§5.5 找第k个最小元

一．问题：

输入：n个有线性序的元素组成的集合S和整数k, 。

输出：S中第k个最小元。

二．算法：

PROCEDURE SELECT (k, S);

BEGIN

1. IF |S| < 50 THEN

BEGIN

1. 用某种排序方法将S排序；
2. 返回S中第k个最小元；

END

ELSE BEGIN

1. 将S分成个子序列，每个子序列5个元素，最后一个

子序列取全部剩余元素；

1. 将每个子序列排序；
2. 设M是所有有序子序列的中间那个元素组成的集合；
3. m := SELECT();
4. 设和分别是S中小于m，等于m和大于m的

那些元素组成的集合；

1. IF THEN RETURN( SELECT(k, ) )

ELSE

10. IF THEN RETURN(m)

11. ELSE RETURN();

END

END;

三．算法分析:

设是从个元素的集合中选第个最小元所需的时间。，递归调用SELECT()至多需时间。

和中每个至多含个元素, 这是因为M中至少有个元素大于等于m, 对其中每个元素，S中有两个不同元素不小于它。因此，至多含有个元素，当时，。

类似可证。在第10行或12行的递归调用至多需时间，其它语句至多需时间。故存在常数，使得

用归纳法可证，。

故本算法需时间。

§5.6 查找

一．二分查找

PROCEDURE Binary\_Search (n, F, x; VAR i);

/\* 用二分查找方法在文件F中找记录x, 若x在F中，则将其序号送给i, 否则i为0。n为F的记录个数 \*/

BEGIN

l := 1; h := n; i := 0;

WHILE l h DO

BEGIN

m := (l + h) DIV 2;

IF x.key < F[m].key THEN h := m-1

ELSE IF x.key > F[m].key THEN l := m+1

ELSE BEGIN /\* x.key = F[m].key \*/

i := m;

l := h+1;

END;

END;

END;

二．算法分析

二分查找可用二叉树形象地说明。

最坏情况的时间与二叉树的深度成比例。

当, 有，因而二叉树的深度。故最坏情况下，时间复杂度为。

平均时间复杂性分析：

查找第i层(1)需要i次比较，第i层有个记录。设每个记录的查找概率相同，则。平均查找次数：

其中，l为叶结点个数，, 。

先用归纳法证明：

。

当时，，公式成立。假设公式对m成立，现在证对亦成立。

故

考虑到，故 。

于是

又考虑到

故，于是

从而平均时间复杂度为。

第六章 图的算法

§6.1 图的遍历

一．几个概念

1. 生成树

给定一个图，的一棵生成树是一棵树，且及。

2. 深度优先搜索：深度优先搜索的思想是，从任一顶点出发，找它的一个相邻未访问过的顶点，作为下一个要访问的顶点，再从下一个顶点出发，找它的一个相邻未访问过的顶点，作为下一个要访问的顶点，如此下去。如果当前顶点无相邻未访问的顶点，则退到上一顶点，找它的其它未访问过的相邻顶点，作为下一个要访问的顶点。

3. 宽度优先搜索：宽度优先搜索的思想是，从任一顶点出发，先逐个访问完它的所有相邻点，再从每一个相邻点出发，访问它们的所有相邻未访问的顶点。

4. 深先搜索生成树和深先搜索生成森林：按深先搜索的方法遍历图所得到的树或森林。

5. 回边和树边

例子：

图6.1

二．深度优先遍历图的算法

1. 递归算法：

输入：图和邻接表。

输出：把分成树边集和回边集。

PROCEDURE Search (v)

BEGIN

1. mark v “old”;
2. FOR each vertex w on L[v] DO

3. IF w is marked “new” THEN

BEGIN

4. add (v, w) to T;

5. Search(w);

END

END;

主程序：

BEGIN

6. T := ;

7. FOR all v in V DO mark v “new”;

8. WHILE there exists a vertex v in V marked “new” DO

9. Search(v);

END;

2.算法分析：

最坏情况下时间复杂度：，其中，n为图的顶点数，e为图的边数。

第7, 8行找所有顶点和找所有“新”顶点需时间。Search(v)过程(不含递归调用)所花的时间与跟v相邻的顶点数成正比。对每个顶点v，Search(v)只调用一次。故Search总共花的时间与成正比。故算法的时间复杂度为O(MAX(n, e))。

1. 几个性质:

引理6.1：如果(v, w)是一条回边，那么在深度优先搜索生成森林中，v是w的祖先，或相反。

证明：不失一般性，设v比w先访问。当v被访问时，w仍标记为“new”，由Search(v)访问的所有标记为“new”的顶点在生成森林中都是v的后代，并且Search(v)在w被访问前不可能结束，这因为w在中。从而有本引理的结论。

\*深度优先搜索编号：DFNUMBER[v]

4. 非递归算法:

PROCEDURE DFS (i : integer; adjlist : 图的邻接表；VAR visit :

访问标志数组)；

/\*非递归深度优先搜索遍历图，i为起始顶点标号\*/

VAR

stack : ARRAY [1max] OF integer;

top : integer; j :integer; p : 结点指针；

BEGIN

FOR j := 1 TO max DO

visit[j] := false;

WRITE(“v”, i : 1);

top := 1;

stack[top] := i; /\* 起始结点i进栈 \*/

visit[i] := true; /\* 置起始顶点已访问标志 \*/

REPEAT

p := adjlist[stack[top]]; /\* 取栈顶顶点的邻接表 \*/

WHILE (pNIL) AND visit[pvertex] DO

p := p.link; /\* 找下一个未访问过的相邻点 \*/

IF p = NIL THEN top := top1 /\*当前顶点的相邻点都已访问过\*/

ELSE BEGIN /\* 访问下一个未访问过的相邻点 \*/

i := P.vertex;

WRITE(“v”, i : 1);

visit[i] := true;

top := top+1;

stack[top] := i ;

END

UNTIL top = 0;

END;

三．宽度优先遍历图的算法

1. 算法：

PROCEDURE BFS (i : integer; adjlist : 图的邻接表；VAR visit :

访问标志数组)；

/\* 非递归宽度优先搜索遍历图，i : 起始顶点标号 \*/

VAR

queue : ARRAY [1max] OF integer;

j, hp, tp : integer;

p : 顶点指针；

BEGIN

FOR j := 1 TO max DO

visit[j] := false;

hp := 1; tp := 1;l

WRITE(“v”, i : 1); /\* 输出起始顶点 \*/

visit[i] := true; /\* 置起始顶点已访问标志 \*/

queue[hp] := i ; /\* 起始顶点进队列 \*/

REPEAT

p := adjlist[queue[hp]]; /\* 取队列头顶点的邻接表 \*/

WHILE p NIL DO

BEGIN /\* 当该顶点有相邻顶点时做 \*/

i := p.vertex; /\* 取该相邻点的编号 \*/

IF NOT visit[i] THEN /\* 若未访问过 \*/

BEGIN

WRITE(“v”, i : 1);

visit[i] := true; /\* 置已访问标志 \*/

tp := tp+1;

queue[tp] := i ; /\* 顶点i进队列 \*/

END;

p := p.link;

END;

hp := hp+1; /\* 队列头元素出队列 \*/

UNTIL hp > tp;

END;

2. 算法分析

最坏情况下时间复杂度：。

作业10：

1.设

证明：。从而完成找第k个最小元算法的复杂性分析。

2. 设是一个无向图。给出一个时间的算法，求中的一条路径，经过中每一条边(在每一个方向上)恰好一次。如果你有足够的零钱，试找出一个走出迷宫的方法。