

# CSE 5311 Lab Assignment 1

Due July 9, 2013

## Goals:

1. Review of binary search trees.
2. Understanding of randomized search trees (treaps).

## Requirements:

1. Write (and test) a C/C++ program that uses a treap to implement the dynamic programming algorithm for finding a *heaviest strictly increasing subsequence* in expected  $O(n \log n)$  time. Your program must compile and execute on at least one of OMEGA or Visual Studio.

The input to your program will be the same as for the provided program `HISstrict2.c` (which runs in  $O(n^2)$  time due to its limited data structures). The first line of the input gives `n`, the number of `(value, weight)` pairs appearing on the next `n` lines. Each `value` will be non-negative and each `weight` will be positive.

2. Email your code (as attachments) to `derekwwhite@mavs.uta.edu` before 5:45 pm on July 9. The subject should include your name as recorded by the University.

## Getting Started:

1. You may borrow from the code at `http://www.cs.fiu.edu/~weiss` (or other places - besides each other), but be sure to give appropriate credit in your comments.
2. `n` will not exceed 2,000,000.
3. Be careful with the randomly generated priorities.
4. After processing each pair, all `values` in the treap should be unique.

For each `value`, the treap will store the total weight (`tWeight`) for the heaviest strictly increasing sequence ending with this `value`, along with the index of the value in the input sequence. If `valuei < valuej` and `tWeighti ≥ tWeightj`, then pair `j` should be deleted from the treap. (In other words, an inorder traversal on the treap will be in ascending order by both `value` and `tWeight`.)

Bottom line: the inefficient “m” tables in `HISstrict2.c` are being replaced with an efficient treap.

5. Your output will be similar to `HISstrict2.c`, including a trace of the number of treap elements.

```
// Heaviest strictly increasing subsequence, 06/08 BPW
// Based on G. Jacobsen and K-P Vo, Heaviest Increasing/Common Subsequence
// Problems and
// D. Gries, The Science of Programming, p. 262
// which is based on M. Fredman, Discrete Mathematics 11 (1975),
// 29-35

// This version uses low- and high-valued sentinels to simplify.

#include <stdio.h>
#include <stdlib.h>

#define MAXN (1000)
```

```

int b[MAXN];          // Input sequence values
int bWt[MAXN];       // bWt[i] is weight for b[i]
int bPred[MAXN+1];   // Predecessor to b[i] in some IS
int bLength;         // Number of entries for b, bWt, and bPred

int m[MAXN+1];       // m[i] is the smallest value for an IS with
int mWt[MAXN+1];     // total weight mWt[i]
int mLink[MAXN+1];   // The value j for the b[j] last used to set m[i] & mWt[i]
int mLength;         // Number of entries in use for m, mWt, mLink

int seq[MAXN+1];     // Result sequence
int seqWt[MAXN+1];   // seqWt[i] is weight for seq[i]
int seqLength;       // Number of entries for seq, seqWt, and seqLength

int findUnneeded(int start,int weight)
{ // Binary search for last mWt element <= weight
int low=start,      // Returned subscript will be no smaller than start-1
  high=mLength-1,
  mid;

while (low<=high)
{
  mid=(low+high)/2;
  if (mWt[mid]<=weight)
    low=mid+1;
  else
    high=mid-1;
}
return high;
}

void mShift(int start,int weight)
// Shifts elements, from start thru mLength-1, of m, mWt, and mLink
// whose mWt is <= weight. This allows an insert at slot start.
{
int j,k;
// Find unneeded entries
j=findUnneeded(start,weight)+1;
if (j==start)
{ // All entries are needed, so shift right one slot to make room
  for (j=mLength-1;j>=start;j--)
  {
    m[j+1]=m[j];
    mWt[j+1]=mWt[j];
    mLink[j+1]=mLink[j];
  }
  mLength++;
}
else
{ // Shift left over unneeded entries
  for (k=start+1;j<mLength;j++,k++)
  {
    m[k]=m[j];
    mWt[k]=mWt[j];
    mLink[k]=mLink[j];
  }
  mLength=k;
}
}
}

```

```

int findSupported(int x)
{
// Binary search to find supported element for seq value x.
// Returned subscript is for slot with value >= x and the
// previous slot is < x.
int mid,low,high;

low=0;
high=mLength-1;
//printf("start search\n");
while (low<=high)
{
//printf("low %d high %d\n",low,high);
mid=(low+high)/2;
if (m[mid]<x)
low=mid+1;
else
high=mid-1;
}
return low;
}

main()
{

int i,j,k,pos;

scanf("%d",&bLength);
for (i=0;i<bLength;i++)
scanf("%d %d",&b[i],&bWt[i]);

// Initialize low- and high-valued sentinels
mLength=2;
m[0]=(-999999999);
mWt[0]=0;
mLink[0]=(-1);
m[1]=999999999;
mWt[1]=999999999;
mLink[1]=(-1);
for (i=0;i<bLength;i++)
{ // Binary search to find supported element
pos=findSupported(b[i]);
// m[pos-1] is supporting element
if (m[pos]>b[i])
{ // Shift items, then use entry pos
mShift(pos,mWt[pos-1]+bWt[i]);
m[pos]=b[i];
mWt[pos]=mWt[pos-1]+bWt[i];
bPred[i]=mLink[pos-1];
mLink[pos]=i;
printf("1: b[%d]=%d has been inserted in m, mLength %d\n",
i,b[i],mLength);
}
}
}

```

```

else
{ // m[pos]==b[i]
  if (mWt[pos-1]+bWt[i]>mWt[pos])
  { // Can improve entry pos
    mWt[pos]=mWt[pos-1]+bWt[i];
    bPred[i]=mLink[pos-1];
    mLink[pos]=i;
    // Find unneeded entries
    j=findUnneeded(pos+1,mWt[pos])+1;
    if (j>pos+1)
    { // Shift left over unneeded entries
      for (k=pos+1;j<mLength;j++,k++)
      {
        m[k]=m[j];
        mWt[k]=mWt[j];
        mLink[k]=mLink[j];
      }
      mLength=k;
    }
    printf("2: b[%d]=%d has improved an entry of m, mLength %d\n",
      i,b[i],mLength);
  }
  else
  {
    bPred[i]=(-2);
    printf("3: b[%d]=%d has been ignored due to strictness\n",i,b[i]);
  }
}
}
printf(" i      b      bWt  bPred\n");
for (i=0;i<bLength;i++)
  printf("%5d %5d %5d %5d\n",i,b[i],bWt[i],bPred[i]);
printf(" i      m      mWt  mLink\n");
for (i=0;i<mLength;i++)
  printf("%5d %5d %5d %5d\n",i,m[i],mWt[i],mLink[i]);
// Get result sequence
i=mLink[mLength-2]; // Ignore high-valued sentinel
seqLength=0;
while (i!=(-1))
{
  seqLength++;
  i=bPred[i];
}
i=mLink[mLength-2]; // Ignore high-valued sentinel
for (j=seqLength-1;j>=0;j--)
{
  seq[j]=b[i];
  seqWt[j]=bWt[i];
  i=bPred[i];
}
printf("HSIS (total weight %d):\n",mWt[mLength-2]);
for (i=0;i<seqLength;i++)
  printf("%4d %4d %4d\n",i,seq[i],seqWt[i]);
}

```