Homework 5
Fundamental Algorithms, Fall 2002, Professor Yap

Due: Thursday Nov 28, in class

1. **Linear Bin Packing Problem (5+15 Points)**
In the Linear Bin Packing Problem discussed in class, you are given

$$M, w = (w_1, w_2, \ldots, w_n)$$

such that $0 \leq w_i \leq M$, and the goal is to find a **solution** $[n_1, n_2, \ldots, n_k]$ that minimizes k. This is a solution if $1 \leq n_1 < n_2 < \cdots < n_k = n$ and $\sum_{j=n_{i-1}+1}^{n_i} w_j \leq M$ holds for all $i = 1, \ldots, k$. Assume $n_0 = 0$.
(a) Suppose we remove the constraint that w_i be non-negative. Show the greedy method fails. NOTE: We had discussed this in class. Here we want you to give a counter example where n is as small as possible. Partial credits when n is larger than necessary.
(b) Give a dynamic programming solution in case w_i can be negative.

SOLUTION:

(a) Let $M = 1, w = (1, 1, -1)$. Greedy method gives the solution $[1, 3]$, but the optimal solution is $[3]$. Thus, $n = 3$ is sufficient.

REMARK: note that this example satisfies $w_i \leq M$ for all i. If you remove this constraint as well, then we can have a $n = 2$ example, namely $M = 1$, $w = (2, -1)$. The greedy algorithm would fail rightaway, but the correct answer is $[2]$.

(b) Consider the subproblems $P_i = (M, (w_i, w_{i+1}, \cdots, w_n))$ and let k_i be the minimum number of bins required for P_i. Suppose inductively that $k_2, k_3, \cdots, k_n$ are known. Then, we can compute k_1 in $O(n)$ time as follows: Let $tillN[1..n]$ be a global array where $tillN[i]$ idicates that in problem P_i, the first bin holds the elements $w_i, w_{i+1}, \cdots, w_{tillN[i]}$. Also, let $b[1..n+1]$ be a global array where $b[i]$ is the number of bins in an optimal solution for P_i. Note that $b[n+1] = 0$ is used as sentry. The following procedure Bins(i) solves the problem P_i (assuming that $M, (w_1, \ldots, w_n)$ is globally known).

```
Procedure Bins(i)
    if i = n then
        b[n+1] = 0, b[n] = 1, tillN[n] = n. Return.
    Bins(i+1) // recursive call
    b[i] ← ∞
    W ← 0
    for j ← i to n do
        W ← W + w_j
        if (W ≤ M) and b[i] > 1 + b[j+1]
            b[i] ← 1 + b[j+1]
            tillN[i] ← j
```

Thus to solve problem $P = P_1$, we simply call $Bins(1)$. The number of bins in the optimal solution can be found in $b[1]$. We can also reconstruct the optimal groupings in the bins by looking at the array $tillN$.

2. **Shift Key in Huffman Code (15 Points)**
We want to encode small as well as capital letters in our alphabet. Thus 'a' and 'A' are to be distinguished. There are two ways to achieve this: (I) View the small and capital letters as distinct symbols. (II) Introduce a special "shift" symbol, and each letter is assumed to be small unless it is preceded by a shift symbol, in which case it is considered a capital. Use the text of this question as your input string. Punction marks and spaces are part of this string. But new lines (CRLF) do not contribute any symbols to the string.
(a) Compute the Huffman code tree for coding the above string using method (I). Note that the string begins with the words "We want to en..." and ends with "...bute any symbols to the string." Be sure to compute the number of bits in the Huffman code for this string.
(b) Same as part (a) but using method (II).
(c) Discuss the pros and cons.

SOLUTION:

(a)

alphabet	count	alphabet	count	alphabet	count	alphabet	count
a	38	b	9	c	16	d	15
e	43	f	4	g	4	h	16
i	33	k	1	l	22	m	9
n	24	o	22	p	9	q	1
r	18	s	44	t	45	u	13
v	1	w	7	x	1	y	8
“	1	”	1	‘	2	’	2
(	3	)	3	,	2	.	7
:	1	A	1	B	1	C	1
F	1	I	4	L	1	P	1
R	1	T	2	U	1	V	1
W	1					space	98

Please refer to figure 1 for the corresponding Huffman Code Tree.

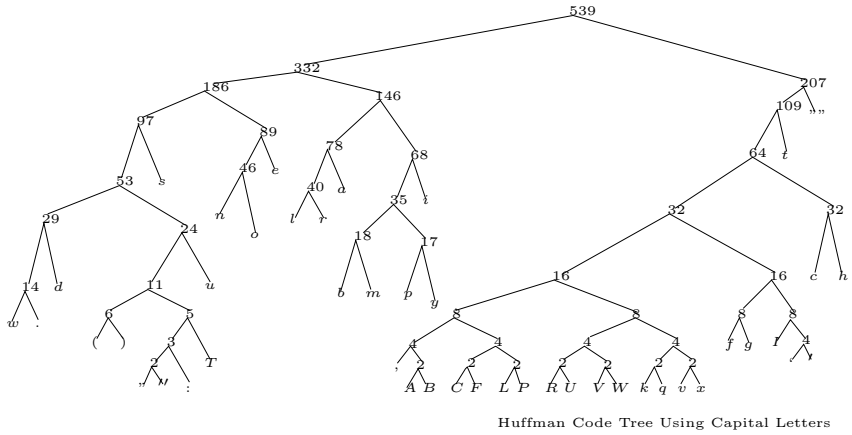


Figure 1: Huffman Code Tree (a)

When we use this Huffman encoding, the length of the string is 2385.

(b)

alphabet	count	alphabet	count	alphabet	count	alphabet	count
a	39	b	10	c	17	d	15
e	43	f	5	g	4	h	16
i	37	k	1	l	23	m	9
n	24	o	22	p	10	q	1
r	19	s	44	t	47	u	14
v	2	w	8	x	1	y	8
“	1	”	1	‘	2	’	2
(	3	)	3	,	2	.	7
:	1	shift	12	space	98		

Please refer to figure 2 for the corresponding Huffman Code Tree.

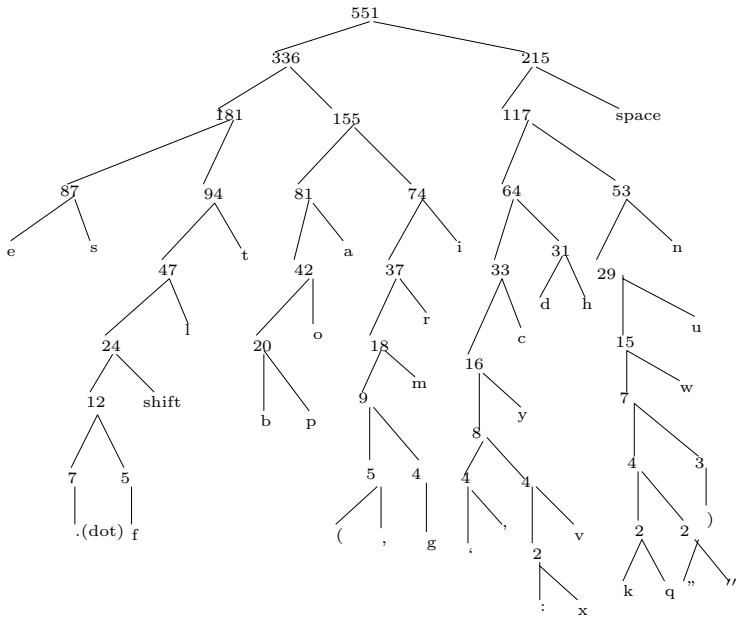


Figure 2: Huffman Code Tree (b)

When we use this encoding, the length of the string is 2365.

(c) In this case, the “shift encoding” turns out better than the normal encoding. If there are quite a few capital letters used then the shift encoding turns better till a certain point as it improves the representation for the corresponding small letters and for the “shift” key itself. But as the fraction of capital letters in the text increases (say the text consists only of capital letters), then the shift encoding obviously yields a much larger encoded text and is not useful.

3. **Amortization (20 Points)**
Do Exercise 1.1 in the handout on amortization. We generalize the example of incrementing binary counters. Suppose we have a collection of binary counters, all initialized to 0. We want to perform a sequence of operations, each of the type

$$\texttt{Inc}(C), \quad \texttt{Double}(C), \quad \texttt{Add}(C, C')$$

where C, C' are names of counters. The operation $\texttt{Inc}(C)$ increments the counter C by 1; $\texttt{Double}(C)$ doubles the counter C; finally, $\texttt{Add}(C, C')$ adds the contents of C' to C while simultaneously set the counter C' to zero. Show that this problem has amortized constant cost per operation.

To be precise, we need to define the cost model. The cost to double a counter C is just 1 (you only need to prepend a single bit to C). The cost of $\texttt{Add}(C, C')$ is the number of bits that the standard algorithm needs to look at (and possibly change) when when adding C and C'. E.g., if $C = 1001, 1101$ and $C' = 110$, then $C + C' = 1010, 0011$ and the cost is 9. This is because the algorithm only has to look at 6 bits of C and 3 bits of C'.

Let m be the number of counters and let us represent each binary counter C_i, $1 \leq i \leq m$, with a linked list that initially is empty. We also denote the value stored in C_i by 'C_i'. Let L_i be the length of C_i, O_i be the number of 1's in C_i, and E_i be the length of the maximum suffix of 1's in C_i in the current state.
Now define the potential function of the set of counters as follows:

$$\Phi = \sum_{k=1}^{m} (L_i + O_i).$$

For each operation on the set of counters, we will show that it has a constant amortized cost. Consider the most interesting case of $\texttt{Add}(C_1, C_2)$. In this case, $\Delta\Phi \leq -\min(L_1, L_2) - K$ where K is the number of carry bits beyond $\min(L_1, L_2)$ in the addition process. This release enough potential, except for some small constant A, to pay for the cost of addition. This small constant A can be our amortized cost.

(a) Inc(C_i): The actual cost is $E_i + 1$ since it resets E_i bits and sets one bit to a 1. The number of 1's in C_i after this operation is $O_i - E_i + 1$ and the length of C_i is at most $L_i + 1$. Thus the potential difference is

$$\begin{aligned} \Delta\Phi &\leq [(O_i - E_i + 1) + (L_i + 1)] - [O_i + L_i] \\ &= 2 - E_i. \end{aligned}$$

The amortized cost is therefore

$$(E_i + 1) + \Delta\Phi \leq (E_i + 1) + (2 - E_i) = 3.$$

(b) Double(C_i): The actual cost is 1 since it is the cost of appending a 0-bit to the end of the linked list for C_i. After this operation, the number of 1's is not changed but the length is increased by 1. Thus the potential difference is 1. Therefore the amortized cost is 2.

(c) Add(C_i, C_j): Let us assume that $C_i \geq C_j$. Let t_{01} be the number of times a bit is flipped from 0 to 1 in C_i and t_{10} be the number of times a bit is flipped from 1 to 0 when C_j is added to C_i. After the addition, the number of 1's in C_i is $O_i + t_{01} - t_{10}$ and the length of C_i is at most $L_i + 1$. The actual cost of the addition is at most $L_j + t_{10} + 1$ because all bits of C_j is scanned to be added to C_i and after that a carry will flip bits of C_i from 1 to 0. Thus the potential difference is

$$\begin{aligned} \Delta\Phi &\leq [(L_i + 1) + (O_i + t_{01} - t_{10})] - [(L_i + L_j) + (O_i + O_j)] \\ &= 1 + t_{01} - t_{01} - L_j - O_j. \end{aligned}$$

Therefore the amortized cost is

$$\begin{aligned} (L_j + t_{10} + 1) + (1 + t_{01} - t_{10} - L_j - O_j) &= 2 + (t_{01} - O_j) \\ &\leq 2, \end{aligned}$$

since $O_j \geq t_{01}$.

4. **Splay Trees (15 Points)**
Do Exercise 2.1 in the handout on amortization. Perform the following splay tree operations, starting from an initially empty tree.

$$Ins(3, 2, 1, 6, 5, 4, 9, 8, 7), LookUp(3), Del(7), Ins(12, 15, 14, 13), Split(8).$$

Show the splay tree after each step.

SOLUTION:

Please refer to figure 3

5. **Prim's Algorithm (20 Points)**
Hand simulate Prim's algorithm on the following graph (figure 4) beginning with v_1.

It amounts to filling in the following table, row by row. The vertex set V is partitioned into the two sets, $S \subseteq V$ and $U = V \setminus S$. The set U is in a priority queue. Each row of the table represents the array $lc[u]$ ($u \in U$) where $lc[u]$ is the least cost of an edge that connects u to any node in S. We also maintain a set T of edges. The set T forms a minimum spanning tree for S. The table also has an entry $mst[T]$ for the sum of the weights of edges in T. We have filled in the first two rows already.

i	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	$mst[T]$	New Edge
1	$\underline{1}$	3	2	∞	∞	∞	∞	∞	∞	∞	∞	1	(v_1, v_2)
2	*	"	$\underline{2}$	7	"	"	"	"	"	"	"	3	(v_1, v_4)

Note that the minimum cost in each row is underscored, indicating the item to be removed from the priority queue (or the set U). A 33 entry just means it is unchanged form before. An * entry means that the node is no longer in U.

SOLUTION: The full table is as follows:

i	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	$mst[T]$	New Edge
1	$\underline{1}$	3	2	∞	∞	∞	∞	∞	∞	∞	∞	1	(v_1, v_2)
2	*	"	$\underline{2}$	7	"	"	"	"	"	"	"	3	(v_1, v_4)
3	*	3	*	"	7	"	1	6	"	"	"	4	(v_4, v_8)
4	*	$\underline{3}$	*	"	"	"	*	3	"	"	"	7	(v_1, v_3)
5	*	*	*	"	"	"	*	$\underline{3}$	"	"	"	10	(v_8, v_9)
6	*	*	*	"	$\underline{4}$	"	*	*	6	"	"	14	(v_9, v_6)
7	*	*	*	"	*	2	*	*	$\underline{1}$	4	"	15	(v_6, v_{10})
8	*	*	*	"	*	$\underline{2}$	*	*	*	2	"	17	(v_6, v_7)
9	*	*	*	5	*	*	*	*	*	$\underline{2}$	6	19	(v_7, v_{11})
10	*	*	*	$\underline{5}$	*	*	*	*	*	*	"	24	(v_7, v_{11})
11	*	*	*	*	*	*	*	*	*	*	$\underline{6}$	30	(v_7, v_{12})

6. **Kruskal's Algorithm and Union Find (10+20 Points)**
We want to simulate Kruskal's algorithm on the same graph as the previous question.

(a) First show the list of edges, sorted by non-decreasing weight. View vertices v_1, v_2, v_3, etc as the integers $1, 2, 3$, etc. We want you to break ties as follows: assume each edge has the form (i, j) where $i < j$. When the weights of (i, j) and (i', j') are equal, then we want (i, j) to appear before (i', j') iff (i, j) is less than (i', j') in the lexicographic order, i.e., either $i < i'$ or $(i = i'$ and $j < j')$.

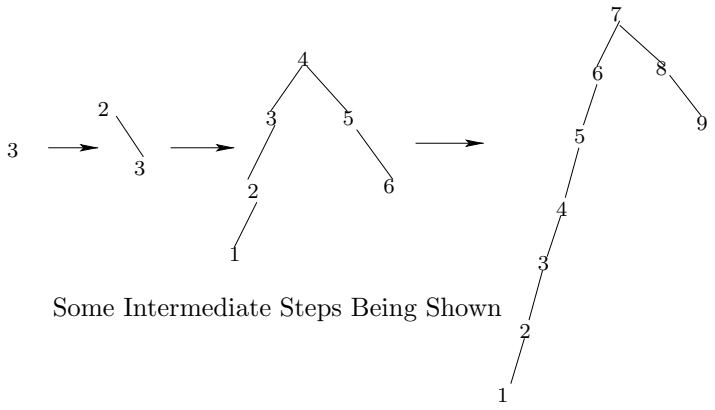
(b) We want you to maintain the union-find data structure needed to answer the basic question in Kruskal's algorithm (namely, does adding this edge creates a cycle?). The algorithms for the Union and Find MUST use the 2 heuristics we discussed: rank heuristic and path compression. At each stage of Kruskal's algorithm, when we consider an edge (i, j), we want you to perform the corresponding $Find(i), Find(j)$ and, if necessary, $Union(Find(i), Find(j))$. You must show the result of each of these operations on the union-fine data structure. SOLUTION:

(a) The edges listed in non-decreasing order of weight and in lexicographic order is as follows:

weight	Edges in lexicographic order
1	$(1,2),(4,8),(6,10)$
2	$(1,4),(6,7),(7,11),(10,11)$
3	$(1,3),(3,9),(7,10),(8,9)$
4	$(6,9),(6,11)$
5	$(5,7)$
6	$(3,4),(4,9),(7,12),(9,10)$
7	$(2,5),(3,8),(4,6)$
8	$(4,5),(5,12)$
9	$(11,12)$

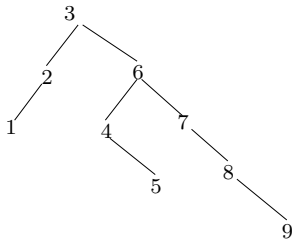
(b) Please refer to figure 5.

After Ins$(3, 2, 1, 6, 5, 4, 9, 8, 7)$

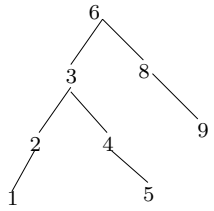


Some Intermediate Steps Being Shown

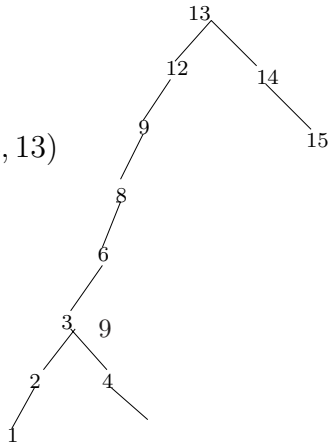
LookUp(3)

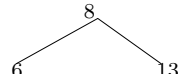


Del(7)



Ins$(12, 15, 14, 13)$





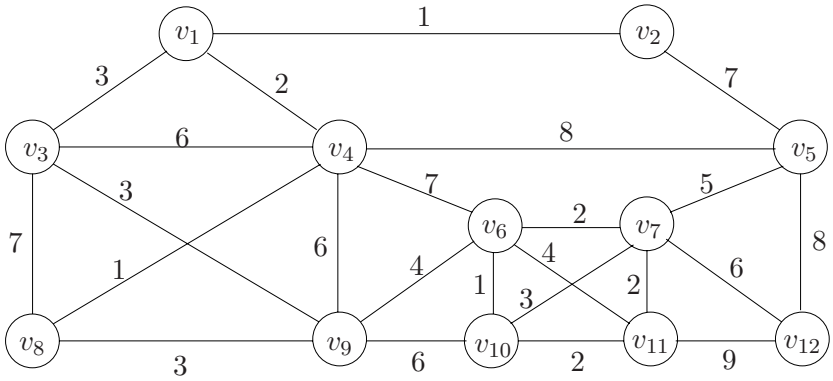
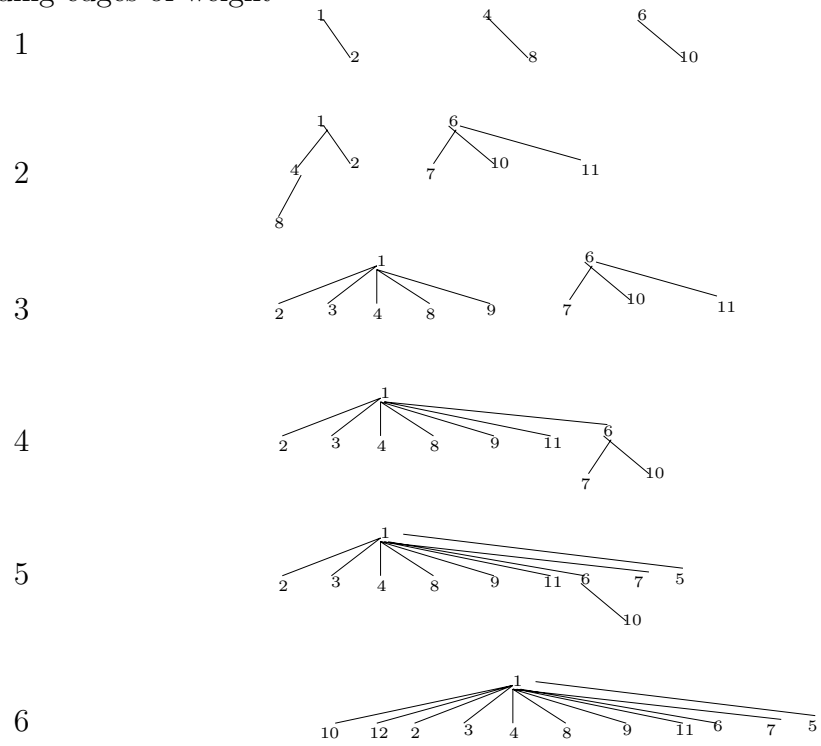


Figure 4: Graph of a House

After adding edges of weight



At this stage, all the vertices are in the MST.

The edges taken are $(1,2),(4,8),(6,10),(1,4),(6,7),(7,11),(1,3),(3,9),(6,9)$ $(5,7),(7,12)$

Figure 5: Kruskal's Algorithm with Union-Find