§3 The Search Tree ADT - Binary Search Trees

(1) 每个结点都有一个独一无二int类型的key (3) 左右子树皆为二叉搜索树

(2) left child < root < right child

Operation: MakeEmpty() Insert()
Find(X) Delete()
leftmost ← FindMIN() Retrieve()
rightmost ← FindMAX()

```
Find()
while (T) {
  if (X == T->Element)
    return T;
  if (X < T->Element)
    T=T->left;
  else
    T=T->right;
}
return NULL;
```

T(N)=O(d) d为X的深度

Delete()
① 删除叶子结点：将其父结点设为NULL
② 删除degree1的点：用子结点代替它
③ 删除degree2的点：(1) 用左子树中最大或右子树最小的点代替
(2) Delete(替代点)

删除点时可仅标记"delete"
当重新加node时常无需重新分配空间

Chap 5 Priority Queues (Heaps)

对象：一个有着0个或多个元素的有限有序列表
操作：Initialize() Insert() DeleteMin() FindMin()

四大实现方式：
① Array 因为插入O(1) 对于删除O(n) 所以最优.
② Linked List
③ Ordered Array
④ Ordered Linked List

§3 Binary Heap

A complete binary tree of height h has between 2^h and $2^{h+1}-1$ nodes

$h = \lfloor \log N \rfloor$

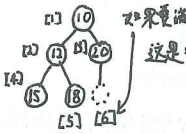
Lemma：如果完全二叉树(有n个结点)用序列化结构(即数组)表示，则对于任何index为i的点，$1 \le i \le n$，我们有：(index从1开始用)

(1) 父结点index $parent(i) = \begin{cases} \lfloor i/2 \rfloor & \text{if } i \neq 1 \\ \text{None} & \text{if } i = 1 \end{cases}$

(2) 左子结点index $left\_child(i) = \begin{cases} 2i & \text{if } 2i < n \\ \text{None} & \text{if } 2i > n \end{cases}$

(3) 右子结点index $right\_child(i) = \begin{cases} 2i+1 & \text{if } 2i+1 \le n \\ \text{None} & \text{if } 2i+1 > n \end{cases}$

max heap：任何父结点均比子结点大
min heap：任何子结点均比父结点大



如果满足完全二叉树，这是唯一一个可插入的点.

Basic Heap Operation

(1) insertion：如果唯一可插入点的父结点 < 新点，直接插入
如果反之，则将父结点与新点交换，然后再比较新点现在的父结点大小比，直至符合要求

```
for (i = ++H->size; H->Elements[i/2] > X; i/=2)  <= Percolate(过滤)up
  H->Elements[i] = H->Elements[i/2];             <= 比swap快
H->Elements[i] = X
```

T=O(logN)

(2) deleteMin：删除根然后重组完全二叉树即可 T=O(logN)

(3) DecreaseKey(P,Δ,H) 将位置P的结点的键值降低/减一正值Δ

(4) IncreaseKey(P,Δ,H)

(5) Delete(P,H)：等同于 DecreaseKey(P,∞,H); DeleteMin(H);

(6) BuildHeap(H) 给定一串数，建立Heap

d-Heaps — All nodes have d children

① DeleteMin：$O(d \log_d N)$ ② /2或*2为bit位移，但/d或*d不是.

Chap 8 The disjoint set

例：给定集合{1,2,3,4,5,6,7,8,9,10,11,12}与9种关系 1≡4, 3≡1, 6≡10, 8≡9, 7≡4, 6≡8, 3≡5, 2≡11, 11≡12.

equivalence class：{2,4,7,11,12} {1,3,5} {6,8,9,10}

Op: Union(i,j) $S = S_i \cup S_j$
Find(i) 找到含有元素i的set S_j

实现：S[element]为element的父结点

S[root]=0

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
S	5	0	5	2	0	10	2	10	10	0	2	2

Union操作只要把S[root]改为对应值即可. Find操作：

```
for(; S[X]>0; X=S[X]);
return X;
```

改进：S[root]=-size 时间复杂度：$O(N+M\log_2 N)$

height(Tree) $\le \lfloor \log_2 N \rfloor + 1$ union-by-size — Always change the smaller tree
union-by-height — Always change the shallow tree

Ackermann's Function

$$A(i,j) = \begin{cases} 2^j & i=1 \text{ and } j \ge 1 \\ A(i-1,2) & i \ge 2 \text{ and } j = 1 \\ A(i-1, A(i,j-1)) & i \ge 2 \text{ and } j \ge 2 \end{cases}$$

Chap 9 Graph Algorithms

G(V,E) undirected graph $(V_i,V_j) = (V_j,V_i)$

Directed graph (digraph)：$\langle V_i,V_j \rangle = V_i \to V_j \neq \langle V_j,V_i \rangle$ (tail → head)

tree：graph that is connected and acyclic

DAG：a directed acyclic graph

强连通图：图内任两点对都有path由A至B以及由B至A (Strongly connected directed graph G)

弱连通为无向图中任两点均有路

Strongly connected component：最大为强连通图的子图

完全图：有最大数量的边

Length of a path = number of edge on the path

Degree：

indegree(V)=3
out-degree(V)=1
degree(V)=4

Graph G with n vertices and e edges

$e = \left(\sum_{i=0}^{n-1} d_i\right)/2$ where $d_i = degree(V_i)$

图的表示法

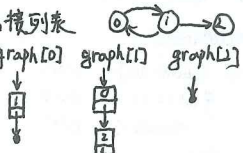
① 临接矩阵 T and S both $O(n^2)$

$$adj\_mat[i][j] = \begin{cases} 1 & \text{if } (V_i,V_j) \text{ or } \langle V_i,V_j \rangle \in E(G) \\ 0 & \text{otherwise} \end{cases}$$

$degree(i) = \sum_{j=0}^{n-1} adj\_mat[i][j]$ (undirected)

$+ \sum_{j=0}^{n-1} adj\_mat[j][i]$ (directed)

② 临接列表

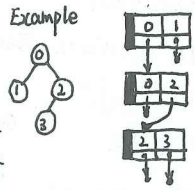


S=(n+2e) ptrs + 2e ints

③ 临接multilists

④ 权重边 adj_mat[i][j]=weight

Example

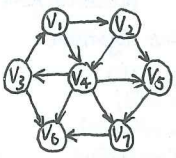


拓扑排序：对DAG进行排序，将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若边(u,v)∈E(G)则u在线性序列中出现在v之前

拓扑排序不一定唯一

算法：以indegree=0的点为起点，摘掉该点，更新所有其他点的indegree.
当其他某一点为0时，继续摘掉/更新，直至最后点.
摘除顺序即为拓扑排序.

最短路径算法



	Dist	Path
V_1	1	V_3
V_2	2	V_1
V_3	0	0
V_4	2	V_1
V_5	3	V_2
V_6	1	V_3
V_7	3	V_4

1. Single-Source Shortest-Path Problem

无权重最短路径

① 宽度优先(Breadth-first search)

Table[i].dist = distance from S to V_i
Table[i].Known = 1 if V_i is checked or 0 if not
Table[i].path = last point

② Dijkstra's algorithm

```
void Dijkstra(Table T){
  Vertex V,W;
  for(;;){
    V = smallest unknown distance vertex;
    if(V == NotAVertex)
      break;
    T[V].Known = true;
    for(each W adjacent to V)
      if(!T[W].Known)
        if(T[V].Dist + Cvw < T[W].Dist){
          Decrease(T[W].Dist to T[V].Dist);
          T[W].Path = V;
        }
  }
}
```

实现1
仅简单扫描table
$T = O(|V|^2 + |E|)$
图密时有利

实现2
放在priority queue中，调用DeleteMin找最小. $O(\log |V|)$
$T = O(|E| \log |V|)$
要求|E| DeleteMin
|E| space