貪心算法：最小生成樹

算法愛好者 前天

（給算法愛好者加星標，修煉編程內功）

> 來源：獨酌逸醉
> www.cnblogs.co m/chinazhangjie/archive/2010/12/02/1894314.html

【前言】前幾天發的《一文搞懂貪心算法》中提到了使用使用貪心算法來計算最短路問題，今天接著給大家分享下在最小生成樹的兩種算法中的貪心思想。

希望能對大家有所幫助。

設G = (V,E)是無向連通帶權圖，即一個網絡。E中的每一條邊（v,w）的權為c[v][w]。

如果G的子圖G'是一棵包含G的所有頂點的樹，則稱G'為G的生成樹。

生成樹上各邊權的總和稱為生成樹的耗費。在G的所有生成樹中，耗費最小的生成樹稱為G的最小生成樹。

構造最小生成樹的兩種方法：Prim算法和Kruskal算法。

一、最小生成樹的性質

設G = (V,E)是連通帶權圖，U是V的真子集。如果(u,v)∈E,且u∈U,v∈VU,且在所有這樣的邊中，(u,v)的權c[u][v]最小，那麼一定存在G的一棵最小生成樹，它意(u,v)為其中

一條邊。這個性質有時也稱為MST性質。

二、Prim算法

設G = (V,E)是連通帶權圖，V = {1,2,…,n}。構造G的最小生成樹Prim算法的基本思想是：首先置S = {1}，然後，只要S是V的真子集，就進行如下的貪心選擇：選取滿足條件i ∈S,j ∈V – S ,且c[i][j]最小的邊，將頂點j添加到S中。這個過程一直進行到S = V時為止。在這個過程中選取到的所有邊恰好構成G的一棵最小生成樹。

如下帶權圖：

生成過程：

1 -> 3 : 1

3 -> 6 : 4

6 -> 4: 2

3 -> 2 : 5

2 -> 5 : 3

實現：

```
#include <iostream>
#include <vector>
#include <limits>
using namespace std ;

struct TreeNode
```

```
{
public:
    TreeNode (int nVertexIndexA = 0, int nVertexIndexB = 0, int nWeight = 0)
        : m_nVertexIndexA (nVertexIndexA),
        m_nVertexIndexB (nVertexIndexB),
        m_nWeight (nWeight)
    { }
public:
    int m_nVertexIndexA ;
    int m_nVertexIndexB ;
    int m_nWeight ;
} ;

class MST_Prim
{
public:
    MST_Prim (const vector<vector<int> >& vnGraph)
    {
        m_nvGraph = vnGraph ;
        m_nNodeCount = (int)m_nvGraph.size () ;
    }
    void DoPrim ()
    {
        // 是否被访问标志
        vector<bool> bFlag (m_nNodeCount, false) ;
        bFlag[0] = true ;

        int nMaxIndexA ;
```

```
        int nMaxIndexB ;
        int j = 0 ;
        while (j < m_nNodeCount - 1) {
            int nMaxWeight = numeric_limits<int>::max () ;
            // 找到当前最短路径
            int i = 0 ;
            while (i < m_nNodeCount) {
                if (!bFlag[i]) {
                    ++ i ;
                    continue ;
                }
                for (int j = 0; j < m_nNodeCount; ++ j) {
                    if (!bFlag[j] && nMaxWeight > m_nvGraph[i][j]) {
                        nMaxWeight = m_nvGraph[i][j] ;
                        nMaxIndexA = i ;
                        nMaxIndexB = j ;
                    }
                }
                ++ i ;
            }
            bFlag[nMaxIndexB] = true ;
            m_tnMSTree.push_back (TreeNode(nMaxIndexA, nMaxIndexB, nMaxWeight)) ;
            ++ j ;
        }
        // 输出结果
        for (vector<TreeNode>::const_iterator ite = m_tnMSTree.begin() ;
                ite != m_tnMSTree.end() ;
                ++ ite ) {
```

```
            cout << (*ite).m_nVertexIndexA << "->"
                << (*ite).m_nVertexIndexB << " : "
                << (*ite).m_nWeight << endl ;
        }
    }
private:
    vector<vector<int> > m_nvGraph ;    // 无向连通图
    vector<TreeNode>    m_tnMSTree ;    // 最小生成树
    int    m_nNodeCount ;
} ;

int main()
{
    const int cnNodeCount = 6 ;
    vector<vector<int> > graph (cnNodeCount) ;
    for (size_t i = 0; i < graph.size() ; ++ i) {
        graph[i].resize (cnNodeCount, numeric_limits<int>::max()) ;
    }
    graph[0][1]= 6 ;
    graph[0][2] = 1 ;
    graph[0][3] = 5 ;
    graph[1][2] = 5 ;
    graph[1][4] = 3 ;
    graph[2][3] = 5 ;
    graph[2][4] = 6 ;
    graph[2][5] = 4 ;
    graph[3][5] = 2 ;
    graph[4][5] = 6 ;
```

```
    graph[1][0]= 6 ;
    graph[2][0] = 1 ;
    graph[3][0] = 5 ;
    graph[2][1] = 5 ;
    graph[4][1] = 3 ;
    graph[3][2] = 5 ;
    graph[4][2] = 6 ;
    graph[5][2] = 4 ;
    graph[5][3] = 2 ;
    graph[5][4] = 6 ;

    MST_Prim mstp (graph) ;
    mstp.DoPrim () ;
     return 0 ;
}
```

三、Kruskal算法

當圖的邊數為e時，Kruskal算法所需的時間是O(eloge)。當e = Ω(n^2)時，Kruskal算法比Prim算法差；但當e = o(n^2)時，Kruskal算法比Prim算法好得多。

給定無向連同帶權圖G = (V,E),V = {1,2,...,n}。Kruskal算法構造G的最小生成樹的基本思想是：

（1）首先將G的n個頂點看成n個孤立的連通分支。將所有的邊按權從小大排序。（2）從第一條邊開始，依邊權遞增的順序檢查每一條邊。

並按照下述方法連接兩個不同的連通分支：當查看到第k條邊(v,w)時，如果端點v和w分別是當前兩個不同的連通分支T1和T2的端點是，就用邊(v,w)將T1和T2連接成一個連通分支，然後繼續查看第k+1條邊；如果端點v和w在當前的同一個連通分支中，就直接再查看k+1條邊。

這個過程一個進行到只剩下一個連通分支時為止。此時，已構成G的一棵最小生成樹。

Kruskal算法的選邊過程：

1 -> 3 : 1

4 -> 6 : 2

2 -> 5 : 3

3 -> 4 : 4

2 -> 3 : 5

實現：

```
#include <iostream>
#include <vector>
#include <queue>
#include <limits>
using namespace std ;

struct TreeNode
{
public:
```

```
    TreeNode (int nVertexIndexA = 0, int nVertexIndexB = 0, int nWeight = 0)
        : m_nVertexIndexA (nVertexIndexA),
        m_nVertexIndexB (nVertexIndexB),
        m_nWeight (nWeight)
    { }
    friend
    bool operator < (const TreeNode& lth, const TreeNode& rth)
    {
        return lth.m_nWeight > rth.m_nWeight ;
    }

public:
    int m_nVertexIndexA ;
    int m_nVertexIndexB ;
    int m_nWeight ;
} ;

//  并查集
class UnionSet
{
public:
    UnionSet (int nSetEleCount)
        : m_nSetEleCount (nSetEleCount)
    {
        __init() ;
    }
    // 合并i，j。如果i，j同在集合中，返回false。否则返回true
    bool Union (int i, int j)
```

```
    {
        int ifather = __find (i) ;
        int jfather = __find (j) ;
        if (ifather == jfather )
        {
            return false ;
            // copy (m_nvFather.begin(), m_nvFather.end(), ostream_iterator<int> (cout,
            // cout << endl ;
        }
        else
        {
            m_nvFather[jfather] = ifather ;
            // copy (m_nvFather.begin(), m_nvFather.end(), ostream_iterator<int> (cout,
            // cout << endl ;
            return true ;
        }

    }

private:
    // 初始化并查集
    int __init()
    {
        m_nvFather.resize (m_nSetEleCount) ;
        for (vector<int>::size_type i = 0 ;
            i < m_nSetEleCount;
            ++ i )
        {
```

```
            m_nvFather[i] = static_cast<int>(i) ;
            // cout << m_nvFather[i] << " " ;
        }
        // cout << endl ;
        return 0 ;
    }
    // 查找index元素的父亲节点 并且压缩路径长度
    int __find (int nIndex)
    {
        if (nIndex == m_nvFather[nIndex])
        {
            return nIndex;
        }
        return  m_nvFather[nIndex] = __find (m_nvFather[nIndex]);
    }

private:
    vector<int>                m_nvFather ;    // 父亲数组
    vector<int>::size_type m_nSetEleCount ;    // 集合中结点个数
} ;

class MST_Kruskal
{
    typedef priority_queue<TreeNode> MinHeap ;
public:
    MST_Kruskal (const vector<vector<int> >& graph)
    {
        m_nNodeCount = static_cast<int>(graph.size ()) ;
```

```
        __getMinHeap (graph) ;
    }
    void DoKruskal ()
    {
        UnionSet us (m_nNodeCount) ;
        int k = 0 ;
        while (m_minheap.size() != 0 && k < m_nNodeCount - 1)
        {
            TreeNode tn = m_minheap.top () ;
            m_minheap.pop () ;
            // 判断合理性
            if (us.Union (tn.m_nVertexIndexA, tn.m_nVertexIndexB))
            {
                m_tnMSTree.push_back (tn) ;
                ++ k ;
            }
        }
        // 输出结果
        for (size_t i = 0; i < m_tnMSTree.size() ; ++ i)
        {
            cout << m_tnMSTree[i].m_nVertexIndexA << "->"
                << m_tnMSTree[i].m_nVertexIndexB << " : "
                << m_tnMSTree[i].m_nWeight
                << endl ;
        }
    }

private:
```

```
    void __getMinHeap (const vector<vector<int> >& graph)
    {
        for (int i = 0; i < m_nNodeCount; ++ i)
        {
            for (int j = 0; j < m_nNodeCount; ++ j)
            {
                if (graph[i][j] != numeric_limits<int>::max())
                {
                    m_minheap.push (TreeNode(i, j, graph[i][j])) ;
                }
            }
        }
    }
private:
    vector<TreeNode>    m_tnMSTree ;
    int                    m_nNodeCount ;
    MinHeap                m_minheap ;
} ;


int main ()
{
    const int cnNodeCount = 6 ;
    vector<vector<int> > graph (cnNodeCount) ;
    for (size_t i = 0; i < graph.size() ; ++ i)
    {
        graph[i].resize (cnNodeCount, numeric_limits<int>::max()) ;
    }
```

```
    graph[0][1]= 6 ;
    graph[0][2] = 1 ;
    graph[0][3] = 3 ;
    graph[1][2] = 5 ;
    graph[1][4] = 3 ;
    graph[2][3] = 5 ;
    graph[2][4] = 6 ;
    graph[2][5] = 4 ;
    graph[3][5] = 2 ;
    graph[4][5] = 6 ;

    graph[1][0]= 6 ;
    graph[2][0] = 1 ;
    graph[3][0] = 3 ;
    graph[2][1] = 5 ;
    graph[4][1] = 3 ;
    graph[3][2] = 5 ;
    graph[4][2] = 6 ;
    graph[5][2] = 4 ;
    graph[5][3] = 2 ;
    graph[5][4] = 6 ;

    MST_Kruskal mst (graph);
    mst.DoKruskal () ;
}
```

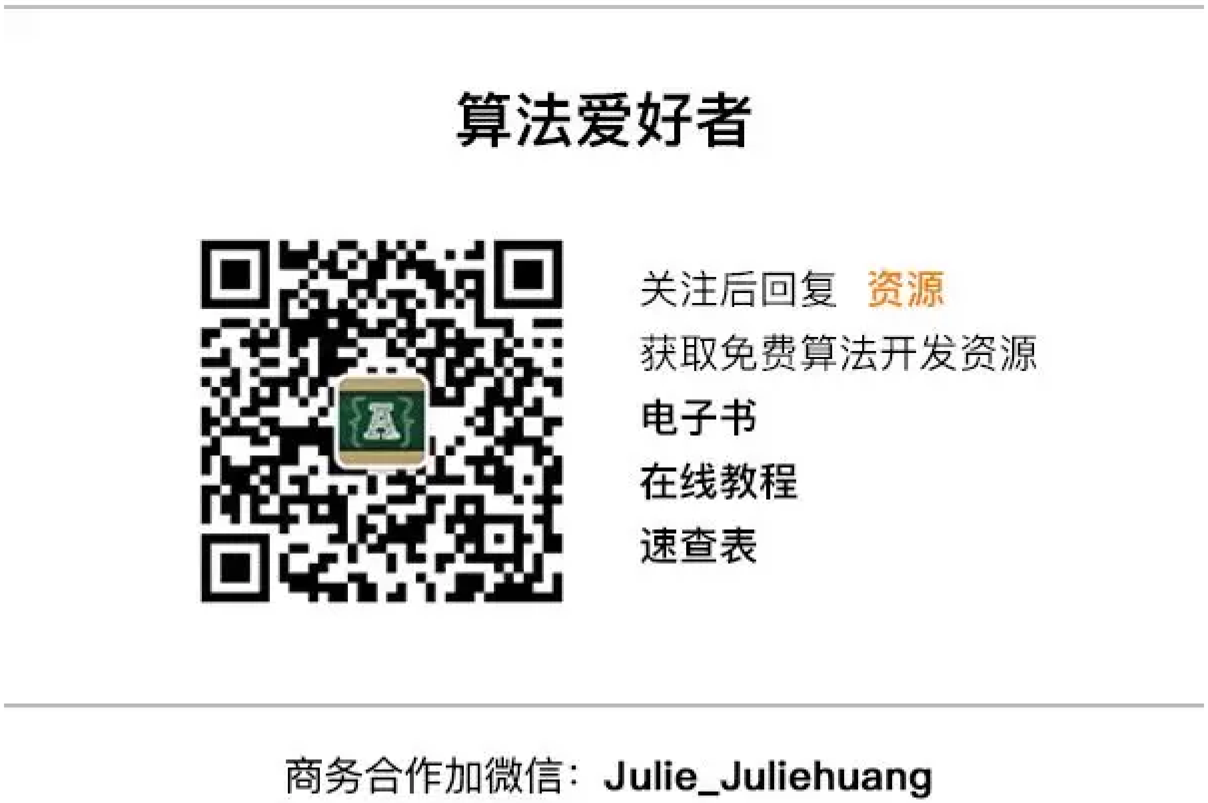
- EOF -

推薦閱讀 點擊標題可跳轉

1、一文搞懂貪心算法

2、ID生成器&雪花算法

3、三種洗牌算法簡介



點贊和在看就是最大的支持♡

10 天5 千Star！21 歲本科生給程序員開發的十六進制編輯器

開源前哨