```
#define N 9
typedef struct Vertice{
    int pi;
    int key;
    int id;
    bool flag;
}Vertice;

typedef struct Aresta {
   int u;
   int v;
   int peso;
}Aresta;

Vertice vertices[N];
list<Aresta> Adj[N];

void inicializar_vertices(){
    for(int i=0; i<N; i++)
            vertices[i].id = i;
}

void inicializa_grafo(){
   enum Vertices {a,b,c,d,e,f,g,h,i};
                    //0 1 2 3 4 5 6 7 8
   Adj[a].push_back((Aresta){a,b,4});
   Adj[a].push_back((Aresta){a,h,8});
   Adj[b].push_back((Aresta){b,a,4});
   Adj[b].push_back((Aresta){b,c,8});
   Adj[b].push_back((Aresta){b,h,11});
   Adj[c].push_back((Aresta){c,b,8});
   Adj[c].push_back((Aresta){c,d,7});
   Adj[c].push_back((Aresta){c,f,4});
   Adj[c].push_back((Aresta){c,i,2});
   Adj[d].push_back((Aresta){d,c,7});
   Adj[d].push_back((Aresta){d,e,9});
   Adj[d].push_back((Aresta){d,f,14});
   Adj[e].push_back((Aresta){e,d,9});
   Adj[e].push_back((Aresta){e,f,10});
   Adj[f].push_back((Aresta){f,c,4});
   Adj[f].push_back((Aresta){f,d,14});
   Adj[f].push_back((Aresta){f,e,10});
   Adj[f].push_back((Aresta){f,g,2});
   Adj[g].push_back((Aresta){g,f,2});
   Adj[g].push_back((Aresta){g,h,1});
   Adj[g].push_back((Aresta){g,i,6});
```

```
  Adj[h].push_back((Aresta){h,a,8});
  Adj[h].push_back((Aresta){h,b,11});
  Adj[h].push_back((Aresta){h,g,1});
  Adj[h].push_back((Aresta){h,i,7});
  Adj[i].push_back((Aresta){i,c,2});
  Adj[i].push_back((Aresta){i,g,6});
  Adj[i].push_back((Aresta){i,h,7});

  inicializar_vertices();
}
```

Para imprima os vértices, seus predecessores e a chave, logo após executar o algoritmo, ou seja as arestas da árvore. EXEMPLO :

```
int main(){
   inicializa_grafo();
   AGM_PRIM( 0 );
   for (int i= 1; i<N; i++)
        cout << vertices[i].pi << " - "<< vertices[i].id << " - "<< vertices[i].key << endl;
   return 0;
}
```

RESULTADO : a -> b - 4,
b -> c - 8,
c -> d - 7,
d -> e - 9,
c -> f - 4,
f -> g - 2,
g -> h - 1 e
c -> i - 2.