



# FINAL PROJECT REPORTS

## d a t a s t r u c t u r e d



Nuzha Musyafira | 5116100014

# TOPICS:: Number Maze

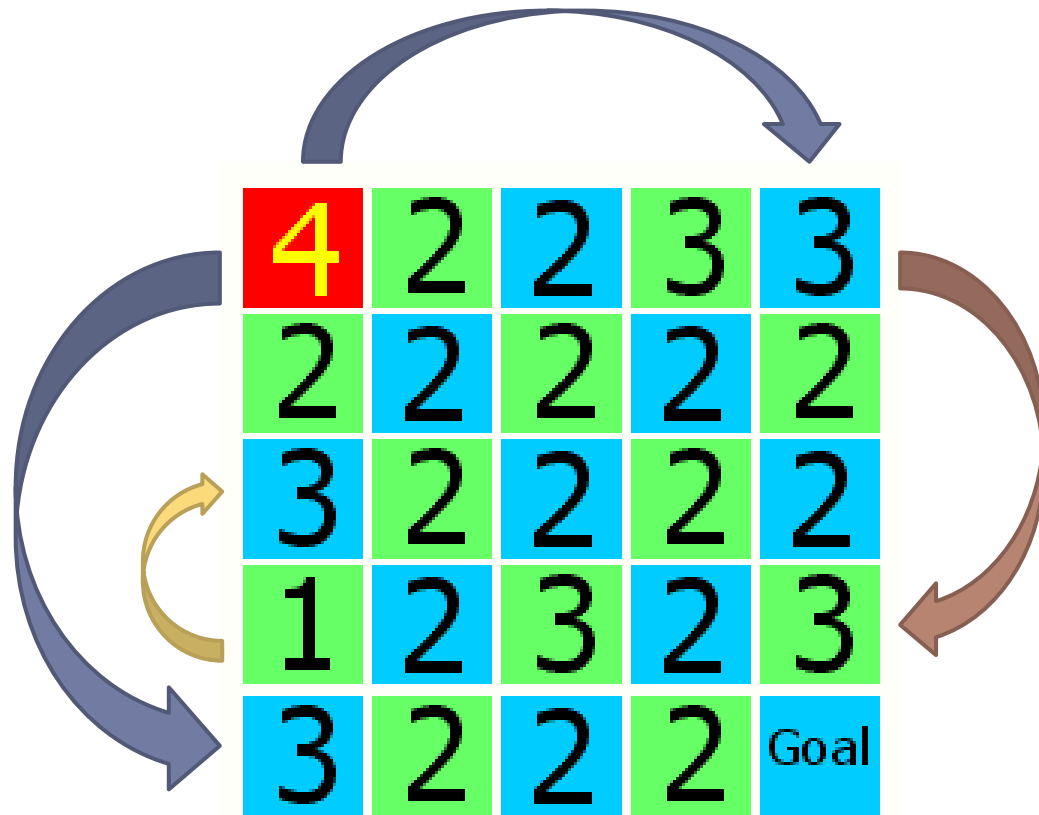
---

- ▶ This is a “Number Maze,” and it is interactive. Here is the rule :
  - ▶ You begin on the square at the upper left (as in this example is shown in red). You have to make a series of moves that will take you to the square marked **Goal**. The number in each square indicates how far you must move—horizontally or vertically—when you leave the square.
  - ▶ The purpose of the program being created will show the sequence of which numbers we need to visit to reach the **Goal**.
- ▶ note: the **Goal** mark will be shown as **0** in program.
- ▶ The topics were taken because it could represent the concept of directed graph.



## ::Solutions Sample

---



- ▶ 4 -down- 3 -up- 2-right- 2 -down- 3 -up- 2 -down- 2 -down- 2 -right- 0
- ▶ 4 -down- 3 -up- 2-right- 2 -down- 3 -up- 2 -down- 2 -right- 2 -down- 0

# ::Sourcecode

---

- ▶ There are two sourcecodes which can be implemented:
  - ▶ The first one is in C, it uses array to generate the properties of graph
  - ▶ The second one is in C++, it uses class and list library to construct the graph



# ::in C

```
1  #include<stdio.h>
2  int c,b;
3  int mz[100];
4  char arah[100][100];
5  struct gr{
6      int a;
7      int list[100];
8  };
9  struct gr adj[100];
10 void addEdge(int u, int v){
11     adj[u].list[adj[u].a]=v;
12     adj[u].a++;
13 }
```

`int c,b` nantinya berperan sebagai size dari maze itu sendiri. Variabel ini diset *global* karena nantinya akan banyak dibutuhkan di luar `main()`.

`int mz[100]` nantinya adalah wadah untuk menyimpan nilai path sebenarnya, karena pada saat proses *finding path*, yang digunakan adalah indeks dari nilai untuk mencegah adanya nilai antar vertex yang sama. Diset *global* karena banyak digunakan di luar `main()`.

`char arah[x][y]` digunakan untuk menyimpan arah dari vertex `x` ke vertex `y`.

`struct gr` merupakan *struct*, fungsinya nanti yaitu menampung informasi antara lain koneksi dari antar vertex (yang disimpan di `list[]`), dan jumlah koneksi yang dimiliki tiap vertex (`a`).

`struct gr adj[]` merupakan type data struct dengan nama `adj[]`, `adj[]` mewakili adjacency list dari vertex indeks ke sekian (`adj[vertex]`).

`void addEdge()` adalah *function* untuk menambahkan edge, dengan `int u` sebagai head dan directed ke `int v`. `adj[u].list[adj[u].a]=v` berarti tambahkan `v` ke list ke `a` milik `u`, `a` yang otomatis dimulai dari `0` akan bertambah seiring bertambahnya koneksi vertex `u`.

```

14 void printall(int u, int d, int *vis, int *pth, int ind){
15     vis[u]=1;
16     pth[ind]=u;
17     ind++;
18     int i;
19     if(u==d){
20         for(i=0;i<ind-1;i++){
21             printf("%d ", mz[pth[i]]);
22             if(arah[pth[i]][pth[i+1]]=='u') printf("-up- ");
23             if(arah[pth[i]][pth[i+1]]=='d') printf("-down- ");
24             if(arah[pth[i]][pth[i+1]]=='l') printf("-left- ");
25             if(arah[pth[i]][pth[i+1]]=='r') printf("-right- ");
26         }
27         printf("%d\n\n", mz[pth[ind-1]]);
28     }

```

`void printall()` merupakan *function* rekursif untuk menampilkan semua kemungkinan path yang ada dari vertex indeks ke `u` ke `d`. `int *vis` dan `int *pth` menggunakan asterisk karena akan bertindak sebagai array nantinya. Mula-mula `vis[u]` (source) diisi dengan `1`, yang menandakan telah *visited*, sehingga tidak perlu diakses lagi ke selanjutnya. Kemudian, `pth[]` yang berfungsi menyimpan indeks-indeks yang berkemungkinan menjadi path diisi dengan `u` sebagai nilai pertamanya, karena path selalu dimulai dengan source. `ind` bertambah ketika vertex yang dilalui bertambah. Jika `u==d`, yang berarti telah dicapai vertex tujuan, maka path akan diprint. Untuk memudahkan pembacaan output, maka setiap vertex yang berhubungan pasti memiliki arah. Pada `main()`, setiap arah vertex dinotasikan dengan `d, u, r, l`. Looping hanya dilakukan hingga `<ind-1` karena ketika di destination, tidak membutuhkan keterangan arah ke mana lagi. Path yang di print bukan nilai yang tersimpan di index, melainkan nilai asli, sehingga yang ditampilkan adalah `mz[pth[i]]`.

```

29 }
30 }
31     for(i=0;i<adj[u].a;i++){
32         if(vis[adj[u].list[i]]!=1) printall(adj[u].list[i],d,vis,pth,ind);
33     }
34     ind--;
35     vis[u]=0;
36 }

```

Kelanjutan dari *branch* pada *if* sebelumnya, jika *u* tidak sama dengan *d*, maka *else* dari *function* akan move ke seluruh koneksi yang dimiliki vertex *u* yang belum visited dan mencoba semua kemungkinan yang ada dari vertex-vertex koneksi dari koneksi lainnya. Apabila *u==d*, maka *function* akan print path, kemudian *return* ke proses sebelumnya. *ind--* dilakukan setelah proses karena ketika beralih ke vertex lain, vertex sebelumnya dianggap dikeluarkan dari *pth[]*. *vis[u]=0* berarti vertex yang sebelumnya ditandai sekarang menjadi not visited, karena berkemungkinan untuk bisa dikunjungi dari sisi vertex yang lain untuk kemungkinan path lainnya.

```

37 void print(int s, int d){
38     int visited[100];
39     int path[100];
40     int index=0;
41     int i;
42     for(i=0;i<c*b;i++) visited[i]=0;
43     printall(s,d,visited,path,index);
44 }

```

Pada umumnya, *function* ini merupakan step untuk *declaration* untuk kebutuhan di *function printall()*. *int s* merepresentasikan index source, *int d* destination. *int visited[]* untuk menandai apakah indeks ke sekian telah diakses/dilewati atau tidak, menggunakan *type data int*, 1 sebagai *true* dan 0 sebagai *false* karena pada C tidak dapat menggunakan *bool*. *int path[]* menyimpan vertex yang telah dilalui yang berhasil terangkai membentuk path. *int index=0* sebagai bentuk inisialisasi untuk *function printall()* nantinya. Pada *function* ini, semua indeks diset sebagai not visited (0), karena kondisi mula-mula. Setelah semua inisialisasi selesai, barulah *function* ini memanggil *function printall()* yang telah dijabarkan sebelumnya.

```

45 int main(){
46     int i,j;
47     printf("Enter the size: ");
48     scanf("%d %d", &c,&b);
49     printf("\n");
50     for(i=0;i<c;i++){
51         for(j=0;j<b;j++) scanf("%d", &mz[i*c+j]);
52     }

```

`int i,j` melambangkan indeks untuk *looping*.

`c` dan `b` adalah ukuran maze, `c` melambangkan panjang dan `b` adalah lebar.

*Looping* dilakukan sebanyak  $c*b$  kali untuk menginputkan data atau vertex pada maze. Indeks pada `mz[]` adalah  $i*c+j$  karena mewakili vertex urut dari pojok kiri atas yang dimulai dari 0 dan diakhiri di pojok kanan bawah oleh  $c*b-1$ .



```

53 for(i=0;i<c;i++){
54     for(j=0;j<b;j++){
55         int p=mz[i*c+j];
56         if(p!=0){
57             if(i+p<c){
58                 addEdge(i*c+j,(i+p)*c+j);
59                 arah[i*c+j][(i+p)*c+j]='d';
60             } //bawah
61             if(i-p>=0){
62                 addEdge(i*c+j,(i-p)*c+j);
63                 arah[i*c+j][(i-p)*c+j]='u';
64             } //atas
65             if(j+p<b){
66                 addEdge(i*c+j,i*c+(j+p));
67                 arah[i*c+j][i*c+(j+p)]= 'r';
68             } //kanan
69             if(j-p>=0){
70                 addEdge(i*c+j,i*c+(j-p));
71                 arah[i*c+j][i*c+(j-p)]= 'l';
72             } //kiri
73         }
74     }
75 }

```

Looping ini dimaksudkan untuk menyimpan informasi-informasi indeks vertex yang saling berhubungan dan arahnya. **int p** adalah nilai asli dari vertex, berfungsi menentukan vertex mana yang dapat terhubung, sebab, terhubung-tidaknya 2 vertex ditentukan dari apakah keduanya saling berjarak sebesar **p**, entah vertikal ataupun horizontal. Jika **p=0**, tidak perlu dilakukan apa-apa karena **0** tidak dapat terhubung ke manapun. Sebelum menentukan koneksi, perlu dicek juga apakah **i+p<c** untuk arah bawah, **i-p>=0** untuk atas, **j+p<b** untuk arah kanan, dan **j-p>=0** untuk ke kiri. Hal tersebut dilakukan untuk menghindari *bugs*, misal vertex indeks ke **0** bernilai **4**, jika tidak dilakukan pengecekan **j-p>=0**, maka **4** akan terhubung ke vertex ke-**4** di sebelah kirinya yaitu *random memory* atau memicu *error*. Arah antar vertex tidak perlu diassign dengan *string* karena **1 char** saja sudah cukup sebab inisial tiap *direction* sudah saling berbeda.

```
76  
77  
78  
79 }
```

```
printf("\n");  
print(0, c*b-1);  
return 0;
```

Pemanggilan *function* `print()`,  
dengan `0` sebagai *source*, karena  
dimulai dari pojok kiri atas  
(indeks=`0`), dan `c*b-1` sebagai  
*destination* karena diakhiri di  
pojok kanan bawah.

# ::in C++

```
1  #include<iostream>
2  #include<list>
3  using namespace std;
4  int mz[100];
5  char arah[100][100];
6  class Graph{
7      public:
8          int V; //jumlah vertex
9          list<int> *adj; //pointer ke array list
10         void printAllPathsUtil(int, int, bool [], int [], int); //fungsi rekursif untuk printallpath
11         Graph(int V); //konstruktor
12         void addEdge(int u, int v);
13         void printAllPaths(int s, int d);
14     };
15     Graph::Graph(int V){
16         this->V=V;
17         adj=new list<int>[V];
18     }
19     void Graph::addEdge(int u, int v){
20         adj[u].push_back(v); //tambahkan v ke list u
21     }
22     void Graph::printAllPaths(int s, int d){ //print semua path dari source ke dest
23         bool *visited=new bool[V]; //tanda semua vertex sbg blm visited mula2
24         int *path=new int[V]; //membentuk array utk menyimpan path
25         int path_index=0; //inisialisasi path[] dengan kosong
26         for(int i=0;i<V;i++) visited[i]=false; //inisialisasi semua vertex blm visited
27         printAllPathsUtil(s,d,visited,path,path_index); //memanggil fungsi rekursif untuk print semua path
28     }
29     void Graph::printAllPathsUtil(int u, int d, bool visited[], int path[], int path_index){
30         visited[u]=true;
```

```

31 path[path_index]=u;
32 path_index++;
33 if(u==d){
34     for(int i=0;i<path_index-1;i++){
35         printf("%d ", mz[path[i]]);
36         if(arah[path[i]][path[i+1]]=='u') printf("-up- ");
37         if(arah[path[i]][path[i+1]]=='d') printf("-down- ");
38         if(arah[path[i]][path[i+1]]=='l') printf("-left- ");
39         if(arah[path[i]][path[i+1]]=='r') printf("-right- ");
40     }
41     printf("%d\\n\\n", mz[path[path_index-1]]);
42 }
43 else{
44     list<int>::iterator i;
45     for(i=adj[u].begin();i!=adj[u].end();i++){
46         if(!visited[*i]) printAllPathsUtil(*i,d,visited,path,path_index);
47     }
48 }
49 path_index--;
50 visited[u]=false;
51 }
52 int main(){
53     int a,b,i,j;
54     printf("Enter the size: ");
55     scanf("%d %d", &a,&b);
56     printf("\\n");
57     Graph g(a*b);
58     for(i=0;i<a;i++){
59         for(j=0;j<b;j++){
60             scanf("%d", &mz[i*a+j]);

```

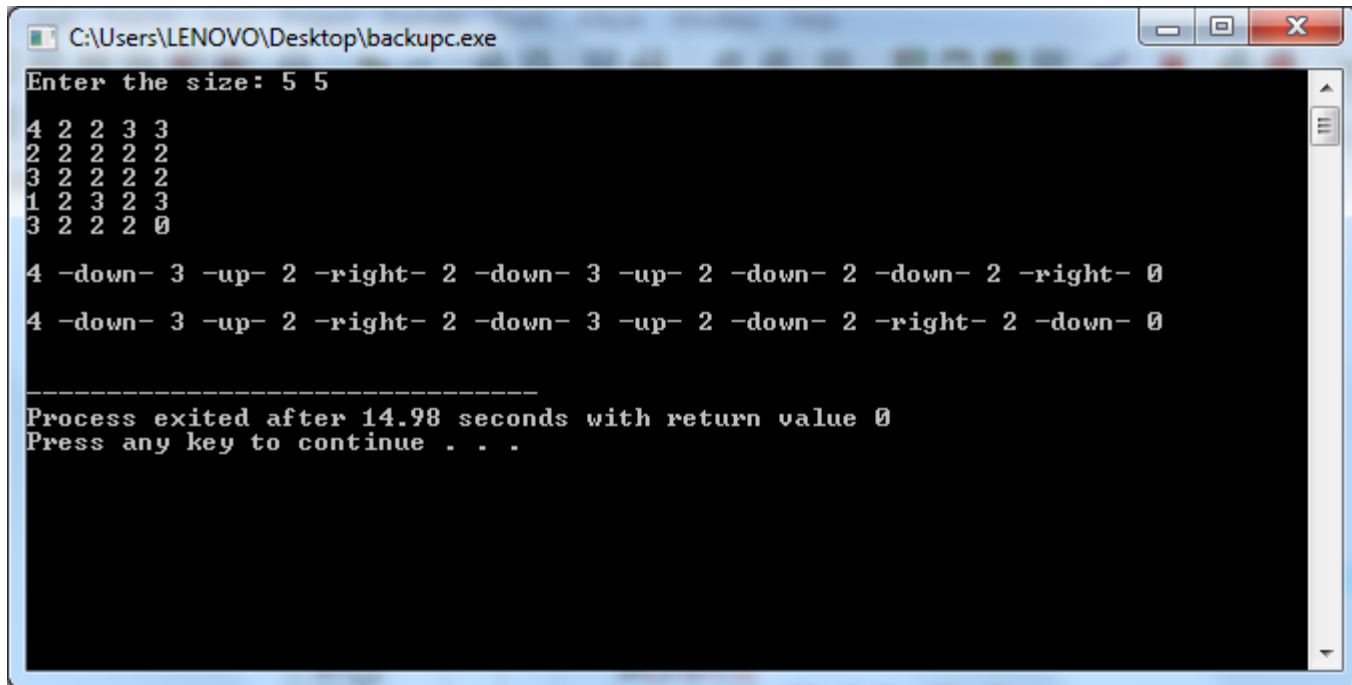
```

61     }
62 }
63 for(i=0;i<a;i++){
64     for(j=0;j<b;j++){
65         int p=mz[i*a+j];
66         if(p!=0){
67             if(i+p<a){
68                 g.addEdge(i*a+j,(i+p)*a+j);
69                 arah[i*a+j][(i+p)*a+j]='d';
70             } //bawah
71             if(i-p>=0){
72                 g.addEdge(i*a+j,(i-p)*a+j);
73                 arah[i*a+j][(i-p)*a+j]='u';
74             } //atas
75             if(j+p<b){
76                 g.addEdge(i*a+j,i*a+(j+p));
77                 arah[i*a+j][i*a+(j+p)]= 'r';
78             } //kanan
79             if(j-p>=0){
80                 g.addEdge(i*a+j,i*a+(j-p));
81                 arah[i*a+j][i*a+(j-p)]= 'l';
82             } //kiri
83         }
84     }
85 }
86 printf("\n");
87 g.printAllPaths(0,a*b-1);
88 }

```

# ::Run in C

---



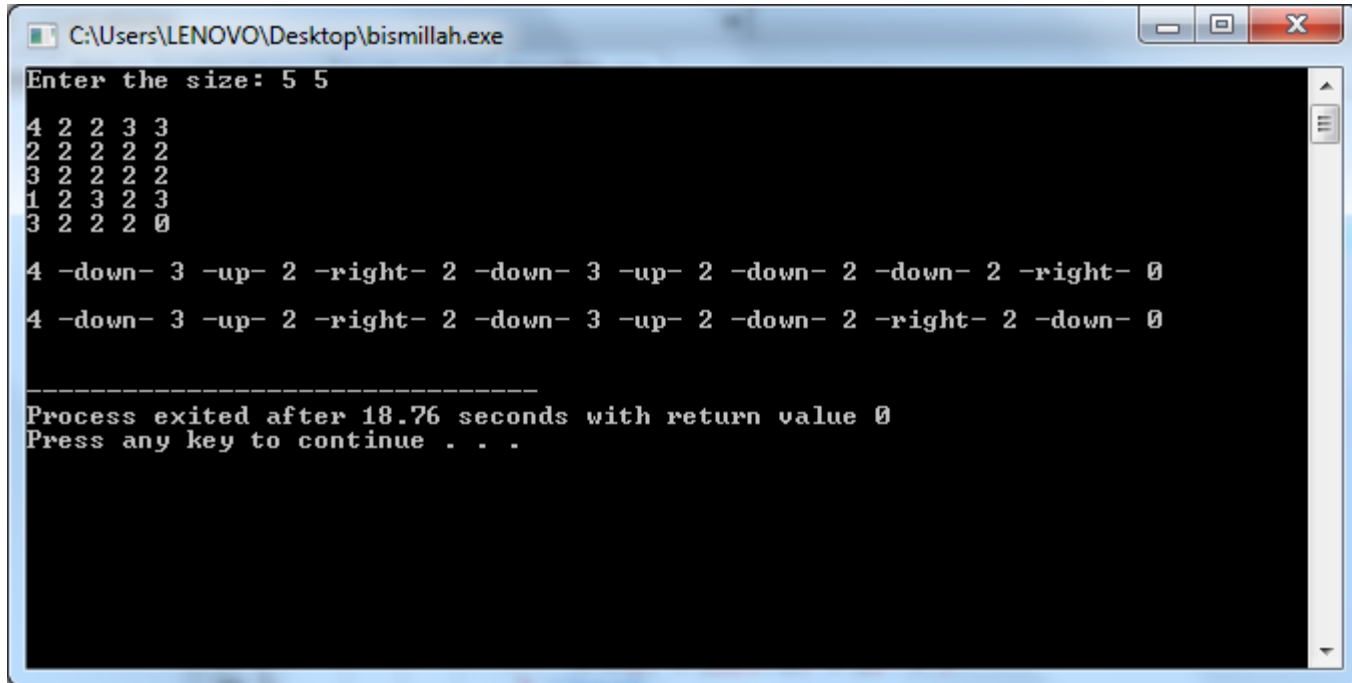
```
C:\Users\LENOVO\Desktop\backupc.exe
Enter the size: 5 5
4 2 2 3 3
2 2 2 2 2
3 2 2 2 2
1 2 3 2 3
3 2 2 2 0

4 -down- 3 -up- 2 -right- 2 -down- 3 -up- 2 -down- 2 -down- 2 -right- 0
4 -down- 3 -up- 2 -right- 2 -down- 3 -up- 2 -down- 2 -right- 2 -down- 0

-----
Process exited after 14.98 seconds with return value 0
Press any key to continue . . .
```

# ::Run in C++

---



```
C:\Users\LENOVO\Desktop\bismillah.exe
Enter the size: 5 5
4 2 2 3 3
2 2 2 2 2
3 2 2 2 2
1 2 3 2 3
3 2 2 2 0

4 -down- 3 -up- 2 -right- 2 -down- 3 -up- 2 -down- 2 -down- 2 -right- 0
4 -down- 3 -up- 2 -right- 2 -down- 3 -up- 2 -down- 2 -right- 2 -down- 0

-----
Process exited after 18.76 seconds with return value 0
Press any key to continue . . .
```