CSCI 570 Spring 2015 Discussion 8

1. A palindrome is a string that reads the same forwards and backwards. Suppose we are given a string $S = s_1 s_2 s_3 s_n$, and we want to find the longest palindrome that can be formed by deleting zero or more characters from the string.

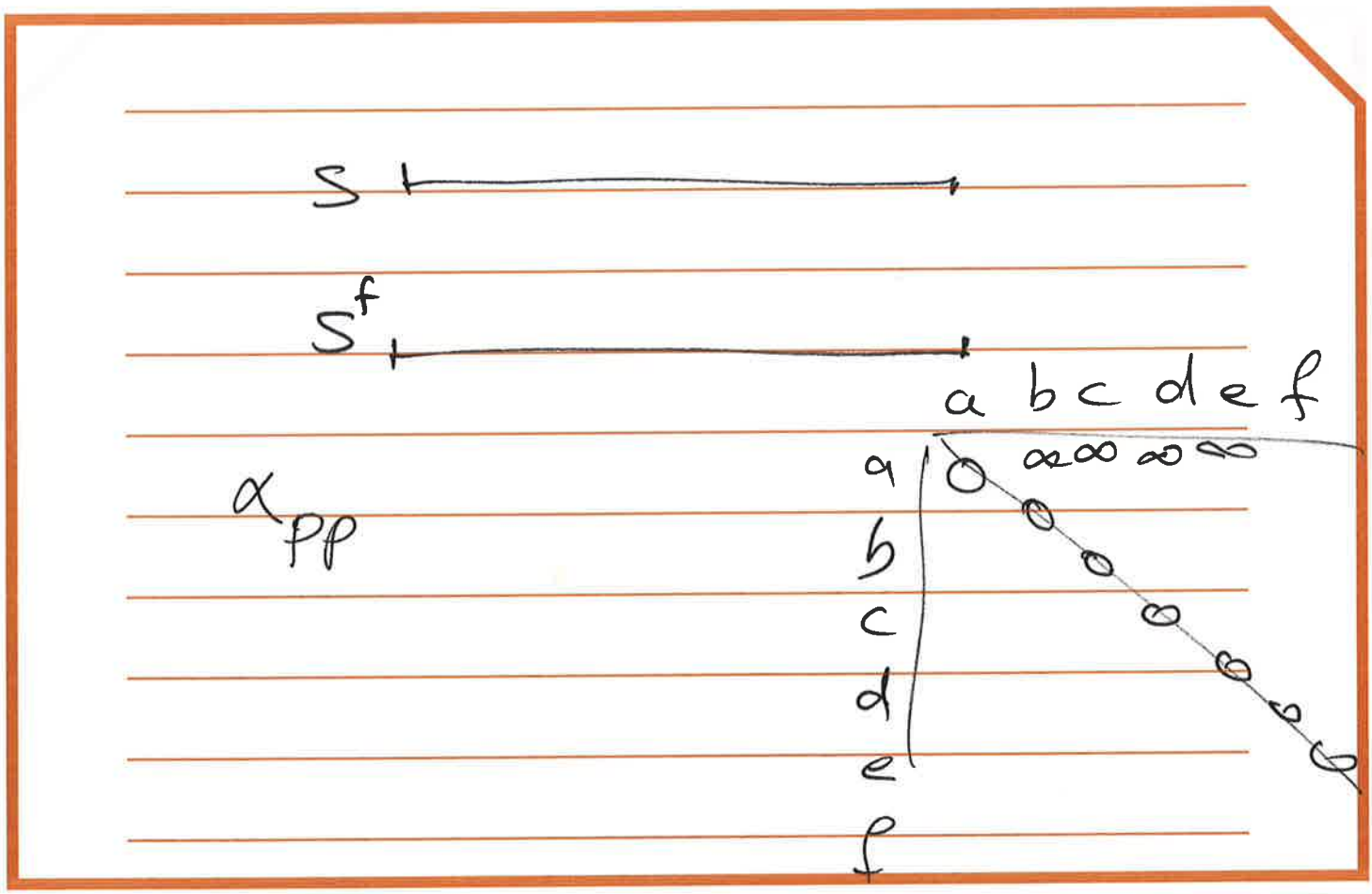
2. When their respective sport is not in season, USC's student-athletes are very involved in their community, helping people and spreading goodwill for the school. Unfortunately, NCAA[1] regulations limit each student-athlete to at most one community service project per semester, so the athletic department is not always able to help every deserving charity. For the upcoming semester, we have S student-athletes who want to volunteer their time, and B buses to help get them between campus and the location of their volunteering[2]. There are F projects under consideration; project i requires s_i student-athletes and b_i buses to accomplish, and will generate $g_i > 0$ units of goodwill for the university. Our goal is to maximize the goodwill generated for the university subject to these constraints.
   Note that each project must be undertaken entirely or not done at all -- we cannot choose, for example, to do half of project i to get half of g_i goodwill.
   For full credit, your algorithm should have runtime $O(FBS)$.

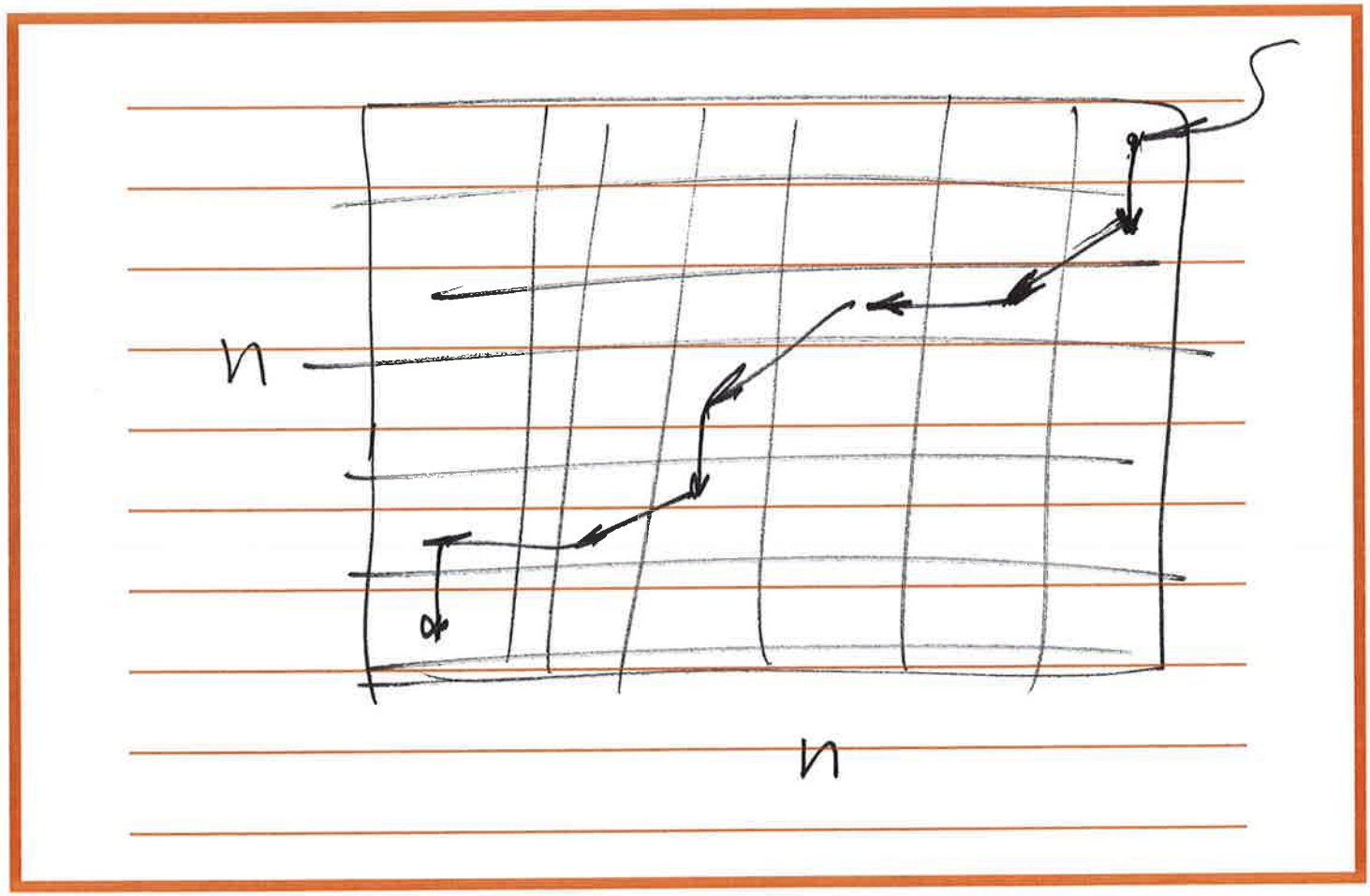
3. Suppose you are organizing a company party. The corporation has a hierarchical ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Each employee j has a value v_j (a positive integer), representing how enjoyable their presence would be at the party. Our goal is to determine which employees to invite, subject to these constraints, to maximize the total value of invitees.

---

[1] Regulations mentioned in this problem are not necessarily accurate to reality and should be considered parody.

[2] The NCAA also won't permit student-athletes to use their own vehicles for these purposes. Furthermore, each bus can only be used for one project.


S
S^f
a b c d e f
a
b
c
d
e
f
∞ ∞ ∞ ∞
α_PP


n
n

S 

Case 1 – if $S_1 = S_n$ then

$$OPT(1,n) = OPT(2,n-1) + 2$$

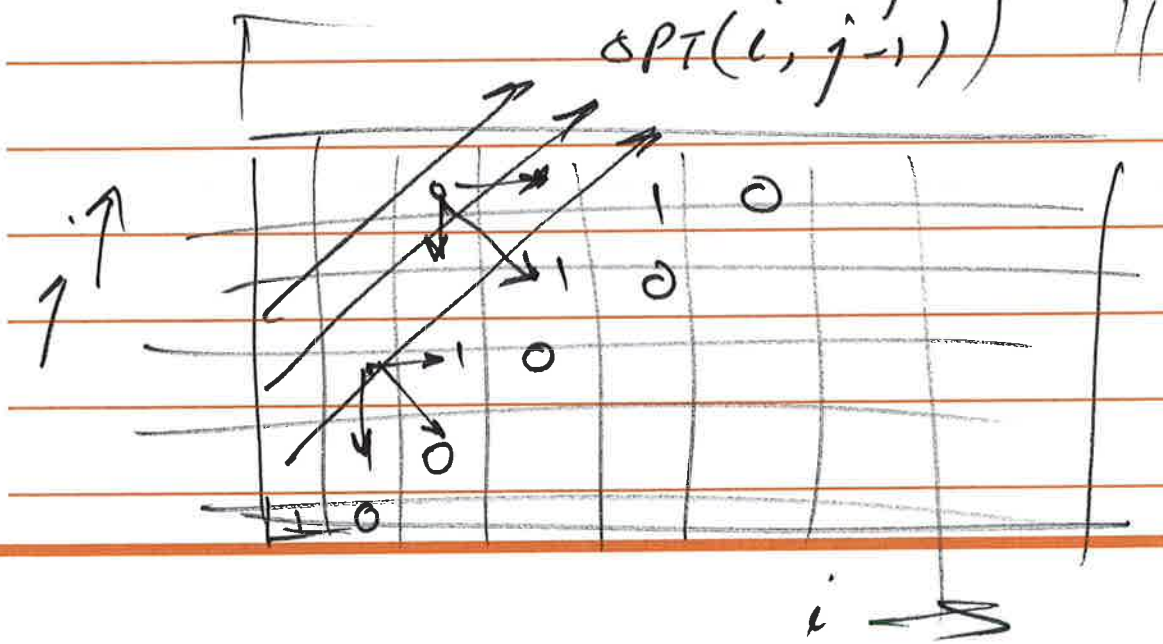
Case 2 – if S_1 is not part of the palindrome

$$OPT(1,n) = OPT(2,n)$$

Case 3 – if S_n is not part of the palindrome

$$OPT(1,n) = OPT(1,n-1)$$

$$OPT(i,j) = \max\begin{cases} 2 + OPT(i+1,j-1) & \text{if } S_i = S_j, \\ OPT(i+1,j), \\ OPT(i,j-1) \end{cases}$$



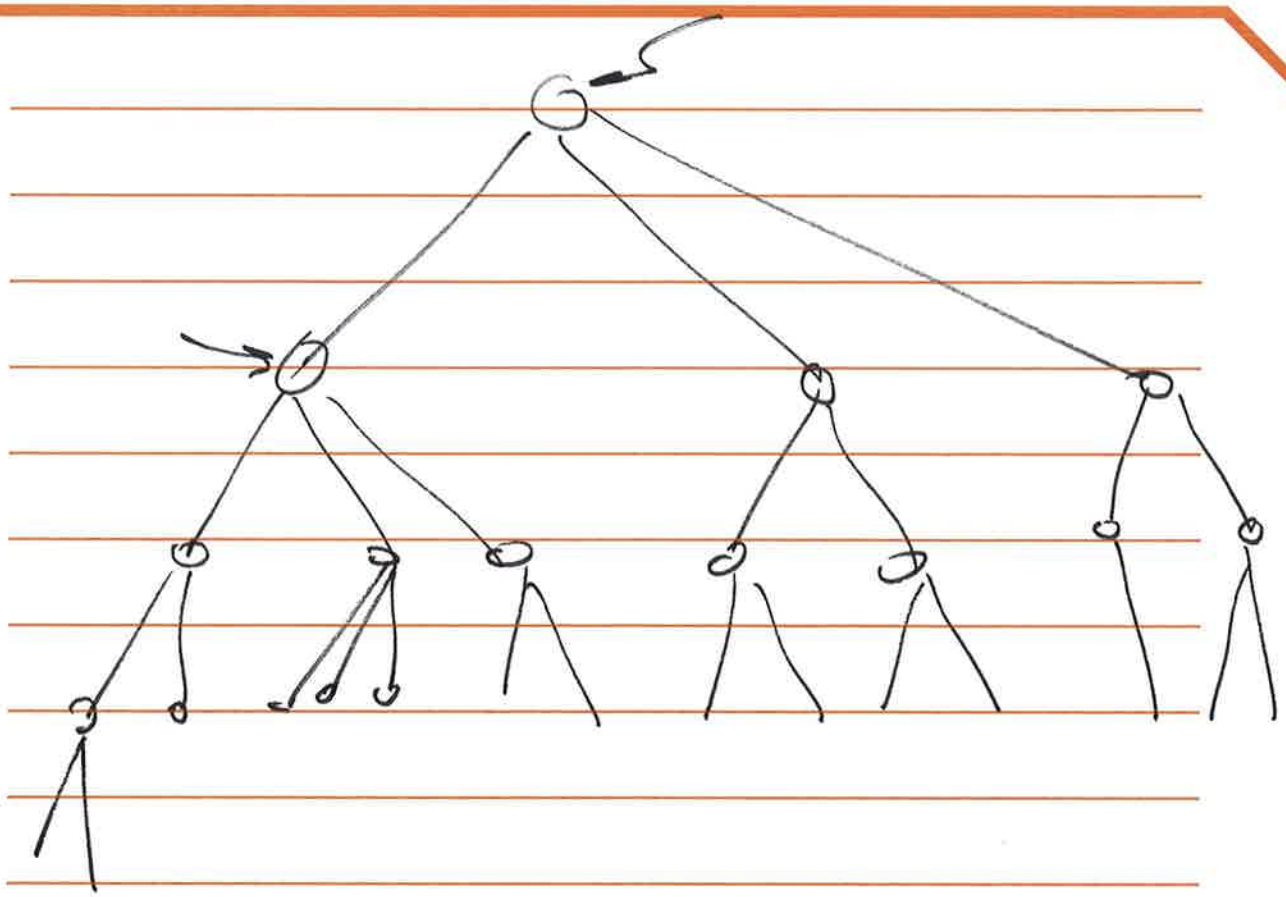
$OPT(F, B, S)$ represents the Max. goodwill we can get from projects 1..F w/ B bases and S students

if $B < b_F$ or $S < s_F$, we can't do project F

$\Rightarrow OPT(F, B, S) = OPT(F-1, B, S)$

otherwise

$$Max.\left(g_F + OPT(F-1, B-b_F, S-s_F),\right.$$
$$\left. OPT(F-1, B, S)\right)$$



$OPT(r)$ is the opt. value of the subtree rooted at r.

if r is part of the opt. sol.

$$\Rightarrow OPT(r) = V_r + \sum opt(g)$$

if r is not part of the opt. sol.

$$\Rightarrow OPT(r) = \sum opt(c)$$

1: Define OPT(i,j) to be the size of the largest palindrome of s_i .. s_j.
Our base case is OPT(i,i) = 1. OPT(i, i-1) = 0 (empty) If i < j, we go onto recursive cases:
If si = sj, we keep both letters and OPT(i,j) = 2 + OPT(i+1, j-1)
Otherwise, we must toss one: OPT(i,j) = max(OPT(i+1, j) , OPT(i, j-1))

The trick to iterative is to fill in the diagonals first.

```
For i = 1 to n-1
        OPT[i,i] = 1
        OPT[i, i-1] = 0
For d = 1 to n
        for i = 1 to n-d
                j = i + d
                fill in OPT[i,j] as per above.
```

to extract palindrome:

```
i = 1, j = n
while i < j
        if si == sj
                i++;
                j--
        else if OPT[i,j] == OPT[i+1, j]
                mark si as to be deleted.
                i++
        else
                mark sj as to be deleted.
                j--
```

2: Define OPT(F,B,S) to be the maximum goodwill we can get from projects 1..F with B buses and S students. If B < b_F or S < s_F, we can't do project F (not enough resources), and OPT(F,B,S) = OPT(F-1, B, S).

Otherwise, we have to decide whether or not to do project F. If we do it, we get its value, plus we can then choose some optimal subset of the remaining projects with the remaining resources: gF + OPT(F-1, B-b_F, S-s_F). Otherwise, we get OPT(F-1, B, S) by omitting it but not spending resources on it. We choose the larger of these.

We can fill in the array iteratively:

```
for i = 1 to F
        for j = 1 to B
                for k = 1 to S
                        fill in OPT[i,j,k] by recurrence above.
```

To get the projects:

```
i = F , j = B, k = S;
while i > 0 :
        if OPT[i,j,k] == OPT[i-1, j, k]
                i = i -1
        else
                output "do project i"
                j = j - b_i
                k = k - s_i
                i = i - 1
```

3: Define OPT(r) to be the optimal value of the subtree rooted at node r. Either node r is selected or isn't; if it is selected, we cannot selects its direct children; we get value v_r + SUM_g OPT(g). If we don't select it, we get SUM_c OPT(c), where g and c are the grand-children and children of node r, respectively. We take the max of those two choices. This technically incorporates the base-case of the leaf nodes (v_l) in it.
In linear time, we can fill in the table in depth first order.