

DAMON THRONE · 岱陌存理模拟赛系列

提高组

题目名称	半选	端序	括价	组树
题目类型	传统型	传统型	传统型	传统型
目录	half	ends	brac	comp
可执行文件名	half	ends	brac	comp
输入文件名	half.in	ends.in	brac.in	comp.in
输出文件名	half.out	ends.out	brac.out	comp.out
每个测试点时限	1.0 秒	1.0 秒	1.0 秒	1.0 秒
内存限制	512 MiB	512 MiB	512 MiB	512 MiB
测试点数目	20	20	20	20
测试点是否等分	是	是	是	是

提交源程序文件名

对于 C++ 语言	half.cpp	ends.cpp	brac.cpp	comp.cpp
-----------	----------	----------	----------	----------

编译选项

对于 C++ 语言	-O2 -std=c++14 -static
-----------	------------------------

【考生须知】

1. 文件名（程序名和输入输出文件名）必须使用英文小写。
2. main 函数的返回值类型必须是 int，程序正常结束时的返回值必须是 0。
3. 提交的程序代码文件的放置位置请务必参照各自的具体要求。
4. 因违反上述要求而出现的错误或问题，申诉不予受理。
5. 若无特殊说明，结果的比较方式为全量比较（中间过程行不能为空格和文字符号）。
6. 选择提交的程序源文件必须不大于 100KB。
7. 程序内使用的变量命名与题目中的命名要求一致。
8. 全国统一使用的机器配置为：Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz，内存 32GB。以上配置以此配置为准。
9. 只提供 Linux 格式附带样例文件。
10. 请确保在当前最新公布的 NOI Linux 下进行，各语言的编译器版本应为标准版本。

半选 (half)

【题目描述】

Y 同学面前有 n 个任务，第 i 个任务需要 a_i 分钟。他可以选择任意若干个任务完成，但总用时不能超过 T 。

请输出他能达到的最大总用时。

【输入格式】

从文件 half.in 中读入数据。

第一行包含两个整数 n, T 。

第二行包含 n 个整数 $a_1, a_2, \dots, a_n$ 。

【输出格式】

输出到文件 half.out 中。

输出一行一个整数，表示不超过 T 的最大总用时。

【样例 1 输入】

```
1 6 31
2 5 8 12 7 20 4
```

【样例 1 输出】

```
1 31
```

【数据范围与约定】

对于 100% 的数据，保证 $1 \leq n \leq 40$ ， $1 \leq T, a_i \leq 10^9$ 。

测试点编号	分值	$n \leq$	特殊性质
1 ~ 2	10	20	无
3 ~ 5	15	30	所有 a_i 很小
6 ~ 8	15	34	很多 $a_i > T$
9 ~ 12	20	36	a_i 接近 T/n
13 ~ 16	20	38	无
17 ~ 20	20	40	无

【题解】

solve

题意概述

从 n 个数中任选若干个，使总和不超过 T 且尽量大。

样例分析

样例中需要比较所有不超过 31 的任务组合，最优总和由程序计算得到。

部分分做法

当 $n \leq 20$ 时，可以枚举所有子集。

满分做法

把任务分成左右两半。分别枚举每一半的所有子集和，并丢弃超过 T 的和。将右半部分的子集和排序。

枚举左半部分和 x ，在右半部分中二分找到不超过 $T - x$ 的最大值，更新答案。

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  vector<ll> sums(vector<ll> a, ll t) {
6      vector<ll> r(1, 0);
7      for (ll v : a) {
8          int n = r.size();
9          for (int i = 0; i < n; i++) {
10             if (r[i] + v <= t) r.push_back(r[i] + v);
11         }
12     }
13     return r;
14 }
15
16 int main() {
17     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
18     int n;
19     ll t;
20     cin >> n >> t;
21     vector<ll> a(n), l, r;

```

```

22     for (int i = 0; i < n; i++) cin >> a[i];
23     for (int i = 0; i < n; i++) {
24         if (i < n / 2) l.push_back(a[i]);
25         else r.push_back(a[i]);
26     }
27     auto x = sums(l, t), y = sums(r, t);
28     sort(y.begin(), y.end());
29     ll ans = 0;
30     for (ll v : x) {
31         auto it = upper_bound(y.begin(), y.end(), t - v);
32         if (it != y.begin()) {
33             --it;
34             ans = max(ans, v + *it);
35         }
36     }
37     cout << ans << '\n';
38     return 0;
39 }

```

复杂度

时间复杂度 $O(2^{n/2} \log 2^{n/2})$ ，空间复杂度 $O(2^{n/2})$ 。

端序 (ends)

【题目描述】

Y 同学写下了一个长度为 n 的序列 $a_1, a_2, \dots, a_n$ 。

他想从中选出一个长度至少为 2 的子序列。若选出的第一个数不大于最后一个数，这个子序列就被称为稳定的。

请计算稳定子序列的数量，对 998244353 取模。

【输入格式】

从文件 ends.in 中读入数据。

第一行包含一个整数 n 。

第二行包含 n 个整数 $a_1, a_2, \dots, a_n$ 。

【输出格式】

输出到文件 ends.out 中。

输出一行一个整数，表示答案。

【样例 1 输入】

```
1 5
2 3 1 4 1 5
```

【样例 1 输出】

```
1 20
```

【数据范围与约定】

对于 100% 的数据，保证 $2 \leq n \leq 2 \times 10^5$ ， $1 \leq a_i \leq 10^9$ 。

测试点编号	分值	$n \leq$	特殊性质
1 ~ 2	10	20	无
3 ~ 5	15	2000	a_i 两两不同
6 ~ 8	15	5000	a_i 只取很少种值
9 ~ 12	20	30000	序列单调
13 ~ 16	20	100000	无
17 ~ 20	20	2×10^5	无

【题解】

solve

题意概述

统计长度至少为 2，并且第一个被选元素不大于最后一个被选元素的子序列数量。

样例分析

样例中可以枚举每一对作为子序列的首尾。如果首尾位置为 $i < j$ 且 $a_i \leq a_j$ ，中间的元素可以任意选或不选。

部分分做法

小数据可以枚举所有子序列。

稍大的数据可以枚举首尾位置 (i, j) ，若 $a_i \leq a_j$ ，贡献 2^{j-i-1} 。

满分做法

每个合法子序列由它的首尾位置唯一确定。对固定的右端点 j ，所有合法左端点 i 的贡献为

$$2^{j-i-1} = 2^{j-1} \cdot 2^{-i}.$$

从左到右扫描，用树状数组按 a_i 的大小维护 2^{-i} 的和。处理位置 j 时查询所有 $a_i \leq a_j$ 的前缀和，再乘以 2^{j-1} 即可。

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  const ll mod = 998244353;
5
6  struct BIT {
7      int n;
8      vector<ll> t;
9      BIT(int n = 0): n(n), t(n + 1) {}
10     void add(int x, ll v) {
11         for (; x <= n; x += x & -x) t[x] = (t[x] + v) % mod;
12     }
13     ll sum(int x) {
14         ll r = 0;
15         for (; x; x -= x & -x) r = (r + t[x]) % mod;
16         return r;
17     }

```

```

18 };
19
20 int main() {
21     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
22     int n;
23     cin >> n;
24     vector<ll> a(n + 1), xs;
25     for (int i = 1; i <= n; i++) {
26         cin >> a[i];
27         xs.push_back(a[i]);
28     }
29     sort(xs.begin(), xs.end());
30     xs.erase(unique(xs.begin(), xs.end()), xs.end());
31     vector<ll> p2(n + 1, 1), iv(n + 1, 1);
32     ll inv2 = (mod + 1) / 2;
33     for (int i = 1; i <= n; i++) {
34         p2[i] = p2[i - 1] * 2 % mod;
35         iv[i] = iv[i - 1] * inv2 % mod;
36     }
37     BIT bit(xs.size());
38     ll ans = 0;
39     for (int i = 1; i <= n; i++) {
40         int r = lower_bound(xs.begin(), xs.end(), a[i]) -
xs.begin() + 1;
41         ans = (ans + p2[i - 1] * bit.sum(r)) % mod;
42         bit.add(r, iv[i]);
43     }
44     cout << ans << '\n';
45     return 0;
46 }

```

复杂度

时间复杂度 $O(n \log n)$ ，空间复杂度 $O(n)$ 。

括价 (brac)

【题目描述】

Y 同学有一个长度为 n 的字符串 s ，其中每个字符是 (、) 或 ?。

他要把每个 ? 替换成 (或)。对于第 i 个 ?：

- 若替换成 (，需要花费 a_i ；
- 若替换成)，需要花费 b_i 。

请你求出使最终字符串成为合法括号序列的最小花费。若无法做到，输出 -1 。

【输入格式】

从文件 `brac.in` 中读入数据。

第一行包含一个整数 n 。

第二行包含字符串 s 。

接下来若干行，按 ? 在字符串中从左到右出现的顺序，每行包含两个整数 a_i, b_i 。

【输出格式】

输出到文件 `brac.out` 中。

输出一行一个整数，表示最小花费；若无解，输出 -1 。

【样例 1 输入】

```
1 2
2 ??
3 3 5
4 4 1
```

【样例 1 输出】

```
1 4
```

【数据范围与约定】

对于 100% 的数据，保证 $1 \leq n \leq 2 \times 10^5$ ， $1 \leq a_i, b_i \leq 10^9$ 。

测试点编号	分值	$n \leq$	$a_i, b_i \leq$	特殊性质
1 ~ 2	10	20	100	无
3 ~ 5	15	2×10^5	10^9	特殊性质 A
6 ~ 8	15	2×10^5	10^9	特殊性质 B
9 ~ 12	20	5×10^4	10^9	无
13 ~ 16	20	2×10^5	10^9	特殊性质 C
17 ~ 20	20	2×10^5	10^9	无

- 特殊性质 A: 保证 s 中只有 ?。
- 特殊性质 B: 保证 s 的第一个字符是)。
- 特殊性质 C: 保证 s 中没有 ?。

【题解】

solve

题意概括

字符串中有若干个 `?`，每个位置替换成左右括号的代价不同。要求把字符串变成合法括号序列，并使总代价最小。数据范围为 $n \leq 2 \times 10^5$ 。

保留样例：

```
1 2
2 ??
3 3 5
4 4 1
```

对应输出：

```
1 4
```

样例分析

把第一个 `?` 替换成 `(`，花费 3；把第二个 `?` 替换成 `)`，花费 1。得到字符串 `()`，总花费为 4。

题目分析

- **算法 1**：小数据可以枚举所有 `?` 的替换方式，检查括号序列是否合法，并取最小代价。若有 k 个 `?`，复杂度为 $O(2^k n)$ ，期望得分：10 分。
- **算法 2**：对特殊性质 A，可以先确定必须有多少个位置变成左括号，再用前缀条件筛掉不合法选择。对特殊性质 B，首字符固定为右括号时必定无解。对特殊性质 C，直接检查原串是否合法。期望得分：30 分。
- **算法 3**：先把所有 `?` 都暂时当作 `)`，并支付对应代价 b_i 。从左到右扫描时，维护当前括号余额。如果余额变成负数，说明当前前缀右括号过多，必须把之前某一个 `?` 改成 `(`。

把第 i 个 `?` 从 `)` 改成 `(` 的额外代价是 $a_i - b_i$ 。为了让总代价最小，每次需要补救前缀时，都选择目前出现过且尚未被修改的 `?` 中额外代价最小的那个，用小根堆维护即可。扫描结束后若余额为 0，得到最小代价；否则无解。时间复杂度为 $O(n \log n)$ ，期望得分：100 分。

参考实现

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

```

3  using ll = long long;
4
5  int main() {
6      ios::sync_with_stdio(0);
7      cin.tie(0);
8      cout.tie(0);
9
10     int n;
11     string s;
12     cin >> n >> s;
13     int k = 0;
14     for (char c : s) k += c == '?';
15     vector<ll> a(k), b(k);
16     for (int i = 0; i < k; i++) cin >> a[i] >> b[i];
17
18     if (n & 1) {
19         cout << -1 << '\n';
20         return 0;
21     }
22
23     priority_queue<ll, vector<ll>, greater<ll>> q;
24     ll ans = 0;
25     int bal = 0, id = 0;
26     for (char c : s) {
27         if (c == '(') {
28             bal++;
29         } else if (c == ')') {
30             bal--;
31         } else {
32             ans += b[id];
33             q.push(a[id] - b[id]);
34             id++;
35             bal--;
36         }
37         if (bal < 0) {
38             if (q.empty()) {
39                 cout << -1 << '\n';

```

```

40         return 0;
41     }
42     ans += q.top();
43     q.pop();
44     bal += 2;
45 }
46 }
47
48 cout << (bal == 0 ? ans : -1) << '\n';
49 return 0;
50 }
```

复杂度分析

最终正解的时间复杂度为 $O(n \log n)$ 。

空间复杂度为 $O(n)$ 。

组树 (comp)

【题目描述】

Y 同学有一棵 n 个点的树。他会选择若干个点，至少选择一个，并只保留这些点以及两端都被选择的边。

对每个 $k = 1, 2, \dots, n$ ，请计算有多少种选择点的方案，使保留下来的图恰好有 k 个连通块。答案对 998244353 取模。

【输入格式】

从文件 comp.in 中读入数据。

第一行包含一个整数 n 。

接下来 $n - 1$ 行，每行包含两个整数 u, v ，表示树上的一条边。

【输出格式】

输出到文件 comp.out 中。

输出 n 行，第 k 行表示恰好有 k 个连通块的方案数。

【样例 1 输入】

```
1 5
2 1 2
3 1 3
4 3 4
5 3 5
```

【样例 1 输出】

```
1 17
2 11
3 3
4 0
5 0
```

【数据范围与约定】

对于 100% 的数据，保证 $1 \leq n \leq 2000$ 。

测试点编号	分值	$n \leq$	特殊性质
1 ~ 2	10	12	无
3 ~ 5	15	200	树是一条链
6 ~ 8	15	400	树是菊花图
9 ~ 12	20	800	树是完全二叉树的一部分
13 ~ 16	20	1200	无
17 ~ 20	20	2000	无

【题解】

solve

题意概述

对树的所有非空点集，统计诱导子图中连通块数量，并按连通块个数输出方案数。

样例分析

样例中选择一个点会形成一个连通块；选择若干互不相邻的点会形成多个连通块；选择相邻点时连通块会合并。

部分分做法

小数据可以枚举所有点集并 BFS 统计连通块。

若树是一条链，问题退化为统计二进制串中连续 1 段的数量，可以用线性 DP。

满分做法

根树。对每个节点 u 维护两个多项式：

- $dp0_u[k]$ ：在 u 的子树中选择点，且不选择 u ，形成 k 个连通块的方案数；
- $dp1_u[k]$ ：在 u 的子树中选择点，且选择 u ，形成 k 个连通块的方案数。

合并儿子 v 时，若 u 不选，儿子子树的两种状态都只是把连通块数相加。若 u 选而 v 也选，则边 (u, v) 会把包含 u 与包含 v 的两个连通块合并，所以总连通块数减少 1。

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  const ll mod = 998244353;
5
6  int main() {
7      ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
8      int n;
9      cin >> n;
10     vector<vector<int> > g(n + 1);
11     for (int i = 1; i < n; i++) {
12         int u, v;
13         cin >> u >> v;
14         g[u].push_back(v);
15         g[v].push_back(u);
16     }
17     vector<int> par(n + 1), ord;

```

```

18     ord.push_back(1);
19     for (int qi = 0; qi < (int)ord.size(); qi++) {
20         int u = ord[qi];
21         for (int v : g[u]) {
22             if (v == par[u]) continue;
23             par[v] = u;
24             ord.push_back(v);
25         }
26     }
27     vector<vector<ll> > dp0(n + 1), dp1(n + 1);
28     vector<int> sz(n + 1, 1);
29     for (int oi = n - 1; oi >= 0; oi--) {
30         int u = ord[oi];
31         vector<ll> a0(1, 1), a1(2);
32         a1[1] = 1;
33         int su = 1;
34         for (int v : g[u]) {
35             if (par[v] != u) continue;
36             int sv = sz[v];
37             vector<ll> n0(su + sv + 1), n1(su + sv + 1), bt(sv +
1);
38             for (int j = 0; j <= sv; j++) {
39                 ll x = j < (int)dp0[v].size() ? dp0[v][j] : 0;
40                 ll y = j < (int)dp1[v].size() ? dp1[v][j] : 0;
41                 bt[j] = (x + y) % mod;
42             }
43             for (int i = 0; i < (int)a0.size(); i++) if (a0[i]) {
44                 for (int j = 0; j < (int)bt.size(); j++) if
(bt[j]) {
45                     n0[i + j] = (n0[i + j] + a0[i] * bt[j]) % mod;
46                 }
47             }
48             for (int i = 0; i < (int)a1.size(); i++) if (a1[i]) {
49                 for (int j = 0; j < (int)dp0[v].size(); j++) if
(dp0[v][j]) {
50                     n1[i + j] = (n1[i + j] + a1[i] * dp0[v][j]) %
mod;

```

```

51         }
52         for (int j = 1; j < (int)dp1[v].size(); j++) if
(dp1[v][j]) {
53             n1[i + j - 1] = (n1[i + j - 1] + a1[i] *
dp1[v][j]) % mod;
54         }
55     }
56     su += sv;
57     a0.swap(n0);
58     a1.swap(n1);
59 }
60 sz[u] = su;
61 dp0[u] = a0;
62 dp1[u] = a1;
63 }
64 for (int k = 1; k <= n; k++) {
65     ll ans = 0;
66     if (k < (int)dp0[1].size()) ans += dp0[1][k];
67     if (k < (int)dp1[1].size()) ans += dp1[1][k];
68     cout << ans % mod << '\n';
69 }
70 return 0;
71 }

```

复杂度

时间复杂度 $O(n^2)$ ，空间复杂度 $O(n^2)$ 。