Actividad Ruta Critica

Daniel F. Santos B, Brayan Leon, Andres Jaramillo
20131020016, 20141001002, 20182020145,

Octubre 2020

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Ciencias de la Computacion III

Universidad Distrital Francisco Jose de Caldas

1. Introduccion

El presente documento presenta el informe sobre la actividad de Ruta critica o el conocido metodo CPM, este metodo permite hallar la ruta cirtica para el desarrollo de un proyecto y por el cual se puede estimar cual es la tarea que es primordial, y la cual no tiene ningun tiempo de holgura.

2. Marco teorico

El método de la ruta crítica o CPM (Critical Path Method) nos informará de las actividades necesarias e indispensables para que nuestro proyecto concluya según lo planificado. Con ella, sabremos la duración total del mismo y el estado de urgencia de las actividades marcadas en un cronograma. ¿Cómo se consigue? Gracias a un algoritmo basado en la teoría de redes.

3. Requerimientos

Colas

1. Diseña un software que simule el funcionamiento de una ruta critica .
2. Estimar la ruta cirtica para un grupo de tareas relacionadas
3. Llevar a la practica el modelo CPM.

1. Diseña un software que simule el funcionamiento de una cola.
2. La cocina de un restaurante debe tener su vajilla lavada con respecto al orden de llegada.
3. Llevar a la practica la teoria de pilas.

4. Codigo

Para el desarrollo del ejercicio se hizo uso del lenguaje C, mediante el entorno de desarrollo Dev C++, por el cual se planteo esta clase que se ve a continuacion:

En la cual podemos identificar lo siguiente:

```
struct node{
  char name;
  vector<node*> childs;
  int duration;
  int t1, t2 , t3, t4;
  node(){};
  node( char _name, int _d): name(_name), duration(_d){
    t1 = t2  = t4;
    t3 = 1e5;
  }
};
```

Figura 1: estructura Nodo

1. Atributos:
   a) name: que define el nombre del nodo
   b) duration : de tipo entero, define la duracion de la tarea en este caso

```
int main(){
  int n;
  cin >>n ;
  char from, to;
  int duration;
  cout << "times\n";
  while(n--){
    cin >> to >> from >> duration;
    int id1 = from - 'A';
    int id2 = to - 'A';
    if( from == '-') id1 = 0;
    node *aux = new node(to, duration);
    graph[id1].push_back(aux);
    desc[id2] = aux;
  }
  // compute the times t1, t2, t3, t4
  // we know that the root is 'A', then
  int root = 'A' - 'A';
  desc[root]->t2 = desc[root]->duration;
  dfs(root, desc[root]->duration);

  cout << "Node t1 t2 t3 t4 holgura(t4-t1)"<<endl;

  for( int i = 0; i< 100; i++){
    if( desc[i]){
      int holgura = desc[i]->t4 - desc[i]->t1;
      cout << desc[i]->name << " " << desc[i]->t1 <<  " " << desc[i]->t2 << " " ;
      cout << desc[i]->t3 << " " << desc[i]->t4 << " " <<holgura<<endl;
    }
  }
  // now find the critical path
  dfs2(root);
  cout <<"Critic path\nbegin->";
  for(char c: critic_path){
    cout << c << "->";
  }
  cout<<"end";
  cout <<endl;
```

Figura 2: Metodo principal

Aqui podemos ver la implementacion de la estructura, en la cual segun la duracion se agrega un numero de nodos,la cual simula la ruta critica.
En la salida podremos observar el funcionamiento del algoritmo donde se establecen los tiempos minimos y maximos y por consiguiente su holgura .

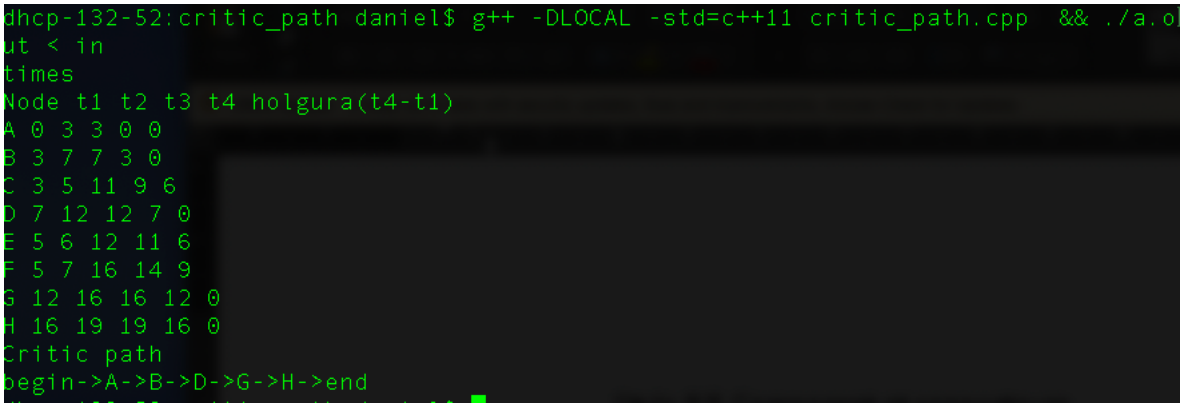

Figura 3: Salida del codigo

Acabamos de ver en la figura 3 la funcionalidad del programa de la ruta critica, ahora podemos observar los metodos utilizados en el programa para la determinacion de la holgura y de las tareas respecto a los tiempos.

Como se ve en la figura 4, Podemos observar la descripcion de los nodos estableciendo las respectivas relaciones, esta salida representa la descripcion del grafo que realiza el programa.

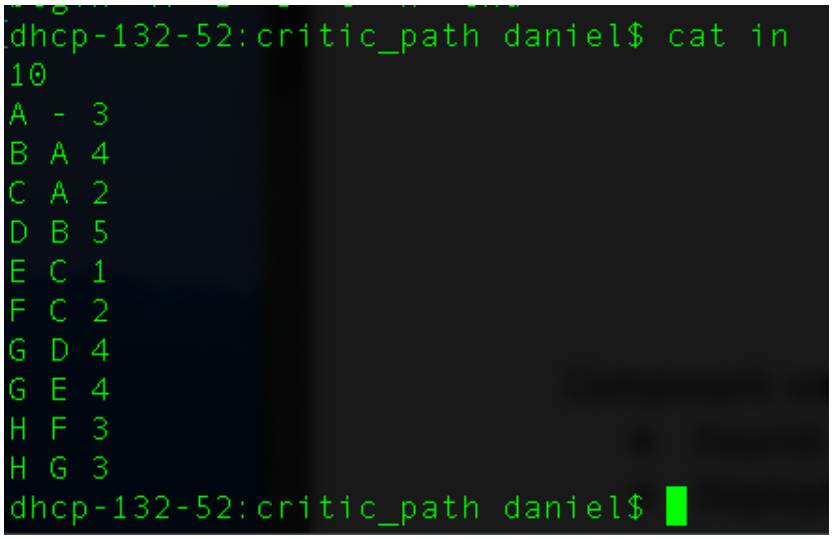

Figura 4: Salida descripcion del nodo

```
vector< node* >  graph[100];
vector<char> critic_path;
map<int, int> vis;
node*  desc[100];

int dfs( int src, int acum){
  for( node* son: graph[src]){
    int id = son->name - 'A';
    if( id != src){
      desc[id]->t1 = max(acum, desc[id]->t1);
      desc[id]->t2 = max(desc[id]->t2, acum+desc[id]->duration);
      desc[src]->t3 = min( desc[src]->t3, dfs(id,desc[id]->t2 ));
    }
  }
  if( size(graph[src])  == 0) desc[src]->t3 =  desc[src]->t2;
  desc[src]->t4 = desc[src]->t3 - desc[src]->duration;
  return desc[src]->t4;
}
```

Figura 5: Relacion multiple entre nodos y Metodo dfs

Podemos observar que un nodo puede apuntar a ningun o mas nodos, de esta manera se establace la relacion entre las tareas permitiendo que algun nodo se la tarea final al solo tener un predecesor y no un sucesor.

Por otra parte podemos identificar que el metodo dfs se encarga de determinar los tiempos maximos y minimos segun la relacione entre las tareas para asi encontrar el tiempo de holgura y por lo tanto determinar si una tarea es cirtica, y por consiguiente hace parte de la ruta

```
void dfs2( int src){
  if( desc[src]->t2 == desc[src]->t3){
    critic_path.push_back(desc[src]->name);
  }
  for( node* son: graph[src]){
    int id = son->name - 'A';
    if( id != src && vis[id]==0){
      vis[id] = 1;
      dfs2(id);
    }
  }
}
```

Figura 6: Metodo dfs2

Este metodo permite establecer las relaciones entre padre e hijo o en palabras mas concretas los sucesores y predecesores, del nodo o en un caso practico de las tareas que hacen parte del grafo.

5. Conclusion

1. Se pudo establecer la correcta implementacion de la ruta critica .
2. Se llevo a la practica el uso del metodo PERT, para establecer una ruta critica en tareas preestablecidas con tiempos determinados
3. Se ha podido observar la importancia de este tipo de estruturas o maneras de funcionamiento de un sistema para el uso de compiladores o analizadores sintacticos.

6. Referencias

1. Joyanes Aguilar, L. (2003). Fundamentos de programación: algoritmos y estructura de datos y objetos.
2. Engineer4Free(2014) . Use forward and backward pass to determine project duration and critical path