Info

Name:Ziyi Guo
Student number:520137

Running time

The basic A* expands 507 nodes until it finds the goal nodes
The priority one expands 111 nodes until it reaches goal.
The time that the same algorithm differs time to time, which I consider has a lot to do with the HashMap structure and how it store the node. During searching for the least cost node, it will expand the smallest f-cost node, which there always are ties, and then we cannot decide which one will be expanded first. That's my guess about it.

The out put here is like this.

```
The Naive one is like this:
The shortest path is 26 steps and took 507 recursions
[0,0]S
[1,0]E
[1,1]S
[2,1]E
[2,2]S
[3,2]E
[3,3]E
[3,4]E
[3,5]E
[3,6]E
[3,7]E
[3,8]E
[3,9]E
[3,10]S
[4,10]S
[5,10]E
[5,11]S
[6,11]S
[7,11]W
[7,10]W
[7,9]S
[8,9]S
[9,9]E
[9,10]E
[9,11]S
[10,11]S
[11,11]
```

The codes are in three file—Astar.java, DisPlay.java, Search.java.

Astar.java define the class the run the astar search, the naive one and the DP one,with a variable 'method' which is used to construct

the class to determine which method to use.
The function is defined as 'run', which will run the astar algorithm recursively, to find the best route.

```
	public static void run(HashMap<String, Integer> on,HashMap<String, Integer>
visited,List<String> route,int[] dest)
	{
		count ++ ;
		int[] now = {-1,-1};
		int dis = -1;

		int min = Collections.min(on.values());
		Iterator<?> iterator = on.entrySet().iterator();
		while(iterator.hasNext())// find the node unexpanded with minimum f
cost
		{
			Entry<?, ?> entry = (Entry<?, ?>) iterator.next();
			if(entry.getValue().equals(min))
			{
				String s = entry.getKey().toString();
				int start = s.lastIndexOf("[");
				int mid = s.lastIndexOf(",");
				int end = s.lastIndexOf("]");
				now[0] = Integer.parseInt(s.substring(start+1, mid));
				now[1] = Integer.parseInt(s.substring(mid+1, end));

				dis = min - (int) Math.sqrt(Math.pow(dest[0]-now[0],
2)+Math.pow(dest[1]-now[1], 2));
				on.remove(entry.getKey());
				if(method == 2)
					visited.put(s.substring(start), dis);
				route.add(s);
				break;
			}
		}

		if(now[0]==dest[0] && now[1]==dest[1])//if reached the goal
		{
			System.out.println("The shortest path is " + dis+" steps and
took "+count+" recursions");
			route.add("["+dest[0]+","+dest[1]+"]");
			disPlay d = new disPlay(route,"["+now[0]+","+now[1]+"]",init);
			d.dislay();
			return ;
		}

		HashMap<String, Integer> map = new HashMap<String, Integer>(); // map
stands for the neighbors around the current node
		map.put("N[" + (now[0] - 1) + "," + now[1]+"]",(int)
Math.sqrt((Math.pow(dest[0] - now[0] + 1, 2) + Math.pow(dest[1]- now[1], 2))));
		map.put("E[" + now[0] + "," + (now[1] + 1)+"]",(int)
Math.sqrt((Math.pow(dest[0] - now[0], 2) + Math.pow(dest[1]- now[1] - 1, 2))));
		map.put("S[" + (now[0] + 1) + "," + now[1]+"]",(int)
Math.sqrt((Math.pow(dest[0] - now[0] - 1, 2) + Math.pow(dest[1]- now[1], 2))));
		map.put("W[" + now[0] + "," + (now[1] - 1)+"]",(int)
```

```
Math.sqrt((Math.pow(dest[0] - now[0], 2) + Math.pow(dest[1]- now[1] + 1, 2))));

            if(method == 2){
                  if (now[0] - 1 < 0 || maze[now[0] - 1][now[1]] == 1 // delete
the neighbors don't exist,marked as wall or has been expanded
                              || visited.containsKey(("[" + (now[0] - 1) + "," +
now[1] + "]"))) {
                        map.remove("N[" + (now[0] - 1) + "," + now[1] + "]");
                  }
                  if (now[1] + 1 >= maze[0].length
                              || maze[now[0]][now[1] + 1] == 1
                              || visited.containsKey(("[" + (now[0]) + "," +
(now[1] + 1) + "]"))) {
                        map.remove("E[" + now[0] + "," + (now[1] + 1) + "]");
                  }
                  if (now[0] + 1 >= maze.length
                              || maze[now[0] + 1][now[1]] == 1
                              || visited.containsKey(("[" + (now[0] + 1) + "," +
now[1] + "]"))) {
                        map.remove("S[" + (now[0] + 1) + "," + now[1] + "]");
                  }
                  if (now[1] - 1 < 0
                              || maze[now[0]][now[1] - 1] == 1
                              || visited.containsKey(("[" + (now[0]) + "," +
(now[1] - 1) + "]"))) {
                        map.remove("W[" + now[0] + "," + (now[1] - 1) + "]");
                  }
            }
            else
            {
                  if (now[0] - 1 < 0 || maze[now[0] - 1][now[1]] == 1){ // delete
the neighbors don't exist,marked as wall or has been expanded
                        map.remove("N[" + (now[0] - 1) + "," + now[1] + "]");
                  }
                  if (now[1] + 1 >= maze[0].length
                              || maze[now[0]][now[1] + 1] == 1) {
                        map.remove("E[" + now[0] + "," + (now[1] + 1) + "]");
                  }
                  if (now[0] + 1 >= maze.length
                              || maze[now[0] + 1][now[1]] == 1) {
                        map.remove("S[" + (now[0] + 1) + "," + now[1] + "]");
                  }
                  if (now[1] - 1 < 0
                              || maze[now[0]][now[1] - 1] == 1) {
                        map.remove("W[" + now[0] + "," + (now[1] - 1) + "]");
                  }
            }

            if(!map.isEmpty())
            {
                  Iterator<?> it = map.entrySet().iterator();//put the available
neighbors in the queue
                  while(it.hasNext())
                  {
                        Entry<?, ?> en = (Entry<?, ?>) it.next();
```

```
                        String string  = en.getKey().toString();
                        String string2 = en.getValue().toString();
                        on.put("["+now[0]+","+now[1]+"]"+string, dis +
Integer.parseInt(string2) + 1);
                  }
            }
            run(on, visited,route, dest);
      }
```

The DisPlay.java file display the best node that astar has found, it's about the data structure I was using for store the node, so it's basically nothing to tell about. The code:

```
      public void dislay()
            {
                  ArrayList<String> r = new ArrayList<String>();
                  r.add(route.get(route.size()-1));
                  String destString = "["+init[0]+","+init[1]+"]";
                  while(!now.contains(destString))
                  {
                        Iterator<String> iterator = route.iterator();
                        while(iterator.hasNext())
                        {
                              String s = iterator.next();
                              if(s.endsWith(now))
                              {
                                    route.remove(s);
                                    now = s.substring(0,s.indexOf("]")+1);
                                    r.add(s.substring(0,s.indexOf("]")+2));
                                    break;
                              }
                        }
                  }
                  int size = r.size();
                  for(int i=size-1;i>=0;i--)
                  {
                        System.out.println(r.get(i));
                  }

            }
```

The Search.java file just initialize the Astar class and call the functions to find the best path and to display it. Code:

```
      public class Search {

            /**
             * @param args
             */
            public static void main(String[] args) {
                  // TODO Auto-generated method stub
                  int[] init = {0,0};
                  int[] dest = {11,11};
```

```
                System.out.println("The Naive one is like this:");
                Astar a = new Astar(init,dest,1);//the basic one
                System.out.println("\n\nThe Priority Queue is like this:");
                Astar b = new Astar(init,dest,2);//the dynamic programming one
        }

    }
```

The time I used for this assignment.

I am really ashamed to talk about it, at least it took me 8 hours to think and try about the astar algorithm. I didn't understand it clearly, and failed times for coding. Even now I'm not sure whether my code is right or not, as the naive one doesn't take the whole night as the TA said.
But I'll try to do it better next time.