Connected Component Labelling

I have asked a similar question some days ago, but I have yet to find an efficient way of solving my problem. I'm developing a simple console game, and I have a 2D array like this:

```
1,0,0,0,1
1,1,0,1,1
0,1,0,0,1
1,1,1,1,0
0,0,0,1,0
```

I am trying to find all the areas that consist of neighboring 1's (4-way connectivity). So, in this example the 2 areas are as following:

```
1
1,1
  1
1,1,1,1
      1
```

and :

```
      1
    1,1
      1
```

The algorithm, that I've been working on, finds all the neighbors of the neighbors of a cell and works perfectly fine on this kind of matrices. However, when I use bigger arrays (like 90*90) the program is very slow and sometimes the huge arrays that are used cause stack overflows.

One guy on my other question told me about connected-component labelling as an efficient solution to my problem.

Can somebody show me any C++ code which uses this algorithm, because I'm kinda confused about how it actually works along with this disjoint-set data structure thing...

Thanks a lot for your help and time.

algorithm multidimensional-array area neighbours

edited Jan 22 '13 at 19:06
svick
103k 18 156 253

asked Jan 22 '13 at 18:14
user1981497
78 2 8

4 Answers

I'll first give you the code and then explain it a bit:

```
// direction vectors
const int dx[] = {+1, 0, -1, 0};
const int dy[] = {0, +1, 0, -1};

// matrix dimensions
int row_count;
int col_count;

// the input matrix
int m[MAX][MAX];

// the labels, 0 means unlabeled
int label[MAX][MAX];

void dfs(int x, int y, int current_label) {
  if (x < 0 || x == row_count) return; // out of bounds
  if (y < 0 || y == col_count) return; // out of bounds
  if (label[x][y] || !m[x][y]) return; // already labeled or not marked with 1 in m

  // mark the current cell
  label[x][y] = current_label;

  // recursively mark the neighbors
  for (int direction = 0; direction < 4; ++direction)
    dfs(x + dx[direction], y + dy[direction], current_label);
}
```

```
void find_components() {
  int component = 0;
  for (int i = 0; i < row_count; ++i)
    for (int j = 0; j < col_count; ++j)
      if (!label[i][j] && m[i][j]) dfs(i, j, ++component);
}
```

This is a common way of solving this problem.

The direction vectors are just a nice way to find the neighboring cells (in each of the four directions).

The **dfs** function performs a *depth-first-search* of the grid. That simply means it will visit all the cells reachable from the starting cell. Each cell will be marked with **current_label**

The **find_components** function goes through all the cells of the grid and starts a component labeling if it finds an unlabeled cell (marked with 1).

This can also be done iteratively using a stack. If you replace the stack with a queue, you obtain the *bfs* or *breadth-first-search*.

answered Jan 22 '13 at 19:05

gus

409 3 5

Union find seem like what you want.

Code:

```
int main()
{
  ...
  bool input[w][h];
  // set up input
  int component[w*h];
  for (int i = 0; i < w*h; i++)
    component[i] = i;
  for (int x = 0; x < w; x++)
  for (int y = 0; y < h; y++)
  {
    unionCoords(x, y, x+1, y);
    unionCoords(x, y, x, y+1);
  }
  ...
}

void unionCoords(int x, int y, int x2, int y2)
{
  if (y2 < h && x2 < w && input[x][y] && input[x2][y2])
    union(x*h + y, x2*h + y2);
}

void union(int a, int b)
{
  // get the root component of a and b, and set the one's parent to the other
  while (component[a] != a)
    a = component[a];
  while (component[b] != b)
    b = component[b];
  component[b] = a;
}
```

This will result in each group of 1's being in a group. Code to print grid with components: (you'll notice that each 0 is in its own group)

```
for (int y = 0; y < h; y++)
{
  for (int x = 0; x < w; x++)
  {
     int c = x*h + y;
     while (component[c] != c) c = component[c];
     cout << (char)('a'+c);
  }
  cout << "\n";
}
```

Using code similar to the above, it's easy to extract the groups.

There are various optimisations to improve the efficiency of union find, the above is just a basic implementation.

edited Jan 22 '13 at 20:22

answered Jan 22 '13 at 19:07

Dukeling

34.3k 8 33 71

Thanks a lot for this. It looks that it is exactly what I was trying to find! – user1981497 Jan 22 '13 at 19:58

One question : Isn't the int component[*wh]; part supposed to be : int component [*wy]; ? – user1981497 Jan 22 '13 at 20:18

@user1981497 Actually `bool input[w][y];` was supposed to be `bool input[w][h];` (fixed it). `w` is width and `h` is height. – Dukeling Jan 22 '13 at 20:25

Thank you very much... I'm testing your code right now...! – user1981497 Jan 22 '13 at 20:34

Why can't I use a secondary 2d array with the labels instead of the union find structure? – user1981497 Jan 23 '13 at 8:10

You could also try this transitive closure approach, however the triple loop for the transitive closure slows things up when there are many separated objects in the image, suggested code changes welcome

Cheers

Dave

```
void CC(unsigned char* pBinImage, unsigned char* pOutImage, int width, int height, int
CON8)
{
int i, j, x, y, k, maxIndX, maxIndY,  sum, ct, newLabel=1, count, maxVal=0, sumVal=0,
maxEQ=10000;
int *eq=NULL, list[4];
int bAdd;

memcpy(pOutImage, pBinImage, width*height*sizeof(unsigned char));

unsigned char* equivalences=(unsigned char*) calloc(sizeof(unsigned char), maxEQ*maxEQ);

// modify labels this should be done with iterators to modify elements
// current column
for(j=0; j<height; j++)
{
    // current row
    for(i=0; i<width; i++)
    {
        if(pOutImage[i+j*width]>0)
        {
            count=0;

            // go through blocks
            list[0]=0;
            list[1]=0;
            list[2]=0;
            list[3]=0;

            if(j>0)
            {
                if((i>0))
                {
                    if((pOutImage[(i-1)+(j-1)*width]>0) && (CON8 > 0))
                        list[count++]=pOutImage[(i-1)+(j-1)*width];
                }

                if(pOutImage[i+(j-1)*width]>0)
                {
                    for(x=0, bAdd=true; x<count; x++)
                    {
                        if(pOutImage[i+(j-1)*width]==list[x])
                            bAdd=false;
                    }

                    if(bAdd)
                        list[count++]=pOutImage[i+(j-1)*width];
                }

                if(i<width-1)
                {
                    if((pOutImage[(i+1)+(j-1)*width]>0) && (CON8 > 0))
                    {
                        for(x=0, bAdd=true; x<count; x++)
                        {
                            if(pOutImage[(i+1)+(j-1)*width]==list[x])
                                bAdd=false;
                        }

                        if(bAdd)
                            list[count++]=pOutImage[(i+1)+(j-1)*width];
                    }
                }
            }

            if(i>0)
            {
                if(pOutImage[(i-1)+j*width]>0)
                {
                    for(x=0, bAdd=true; x<count; x++)
                    {
                        if(pOutImage[(i-1)+j*width]==list[x])
                            bAdd=false;
                    }

                    if(bAdd)
```

```
                    list[count++]=pOutImage[(i-1)+j*width];
                }
            }

            // has a neighbour label
            if(count==0)
                pOutImage[i+j*width]=newLabel++;
            else
            {
                pOutImage[i+j*width]=list[0];

                if(count>1)
                {
                    // store equivalences in table
                    for(x=0; x<count; x++)
                        for(y=0; y<count; y++)
                            equivalences[list[x]+list[y]*maxEQ]=1;
                }

            }
        }
    }
}

 // floyd-Warshall algorithm - transitive closure - slow though :-(
 for(i=0; i<newLabel; i++)
    for(j=0; j<newLabel; j++)
    {
        if(equivalences[i+j*maxEQ]>0)
        {
            for(k=0; k<newLabel; k++)
            {
                equivalences[k+j*maxEQ]= equivalences[k+j*maxEQ] ||
equivalences[k+i*maxEQ];
            }
        }
    }


eq=(int*) calloc(sizeof(int), newLabel);

for(i=0; i<newLabel; i++)
    for(j=0; j<newLabel; j++)
    {
        if(equivalences[i+j*maxEQ]>0)
        {
            eq[i]=j;
            break;
        }
    }


free(equivalences);

// label image with equivalents
for(i=0; i<width*height; i++)
{
    if(pOutImage[i]>0&&eq[pOutImage[i]]>0)
        pOutImage[i]=eq[pOutImage[i]];
}

free(eq);
}
```

edited May 3 '13 at 11:10

answered May 3 '13 at 11:05

ejectamenta

198 1 11

---

very useful Document =>
https://docs.google.com/file/d/0B8gQ5d6E54ZDM204VFVxMkNtYjg/edit

java application - open source - extract objects from image - connected componen labeling =>
https://drive.google.com/file/d/0B8gQ5d6E54ZDTVdsWE1ic2lpaHM/edit?usp=sharing

```
    import java.util.ArrayList;

public class cclabeling

{

 int neighbourindex;ArrayList<Integer> Temp;

 ArrayList<ArrayList<Integer>> cc=new ArrayList<>();

 public int[][][] cclabel(boolean[] Main,int w){

 /* this method return array of arrays "xycc" each array contains

 the x,y coordinates of pixels of one connected component

 - Main => binary array of image

 - w => width of image */
```

```
long start=System.nanoTime();
int len=Main.length;int id=0;
int[] dir={-w-1,-w,-w+1,-1,+1,+w-1,+w,+w+1};
for(int i=0;i<len;i+=1){
if(Main[i]){
Temp=new ArrayList<>();
Temp.add(i);
for(int x=0;x<Temp.size();x+=1){
id=Temp.get(x);
for(int u=0;u<8;u+=1){
neighbourindex=id+dir[u];
 if(Main[neighbourindex]){
 Temp.add(neighbourindex);
 Main[neighbourindex]=false;
 }
 }
Main[id]=false;
}
cc.add(Temp);
    }
}
int[][][] xycc=new int[cc.size()][][];
int x;int y;
for(int i=0;i<cc.size();i+=1){
 xycc[i]=new int[cc.get(i).size()][2];

 for(int v=0;v<cc.get(i).size();v+=1){
 y=Math.round(cc.get(i).get(v)/w);
 x=cc.get(i).get(v)-y*w;
 xycc[i][v][0]=x;
 xycc[i][v][1]=y;
 }

}
long end=System.nanoTime();
long time=end-start;
System.out.println("Connected Component Labeling Time =>"+time/1000000+" milliseconds");
System.out.println("Number Of Shapes => "+xycc.length);
 return xycc;

 }
}
```

edited Jul 2 '14 at 18:45

answered Jul 2 '14 at 18:31

ostaz

1 1