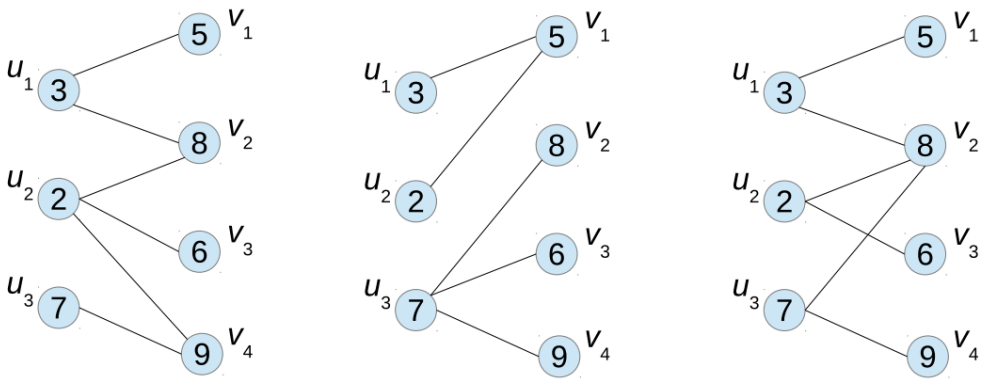
Zhiqiang Zhang
CS4040
11/23/2014

The Report of Dynamic Programming

Problem:

Given two lists u and v, and connect each vertex. Each vertex connects at least one vertex, and the cost of the connecting vertex is absolute value of u[ ] – v[ ]. However, the connecting edges cannot cross each other.

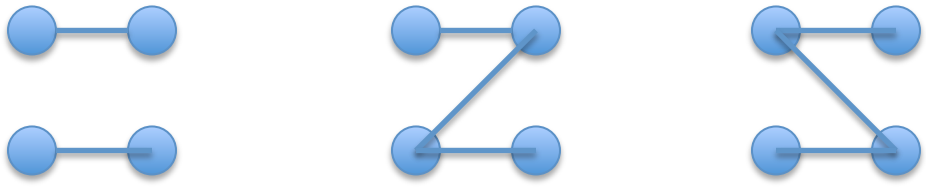
Example:



(a) (b) (c)

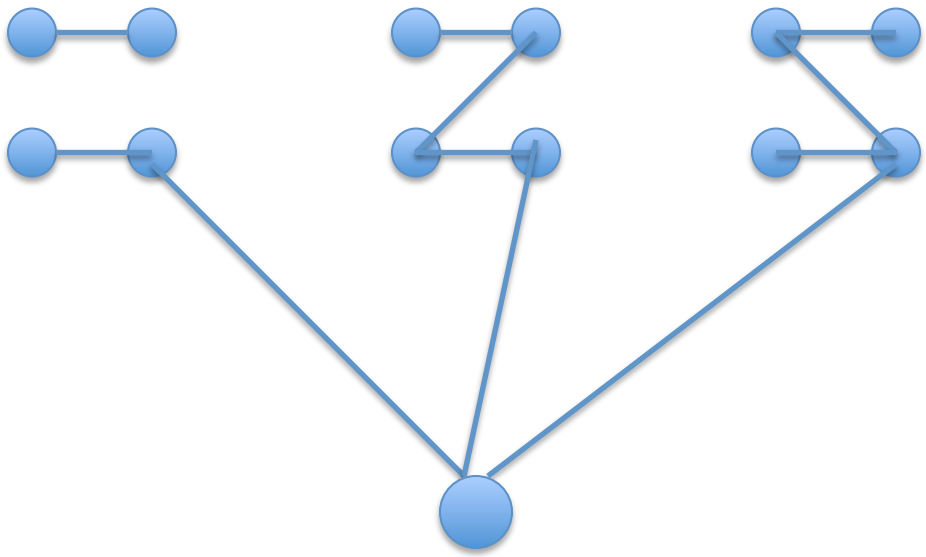
(a) is the legal connecting, but it is not minimum value of path;
(b) is legal connecting, and it is minimum value of path;
(c) is not legal, because connecting cross each other.

To observe the problem, we can see that the first one of u list should just connect the first one of v list. Also, the last one of u list should just connect last one of v list. So, we can assume we just have four notes. There is only three ways to connect them.



(Fig 1)

If you add any one more node, the optimal solution is just added in the minimum structure.



(a) (b) (c)

(Fig 2)

The addition note only can add in the one of the a, b or c. According to this, we can make a two dimensions matrix. For example, we have u list: 3 2 7, v list 5 8 6 9.

	0	1	2	3	4
0	0	∞	∞	∞	∞
1	∞	Abs(3-5)+0 2	Abs(3-8)+2 7	Abs(3-6)+7 10	Abs(3-9)+10 16
2	∞	Abs(2-5)+2 5	Abs(2-8)+2 8	Abs(2-6)+7 11	Abs(2-9)+10 17
3	∞	Abs(7-5)+2 7	Abs(7-8)+5 6	Abs(7-6)+6 7	Abs(7-9)+7 9

Basically, what you are going to do is that you will check the three situations, which are minimum one. Then, whenever you add the new node, you just go to find the best situation as fig 2 in the matrix. Thus, in the end of the matrix (3,4) is the optimal value.

After this, the optimal path is easily to found, you just back track and find the minimum of three situations have already computed.

	0	1	2	3	4
0	0	∞	∞	∞	∞
1	∞	Abs(3-5)+0 2	Abs(3-8)+2 7	Abs(3-6)+7 10	Abs(3-9)+10 16
2	∞	Abs(2-5)+2 5	Abs(2-8)+2 8	Abs(2-6)+7 11	Abs(2-9)+10 17
3	∞	Abs(7-5)+2 7	Abs(7-8)+5 6	Abs(7-6)+6 7	Abs(7-9)+7 9

Thus, to solve the problem, first of all characterize the structure of an optimal Solution: just find the three situations minimum one.

smallest(M[i-1][j],M[i][j-1],M[i-1][j-1]);

Second, recursively define the value of an optimal solution.

Recursively the formula to compute optimal value for each cell.

M[i]+M[j] = abs(u[i]-v[j])+ smallest(M[i-1][j],M[i][j-1],M[i-1][j-1]);

After that, Compute the value of an optimal solution in a bottom fashion,

void make_table(vector<int>&u, vector<int>&v, vector< vector<int> > &D){

```
	D[0][0] = 0;

	for(int i = 1; i < u.size(); i++)
		for(int j = 1; j < v.size(); j++){
			if(i == 1&&j==1){
				D[i][j] = abs(u[i]-v[j]) + D[0][0];
			}
			else if(i == 1&&j !=1){
				D[i][j] = abs(u[i]-v[j]) + D[i][j-1];
			}
			else if(i != 1&& j ==1){
				D[i][j] = abs(u[i]-v[j]) + D[i-1][j];
			}
			else{
				D[i][j] = abs(u[i]-v[j])+smallest(D[i-1][j],D[i][j-1],D[i-
1][j-1]);
		}
	}

}
```

This is to make the matrix, and the time complexity is O(n*m) = O(n^2), as double loop. Also, two dimension array is the O(n*m) = O(n^2) complexity.

Last, construct an optimal solution from the computed information. First of all, just print out the last cell of matrix is the optimal value of the path. Next print the optimal path.

```
void find_path(int x, int y, vector< vector<int> >&path,vector<int>&u,
vector<int>&v){
	int track;
	if(x<=1&&y<=1)
		return;
		else{
			if(x == 1&&y ==1){
				cout<<"("<<u[x]<<","<<v[y]<<")"<<" ";
			}
			else if(x == 1&&y != 1){
				 y = y-1;
				cout<<"("<<u[x]<<","<<v[y]<<")"<<" ";
			}
			else if(x !=1 && y==1){
				x = x-1;
				cout<<"("<<u[x]<<","<<v[y]<<")"<<" ";
			}
			else{
```

```
                track = smallest(path[x-1][y],path[x][y-1],path[x-1][y-1]);
                if(path[x-1][y] == track)
                        x = x-1;
                        else if(path[x][y-1] == track)
                                y = y-1;
                                else{
                                        x = x-1;
                                        y = y-1;
                                }
                find_path(x,y,path,u,v);
                cout<<"("<<u[x]<<","<<v[y]<<")"<<" ";
                }
        }

}
```

The worst case of the find the optimal path of the matrix, is the O(n+m), thus overall, the time complexity is the O(n). There is no time complexity this part, because it use the space as last function.