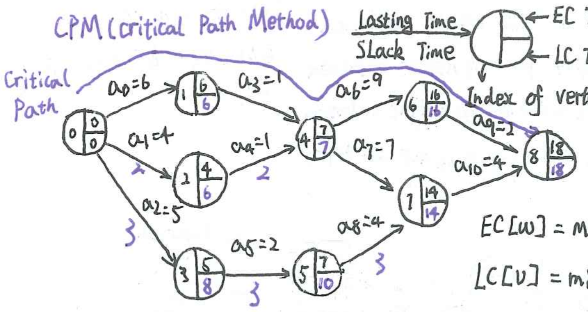
③ Acyclic Graphs

Application: AOE (Activity On Edge) Networks

$EC[j] / LC[j]$ ::= the earliest \ latest completion time for node v_j

CPM (Critical Path Method)



EC 从前向后

LC 从后向前

$EC[w] = \max\{EC[v] + C_{v,w}\}$

$LC[v] = \min\{LC[w] - C_{v,w}\}$

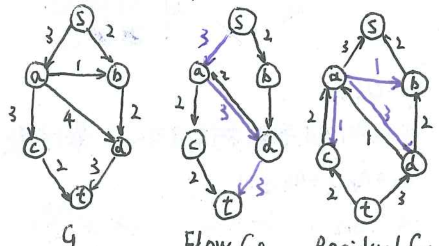
Slack Time of $<v,w> = LC[w] - EC[v] - C_{v,w}$

Critical Path ::= path consisting entirely of zero-slack edges

所有对最短路径问题

方一 用single-source algorithm |V|次 $T = O(|V|^3)$ 稀疏图上效果不错

Network flow problem



G　　Flow Gf　　Residual Gr

Step1: 寻找任一条从s到t的路

Step2: 选择该路最小边作为流量加至Gf

Step3: 更新Gr, 移除0流量边

Step4: If (有从s→t的路)

Goto Step1;

Else

End

每加一个edge(v,w)到Gf, 都要加一edge(w,v)到Gr

始终选流量最大路径 $T = O(|E|^2 \log|V|)$

始终选择最小边 $T = O(|E|^2 |V|)$

Minimum Spanning Tree

边数量为|V|-1, 覆盖每个点, 当且仅当G为connected时存在

加一个非树边到树上就会成环.

① Prim's Algorithm

选取一个点 从为根开始生长.

每环只加一条边, 该边设为edge(u,v)

其中u属于树, v不属于树.

选取所有edge(u,v)中cost最小的.

反复该过程, 直至无点剩余

```
void DFS ( Vertex V ){
    visited[V] = true;
    for (each W adjacent to V)
        if (!visited[W])
            DFS(W);
}
```

② Kruskal's Algorithm

```
void Kruskal (Graph G)
{  T = { };
   while (T contains less than |V|-1 edges
          && E is not empty){
     choose a least cost edge (v,w) from E;
     delete (v,w) from E
     if ((v,w) does not create a cycle in T)
        add (v,w) to T;
     else
        discard (v,w);
   }
   if (T contains fewer than |V|-1 edges)
      Error("No spaining tree");
}
```

2. Biconnectivity

articulation point

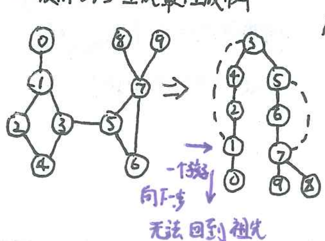
如果从G中删除v, 使得新图G'拥有至少2个子连通分量, 那么v就是articulation point (咬合点)

G没有咬合点则G为Biconnectivity.

一条边不可能由两个biconnected子图共享

寻找biconnected components的算法

① 使用DFS生成最小生成树



是否为咬合点判定

① root: 有两个及以上的child就是

② 非root: (1) 至少1个子结

(2) 向下移动一步后, 没有办法回到祖先

3. Euler Circuits

Euler tour: 一笔画全部边且不重复 ← ① 必须恰好有两个点奇数degree ② 起始点为①的两点之一

Euler circuits: 一笔画全部边且不重复. 此外结束于起始点. ① 图为联通图 ② 所有点都为偶数度

Hamilton cycle: 一个访问所有点的环

Chap 6 sorting

1. 逆序

$i<j$ 但 $A[i]>A[j]$ 称为一对逆序. 34, 8, 64, 51, 32, 21 有9个逆序.

$T(N, I) = O(I+N)$ I为逆序数

一个数列中有逆序数为 $N(N-1)/4$

2. Shell sort

5 sort: 第1、6、11…个进行排序, 第2、7、12…个进行排序, 第3、8、13…个进行排序…

3 sort: (上述结果) 第1、4、7…个进行排序, 第2、5、8…个进行排序, 第3、6、9…个进行排序

1 sort: (上述结果) 第1、2、3…进行排序

需指定一个上升列 (例中 $H_1=1, H_2=3, H_3=5$)

$H_{起始} = \lfloor N/2 \rfloor$, $h_k = \lfloor h_{k+1}/2 \rfloor$　　Worst-case: $\Theta(N^2)$

Hibbard 递增列: $h_k = 2^k - 1$　　worst-case: $\Theta(N^{3/2})$

$T_{avg-Hibbard}(N) = O(N^{5/4})$　　大规模数据用shell sort效果佳

3. Heapsort

```
void Heapsort (ElementType A[], int N){
  int i;
  for (i = N/2; i>=0; i--) /* BuildHeap*/
     PercDown(A, i, N);
  for(i = N-1; i>0; i--){
     swap (&A[0], &A[i]); /*DeleteMax*/
     PercDown(A, 0, i);
  }
}
```

$T = 2N\log N - O(N\log\log N)$

PercDown (A, i, N)

对前N位数列, 将第i位元素与其最大的子结点比较, 若孩子大于该元素, 则对调, 且继续与子结点比较 否则停下来

Naive版: 建个priority heap, 然后逐个DeleteMin

4. Mergesort

分成n小块, 各小块内部排列好. 然后n/2小块…

$T(1) = 1$

$T(N) = 2T(N/2) + O(N) = 2^k T(N/2^k) + k\cdot O(N) = N * T(1) + \log N * O(N)$

$= O(N + N\log N)$

5. Quick Sort

pivot = A[]中某元素　$A_1 = \{a \in S \mid a \le pivot\}$ and $A_2 = \{a \in S \mid a \ge pivot\}$

A = QuickSort $(A_1, N_1) \cup \{pivot\} \cup$ QuickSort (A_2, N_2)　最佳 $T(N) = O(N\log N)$

对pivot的选取: Pivot = median (left, center, right)

QuickSort 在 $N \le 20$ 的情况下慢于insertion sort

解决方案: 小于20时转用其他排序

```
void Qsort ( int A[], int Left, int Right){
  int i, j, Pivot;
  if (Left + cutoff <= right){  // 防止区间过小
    Pivot = Median3 (A, Left, Right)
    // 将最左、正中、最右排序后, 将正中与最右前一位
    // 交换, 返回当前状态的最右前一位
    i = left;  j = Right - 1;
    for(;;){
      while (A[++i] < Pivot){}  // 从左扫描
      while (A[--j] > Pivot){}  // 从右扫描
      if (i<j)
        swap(&A[i], &A[j]);
      else  break;
    }
    Swap (&A[i], &A[Right-1]);
    Qsort (A, Left, i-1);
    Qsort (A, i+1, Right);
  }
  else
    InsertionSort (A+Left, Right - Left+1);
}
```

$T(N) = T(i) + T(N-i-1) + cN$

Worst Case:

$T(N) = T(N-1) + cN \Rightarrow T(N) = O(N^2)$

Best case:

$T(N) = 2T(N/2) + cN \Rightarrow T(N) = O(N\log N)$

Average case:

$T(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} T(j)\right] + cN \Rightarrow T(N) = O(N\log N)$