

1.邻接矩阵

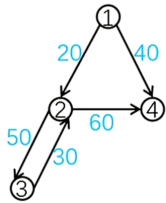
1. 邻接矩阵

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: $O(n^2)$

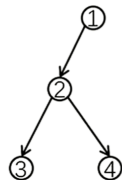
应用: 只在点数不多的稠密图 (例 $n = 10^3, m = 10^6$) 上使用



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:

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  //边权
4  const int N=1e3+10;
5  int w[N][N];
6  bool vis[N];
7  //n个点,m条边.
8  int n,m;
9  void dfs(int u){
10     vis[u]=true;
11     for(int v=1;v<=m;v++){
12         //u到v有边
13         if(w[u][v]){
14             printf("%d到%d的权值为: %d\n",u,v,w[u][v]);
15             if(vis[v]) continue;
16             dfs(v);
17         }
18     }
19 }
20 int main()
21 {
22     cin>>n>>m;
23     for(int i=1;i<=m;i++){
24         int a,b,c;
25         //a->b的边的权值为c
26         cin>>a>>b>>c;
27         w[a][b]=c;
28         //无向图就加上下面这句.
29         //w[b][a]=c;
30     }
31     dfs(1);
32     return 0;
}
```

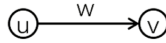
2. 边集数组

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: $O(m)$

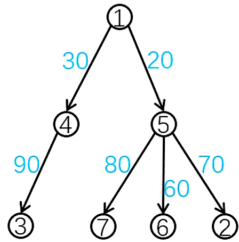


应用: 在 Kruskal 算法中, 需要将边按边权排序, 直接存边

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:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  //边权
4  const int N=1e6+10;
5  struct node{
6      int u,v,w;
7  }e[N];
8  int vis[N];
9  int n,m;
10 void dfs(int u){
11     vis[u]=true;
12     for(int i=1;i<=m;i++){
13         //第i条边的起点为u
14         if(e[i].u==u){
15             int v=e[i].v,w=e[i].w;
16             printf("%d到%d的权值为:%d\n",u,v,w);
17             if(vis[v]) continue;
18             dfs(v);
19         }
20     }
21 }
22 int main()
23 {
24     cin>>n>>m;
25     for(int i=1;i<=m;i++){
26         int a,b,c;
27         //a->b的边的权值为c
28         cin>>a>>b>>c;
29         e[i]={a,b,c};
  
```

```

30     //e[i]={b,a,c};
31 }
32 dfs(1);
33 return 0;
34 }

```

3.邻接表

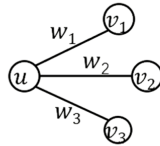
3. 邻接表

出边数组 $e[u][i]$ 存储 u 点的所有出边的 {终点 v , 边权 w }

时间复杂度: $O(n + m)$

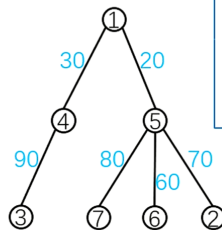
空间复杂度: $O(n + m)$

应用: 各种图, 不能处理反向边



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

尾插法



输出

```

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

```

```

struct edge{int v,w};
vector<edge> e[N]; //边集

void dfs(int u,int fa){
    for(auto ed : e[u]){
        int v=ed.v, w=ed.w;
        if(v==fa) continue;
        printf("%d,%d,%d\n",u,v,w);
        dfs(v, u);
    }
}

int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>a>>b>>c;
        e[a].push_back({b,c});
        e[b].push_back({a,c});
    }
    dfs(1, 0);
    return 0;
}

```

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:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  //边权
4  const int N=1e6+10;
5  struct edge{
6      int v,w;
7  };
8  vector<edge> e[N];
9  int vis[N];
10 int n,m;
11 void dfs(int u,int fa){
12     for(auto ed:e[u]){
13         int v=ed.v,w=ed.w;
14         if(v==fa) continue;
15         printf("%d到%d的权值为:%d\n",u,v,w);
16         dfs(v,u);
17     }
18 }
19 int main()
20 {
21     cin>>n>>m;
22     for(int i=1;i<=m;i++){
23         int a,b,c;

```

```

24     //a→b的边的权值为c
25     cin>>a>>b>>c;
26     e[a].push_back({b,c});
27     e[b].push_back({a,c});
28 }
29 dfs(1,0);
30 return 0;
31 }

```

vis判重做法:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  //边权
4  const int N=1e6+10;
5  struct edge{
6      int v,w;
7  };
8  vector<edge> e[N];
9  int vis[N];
10 int n,m;
11 void dfs(int u){
12     vis[u]=true;
13     for(auto ed:e[u]){
14         int v=ed.v,w=ed.w;
15         if(vis[v]) continue;
16         printf("%d到%d的权值为:%d\n",u,v,w);
17         dfs(v);
18     }
19 }
20 int main()
21 {
22     cin>>n>>m;
23     for(int i=1;i<=m;i++){
24         int a,b,c;
25         //a→b的边的权值为c
26         cin>>a>>b>>c;
27         e[a].push_back({b,c});
28         e[b].push_back({a,c});
29     }
30     dfs(1);
31     return 0;
32 }

```

4.链式邻接表

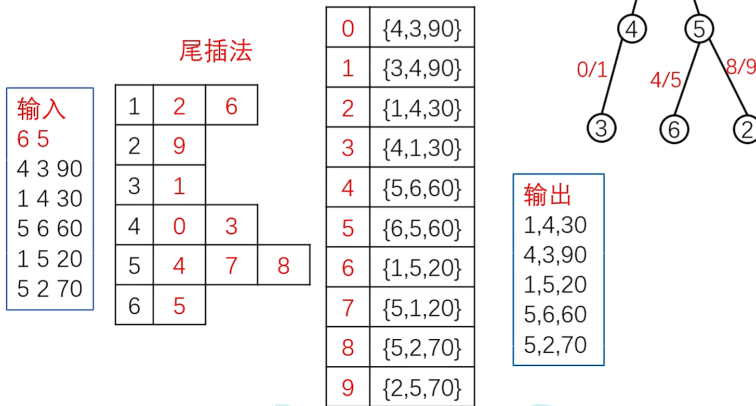
4. 链式邻接表

边集数组 $e[j]$ 存储第 j 条边的 {起点 u , 终点 v , 边权 w }
表头数组 $h[u][i]$ 存储 u 点的所有出边的编号

时间复杂度: $O(n + m)$

空间复杂度: $O(n + m)$

应用: 各种图, 能处理反向边



```
struct edge{int u,v,w;};
vector<edge> e;//边集
vector<int> h[N];//点的所有出边

void add(int a,int b,int c){
    e.push_back({a,b,c});
    h[a].push_back(e.size()-1);
}

void dfs(int u,int fa){
    for(int i=0;i<h[u].size();i++){
        int j=h[u][i];
        int v=e[j].v,w=e[j].w;
        if(v==fa) continue;
        printf("%d,%d,%d\n",u,v,w);
        dfs(v,u);
    }
}

int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>a>>b>>c;
        add(a,b,c);
        add(b,a,c);
    }
    dfs(1, 0);
    return 0;
}
```

代码:

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int n,m;
4  const int N=1e4+10;
5  struct edge{
6      int u,v,w;
7  };
8  vector<edge> e;
9  vector<int> h[N];
10 void add(int a,int b,int c){
11     e.push_back({a,b,c});
12     //边的编号从0开始, 若从1开始则不需要-1
13     h[a].push_back(e.size()-1);
14 }
15 void dfs(int u,int fa){
16     for(int i=0;i<h[u].size();i++){
17         //第j条边
18         int j=h[u][i];
19         int v=e[j].v,w=e[j].w;
20         if(v==fa) continue;
21         printf("%d到%d的权值为:%d\n",u,v,w);
22         dfs(v,u);
23     }
24 }
25 int main()
26 {
27     cin>>n>>m;
28     for(int i=1;i<=m;i++){
29         int a,b,c;
30         cin>>a>>b>>c;
31         add(a,b,c);
32         //无向边
```

```

33         add(b,a,c);
34     }
35     dfs(1,0);
36     return 0;
37 }

```

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。

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=510,M=3000;
4  int n,m,a,b,c;
5  struct edge{int v,w,ne;};
6  edge e[M]; //边集
7  int idx,h[N]; //点的第一条出边
8
9  void add(int a,int b,int c){
10     e[idx]={b,c,h[a]};
11     h[a]=idx++;
12 }
13 void dfs(int u,int fa){
14     for(int i=h[u];~i;i=e[i].ne){
15         int v=e[i].v, w=e[i].w;
16         if(v==fa) continue;
17         printf("%d,%d,%d\n",u,v,w);
18         dfs(v,u);
19     }
20 }
21 int main(){
22     cin>>n>>m;
23     memset(h,-1,sizeof h);
24     for(int i=1;i<=m;i++){
25         cin>>a>>b>>c,
26         add(a,b,c);
27         add(b,a,c);
28     }
29     dfs(1, 0);
30     return 0;
31 }

```