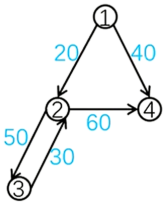
1.邻接矩阵

1. 邻接矩阵
二维数组 $w[u][v]$ 存储从点 u 到点 v 的边的权值

时间复杂度：$O(n^2)$
空间复杂度：$O(n^2)$

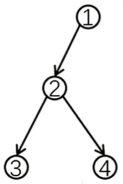
应用：只在点数不多的稠密图 (例 $n=10^3, m=10^6$) 上使用



输入
4 5
1 2 20
1 4 40
2 3 50
2 4 60
3 2 30

0	20	0	40
0	0	50	60
0	30	0	0
0	0	0	0

输出
1,2,20
2,3,50
3,2,30
2,4,60
1,4,40




```
int w[N][N];//边权
int vis[N];

void dfs(int u){
  vis[u]=true;
  for(int v=1;v<=n;v++)
    if(w[u][v]){
      printf("%d,%d,%d\n",u,v,w[u][v]);
      if(vis[v]) continue;
      dfs(v);
    }
}
int main(){
  cin>>n>>m;
  for(int i=1;i<=m;i++){
    cin>>a>>b>>c;
    w[a][b]=c;
    // w[b][a]=c;
  }
  dfs(1);
  return 0;
}
```

代码：

```cpp
#include<bits/stdc++.h>
using namespace std;
//边权
const int N=1e3+10;
int w[N][N];
bool vis[N];
//n个点,m条边.
int n,m;
void dfs(int u){
    vis[u]=true;
    for(int v=1;v<=m;v++){
        //u到v有边
        if(w[u][v]){
            printf("%d到%d的权值为：%d\n",u,v,w[u][v]);
            if(vis[v]) continue;
            dfs(v);
        }
    }
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int a,b,c;
        //a→b的边的权值为c
        cin>>a>>b>>c;
        w[a][b]=c;
        //无向图就加上下面这句.
        //w[b][a]=c;
    }
    dfs(1);
    return 0;
```

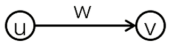
```
}
```

2.边集数组

2. 边集数组

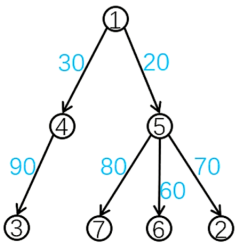
边集数组 $e[i]$ 存储第 i 条边的{起点u,终点v,边权w}

时间复杂度： $O(nm)$
空间复杂度： $O(m)$



应用：在 Kruskal 算法中，需要将边按边权排序，直接存边

输入
7 6
4 3 90
1 4 30
5 7 80
5 6 60
1 5 20
5 2 70



输出
1,4,30
4,3,90
1,5,20
5,7,80
5,6,60
5,2,70

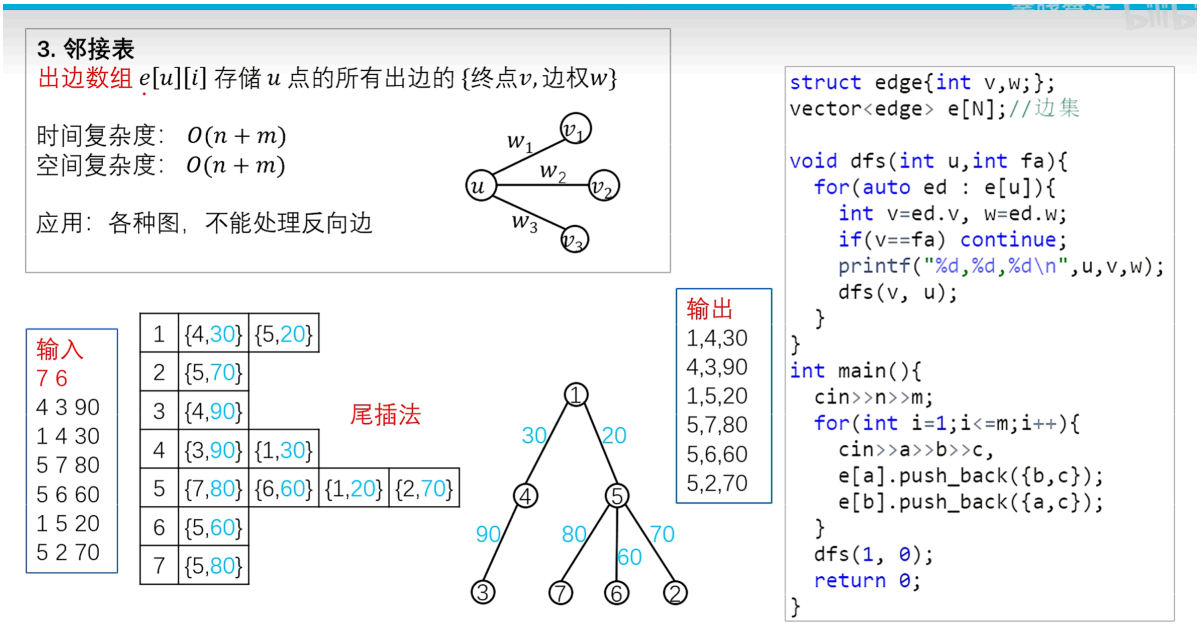
```
struct edge{
  int u,v,w;
}e[M];//边集
int vis[N];

void dfs(int u){
  vis[u]=true;
  for(int i=1;i<=m;i++)
    if(e[i].u==u){
      int v=e[i].v,w=e[i].w;
      printf("%d,%d,%d\n",u,v,w);
      if(vis[v]) continue;
      dfs(e[i].v);
    }
}
int main(){
  cin>>n>>m;
  for(int i=1;i<=m;i++){
    cin>>a>>b>>c;
    e[i]={a,b,c};
    // e[i]={b,a,c};
  }
  dfs(1);
  return 0;
}
```

代码：

```
#include<bits/stdc++.h>
using namespace std;
//边权
const int N=1e6+10;
struct node{
    int u,v,w;
}e[N];
int vis[N];
int n,m;
void dfs(int u){
    vis[u]=true;
    for(int i=1;i<=m;i++){
        //第i条边的起点为u
        if(e[i].u==u){
            int v=e[i].v,w=e[i].w;
            printf("%d到%d的权值为:%d\n",u,v,w);
            if(vis[v]) continue;
            dfs(v);
        }
    }
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int a,b,c;
        //a→b的边的权值为c
        cin>>a>>b>>c;
        e[i]={a,b,c};
```

```
        //e[i]={b,a,c};
    }
    dfs(1);
    return 0;
}
```

3.邻接表



缺点：

没有存储边的编号，无法处理网络流中需要查找边的编号的问题.

父节点判重做法：

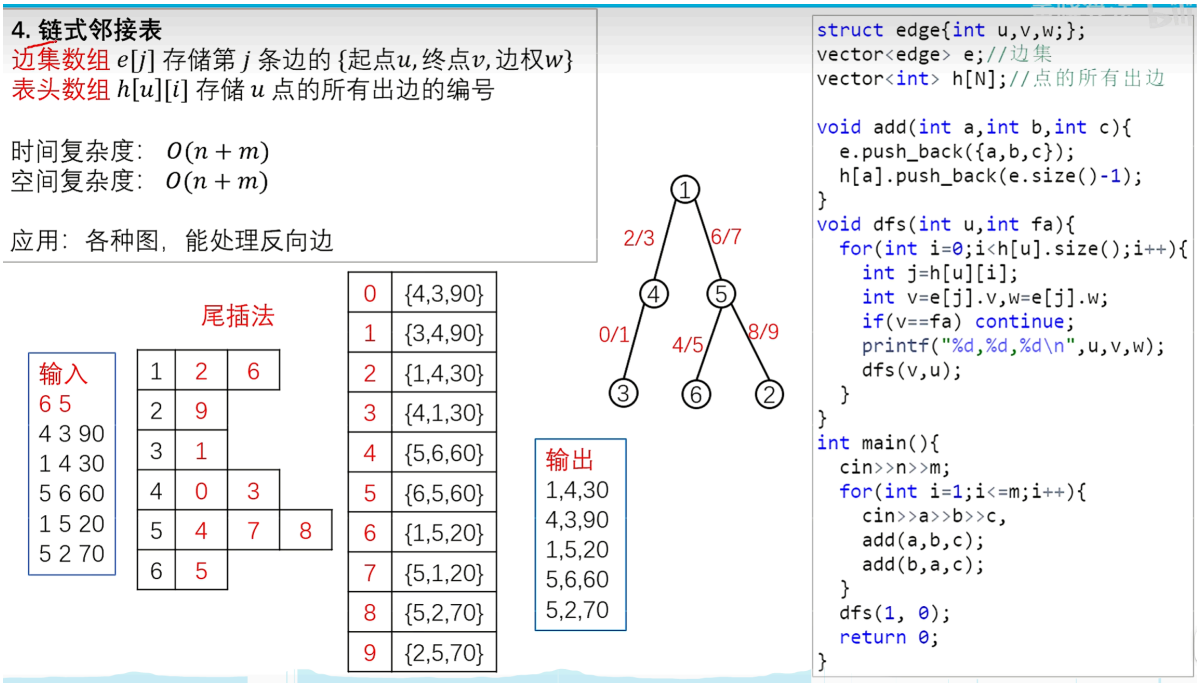
```
#include<bits/stdc++.h>
using namespace std;
//边权
const int N=1e6+10;
struct edge{
    int v,w;
};
vector<edge> e[N];
int vis[N];
int n,m;
void dfs(int u,int fa){
    for(auto ed:e[u]){
        int v=ed.v,w=ed.w;
        if(v==fa) continue;
        printf("%d到%d的权值为:%d\n",u,v,w);
        dfs(v,u);
    }
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int a,b,c;
```

```
        //a→b的边的权值为c
        cin>>a>>b>>c;
        e[a].push_back({b,c});
        e[b].push_back({a,c});
    }
    dfs(1,0);
    return 0;
}
```

vis判重做法：

```
#include<bits/stdc++.h>
using namespace std;
//边权
const int N=1e6+10;
struct edge{
    int v,w;
};
vector<edge> e[N];
int vis[N];
int n,m;
void dfs(int u){
    vis[u]=true;
    for(auto ed:e[u]){
        int v=ed.v,w=ed.w;
        if(vis[v]) continue;
        printf("%d到%d的权值为:%d\n",u,v,w);
        dfs(v);
    }
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int a,b,c;
        //a→b的边的权值为c
        cin>>a>>b>>c;
        e[a].push_back({b,c});
        e[b].push_back({a,c});
    }
    dfs(1);
    return 0;
}
```

4.链式邻接表



代码：

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,m;
const int N=1e4+10;
struct edge{
    int u,v,w;
};
vector<edge> e;
vector<int> h[N];
void add(int a,int b,int c){
    e.push_back({a,b,c});
    //边的编号从0开始，若从1开始则不需要-1
    h[a].push_back(e.size()-1);
}
void dfs(int u,int fa){
    for(int i=0;i<h[u].size();i++){
        //第j条边
        int j=h[u][i];
        int v=e[j].v,w=e[j].w;
        if(v==fa) continue;
        printf("%d到%d的权值为:%d\n",u,v,w);
        dfs(v,u);
    }
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
        //无向边
```

```
        add(b,a,c);
    }
    dfs(1,0);
    return 0;
}
```

5.链式前向星

链式前向星本质上是在用普通数组来模拟邻接表的过程。

```
#include<bits/stdc++.h>
using namespace std;
const int N=510,M=3000;
int n,m,a,b,c;
struct edge{int v,w,ne;};
edge e[M];//边集
int idx,h[N];//点的第一条出边

void add(int a,int b,int c){
  e[idx]={b,c,h[a]};
  h[a]=idx++;
}
void dfs(int u,int fa){
  for(int i=h[u];~i;i=e[i].ne){
    int v=e[i].v, w=e[i].w;
    if(v==fa) continue;
    printf("%d,%d,%d\n",u,v,w);
    dfs(v,u);
  }
}
int main(){
  cin>>n>>m;
  memset(h,-1,sizeof h);
  for(int i=1;i<=m;i++){
    cin>>a>>b>>c,
    add(a,b,c);
    add(b,a,c);
  }
  dfs(1, 0);
  return 0;
}
```