

# C++算法

堆

heap

二叉堆

binary heap

# 二叉堆

binary heap

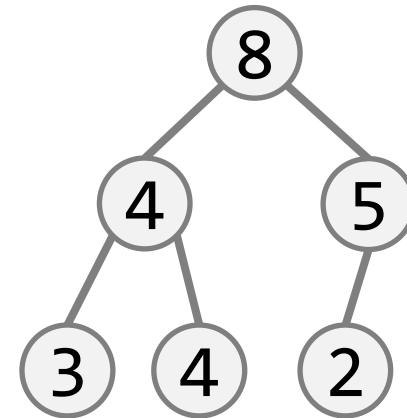
二叉堆是特殊的完全二叉树

每个节点 $u$ 带有点权 $a[u]$

父子间点权有固定大小关系

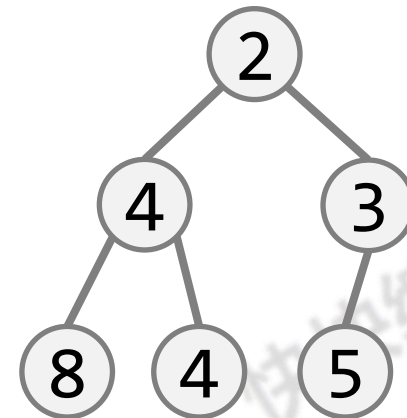
大  
根  
堆

所有父节点的点权  
大于等于  
子节点的点权



小  
根  
堆

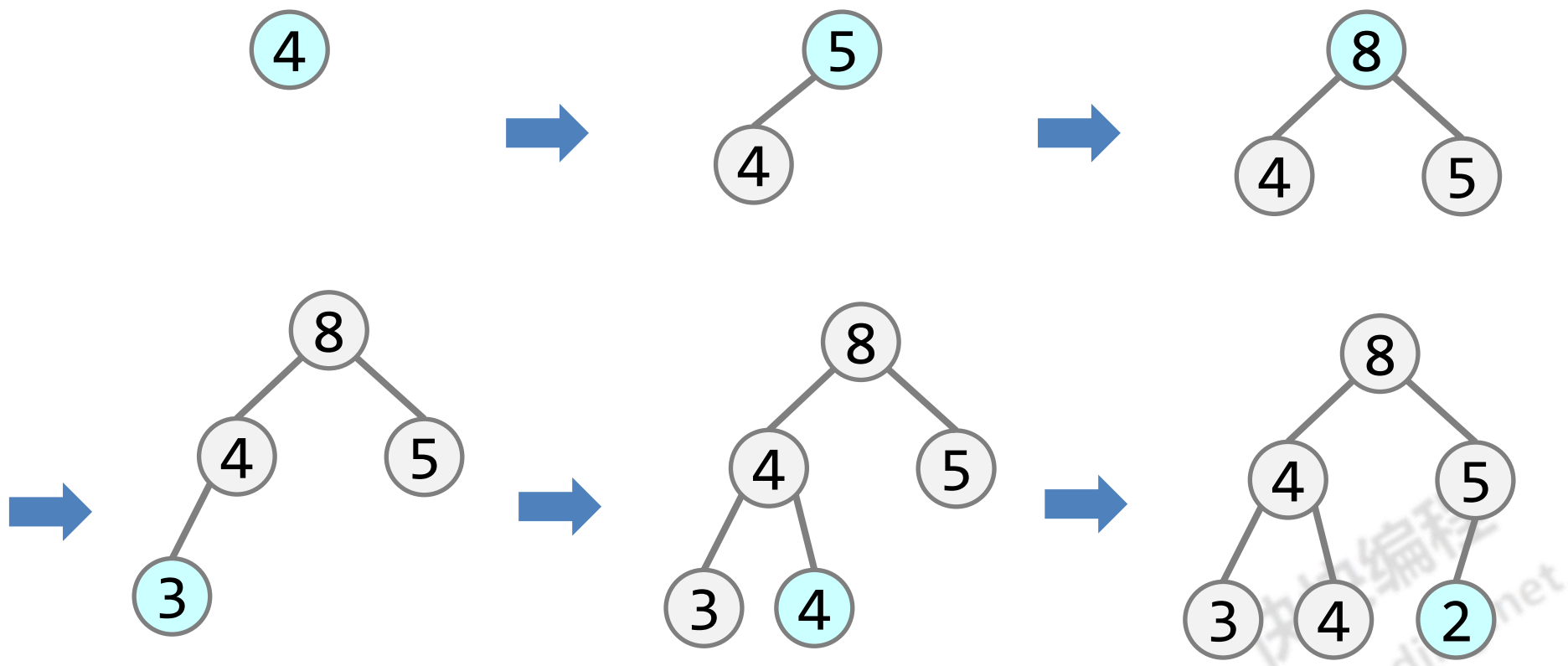
所有父节点的点权  
小于等于  
子节点的点权



大根堆	依次插入新节点
	保持完全二叉树
	单次插入复杂度 $O(\log n)$

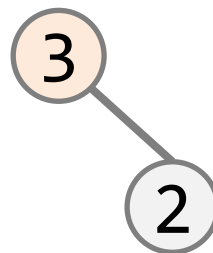
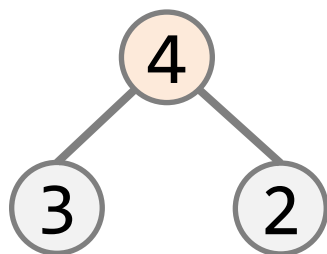
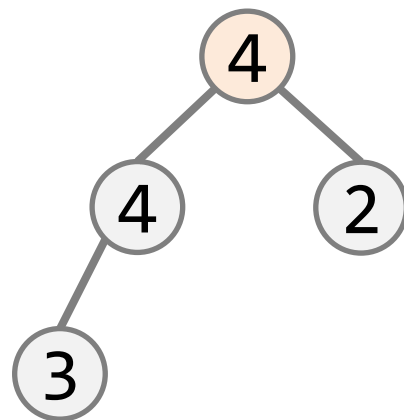
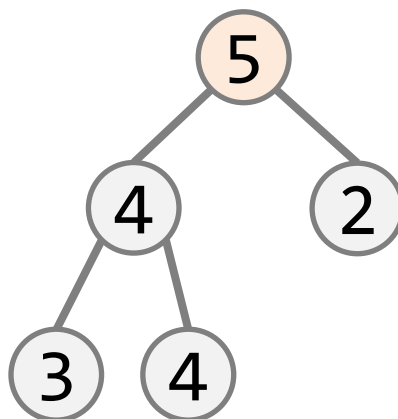
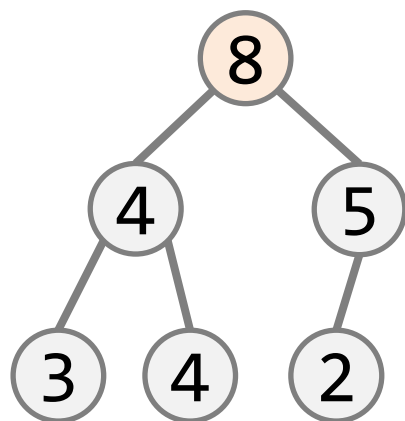
二叉堆
binary heap

4 5 8 3 4 2



大根堆	删除根节点
	保持完全二叉树
	单次删根复杂度 $O(\log n)$

二叉堆
binary heap



2个儿子挑  
1个继承王位

# 优先队列

`priority_queue`

# 优先队列

priority\_queue

用二叉堆实现

默认大根堆

请预测输出结果

$O(1)$

$O(\log n)$

如何改成小根堆

```
1  #include<iostream>
2  #include<queue> ←
3  using namespace std;
4  int main(){
5      priority_queue<int> q;
6      int a[5]={3,1,4,1,5};
7      for(int i=0;i<5;++i)
8          q.push(a[i]);
9      while(!q.empty()){
10         cout<<q.top()<<endl;
11         q.pop();
12     }
13     return 0;
14 }
```

# 优先队列

priority\_queue

用二叉堆实现

小根堆方法1

```
1 #include<iostream>
2 #include<queue>
3 using namespace std;
4 int main(){
5     priority_queue<int> q;
6     int a[5]={3,1,4,1,5};
7     for(int i=0;i<5;++i)
8         q.push(-a[i]);
9     while(!q.empty()){
10         cout<<-q.top()<<endl;
11         q.pop();
12     }
13     return 0;
14 }
```




# 优先队列

priority\_queue

用二叉堆实现

小根堆方法2

```
priority_queue<int, vector<int>, greater<int> > q;  
int a[5]={3,1,4,1,5};  
for(int i=0;i<5;++i)  
    q.push(a[i]);  
while(!q.empty()){  
    cout<<q.top()<<endl;  
    q.pop();  
}
```



priority\_queue

结构体

大根堆

```
4 struct Node{
5     int c,u;
6     bool operator<(const Node&x)const{
7         return c<x.c;
8     }
9 };
10 int main(){
11     priority_queue<Node> q;
12     q.push((Node){30,1});
13     q.push((Node){10,2});
14     q.push((Node){20,3});
15     while(!q.empty()){
16         Node rt=q.top();
17         cout<<rt.c<<" "<<rt.u<<endl;
18         q.pop();
19     }
```

默认大根堆  
调用<判断大小

重载小于号<

必须知道任意  
2个Node的小于规则

如何改成小根堆

```
4 struct Node{
5     int c,u;
6     bool operator<(const Node&x) const{
7         return c>x.c;
8     }
9 };
10 int main(){
11     priority_queue<Node> q;
12     q.push((Node){30,1});
13     q.push((Node){10,2});
14     q.push((Node){20,3});
15     while(!q.empty()){
16         Node rt=q.top();
17         cout<<rt.c<<" "<<rt.u<<endl;
18         q.pop();
19     }
```

# 最短路问题

# 图论要素

# 现场挑战 快快编程943

快快编程  
kkcoding.net

# 图论要素识别

无向图

正权图

多源单汇

单源多汇

可能不连通

$n \leq 20000$ ,  $m \leq 200000$ ,  
稀疏图  $O(E)$  约等于  $O(V)$

算法选择

Dijkstra堆优化

复杂度  $O(E \log E)$

# Dijkstra 朴素版

复杂度  
 $O(V^2)$

```
fill(dst, dst+n+9, INF);
dst[1]=0;
for(int k=1; k<=n-1; k++){
    int u=n+1;
    for(int j=1; j<=n; j++)
        if(!ok[j] && dst[j]<dst[u]) u=j;
    if(dst[u]>=INF) break;
    ok[u]=1;
    for(int i=0; i<to[u].size(); i++)
        dst[to[u][i]]=min(dst[to[u][i]], dst[u]+w[u][i]);
}
```

速度瓶颈在于  
n次打擂台

每次打擂台 $O(n)$



```
6  bool ok[N];
7  ll n,m,dst[N],x[N];
8  vector<ll> to[N],w[N];
9  struct Node{
10     ll c,u;
11     bool operator<(const Node&a) const{
12         return c>a.c;
13     }
14 };

```

# Dijkstra 邻接矩阵

复杂度  
 $O(V^2)$

```
15 void Dijkstra(){
16     fill(dst, dst+n+9, INF);
17     priority_queue<Node> q;
18     dst[1]=0;
19     q.push((Node){0,1});
20     while(!q.empty()){
21         ll u=q.top().u; q.pop();
22         if(ok[u])continue;
23         ok[u]=1;
24         for(ll i=0;i<to[u].size();i++){
25             ll v=to[u][i];
26             if(dst[v]<=dst[u]+w[u][i])continue;
27             dst[v]=dst[u]+w[u][i];
28             q.push((Node){dst[v],v});
29         }
30     }
31 }
```

# 现场挑战

## 快快编程944

快快编程  
kkcoding.net

# 图论要素识别

有向图

负权图

多源单汇

单源多汇

可能不连通

判负权回路

$n \leq 2000$ ,  $m \leq 100000$ , 稀疏图

算法选择

SPFA

改Bellman-Ford

# Bellman-Ford分析+改进

```
10 void BellmanFord(){
11     fill(d,d+n+1,INF);
12     d[1]=0;
13     for(ll k=1;k<=n-1;k++)
14         for(ll i=1;i<=n;i++)
15             for(ll j=0;j<to[i].size();j++)
16                 d[to[i][j]]=min(d[to[i][j]],d[i]+w[i][j]);
17 }
```

共 $n-1$ 轮迭代更新

若某轮迭代所有点都没更新

若某点在某轮迭代没更新

该轮更新过的点入队

# SPFA

in[i]是什么

```
8 void spfa(){
9     fill(dst,dst+n+9,INF);
10    queue<int> q;
11    dst[1]=0; in[1]=1; q.push(1);
12    while(!q.empty()){
13        int v,u=q.front(); q.pop(); in[u]=0;
14        for(int i=0;i<to[u].size();i++)
15            if(dst[v=to[u][i]]>dst[u]+w[u][i]){
16                dst[v]=dst[u]+w[u][i];
17                if(!in[v])q.push(v);
18            }
19    }
20 }
```

in[v]忘了改

# SPFA判负环

cnt[i]是什么

```
9 fill(dst,dst+n+9,INF);
10 queue<int> q;
11 dst[1]=0; in[1]=1; cnt[1]=1; q.push(1);
12 while(!q.empty()){
13     int u=q.front(); q.pop(); in[u]=0;
14     for(int i=0,v;i<to[u].size();i++)
15         if(dst[v=to[u][i]]>dst[u]+w[u][i]){
16             dst[v]=dst[u]+w[u][i];
17             if(!in[v])q.push(v),in[v]=1,cnt[v]++;
18             if(cnt[v]==n)return -1;
19         }
20 }
21 return 0;
22 }
```

# 卡SPFA

请构造一类正权图

放Dijkstra堆优化AC

卡SPFA超时

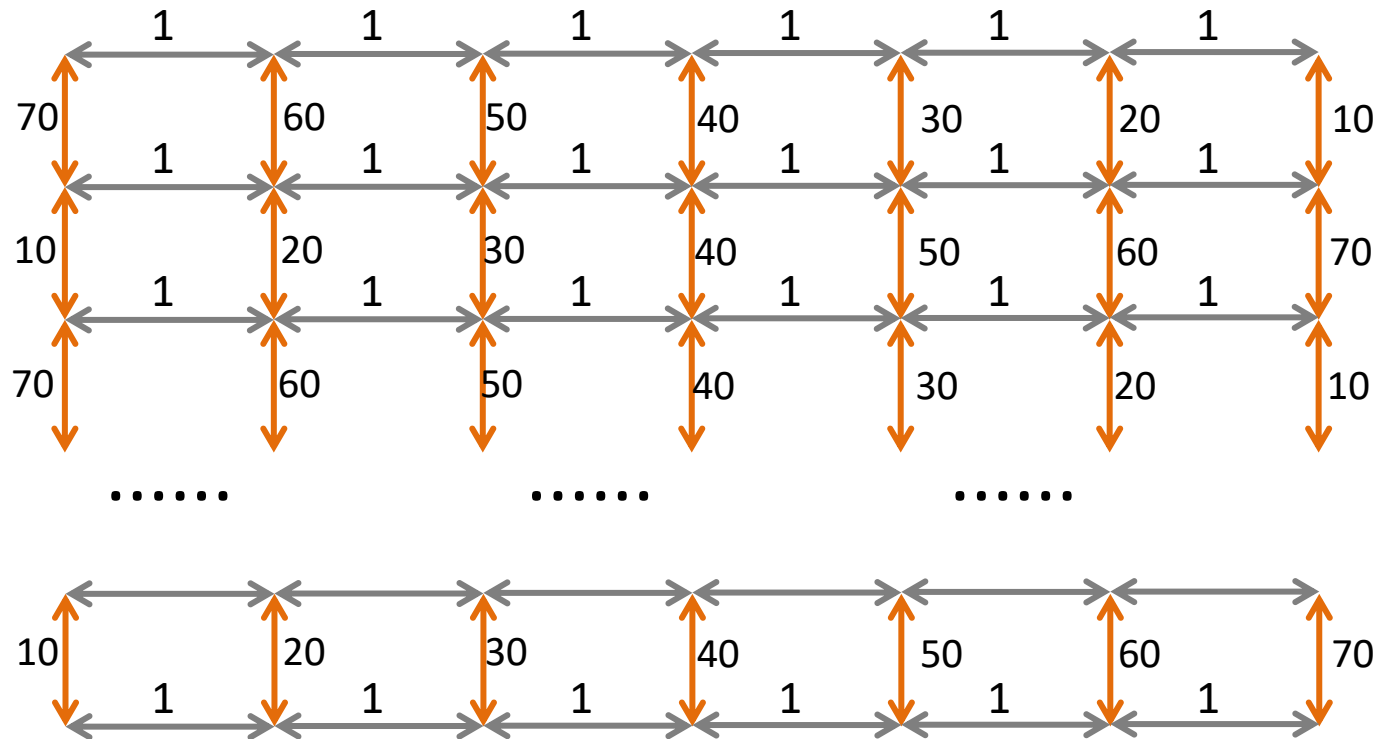
达到最差情况复杂度  
 $O(VE)$

5分钟后请同学分享

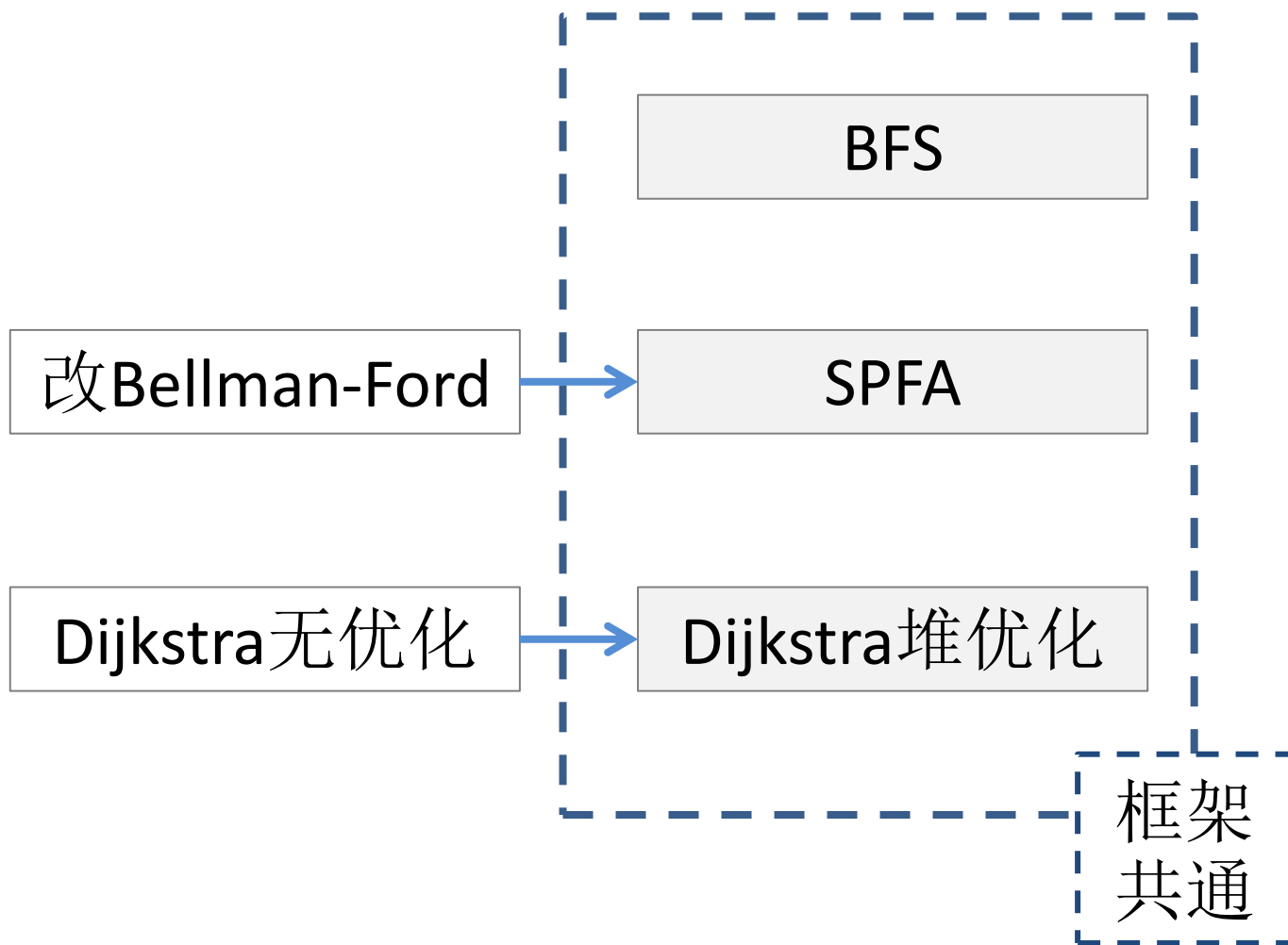


# 卡SPFA 网格图

SPFA退化到 $O(VE)$ ,  
需要每一轮都有 $O(E)$ 边更新  
更新轮次 $O(V)$ 轮



# 算法对比



# 快快编程作业

941

942

943

944