Assignment 7

Q1. Polyglot Optimization

You are given a list of N projects. Each project requires proficiency in a certain set of languages. You are a polyglot programmer, skilled in a variety of languages, and can work on any project as long as you know all the required languages for that project.

Each project has a value associated with it, representing the reward or benefit of completing that project. You want to maximize your total reward by selecting a subset of projects such that for each project in the subset, you know all the required languages.

Input:

- An integer N representing the number of projects.
- An array values of size N, where values[i] is the value of the i-th project.
- A list of lists required_languages of size N, where required_languages[i] is a list of languages required for the i-th project.
- A set known_languages containing the languages you know.

Output:

- The maximum total value of the projects you can complete.

Example:

```
#include <bits/stdc++.h>
using namespace std;

int maxProjectValue(int N, vector<int>& values, vector<vector<string>>& required_languages, set<string>&
known_languages) {
   // Your dynamic programming solution goes here
   return max_value;
}

int main() {
   int N = ;
   vector<int> values = {50, 30, 70, 20, 90, 10, 60, 40, 80, 100};
   vector<vector<string>> required_languages = {{"Python", "C++"}, {"Java", "Python"}, {"C++", "Java", "Python"},
{"Python"}, {"Java", "Go"}, {"C++"}, {"Python", "Go"}, {"C++", "Python"}, {"Java", "Python", "Go"}, {"C++", "Java", "Go"}};
   vector <string>={"Python", "C++", "Java"};
 int result = maxProjectValue(N, values, required_languages, known_languages);
   cout << "Maximum Project Value: " << result << endl;
   return 0;
}
```

Q2. Knapsack Problem

You are given a list of N items, each with a weight and a value. You have a knapsack that can hold a maximum weight W. Additionally, you have some constraints and additional features:

1. Each item may have a dependency on another item. If you include item i, you must also include its dependent item j.
2. Some items can be grouped into categories, and you can select at most one item from each category.
3. Each item can be taken multiple times, but only up to a maximum limit L[i] for item i.

Your task is to determine the maximum value you can achieve while respecting the knapsack's weight limit, the dependency constraints, the category constraints, and the item repetition limits.

Input:

- An integer N representing the number of items.
- An integer W representing the maximum weight the knapsack can hold.
- An array weights of size N, where weights[i] is the weight of the i-th item.
- An array values of size N, where values[i] is the value of the i-th item.
- An array dependencies of size N, where dependencies[i] is the index of the item that i-th item depends on, or -1 if there is no dependency.
- An array categories of size N, where categories[i] is the category of the i-th item.
- An array limits of size N, where limits[i] is the maximum number of times the i-th item can be taken.

Output:

- The maximum value that can be achieved within the given constraints.

Example:

```
#include <bits/stdc++.h>
using namespace std;

int knapsack(int N, int W, vector<int>& weights, vector<int>& values, vector<int>& dependencies, vector<int>&
categories, vector<int>& limits) {
   // Your dynamic programming solution goes here
}

int main() {
   int N = 10;
   int W = 100;
   vector<int> weights = {10, 20, 30, 40, 15, 25, 35, 45, 55, 50};
   vector<int> values = {60, 100, 120, 240, 150, 90, 200, 170, 250, 300};
   vector<int> dependencies = {-1, 0, -1, 2, -1, -1, 1, 4, -1, 7} // Items 1 depends on 0, 3 depends on 2, 6 depends on 1, 7
depends on 4, 9 depends on 7
   vector<int> categories = {1, 0, 0, 1, 2, 2, 3, 3, 4, 4} // Categories constrain the selection to at most one item per category
vector<int> limits = {1, 2, 1, 1, 1, 1, 1, 1, 1, 1} // Maximum times each item can be taken
   int result = knapsack(N, W, weights, values, dependencies, categories, limits);
   cout << "Maximum Knapsack Value: " << result << endl;
   return 0;
}
```