



Leonardo 的 ACM 模板

模板用的好，金牌少不了

作者：列奥那多是勇者

组织：狗头吧

时间：August 5, 2022

版本：0.0.1

邮箱：azraelzarkli@gmail.com



狗头吧在什么时候都是 acmer 最好的救济

写在前面

模板使用须知

本模板使用了 elegantbook-magic-revision latex 模板 (<https://github.com/Azure1210/elegantbook-magic-revision>)，在此对其表达感谢。

本文档的题显版本相关题目中包含 AcWing 题目，部分是需要买课才能查看的，这类题目可以到其它算法网站(如洛谷)上查找。



目录

1

第 1 部分 * STL

第 1 章	STL	2
1.1	pair	2
1.2	vector	2

2

第 2 部分 * 数据结构

第 2 章	数据结构	5
2.1	union find	5

3

第 3 部分 * 基础算法

第 3 章	排序算法	8
3.1	快速排序	8
3.2	归并排序	8
第 4 章	双指针算法	10

4

第 4 部分 * 进阶算法

第 5 章	图论	12
5.1	二分图	12
5.1.1	检测二分图	12

5

第 5 部分 * 附录

第 6 章	附录	15
6.1	比赛初始代码模板	15

第一部分

STL

第1部分目录

第1章 STL

1.1 pair

1.2 vector

2

2

2



第 1 章 STL

❖ 1.1 pair

```
1 pair<int, int> a{1,2};  
2 pair<int, int> b{3,1};  
3 cout << (a < b) << endl;  
4 // vector<pair<int, int>> c{a, b};  
5 vector<pair<int, int>> c{{1,2}, {1,1}, {0,3}};  
6 // sort c according to the first param ascendingly (升序);  
7 sort(c.begin(), c.end());  
8 // sort c according to the second param ascendingly (升序); 返回值不为true交换  
9 sort(c.begin(), c.end(), [](const pair<int,int> &a, const pair<int, int> &b){return a.second < b.second;});  
10 // multiple condition  
11 sort(c.begin(), c.end(), [](const pair<int,int> &a, const pair<int, int> &b){return a.first < b.first ||  
    a.second < b.second;});
```

❖ 1.2 vector

```
1 #include <vector>  
2 vector<pair<int,int>> a;  
3 vector<int> b(3, 2); // 默认0, 手动2  
4 vector<int> d(b); // 用b创建d  
5 d.resize(20); // 重新分配空间, 若空间小于原本, 则删除  
6 vector<vector<int>> c{vector<int>{1,2}};  
7 vector<vector<int>> e(10,vector<int>(10));  
8 c.resize(2, vector<int>(2)); // 分配二维空间  
9 c.empty(); c.size(); c.clear();  
10 a.emplace_back(1, 2);
```

```
11 a.pop_back();  
12 a.begin(); a.end(); a.rbegin(); a.rend();
```



第二部分 数据结构

第 2 部分目录

第 2 章 数据结构

5

2.1 union find

5



第 2 章 数据结构

◆ 2.1 union find

并查集是一种树形的数据结构，顾名思义，它用于处理一些不交集的合并及查询问题。它支持两种操作：

- **查找 (Find):** 确定某个元素处于哪个子集；
- **合并 (Union):** 将两个子集合并成一个集合。

```
1 const int N = 100; // to be changed
2
3 class union_find
4 {
5     private:
6         int parent[N];
7         int size;
8
9     public:
10        int count;
11
12    union_find(int n)
13    {
14        size = n;
15        count = n; // not connected at the beginning
16        for (int i = 0; i < n; i++) parent[i] = i;
17    }
18
19    // int count() // the number of connected parts
20    // return this.count;
21
22    int find(int x)
23    { // basic func for implementing other functions
24        assert(x < size);
```

```
25     if (parent[x] != x) parent[x] = find(parent[x]);
26     return parent[x];
27 }
28
29 void union_parts(int p, int q)
30 { /* connect two parts
31     assert(p < size || q < size);
32     int rootP = find(p);
33     int rootQ = find(q);
34     if (rootP == rootQ) return;
35     parent[rootQ] = rootP;
36     count--;
37 }
38
39 bool connected(int p, int q)
40 { /* whether p and q are in the same part
41     assert(p < size || q < size);
42     int rootP = find(p);
43     int rootQ = find(q);
44     return rootP == rootQ;
45 }
46 };
```

相关题目：

1. LeetCode 990. Satisfiability of Equality Equations: <https://leetcode.cn/problems/satisfiability-of-equality-equations/>



第三部分

基础算法



第3部分目录

第3章 排序算法	8
3.1 快速排序	8
3.2 归并排序	8
第4章 双指针算法	10



第3章 排序算法

◆ 3.1 快速排序

```
1 void quick_sort(int q[], int l, int r)
2 {
3     if (l >= r) return;
4
5     int i = l - 1, j = r + 1, x = q[l + r >> 1];
6     // x取l和r的中间值避免了某些情况下x取l或者r导致的无限循环问题
7     while (i < j)
8     {
9         do i ++ ; while (q[i] < x);
10        do j -- ; while (q[j] > x);
11        if (i < j) swap(q[i], q[j]);
12    }
13    quick_sort(q, l, j), quick_sort(q, j + 1, r);
14 }
```

◆ 3.2 归并排序

```
1 // 需要自己定义tmp数组(n)
2 void merge_sort(int q[], int l, int r)
3 {
4     if (l >= r) return;
5
6     int mid = l + r >> 1;
7     merge_sort(q, l, mid);
8     merge_sort(q, mid + 1, r);
```

```
9  
10 int k = 0, i = l, j = mid + 1; // j 只能取 mid+1  
11 while (i <= mid && j <= r)  
12     if (q[i] <= q[j]) tmp[k ++] = q[i ++];  
13     else tmp[k ++] = q[j ++];  
14  
15 while (i <= mid) tmp[k ++] = q[i ++];  
16 while (j <= r) tmp[k ++] = q[j ++];  
17  
18 for (i = l, j = 0; i <= r; i ++, j ++ ) q[i] = tmp[j];  
19 }
```

相关题目：

1. AcWing 788. 逆序对的数量: <https://www.acwing.com/problem/content/790/>



第 4 章 双指针算法

第四部分 进阶算法

第 4 部分目录

第 5 章 图论	12
5.1 二分图	12



第 5 章 图论

◆ 5.1 二分图

二分图常用来判断是否能分成两组

5.1.1 检测二分图

```
1 // 自己写的，经过leetcode检验
2 int mark[N]{0}; // 记录分组
3
4 bool traverse(vector<vector<int>> &graph, int root, int flag = 1, int prev = -1)
5 { // graph: 无向图(双向图)的邻接矩阵; flag: 判断是否是哪组; prev 防止两者来回访问
6     if (!mark[root]) mark[root] = flag;
7     else if (mark[root] != flag) return false;
8     else return true;
9
10    for (auto &node: graph[root])
11        if (node != prev && !traverse(graph, node, -flag, root)) return false;
12
13    return true;
14 }
15
16 bool isBipartite(vector<vector<int>>& graph)
17 { // 主函数
18     int n = graph.size();
19     for (int i = 0; i < n; i++)
20         if (!mark[i] && !traverse(graph, i)) return false;
21     return true;
22 }
```

相关题目：

1. LeetCode 886. Possible Bipartition: <https://leetcode.cn/problems/possible-bipartition/>

第五部分

附录

第 5 部分目录

第 6 章 附录	15
6.1 比赛初始代码模板	15



第 6 章 附录

◆ 6.1 比赛初始代码模板

```
1 #pragma GCC optimize(3) // 03
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 #define rep(i, a, b) for(int i=a; i < (b); i++)
6 #define trav(a,x) for (auto& a: x)
7 #define all(x) begin(x), end(x)
8 #define rall(x) rbegin(x), rend(x)
9 #define sz(x) (int)(x).size()
10 #define endl "\n" // promote speed
11 typedef long long ll;
12 typedef long double ld;
13 typedef pair<int, int> pii;
14 typedef pair<ll, ll> pll;
15 typedef vector<int> vi;
16
17
18 void solve(int tcase){
19
20 }
21
22 int main(int argc, char *argv[]){
23     ios::sync_with_stdio(false);cin.tie(0);
24
25     int T;
26     cin >> T;
27     rep(t, 1 , T + 1){
```

```
28     solve(t);  
29 }  
30 }
```