HW5 Readme

5-1 思路

- 這一題要找最長河流，於是我先將所有河流起點連到終點的距離相加。接著再利用**qsort由大到小排序**。此時，還有一點需要注意，就是相同距離的河流rank是一樣的，所以再利用for迴圈，掃過一次排序過後的陣列搭配 **binary search** 去紀錄真正的rank。

CODE分析

Qsort campare function(由大到小) 和 Binary search

```
ll cmp ( const void *a , const void *b )
{
    return *(ll *)b - *(ll *)a;
}
ll BinarySearch( ll* arr ,  ll n , ll k )
{
    ll left = -1, right = n;
    while( left < right - 1 )
    {
        ll mid = ( left + right ) / 2;
        if( arr[mid] > k )
        left = mid;
        else
        right = mid;
    }
    return right;
}// use binary search to process rank array
```

FindRiverSource函式

```
void FindRiverLenth( void )
//find all river source distance to the origin
{
    for( ll i = 0 ; i < num_river ; i++)
    {
        ll v = RiverInfos[ i ].nearcon ;
        store_river[ i ] += RiverInfos[ i ].neardis ;
        for( ll j = num_confluence  ; j >= 0 ; j-- )
        {
            if( j == v || v < j ) continue ;
            if( adj[ v ][ j ] <= 0 ) continue ;
            else
            {
                store_river[ i ] += adj[ v ][ j ] ;
                v = j ;
            }
        }
         RiverInfos[ i ].totdis = store_river[ i ] ;
    }
}
# 從每個河流的起點連到point zero，並且把途中的距離都加到
  store_river 裡面。
```

主函式

```
int main( )
{
    scanf("%lld %lld",&num_river,&num_confluence) ;
    for( ll i = 0 ; i < num_river ; i++ )
    {
        scanf("%s %lld %lld",RiverInfos[ i ].name , &RiverInfos[
        RiverInfos[ i ].idx = num_confluence+1+i ;
        adj[  RiverInfos[ i ].idx ][ RiverInfos[ i ].nearcon ] =
        adj[  RiverInfos[ i ].nearcon ][ RiverInfos[ i ].idx ] =
    }
    scanf("%lld",&num_relation) ;
    for( ll i = 0 ; i < num_relation ; i++ )
    {
        ll fc , sc , dis ;
        // fc = first source point
        // sc = second source point
        // dis = the distance between first and second point
        scanf("%lld %lld %lld",&fc,&sc,&dis) ;
        adj[ fc ][ sc ] = dis ;
        adj[ sc ][ fc ] = dis ;
    }
    FindRiverLenth( );
    qsort( store_river , num_river , sizeof( ll ) , cmp )  ;
    ll rank[ 100000 ] ;
    rank[ 0 ] = 1 ;
    ll cnt = 1 ;
    for( ll i = 1 ; i < num_river ; i++ )
    {
        if( store_river[ i ] == store_river[ i - 1 ] )
        {
            rank[ i ] = cnt ;
        }
        else
        {
            cnt++ ;
            rank[ i ] = cnt ;
        }
    }// the real rank of river lenth
    for( ll i = 0 ; i < num_river ; i++ )
    {
        printf("%s ",RiverInfos[ i ].name) ;
        ll ans = BinarySearch( store_river , num_river ,
                               RiverInfos[ i ].totdis ) ;
        printf("%lld\n",rank[ ans ] ) ;
```

```
44              printf( %lld\n ,rank[ ans ] ) ;
45          }
46          return 0 ;
47      }
```

5-2 思路

- 這題要輸出dfn和low值，並且找出**關節點**。
- 找出關節點的條件有分兩種
- 一種是root ， 若children > 1 則為關節點。
- 另一種非root，則用 **low[j] >= dfn[u]** 去判斷。
  其中 j 是 u 的 children 。

CODE分析

CalculateDfnLow 函式

```
void CalculateDfnLow( int u , int v )
{
    /* compute dfn and low while performing a dfs search
        beginning at vertex u,
        v is the parent of u (if any) */
    visited[ u ] = 1 ;
    dfn[ u ] = cnt , low[ u ] = cnt ;
    cnt++ ;
    int children = 0 ;
    for( int j =  0 ; j < num_vertex ; j ++ )
    {
        if( adj[ u ][ j ] == 0 ) continue ;
        if( visited[ j ] == 0 )
        {
            children++ ;
            CalculateDfnLow( j , u ) ;
            low[ u ] = MIN2( low[ u ] , low[ j ] ) ;
            if( v != -1 && low[ j ] >= dfn[ u ])
            {
                isAP[ u ] = 1 ;
            }
        }
        else if( j != v )
        {
            low[ u ] = MIN2( low[ u ] , dfn[ j ]);
        }
    }
    if( v == - 1 && children > 1 )
    {
        isAP[ u ] = 1 ;
    }
}
# 首先是depth first search ， 將dfn的值找出。
# 再來是low值的規則
  最多只能有一次走back edge
  並且要依照這個規則
  low(u)=min{dfn(u),
  min{low(w)|
  w is a child of u},min{dfn(w)|(u,w) is a back edge}
  找最小值
# 最後是找尋的過程中，用isAP來紀錄是否為關節點，用的是思路的規則。
  兩種情形，root 和 非root 。
```

主函式

```
int main( )
{
    scanf("%d %d", &ini_vertex , &num_vertex ) ;
    for( int i = 0 ; i < num_vertex ; i++ )
    {
        for( int j = 0 ; j < num_vertex ; j++ )
        {
            scanf("%d",&adj[ i ][ j ]) ;
        }
    }// USE adjacency matrix to store graph

    CalculateDfnLow( ini_vertex-1 , -1 ) ;

    for( int i = 0 ; i < num_vertex ; i++ )
    {
       printf("%d ",dfn[ i ]) ;
    }

    printf("\n") ;

    for( int i = 0 ; i < num_vertex ; i++ )
    {
        printf("%d ",low[ i ]) ;
    }

    printf("\n") ;

    for( int i = 0 ; i < num_vertex ; i++ )
    {
        if( isAP[ i ] == 1 )
            printf("%d ",i+1) ;
    }


    return 0 ;
}
# 這邊從題目給的起點開始進行depth first探索
# 並且在起點的parent上設為-1 ， 方便區別。
```

5-3 思路

- 這一題要處理**all pairs shortest path**的問題
- 有兩種方法
  第一種是做n次 Bellman
  第二種是利用二維陣列 Ak 去計算all pairs path
  // **no intermediate vertex of index greater than k**
  這邊的k指的是**經過的點**
  相較於bellman 的 k 指的是**走過的步數**。
  而我使用的是第二種方式。

CODE 分析

計算all pairs lenth函式

```
void FindAllPairsPath( void )
{
    /* dist[][] will be the output matrix
      that will finally have the shortest
      distances between every pair of vertices */
    int dist[n][n], i, j, k;

    /* Initialize the solution matrix*/
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            dist[i][j] = graph[i][j];

    /* Add all vertices one by one to*/
    for (k = 0; k < n; k++)
    {
        // Pick all vertices as source one by one
        for (i = 0; i < n; i++)
        {
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < n; j++)
            {
                // If vertex k is on the shortest path from
                // i to j,
                // then update the value of dist[i][j]
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    // Print the shortest distance matrix
    CalculateLenth(dist);
}
# 一開始走不到的地方設為INF 而自己連自己設為 0
# if(dist[i][k] + dist[k][j] < dist[i][j])
      dist[i][j] = dist[i][k] + dist[k][j];
# 上式用於更新 Ak 陣列，每次經過k這個點有更短的路徑就更新。
# 最後利用CalculateLenth將每一個點至其他點的距離相加
```

```
void CalculateLenth(int dist[][n])
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            totaldistance[ i ] += dist[ i ][ j ] ;
        }
    }
}
```

存圖方式

- 利用adjacency matrix 去紀錄vertex之間的關係。

```
for( int i = 0 ; i < m ; i++ )
        {
            int nodea , nodeb , weight ;
            scanf("%d %d %d",&nodea,&nodeb,&weight) ;
            graph[ nodea-1 ][ nodeb-1 ] = weight ;
            graph[ nodeb-1 ][ nodea-1 ] = weight ;
        }
```