CSP-J中，哈希表算法考点

一、哈希表基础定义与核心性质（必考基础）

1. 核心概念

- 哈希表（散列表）定义：一种基于“键值对”存储的数据结构，通过哈希函数将键（key）映射到对应的存储位置（哈希地址），实现高效的插入、查询、删除操作；
- 关键术语：键（key，用于映射的标识，如学生学号）、值（value，存储的数据，如学生姓名）、哈希函数（映射规则）、哈希地址（映射后的存储位置）、冲突（不同键映射到同一哈希地址）；
- CSP-J适配场景：替代数组（解决键值非连续、范围过大的存储问题）、快速计数、去重、查找匹配元素（如两数之和、字符串匹配入门）。

2. 核心性质（CSP-J解题关键）

- 效率优势：理想情况下，插入、查询、删除的时间复杂度均为O(1)，远优于数组遍历（O(n)）、排序+二分（O(nlogn)）；
- 冲突不可避免：无论哈希函数设计如何，都可能出现不同键映射到同一地址的情况，需掌握基础冲突解决方法；
- 空间换时间：哈希表需额外分配空间存储数据及处理冲突，空间复杂度通常为O(n)，适合数据量适中（n≤1e5）的场景。

3. 考情说明

不直接考查哈希表原理默写，侧重实际应用（用C++自带容器实现哈希逻辑），偶考冲突解决的基础思路，是提升解题效率的核心工具，贯穿基础应用题、字符串题。

二、CSP-J适配的哈希表实现方式（重点掌握）

CSP-J中无需手动实现哈希表（手写难度高，不考），优先使用C++ STL容器，核心掌握两种常用容器的用法，覆盖所有考点。

1. unordered_map（首选，键值对映射）

- 核心特点：存储键值对（key唯一，一个键对应一个值），底层基于哈希表实现，支持快速查询键对应的 value；
- 适配场景：统计元素出现次数、键值映射（如将字符串映射为数字）、快速判断键是否存在；
- 注意事项：CSP-J中需包含<unordered_map>头文件，键需支持哈希（如int、string、char，不支持vector等复杂类型）。

2. unordered_set（次选，去重与存在性判断）

- 核心特点：仅存储键（key唯一，无对应value），底层基于哈希表实现，核心用于去重和快速判断元素是否存在；
- 适配场景：数组去重、判断元素是否出现过、筛选唯一元素；
- 注意事项：与unordered_map用法类似，需包含<unordered_set>头文件，键需支持哈希。

3. 补充说明（避坑重点）

- 不考内容：手动实现哈希表、复杂哈希函数设计、高级冲突解决算法（如红黑树优化）；
- 替代方案：若题目禁止使用unordered系列容器（极少数情况），可用map/set替代（底层是红黑树，时间复杂度O(logn)，CSP-J中可接受）。

三、哈希表核心算法与模板（CSP-J必考）

1. 元素计数与去重（高频基础）

核心考点：统计数组、字符串中元素出现次数，去除重复元素，是哈希表最基础的应用，适配多道J组真题。

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <unordered_set>
using namespace std;

// 1. 用unordered_map统计元素出现次数（适配数组、字符串）
void countElements(vector<int>& nums) {
    unordered_map<int, int> cnt; // key：元素，value：出现次数
    for (int x : nums) {
        cnt[x]++; // 插入元素，若已存在则次数+1，不存在则初始化为1
    }
    // 遍历输出结果
    for (auto& p : cnt) {
        cout << "元素 " << p.first << " 出现 " << p.second << " 次" << endl;
    }
}

// 2. 用unordered_set给数组去重
vector<int> removeDuplicates(vector<int>& nums) {
    unordered_set<int> st(nums.begin(), nums.end()); // 插入所有元素，自动去重
    vector<int> res(st.begin(), st.end()); // 转换为数组
    return res;
}
```

```
25
26  // 3. 字符串字符计数（真题高频场景）
27  void countChars(string& s) {
28      unordered_map<char, int> cnt;
29      for (char c : s) {
30          cnt[c]++;
31      }
32      for (auto& p : cnt) {
33          cout << "字符 '" << p.first << "' 出现 " << p.second << " 次" << endl;
34      }
35  }
36
37  int main() {
38      vector<int> nums = {1, 2, 2, 3, 3, 3, 4};
39      countElements(nums);
40
41      vector<int> unique_nums = removeDuplicates(nums);
42      cout << "去重后数组: ";
43      for (int x : unique_nums) cout << x << " ";
44      cout << endl;
45
46      string s = "abacbc";
47      countChars(s);
48      return 0;
49  }
50
```

2. 快速查找与匹配（真题高频）

核心考点：利用哈希表O(1)查询优势，解决“两数之和”“判断两个数组是否有交集”等问题，替代暴力遍历，提升解题效率。

```
1   #include <iostream>
2   #include <vector>
3   #include <unordered_map>
4   #include <unordered_set>
5   using namespace std;
6
7   // 示例1: 两数之和（CSP-J基础真题考点）
8   // 给定数组和目标值，找到两个元素下标，使它们的和等于目标值（假设唯一解）
9   vector<int> twoSum(vector<int>& nums, int target) {
10      unordered_map<int, int> mp; // key: 元素值，value: 元素下标
11      for (int i = 0; i < nums.size(); i++) {
12          int complement = target - nums[i];
13          // 查找互补元素是否存在，存在则返回下标
```

```
        if (mp.find(complement) != mp.end()) {
            return {mp[complement], i};
        }
        mp[nums[i]] = i; // 不存在则插入当前元素和下标
    }
    return {}; // 无解决方案（题目通常保证有解）
}

// 示例2：判断两个数组是否有交集
bool hasIntersection(vector<int>& nums1, vector<int>& nums2) {
    unordered_set<int> st(nums1.begin(), nums1.end());
    for (int x : nums2) {
        if (st.find(x) != st.end()) {
            return true; // 找到交集元素
        }
    }
    return false;
}

int main() {
    vector<int> nums = {2, 7, 11, 15}, target = 9;
    vector<int> res = twoSum(nums, target);
    cout << "两数之和下标：" << res[0] << " " << res[1] << endl;

    vector<int> nums1 = {1, 2, 3}, nums2 = {3, 4, 5};
    cout << "是否有交集：" << (hasIntersection(nums1, nums2) ? "是" : "否") <<
endl;
    return 0;
}

```

3. 冲突解决基础（仅考思路，不考手写）

CSP-J仅考查两种基础冲突解决思路，无需手动实现，理解即可应对选择题、判断题。

- 链地址法（拉链法）：最常用，将同一哈希地址的元素存储在一个链表中，查询时遍历对应链表即可，unordered_map/unordered_set底层采用此思路；
- 开放地址法：当发生冲突时，寻找下一个空闲的哈希地址存储元素（如线性探测、二次探测），理解核心逻辑即可，不考代码实现。

四、CSP-J哈希表避坑与备考要点

1. 易错点

- 头文件遗漏：使用unordered_map/unordered_set必须包含对应头文件，否则会编译错误；

- 键的选择：避免使用无法哈希的类型作为键（如vector、数组），优先用int、char、string；
- 效率误区：当数据量极小时（n≤100），暴力遍历与哈希表效率差距不大，可灵活选择；数据量较大时（n≥1e3），优先用哈希表。

2. 备考优先级

- 必掌握：unordered_map/unordered_set的基本用法（插入、查询、遍历）、元素计数、去重、两数之和类匹配问题；
- 选掌握：冲突解决的两种基础思路（链地址法、开放地址法）；
- 不考内容：手写哈希表、复杂哈希函数设计、高级冲突解决算法、哈希表扩容机制。

3. 刷题建议

聚焦基础应用题，重点刷“元素计数”“去重”“两数之和”“交集判断”类真题，每类题型刷2-3道，熟练掌握unordered_map和unordered_set的用法即可；无需刷偏题、难题，考点固定且简单。

（注：文档部分内容可能由 AI 生成）