发信人: violinist (巷战狙击手), 信区: Algorithm
标　题: 带权图最大权匹配--KM算法[转]
发信站: 日月光华 (2007年05月27日11:36:50 星期天)

带权图最大权匹配--KM算法[转]

KM算法是通过给每个顶点一个标号（叫做顶标）来把求最大权匹配的问题转化为求完备匹配的问题的。设顶点Xi的顶标为A[i]，顶点Yi的顶标为B[i]，顶点Xi与Yj之间的边权为w[i,j]。在算法执行过程中的任一时刻，对于任一条边(i,j)，A[i]+B[j]>=w[i,j]始终成立。KM算法的正确性基于
以下定理：
　　若由二分图中所有满足A[i]+B[j]=w[i,j]的边(i,j)构成的子图（称做相等子图）有完备匹配，那么这个完备匹配就是二分图的最大权匹配。
　　这个定理是显然的。因为对于二分图的任意一个匹配，如果它包含于相等子图，那么它的边权和等于所有顶点的顶标和；如果它有的边不包含于相等子图，那么它的边权和小于所有顶点的顶标和。所以相等子图的完备匹配一定是二分图的最大权匹配。
　　初始时为了使A[i]+B[j]>=w[i,j]恒成立，令A[i]为所有与顶点Xi关联的边的最大权，B[j]=0。如果当前的相等子图没有完备匹配，就按下面的方法修改顶标以使扩大相等子图，直到相等子图具有完备匹配为止。
　　我们求当前相等子图的完备匹配失败了，是因为对于某个X顶点，我们找不到一条从它出发的交错路。这时我们获得了一棵交错树，它的叶子结点全部是X顶点。现在我们把交错树中X顶点的顶标全都减小某个值d，Y顶点的顶标全都增加同一个值d，那么我们会发现：

两端都在交错树中的边(i,j)，A[i]+B[j]的值没有变化。也就是说，它原来属于相等子图，现在仍属于相等子图。
两端都不在交错树中的边(i,j)，A[i]和B[j]都没有变化。也就是说，它原来属于（或不属于）相等子图，现在仍属于（或不属于）相等子图。
X端不在交错树中，Y端在交错树中的边(i,j)，它的A[i]+B[j]的值有所增大。它原来不属于相等子图，现在仍不属于相等子图。
X端在交错树中，Y端不在交错树中的边(i,j)，它的A[i]+B[j]的值有所减小。也就说，它原来不属于相等子图，现在可能进入了相等子图，因而使相等子图得到了扩大。
　　现在的问题就是求d值了。为了使A[i]+B[j]>=w[i,j]始终成立，且至少有一条边进入相等子图，d应该等于min{A[i]+B[j]-w[i,j]|Xi在交错树中，Yi不在交错树中}。
　　以上就是KM算法的基本思路。但是朴素的实现方法，时间复杂度为O(n4)——需要找O(n)次增广路，每次增广最多需要修改O(n)次顶标，每次修改顶标时由于要枚举边来求d值，复杂度为O(n2)。实际上KM算法的复杂度是可以做到O(n3)的。我们给每个Y顶点一个“松弛量”函数slack，每
次开始找增广路时初始化为无穷大。在寻找增广路的过程中，检查边(i,j)时，如果它不在相等子图中，则让slack[j]变成原值与A[i]+B[j]-w[i,j]的较小值。这样，在修改顶标时，取所有不在交错树中的Y顶点的slack值中的最小值作为d值即可。但还要注意一点：修改顶标后，要把所有的
slack值都减去d。

CODE:

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;

const int size = 160;
const int INF = 100000000;  //  相对无穷大

bool map[size][size];         // 二分图的相等子图，map[i][j] = true 代表Xi与Yj
有边
bool xckd[size], yckd[size];  // 标记在一次DFS中，Xi与Yi是否在交错树上
int match[size];             // 保存匹配信息，其中i为Y中的顶点标号，match[i]为
X中顶点标号

bool DFS(int, const int);

void KM_Perfect_Match(const int n, const int edge[][size]) {
```

```
   int i, j;
   int lx[size], ly[size];   //  KM算法中Xi与Yi的标号
   for(i = 0; i < n; i++) {
      lx[i] = -INF;
      ly[i] = 0;
      for(j = 0; j < n; j++) {
         lx[i] = max(lx[i], edge[i][j]);
      }
   }
    bool perfect = false;
    while(!perfect) {
     //  初始化邻接矩阵
        for(i = 0; i < n; i++) {
            for(j = 0; j < n; j++) {
            if(lx[i]+ly[j] == edge[i][j]) map[i][j] = true;
            else map[i][j] = false;
         }
      }
     // 匹配过程
      int live = 0;
      memset(match, -1, sizeof(match));
      for(i = 0; i < n; i++) {
         memset(xckd, false, sizeof(xckd));
         memset(yckd, false, sizeof(yckd));
         if(DFS(i, n)) live++;
         else {
            xckd[i] = true;
            break;
         }
      }
      if(live == n) perfect = true;
      else {
        // 修改标号过程
         int ex = INF;
         for(i = 0; i < n; i++) {
            for(j = 0; xckd[i] && j < n; j++) {
               if(!yckd[j]) ex = min(ex, lx[i]+ly[j]-edge[i][j]);
          }
         }
         for(i = 0; i < n; i++) {
            if(xckd[i]) lx[i] -= ex;
            if(yckd[i]) ly[i] += ex;
         }
      }
   }
}

// 此函数用来寻找是否有以Xp为起点的增广路径，返回值为是否含有增广路

bool DFS(int p, const int n)
{
   int i;
   for(i = 0; i < n; i++) {
      if(!yckd[i] && map[p][i]) {
         yckd[i] = true;
         int t = match[i];
         match[i] = p;
         if(t == -1 || DFS(t, n)) {
            return true;
         }
         match[i] = t;
         if(t != -1) xckd[t] = true;
      }
   }
```

```
   return false;
}

int main()
{
   int n, edge[size][size]; //  edge[i][j]为连接Xi与Yj的边的权值
   int i;

   /*****************************************************
    *       在此处要做的工作 ：
    *       读取二分图每两点间边的权并保存在edge[][]中，
    *  若X与Y数目不等，应添加配合的顶点
    *       保存二分图中X与Y的顶点数n,若上一步不等应保
    *  存添加顶点完毕后的n
    *****************************************************/

   KM_Perfect_Match(n, edge);
   int cost = 0;
   for(i = 0; i < n; i++) {
      cost += edge[match[i]][i];
   }
   // cost 为最大匹配的总和，match[]中保存匹配信息

   return 0;
}
```

另附O(N^3)的算法代码:

```
#include <cstdio>
#include <queue>
#include <algorithm>
using namespace std;

const int N = 128;
const int INF = 1 << 28;

class Graph {
private:
   bool xckd[N], yckd[N];
   int n, edge[N][N], xmate[N], ymate[N];
   int lx[N], ly[N], slack[N], prev[N];
   queue<int> Q;
   bool bfs();
   void agument(int);
public:
   bool make();
   int KMMatch();
};
bool Graph::make() {
   int house[N], child[N], h, w, cn = 0;
   char line[N];
   scanf("%d %d", &h, &w);
   if(w == 0) return false;
   scanf("\n"); n = 0;
   for(int i = 0; i < h; i++) {
      gets(line);
      for(int j = 0; line[j] != 0; j++) {
         if(line[j] == 'H') house[n++] = i * N + j;
         if(line[j] == 'm') child[cn++] = i * N + j;
      }
   }
   for(int i = 0; i < n; i++) {
```

```
      int cr = child[i] / N, cc = child[i] % N;
      for(int j = 0; j < n; j++) {
         int hr = house[j] / N, hc = house[j] % N;
         edge[i][j] = -abs(cr-hr) - abs(cc-hc);
      }
   }
   return true;
}
bool Graph::bfs() {
   while(!Q.empty()) {
      int p = Q.front(), u = p>>1; Q.pop();
      if(p&1) {
         if(ymate[u] == -1) { agument(u); return true; }
         else { xckd[ymate[u]] = true; Q.push(ymate[u]<<1); }
      } else {
         for(int i = 0; i < n; i++)
            if(yckd[i]) continue;
            else if(lx[u]+ly[i] != edge[u][i]) {
               int ex = lx[u]+ly[i]-edge[u][i];
               if(slack[i] > ex) { slack[i] = ex; prev[i] = u; }
            } else {
               yckd[i] = true; prev[i] = u;
               Q.push((i<<1)|1);
            }
      }
   }
   return false;
}
void Graph::agument(int u) {
   while(u != -1) {
      int pv = xmate[prev[u]];
      ymate[u] = prev[u]; xmate[prev[u]] = u;
      u = pv;
   }
}
int Graph::KMMatch() {
   memset(ly, 0, sizeof(ly));
   for(int i = 0; i < n; i++) {
      lx[i] = -INF;
      for(int j = 0; j < n; j++) lx[i] >?= edge[i][j];
   }
   memset(xmate, -1, sizeof(xmate)); memset(ymate, -1, sizeof(ymate));
   bool agu = true;
   for(int mn = 0; mn < n; mn++) {
      if(agu) {
         memset(xckd, false, sizeof(xckd));
         memset(yckd, false, sizeof(yckd));
         for(int i = 0; i < n; i++) slack[i] = INF;
         while(!Q.empty()) Q.pop();
         xckd[mn] = true; Q.push(mn<<1);
      }
      if(bfs()) { agu = true; continue; }
      int ex = INF; mn--; agu = false;
      for(int i = 0; i < n; i++)
         if(!yckd[i]) ex <?= slack[i];
      for(int i = 0; i < n; i++) {
         if(xckd[i]) lx[i] -= ex;
         if(yckd[i]) ly[i] += ex;
         slack[i] -= ex;
      }
      for(int i = 0; i < n; i++)
         if(!yckd[i] && slack[i] == 0) { yckd[i] = true; Q.push((i<<1)|1); }
```

```
    }
    int cost = 0;
    for(int i = 0; i < n; i++) cost += edge[i][xmate[i]];
    return cost;
}

int main()
{
    Graph g;

    while(g.make()) printf("%d\n", -g.KMMatch());
    return 0;
}
```

--
※ 来源:·日月光华 bbs.fudan.edu.cn·HTTP [FROM: 211.65.100.130]