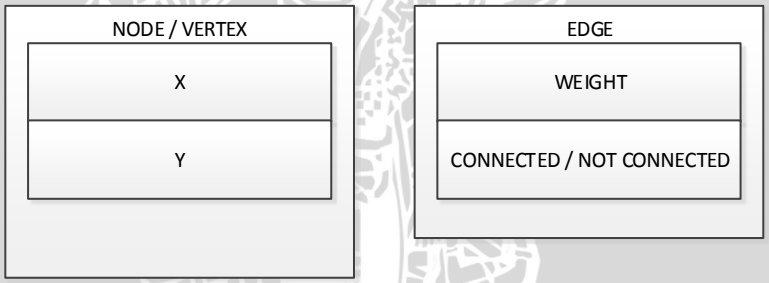
BAB IV
PERANCANGAN DAN IMPLEMENTASI

Bab ini menjelaskan mengenai langkah-langkah yang akan dilakukan untuk merancang dan mengimplementasikan aplikasi konversi citra labirin ke dalam *edge* dan *vertex*. Perancangan dan implementasi dikerjakan dengan beberapa tahap. Meliputi pengolahan citra labirin ke dalam bentuk citra *grayscale* dan *binary*, proses pencarian *node*, mengubah *node* kedalam *graph*, serta menyimpannya kedalam bentuk *plaintext*.

4.1 Struktur Data

Dalam pemrograman pengolahan citra labirin ini menggunakan struktur *data* berbentuk *graf* sebagai hasil akhir dari proses pengolahan citra labirin yang disimpan dalam bentuk *plaintext*. *Graph* yang digunakan dalam program ini adalah tipe *graph* tidak berarah dan berbobot. *Graph* ini merupakan bentuk *bidirectional graph* yang mempunyai nilai pada *edge* nya. Ilustrasi *edge* dan *vertex* seperti ditunjukkan pada gambar 4.1.

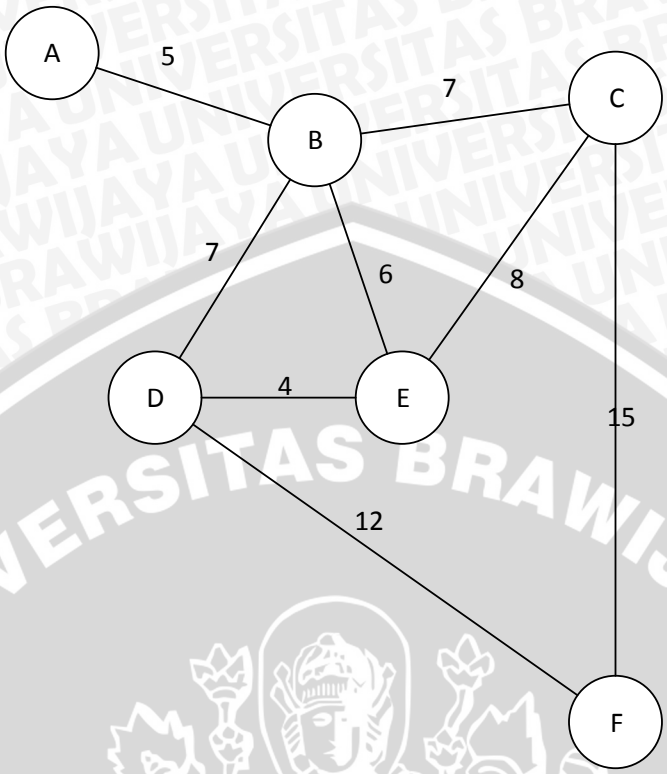


Gambar 4.1 Ilustrasi *Edge* dan *Vertex*

Pada *vertex* terdiri x, y. X dan y digunakan untuk menyimpan koordinat dari titik *vertex* pada *citra* area labirin. *Weight* adalah nilai pada *edge* yang menghubungkan antar masing – masing *vertex*. *Connected / not connected* digunakan untuk menyimpan status dari masing-masing *vertex*, apakah *vertex* tersebut terhubung atau tidak dengan *vertex* tetangga atau *vertex* terdekatnya.

Didalam suatu area labirin, *vertex* yang ada mempunyai identitas yang berbeda-beda tergantung dari letak dan status masing-masing *vertex* tersebut. Untuk mewakili area labirin berupa *graph*, maka jumlah *vertex* yang disimpan akan tergantung dari jumlah persimpangan dan jalan buntu yang ada. Secara keseluruhan *graph* dari *ouput* program adalah tipe *graph* tidak berarah dan

berbobot yang dapat diilustrasikan seperti pada gambar 4.2 .



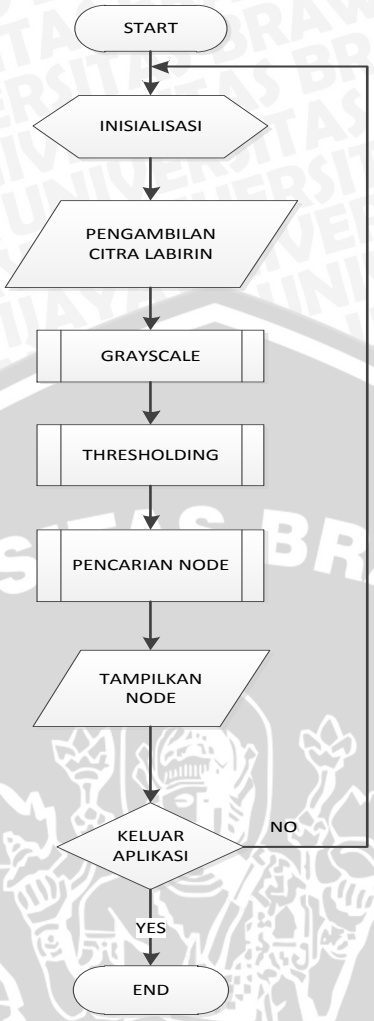
Gambar 4.2 Ilustrasi *Undirected Graph, Weighted Edges*

4.2. Perancangan Program

Pada bagian perancangan aplikasi yang akan dibuat menggunakan bahasa pemrograman *Microsoft visual studio C# 2008* dan aplikasi yang dibuat nantinya dapat melakukan hal-hal sebagai berikut :

1. Mengolah citra labirin sehingga dapat memperoleh *edge* dan *vertex*, serta dapat melakukan *overlay node* kedalam citra labirin.
2. Melakukan penyimpanan *edge* dan *vertex* yang telah ditemukan kedalam bentuk *graph* dengan format *plaintext*.

Flowchart dari perancangan program secara umum seperti yang ditunjukan oleh gambar 4.3.



Gambar 4.3 *Flowchart* program secara umum

4.2.1 Perbaikan Citra

Pengambilan citra berasal dari citra labirin hasil *scanner* atau berupa citra labirin *digital* dengan format citra yang dapat berupa *jpeg, jpg, png* dan *bmp*. Setelah citra labirin di *load* kedalam program maka langkah selanjutnya adalah melakukan perbaikan citra untuk mempermudah dalam proses pencarian *node*. Perbaikan citra dilakukan secara berurutan yaitu mengkonversi citra *true color* ke dalam bentuk *grayscale* dan kemudian mengkonversi ke dalam bentuk citra biner.

Pada pengubahan sebuah gambar menjadi *grayscale* dapat dilakukan dengan cara mengambil semua *pixel* pada gambar kemudian warna tiap *pixel* akan diambil informasi mengenai 3 warna dasar yaitu merah, biru dan hijau, ketiga warna dasar ini akan dijumlahkan sesuai algoritma citra *grayscale*. Nilai yang didapatkan dari penjumlahan tadi inilah yang akan dipakai untuk memberikan

warna pada *pixel* gambar sehingga warna menjadi *grayscale*, tiga warna dasar dari sebuah *pixel* akan di*set* dengan nilai tersebut. Secara matematis, perhitungan citra *grayscale* ditunjukkan pada rumus 2.1.

Grayscale = 0.30R + 0.559G + 0.11B (2.1)

Dengan R adalah merah, G adalah hijau dan B adalah biru. Setelah diubah ke dalam bentuk citra *grayscale*, selanjutnya adalah mengubah citra *grayscale* ke dalam citra biner dengan cara mengelompokkan nilai *pixelnya*, jika *pixel* tersebut bernilai kurang dari 128 maka nilai *pixel* tersebut akan diubah kedalam *pixel* 0 (hitam), begitu juga sebaliknya jika nilai *pixel* tersebut lebih dari 128 maka akan diubah kedalam *pixel* 255 (putih). Implementasi programnya seperti ditunjukkan pada gambar 4.4.

```
for (int x = 0; x < sourceBmp.Width; x++)
{
    for (int y = 0; y < sourceBmp.Height; y++)
    {
        Color originalColor = sourceBmp.GetPixel(x, y);

        int intValue = (int)((originalColor.R * .30) + (originalColor.G * .59) + (originalColor.B * .11));
        grayscaleBmp.SetPixel(x, y, Color.FromArgb(intValue, intValue, intValue));

        if (intValue < 128)
            intValue = 0;
        else
            intValue = 255;

        binaryBmp.SetPixel(x, y, Color.FromArgb(intValue, intValue, intValue));
        finalBmp.SetPixel(x, y, Color.FromArgb(intValue, intValue, intValue));
    }
}
```

Gambar 4.4 Program *Grayscale* dan *Threshold*

4.2.2 Pencarian Nilai Border

Saat citra labirin sudah melalui proses perbaikan citra dan citra sudah dikonversi ke dalam bentuk biner, hal selanjutnya adalah dengan melakukan proses pemecahan labirin hingga nantinya diperoleh *edge* dan *vertex* beserta seluruh data citra labirin meliputi lebar dinding, lebar lorong, serta jumlah *vertex*nya.

Tahap awal dalam pengolahan citra labirin ini yaitu menentukan jumlah titik bagi yang nantinya digunakan sebagai titik – titik pencarian untuk mencari nilai lebar *border* pada citra labirin dan mencari nilai lebar dinding dan lorong citra labirin. Titik bagi yang digunakan pada program ini adalah 100 titik bagi, hal tersebut dimaksudkan untuk mendapatkan keakuratan data yang diperoleh baik nilai lebar *border* maupun nilai lebar dinding dan lorong.

```
private const int NUM_OF_SAMPLE = 101;
```

Untuk menghindari terjadinya duplikasi pada saat pengambilan sampel, maka dalam pengambilan sampel dibuat suatu jarak yang konstan antar masing-masing titik sample. Oleh karena itu *width* dan *height* dibagi dengan jumlah titik sampel tersebut untuk mendapatkan jarak antara masing-masing titik sampel.

```
xSpliter = _bitmap.Width / NUM_OF_SAMPLE,
ySpliter = _bitmap.Height / NUM_OF_SAMPLE;
```

Proses pemecahan labirin dimulai dari (*xPosition*, *yPosition*) pada *pixel* 0,0. Proses pengecekan dari sumbu *x* dari sisi atas kebawah dan dari sisi bawah ke atas. Pengambilan sampel sesuai titik bagi yang telah ditentukan, jika pada salah satu titik bagi didapati nilai *pixel* putih maka program mengambil nilai dari titik tersebut sampai didapatkan *pixel* hitam sebelum melanjutkan ke titik bagi berikutnya, tetapi jika pada titik bagi yang didapatkan adalah *pixel* hitam maka nilai putih tetap disimpan dengan nilai 0 (nol). Sama halnya dengan pengecekan dari sumbu *x*, pada sumbu *y* pengecekan dilakukan dari sisi kiri ke kanan dan sebaliknya.

Saat proses pemecahan awal selesai, nilai dari masing-masing sisi dikelompokkan untuk mendapatkan nilai titik putih yang paling banyak muncul, yang digunakan sebagai nilai *border* dari masing-masing sisi (*border* atas, *border* bawah, *border* kiri, dan *border* kanan).

4.2.3 Pencarian Ketebalan Dinding dan Lorong

Proses pencarian ketebalan dinding dan lorong sama dengan pencarian *border* dengan menggunakan titik bagi untuk mendapatkan data sampelnya, yang berbeda pada titik awal pencariannya. Proses pencarian dimulai dari (*xStart, yStart*), sedangkan jarak antar masing-masing titik bagi bukan berdasarkan pembagian *height* dan *width* citra dengan titik bagi melainkan dari nilai (*xEnd – xStart*) dan (*yEnd – yStart*) sebagai pengganti nilai *height* dan *width* citra.

```
xStart = _bitmap.Width - (_bitmap.Width - LeftBorder),
xEnd   = _bitmap.Width - RightBorder;

yStart = _bitmap.Height - (_bitmap.Height - UpperBorder),
yEnd   = _bitmap.Height - BottomBorder;

xSpliter = (xEnd - xStart) / NUM_OF_SAMPLE,
ySpliter = (yEnd - yStart) / NUM_OF_SAMPLE;

xPosition = xStart,
yPosition = yStart;
```

Pencarian ketebalan dinding dan lorong labirin dilakukan dalam dua tahap, yaitu dari sisi atas dan sisi kiri. Proses pertama adalah dari sisi atas dilakukan secara vertikal dimulai dari (*xPosition, y*). Proses pencarian dimulai dengan mendapatkan nilai *pixel* hitam dari sisi atas bergerak kebawah hingga bertemu *pixel* putih, selanjutnya mendapatkan nilai *pixel* putih dan akan berhenti saat pencarian bertemu *pixel* hitam. Dengan asumsi nilai *pixel* hitam adalah nilai sampel dinding dan nilai *pixel* putih adalah nilai sampel lorong. Pencarian dari sisi atas dilakukan sampai semua titik bagi sudah diproses, selanjutnya baru dilanjutkan dari sisi kiri, yang dimulai dari (*x, yPosition*). Proses pencarian dari sisi kiri sama halnya dengan proses dari sisi atas, hanya proses ini dilakukan secara *horizontal*.

Saat kedua proses selesai, nilai dari dinding dan lorong labirin dikelompokkan menjadi satu baik dari sisi atas maupun sisi kiri. Setelah itu dicari nilai yang paling banyak keluar sebagai tebal dinding dan lorong citra tersebut.

```
_whiteWall = (int)Math.Round((decimal)(topWhiteValue + sideWhiteValue) / 2, MidpointRounding.ToEven);
_blackWall = (int)Math.Round((decimal)(topBlackValue + sideBlackValue) / 2, MidpointRounding.ToEven);
```

4.2.4 Pencarian Titik Tengah Hitam dan Putih

Setelah mendapatkan ketebalan dinding dan lorong labirin, selanjutnya adalah melakukan pencarian titik tengah dari dinding dan lorong, yang nantinya akan diseleksi sebagai *node*. Dalam proses pencarian titik tengah ini terdapat dua komponen penting yang digunakan, yaitu nilai *x* (*horizontal*) dan nilai *y* (vertikal). Proses pertama adalah menentukan *startPoint*, untuk sumbu *x* (*horizontal*) nilai *stepPoint* awal adalah lebar dinding (hitam) + setengah lebar jalan (putih).

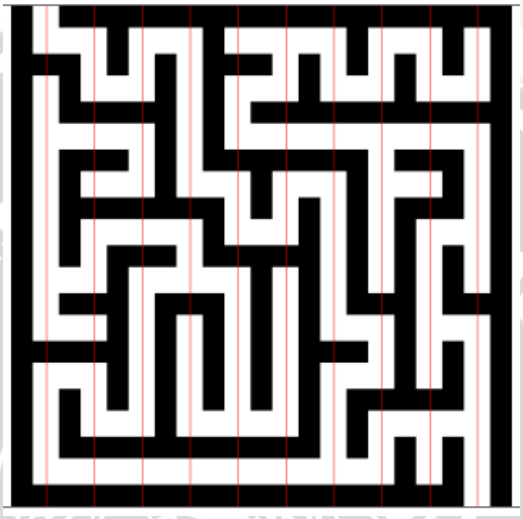
```
startPoint.X    = _bitmap.Width - (_bitmap.Width - LeftBorder);
startPoint.Y    = _bitmap.Height - (_bitmap.Height - UpperBorder);

stopPoint.X = _bitmap.Width – RightBorder;
stopPoint.Y = _bitmap.Height - BottomBorder;
```

Setelah didapatkan *stepPoint* awal selanjutnya mencari titik tengah berikutnya dengan *stepPoint* + *step*. Dalam hal ini *step* adalah

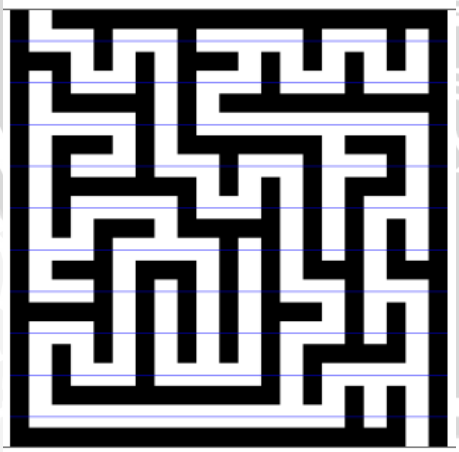
nilai lebar dinding + lebar lorong. Setelah ditemukan titik kedua, untuk titik-titik tengah selanjutnya didapat dengan menambahkan *step* pada titik tersebut. Proses untuk garis *x* (*horizontal*) berhenti saat pada titik *stepPoint* lebih besar atau sama dengan *stopPoint*.

Hasil titik tengah tersebut di visualisasikan kedalam citra labirin seperti pada gambar 4.5.

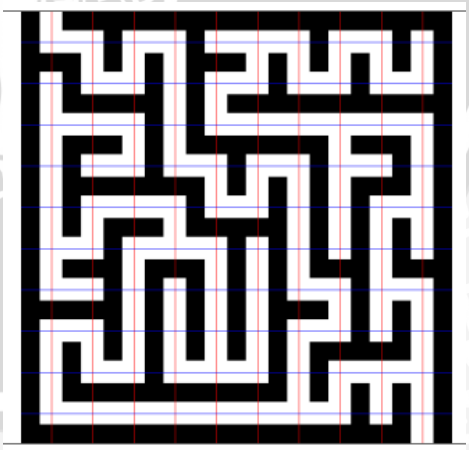


Gambar 4.5 Visualisasi nilai *x* (*horizontal*)

Untuk proses selanjutnya adalah proses berdasarkan sumbu *y* (vertikal).Dalam proses ini, langkah-langkah yang dilakukan sama seperti pada sumbu *x*, tetapi proses kedua ini secara vertikal. Hasil titik tengah pada sumbu *y* (vertikal) seperti ditunjukkan oleh gambar 4.6.



Gambar 4.6 Visualisasi nilai *y* (vertikal)



Gambar 4.7 Titik pertemuan x dan y

4.2.5 Pencarian Titik yang *Valid*

Dari titik pertemuan dari proses sebelumnya, hal selanjutnya adalah melakukan seleksi terhadap titik tersebut sehingga didapatkan *node* atau *vertex* yang *valid*. Dalam menentukan bahwa suatu *meetPoint*

yang ada termasuk kedalam *node* yang *valid* memerlukan beberapa tahapan seleksi. Pertama adalah *meetPoint* tersebut masih berada di dalam area citra labirin dan mempunyai nilai *pixel* putih.

(finalBmp.GetPixel(meetPoint[i].X + 1, meetPoint[i].Y + 1) == Color.FromArgb(255, 255, 255))

Tahapan kedua adalah melakukan penyeleksian *meetPoint* tersebut berdasarkan batasan tiap – tiap sisi nya untuk mendapatkan *node* yang *valid*. Pengecekan *meetPoint* untuk tiap – tiap sisinya dengan melakukan pengecekan ke sisi atas, bawah, kiri dan kanan. Pengecekan dimulai dari titik tengah atau *meetPoint* pada *pixel* putih bergerak sesuai sisi yang dihitung dengan konsep mencari titik hitam dengan batasan setengah lebar lorong + setengah lebar dinding.

(CheckNode(finalBmp, meetPoint[i], halfBlack + halfWhite))

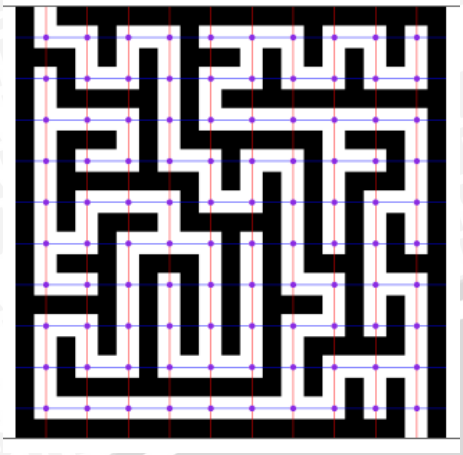
Untuk parameter *node* yang *valid* seperti ditunjukkan pada gambar 4.8.

```
top = CheckNodeHelper(bitmap, new Point(point.X, point.Y - step), point);
bottom = CheckNodeHelper(bitmap, point, new Point(point.X, point.Y + step));
left = CheckNodeHelper(bitmap, new Point(point.X - step, point.Y), point);
right = CheckNodeHelper(bitmap, point, new Point(point.X + step, point.Y));

if (left && top)
    isNode = true;
if (top && right)
    isNode = true;
if (right && bottom)
    isNode = true;
if (bottom && left)
    isNode = true;
if (left && top && right && bottom)
    isNode = true;
if (left && !top && !right && !bottom)
    isNode = true;
if (!left && top && !right && !bottom)
    isNode = true;
if (!left && !top && right && !bottom)
    isNode = true;
if (!left && !top && !right && bottom)
    isNode = true;
```

Gambar 4.8 Program pengecekkan *node*

Setelah didapatkan *node* yang *valid*, *node* – *node* tersebut di *overlay* kedalam citra labirin seperti ditunjukan oleh gambar 4.9.



Gambar 4.9 *Node* awal yang belum di seleksi

Gambar 4.10 *Node* yang sudah valid

4.2.6 Pencarian *Gateway*

Dalam suatu citra labirin *gateway* adalah suatu hal terpenting, karena *gateway* merupakan titik awal untuk melakukan suatu pencarian jalur terbaik. Untuk mencari *gateway* dilakukan pada keempat sisi citra labirin. Untuk dari sisi atas, pertama *StartPoint.Y* ditambah dengan setengah nilai dinding, dari titik tersebut pencarian dilakukan sesuai dengan sumbu x, mulai dari *StartPoint.X* sampai dengan *StopPoint.X* jika pada titik tersebut bertemu *pixel* putih maka akan dicek dahulu dengan *CheckNodeHelper* apakah termasuk *gateway* atau *noise*. Jika titik tersebut merupakan *gateway*, maka akan dibuat suatu *vertex* pada tengah *gateway* tersebut yaitu dengan menambahkan nilai setengah putih pada sumbu x. Pencarian selanjutnya sesuai dengan sumbu x dari titik terakhir hingga bertemu *pixel* putih dan kembali dilakukan pengecekan dengan *CheckNodeHelper*.

Untuk proses dari sisi sebelah kiri, bawah dan kanan sama halnya dengan sisi atas hanya terdapat perbedaan secara *horizontal* dan vertikal serta *startPoint* dan *stopPoint* nya. Hasil *gateway* yang sudah didapatkan seperti ditunjukkan oleh gambar 4.11.

Gambar 4.11 *Node gateway* dan *node valid*

4.2.7 Menghubungkan Antar *Node*

Setelah didapat semua *node* dan *gateway*, selanjutnya adalah menghubungkan *node* tersebut dan mendapatkan jarak antar masing – masing *node* yang terhubung. Sebelum melakukan proses pencarian *edge*, untuk mempermudah proses tersebut terlebih dahulu *node – node* yang sudah ada dikelompokan sesuai letak dan sumbunya, yaitu *groupX* untuk *node* pada sumbu *horizontal* dan *groupY* untuk *node* pada sumbu vertikal. Setelah dikelompokkan selanjutnya adalah melakukan inisialisasai bahwa semua node terhubung.

```
bool connected = true;
```

Selanjutnya adalah dengan melakukan pengecekan secara *horizontal* terdahap dua *node* yang berdekatan dan berada pada sumbu yang sama, jika diantara *node* tersebut tidak terdapat *pixel* hitam (dinding) maka *node* tersebut termasuk dalam *node* yang terhubung. Koordinat *node* tersebut disimpan dan dicari nilai *edge*-nya. Cara mencari nilai *edge*-nya adalah dengan dengan mengurangi koordinat *node* selanjutnya dengan *node* sekarang, dan untuk mendapatkan nilai dalam blok maka hasil pengurangan tersebut dibagi dengan lebarPutih (lorong).

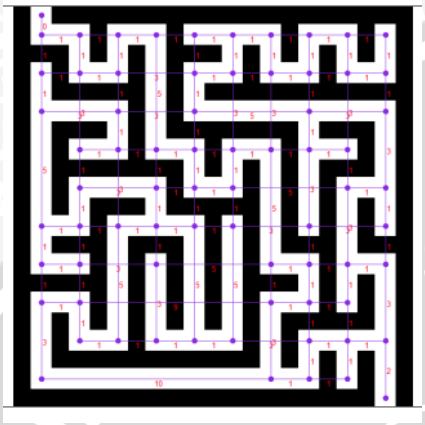
```
(group.xMember.ElementAt(i + 1).X - group.xMember.ElementAt(i).X) / LebarPutih));
```

Pengecekan selanjutnya secara vertikal berdasarkan sumbu *y*, dengan metode yang sama seperti pada tahap pertama.

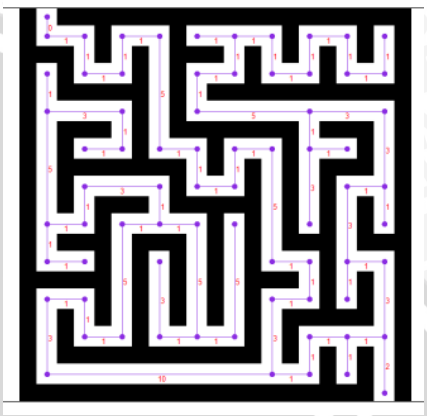
```
(group.yMember.ElementAt(i + 1).Y - group.yMember.ElementAt(i).Y) /
```

LebarPutih));

Setelah semua data *node*, *edge* dan nilai *edge*nya didapatkan semua disimpan dalam bentuk *graph* yang nantinya akan mempermudah untuk proses selanjutnya. Hasil dari pencarian hubungan antar *node* serta nilai *edge*-nya seperti pada gambar 4.13.



Gambar 4.12 Inisialisasi awal



Gambar 4.13 *Edge* hasil proses program

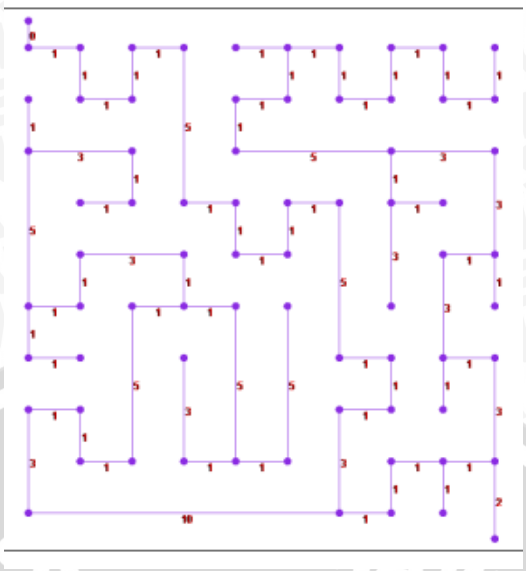
4.2.8 Proses Visualisasi *Graph*

Untuk dapat melakukan visualisasi hasil proses pemecahan labirin dalam bentuk *graph* maupun citra labirin dengan *node* dan *edge*-nya, maka diperlukan suatu proses penggambaran ulang seluruh data citra labirin yang telah diproses baik dari nilai *border*, ketebalan dinding dan lorong, letak *node* serta nilai *edge*-nya yang sudah disimpan kedalam bentuk *graph*.

Proses pertama adalah melakukan penggambaran *background* dengan warna dasar hitam serta nilai *height* dan *width* citra yang sesuai dengan citra asli.

```
Bitmap result = new Bitmap(width, height);
  for (int x = 0; x < width; x++)
      for (int y = 0; y < height; y++)
          result.SetPixel(x, y, Color.FromArgb(0, 0, 0));
 return result;
```

Setelah menggambar *background* selanjutnya adalah memvisualisasikan semua *data* yang terdapat pada citra labirin yang telah diproses. Proses awal dimulai dari koodinat *x,y* (0,0) dan diakhiri pada nilai *pixel* terakhir. Hasil visualisasi *graph* seperti ditunjukan oleh gambar 4.14.

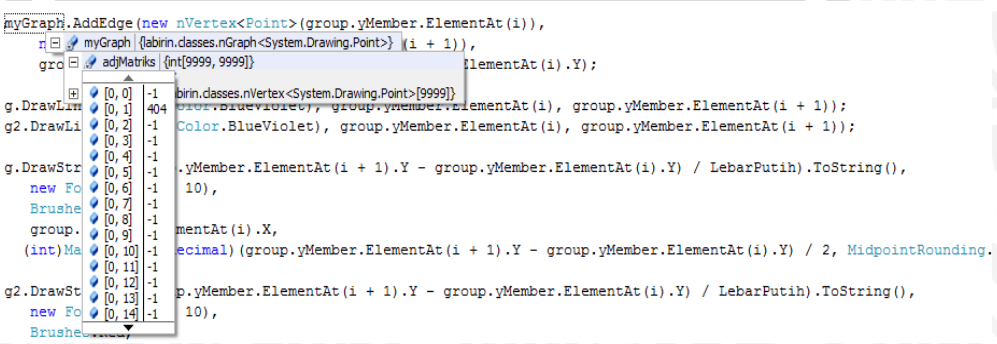


Gambar 4.14 Visualisasi *Graph*

4.2.9 Proses *Save* Citra Labirin ke Dalam *Plaintext*

Dalam suatu program yang telah dibuat memerlukan suatu model *output data* yang nantinya dapat digunakan secara umum dan program tersebut dapat juga memproses hasil *output* yang telah dibentuk. Untuk itu *output* dari program pengolahan citra ke dalam *edge* dan *vertex* diwujudkan dalam bentuk *plaintext* yang menyimpan semua *info* dari *graph* yang telah dihasilkan dari citra labirin. Sebelum dikonversi kedalam bentuk *plaintext*, bentuk dari *graph* hasil proses pengolahan citra labirin adalah dalam bentuk *array* matriks x,y.

Dalam matriks yang merepresentasikan data dari *vertex* dan *edge*, setiap *vertex* yang tidak berhubungan diberi nilai default yaitu -1, tetapi jika ada *vertex* yang berhubungan pada matriksnya diberi nilai jarak dari antar *vertex* tersebut. Visualisasi matriks *edge* dan *vertex* seperti pada gambar 4.16.



Gambar 4.15 Isi *graph* hasil pengolahan citra

Tabel 4.1 Representasi matrik citra labirin

y/x	25	76	127	177	229	278	331	379	433	480	535	581	637
25	0	3	0	0	0	0	0	0	0	0	0	0	0
76	0	3	1	1	1	1	1	1	1	3	0	3	0
127	0	0	0	0	0	0	0	0	0	1	0	1	0
177	0	3	1	1	1	1	1	3	0	1	0	1	0
229	0	1	0	0	0	0	0	1	0	1	0	1	0
278	0	1	0	3	1	3	0	1	0	1	0	1	0
331	0	1	0	1	0	1	0	1	0	1	0	1	0
379	0	1	0	1	0	3	0	1	0	1	0	1	0
433	0	1	0	1	0	0	0	1	0	1	0	1	0
480	0	1	0	3	1	1	1	3	0	3	1	3	0
535	0	1	0	0	0	0	0	0	0	1	0	0	0
581	0	3	1	1	1	1	1	1	1	3	1	3	0
637	0	0	0	0	0	0	0	0	0	0	0	3	0

Tabel 4.1 merupakan representasi matrik citra labirin yang telah diolah, dalam tabel tersebut merepresentasikan letak *vertex* dan *edge* pada *meetPoint* antara sumbu *x* (*vertical*) dan sumbu *y* (*horizontal*) dalam satuan *pixel*. Dalam hal ini koordinat dari sumbu *x* dan *y* adalah garis tengah pada lorong dan dinding citra labirin. Nilai 0 pada *meetPoint* adalah representasi dari *pixel* hitam atau dinding, nilai 3 merepresentasikan *vertex*, serta nilai 1 merepresentasikan *edge* yang terdapat antara *vertex*.

Tabel 4.2 Representasi matrik *edge* dan *vertex* labirin

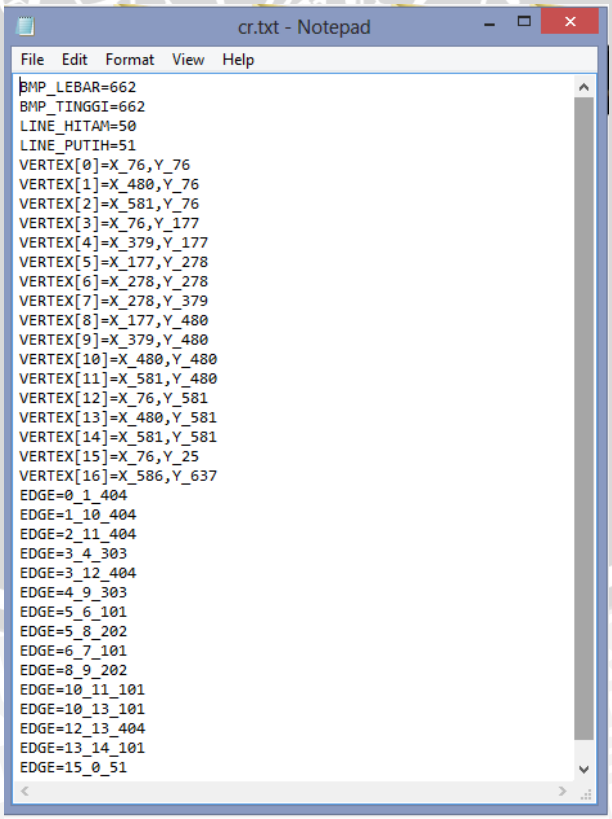
VERTEK	V0	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
V0	0	404	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	51	-1
V1	404	0	-1	-1	-1	-1	-1	-1	-1	-1	404	-1	-1	-1	-1	-1	-1
V2	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	404	-1	-1	-1	-1	-1
V3	-1	-1	-1	0	303	-1	-1	-1	-1	-1	-1	-1	404	-1	-1	-1	-1
V4	-1	-1	-1	303	0	-1	-1	-1	-1	303	-1	-1	-1	-1	-1	-1	-1
V5	-1	-1	-1	-1	-1	0	101	-1	202	-1	-1	-1	-1	-1	-1	-1	-1
V6	-1	-1	-1	-1	-1	101	0	101	-1	-1	-1	-1	-1	-1	-1	-1	-1
V7	-1	-1	-1	-1	-1	-1	101	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
V8	-1	-1	-1	-1	-1	202	-1	-1	0	202	-1	-1	-1	-1	-1	-1	-1
V9	-1	-1	-1	-1	303	-1	-1	-1	202	0	-1	-1	-1	-1	-1	-1	-1
V10	-1	404	-1	-1	-1	-1	-1	-1	-1	-1	0	101	-1	101	-1	-1	-1
V11	-1	-1	404	-1	-1	-1	-1	-1	-1	-1	101	0	-1	-1	-1	-1	-1
V12	-1	-1	-1	404	-1	-1	-1	-1	-1	-1	-1	-1	0	404	-1	-1	-1
V13	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	404	0	101	-1	-1
V14	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	101	-1	-1	101	0	-1	-1
V15	51	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1
V16	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0

Tabel 4.2 merupakan representasi dari *undirected graph weighted edge* hasil pengolahan citra labirin. Dalam tabel tersebut merepresentasikan relasi antar *vertex* beserta dengan nilai *edge* yang terdapat pada *vertex* yang berhubungan. Nilai -1 (negatif) merepresentasikan bahwa *vertex* tersebut tidak terhubung dengan *vertex* terdekatnya, jika tedapat nilai positif maka angka tersebut merepresentasikan nilai *edge* yang menghubungkan *vertex* tersebut. Dalam hal ini nilai *edge* antar *vertex* dalam satuan *pixel*. Nilai 0 merepresentasikan relasi *vertex* tersebut dengan dirinya sendiri, jadi pada dasarnya suatu *vertex* dengan dirinya sendiri terhubung dengan nilai 0 (nol).

Info dari citra labrin, *edge* dan *vertex* diubah kedalam bentuk *string* yang dapat disimpan dalam bentuk *plaintext*.

```
myGraph.GetVertex(i).Value.X.ToString(),
myGraph.GetVertex(i).Value.Y.ToString();
```

Hasil *output* program yang berupa *plaintext* seperti ditunjukan oleh gambar 4.16.



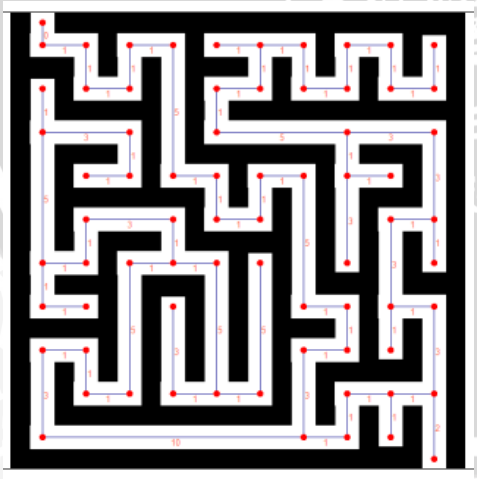
Gambar 4.16 Hasil *output* berupa *plaintext*

Dari gambar 4.17 dapat dilihat hasil output program yang didalamnya terdapat informasi tentang besar citra labirin, lebar dinding dan lorong labirin, letak node serta panjang edge yang menghubungkan antar node.

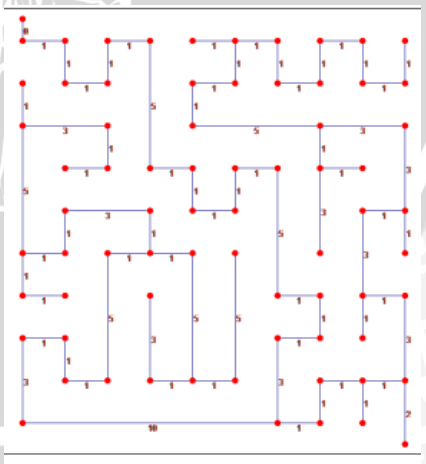
4.2.10 Proses *Load Output Plaintext*

Dari hasil proses program yang berupa *graph* dan di simpan ke dalam *format plaintext*, maka program yang dibuat harus dapat memproses hasil *output* tersebut dan mevisualisasikan kedalam bentuk citra labirin dan kedalam bentuk *graph*. Proses dari *load* hasil *output* berupa *plaintext* tersebut hampir sama seperti proses visualisasi graph ke dalam citra, yaitu proses penggambaran ulang. Untuk dapat merubah info labirin yang berupa *plaintext* kedalam bentuk citra dan *graph* secara sempurna.

Untuk proses load dimulai dengan mengambil nilai *width* dan *height* citra setelah itu baru mengambil nilai lebar dinding dan lorong. Setelah selesai menggambar dinding dan lorong baru mengambil nilai *vertex* dan *edge*nya untuk divisualisasikan kedalam citra labirin. Hasil proses *load plaintext* ke dalam program seperti ditunjukkan pada gambar 4.18 dan 4.19.

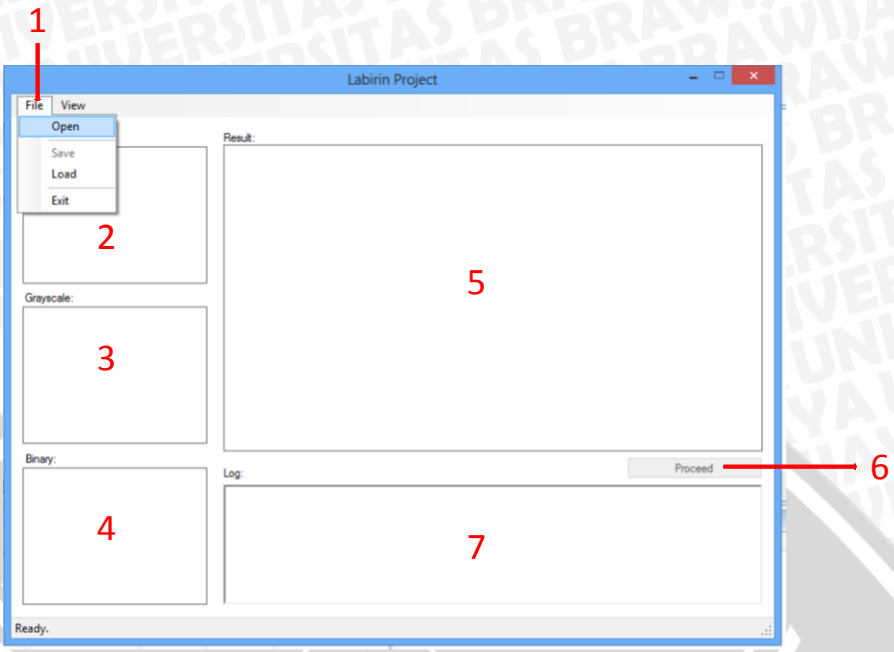


Gambar 4.17 Hasil *load* berupa citra

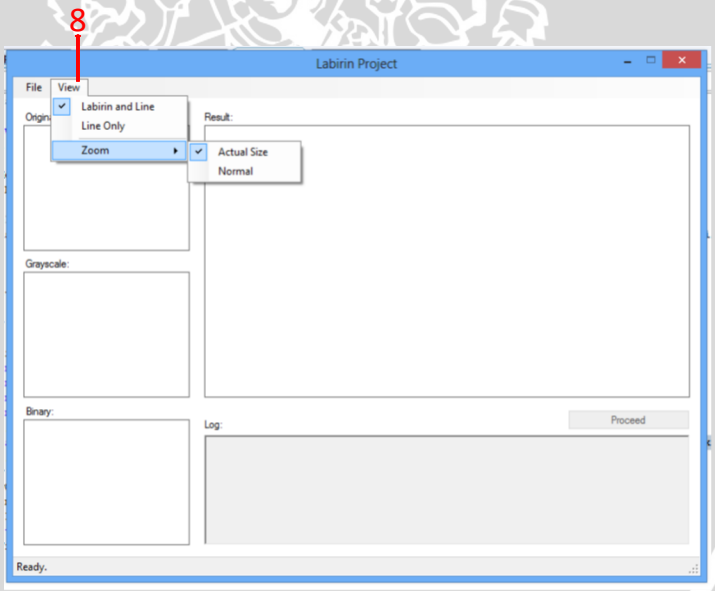


Gambar 4.18 Hasil *load* berupa *graph*

4.3 Perancangan *User Interface*



Gambar 4.19 Desain *user interface* program



Gambar 4.20 Desain *user interface* program

Design aplikasi ini terdiri dari 1 jendela, yang berisi beberapa *menu* untuk mendukung kerja dari aplikasi ini, diantaranya :

1. *Menu strip file* yang berisi *tool* seperti :
   - *Open* : untuk me*load* citra labirin ke dalam program.
   - *Save* : untuk melakukan penyimpanan hasil program, berupa *plaintext*.

- *Load* : untuk melakukan *load graf* ke dalam aplikasi, dengan ekstensi *input*an berupa *plaintext* (*.txt*).
- *Exit* : digunakan untuk keluar dari program.

2. *Picture box* citra labirin asli yang menampilkan citra labirin asli yang akan diproses.
3. *Picture box* citra *graysacle* yang menampilkan citra labirin yang telah melalui proses *grayscalling*.
4. *Picture box* citra *binary* yang menampilkan citra labirin yang telah melalui proses *binary*.
5. *Picture box* citra hasil (*result*) yang menampilkan citra labirin yang telah di proses.
6. *Button* proses yang memiliki fungsi sebagai tanda proses pengolahan citra dimulai.
7. *Rich text box* yang memiliki fungsi menampilkan *list data* dari hasil pengolahan citra oleh algoritma yang dibuat.
8. *Menu strip view* yang berisi *tool* seperti :
    - *Labirin* and *line* : untuk menampilkan hasil berupa citra labirin dengan atribut *node*, *edge* serta jarak.
    - *Line* : untuk menampilkan bentuk *graph* hasil pengolahan dengan atribut *node*, *edge* serta jarak.
    - *Zoom Actual View* : untuk menampilkan gambar lebih detail (*zoom*).
    - *Zoom Normal* : untuk menampilkan hasil *output* citra normal.