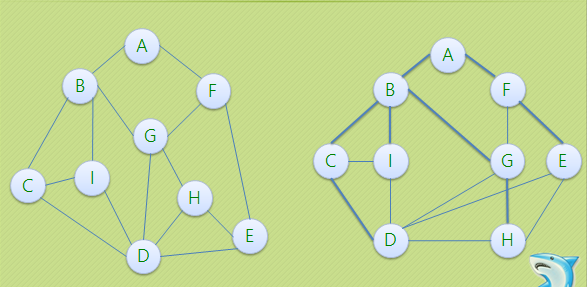
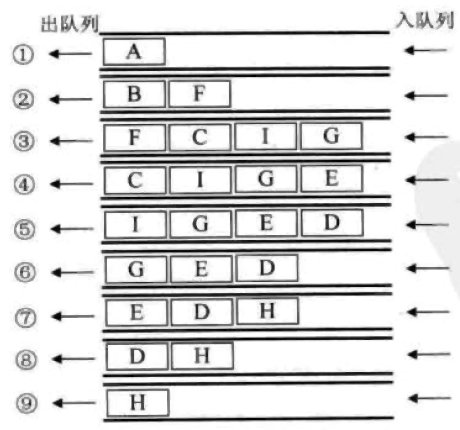
61图的遍历（广度优先遍历）

广度优先遍历

* 广度优先遍历（BreadthFirstSearch），又称为广度优先搜索，简称BFS。
* 如果以之前我们找钥匙的例子来讲，运用深度优先遍历意味着要先彻底查找完一个房间再开始另一个房间的搜索。
* 但我们知道，钥匙放在沙发地下等犄角旮旯的可能性极低，因此我们运用新的方案：先看看钥匙是否放在各个房间的显眼位置，如果没有，再看看各个房间的抽屉有没有。这样逐步扩大查找的范围的方式我们称为广度优先遍历。



* 那么要实现对图的广度遍历，我们可以利用队列来实现：
* 定义右手原则



出队顺序即为广度优先遍历的过程：

ABFCIGEDH

* 代码！代码！！
* 小甲鱼这里演示给大家邻接矩阵的代码编写，大家课后一定要自行完成邻接表的广度遍历算法编写哦~

BTSTraverse.c

// 邻接矩阵的广度遍历算法

void BFSTraverse(MGraph G)

{

int i, j;

Queue Q;

for( i=0; i < G.numVertexes; i++ )

{

visited[i] = FALSE;

}

initQueue( &Q );

for( i=0; i < G.numVertexes; i++ )

{

if( !visited[i] )

{

printf("%c ", G.vex[i]);

visited[i] = TRUE;

EnQueue(&Q, i);

while( !QueueEmpty(Q) )

{

DeQueue(&Q, &i);

for( j=0; j < G.numVertexes; j++ )

{

if( G.art[i][j]==1 && !visited[j] )

{

printf("%c ", G.vex[j]);

visited[j] = TRUE;

EnQueue(&Q, j);

}

}

}

}

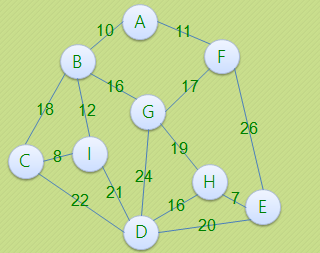
}

}

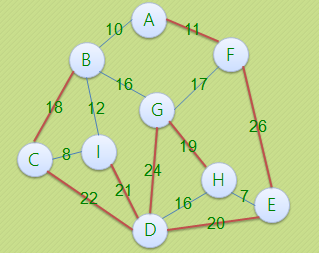
# 62最小生成树（普里姆算法）

## 最小生成树

* 小苍童鞋的难题：

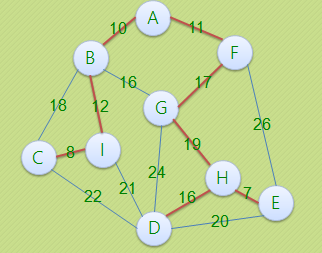


## 方案一



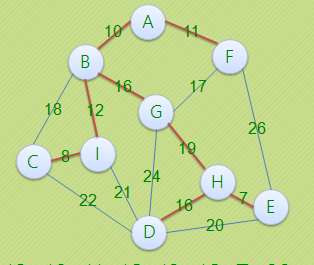
* 成本：11+26+20+22+18+21+24+19=161

## 方案二



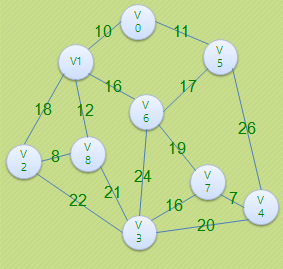
* 成本：8+12+10+11+17+19+16+7=100

## 方案三



* 成本：8+12+10+11+16+19+16+7=99

普里姆算法



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **V0** | **V1** | **V2** | **V3** | **V4** | **V5** | **V6** | **V7** | **V8** |
| **V0** | 0 | 10 | ∞ | ∞ | ∞ | 11 | ∞ | ∞ | ∞ |
| **V1** | 10 | 0 | 18 | ∞ | ∞ | ∞ | 16 | ∞ | 12 |
| **V2** | ∞ | 18 | 0 | 22 | ∞ | ∞ | ∞ | ∞ | 8 |
| **V3** | ∞ | ∞ | 22 | 0 | 20 | ∞ | ∞ | 16 | 21 |
| **V4** | ∞ | ∞ | ∞ | 20 | 0 | 26 | ∞ | 7 | ∞ |
| **V5** | 11 | ∞ | ∞ | ∞ | 26 | 0 | 17 | ∞ | ∞ |
| **V6** | ∞ | 16 | ∞ | ∞ | ∞ | 17 | 0 | 19 | ∞ |
| **V7** | ∞ | ∞ | ∞ | 16 | 7 | ∞ | 19 | 0 | ∞ |
| **V8** | ∞ | 12 | 8 | 21 | ∞ | ∞ | ∞ | ∞ | 0 |

代码分析论

没有动画演示你说个J8！

Prim.c

// Prim算法生成最小生成树

void MiniSpanTree\_Prim(MGraph G)

{

int min, i, j, k;

int adjvex[MAXVEX]; // 保存相关顶点下标

int lowcost[MAXVEX]; // 保存相关顶点间边的权值

lowcost[0] = 0; // V0作为最小生成树的根开始遍历，权值为0

adjvex[0] = 0; // V0第一个加入

// 初始化操作

for( i=1; i < G.numVertexes; i++ )

{

lowcost[i] = G.arc[0][i]; // 将邻接矩阵第0行所有权值先加入数组

adjvex[i] = 0; // 初始化全部先为V0的下标

}

// 真正构造最小生成树的过程

for( i=1; i < G.numVertexes; i++ )

{

min = INFINITY; // 初始化最小权值为65535等不可能数值

j = 1;

k = 0;

// 遍历全部顶点

while( j < G.numVertexes )

{

// 找出lowcost数组已存储的最小权值

if( lowcost[j]!=0 && lowcost[j] < min )

{

min = lowcost[j];

k = j; // 将发现的最小权值的下标存入k，以待使用。

}

j++;

}

// 打印当前顶点边中权值最小的边

printf("(%d,%d)", adjvex[k], k);

lowcost[k] = 0; // 将当前顶点的权值设置为0，表示此顶点已经完成任务，进行下一个顶点的遍历

// 邻接矩阵k行逐个遍历全部顶点

for( j=1; j < G.numVertexes; j++ )

{

if( lowcost[j]!=0 && G.arc[k][j] < lowcost[j] )

{

lowcost[j] = G.arc[k][j];

adjvex[j] = k;

}

}

}

}