图的基本应用

1. 图的存储:

**邻接矩阵**

#include<bits/stdc++.h>  
using namespace std;  
#define MAXN 1005 //图的最大点数量  
int n;  
int v[MAXN][MAXN];  
int main(){  
 cin>>n;  
 for(int i=1;i<=n;++i)  
 for(int j=1;j<=n;++j)  
 cin>>v[i][j];//读入邻接矩阵  
 //下面的代码将找到与点i有直接连接的每一个点以及那条边的长度  
 for(int i=1;i<=n;++i)  
 for(int j=1;j<=n;++j)  
 if(v[i][j]>0)  
 cout<<"edge from point"<<i<<"to point"<<j<<"with length"<<v[i][j]<<'\n';  
 return 0;  
}

**邻接表**

输入一个有向图的点数、边数、每一条边的起点终点边权，输出有向图的邻接矩阵，代码如下：

#include<bits/stdc++.h>  
using namespace std;  
#define MAXN 1005  
struct edge{//记录边的终点，边权的结构体   
 int to,cost;  
};  
int n,m;//表示图中有n个点m条边  
vector<edge> p[MAXN];  
int v[MAXN][MAXN];  
int main() {  
 cin>>n>>m;  
 for(int i=1;i<=m;i++){  
 int u,v,l;  
 cin>>u>>v>>l;  
 p[u].push\_back((edge){v,l});  
 /\*p[v].push\_back((edge){u,l});//当用无向图的邻接表时需要加一条反方向的边\*/  
 }  
 //把邻接表转换为邻接矩阵  
 for(int i=1;i<=n;i++)  
 for(int j=0;j<p[i].size();j++)  
 v[i][p[i][j].to]=p[i][j].cost;  
 //输出邻接矩阵  
 for(int i=1;i<=n;i++) {  
 for(int j=1;j<=n;j++)  
 cout<<v[i][j]<<' ';  
 cout<<'\n';  
 }  
 return 0;  
}

1. 图的遍历

假设博客里面一共有n(n<=10^5)篇文章(编号为1到n)以及m(m<=10^6)条参考文献引用关系。设计一种方法，可以不重复、不遗漏地看完所有能看到的文章。

**思路1：深度优先遍历**

代码如下：

#include<bits/stdc++.h>  
using namespace std;  
#define MAXN 100005  
int n,m;  
vector<int> p[MAXN];  
bool u[MAXN];  
void solve(int x){  
 cout<<x<<'';//此时输出的就是看文献的顺序  
 for(int i=0,sz=p[x].size();i<sz;i++)  
 if(!u[p[x][i]]){  
 u[p[x][i]]=true;  
 solve(p[x][i]);  
 }  
}  
int main() {  
 cin>>n>>m;  
 for(int i=1;i<=m;i++){  
 int x,y;  
 cin>>x>>y;  
 p[x].push\_back(y);//用邻接表记录下文献x有参考文献y   
 }  
 u[1]=true;  
 solve(1);  
 return 0;  
}

**思路2：广度优先遍历**

代码如下：

#include<bits/stdc++.h>  
using namespace std;  
#define MAXN 100005  
int n,m;  
vector<int> p[MAXN];  
queue<int>q;  
bool u[MAXN];  
int main() {  
 cin>>n>>m;  
 for(int i=1;i<=m;i++){  
 int x,y;  
 cin>>x>>y;  
 p[x].push\_back(y);//用邻接表记录下文献x有参考文献y   
 }  
 u[1]=true;  
 q.push(1);  
 while(!q.empty()){  
 int x=q.front();  
 q.pop();  
 cout<<x<<'';  
 for(int i=0,sz=p[x].size();i<sz;i++)  
 if(!u[p[x][i]]){  
 u[p[x][i]]=true;  
 q.push(p[x][i]);  
 }  
 }  
 return 0;  
}

**深度优先遍历可以被看作在图中的深度优先搜索，广度优先遍历可以被看作在图中的广度优先搜索。**