# 链式前向星及其简单应用

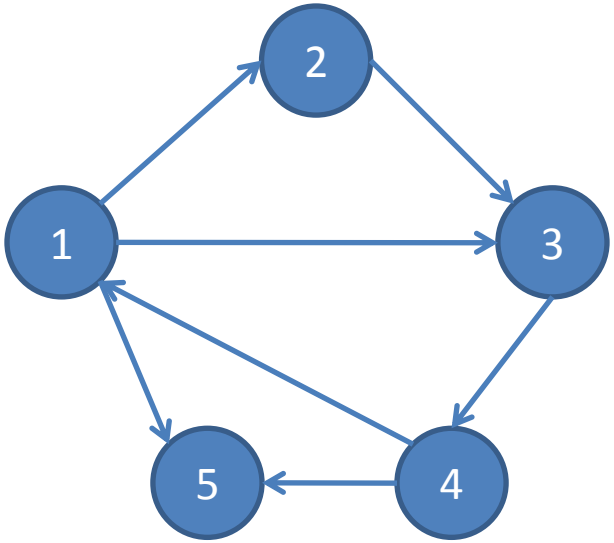
Malash

# 链式前向星及其简单应用

- 链式前向星是图的一种存储方式，采用数组模拟链表实现前向星的功能，但可拓展功能比前向星多。
- 首先，我们先了解一下“前向星”。

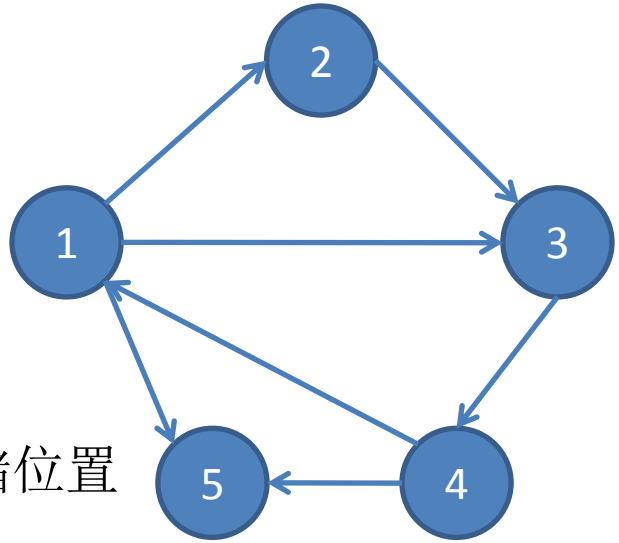
# 链式前向星及其简单应用

- 前向星是一个特殊的边集数组，我们将边集数组中的每条边按照起点排序（必要时起点相同的边再按终点排序），并记录下以某个点为起点的所有边的在数组中的起始位置和存储长度，前向星就构造好了。

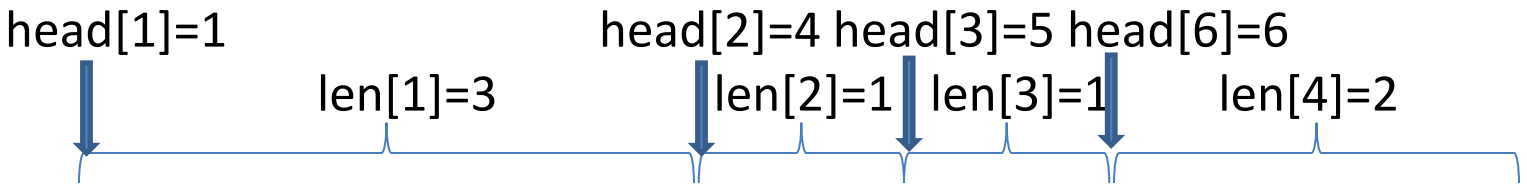


# 链式前向星及其简单应用

- 例如对于此图：



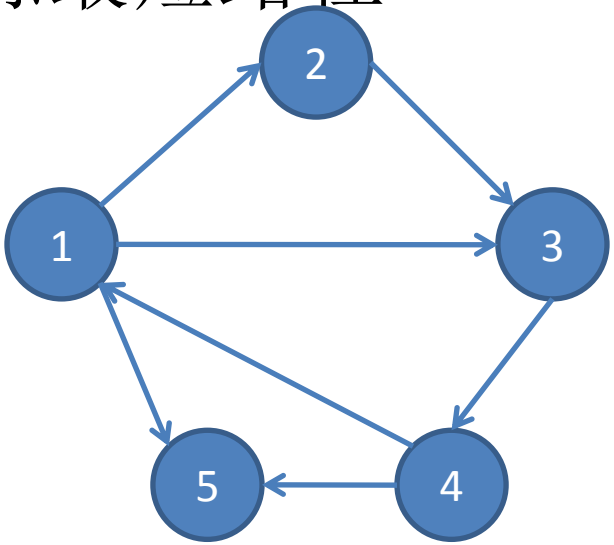

len[i]记录所有以i为起点的边在数组中的存储长度
head[i]记录以i为起点的边集在数组中的第一个存储位置




| 编号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 起点u | 1 | 1 | 1 | 2 | 3 | 4 | 4 |
| 终点v | 2 | 3 | 5 | 3 | 4 | 1 | 5 |

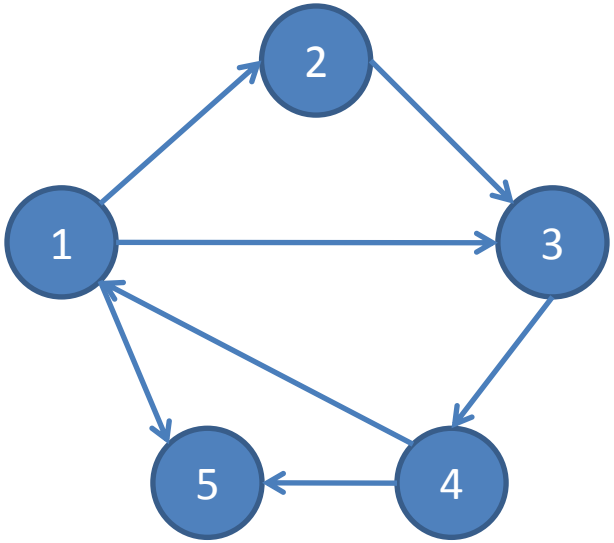
# 链式前向星及其简单应用

- 利用前向星，我们可以以O(1)的时间找到以i为起点的第一条边，以O(len[i])的时间找到以i为起点的所有边。
- 前向星特别适合用来优化稀疏图的深度优先搜索、广度优先搜索、单源最短路径（SPFA）。



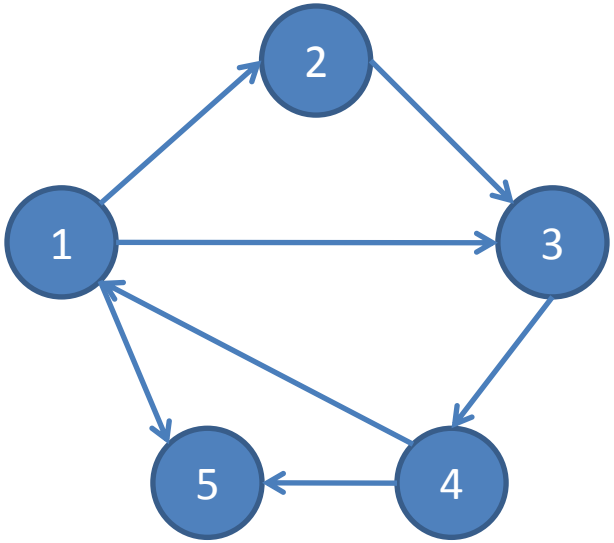
# 链式前向星及其简单应用

- 但对所有边按起点排序，以快排计算，至少为O(eloge)的时间复杂度，对于最好时间复杂度为O(ke),k≈2的SPFA来说，可能优化效果不明显或适得其反。
- 有没有更高效的前向星？
- 有，链式前向星



# 链式前向星及其简单应用

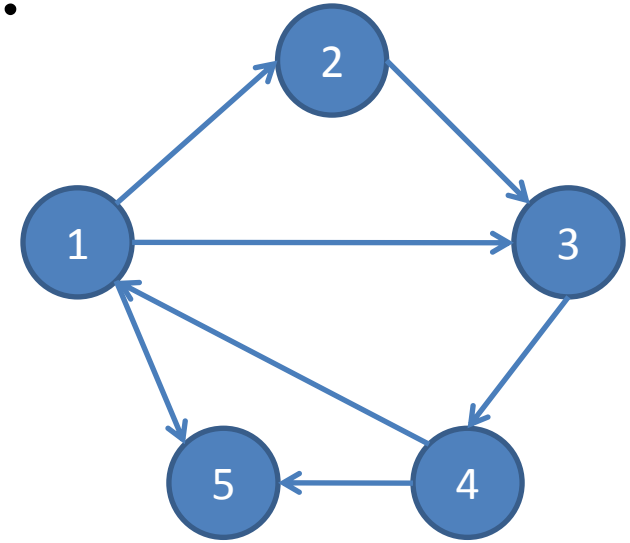
- 我们考察前向星与邻接链表可以发现，前向星的构造主要耗时在频繁的交换（如果使用$O(n^2)$插入排序则更明显），如果我们将链表引入前向星，则可以不使用排序（交换）也得到同样的效果。



# 链式前向星及其简单应用

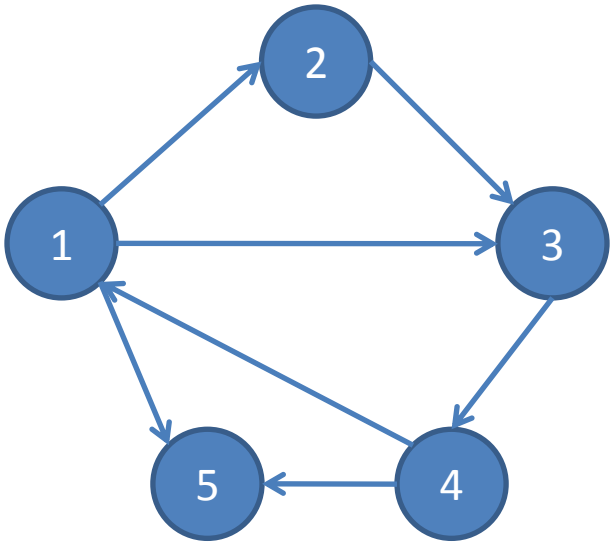
- 我们用head[i]表示以i为起点的第一条边的存储位置，next[i]表示与第i条边同起点的下一条边的存储位置，e[i]表示第i条边的终点。
- 例如对于此图，我们可以这样存储
- 若输入顺序为（起点 终点）：

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



# 链式前向星及其简单应用

- 存储数据如下：




| head[1]=6 | head[2]=2 | head[3]=3 | head[4]=7 | head[5]=0 |
|---|---|---|---|---|

| e[1]=2 | e[1]=2 | e[3]=4 | e[4]=3 | e[5]=1 | e[6]=5 | e[7]=5 |
|---|---|---|---|---|---|---|
| next[1]=0 | next[1]=0 | next[3]=0 | next[4]=1 | next[5]=0 | next[6]=4 | next[7]=5 |

# 链式前向星及其简单应用

- 为了方便理解，我们介绍完查询后再讲构造
- 找出所有起点为1的边

| head[1]=6 | head[2]=2 | head[3]=3 | head[4]=7 | head[5]=0 |
|---|---|---|---|---|

| e[1]=2 | e[1]=2 | e[3]=4 | e[4]=3 | e[5]=1 | e[6]=5 | e[7]=5 |
|---|---|---|---|---|---|---|
| next[1]=0 | next[1]=0 | next[3]=0 | next[4]=1 | next[5]=0 | next[6]=4 | next[7]=5 |

# 链式前向星及其简单应用

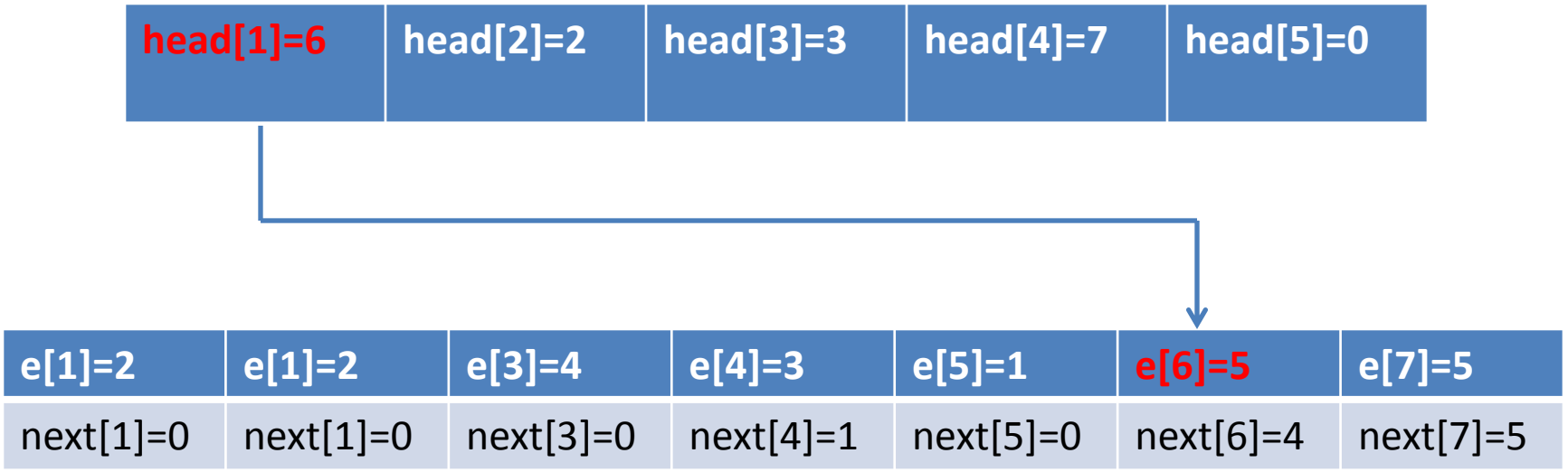
- 找出所有起点为1的边

| head[1]=6 | head[2]=2 | head[3]=3 | head[4]=7 | head[5]=0 |
|---|---|---|---|---|

| e[1]=2 | e[1]=2 | e[3]=4 | e[4]=3 | e[5]=1 | e[6]=5 | e[7]=5 |
|---|---|---|---|---|---|---|
| next[1]=0 | next[1]=0 | next[3]=0 | next[4]=1 | next[5]=0 | next[6]=4 | next[7]=5 |

# 链式前向星及其简单应用

- 找出所有起点为1的边
- 输出e[6]=5



# 链式前向星及其简单应用

- 找出所有起点为1的边
- 输出e[4]=3

| head[1]=6 | head[2]=2 | head[3]=3 | head[4]=7 | head[5]=0 |
|---|---|---|---|---|

| e[1]=2 | e[1]=2 | e[3]=4 | e[4]=3 | e[5]=1 | e[6]=5 | e[7]=5 |
|---|---|---|---|---|---|---|
| next[1]=0 | next[1]=0 | next[3]=0 | next[4]=1 | next[5]=0 | next[6]=4 | next[7]=5 |

# 链式前向星及其简单应用

- 找出所有起点为1的边
- 输出e[1]=2

| head[1]=6 | head[2]=2 | head[3]=3 | head[4]=7 | head[5]=0 |
|---|---|---|---|---|

| e[1]=2 | e[1]=2 | e[3]=4 | e[4]=3 | e[5]=1 | e[6]=5 | e[7]=5 |
|---|---|---|---|---|---|---|
| next[1]=0 | next[1]=0 | next[3]=0 | next[4]=1 | next[5]=0 | next[6]=4 | next[7]=5 |

# 链式前向星及其简单应用

- 找出所有起点为1的边
- next[1]=0，输出结束

| head[1]=6 | head[2]=2 | head[3]=3 | head[4]=7 | head[5]=0 |
|---|---|---|---|---|

| e[1]=2 | e[1]=2 | e[3]=4 | e[4]=3 | e[5]=1 | e[6]=5 | e[7]=5 |
|---|---|---|---|---|---|---|
| next[1]=0 | next[1]=0 | next[3]=0 | next[4]=1 | next[5]=0 | next[6]=4 | next[7]=5 |

# 链式前向星及其简单应用

```
procedure printEdge(u:longint);//输出所有以u为起点的边的终点
var
  i:longint;
begin
  i:=head[u];                  //得到第一个地址
  while i<>0 do                //未输出完时i<>0
  begin
    write(e[i],‘ ’);           //输出或用于计算
    i:=next[i];                //取下一个地址
  end;
end;
```

# 链式前向星及其简单应用

- 对于构造，类似于向链表中插入元素
- procedure insertDirectedEdge(i,u,v:longint);
  //插入第i条有向边，起点为u，终点为v
- begin
- e[i]:=v; //记录终点
- next[i]:=head[u]; //下一个指向原首元素
- head[u]:=i; //首指针指向新加的边
- end;

- 链式前向星的构造时间复杂度为O(e)，查询与普通前向星相同。

# 链式前向星及其简单应用

- 通过考察代码，我们可以发现，链式前向星并不改变边表原来的存储顺序，只是建立了元素之间的关系。
- 如果我们将“链式前向星”改成“链表前向星”，可以发现这实际上就是邻接链表。
- 或者我们链式前向星是邻接链表的数组实现形式。
- 但是，为什么有些题我们不用邻接链表而用链式前向星呢？原因基本如下：
- ①指针操作比较复杂，极易写错
- ②指针占用空间较大（4字节，相当于int64）
- ③无向图链式前向星实现简单、经典，存储权值可节省一半空间（相对于邻接链表）

# 链式前向星及其简单应用

- 第③条我需要解释一下
- ```
  procedure insertUndirectedEdge(i,u,v,w:longint);
  ```
- ```
        //插入第i条无向边，两端点为u和v，正反权值为均w
  ```
- ```
        //将一条无向边拆成两条有向边
  ```
- ```
  begin
  ```
- ```
      e[i<<1]:=v;                          //插入u->v
  ```
- ```
      next[i<<1]:=head[u];
  ```
- ```
      head[u]:=i<<1;
  ```
- ```
      e[i<<1+1]:=u;
  ```
- ```
      next[i<<1+1]:=head[v]; //插入v->u
  ```
- ```
      head[v]:=i<<1+1;
  ```
- ```
      weight[i]:=w;              //两条有向边共用一个权值
  ```
- ```
                                 //拆分后第j条边的权值为weight[j>>1]
  ```
- ```
  end;
  ```

- 由于利用了原边集的有序性，使得权值的存储占用更少的空间
  ```

# 链式前向星及其简单应用

- 至此，我们可以发现：链式前向星是虽然用到了模拟指针但易于理解，实现简单、经典，编程复杂度极低，时空复杂度较低，是一个优秀的数据结构。
- 利用链式前向星，我们可以快速高效地解决许多问题。

# 链式前向星及其简单应用

- 应用一：SPFA
- 优化后的时间复杂度为O(ke)，k≈2
- N次SPFA用于稀疏图求任意两点间最短路径也很有用

```
procedure SPFA(s:longint);
//求源点为s的单源最短路，存储至dist[s]数组中
var
  i,r,f:longint;
  q:array[1..maxN*5] of longint;            //取k=5防止最坏情况发生
  v:array[1..maxN] of boolean;
begin
  for i:=1 to n do dist[s,i]:=maxLong;
  fillChar(v,sizeof(v),0);
  dist[s,s]:=0; r:=1; f:=1; q[1]:=s;
  v[s]:=true;                               //各种初始化
```

# 链式前向星及其简单应用

```
repeat
    i:=head[q[r]];  //取首指针
    while i<>0 do begin
      if dist[s,e[i]]>dist[s,q[r]]+a[e[i],q[r]] then begin
        dist[s,e[i]]:=dist[s,q[r]]+a[e[i],q[r]];
        if not v[e[i]] then begin
          inc(f);
          q[f]:=e[i];
        end;
      end;
      i:=next[i];   //取下一条边
    end;
    v[q[r]]:=false;
    inc(r);
  until r>f;
end;
```

# 链式前向星及其简单应用

- 应用二：多叉树转二叉树（用于树形DP）
- 应用三：图的遍历
- 伪代码如下：
- proc work(u);　　//u为起点
- i:=head[u];
- while i<>0 do
- begin
- 　对u->e[i]这条边进行操作或访问结点e[i]
- 　i:=next[i];
- end;