CSE 373 Analysis of Algorithms May 14, 2014

Solutions to HW 5

Problem 1

[Implementation of Edit-Distance.]

Problem 2

[8–3]

(a) Say we are given strings A and B of lengths n and m. Then for every i,j, $1 \leq i \leq n$, and $1 \leq j \leq m$, we need to know what is the longest common suffix between the two strings that ends respectively at positions i and j. This gives us a matrix M of dimensions $n \times m$, which we then iterate and find the max value in it. We generate the values in the matrix as follows: if $A[i] = B[j]$ then $M[i, j] = M[i-1, j-1] + 1$, else $M[i, j] = 0$. The base case is $M[i, 0] = 0$ for any i and $M[0, j] = 0$ for any j.

(b) We fix string A and shift string B against it. For each shift we go through aligned characters and record consequitive matches. If we encountered a list of matches longer than have seen before, we save it. The pseudo-code is below:

```
int max_len=0;
String max_str=‘‘’’;
for (int i=0; i<A.length; i++) {
  boolean match = false;
  int cur_len=0;
  String cur_str = ‘‘’’;
  for (int j=0; j<min(B.length, A.length-i); j++) {
    if (A[i+j]==B[j]) {
      cur_len++;
      cur_str.append(B[j]);
    } else {
      if (match) {
        if(cur_len>max_len) {
          max_len = cur_len;
          max_str = cur_str;
        }
        match = true;
        cur_len=0;
        cur_str = ‘‘’’;
      }
      match = false;
    }
```

```
  }
  if(cur_len>max_len) {
    max_len = cur_len;
    max_str = cur_str;
  }
}
```

[8–7]

(a) There are 7 different ways or C(20) = 7. 20 equals to:
1. Twenty 1
2. Fourteen 1, one 6
3. Four 1, two 6
4. Two 1, three 6
5. Ten 1, One 10
6. Four 1, One 6, One 10 7. Two 10

(b) We can solve this problem using Dynamic Programming. We fill matrix C, for which $C[n, k]$ is the number of ways to make a change of n, using first k denominations only.

$C[n, k] = C[n - d_k, k] + C[n, k - 1]$

PROBLEM 3

[8-14] Take $P_w(g, i)$ and $P_b(g, i)$ are respectively the probabilities that a champion wins a championship where there are g games left, and i games are required for her to win, while currently playing as white (black). Then we have the following recurrences:

$P_w(g, i) = w_w P_b(g - 1, i - 1) + w_d P_b(g - 1, i - 1/2) + w_l P_b(g - 1, i).$

Similarly,

$P_b(g, i) = b_w P_w(g - 1, i - 1) + b_d P_w(g - 1, i - 1/2) + b_l P_w(g - 1, i).$

After filling in these two matrices, we obtain the probability that a champion wins an entire championship by looking at an entry $P_w[0, 24]$. The running time of the algorithm is quadratic.

PROBLEM 4

[8–18] Suppose the greedy algorithm didnt give the optimal hight. Lets call the total height given by greedy algorithm H_G. Now consider a non-greedy ordering in which at least one shelf is not filled with books (empty or partially filled) and books are put in the next shelf. Now think about the last shelf and the possible consequence of this change. Leaving a middle shelf partially empty, can make us make an extra shelf on top to put all the book that are pushed forward and dont fit in the last shelf anymore. This means an extra shelf and since all the books have the save heigh h_b, the total height, H_T , is now at least $H_G + h_b$ ($HT \geq H_G + h_b$). Note that leaving some shelves not completely filled would not make an extra shelf as long as there is enough space in the following shelves including the last shelf for all the books that are

pushed forward. This means $H_T = H_G$. Therefore, library height given by greedy is always less than or equal to total height given by any other ordering which means $H_G = H_{opt}$. There are n books and the order is given so we put book 1 to n in O(n) time.

[8–19] (a) Consider the sequence of books with the following heights: 9, 10, 11, 4, 15. Say the width of the shelf is such that the books of heights 9, 10 and 11 could fit on the shelf, and the total height of both shelves would be 11+15 = 26. If we move 11 to the second shelf, then the total height is 10 + 15 = 25.

(b) $C(i)$ – total height of the selves for arrangement of books 0 to i.

$$C(i) = \min_{i \le k \le m} (C(k+1) + \min(h_i, ..., h_k)),$$

where m is the maximum number, such that $\sum_{j=i}^{m} h_j \le L$

Problem 5

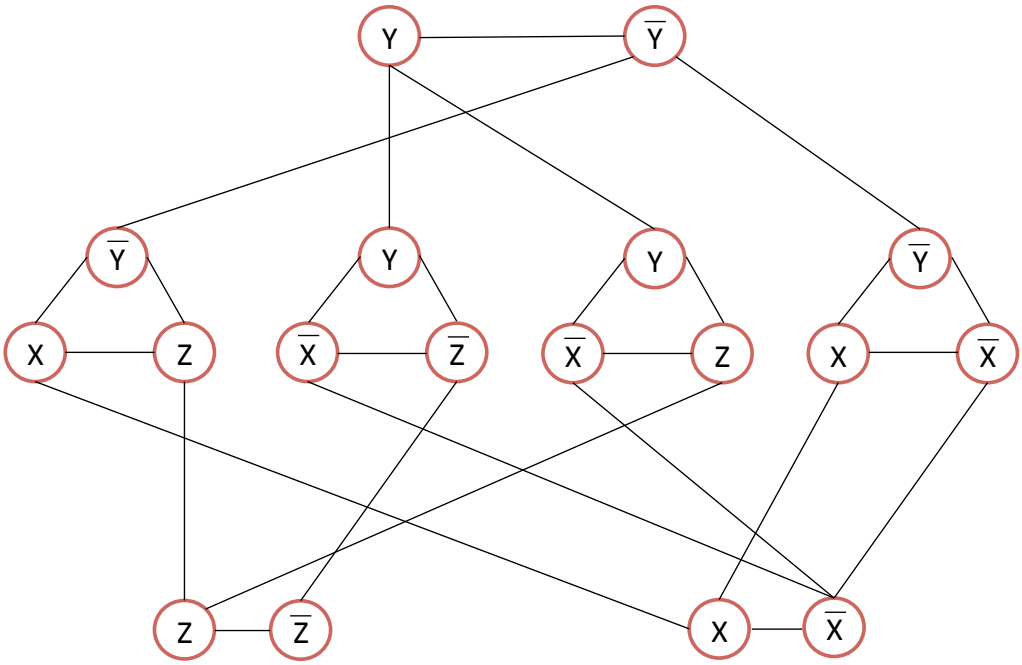
[9–1] $F = (x + y + \bar{z} + w + u + \bar{v}) \cdot (\bar{x} + \bar{y} + z + \bar{w} + u + v) \cdot (x + \bar{y} + \bar{z} + w + u + \bar{v}) \cdot (x + \bar{y})$

Just substotute each clause in F with the following:

$$(x + y + \bar{z} + w + u + \bar{v}) = (x + y + L_1) \cdot (\bar{L_1} + \bar{z} + L_2) \cdot (\bar{L_2} + w + L_3) \cdot (\bar{L_3} + u + \bar{v})$$
$$(\bar{x} + \bar{y} + z + \bar{w} + u + v) = (\bar{x} + \bar{y} + N_1) \cdot (\bar{N_1} + z + N_2) \cdot (\bar{N_2} + \bar{w} + N_3) \cdot (\bar{N_3} + u + v)$$
$$(x + \bar{y} + \bar{z} + w + u + \bar{v}) = (x + \bar{y} + M_1) \cdot (\bar{M_1} + \bar{z} + M_2) \cdot (\bar{M_2} + w + M_3) \cdot (\bar{M_3} + u + \bar{v})$$
$$(x + \bar{y}) = (x + \bar{y} + K)(x + \bar{y} + \bar{K})$$

Problem 6

[9–2]



PROBLEM 7

[9–8] Reduce vertex cover to baseball collector problem. Given a graph $G = (V, E)$, for each node $v_i \in V$, we have a packet P_i. For each edge incident on v_i, we take one card. That is card c_i is in P_j *iff* edge e_i is incident to vertex v_j in the graph G. So to solve the vertex cover problem with input k, we can just solve the baseball card collector with input k and return the same answer.

As vertex cover is NP hard, by this reduction we get that baseball card collector problem is also NP hard.

[9-10] To prove that a problem A is NP-complete, you need to prove that it is in NP and a known NP-complete problem B can be reduced to A. Proving that a problem is in NP is easy, just write a polynomial time verifier that output of A is correct. Then if you find a reduction from A to B, it means that if you could solve A faster than NP-complete, than you could solve B faster, which is impossible. Thus, your problem A is NP-complete. For the key we will not give a proof that a problem is in NP, but you need to provide it. We give only an example of a reduction.

We can reduce Clique problem, which is known to be NP-complete, to the dense subgraph problem. Given input to Clique problem, graph G and integer k, we set $y = \frac{k(k-1)}{2}$ and run Dense Subgraph probem with G, k and y. If graph has a dense subgraph with k vertices and

at least y edges, then G has a clique of k vertices.

PROBLEM 8

[9–11]

To prove that Clique-NoClique is NP-complete we can choose any known NP-complete problem, for example, Clique, and reduce it to Clique-NoClique. To reduce Clique to Clique-NoClique, we simply add k new unconnected vertices to G, input to Clique, and run Clique-NoClique algorithm on the new graph, which will return clique of size k and independent set of size k. The new added nodes will not be included in clique, thus the found clique can be only for graph G. This means that by solving Clique-NoClique, we can solve Clique. Thus Clique-NoClique is NP-complete

[9–12] From the hint, we already know that in the special case of a graph in which every vertex has degree 3, Hamiltonian circuit problem is NP-hard. Now, in this specific type of graph, consider the possible Eulerian cycles. When visiting a vertex, one edge is used to enter the vertex, one edge to leave the vertex. As there are three edges that each vertex is incident on, there is one edge left. In Eulerian cycle you would never be able to use the third edge, because if you do, that means you are visiting the vertex again, and you would have to use an already visited edge to leave the vertex and that is using an edge twice which is against Eulerian cycle. So for every vertex in your cycle you visit it only one time which is a Hamiltonian cycle. Therefore, using this specific graph we showed that the problem of finding the maximal Eulerian cycle (maximum number of edges) reduced from a maximal Hamiltonian cycle problem (maximum number of vertices) and therefore maximal Eulerian problem is NP-complete.

PROBLEM 9

[9–13]

We reduce vertex cover problem to it. Input to vertex cover is graph G containing a set of vertices V, a set of edges E and a positive integer k. We can view each element in E as a subset of V containing only 2 elements (2 vertices, connected by an edge). We set $S = V$ and $C = E$ and use S, C and k as an input for the Hitting Set problem. The Hitting Set blackbox returns S', a subset of $S = V$ of size k, such that at least one element of a set from $C = E$ is in S'. This means that each vertex adjacent to an edge from E, is included into S'. Thus, each edge is covered and S' forms a vertex cover of k vertices.