Bob and Chocolates

```java
public class Contest7Que3 {

   public static void main(String[] args) {
       int A = 10;
       int[] B = {4, 8, 5, 3};
       int[] C = {5, 12, 8, 1};
       System.out.println(solve(A, B, C));
   }

   public static int solve(int A, int[] B, int[] C) {
       int n = B.length;
       int[][] dp = new int[n + 1][A + 1];

       for (int i = 1; i <= n; i++) {
           for (int j = 1; j <= A; j++) {
               int inc = 0, exc = 0;
               // include
               if (j - B[i - 1] >= 0) {
                   inc = C[i - 1] + dp[i - 1][j - B[i - 1]];
               }
               // exclude
               exc = dp[i - 1][j];
               dp[i][j] = Math.max(inc, exc);
           }
       }
       return dp[n][A];
   }

   public static int solveBetter(int A, int[] B, int[] C) {
       int n = B.length;
       int[] prev = new int[A + 1];

       for (int i = 1; i <= n; i++) {
           int[] curr = new int[A + 1];
           for (int j = 1; j <= A; j++) {
               int inc = 0, exc = 0;
               // include
               if (j - B[i - 1] >= 0) {
                   inc = C[i - 1] + prev[j - B[i - 1]];
               }
               // exclude
               exc = prev[j];
               curr[j] = Math.max(inc, exc);
           }
           prev = curr;
       }
```

```java
        return prev[A];
    }
}
```

Alice and Pairs

```java
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

public class AliceAndPairs {


    public static int solve(int A, int[][] B) {
        List<Integer> graph[] = new List[A + 1];
        for (int i = 0; i <= A; i++) {
            graph[i] = new ArrayList<>();
        }
        buildGraph(A, B, graph);
        return bfs(A, graph);
    }

    private static int bfs(int A, List<Integer>[] graph) {
        Queue<Integer> q = new LinkedList<>();
        boolean[] visited = new boolean[A + 1];
        q.add(1);
        visited[1] = true;

        int ans = 0;
        while (!q.isEmpty()) {
            int lvl = q.size();
            while (lvl > 0) {
                int curr = q.poll();
                if (curr == A) {
                    return ans;
                }
                List<Integer> neighbours = graph[curr];
                for (int neigh : neighbours) {
                    if (!visited[neigh]) {
                        q.add(neigh);
                        visited[neigh] = true;
                    }
                }
                lvl--;
            }
            ans++;
        }
        return -1;
    }

    private static void buildGraph(int A, int[][] B, List<Integer>[] graph) {
        int n = B.length;
        for (int i = 0; i < n; i++) {
            int u = B[i][0];
            int v = B[i][1];
            graph[u].add(v);
            graph[v].add(u);
        }
    }

}
```