# Índice

Introdução pág.2

Descrição do Problema pág.2

Resolução do problema pág.2

Arquitectura do Problema pág.3

Divisão Lógica e Funcional dos Módulos pág.3

Estruturas de Dados pág.3

Descrição dos Algoritmos Utilizados pág.4

Descrição dos Subsistemas pág.5

Análise Crítica pág.7

Exemplos pág.8

Anexos pág.9

# Introdução

O objectivo do presente projeto é desenvolver e aplicar as metodologias e conceitos da disciplina de algoritmos e estruturas de dados, adquiridos ao longo dos laboratórios e aulas teóricas, realizando a implementação de um programa de cálculo de folgas de um circuito.

* 1. **Descrição do Problema**

Com a realização deste projeto pretende-se a implementação de um programa que calcule as folgas de todos os nós de um circuito lógico com determinadas características. O programa irá receber um ficheiro com a descrição de um circuito lógico electrónico, incluindo a sua topologia assim como a caracterização de atrasos.

O ficheiro recebido contem a informação da quantidade de nós presentes no circuito como também informações sobre o tipos de nó(entrada, saída ou intermedio),os atrasos associados ou até mesmo portas lógicas(AND, OR,NAND,NOR,XOR,NOT,BUFER).

Relativamente ao funcionamento, este deve prever três modos distintos:

1º Modo: Cálculo de folgas para todos os nós;

2º Modo: Cálculo dos instantes de chegada para portas lógicas com geração aleatória do perfil de entrada;

3º Modo: Cálculo dos instantes de chegada para portas lógicas com leitura do perfil de entrada em ficheiro.

O programa deverá devolver um ficheiro de saída, para cada um dos casos, com o mesmo nome que o ficheiro de entrada mas com a extensão alterada de “.in” para “.out”.

Este ficheiro de saída deverá apresentar para cada nó do ficheiro lido o rat(instante de chegada requerido), o at(instante de chegada real) e o sl(slack,ou seja, a folga) correspondente.

* 1. **Resolução do problema**

Para a resolução do problema da implementação do circuito lógico em que se calcula as folgas de cada nó, começamos por determinar quais as estruturas que poderiam representar internamente o circuito(com todas a informações armazenadas), tentando encontrar um equilíbrio entre a eficiência assim como a simplicidade e funcionalidade da implementação.

A escolha para a representação do circuito recaiu num vector de nós e uma matriz de adjacências, que alojam todas as informações dos nós e das arestas.

Seguidamente pensamos como iria funcionar o programa, quais as funções necessárias(para o seu funcionamento) e quais os seus parâmetros.

# Arquitetura do programa

Em primeiro lugar, é aberto e lido o ficheiro com a informação dos nós e respectivas arestas, nomeadamente, o seu número e tipo(input, output ou middle),o atraso intrínseco(dno e dei), o atraso característico(dnc e den),o tempo de chegada requerido(rat) e tempo de chegada actual (at). Esta informação é alocada em estruturas próprias, sendo que usamos um vector de nós e uma matriz de adjacências para guardar as informações do grafo. O ficheiro de entrada é fechado.

Depois, corremos o algoritmo do Dijkstra para calcular o tempo de chegada requerido e o tempo de chegada actual para todos os nós. Calculados estes valores, apenas temos de os subtrair para termos o slack(folga) para todos os nós.

O programa retorna o ficheiro saída com as folgas para todos os nós de um dado circuito.

# Estruturas de dados

Para representar o circuito utilizaram-se as seguintes estruturas:

-Estrutura com a informação dos nós

struct node{

int number;

unsigned int innode;

unsigned int outnode;

unsigned int door;

double dno;

double dnc;

double charge;

double dn;

double at;

double rat;

double distance;

double slack;

int visited;

};

Para a estrutura de nós:

* *int number* , ou seja, o número de cada nó;
* unsigned int innode, o número do nó de entrada, *unsigned int outnode*, o número do nó de saída, *unsigned int door*, a porta lógica;
* *double dno, dnc, charge, dn, at , rat, distance, slack*, que correspondem respectivamente, ao atraso intríseco, atraso caracteristico, atraso, atraso de chegada associado a um nó, instante de chegada requerido, distância e a folga associado a um nó;
* *int visited*, identifica se um nó já foi visitado ou não;

-Estrutura com a informação das arestas:

struct edge{

double de;

double dei;

double den;

};

Para a estrutura de arestas:

-*double de,dei,den,* atraso, atraso intriseco, atraso caracteristico.

Temos então, como referido anteriormente, que optámos pela utilização de um vector para guardar a informação dos nós e uma matriz de adjacências para a informação das arestas(ambas definidas em cima).

# Descrição dos Algoritmos Utilizados

Para resolver o problema no enunciado do projecto resolvemos utilizar o algoritmo do Dijkstra adptando-o á estrutura de dados utilizada.

Codificamos o Dijkstra em duas funções diferentes. Uma para calcular o atraso de chegada e o instante de chegada requerido.

A lógica por detrás do algoritmo que utilizamos é fornecer á função o ponto(nó) de partida e, sucessivamente calculamos o nó com menor distância para o at e maior no rat de modo a procedermos ao cálculo das distâncias aos seus adjacentes até todos os nós estarem verificados.

Foi adicionado código no sentido em que, caso o cálculo de uma distância influenciar os dados de nós já visitados, a função “desvisitará” os nós influenciados e calculará as distâncias novamente.

No fim dos cálculos cada algoritmo do Disjktra retorna um vector de tamanho n com as distâncias calculadas, que posteriormente serão comparados com os valores armazenados na lista de nós, substituindo a caso os parâmetros das comparações o verifiquem.

1. **Subsistemas Adicionais**

* **Libertação de Memória**

void free\_memory(char \*nameInput, char \*nameOutput, char \*mode\_s, FILE \*fpIn, FILE \*fpOut, node \*node\_list, edge \*\*edge\_matrix, int n, int \*i)

Função utilizada para libertar memória alocada.

Esta vai libertar a informação alocada no programa, nomeadamente, informação alocada na matriz de adjacencias.

* **Lê ficheiro**

void read\_file(FILE\* fpin, FILE \*fpout, int mode, char \*nameInput, char \*nameOutput, char \*mode\_s)

Função que lê o ficheiro recebido, selecciona a informação e guarda na estrutura apropriada.

Esta lê as várias linhas do ficheiro.in e retira a informação necessária para a construção do circuito e consecutivamente, calculo das folgas respondentes de cada nó.

Os três inteiros recebidos no inicio do ficheiro, vão corresponder ao número de nós total, número de nós in(entrada) e número de nós out(saida).

Continua a ler todas as linhas seguintes do ficheiro, é avaliada a letra e podemos concluir que, caso seja:

-'i', trata-se de um nó in cujo valor correspondente é x(“i nx”)

-'o', trata-se de um nó out cujo valor correspondente é y(“o ny”).

-'n', o nó x, tem determinada porta lógica e determinado valor dno, dnc e m(“n nx str dno dnc m”).

-'e', um aresta do nx para o nó ny tem um atraso intriseco,dei, e atraso caracteristico,den(“e nx ny dei den”).

-'a', para um nó x, o valor do seu instante de chegada(da informação) real(“a nx t”).

-'r',para um nó y, o valor do seu instante de chegada(da informação) requerido(“r ny rt”).

A função irá ler todas as linhas do ficheiro e alocar os valores segundo estes critérios.

* **Inserção de Informação**

insert\_edge\_info e insert\_node\_info

São duas funções que recebem respectivamente, a informação das arestas e dos nós e que guardam a informação das arestas na matriz de adjacencias e a informação do nós na lista de nós.

* **Converte porta lógica**

convert\_door

Recebe uma porta lógica e associa esta a um inteiro( de modo a simplificar a sua identificação).

* + - * ***Cálculo dos atrasos***

double reqArrivalTime(double rat\_previous, double De, double Dnode)

Função que recebe os valores o rat anterior, o de e o dnode e calcula o valor do instante de chegada requerido(rat). Rat = rat anterior – de – dnode.

double arrivalTime(double De, double Dnode, double at\_previous)

Função que recebe os valores do at anterior, o de e o dnode e calcula o valor do tempo de chegada. AT = at anterior + de + dnode

double slack(double ratNODE, double atNODE)

Função que recebe o valor do rat e do at (calculados anteriormente) e calcula o slack(folga). Slack = rat - at

double delayEdge(double Dei, double Den, double charge)

Função que recebe o valor dei, den, charge e calcula o delay(atraso) da aresta. Delayedge = dei + (charge x den)

double delayNode(double Dno, double Dnc, double fanout)

Função que recebe o valor dno, dnc, fanout e calcula o delay(atraso) do node. Delaynode = Dno + (fanout x Dnc)

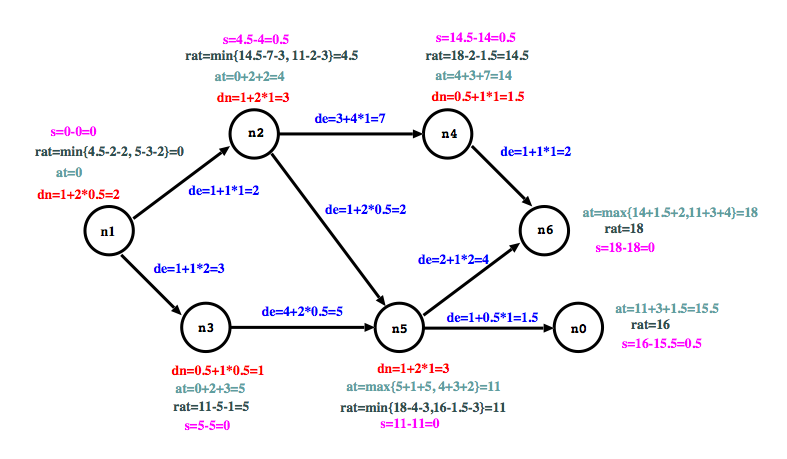
# Análise crítica

Embora tenhamos começado a fazer o modo 2 de funcionamento, não o terminamos devido a problemas na implementação do Djkstra para calcular o rat e at, podendo se ainda verificar esporadicamente alguns valores incorrectos do rat em ficheiros com grande dimensão. Contudo, caso tivéssemos mais tempo, estes erros seriam eliminados e a implementação do modo 2 terminada.

Durante a codificação do projecto surgiram várias questões, nomeadamente a estrutura mais correcta e os algoritmos para calcular o at e o rat mais indicados, de forma a não haver perdas de memória, complexidade menor e maior eficiência no programa.

Consideramos, então, que a falta de pesquisa que levou à constante alteração de estruturas do projecto poderá ter prejudicado o projecto por nos ocupar mais tempo, necessário para outras tarefas e, consequentemente, impossibilitando a conclusão do modo 2 e o total funcionamento do modo 1 (alguns valores incorrectos em ficheiros de grande dimensão).

1. **Exemplos**



|  |  |
| --- | --- |
| INPUT:  7 1 2  i n1  o n6  o n0  n n1 NOT 1 0.5 1  n n2 NOT 1 1 1  n n3 NOT 0.5 0.5 1  n n4 NOT 0.5 1 1  n n5 NAND 1 1 0.5  n n6 NOR 0 0 1  n n0 NOT 0 0 1  e n1 n2 1 1  e n1 n3 1 2  e n2 n4 3 4  e n2 n5 1 2  e n3 n5 4 2  e n4 n6 1 1  e n5 n6 2 2  e n5 n0 1 0.5  a n1 0  r n6 18  r n0 16 | OUTPUT:  n0 16.000000 15.500000 0.500000  n1 0.000000 0.000000 0.000000  n2 4.500000 4.000000 0.500000  n3 5.000000 5.000000 0.000000  n4 14.500000 14.000000 0.500000  n5 11.000000 11.000000 0.000000  n6 18.000000 18.000000 0.000000 |