ECE20010 Data Structures

Chapter 5

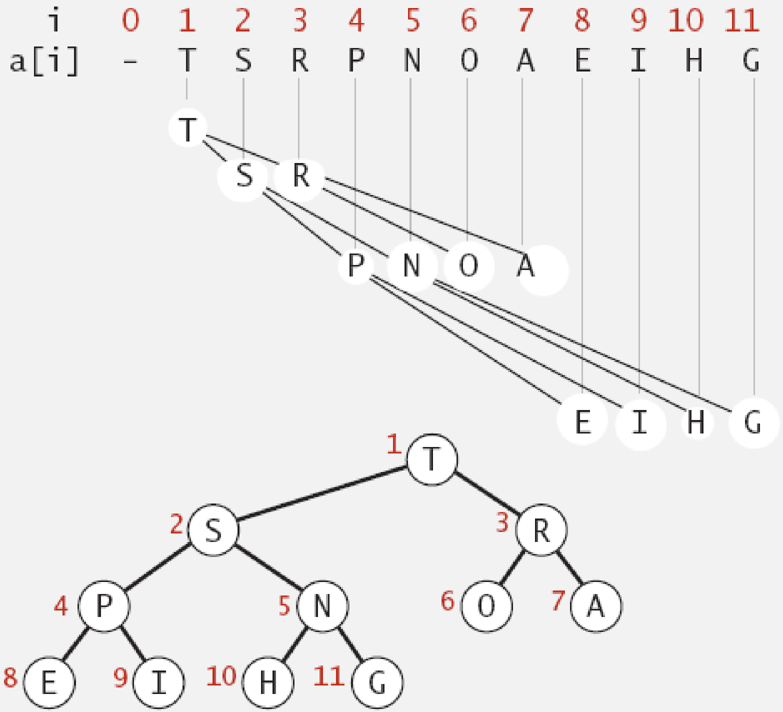
개인별 숙제
입니다.

- *binary search tree*
  - *Implementation*
- *Homework08 (4 points)*
  - *implement Heap or Priority Queue.*
  - *submit it in dropbox after clean it up (-1.0 point)*
  - *by Tuesday May 13, 11:55 PM*

Chapter 5.6 Heaps & Priority Queues

Binary heap: array representation of a heap-ordered complete binary tree

- Heap-ordered binary tree
  - Keys in nodes
  - Parent's key no smaller than children's keys.
- Array representation
  - Indices start at 1.
  - Take nodes in level order.
  - No explicit links needed!



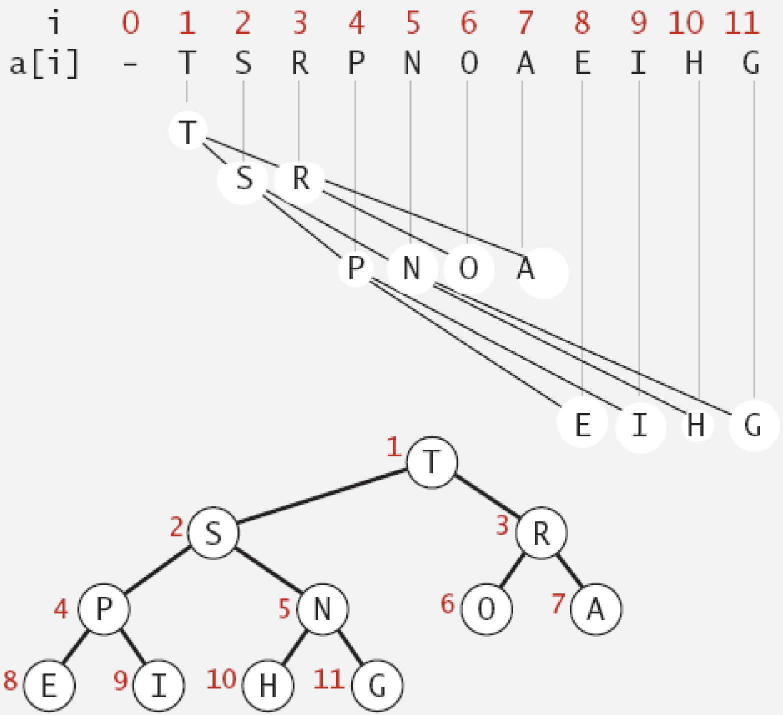
i	0	1	2	3	4	5	6	7	8	9	10	11
a[i]	-	T	S	R	P	N	O	A	E	I	H	G

Heap representations

Chapter 5.6 Heaps & Priority Queues

Binary heap properties:

- Largest key is a[1], which is root of binary tree.
- Use array indices to move through tree.
  - Parent at k is at k/2.
  - Children at k are at 2k and 2k+1.



Heap representations

Heap ADT: heap (or priority queue) structure:

PQ.h

```
typedef struct PQ *pq;
typedef struct PQ {          // heap or min/max priority queue
   Key    *node;             // an array of nodes (key only for simplicity)
   int    capacity;          // array size of node or key, item
   int    N;                 // the number of nodes in the heap or PQ
} PQ;
```

PQc.h

```
#ifndef PQc_h
#define PQc_h

#define PQKeyTypeDefined
typedef   char   Key;              // added for flexibility

#endif
```

Heap ADT: heap (or priority queue) structure:

```
pq newPQ(int capacity);                  // PQ is created with capacity(or array size)
void freePQ(pq p);                       // deallocate PQ
int size(pq p);                          // return nItems in PQ currently
int level(int n);                        // return level based on num of nodes
int capacity(pq p);                      // return its capacity (array size)
int resize(pq p, int size);              // resize the array size (= capacity)
int isFull(pq p);                        // return true/false
int isEmpty(pq p);                       // return true/false

int insertMax(pq p, Key key);            // insert in max queue
int deleteMax(pq p);                     // delete in max queue

// helper functions to support insert/delete functions
int   less(pq p, int i, int j);          // used in MaxPQ
void swap(pq p, int i, int j);           // exchange two node
void swim(pq p, int k);                  // bubble up
void sink(pq p, int k);                  // tickle down

// helper functions to check PQ ADT
int   isMaxHeap(pq p);         // is PQ[1..N] a max Heap?
void showHeap(pq p);           //// Key dependent //// prints details of Heap status
char *toString(pq p);          //// Key dependent //// return a string that has all keys
```

Heap ADT: heap (or priority queue) implementation:

PQ.h

```
typedef struct PQ *pq;
typedef struct PQ {
   Key    *node;
   int    capacity;
   int    N;
} PQ;
```

```
/** instantiate a new pq and return the new pq pointer. */
pq newPQ(int capacity) {
        pq p = (pq)malloc(sizeof(PQ));
        verify(p != NULL, "PQ: cannot allocate memory.");

        p->N = 0;
        p->capacity = capacity < 2 ? 2 : capacity;
        p->node = (Key *)malloc(sizeof(Key)* p->capacity);
        verify(p->node != NULL, "PQ: cannot allocate memory.");
        return p;
}
```

Heap ADT: heap (or priority queue) implementation:

PQ.h

```
typedef struct PQ *pq;
typedef struct PQ {
   Key     *node;
   int      capacity;
   int      N;
} PQ;
```

```
// deallocate a PQ.
void freePQ(pq p) {
   free(p->node);
   free(p);
}
```

```
// Is this pq empty?
int isEmpty(pq p) {
   return (p->N == 0) ? true : false;
}
```

```
// return the number of items in PQ
int size(pq p) {
   return p->N;
}
```

```
// Is this pq full?
int isFull(pq p) {
   return (p->N == p->capacity - 1) ? true : false;
}
```

Heap ADT: heap (or priority queue) implementation:

```
int deleteMax(pq p) {
   if (isEmpty(p)) return INT_MIN;

   int maxKey = p->node[1];
   swap(p, 1, p->N--);
   sink(p, 1);

   if ((p->N > 0) && (p->N == (p->capacity - 1) / 4))
      printf("deleteMAX: PLACE YOUR CODE HERE\n");
   return maxKey;
}
```

```
int insertMax(pq p, Key key) {
   if (isFull(p)) printf("insertMAX: PLACE YOUR CODE HERE\n");

   p->node[++p->N] = key;          // add key and swim up to maintain PQ invariant
   swim(p, p->N);
   return key;
}
```

Heap ADT: heap (or priority queue) implementation:

```
int less(pq p, int i, int j) {
    return p->node[i] < p->node[j];
}
```

```
void swap(pq p, int i, int j) {
    Key t = p->node[i];
    p->node[i] = p->node[j];
    p->node[j] = t;
}
```

```
void swim(pq p, int k) {
    while (k > 1 && less(p, k / 2, k)) {
        swap(p, k / 2, k);
       k = k / 2;
   }
}
```

```
void sink(pq p, int k) {
    int N = p->N;
    while (2 * k <= N) {
        int j = 2 * k;
        if (j <N && less(p, j, j + 1)) j++;
        if (!less(p, k, j)) break;
        swap(p, k, j);
        k = j;
    }
}
```

Chapter 5.6 Heaps & Priority Queues

Checklist:

- resize() dynamically increase or decrease the array node
- insertMax() when it gets full, invoke resize()
- deleteMax() when it gets one quarter full, invoke resize()
- isMaxHeap() check whether or not a given PQ is heap-ordered
- newCBT() with a given array, instantiate a new complete binary tree
  heapify() make a complete binary tree into a (max) heap
- showHeap() instead of one line, print keys by level per line
- **Bug report** list bugs and document them properly (or show input & output)
  If **unreported bug** is found , a penalty will be applied