**Free Groups Problem from last Midterm**

**public** **boolean** parenthesize(String s)

{

String answer = **new** String("");

answer = recursive(s);

**return** answer.contains("a");

}

**private** String recursive(String s)

{

String result = **new** String("");

**if**(s.length() == 1)

{

result += "" + s.charAt(0);

**return** result;

}

**else** **if**(s.length() == 2)

{

result += "" + getval(s.charAt(0), s.charAt(1));

**return** result;

}

**else**

{

**char** sub;

**for**(**int** i = 0; i < s.length() - 1; i++)

{

String keep = **new** String("");

**if**(i != 0)

{

**if**(i == 1)

{

keep += "" + s.charAt(0);

}

**else**

{

keep += "" + s.substring(0, i - 1);

}

}

sub = getval(s.charAt(i), s.charAt(i+1));

keep += "" + sub;

**if**(i != s.length() - 2)

{

**if**(i+2 == s.length() - 1)

{

keep += "" + s.charAt(i+2);

}

**else**

{

keep += "" + s.substring(i+2, s.length());

}

}

result += recursive(keep);

}

}

**return** result;

}

**private** **char** getval(**char** one, **char** two)

{

**return** matrix[((**int**)one) - 97][((**int**)two) -97];

}

**Dijkstra's alg.**

**Dijkstra(G,w,s)**

Initialize-Single-Source(G,s)

S<- null

Q <- V[G]

while Q != null

do u <- Extract-Min(Q)

S <- S Union {u}

for each vertex v in Adj[u]

do Relax(u,v,w)

**Initialize-Single-Source(G,s)**

for each vertex v in V[G]

do d[v] <- infinity

pi[v] <- null

d[s] <- 0

**Relax(u,v,w)**

if d[v] > d[u] + w(u,v)

then d[v] <- d[u] + w(u,v)

pi[v] <- u

**Is Text problem from Midterm 1**

**public** **boolean** isText()

{

ArrayList<Integer> start = **new** ArrayList<Integer>(0);

ArrayList<Integer> end = **new** ArrayList<Integer>(0);

Integer[] startf = **new** Integer[0];

Integer[] endf = **new** Integer[0];

**int** counter = 0;

**for**(**int** i = 0; i < text.length(); i++)

{

**for**(**int** j = i; j < text.length(); j++)

{

**if**(dict(i,j))

{

start.add(counter, i); end.add(counter, j);

counter++;

}

}

}

//Modified Scheduling Algorithm

//Sort jobs by finish times

startf = (Integer[]) start.toArray(startf);

endf = (Integer[]) end.toArray(endf);

//Bubble sort end times

**int** temp, count, index;

**for**(count = 0; count < endf.length-1; count++)

{

**for**(index = 0; index < endf.length-1-count; index++)

{

**if**(endf[index] > endf[index+1])

{

temp = endf[index];

endf[index] = endf[index+1];

endf[index+1] = temp;

temp = startf[index];

startf[index] = startf[index+1];

startf[index+1] = temp;

}

}

}

**return** TestIt(startf, endf, 0);

}

**public** **boolean** TestIt(Integer[] start, Integer[] end, **int** i)

{

**if**(i == 0)

{

**for**(**int** j = 0; j < start.length; j++)

{

**if**(start[j] == 0)

{

**if**(TestIt(start, end, end[j] + 1))

{

**return** **true**;

}

}

}

**return** **false**;

}

**else** **if**(i == text.length())

{

**return** **true**;

}

**else**

{

**for**(**int** j = 0; j < start.length; j++)

{

**if**(start[j] == i)

{

**if**(TestIt(start,end,end[j] + 1))

{

**return** **true**;

}

}

}

**return** **false**;

}

}

**Rod Cutting**

**public** **int** cutIterative(**int** N)

{

**int**[] Cost = **new** **int**[N + 1];

Cost[1] = Price[1];

**for**(**int** j = 2; j <= N; j++)

{

Cost[j] = Price[j];

**for**(**int** i = 1; i < j; i++)

{

**if**(Cost[j] < (Cost[i] + Cost[j-i]))

{

Cost[j] = Cost[i] + Cost[j-i];

}

comp++;

}

}

**return** max(Cost);

}

**public** **int** cutRecursive(**int** N)

{

**int**[] Cost = **new** **int**[N + 1];

Cost[0] = Price[N];

**for**(**int** i = 1; i < N; i++)

{

Cost[i] = cutRecursive(i) + cutRecursive(N-i);

comp++;

}

**return** max(Cost);

}

**public** **int** cutRecursiveMem(**int** N)

{

**int**[] Solutions = **new** **int**[N + 1];

**for**(**int** i = 0; i < Solutions.length; i++)

{

Solutions[i] = Integer.*MIN\_VALUE*;

}

**return** MemorizedSolution(Solutions, N);

}

**private** **int** MemorizedSolution(**int**[] Solutions, **int** i)

{

**if**(Solutions[i] != Integer.*MIN\_VALUE*)

{

**return** Solutions[i];

}

**else**

{

**if**(i == 0)

{

Solutions[i] = 0;

}

**else**

{

**int** temp = Price[i];

**for**(**int** j = 1; j < i; j++)

{

temp = Math.*max*(temp,Price[j]+MemorizedSolution(Solutions, i-j));

comp++;

}

Solutions[i] = temp;

}

}

comp++; //if statement

**return** Solutions[i];

}

**Longest Common Subsequence**

**public** **void** findLCS(String x, String y)

{

ax = x;

ay = y;

M = x.length();

N = y.length();

opt = **new** **int**[M+1][N+1];

//Compute length of LCS and all subproblems

**for**(**int** i = M-1; i >= 0; i--)

{

**for**(**int** j = N-1; j >= 0; j--)

{

**if**(x.charAt(i) == y.charAt(j))

{

opt[i][j] = opt[i+1][j+1] + 1;

}

**else**

{

opt[i][j] = Math.*max*(opt[i+1][j], opt[i][j+1]);

}

}

}

}

**public** String getLCS()

{

String ret = **new** String("");

**int** M = ax.length();

**int** N = ay.length();

**int** i = 0, j = 0;

**while**(i < M && j < N) {

**if** (ax.charAt(i) == ay.charAt(j)) {

ret+=(ax.charAt(i));

i++;

j++;

}

**else** **if** (opt[i+1][j] >= opt[i][j+1]) i++;

**else** j++;

}

**return** ret;

}