导航

- 首页
- 社区主页
- 当前事件
- 最近更改
- 随机页面
- 使用帮助
- NOCOW地图
- 新手试练场

搜索

进入 搜索

工具箱

- 链入页面
- 链出更改
- 特殊页面
- 可打印版
- 永久链接

Kruskal算法

基本思想

假设WN=(V,{E})是一个含有n个顶点的连通网，则按照克鲁斯卡尔算法构造最小生成树的过程为：先构造一个只含n个顶点，而边集为空的子图，若将该子图中各个顶点看成是各棵树上的根结点，则它是一个含有n棵树的一个森林。之后，从网的边集E中选取一条权值最小的边，若该条边的两个顶点分属不同的树，则将其加入子图，也就是说，将这两个顶点分别所在的两棵树合成一棵树；反之，若该条边的两个顶点已落在同一棵树上，则不可取，而应该取下一条权值最小的边再试之。依次类推，直至森林中只有一棵树，也即子图中含有n-1条边为止。

PASCAL代码

```
Procedure kruskal(V,E);
begin
sort(E,1,m);//将边按照权值排序
for t:=1 to m do begin
   if getfather(edge[t].u)<>getfather(edge[t].v) then begin  //利用并查集判断两个顶点是否在同一集合内
      tot:=tot+edge[t].data;//计算权值和
      union(edge[t].u,edge[t].v);//合并顶点
      inc(k);//合并次数
      end;
   end;
if k=n-1 then 形成了一棵最小生成树
         else 不存在这样的最小生成树;
end;
```

C++语言代码

```
struct KRUSKAL
{
        const int MAXN = 109;
        const int MAXE = 5009;

        struct EDGE
        {
                int u, v, length, choose;
        }        edge[ MAXE ];

        int path[ MAXN ];
        int N, edgecnt, sum;

        void Addedge(int u, int v, int len)
        {
                ++edgecnt;
                edge[ edgecnt ].u = u;
                edge[ edgecnt ].v = v;
                edge[ edgecnt ].length = len;
                edge[ edgecnt ].choose = false;
```

```
                return ;
        }

        void Set()
        {
                for (int i = 1; i <= N; i++)
                        path[i] = i;
                return ;
        }

        int Find_Path(int x)
        {
                if (x != path[x]) path[x] = Find_Path( path[x] );
                return path[x];
        }

        int Work()
        {
                int cnt = 0, x, y;
                Qsort(1, edgecnt);        // i < j -> edge[i].length < edge[j].length
                Set();
                for (int i = 1; i <= E && cnt < N - 1; i++)
                {
                        x = Find_Path( edge[i].u );
                        y = Find_Path( edge[i].v );
                        if (x == y) continue;
                        path[x] = path[y];
                        edge[i].choose = true, ++cnt;
                        sum += edge[i].length;
                }
                return sum;
        }
}       Kruskal;
```

优化

在判断两个顶点是否在同一集合内时可用并查集

代码可以如下

```
procedure union(x,y:longint);
begin
  if r[y]<r[x] then
  begin
    y:=y xor x;
    x:=y xor x;
    y:=y xor x
  end;//交换，有疑问者请参照xor的运算

  p[x]:=y;
  if r[y]=r[x] then inc(r[y]);
end;
function find(x:longint):longint;
begin
  if x<>p[x] then p[x]:=find(p[x]);
  exit(p[x]);
end;
function clear:boolean;
var
  i,t:longint;
begin
  clear:=true;
  t:=find(1);
  for i:=2 to m do if find(i)<>t then exit(false);
end;
procedure solve;
var
  i,k:longint;
begin
  for i:=1 to n do
  begin
    if clear then exit;
    if find(a[i].f)<>find(a[i].t) then
    begin
      inc(ans,a[i].c);
      union(find(a[i].f),find(a[i].t));
    end;
  end;
end;//注:  使用前请排序
```

```
/*使用Union-Find判断是否在一个集合，代码比较STL-style
```

```
        Author:YangZX*/
#include <iostream>
using namespace std;
const int MAXV = 1024, MAXE = 100001;
int n, m, f[MAXV], ans, cnt;
struct edge{
        int f, t, w;
}es[MAXE];
bool cmp(const edge &a, const edge &b){
        return a.w < b.w;
}
void Fill(int &a){
        static int cnt = 0;
        a = ++cnt;
}
int get(int x){
        return x == f[x] ? x : f[x] = get(f[x]);
}
void Kruskal(const edge &e){
        if(get(e.f) != get(e.t)){
                f[get(e.f)] = get(e.t);
                ans += e.w;
                cnt++;
        }
}
void Read(edge &e){
        cin>>e.f>>e.t>>e.w;
}
int main()
{
        cin>>n>>m;
        for_each(es+1, es+m+1, Read);
        make_heap(es+1, es+m+1, cmp);
        sort_heap(es+1, es+m+1, cmp);
        for_each(f+1, f+n+1, Fill);
        for_each(es+1, es+m+1, Kruskal);
        cout<<(cnt < n-1 ? -1: ans)<<endl;
        return 0;
}
```

算法实例

VOJP1045

- 题目简述:

1、输入:

```
第一行一个正实数S:要求最小生成树中所有边的长度不大于s
第二行一个正整数n:有n个结点
接下来一共有m行，第i行有两个整数xi,yi和一个实数si，表示点xi和点yi间有一条边，边的长度为si。
输入保证xi不等于yi，两个点之间不会有两条边。
```

2、输出:

```
若存在这样的最小生成树，则输出（其中<X>代表最少的电缆线长度，保留两位小数）:
Need <X> miles of cable
否则输出:
Impossible
```

- 代码

```
const
  maxn=1000000;
var
  i,j,k,m,n,p,q,x,y:longint;
  f,u,v:array[1..maxn] of longint;
  ans,s:extended;
  e:array[1..maxn] of real;
procedure swap(var x,y:longint);
var
  t:longint;
begin
  t:=x;
  x:=y;
  y:=t;
end;
```

```
procedure qsort(l,r:longint);
var
  i,j:longint;
  x,y:extended;
begin
  i:=l;
  j:=r;
  x:=e[random(r-l)+l];
  repeat
    while e[i]<x do inc(i);
    while e[j]>x do dec(j);
    if i<=j then
      begin
        y:=e[i];
        e[i]:=e[j];
        e[j]:=y;
        swap(u[i],u[j]);
        swap(v[i],v[j]);
        inc(i);
        dec(j);
      end;
  until i>j;
  if l<j then qs(l,j);
  if i<r then qs(i,r);
end;
function get(x:longint):longint;
begin
  if f[x]=0 then exit(x);
  if f[f[x]]=0 then exit(f[x]);
  f[x]:=get(f[x]);
  exit(f[x]);
end;
procedure init;
begin
  fillchar(e,sizeof(e),0);
  fillchar(u,sizeof(u),0);
  fillchar(v,sizeof(v),0);
  fillchar(f,sizeof(f),0);
  readln(s);
  readln(n);
  m:=0;
  while not eof do
    begin
      inc(m);
      readln(u[m],v[m],e[m]);
    end;
end;
procedure outit;
begin
  if (k=n-1) and (ans<=s) then
    writeln('Need ',ans:0:2,' miles of cable')
  else
    writeln('Impossible');
end;
begin
  init;
  qsort(1,m);
  // Kurskal
  k:=0;
  for i:=1 to m do
    begin
      p:=get(u[i]);
      q:=get(v[i]);
      if p<>q then
        begin
          ans:=ans+e[i];
          f[p]:=q;
          inc(k);
        end;
    end;
  outit;
end.
```

图论及图论算法 [编辑]

图 - 有向图 - 无向图 - 连通图 - 强连通图 - 完全图 - 稀疏图 - 零图 - 树 - 网络

基本遍历算法：宽度优先搜索 - 深度优先搜索 - A* - 并查集求连通分支 - Flood Fill

最短路：Dijkstra - Bellman-Ford（SPFA） - Floyd-Warshall - Johnson算法

最小生成树：Prim - Kruskal

强连通分支：Kosaraju - Gabow - Tarjan

网络流：增广路法（Ford-Fulkerson，Edmonds-Karp，Dinic） - 预流推进 - Relabel-to-front

图匹配 - 二分图匹配：匈牙利算法 - Kuhn-Munkres - Edmonds' Blossom-Contraction

Kruskal算法是一个小作品，欢迎帮助扩充这个条目。

2个分类：图论 | 小作品