证明及程序的证明

赵建华

证明的定义

- 公式f的证明是一系列公式的序列

f1,f2,...,fk(f)

其中

  - fi要么是公理的实例化
  - 要么可以由之前的公式，根据proof rule推导得到。

证明的结构化

- 假设证明目标是f，我们可以先证明子目标f1,f2,...,fn，f可以由这些子目标推导得到。
- 对于fi，又可以给出相应的证明过程
- 书写方式见右

f1

    f1的证明

f2

    f2的证明

…

fk

    fk的证明

f

反证法

- 要证明p=>q
  - 假设q不成立，证明p也不成立，即not q=>not p
  - 或者假设p成立，q不成立（即not q），然后证明矛盾，实际上即p and not q => false
- 按照证明的定义，即
  - 首先推导出not q=>not p 或者p and not q => false
  - 然后推导出p=>q

数学归纳法

- 自然数的性质$\forall n.\,P(n)$的证明分成基础步骤和归纳步骤
  - P(1)成立（或者某个基础P(k)成立）
  - 假设P(k)成立，推导出P(k+1)也成立，即P(k)=>P(k+1)
- 经常碰到的问题
  - 对于要证明的P，P(k)=>P(k+1)未必成立，此时需要找一个更强的性质P'(k)
  - 被证明的性质针对的不是自然数，而是其它结构，比如：图，树，公式，...

数学归纳法和循环

- 循环语句

  while(condition)

  statement
- 假设我们要证明：当循环语句结束时，性质P成立。
- 基本方法：寻找一个性质P’，使得
  - –P’在循环语句之前成立，
  - –如果P’ and condition在循环体之前成立，那么循环体运行结束时，p’仍然成立
  - –P’ and not condition => P
- 有关P’的另一个描述：对于任意的n，循环体执行了n次之后P’仍然成立
- 是否足够保证程序正确？

循环的例子

```
for(i = 0; i<100; i++)
        A[i] = 0;
```

- 要证明循环结束之后，数组A的前100个元素都等于零
  - P’是什么？
- 如何证明循环结束时，i=100?

单链表搜索

```
cur = first;
while(cur != null && cur->Data != x)
      cur = cur->link;
```

- 假设循环开始时，first指向一个单链表，
- 要求证明
  - 当循环结束时，要么cur==null，且first所指单链表中没有结点的Data字段为x；
  - 要么cur!=null，且cur->Data==x.

递归函数的证明

- 假设f是递归函数

  f(x)

  {... f(g1(x))...f(g2(x))...

  }
- 我们要证明f(x0)执行结束后，性质P成立
- 步骤
  - 设定f的性质，即f(x)执行完成后，P’(x)成立，而P’(x)=>P
  - 可以假设f(g1(x))之后，P’(g1(x))和P’(g2(x))成立
  - 是否足够保证函数正确？

递归函数的例子

- Qsort的例子

```
QSort(l,r)
{
   int i = Partion(l,r);
   QSort(l,i-1);
   QSort(i+1,r);
}
```

如何证明一些图论算法的正确性？

- Dijkstra算法
- Folyd-Warshall算法
- Huffman树构造算法
- 最小生成树

单源最短路径的Dijkstra算法

P1(X) :$\forall i \in X.(dist[i]$即从0到i的最短路径长度)

P2(X) : $\forall i \in V - X.(dist[i]$即从0到$i$的，且只有$i$不在$X$内的最短路径的长度)

P3(X) : 从0到X中结点的最短路径都短于从0到V-X中结点最短路径的长度

当循环结束时，P1(V)成立，即Dist[i]存放从0到i的最短路径

```
S←{v0};    Dist[0] = 0;
for j = 1, 2, …, n-1 {dist[j]←Edge(0,j);}
//P1(S) and P2(S) and P3(S)
while ( S != V )
{   k = 在V-S中，使dist[k]最小的下标；
    //由P2(S)和P3(S)可推导得到
        //dist[k]是从0到k的最短路径长度；
        //且对于V-S中的任意其它顶点x，从0到x的最短路径大于等于disk[k]；
    //即P1(S∪{k}) ∧ P3(S∪{k})
    S ← S∪{k};
    //S的新值是S∪{k}，因此P1(S)和P3(S)成立，但P'(2)不一定成立；
    对于每一个 i∈V-S，
        if(dist[k]+Edge[k][i]<dist[i])  dist[i] ← dist[k]+Edge(k,i);
    //P1(S)，P2(S)和P3(S)均成立
}
```

求最小生成树的Kruskal算法

P(X) 表示：X中不包含回路，且如将X中元素从小到大排列（如权值相等则任意排列），那么第i个元素是所有不与前i-1个元素形成回路的最小边（之一）

Q(X)表示：如果边e不和X中的边构成回路，那么e的权值大于等于X中的任意边。

```
E1={}
//P(E1) and Q(E1)成立
while (|E1|<n-1)
{   //P(E1) and Q(E1)成立
    在E-E1中选不与E1中已有边构成回路的、权值最小的边e;
    //显然E1+{e}中不包含回路,
    //由Q(E1)可知e的权值不小于E1中所有边的权值,
    //且如果e的权值和E1中某些边相等，那么e和这些相等的边任意排列都不会违反P的定义,
    //因此P(E1+{e}),
    //因为e是满足条件的最小边，因此所有不和E1中边构成回路的边的权值都不小于e,
    //因此Q(E1+{e});
    //也就是说：P(E1+{e}) and Q(E1+e)
    E1=E1+{e};
    //E1被赋予新值(E1+{e})，因此P(E1)和Q(E1)仍然成立
}
```

循环结束时，E1具有n-1条边，且P(E1)成立。可以推出E1是最小生成树

Prim算法

显然，在循环开始时，G’和G实际上同构，E1是空集，因此INV成立；

当循环结束时，U=V，因此G’只有一个结点，因此MST(G)是空集，因此
INV and (U=V) => MST(G) = E1

假设INV在循环体开始时成立，选取的边必然是和u’相关联的最短边，根据MST的性质可知，将e的另外一个顶点v加入U之后（相当于v和u’合并）得到的新图G’’满足MST(G’) = MST(G’’) + e,
当循环体结束时，G’相当于G’’，而E1增加了e，因此MST(G) = MST(G’) + E1。

任选顶

while(U

{

}

循环结

设原来

G’={V

循环不变式：INV：MST(G) = MST(G’)+E1

Floyd算法的具体实现

fo

fo

三层循环的INV：

最外层的k循环：

a[x][y]=从x到y，不经过序号大于等于k的结点的最短路径长度；

中间的i循环：

如x小于i，a[x][y]表示从x到y，不经过序号大于k的结点的最短路径长度；

如x大于等于i，a[x][y]表示从x到y，不经过序号大于等于k的结点的最短路径长度。

j循环：????

};