PROBLEM SET 6

Zhiqiang Xie 77892769

Problem 1: Knapsack Variants

Here I set $OPT(i, w)$ to be the optimal solution with the weight constrain of w and items of $(1, \ldots, i)$

For i from 1 to n, w from 1 to W, apply a bottom-up method:

- If $i = 0$, $OPT(i, w) = 0$
- If $w < w_i$, $OPT(i, w) = OPT(i-1, w)$
- If $w_i \leq w < 2w_i$, $OPT(i, w) = max\{OPT(i-1, w), v_i + OPT(i-1, w-w_i)\}$
- If $2w_i \leq w$, $OPT(i, w) = max\{OPT(i-1, w), v_i + max\{OPT(i, w-w_i), OPT(i-1, w-w_i)\}\}$

Finally, $OPT(n, W)$ is the optimal answer, and we can trace back the DP table to get the corresponding combination. The time complexity $T(n) = O(n) * O(W) \in O(nW)$

Problem 2: Multiplication Card

Here I set $OPT(a, b)$ to be the optimal solution for the card set $C[a:b]$ (including a and b)

For all $a + 2 = b$ to $b - a = n - 1$, apply a bottom-up method:

- If $a + 1 < b$: $OPT(a, b) = min_{a+1 \leq j \leq b-1}\{OPT(a, j) + OPT(j, b) + c_a * c_j * c_b\}$ (Here j means the last card to be taken)
- If $a + 1 = b$: $OPT(a, b) = 0$

Finally, $OPT(1, n)$ is the optimal answer, and we can trace back the DP table to get the corresponding order of taking card. The time complexity $T(n) = O(\sum_{i=2}^{n-2}(i-1)(n-i)) \in O(n^3)$

Problem 3:

Set the start node r to be the root of the tree. And $OPT_0(u, i)$ to be the optimal solution for starting from node u and walk at most i step without returning back to u, $OPT_1(u, i)$ to be the optimal solution for starting from node u and walk at most i step with returning back to u.

initialization:

- For all each u in tree, $OPT_0(u, 0) = OPT_1(u, 0) = w(u)$, $OPT(u, -1) = 0$
- If u is leaf node, $OPT_0(u, i) = OPT_1(u, i) = w(u)$, $\forall i \in [1, k]$

For u From the leaf to the root (post-order), i from 1 to k, apply a bottom-up method:

- $OPT_1(u, i) = max_{v \in children(u)}\{max_{-1 \leq j \leq i-2}\{OPT_1(u, i-j-2) + OPT_1(v, j)\}\}$
- $OPT_0(u, i) =$
  $max_{v \in children(u)}\{max_{0 \leq j \leq i-1}\{OPT_0(u, i-j-2) + OPT_1(v, j), OPT_1(u, i-j-1) + OPT_0(v, j)\}\}$

Finally, $OPT(r, k)$ is the optimal answer, and we can trace back the DP table to get the corresponding routine. The time complexity $T(n) = O(n) * O(\sum_{i=1}^{k} i) \in O(nk^2)$

Problem 4:

Preprocessing:

- Precompute the level of every node (length of the path to the root).
  - This procedure costs $O(n)$ by BFS.
- Precompute the $2^j th$ ancestor for all valid j for each node in the rooted tree.
- Put the result above into a sparse 2D table P such that $P[i][j]$ stores the $2^j th$ ancestor for node i.
- This preprocess costs $O(n \log n)$ time and space.

If $level(v_i) = level(w_i)$:

- Every number can be expressed as a *sum* of distinct *powers* of 2.
- Keep referencing the table by finding the "largest uncommon ancestor" and finally we'll get $LCA(v_i, w_i) = k^{th}$
- This procedure costs $O(\log H)$

Else:

- Replace the node with high level to be its ancestor with the same level to another node.
  - This procedure could cost $O(\log H)$
- Repeat the procedure above.

Where H is the height of the tree.

Therefore, we can guarantee $T(n) = O(n \log n) + m * O(\log H) \in O((n+m)\log n)$

Problem 5:

1. For all $i+1=j$ to $j-i=n-1$, apply a bottom-up method:
   - Calculate $f(1,2), f(2,3), \ldots, f(n-1,n)$ and $w(1,2), w(2,3), \ldots, w(n-1,n)$ by summing
     - The first step costs $O((n-1)*1)$
   - Calculate $f(1,3), f(2,4), \ldots, f(n-2,n)$ and $w(1,3), w(2,4), \ldots, w(n-2,n)$ by the previous result and try different k.
     - This step costs $O((n-2)*2)$
   - ...

   $T(n) = O(\sum_{i=1}^{n-1} i(n-i)) \in O(n^3)$
2. $w(a,c) + w(b,d) = \sum_{k=a}^{d} a_k + \sum_{k=b}^{c} a_k = w(b,c) + w(a,d)$

   $w(b,c) = \sum_{k=b}^{c} a_k \le \sum_{k=a}^{b} a_k + \sum_{k=b}^{c} a_k + \sum_{k=c}^{d} a_k = \sum_{k=a}^{d} a_k = w(a,d)$
3. As for $0 < a < b < c < d$, we have:
   - $f(a,d) = f(a,t) + f(t+1,d) + w(a,d)$
   - $f(b,c) = f(b,k) + f(k+1,c) + w(b,c)$

   Without loss of generality, we can assume $a < t \le k \le c$:

- $f(a,c)+f(b,d)$
  $\leq f(a,t)+f(t+1,c)+w(a,c)+f(b,k)+f(k+1,d)+w(b,d)$
  $\leq f(a,t)+f(t+1,d)+w(a,d)+f(b,k)+f(k+1,c)+w(b,c)$
  $= f(a,d)+f(b,c)$
- Notice an induction is applied $f(t+1,c)+f(k+1,d) \leq f(t+1,d)+f(k+1,c)$ where $t \leq k$. And the equality of the base case is held.

Therefore, $f(a,c)+f(b,d) \leq f(a,d)+f(b,c)$

4. By symmetry, we need only to prove that $K(i,j) \leq K(i,j+1)$

   Since $f(a,c)+f(b,d) \leq f(a,d)+f(b,c)$, for all $0 < a < b < c < d$:

   - $f(b+1,d)+f(c+1,d+1) \leq f(c+1,d)+f(b+1,d+1)$
   - $f(a,b-1)+f(b,d)+w(a,d)+f(a,c-1)+f(c,d+1)+w(a,d+1)$
     $\leq f(a,b-1)+f(b,d+1)+w(a,d+1)+f(a,c-1)+f(c,d)+w(a,d)$
   - $f(a,b-1)+f(b,d)+w(a,d)-(f(a,c-1)+f(c,d)+w(a,d))$
     $\leq f(a,b-1)+f(b,d+1)+w(a,d+1)-(f(a,c-1)+f(c,d+1)+w(a,d+1))$
   - Therefore, for all $b < K(i,j)$, we have $f(a,d) \leq f(a,b-1)+f(b,d)+w(a,d)$. And it applies too for the case $f(a,d+1)$.
   - Which indicates that $K(a,d) \leq K(a,d+1)$

5. $f(i,j) = min_{K(i,j-1) \leq k \leq K(i+1,j)} \{f(i,k-1)+f(k,j)+w(i,j)\}$ if$(i < j)$

   Based on the procedure described above, we can accelerate the part of finding the right k.

   - $K(i,j-1) \leq k \leq K(i+1,j)$
   - Therefore, for each i in $f(a,a+i)$, it's supposed to find all k in $O(n)$ time.

   Therefore, $T(n) = O(\sum_{i=1}^{n-1} i) + n * O(n) \in O(n^2)$