Problem Solving Summary - June 30

1. Partition Equal Subset Sum

Problem: Given an array nums[], return true if it can be partitioned into two subsets with equal sum.
Approach: Use DP (0/1 Knapsack), check if subset with sum = total/2 exists.
Code:
```
bool canPartition(vector<int>& nums) {
    int sum = 0;
    for (int num : nums) sum += num;
    if (sum % 2 != 0) return false;
    int target = sum / 2;
    vector<bool> dp(target + 1, false);
    dp[0] = true;
    for (int num : nums) {
        for (int j = target; j >= num; --j)
            dp[j] = dp[j] || dp[j - num];
    }
    return dp[target];
}
```

2. Combination Sum

Problem: Find all unique combinations of candidates[] that sum to target (reuse allowed).
Approach: Backtracking with same index reuse.
Code:
```
void backtrack(vector<int>& c, int t, vector<vector<int>>& res, vector<int>& cur, int idx) {
    if (t == 0) res.push_back(cur);
    for (int i = idx; i < c.size() && c[i] <= t; ++i) {
        cur.push_back(c[i]);
        backtrack(c, t - c[i], res, cur, i);
        cur.pop_back();
    }
}
vector<vector<int>> combinationSum(vector<int>& c, int t) {
    sort(c.begin(), c.end());
    vector<vector<int>> res; vector<int> cur;
    backtrack(c, t, res, cur, 0);
    return res;
}
```

3. Read Vector Without Size

Approach: Use getline + istringstream.
Code:
```
vector<int> readVector() {
    vector<int> vec; string line;
    getline(cin, line); istringstream iss(line); int num;
    while (iss >> num) vec.push_back(num);
    return vec;
}
```

Problem Solving Summary - June 30

4. Edit Distance

```
Problem: Convert word1 to word2 using insert/delete/replace (min operations).
Approach: Use 2D DP.
Code:
int minDistance(string w1, string w2) {
    int m = w1.size(), n = w2.size();
    vector<vector<int>> dp(m+1, vector<int>(n+1));
    for (int i = 0; i <= m; i++) dp[i][0] = i;
    for (int j = 0; j <= n; j++) dp[0][j] = j;
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j)
            dp[i][j] = (w1[i-1] == w2[j-1]) ? dp[i-1][j-1]
                       : 1 + min({dp[i-1][j], dp[i][j-1], dp[i-1][j-1]});
    return dp[m][n];
}
```

5. Simplify Unix Path

```
Problem: Canonical simplification of path with '.', '..', and '//'.
Approach: Use stack and split by '/'.
Code:
string simplifyPath(string path) {
    vector<string> stk; stringstream ss(path); string part;
    while (getline(ss, part, '/')) {
        if (part == "" || part == ".") continue;
        if (part == ".." && !stk.empty()) stk.pop_back();
        else if (part != "..") stk.push_back(part);
    }
    string res = "/";
    for (int i = 0; i < stk.size(); ++i) res += (i ? "/" : "") + stk[i];
    return res;
}
```

6. Letter Combinations of a Phone Number

```
Problem: Return all letter combinations for a given digit string.
Approach: Backtracking with digit->char mapping.
Code:
void backtrack(string& d, int i, string& cur, vector<string>& res, map<char, string>& phone) {
    if (i == d.size()) res.push_back(cur);
    else for (char c : phone[d[i]]) {
        cur.push_back(c);
        backtrack(d, i+1, cur, res, phone);
        cur.pop_back();
    }
}
vector<string> letterCombinations(string d) {
    if (d.empty()) return {};
    map<char, string> phone = {{'2',"abc"},{'3',"def"},{'4',"ghi"},{'5',"jkl"},
                               {'6',"mno"},{'7',"pqrs"},{'8',"tuv"},{'9',"wxyz"}};
    vector<string> res; string cur;
    backtrack(d, 0, cur, res, phone);
    return res;
}
```

7. Multiply Strings

Problem: Multiply num1 and num2 given as strings (no bigint/int).
Approach: Manual digit multiplication using a result vector.
Code:

```
string multiply(string n1, string n2) {
    int m = n1.size(), n = n2.size();
    if (n1 == "0" || n2 == "0") return "0";
    vector<int> res(m + n, 0);
    for (int i = m-1; i >= 0; --i)
        for (int j = n-1; j >= 0; --j) {
            int mul = (n1[i]-'0') * (n2[j]-'0');
            int sum = mul + res[i+j+1];
            res[i+j+1] = sum % 10;
            res[i+j] += sum / 10;
        }
    string prod = "";
    for (int num : res) if (!(prod.empty() && num == 0)) prod += to_string(num);
    return prod.empty() ? "0" : prod;
}
```