

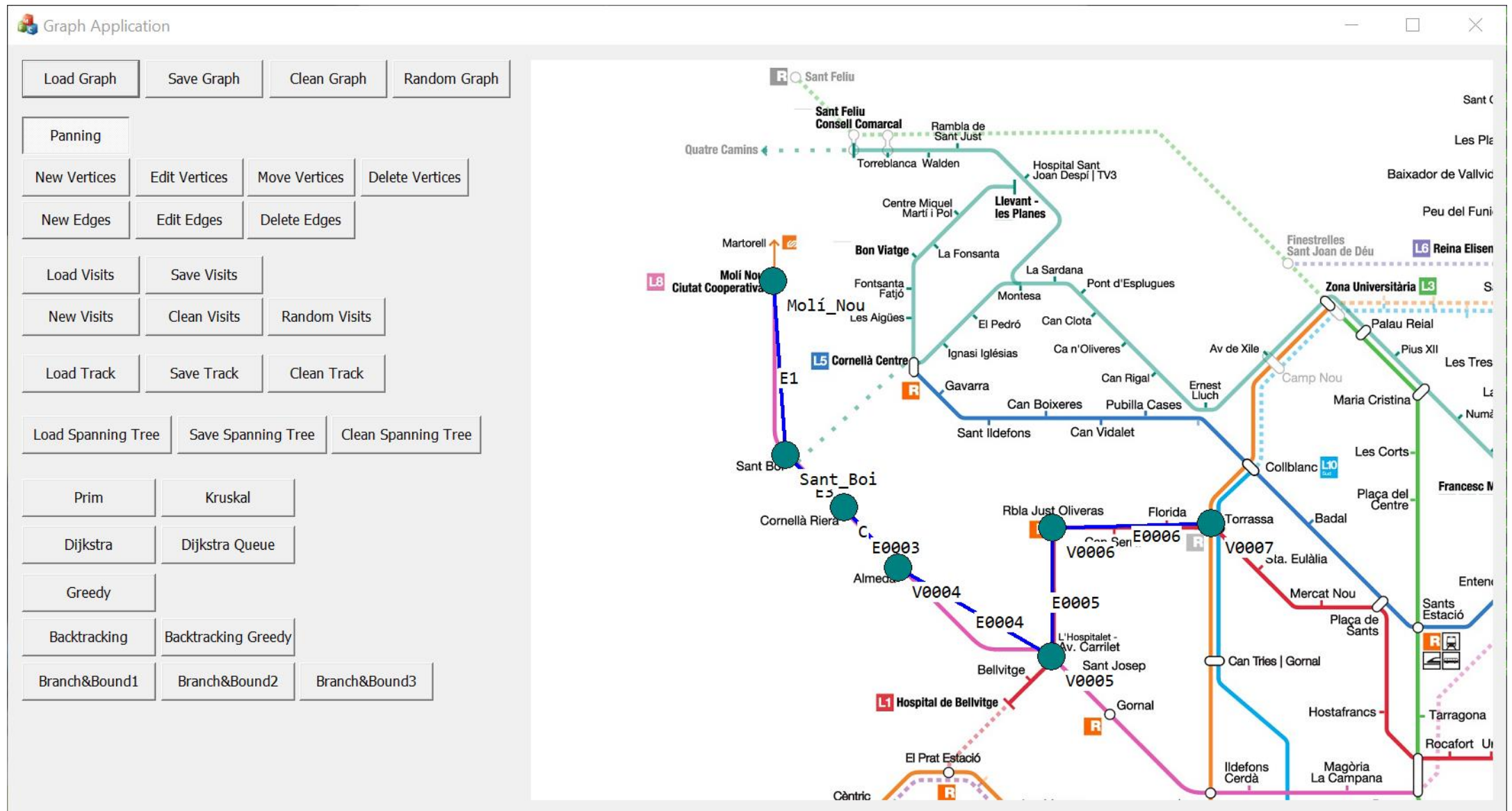
# **Programa GraphApplication**

Interfície d'usuari

Representació del graf, camins, arbres i visites.

Autocorrecció

# GraphApplication



## GraphApplication

- Es un programa per provar algorismes sobre grafs.
  - Arbres de cobertura.
  - Distàncies i camins.
  - Problema del viatjant de comerç.
  - Etc.
- Esta fet amb Visual Studio en C++ amb les llibreries MFC.
- Permet:
  - Crear grafs i llistes de visites manualment, o aleatòriament.
  - Es fàcil afegir algorismes.
  - Visualitza els resultats per comparar.
  - Es pot cridar des de línia de comandes per executar algorismes concrets. Aquesta característica s'utilitza per fer la correcció automàtica de problemes i pràctiques.

# **GraphApplication**

## **Interfície d'usuari**

# Visio general

Creació, llegir i  
escriure grafs

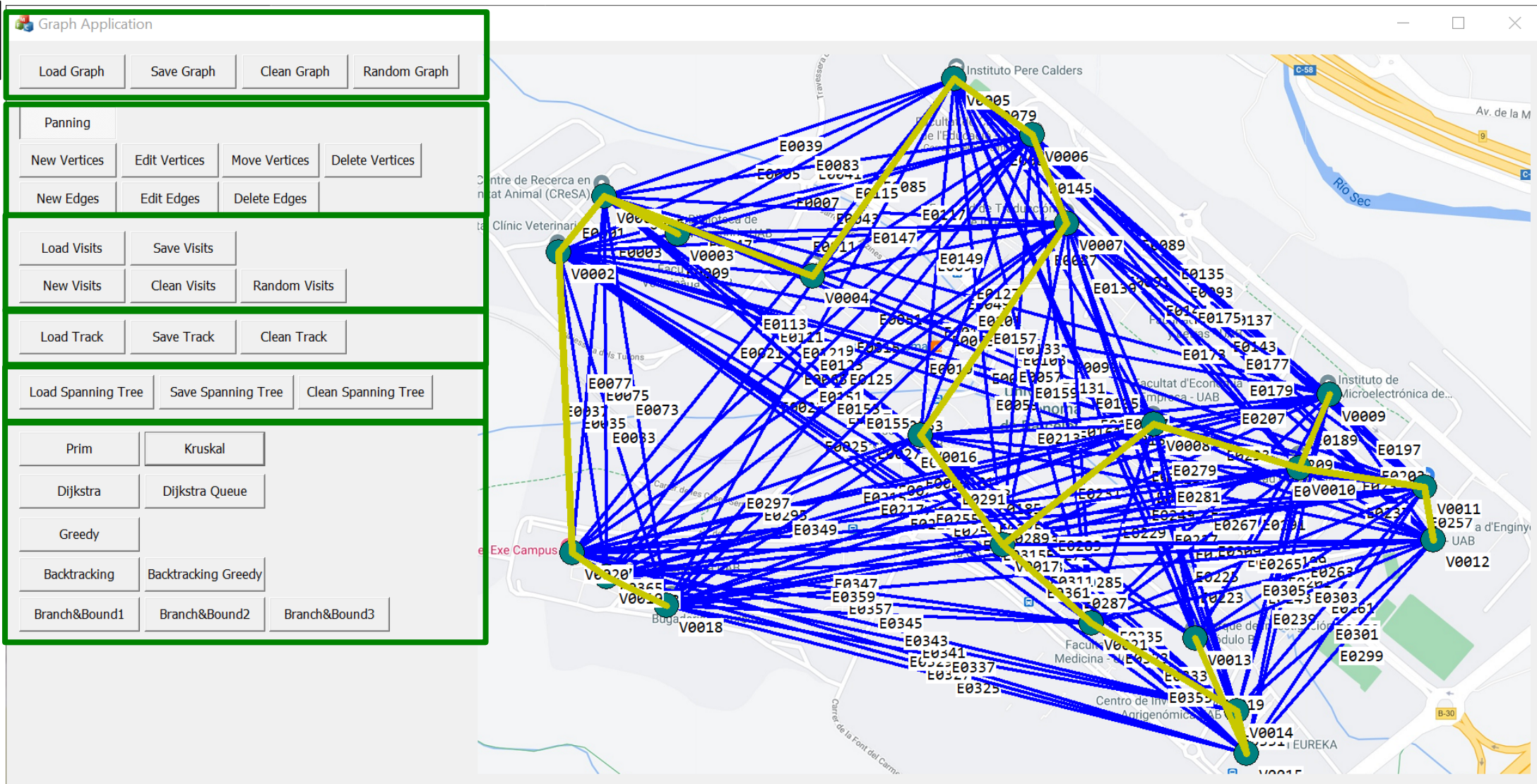
Edició grafs

Creació, edició  
llegir i escriure  
visites

llegir i escriure  
camins

llegir i escriure  
arbres de cobertura

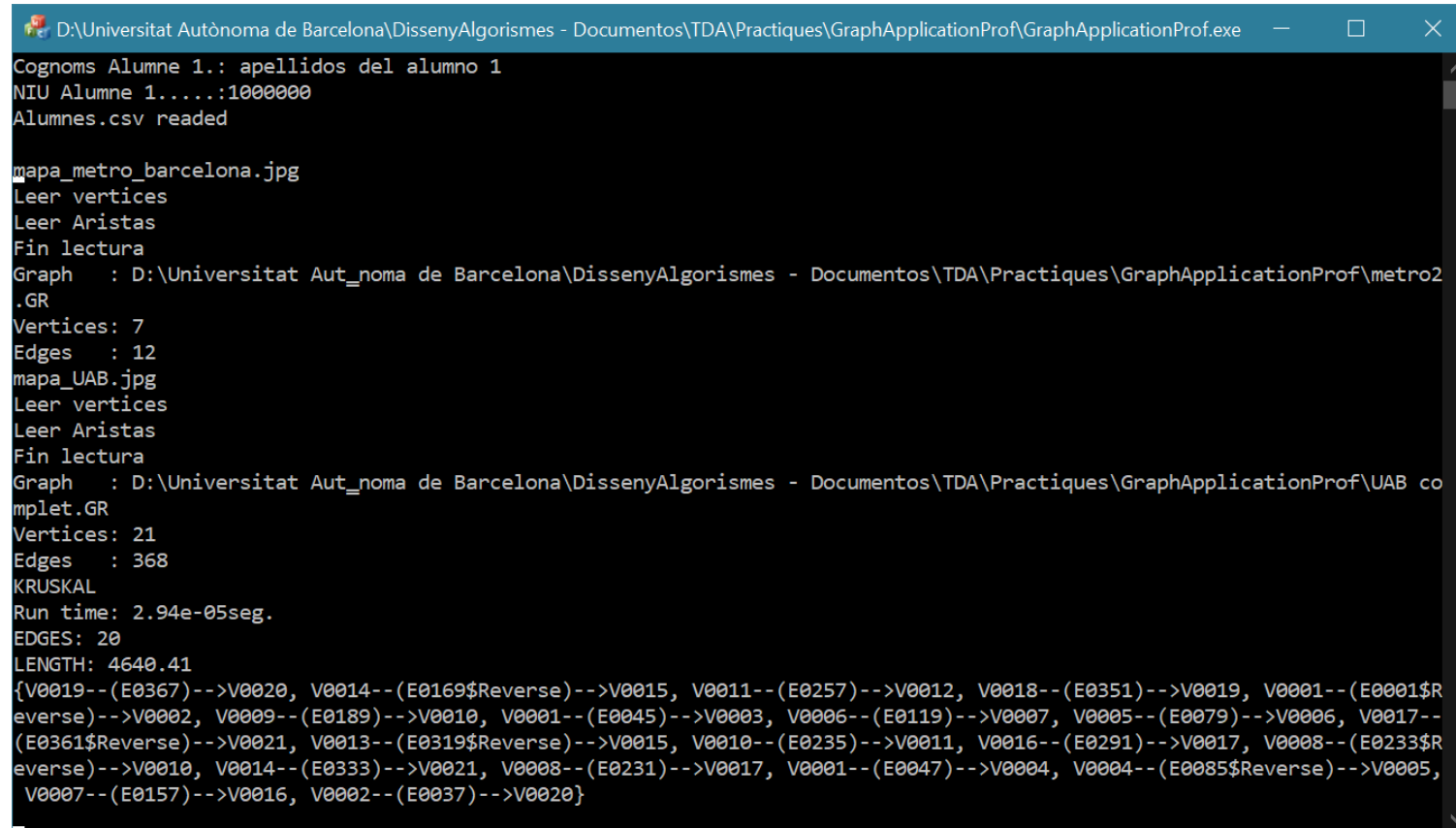
Algorismes



## **Control de la visualització del graf.**

- Botó Panning activat: Moure graf arrastrant amb el botó esquerra del ratolí
- Rodeta polsada: Moure graf arrastrant amb la rodeta.
- Girar rodeta: zoom.
- Doble pulsació rodeta: zoom a veure tot el graf dintre de la finestra.
- Ctrl-0: zoom a veure tot el graf dintre de la finestra
- Ctrl-Alt-0: zoom a mida real
- +: zoom fer mes gran.
- -: zoom fer mes petit

# Consola Negra



```
D:\Universitat Autònoma de Barcelona\DisenyAlgorismes - Documentos\TDA\Practiques\GraphApplicationProf\GraphApplicationProf.exe
Cognoms Alumne 1.: apellidos del alumno 1
NIU Alumne 1.....:1000000
Alumnes.csv readed

mapa_metro_barcelona.jpg
Leer vertices
Leer Aristas
Fin lectura
Graph : D:\Universitat Aut_noma de Barcelona\DisenyAlgorismes - Documentos\TDA\Practiques\GraphApplicationProf\metro2.GR
Vertices: 7
Edges : 12
mapa_UAB.jpg
Leer vertices
Leer Aristas
Fin lectura
Graph : D:\Universitat Aut_noma de Barcelona\DisenyAlgorismes - Documentos\TDA\Practiques\GraphApplicationProf\UAB co
mplet.GR
Vertices: 21
Edges : 368
KRUSKAL
Run time: 2.94e-05seg.
EDGES: 20
LENGTH: 4640.41
{V0019--(E0367)-->V0020, V0014--(E0169$Reverse)-->V0015, V0011--(E0257)-->V0012, V0018--(E0351)-->V0019, V0001--(E0001$R
everse)-->V0002, V0009--(E0189)-->V0010, V0001--(E0045)-->V0003, V0006--(E0119)-->V0007, V0005--(E0079)-->V0006, V0017--
(E0361$Reverse)-->V0021, V0013--(E0319$Reverse)-->V0015, V0010--(E0235)-->V0011, V0016--(E0291)-->V0017, V0008--(E0233$R
everse)-->V0010, V0014--(E0333)-->V0021, V0008--(E0231)-->V0017, V0001--(E0047)-->V0004, V0004--(E0085$Reverse)-->V0005,
V0007--(E0157)-->V0016, V0002--(E0037)-->V0020}
```

- Visualitza que esta fen el programa.
- Es pot escriurà en aquesta finestra utilitzant el stream cout de C++.

## **Estructures de Dades**

Representació del graf

Representació de visites

Representació de camins

Representació d'arbres de cobertura



## Representació d'un graf

- Un graf es representa com una llista de vèrtexs i una llista d'arestes.
- Cada vèrtex te la llista d'apuntadors a les arestes que surten d'ell.
- Cara aresta apunta al vèrtex origen d'on surt i al vèrtex destí on arriba.
- En els graf no dirigits cada aresta es representa com dos arestes dirigides en sentits contraris.
- Un graf pot tenir una imatge de fons associada per la visualització gràfica.
- Els graf es guarden en fitxers amb extensió .GR

```
class CGraph {  
public:  
    list<CVertex> m_Vertices; // llista de vèrtexs  
    list<CEdge> m_Edges; // llista d'arestes  
    bool m_Directed; // graf dirigit o no dirigit  
    string m_Filename; // Fitxer d'on s'ha llegit el graf  
    string m_BackgroundFilename; // Fitxer de la imatge de fons  
    CImage* m_pBackground; // imatge de fons o NULL  
};
```

## Vèrtex

- Cada vèrtex te un nom únic que no es pot repetir per altres vèrtexs.
- La longitud de l'aresta es la distancia entre els vèrtexs que uneix. Està calculada en el camp `m_Length`.
- Les arestes de grafs no dirigits es representen amb dos `CEdge` en sentits contraris. El nom del `CEdge` en sentit contrari es `nom$Reverse`

```
class CVertex {  
public:  
    string m_Name; // Nom del vertex  
    CGPoint m_Point; // Punt del pla del vèrtex.  
    list<CEdge*> m_Edges; // llista d'arestes que surten del vèrtex.  
    // Atributs de Dijkstra  
    double m_DijkstraDistance; // Distancia calcula per Dijkstra  
};
```

## Aresta

- Cada aresta té un nom únic que no es pot repetir per altres arestes.
- La longitud de l'aresta és la distància entre els vèrtexs que uneix. Està calculada en el camp `m_Length`.
- Les arestes de grafs no dirigits es representen amb dos `CEdge` en sentits contraris. El nom del `CEdge` en sentit contrari és `nom$Reverse`

```
class CEdge {  
public:  
    string m_Name; // Nom del edge  
    double m_Length; // Longitud de l'aresta (distància entre vèrtexs)  
    CVertex* m_pOrigin; // Vèrtex d'on surt l'aresta  
    CVertex* m_pDestination; // Vèrtex d'on arriba l'aresta  
    CEdge* m_pReverseEdge; // aresta en sentit contrari per grafs no dirigits  
};
```

## Llista de visites

- Una llista de visites es una llista de vèrtexs d'un graf.
- Una llista de visites està associada a un graf que la suporta.
- Un camí que recorri les visites comença per la primera de la llista i acaba en l'última de la llista. Les visites intermèdies les pot visitar en qualsevol ordre.
- Un vèrtex només pot estar una única vegada en la llista de visites, excepte quan la llista es cíclica i comença i acaba en el mateix vèrtex.
- Extensió dels fitxers de visites: .VIS

```
class CVisits {  
public:  
    list<CVertex*> m_Vertices; // Llista d'apuntadors als vèrtexs del graf  
    CGraph* m_pGraph; // graf que conté els vèrtexs.  
};
```

## Camí

- Una camí es una llista d'arestes d'un graf.
- Un camí està associat a un graf que el suporta.
- Les arestes d'un camí compleixen que el vèrtex on acaba una aresta, comença la següent.
- Extensió dels fitxers d'un camí: .TRK

```
class CTrack {  
public:  
    list<CEdge*> m_Edges;  
    CGraph* m_pGraph;  
};
```

## Arbre de cobertura (Spanning tree)

- Un arbre de cobertura és una llista d'arestes d'un graf.
- Un arbre de cobertura està associat a un graf que el suporta.
- Extensió dels fitxers d'un arbre de cobertura: .TRE

```
class CSpanningTree {  
    public:  
    list<CEdge*> m_Edges;  
    CGraph* m_pGraph;  
};
```

## Exemples de codi: Cua amb prioritat

- Crear un heap utilitzant una cua amb prioritat

```
struct comparator {  
    bool operator()(Element &e1, Element &e2) {  
        return pE1->m_Length > pE2->m_Length;  
    }  
};  
priority_queue<Element, std::vector<Element>, comparator>  
queue;
```

- Afegir un element: `queue.push(e);`
- Llegir el element mes petit: `e=queue.top();`
- Treure l'element mes petit de la cua: `queue.pop();`

## Processar els elements del graf.

- Processar tots els vèrtexs del graf: `CGraph g;`
  - `for (CVertex &v : g.m_Vertices) processar v;`
- Processar totes les arestes del graf: `CGraph g;`
  - `for (CEdge &e : g.m_Edges) processar e;`
- Processar totes les arestes que surten d'un vèrtex `CVertex &v;`
  - `for (CEdge *pE : v.m_Edges) processar *pE;`
- Vèrtexs units per una aresta `CEdge *pE;`
  - `pE->m_pOrigin`
  - `pE->m_pDestination`



## Exemple: Marcar vèrtexs connectats amb un vèrtex

- Afegir un atribut a Cvertex

```
class CVertex {  
    public:  
    string m_Name;  
    CGPoint m_Point;  
    list<CEdge*> m_Edges;  
    bool m_Marca; // Marca  
};
```

## Exemple: Marcar vèrtexs connectats amb un vèrtex

- Funció Marcar

```
void Marcar(CGraph& g, CVertex* pVertex) {
    stack<CVertex*> pila;
    for (CVertex& v : g.m_Vertices) v.m_Marca = false;
    pVertex->m_Marca = true;
    pila.push(pVertex);
    while (!pila.empty()) {
        CVertex* pV = pila.top();
        pila.pop();
        for (CEdge* pE : pV->m_Edges) {
            if (!pE->m_pDestination->m_Marca) {
                pE->m_pDestination->m_Marca = true;
                pila.push(pE->m_pDestination);
            }
        }
    }
}
```

## **Autocorrecció**

## Identificació dels alumnes que fan el lliurament

- Abans de fer qualsevol prova amb GraphApplication.exe s'han de replanar les dades dels alumnes que fan l'entrega al fitxer GraphApplication.cpp

```
// =====  
// IDENTIFICACION DE LOS ALUMNOS =====  
// =====
```

```
CString NombreAlumno1 = "nombre del alumno 1";  
CString ApellidosAlumno1 = "apellidos del alumno 1";  
CString NIUAlumno1 = "0000000"; // NIU alumno1  
  
// No rellenar en caso de grupo de un alumno  
CString NombreAlumno2 = "nombre del alumno 2";  
CString ApellidosAlumno2 = "apellidos del alumno 2";  
CString NIUAlumno2 = ""; // NIU alumno2
```

## Fitxers per l'autocorrecció

- El corrector per entregues fetes amb GraphApplication es Corrector.exe, però no es crida directament. Per utilitzar-lo s'han creat uns fitxers .bat que donen les diverses modalitats de correcció
  - GenerarTiempsDeReferencia*Entrega*.bat
    - Genera els temps de referencia per fer les correccions i genera el fitxer EntregaRunTimes.txt. **Aquet fitxer es necessari per corregir el lliurament.**
  - Corregir*EntregaRelease*.bat
    - Corregeix l'entrega compilada en versió release.
  - Corregir*EntregaDebug*.bat
    - Corregeix l'entrega compilada en versió debug.
  - Corregir*EntregaTempsExtesRelease*.bat
    - Corregeix l'entrega compilada en versió release amb més temps per executar els algorismes (la nota obtinguda no es vàlida).
  - Corregir*EntregaTempsExtesDebug*.bat
    - Corregeix l'entrega compilada en versió debug amb més temps per executar els algorismes (la nota obtinguda no es vàlida).
  - CorregirProf*Entrega*.bat
    - Corregeix GraphApplicationProf.exe que es la versió del professor

## Resultat de la correcció

- El resultat de la correcció es un informe amb el nom
  - Lliurament per un alumne: *NIU\_Lliurament VNumVersió.txt*
  - Lliurament per un dos alumnes: *NIU1\_NIU2\_Lliurament VNumVersió.txt*
- El contingut de l'informe és:
  - Lliurament al que correspon.
  - Data.
  - Executable.
  - Alumnes
  - Proves correctes i fallades.
  - Errors detectats
  - Nota aproximada.

## Exemple d'informe correcte

INFORME DE CORRECCION SpanningTreePrim

=====

Fecha: 29/09/2020

Hora: 15:21:12

Ejecutable: x64\Release\GraphApplication.exe

ALUMNO: 1000000 APELLIDOS PROFESOR, NOMBRE PROFESOR javier@cvc.uab.cat

=====

PRUEBA CORRECTA: UnaAresta::SpanningTreePrim TIEMPO: 0.000002s. (0.000002s.)

-----

SPANNING TREE OK : {V0001--(E0001)-->V0002}

SPANNING TREE PRACTICA: {V0001--(E0001)-->V0002}

...

=====

RESUMEN =====

=====

PRUEBAS OK (7): senseArestes::SpanningTreePrim, UnaAresta::SpanningTreePrim,  
TresArestes::SpanningTreePrim, Desconectat::SpanningTreePrim, UAB\_complet::SpanningTreePrim,  
Graf100::SpanningTreePrim, Graf1000::SpanningTreePrim

PRUEBAS NOK(0):

TIEMPO: 0.032566.s (0.033345s.)

NOTA: 10.000000

## Exemple d'informe amb errors

INFORME DE CORRECCION SpanningTreePrim

=====

Fecha: 29/09/2020

Hora: 15:38:21

Ejecutable: x64\Release\GraphApplication.exe

ALUMNO: 1000000 APELLIDOS PROFESOR, NOMBRE PROFESOR javier@cvc.uab.cat

=====

PRUEBA FALLA: TresArestes::SpanningTreePrim TIEMPO: 0.000005s. (0.000004s.)

-----

SPANNING TREE OK : {V0001--(E0001)-->V0002, V0001--(E0003)-->V0003}

SPANNING TREE PRACTICA: {}

EDGES QUE FALTAN:

EDGE(E0001,V0001-->V0002)

EDGE(E0003,V0001-->V0003)

...

=====

RESUMEN =====

=====

PRUEBAS OK (1): senseArestes::SpanningTreePrim

PRUEBAS NOK(6): TresArestes::SpanningTreePrim ...

TIEMPO: 0.037205.s (0.033345s.)

NOTA: 1.000