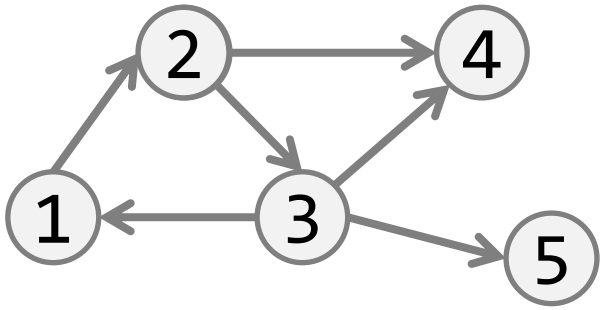
C++算法

图的储存

有向无权图

输入
5 6
1 2
3 4
2 3
3 5
2 4
3 1



有向图
共5个节点
6条有向边

图的储存方式

邻接矩阵

邻接表

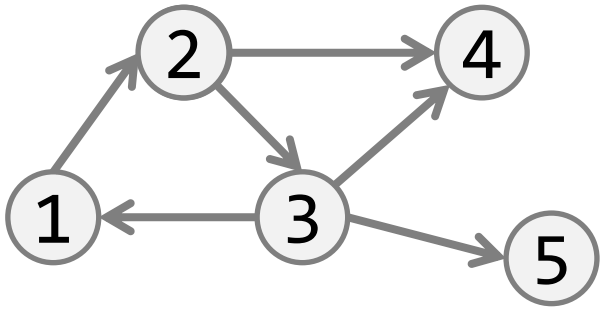
边集数组

前向星

链式前向星

邻接矩阵

输入
5 6
1 2
3 4
2 3
3 5
2 4
3 1

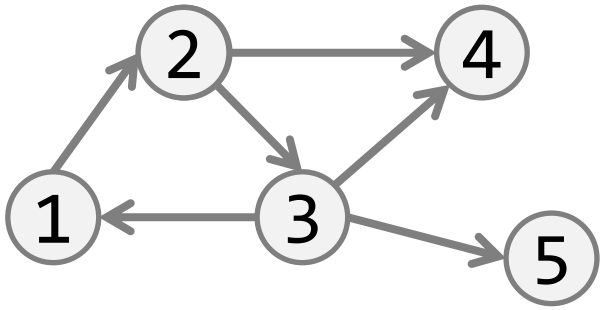


	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	1	0
3	1	0	0	1	1
4	0	0	0	0	0
5	0	0	0	0	0

缺点	稀疏图里 空间浪费

邻接表1

输入
5 6
1 2
3 4
2 3
3 5
2 4
3 1



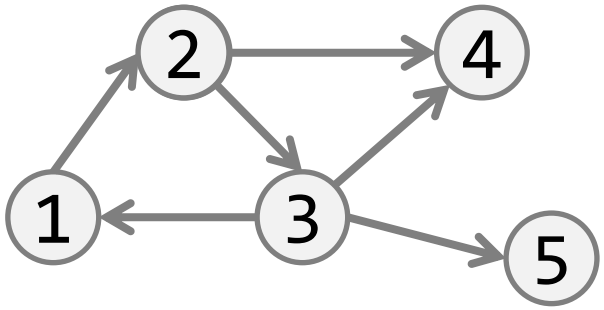
1	{1,2}		
2	{2,3}	{2,4}	
3	{3,4}	{3,5}	{3,1}
4			
5			

```
5 struct edge{int frm,to;};
6 vector<edge> e[N];
7 void add(int u,int v){
8     e[u].push_back((edge){u,v});
9 }
```

发现	edge类型里frm属性冗余

邻接表3

输入
5 6
1 2
3 4
2 3
3 5
2 4
3 1



1	2		
2	3	4	
3	1	4	5
4			
5			

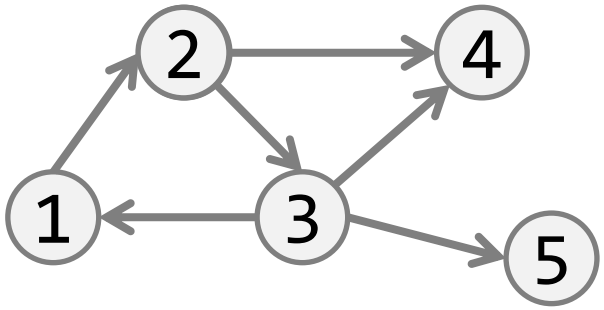
```
5  vector<int> to[N];
6  void add(int u,int v){
7      to[u].push_back(v);
8  }
```

特点	邻居编号从小到大

```
19      for(int u=1;u<=n;++u)
20          sort(to[u].begin(),to[u].end());
```

边集数组

输入
5 6
1 2
3 4
2 3
3 5
2 4
3 1



1	2	3	4	5	6
{1,2}	{3,4}	{2,3}	{3,5}	{2,4}	{3,1}

```
 5 struct edge{int frm,to;};
 6 edge e[M];
 7 int nE;
 8 void add(int u,int v){
 9     ++nE;
10     e[nE]=(edge){u,v};
11 }
```

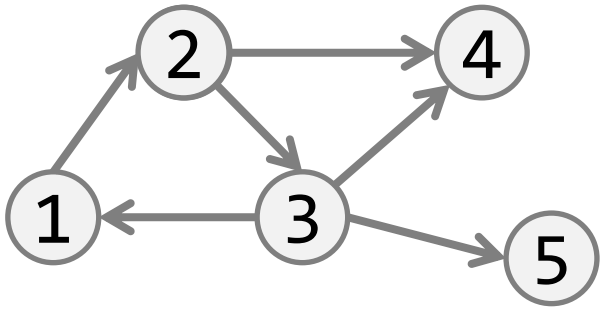
缺点	访问邻居不方便

对边排序
相同起点聚集

前向星

对边排序
相同起点聚集

输入
5 6
1 2
3 4
2 3
3 5
2 4
3 1



1	2	3	4	5	6
{1,2}	{2,3}	{2,4}	{3,1}	{3,4}	{3,5}

```
12 bool cmp(const edge&a,const edge&b){
13     return a.frm<b.frm||a.frm==b.frm&&a.to<b.to;
14 }
```

```
20     for(int i=1;i<=m;++i){
21         cin>>u>>v;
22         add(u,v);
23         ++dgr[u];
24     }
25     sort(e+1,e+1+m,cmp);
26     hd[1]=1;
27     for(int u=2;u<=n;++u)
28         hd[u]=hd[u-1]+dgr[u-1];
```

dgr[u]:节点u的出度

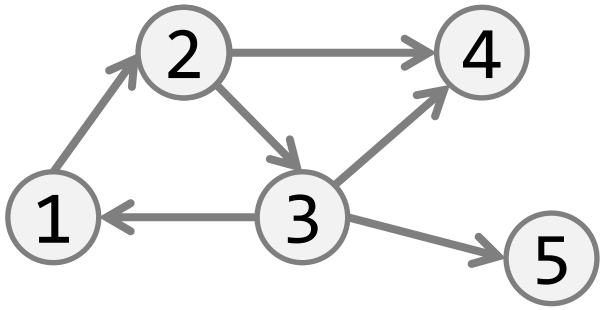
hd[u]:节点u为起点的边在边集数组里从几号开始

缺点：排序复杂度 O(ElogE)

前向星

对边排序
相同起点聚集

输入
5 6
1 2
3 4
2 3
3 5
2 4
3 1

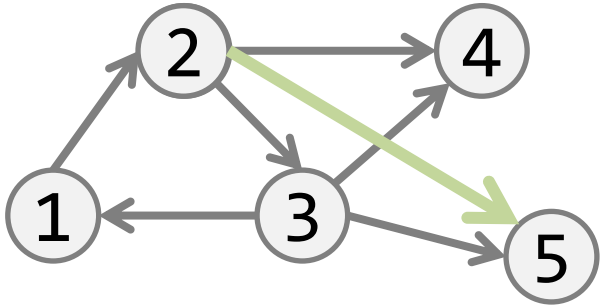


	1	2	3	4	5	6
	{1,2}	{2,3}	{2,4}	{3,1}	{3,4}	{3,5}

hd[]	1	2	4	0	0
dgr[]	1	2	3	0	0

dgr[u]:节点u的出度

hd[u]:节点u为起点的边在边集数组里从几号开始

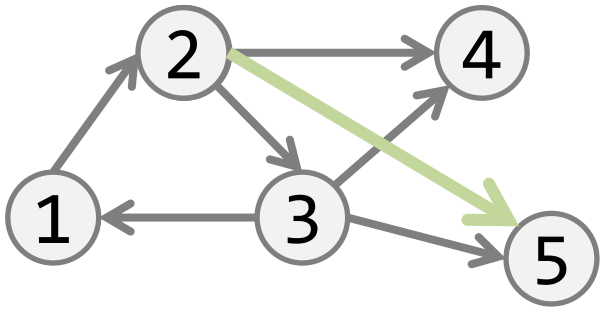


链式前向星1
增加一条边 2->5

	1	2	3	4	5	6	7
	{1,2}	{3,4}	{2,3}	{3,5}	{2,4}	{3,1}	{2,5}
边的nxt属性	0	0	0	2	3	4	5

hd[]	1	2	3	4	5
	1	5	6	0	0

```
 8  void add(int u,int v){
 9      ++nE;
10      e[nE]=(edge){u,v,hd[u]};
11
12  }
```



链式前向星1

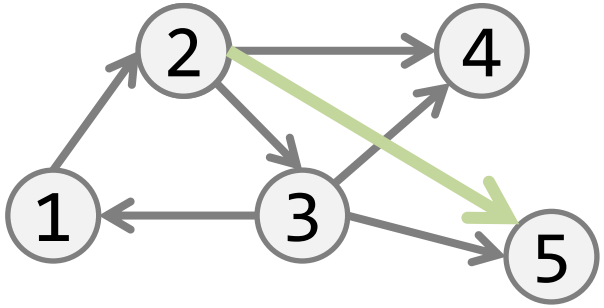
增加一条边
2->5

	1	2	3	4	5	6	7
	{1,2}	{3,4}	{2,3}	{3,5}	{2,4}	{3,1}	{2,5}
边的nxt属性	0	0	0	2	3	4	5

hd[]	1	2	3	4	5
	1	7	6	0	0

```
 8  void add(int u,int v){
 9      ++nE;
10      e[nE]=(edge){u,v,hd[u]};
11      hd[u]=nE;
12  }
```

发现 edge类型里frm属性冗余



链式前向星2

增加一条边
2->5

	1	2	3	4	5	6	7
	2	4	3	5	4	1	5
边的nxt属性	0	0	0	2	3	4	5

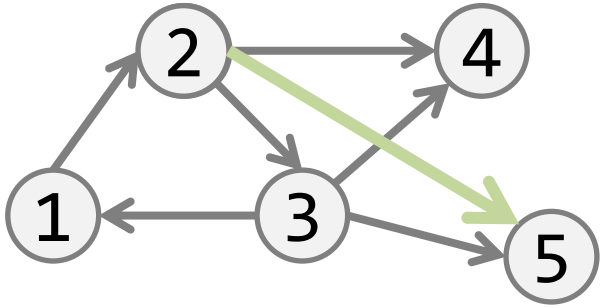
hd[]	1	2	3	4	5
	1	7	6	0	0

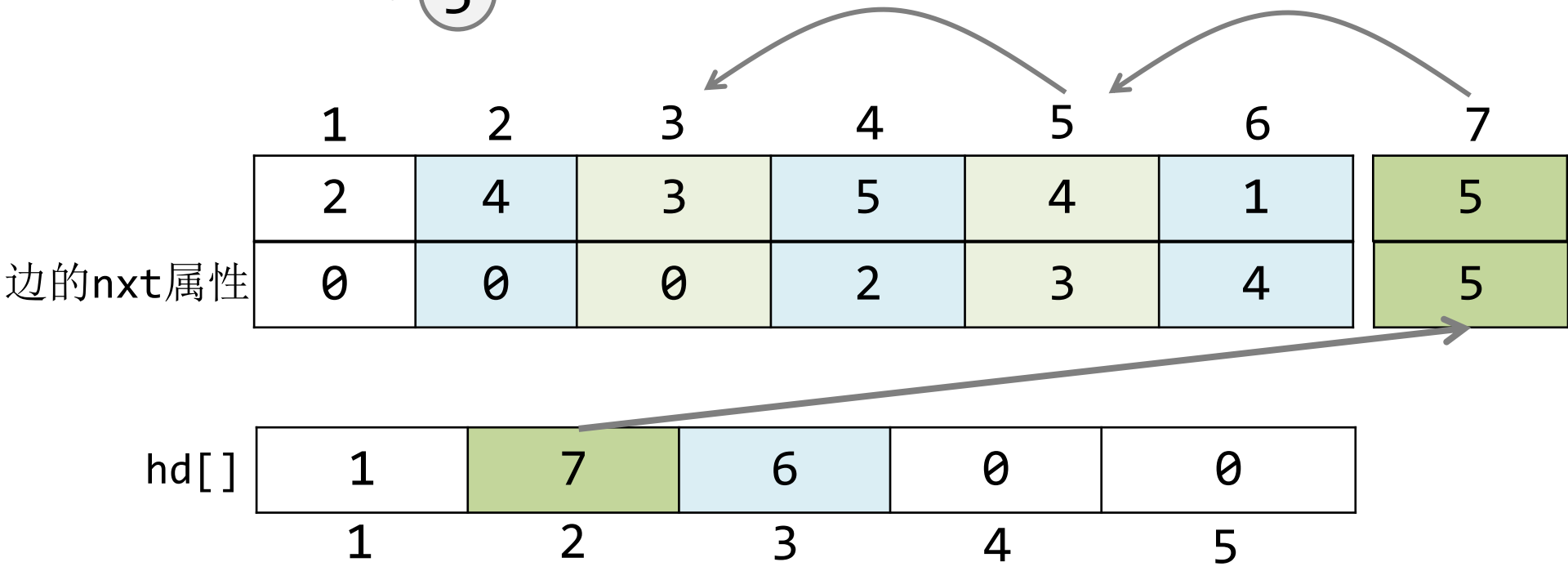
```
5  struct edge{int to,nxt;};
6  edge e[M];
7  int hd[N],nE;
```

```
8  void add(int u,int v){
9      ++nE;
10     e[nE]=(edge){v,hd[u]};
11     hd[u]=nE;
12 }
```

链式前向星2

访问邻居






```
23      for(int u=1;u<=n;++u,cout<<endl)
24          for(int i=hd[u];i;i=e[i].nxt)
25              cout<<u<<"->"<<e[i].to<<" ";
```

链式前向星

基于边集数组
相同起点的边加入链表

hd[u]:节点u为起点的边在边集数组里从几号开始

```
 5 struct edge{int to,nxt;};
 6 edge e[M];
 7 int hd[N],nE;
 8 void add(int u,int v){
 9     ++nE;
10     e[nE]=(edge){v,hd[u]};
11     hd[u]=nE;
12 }
```

链式前向星

基于边集数组
相同起点的边加入链表

hd[u]:节点u为起点的边在边集数组里从几号开始

```
5  struct edge{int to,nxt;};
6  edge e[M];
7  int hd[N],nE;
```

```
23      for(int u=1;u<=n;++u,cout<<endl)
24          for(int i=hd[u];i;i=e[i].nxt)
25              cout<<u<<"->"<<e[i].to<<" ";
```

对于节点u
访问u所有的邻居
需要枚举边集数组里编号i的边

要求	2种方法求解：邻接表,链式前向星

现场挑战
快快编程590

SPFA+邻接表

```
24  const ll N=509;
25  const ll INF=1e9;
26  struct edge{int to,w;};
27  vector<edge> e[N];
```

```
56      cin>>n>>p>>m;
57      for(ll i=0;i<p;++i){
58          ll u,v,w;
59          cin>>u>>v>>w;
60          e[u].push_back((edge){v,-w});
61      }
```

```
62      for(ll i=0;i<m;++i){
63          ll u,v,w;
64          cin>>u>>v>>w;
65          e[u].push_back((edge){v,w});
66          e[v].push_back((edge){u,w});
67      }
```

```
68      ll ans=SPFA();
69      if(ans)cout<<"Yes"<<endl;
70      else cout<<"No"<<endl;
```

SPFA+邻接表

```
29 bool SPFA(){
30     fill(d,d+n+1,INF);
31     queue<int> q;
32     q.push(1);
33     d[1]=0; in[1]=1; cnt[1]=1;
34     while(!q.empty()){
35         int u=q.front(); q.pop();
36         in[u]=0;
37         for(int i=0;i<e[u].size();++i){
38             int v=e[u][i].to;
39             int w=e[u][i].w;
40             int cost=d[u]+w;
41             if(cost>=d[v])continue;
42             if(!in[v])q.push(v);
43             d[v]=cost; in[v]=1; cnt[v]=cnt[u]+1;
44             if(cnt[v]>n)return 1;
45         }
46     }
47     return 0;
48 }
```

cnt[v]:起点到v当前最短路经过几个节点

SPFA+链式前向星

```
37 bool SPFA(){
38     fill(d,d+n+1,INF);
39     queue<int> q;
40     q.push(1);
41     d[1]=0; in[1]=1; cnt[1]=1;
42     while(!q.empty()){
43         int u=q.front(); q.pop();
44         in[u]=0;
45         for(int i=hd[u];i;i=e[i].nxt){
46             int v=e[i].to;
47             int w=e[i].w;
48             int cost=d[u]+w;
49             if(cost>=d[v])continue;
50             if(!in[v])q.push(v);
51             d[v]=cost; in[v]=1; cnt[v]=cnt[u]+1;
52             if(cnt[v]>n)return 1;
53         }
54     }
55     return 0;
56 }
```

cnt[v]:起点到v当前最短路经过几个节点

SPFA+链式前向星

```
24  const ll N=509;
25  const ll M=2509;
26  const ll P=209;
27  const ll INF=1e9;
28  struct edge{int to,w,nxt;};
29  edge e[P+M*2];
30  ll hd[N],nE;
```

高频易错点
前向星使用的边集数组 对无向边忘记大小翻倍

链式前向星

为什么要学？

实际运行速度
比邻接表vector版本更快

虽然时间复杂度没有提升
但是优化了计算量的**常数**

常数优化

计算量5*mn和计算量2*mn速度不同

计算量5*EV和计算量2*EV速度不同

邻接表

#12 AC	#13 AC	#14 AC
39ms / 11.5MB	47ms / 11.5MB	16ms / 11.5MB

链式前向星

#12 AC	#13 AC	#14 AC
29ms / 11.6MB	38ms / 11.6MB	15ms / 11.6MB

同等级时间复杂度的2个算法
计算速度也有快慢区别

快快编程作业

590
943
1685

要求	使用链式前向星