательно идем от вершины к вершине, выбирая максимальный путь.



Вычисляем максимальные пути для остальных вершин:

t’1=0;

t’2=4;

t’3=6;

t’4=7;

t’5 = 16;

t’6 = 13;

t’7=23;

t’8=30.

Видим, что весь объем планируемых работ выполнен не ранее чем за 30 единиц времени.

## Теперь вычислим t’’j для каждой вершины:

t’’j= min(t’’k - ljk),где t’’j — наиболее поздний срок начала выполнения работы, t’’k — наиболее поздний срок окончания работы, а

ljk — длина дуги (j, k).

Идем от начала графа к его началу, против направления дуг.

t’’8= 30;

t’’7= min(0) = 30;

t’’6=min (18, 20) = 18;

t’’5=min (30 − 14) = 16;

t’’4=min (7, 12) = 7;

t’’3= min (16 − 5) = 11;

t’’2= min (7, 10) = 7;

t’’1= min (5, 0, 3) = 0.

Находим максимальное время задержки между выполнениями работ:

τj= t’’i - t’j;

τj= t’’1- t’1 =0;

τj= t’’2 - t’2 =3;

τj= t’’3 - t’3 =5;

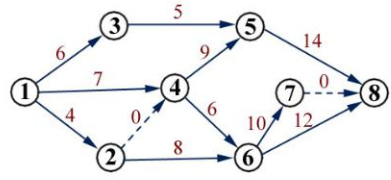
τj= t’’4 - t’4 =0;

τj= t’’5 - t’5 =0;

τj= t’’6 - t’6 =5;

τj= t’’7- t’7 =7;

τj= t’’8 - t’8 =0.



Видно, что резерв работ (1, 4), (4, 5), (5, 8) = 0. Т.е. всякая задержка или увеличение длительности работ приводит к увеличению срока завершения всех работ. Такие работы называются критическими. Эти работы — цепочки дуг соединяющие начальную и конечную вершину графа. Путь их соединяющий называется критическим путем:

m критич={1, 4, 5, 8};

lm\_крит=30.

Итак нахождение критического пути в графе это одна из основных задач сетевого планирования и управления. Нахождение критического пути в графе позволяет установить научно обоснованное выполнение работы.

**Листинг программы**

#include <iostream>

#define INF -1 // Нет вершины

#define MatrixLen 8

int matrix[MatrixLen][MatrixLen] =

{

// 1 2 3 4 5 6 7 8

{0, 4, 6, 7, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 8, 0, 0},

{0, 0, 0, 0, 5, 0, 0, 0},

{0, 0, 0, 0, 9, 6, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 14},

{0, 0, 0, 0, 0, 0, 10, 12},

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0},

};

void printSolution(int dist[], int n, int time)

{

printf("Path [ ");

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

{

if(dist[i] == 0)

printf("%d ", i + 1);

}

printf(" ]\nTime: %d\n", time);

}

//===================================

int minDistance(int dist[], bool shortest[])

{

int min = INF, min\_index;

for (int v = 0; v < MatrixLen; v++)

{

if (shortest[v] == false && dist[v] >= min)

{

min = dist[v], min\_index = v;

}

}

return min\_index;

}

void solution()

{

int dist\_max[MatrixLen],

dist\_relation[MatrixLen],

dist\_solve[MatrixLen];

bool shortestPath[MatrixLen];

for (int i = 0; i < MatrixLen; i++)

{

dist\_max[i] = INF, shortestPath[i] = false;

}

dist\_max[0] = 0;

for (int count = 0; count < MatrixLen; count++)

{

int u = minDistance(dist\_max, shortestPath);

shortestPath[u] = true;

for (int v = 0; v < MatrixLen; v++)

{

if (!shortestPath[v] && matrix[u][v]

&& dist\_max[u] + matrix[u][v] > dist\_max[v])

dist\_max[v] = dist\_max[u] + matrix[u][v];

}

}

for (int i = 0; i < MatrixLen; i++)

{

dist\_relation[i] = INF, shortestPath[i] = false;

}

dist\_relation[0] = dist\_max[MatrixLen - 1];

dist\_relation[MatrixLen - 1] = 0;

int temp;

for (int i = 0; i < MatrixLen; ++i)

{

for (int j = 0; j < i; ++j)

{

temp = matrix[i][j];

matrix[i][j] = matrix[j][i];

matrix[j][i] = temp;

}

}

for (int count = 0; count < MatrixLen; count++)

{

int u = minDistance(dist\_relation, shortestPath);

shortestPath[u] = true;

for (int v = 0; v < MatrixLen; v++)

{

if (!shortestPath[v] && matrix[u][v]

&& dist\_relation[u] + matrix[u][v] > dist\_relation[v])

dist\_relation[v] = dist\_relation[u] + matrix[u][v];

}

if (dist\_relation[count] == INF)

dist\_relation[count] = 0;

dist\_solve[count] = dist\_max[MatrixLen - 1] - dist\_relation[count];

dist\_solve[count] = dist\_max[count] - dist\_solve[count];

//printf("%d = %d - %d\n", dist\_solve[count], dist\_relation[count], dist\_max[MatrixLen - 1]);

}

printSolution(dist\_solve, MatrixLen, dist\_max[MatrixLen - 1]);

}

int main()

{

solution();

system("pause");

return 1;

}

**Результат работы программы**

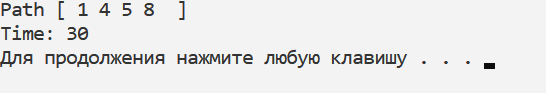
****

Рисунок 1 – результат работы программы