TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

CENTRAL CAMPUS, PULCHOWK

Project On

Critical Path Analysis (CPA)

Submitted by:

Anil Paudel (077BCT010)

Ashim B. Karki (077BCT013)

Bishal Aryal (077BCT021)

Submitted to:

Department of Electronics and Computer Engineering

17th March, 2023

ACKNOWLEDGEMENT

We express our deep gratitude to Prof. Bibha Sthapit for providing us with the basics and firm background on Data Structures and Algorithms (DSA) and the opportunity to prepare a project. The new journey of working on a project will enhance our understanding of the concepts as well as project management. We would like to express our special thanks to Ms. Shanti Tamang for proper guidance throughout the semester during labs and projects.

We pay our deep sense of gratitude to Dr. Jyoti Tandukar(HOD) and Prof. Daya Sagar Baral for providing us with facilities and knowledge for the development of the project.

We would also like to extend our sincere thanks to our friends and seniors for their valuable insights into our project idea which help to overcome the problems faced midway through the project.

ABSTRACT

This project work was given to us as a minor project for our academic session B.E. (Computer) Second Year Second Part. The main aim of this project was to develop a program implementing data structures and algorithms. For this project we made a Critical Path Method ,where a user can input multiple tasks, the time required to complete the project along with their dependencies. The program then calculates the critical path and time required to complete the tasks of the project and visualizes it in graphs as data structure.

TABLE OF CONTENTS

1. OBJECTIVES

- To understand the implementation of Data Structure and Algorithms using C++.
- To use graphical libraries to visualize the data structure.
- To understand the different ways of implementing data structures in C++ programming.
- To learn to work in a team by collaboration through version control systems and systematic breakdown of problems into simpler forms .

2. INTRODUCTION

Critical Path Analysis (CPA) is a project management tool that helps to schedule, coordinate and control activities of a project in order to ensure that the project is completed within a set timeframe. The aim of CPA is to identify the activities that are critical for the completion of the project and the amount of time each activity takes.

CPA works by analyzing the dependencies between tasks and identifying the longest path of tasks that must be completed before the project can be completed. This path is known as the critical path. The critical path determines the minimum amount of time required to complete the project. Any delay in a task on the critical path will cause a delay in the entire project.

By identifying the critical path, CPA allows project managers to identify the tasks that must be closely monitored to ensure that the project is completed on time. It also helps managers to identify non-critical tasks that can be delayed or rescheduled without affecting the completion date of the project.

CPA can be used in a wide range of industries, including construction, engineering, manufacturing, and software development. It provides a structured approach to project management and helps to ensure that projects are completed on time and within budget.

3. APPLICATION

CPA is a project management tool. It can be used to identify a critical path in a project schedule. CPA is commonly used in a variety of industries, including construction, engineering ,software development, and event planning. Overall, the Critical Path Analysis is a powerful tool that can help project managers plan, execute, and monitor complex projects in a variety of industries. Some of the uses are noted down below.

Construction projects: CPA is commonly used in the construction industry to plan and manage the construction of buildings, bridges, highways, and other infrastructure projects.

Software development projects: CPA can be used to plan and manage the development of software applications, including identifying the critical path and allocating resources to ensure that the project is completed on schedule.

Event planning: CPA can be used to plan and manage large events such as conferences, trade shows, and festivals.

Manufacturing projects: CPA can be used to plan and manage the production of goods, including identifying the critical path and ensuring that materials and resources are available when needed.

Research and development projects: CPA can be used to plan and manage the development of new products or technologies, including identifying the critical path and allocating resources to ensure that the project is completed on time.

4. LITERATURE SURVEY

Since this project is based on the implementation of Data structure and Algorithm, different books were suggested by the teachers for this concept. Books like "Data Structures using C and C++ by Yedidyah Langsam, Moshe J. Augenstein and Aaron M. Tenenbaum " were referred to as a reference for the development of a program that assisted us to clear the concepts regarding the data structures and algorithms and make our program development easier.

In order to render the graphics, we used the SFML library. The working structure as well as full functionality of classes and functions were understood from the official documentation of SFML library.

Other various concepts of Data Structures and Algorithms were also used to build the project. Concepts such as graph data structure was understood in our classroom which falls under our course of DSA. Other concepts, such as topological sorting and PERT charts were understood over the internet with the help of various YouTube videos and programming based websites.

5. EXISTING SYSTEMS

Project management system provides all the features including task management, scheduling, resource allocation, budget tracking, and progress reporting. Our system provides the task management feature which involves the Critical Path Method. We have implemented such system and also illustrated the critical path in a network diagram using C++ .

6. METHODOLOGY

A. Tools Used

1. C++ programming language: C++ is used as the main programming language to implement the algorithm for the CPM.
2. Code editor: Primarily Visual Studio and Visual Studio Code was used throughout the project.
3. SFML: SFML is a multimedia library that provides a simple interface for rendering graphics, playing audio, and handling user input. It is used to create a graphical representation of the CPM, showing the vertices and edges, as well as the early and late start and finish times.
4. Version control software: Version control software such as Git was used to manage changes to the code over time and work in a team.

B. Methodology to perform CPA

1.The vertex and edges objects were defined to represent the nodes and edges of the graph.
2.The nodes are the representation of the tasks input by the user and the nodes are linked with each other in a graphical network according to their dependencies.
3.The early start time (EST) and early finish time (EFT) during forward pass were calculated using an iterative algorithm for each vertex.
4. The late finish time (LFT) and the late start time (LST) during the backward pass were calculated using an iterative algorithm for each vertex.
5.The critical path is then calculated by selecting a path whose slack or float time is zero.
6.To draw the graph, the SFML graphics library was implemented. We used the drawing function of SFML to create circles to represent the vertices and the lines to represent the edges and labeled them accordingly.
7. Various attributes were applied to the starting vertex, final vertex and critical path using the functions from SFML library.

C. Block Diagram

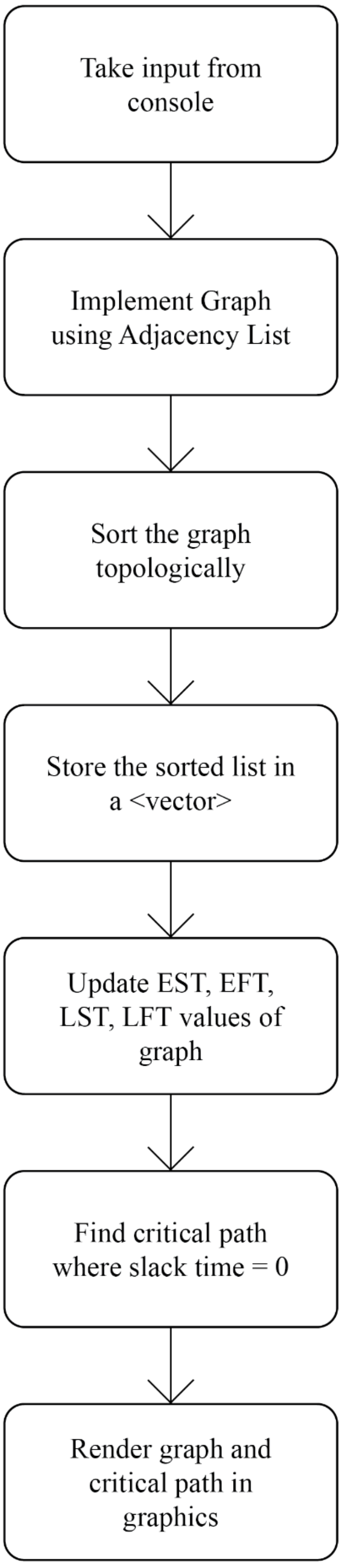
Take input from console
Implement Graph using Adjacency List
Sort the graph topologically
Store the sorted list in a <vector>
Update EST, EFT, LST, LFT values of graph
Find critical path where slack time = 0
Render graph and critical path in graphics

Fig. Block diagram

7. IMPLEMENTATION

The following libraries were included in the header:

```
#include<SFML/Graphics.hpp>
#include<string>
#include<cstdlib>
#include<queue>
#include<iostream>
#include<unordered_map>
```

Each node has the following attributes:

```
class Vertex{
public:
   std::string data;
   int time;
   int  id;
   bool cflag = false;
   int earliest_start_time;
   int early_finish_time;
   int late_start_time;
   int latest_finish_time;
   int slack;
   bool visited;

   //x and y are used to position vertex during graph rendering
   float x = 0;
   float y = 0;

};
```

To implement graphs using adjacency lists the following containers are used:

```
   std::vector<Vertex*> vertices;
   std::unordered_map<Vertex*, std::vector<Vertex*>> edges;
   void add_vertex(Vertex* vertex){
       vertices.push_back(vertex);
   }
```

```
void add_edge(Vertex* v1, Vertex* v2, char last_node){
       edges[v2].push_back(v1);
       if (last_node == 'y' || last_node == 'Y'){
           if (v1->early_finish_time >
finish_vertex->late_start_time){
               finish_vertex->late_start_time =
v1->early_finish_time;
           }
           edges[v1].push_back(finish_vertex);
       }
   }
```

The following code was used to sort the graph topologically:

```
void sort_graph_topologically(){
       std::unordered_map<Vertex *, int> in_degree;
       for (auto vertex : vertices){
           in_degree[vertex] = 0;
       }
       for (auto vertex : vertices){
           for (auto neighbor : edges[vertex]){
               ++in_degree[neighbor];
           }
       }
       std::queue<Vertex *> q;
       for (auto vertex : vertices){
           if (in_degree[vertex] == 0){
               q.push(vertex);
```

```
            }
        }
        std::vector<Vertex *> sorted;
        while (!q.empty()){
            auto current = q.front();
            q.pop();
            sorted.push_back(current);
            for (auto neighbor : edges[current]){
                --in_degree[neighbor];
                if (in_degree[neighbor] == 0){
                    q.push(neighbor);
                }
            }
        }
```

Forward pass was used to calculate early start time and early finish time:

```
if (dependancy->early_finish_time >
new_vertex->earliest_start_time){
                new_vertex->earliest_start_time =
dependancy->early_finish_time;
                new_vertex->early_finish_time =
new_vertex->earliest_start_time + time;
            }
```

The vertices <vector> was reversed to create a backward pass for late finish time and late start time evaluation:

```
vector<Vertex*> reverse_vertices = vertices;
        reverse(reverse_vertices.begin(),
reverse_vertices.end());
for (auto vertex : reverse_vertices){
            auto it = edges.find(vertex);
            if (it != edges.end()){
                for (auto neighbor : it->second){
                    if (vertex->latest_finish_time >
neighbor->late_start_time){
```

```
                        vertex->latest_finish_time =
neighbor->late_start_time;
                        vertex->late_start_time =
vertex->latest_finish_time - vertex->time;
                    }
                }
            }
        }
```

The following function was then used to calculate the Critical path. MinCompletionTime gives the minimum time that is required to complete the project:

```
void calculate_CP(){
       cout << "critical path" << endl;
       for (auto vertex : vertices){
           vertex->slack = vertex->earliest_start_time -
vertex->late_start_time;
           if (vertex->slack == 0){
               cout << vertex->data << "\t";
               vertex->cflag = true;
               MinCompletionTime += vertex->time;
           }
       }
   }
```

8. RESULTS

The program was thus created by meeting all the stated requirements in the initial proposal. The program has a terminal based input platform where users are asked to input the details which include :number of tasks, their names,durations,dependencies and so on.

Thus data inputs are taken and our program firstly implements such data in a directed graph, topologically sorts them, calculates the variables: EST,EFT,LST and LFT . Using those calculations, a valid critical path is defined and finally the path is recognized in the graph network . Our program visualizes the implemented graph network along with aforementioned variables.

9. PROBLEMS FACED AND SOLUTIONS:

One of the primary difficulties we encountered during our project was implementing an adjacency-list graph for nodes that had a user-defined data type. Our nodes contained various information, such as the node's name, flag value, and parameters needed to calculate slack variables. To overcome this challenge, we utilized classes, pointers, and various data structures such as vector and unordered_map available in the STL.

Another hurdle we faced was visualizing the graph as our project was based on C++ and we had limited graphical libraries available. Due to this constraint and limited resources to learn, a significant portion of our project schedule was allocated to graph visualization.

10. CONCLUSION

Thus upon the completion of the project, we are capable of implementing a Version control that was also used for the creation of this project. GitHub was used primarily for pushing the code into our repository from which various branches of the code were created and managed. As we worked in a team, this made it very easy for us to share, add and discuss code to our program.

11. REFERENCES:

- "Graph Data Structure And Algorithms." *GeeksforGeeks*, 15 February 2023, https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
- "Tutorials for SFML 2.5" https://www.sfml-dev.org/tutorials/2.5/..
- Langsam, Yedidyah, et al. *Data Structures using C and C++*. Pearson Education, 2006.
- "Project Scheduling - PERT/CPM | Finding Critical Path." *YouTube*, 17 July 2017, https://youtu.be/-TDh-5n90vk.