Contents

1 Tips

1. pow 返回浮点数可能会导致误差，慎用！！！自己写。
2. 样例数用 long long 可能会 TLE
3. 多组数据记得初始化
4. 小心过程量 overflow
5. Type convertions
6. 在取余的情况下，要避免减法运算的结果出现负数 (...+ MOD) % MOD;
7. fill 注意初始范围为从 1 到 n 的情况
8. 提交换行
9. c++11: long long int abs (long long int n);
10. 输入读取判断其中一个为零
11. 搜索 flag 标记可能要加上方案数
12. 数据范围特别大时考虑不可能情况
13. SegmentTree 开大点以防 RE （1e5 –> 1 << 17)
14. 滚动数组未全部清零

2 Basic Algorithm

2.1 Sort

```
// Bubble Sort
for (int i = 0; i < N; i++)
    for (int j = 0; j < N - i - 1; j++)
        if (A[j] > A[j + 1]) swap(A[j], A[j + 1]);
```

```
// Insertion Sort
for (int i = 1; i < N; i++) {
    int tmp = A[i], j;
    for (j = i - 1; j >= 0 && A[j] > tmp; j--)
        A[j + 1] = A[j];
    A[++j] = tmp;
}
```

```
// Selection Sort
for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
        if (A[i] > A[j]) swap(A[i], A[j])
```

```
// Merge Sort
void merge_sort(int A[], int l, int r) {
    if (l >= r) return ;
    int mid = (l + r) / 2;
    merge_sort(A, l, mid);
    merge_sort(A, mid + 1, r);
    int i, j, k;
    i = l; j = mid + 1; k = l;
    while (i <= mid && j <= r) {
        if (A[i] <= A[j]) B[k] = A[i++];
        else B[k] = A[j++];
        k++;
    }
    while (i <= mid) {
        B[k] = A[i++];
        k++;
    }
    while (j <= r) {
        B[k] = A[j++];
        k++;
```

```
21       }
22       memcpy(A, B, r - l + 1);
23       return ;
24   }
```

```
1   // Quick Sort
2   void quicksort(int A[], int l, int r) {
3       int i = l, j = r, mid = A[(r - l) / 2 + l];
4       while (i <= j) {
5           while (A[i] < mid) i++;
6           while (A[j] > mid) j--;
7           if (i <= j) {
8               swap(A[i], A[j]);
9               ++i; --j;
10          }
11      }
12      if (i < r) quicksort(A, i, r);
13      if (l < j) quicksort(A, l, j);
14      return ;
15  }
```

- Heap Sort（见堆的内容）

2.2 DP

2.2.1 LIS

```
1   int A[MAX_N];
2   long lis(int n) {
3       int dp[MAX_N];
4       fill(dp, dp + n, INF);
5       for (int i = 0; i < n; ++i)
6           *lower_bound(dp, dp + n, A[i]) = A[i];// lds: -A[i]; ln:
                upper_bound
7       return lower_bound(dp, dp + n, INF) - dp;
8   }
```

2.2.2 Knapsack Problem

- 0/1 背包

$$f[i,j] = max(f[i-1,j], f[i-1,j-w[i]]+v[i])$$

```
1   for (int i = 0; i < N; ++i)
2       for (int j = W; j >= w[i]; --j)
3           f[j] = max(f[j - w[i]] + c[i], f[j]);
```

- 完全背包

$$f[i,j] = max(f[i-1,j], f[i-1,j-w[i]]+j[i])$$

```
1   for (int i = 0; i < N; ++i)
2       for (int j = w[i]; j <= W; ++j)
3           f[j] = max(f[j - w[i]] + c[i], f[v]);
```

注意循环顺序的不同背后思路。

- 一个简单的优化：若两件物品 i、j 满足 $w[i] \leq w[j]$ 且 $c[i] \geq c[j]$，则讲物品 j 去掉，不用考虑。
- 转化为 01 背包问题求解：
  - 第 i 种物品转化为 $\frac{V}{w[i]}$ 件费用于价值均不变的物品。
  - 第 i 种物品拆成费用为 $w[i] * 2^k$，价值为 $c[i] * 2^k$ 的若干件物品其中 k 满足 $w[i] * 2^k < V$
- 多重背包

$$f[i,j] = max(f[i-1,j-w[i]*k]+v[i]*k | 0 <= k <= m[i])$$

- 优化：转化为 01 背包问题
  - 将第 i 件物品分成若干件物品，每件物品的系数分别为：$1, 2, 4, ..., 2^{(k-1)}, n[i] - 2^k$
  - ** 根据 w, v 范围改变 DP 对象，可以考虑针对不同价值计算最小的重量。（$f[i][j]$，其中 j 代表价值总和）**

```
1   for (int i = 0; i < N; ++i) {
2       int num = m[i];
3       for (int k = 1; num > 0; k <<= 1) {
4           int mul = min(k, num);
5           for (int j = W; j >= w[i] * mul; --j) {
6               f[j] = max(f[j - w[i] * mul] + v[i] * mul, f[j]);
7           }
8           num -= mul;
9       }
```

```
10 }
```

- 混合三种背包

弄清楚上面三种背包后分情况就好

- 超大背包

    - $1 \leq n \leq 40, \ 1 \leq w_i, v_i \leq 10^{15}$, $1 \leq W \leq 10^{15}$

```
1  int n;
2  ll w[MAX_N], v[MAX_N], W;
3  Pll ps[1 << (MAX_N / 2)]; // (w, v);
4  void solve() {
5      int n2 = n / 2;
6      for (int i = 0; i < 1 << n2; ++i) {
7          ll sw = 0, sv = 0;
8          for (int j = 0; j < n2; ++j)
9              if (i >> j & 1) {
10                 sw += w[j];
11                 sv += v[j];
12             }
13         ps[i] = Pll(sw, sv);
14     }
15     sort(ps, ps + (1 << n2));
16     int m = 1;
17     for (int i = 1; i < 1 << n2; ++i)
18         if (ps[m - 1].second < ps[i].second)
19             ps[m++] = ps[i];
20     ll res = 0;
21     for (int i = 0; i < 1 << (n - n2); ++i) {
22         ll sw = 0, sv = 0;
23         for (int j = 0; j < n - n2; ++j)
24             if (i >> j & 1) {
25                 sw += w[n2 + j];
26                 sv += v[n2 + j];
27             }
28         if (sw <= W) {
29             ll tv = (lower_bound(ps, ps + m, make_pair(W - sw, INF)) -
                    1)->second;
30             res = max(res, sv + tv);
31         }
32     }
33     printf("%lld\n", res);
34 }
```

- 二维费用背包

$$f[i,j,k] = max(f[i-1,j,k], f[i-1,j-a[i],k-b[i]]+c[i])$$

二维费用可由最多取 m 件等方式隐蔽给出。

- 分组背包

$$f[k,j] = max(f[k-1,j], f[k-1,j-w[i]]+v[i]|i \in K)$$

```cpp
for (int k = 0; k < K; ++k)
    for (j = W; j >= 0; --j)
        for (int i = 0; i <= m[k]; ++i)
            f[j] = max(f[j - w[i]] + v[i], f[j]);
```

显然可以对每组中物品应用完全背包中"一个简单有效的优化"

- 有依赖背包

由 NOIP2006 金明的预算方案引申，对每个附件先做一个 01 背包，再与组件得到一个 $V-w[i]+1$ 个物品组。更一般问题，依赖关系由「森林」形式给出，涉及到树形 DP 以及泛化物品，这里不表。

- 背包问题方案总数

$$f[i,j] = sum(f[i-1,j], f[i-1,j-w[i]]+v[i]), f[0,0] = 0$$

更多内容详见「背包九讲」

2.2.3 Maximum Subarray Sum

```cpp
int max_subarray_sum(int A[], int n) {
    int res, cur;
    if (!A || n <= 0) return 0;
    res = cur = a[0];
    for (int i = 0; i < n; ++i) {
        if (cur < 0) cur = a[i];
        else cur += a[i];
        res = max(cur, res);
    }
```

```
10        return res;
11 }
```

2.3 Set

```
1  // 子集枚举
2  int sub = sup;
3  do {
4      sub = (sub - 1) & sup;
5  } while (sub != sup); // -1 & sup = sup;
6  // 势为 k 的集合枚举
7  int comb = (1 << k) - 1;
8  while (comb < 1 << n) {
9      int x = comb & -comb, y = comb + x;
10     comb = ((comb & ~y) / x >> 1) | y;
11 }
12 // 排列组合
13 do {
14
15 } while (next_permutation(A, A + N)); // prev_permutation
```

2.4 Bit operation

```
1  int __builtin_ffs (unsigned int x)
2  //返回x的最后一位1的是从后向前第几位，比如7368 (1110011001000) 返回4。
3  int __builtin_clz (unsigned int x)
4  // 返回前导的0的个数。
5  int __builtin_ctz (unsigned int x)
6  // 返回后面的0个个数，和__builtiin_clz相对。
7  int __builtin_popcount (unsigned int x)
8  // 返回二进制表示中1的个数。
9  int __builtin_parity (unsigned int x)
10 // 返回x的奇偶校验位，也就是x的1的个数模2的结果。
```

3 Data Structure

```
1   // Heap
2   int heap[MAX_N], sz = 0;
3   void push(int x) {
4       int i = sz++;
5
6       while (i > 0) {
7           int p = (i - 1) / 2;
8           if (heap[p] <= x) break;
9           heap[p] = heap[i];
10          i = p;
11      }
12      heap[i] = x;
13  }
14  int pop() {
15      int ret = heap[0];
16      int x = heap[--sz];
17      int i = 0;
18      while (i * 2 + 1 < sz) {
19          int a = i * 2 + 1, b = i * 2 + 2;
20          if (b < sz && heap[b] < heap[a]) a = b;
21          if (heap[a] >= x) break;
22          heap[i] = heap[a];
23          i = a;
24      }
25      heap[i] = x;
26      return ret;
27  }
```

```
1   // Binary Search Tree
2   struct node {
3       int val;
4       node *lch, rch;
5   };
6   node *insert(node *p, int x) {
7       if (p == NULL) {
8           node *q = new node;
9           q->val = x;
10          q->lch = q->rch = NULL;
11          return q;
12      } else {
13          if (x < p->val) p->lch = insert(p->lch, x);
14          else p->rch = insert(p->rch, x);
```

```
15          return p;
16      }
17  }
18  bool find(node *p, int x) {
19      if (p == NULL) return false;
20      else if (x == p->val) return true;
21      else if (x < p->val) return find(p->lch, x);
22      else return find(p->rch, x);
23  }
24  node *remove(node *p, int x) {
25      if (p == NULL) return NULL;
26      else if (x < p->val) p->lch = remove(p->lch, x);
27      else if (x > p->val) p->rch = remove(p->rch, x);
28      else if (p->lch == NULL) {
29          node *q = p->rch;
30          delete p;
31          return q;
32      } else if (p->lch->rch == NULL) {
33          node *q = p->lch;
34          q->rch = p->rch;
35          delete p;
36          return q;
37      } else {
38          // 把左儿子子孙中最大的节点提到需要删除的节点上
39          node *q;
40          for (q = p->lch; q->rch->rch != NULL; q = q->rch);
41          node *r = q->rch;
42          q->rch = r->lch;
43          r->lch = p->lch;
44          r->rch = p->rch;
45          delete p;
46          return r;
47      }
48      return p;
49  }
```

```
1  // Union-find Set
2  int par[MAX_N];
3  int rnk[MAX_N];
4  void init(int n) {
5      for (int i = 0; i < n; ++i) {
6          par[i] = i;
7          rnk[i] = 0;
```

```
 8          }
 9     }
10     int find(int x) {
11         return par[x] == x? x : par[x] = find(par[x]);
12     }
13     bool same(int x, int y) {
14         return find(x) == find(y);
15     }
16     void unite(int x, int y) {
17         x = find(x);
18         y = find(y);
19         if (x == y) return;
20         if (rnk[x] < rnk[y]) {
21             par[x] = y;
22         } else {
23             par[y] = x;
24             if (rnk[x] == rnk[y]) rnk[x]++;
25         }
26     }
```

当然，更快捷简单的做法，是使用 C++ 的 container。

```
 1     // Segment Tree
 2     const int MAX_N = 1 << 17;
 3     int n, dat[2 * MAX_N - 1];
 4     void init(int _n) {
 5         n = 1;
 6         while (n < _n) n <<= 1;
 7         for (int i = 0; i < 2 * n - 1; ++i)
 8             dat[i] = INF;
 9     }
10     void update(int k, int a) {
11         k += n - 1;
12         dat[k] = a;
13         while (k > 0) {
14             k = (k - 1) / 2;
15             dat[k] = min(dat[2 * k + 1], dat[2 * k + 2]);
16         }
17     }
18     // query [a, b), index k in [l, r)
19     // query(a, b, 0, 0, n)
20     int query(int a, int b, int k, int l, int r) {
21         if (r <= a || b <= l) return INF;
```

```
22        if (a <= l && r <= b) return dat[k];
23        else {
24            int v1 = query(a, b, k * 2 + 1, l, (l + r) / 2);
25            int v2 = query(a, b, k * 2 + 2, (l + r) / 2, r);
26            return min(v1, v2);
27        }
28    }
```

```
1  // RMQ
2  int n, dat[2 * MAX_N - 1];
3  void init(int _n) {
4      n = 1;
5      while (n < _n) n <<= 1;
6      for (int i = 0; i < 2 * n - 1; ++i)
7          dat[i] = INF;
8  }
9  void update(int k, int a) {
10     k += n - 1;
11     dat[k] = a;
12     while (k > 0) {
13         k = (k - 1) / 2;
14         dat[k] = min(dat[2 * k + 1], dat[2 * k + 2]);
15     }
16 }
17 // query [a, b), index k in [l, r)
18 // query(a, b, 0, 0, n)
19 int query(int a, int b, int k, int l, int r) {
20     if (r <= a || b <= l) return INF;
21     if (a <= l && r <= b) return dat[k];
22     else {
23         int v1 = query(a, b, k * 2 + 1, l, (l + r) / 2);
24         int v2 = query(a, b, k * 2 + 2, (l + r) / 2, r);
25         return min(v1, v2);
26     }
27 }
```

```
1  //Sparse Table
2  const int MAX_N = 1e5 + 10;
3  const int MAX_K = 31 - __builtin_clz(MAX_N);
4  int n, ST[MAX_N][MAX_K + 1], A[MAX_N];
5  void build(int N) {
6      for (int i = 0; i < N; ++i)
7          ST[i][0] = A[i];
```

```
 8        int k = 31 - __builtin_clz(N);
 9        for (int j = 1; j <= k; ++j)
10            for (int i = 0; i <= N - (1 << j); ++i)
11                ST[i][j] = min(ST[i][j - 1i], ST[i + (1 << (j - 1))][j - 1])
                      ;
12    }
13    int query(int l, int r) {
14        if (l >= r) return 0;
15        int ans = INF, k = 31 - __builtin_clz(r - l);
16        for (int j = k; j >= 0; --j)
17            if (l + (1 << j) - 1 <= r) {
18                ans = min(ans, ST[l][j]);
19                l += 1 << j;
20            }
21        return ans;
22    }
23    int RMQ(int l, int r) {
24        if (l >= r) return 0;
25        int k = 31 - __builtin_clz(r - l);
26        return min(ST[l][k], ST[r - (1 << k)][k]);
27    }
```

---

4 Graph

```
1    struct edge {
2        int from;
3        int to, dis;
4    };
5    vector<edge> G[MAX_V];
6    vector<edge> es;
7    bool vis[MAX_V];
8    int V, E, pre[MAX_V], dist[MAX_V];
9    // int cost[MAX_V][MAX_V];
```

```
1    // Shortest Way
2    void floyed() {
3        for (int k = 1; k <= n; ++k)
4            for (int i = 1; i <= n; ++i)
5                for (int j = 1; i <= n; ++j)
```

```
 6              dis[i][j] = min(dis[i][k] + dis[k][j], dis[i][j]);
 7 }
 8 void dijkstra(int s) {
 9     priority_queue<Pii, vector<Pii>, greater<Pii> > que;// fisrt 是最短
                距离, second 是顶点编号
10     fill(dist, dist + V, INF);
11     dist[s] = 0; que.push(Pii(0, s));
12     while (!que.empty()) {
13         Pii p = que.top(); que.pop();
14         int v = p.second;
15         if (dist[v] < p.first) continue;
16         for (int i = 0; i < G[v].size(); i++) {
17             edge e = G[v][i];
18             if (dist[e.to] > dist[v] + e.dis) {
19                 dist[e.to] = dist[v] + e.dis;
20                 que.push(Pii(dist[e.to], e.to));
21             }
22         }
23     }
24 }
25 void bellman_ford(int s) {
26     fill(dist, dist + V, INF);
27     dist[s] = 0;
28     while (true) {
29         bool update = false;
30         for (int i = 0; i < E; ++i) {
31             edge e = es[i];
32             if (dist[e.from] != INF && dist[e.from] + e.dis < dist[e.to
                ]) {
33                 update = true;
34                 dist[e.to] = dist[e.from] + e.dis;
35             }
36         }
37         if (!update) break;
38     }
39 }
40 bool find_negative_loop() {
41     memset(dist, 0, sizeof dist);
42     for (int i = 0; i < V; ++i)
43         for (int j = 0; j < E; ++j) {
44             edge e = es[j];
45             if (d[e.to] > d[e.from] + e.dis) {
46                 d[e.to] = d[e.from] + e.dis;
```

```
47              if (i == V - 1) return true;
48          }
49        }
50    return false;
51  }
52  void spfa(int s) {
53      queue<int> que;
54      fill(dist, dist + V, INF);
55      fill(vis, vis + V, false);
56      dist[s] = 0; que.push(s); vis[s] = true;
57      while (!que.empty()) {
58          int v = que.front(); que.pop();
59          vis[v] = false;
60          for (int i = 0; i < G[v].size(); ++i) {
61              int u = G[v][i].to;
62              if (dist[u] > dist[v] + G[v][i].dis) {
63                  dist[u] = dist[v] + G[v][i].dis;
64                  if (!vis[u]) {
65                      que.push(u);
66                      vis[u] = true;
67                  }
68              }
69          }
70      }
71  }
```

```
1   // Spanning Tree
2   int prime() {
3       /*
4       fill(dist, dist + V, INF);
5       fill(vis, vis + V, false);
6       dist[0] = 0;
7       int res = 0;
8       while (true) {
9           int v = -1;
10          for (int u = 0; u < V; ++u) {
11              if(!vis[u] && (v == -1 || dist[u] < dist[v])) v = u;
12          }
13          if (v == -1) break;
14          vis[v] = true;
15          res += dist[v];
16          for (int u = 0; u < V; u++)
17              dist[u] = min(dist[u], cost[v][u]);
```

```
18          }
19      //*/
20      priority_queue<Pii, vector<Pii>, greater<Pii> > que;
21      int res = 0;
22      fill(dist, dist + V, INF);
23      fill(vis, vis + V, false);
24      dist[0] = 0;
25      que.push(Pii(0, 0));
26      while (!que.empty()) {
27          Pii p = que.top(); que.pop();
28          int v = p.second;
29          if (vis[v] || dist[v] < p.first) continue;
30          res += dist[v]; vis[v] = true;
31          for (int i = 0; i < G[v].size(); ++i) {
32              edge e = G[v][i];
33              if (dist[e.to] > e.dis) {
34                  dist[e.to] = e.dis;
35                  que.push(Pii(dist[e.to], e.to));
36              }
37          }
38      }
39      return res;
40  }
41  bool cmp(const edge e1, const edge e2) {
42      return e1.dis < e2.dis;
43  }
44  int kruskal() {
45      sort(es.begin(), es.end(), cmp);
46      init(V);
47      int res = 0;
48      for (int i = 0; i < E; ++i) {
49          edge e = es[i];
50          if (!same(e.from, e.to)) {
51              unite(e.from, e.to);
52              res += e.dis;
53          }
54      }
55      return res;
56  }
```

```
1  // SCC
2  int V, cmp[MAX_V];
3  vector<int> G[MAX_V], rG[MAX_V], vs;
```

```
 4  bool used[MAX_V];
 5
 6  void add_edge(int from, int to) {
 7      G[from].push_back(to); rG[to].push_back(from);
 8  }
 9  void dfs(int v) {
10      used[v] = true;
11      for (int i = 0; i < G[v].size(); ++i)
12          if (!used[G[v][i]]) dfs(G[v][i]);
13      vs.push_back(v);
14  }
15  void rdfs(int v, int k) {
16      used[v] = true;
17      cmp[v] = k;
18      for (int i = 0; i < rG[v].size(); ++i)
19          if (!used[rG[v][i]]) rdfs(rG[v][i], k);
20  }
21  int scc() {
22      memset(used, 0, sizeof used);
23      vs.clear();
24      for (int v = 0; v < V; ++v)
25          if (!used[v]) dfs(v);
26      memset(used, 0, sizeof used);
27      int k = 0;
28      for (int i = vs.size() - 1; i >= 0; --i)
29          if (!used[vs[i]]) rdfs(vs[i], k++);
30      return k;
31  }
```

```
 1  // Bipartite Matching
 2  void add_edge(int u, int v) {
 3      G[u].push_back(v); G[v].push_back(u);
 4  }
 5  bool dfs(int v) {
 6      used[v] = true;
 7      rep(i, 0, G[v].size()) {
 8          int u = G[v][i], w = match[u];
 9          if (w < 0 || (!used[w] && dfs(w))) {
10              match[v] = u; match[u] = v;
11              return true;
12          }
13      }
14      return false;
```

```
15  }
16  int bipartite_matching() {
17      int res = 0;
18      memset(match, -1, sizeof match);
19      rep (v, 0, V)
20          if (match[v] < 0) {
21              memset(used, false, sizeof used);
22              if (dfs(v)) ++res;
23          }
24      return res;
25  }
```

```
1   // Network Flow
2   struct edge{
3       int to, cap, rev;
4   };
5   vector<edge> G[MAX_V];
6   int level[MAX_V], iter[MAX_V];
7   void add_edge(int from, int to, int cap) {
8       G[from].push_back((edge){to, cap, static_cast<int>(G[to].size())});
9       G[to].push_back((edge){from, 0, static_cast<int>(G[from].size() - 1)
            });
10  }
11  // Ford-Fulkerson
12  int dfs(int v, int t, int f) {
13      if (v == t) return f;
14      flag[v] = true;
15      rep(i, 0, G[v].size()) {
16          edge &e = G[v][i];
17          if (!flag[e.to] && e.cap > 0) {
18              int d = dfs(e.to, t, min(f, e.cap));
19              if (d > 0) {
20                  e.cap -= d;
21                  G[e.to][e.rev].cap += d;
22                  return d;
23              }
24          }
25      }
26      return 0;
27  }
28  int max_flow(int s, int t) {
29      int flow = 0;
30      for(;;) {
```

```
31          memset(flag, false, sizeof flag);
32          int f = dfs(s, t, INF);
33          if (!f) return flow;
34          flow += f;
35      }
36 }
37 // Dinic
38 void bfs(int s) {
39      memset(level, -1, sizeof(level));
40      queue<int> que;
41      level[s] = 0; que.push(s);
42      while (!que.empty()) {
43          int v = que.front(); que.pop();
44          for (int i = 0; i < G[v].size(); ++i) {
45              edge &e = G[v][i];
46              if (e.cap > 0 && level[e.to] < 0) {
47                  level[e.to] = level[v] + 1;
48                  que.push(e.to);
49              }
50          }
51      }
52 }
53 int dfs(int v, int t, int f) {
54      if (v == t) return f;
55      for (int &i = iter[v]; i < G[v].size(); ++i) {
56          edge &e = G[v][i];
57          if (e.cap > 0 && level[v] < level[e.to]) {
58              int d = dfs(e.to, t, min(f, e.cap));
59              if (d > 0) {
60                  e.cap -= d;
61                  G[e.to][e.rev].cap += d;
62                  return d;
63              }
64          }
65      }
66      return 0;
67 }
68 int max_flow(int s, int t) {
69      int flow = 0;
70      for (;;) {
71          bfs(s);
72          if (level[t] < 0) return flow;
73          memset(iter, 0, sizeof iter);
```

```
74          int f;
75          while ((f = dfs(s, t, INF)) > 0) {
76              flow += f;
77          }
78      }
79 }
```

```
1  // min_cost_flow
2  void add_edge(int from, int to, int cap, int cost) {
3      G[from].push_back((edge){to, cap, cost, (int)G[to].size()});
4      G[to].push_back((edge){from, 0, -cost, (int)G[from].size() - 1});
5  }
6  int min_cost_flow(int s, int t, int f) {
7      int res = 0;
8      fill(h, h + V, 0);
9      while (f > 0) {
10         priority_queue<Pii, vector<Pii>, greater<Pii> > que;
11         fill(dist, dist + V, INF);
12         dist[s] = 0; que.push(Pii(0, s));
13         while (!que.empty()) {
14             Pii p = que.top(); que.pop();
15             int v = p.second;
16             if (dist[v] < p.first) continue;
17             rep(i, 0, G[v].size()) {
18                 edge &e = G[v][i];
19                 if (e.cap > 0 && dist[e.to] > dist[v] + e.cost + h[v] -
                     h[e.to]) {
20                     dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
21                     prevv[e.to] = v;
22                     preve[e.to] = i;
23                     que.push(Pii(dist[e.to], e.to));
24                 }
25             }
26         }
27         if (dist[t] == INF) return -1;
28         rep(v, 0, V) h[v] += dist[v];
29         int d = f;
30         for (int v = t; v != s; v = prevv[v])
31             d = min(d, G[prevv[v]][preve[v]].cap);
32         f -= d;
33         res += d * h[t];
34         for (int v = t; v != s; v = prevv[v]) {
35             edge &e = G[prevv[v]][preve[v]];
```

```
36            e.cap -= d;
37            G[v][e.rev].cap += d;
38        }
39    }
40    return res;
41 }
```

```
1 // stoer_wagner 全局最小割
2 void search() {
3    memset(vis, false, sizeof vis);
4    memset(wet, 0, sizeof wet);
5    S = T = -1;
6    int imax, tmp;
7    rep(i, 0, V) {
8        imax = -INF;
9        rep(j, 0, V)
10            if (!cmb[j] && !vis[j] && wet[j] > imax) {
11                imax = wet[j];
12                tmp = j;
13            }
14        if (T == tmp) return;
15        S = T; T = tmp;
16        mc = imax;
17        vis[tmp] = true;
18        rep(j, 0, V)
19            if (!cmb[j] && !vis[j])
20                wet[j] += G[tmp][j];
21    }
22 }
23 int stoer_wagner() {
24    memset(cmb, false, sizeof cmb);
25    int ans = INF;
26    rep(i, 0, V - 1) {
27        search();
28        ans = min(ans, mc);
29        if (ans == 0) return 0;
30        cmb[T] = true;
31        rep(j, 0, V)
32            if (!cmb[j]) {
33                G[S][j] += G[T][j];
34                G[j][S] += G[j][T];
35            }
36    }
```

```
37        return ans;
38  }
```

```
1   // LCA--Doubling
2   const int MAX_LOG_V = 32 - __builtin_clz(MAX_V);
3   vector<int> G[MAX_V];
4   int root, parent[MAX_LOG_V][MAX_V], depth[MAX_V];
5   void dfs(int v, int p, int d) {
6       parent[0][v] = p;
7       depth[v] = d;
8       for (int i = 0; i < G[v].size(); i++)
9           if (G[v][i] != p) dfs(G[v][i], v, d + 1);
10  }
11  void init(int V) {
12      dfs(root, -1, 0);
13      for (int k = 0; k + 1 < MAX_LOG_V; k++)
14          for (int v = 0; v < V; v++)
15              if (parent[k][v] < 0) parent[k + 1][v] = -1;
16              else parent[k + 1][v] = parent[k][parent[k][v]];
17  }
18  int lca(int u, int v) {
19      if (depth[u] > depth[v]) swap(u, v);
20      for (int k = 0; k < MAX_LOG_V; k++)
21          if ((depth[v] - depth[u]) >> k & 1)
22              v = parent[k][v];
23      if (u == v) return u;
24      for (int k = MAX_LOG_V - 1; k >= 0; k--)
25          if (parent[k][u] != parent[k][v])
26              u = parent[k][u], v = parent[k][v];
27      return parent[0][u];
28  }
29  // LCA--RMQ
30  vector<int> G[MAX_V];
31  int root, vs[MAX_V * 2 - 1], depth[MAX_V * 2 - 1], id[MAX_V];
32  int ST[2 * MAX_V][MAX_K];
33  void rmq_init(int* A, int N) {
34      for (int i = 0; i < N; i++)
35          ST[i][0] = i;
36      int k = 31 - __builtin_clz(N);
37      for (int j = 1; j <= k; j++)
38          for (int i = 0; i <= N - (1 << j); ++i)
39              if (A[ST[i][j - 1]] <= A[ST[i + (1 << (j - 1))][j - 1]])
40                  ST[i][j] = ST[i][j - 1];
```

```
41            else ST[i][j] = ST[i + (1 << (j - 1))][j - 1];
42  }
43  int query(int l, int r) {
44      if (l >= r) return -1;
45      int k = 31 - __builtin_clz(r - l);
46      return (depth[ST[l][k]] <= depth[ST[r - (1 << k)][k]]) ? ST[l][k] :
47          ST[r - (1 << k)][k];
47  }
48  void dfs(int v, int p, int d, int &k) {
49      id[v] = k;
50      vs[k] = v;
51      depth[k++] = d;
52      for (int i = 0; i < G[v].size(); i++) {
53          if (G[v][i] != p) {
54              dfs(G[v][i], v, d + 1, k);
55              vs[k] = v;
56              depth[k++] = d;
57          }
58      }
59  }
60  void init(int V) {
61      int k = 0;
62      dfs(root, -1, 0, k);
63      rmq_init(depth, 2 * V - 1);
64  }
65  int lca(int u, int v) {
66      return vs[query(min(id[u], id[v]), max(id[u], id[v]) + 1)];
67  }
```

5 Computational Geometry

```
1  const double EPS = 1e-8;
2  int sgn(double x) { return x < -EPS ? -1 : x > EPS ? 1 : 0;}
3  struct Point {
4      double x, y;
5      Point(double x = 0, double y = 0) : x(x), y(y) {}
6      Point operator + (Point p) { return Point(x + p.x, y + p.y); }
7      Point operator - (Point p) { return Point(x - p.x, y - p.y); }
8      Point operator * (double d) { return Point(x * d, y * d); }
```

```
 9      bool operator < (Point p) {
10          return x != p.x? x < p.x : y < p.y;
11      }
12      double dot(Point p) { return add(x * p.x, y * p.y); }// 内积
13      double det(Point p) { return add(x * p.y, -y * p.x);}// 外积
14  };
15  bool on_seg(Point p1, Point p2, Point q) {
16      return (p1 - q).det(p2 - q) == 0 && (p1 - q).dot(p2 - q) <= 0;
17  }
18  P intersection(Point p1, Point p2, Point q1, Point q2) {
19      return p1 + (p2 - p1) * ((q2 - q1).det(q1 - p1) / (q2 - q1).det(p2 -
            p1));
20  }
21  // 凸包
22  int convex_hull(Point *ps, int n, Point *ch) {
23      sort(ps, ps + n);
24      int k = 0;
25      for (int i = 0; i < n; ++i) {
26          while (k > 1 && (ch[k - 1] - ch[k - 2].det(ps[i] - ch[k - 1]))
                <= 0) k--;
27          ch[k++] = ps[i];
28      }
29      for (int i = n - 2, t = k; i >= 0; --i) {
30          while (k > t && (ch[k - 1] - ch[k - 2].det(ps[i] - ch[k - 1]))
                <= 0) k--;
31          ch[k++] = ps[i];
32      }
33      return k - 1;
34  }
```

Simpson 公式——二次函数近似原函数积分:

$$\int_a^b f(x)dx \approx \frac{b-a}{6} * \left(f(a) + 4f(\frac{a+b}{2}) + f(b) \right)$$

6 Math Problem

```
1  // returning count of nk in range [l, r], from Infinity
2  template<typename T> T mps(T l, T r, T k) {
3      return ((r - (r % k + k) % k) - (l + (k - l % k) % k)) / k + 1;
```

```
 4 }
 5 template<typename T> T gcd(T a, T b) {
 6     //return (b)? gcd(b, a  % b) : a;
 7     while (b) { T t = a % b; a = b; b = t; } return a;
 8 }
 9 template<typename T> T lcm(T a, T b) {
10     return a / gcd(a, b) * b;
11 }
12 // find (x, y) s.t. a x + b y = gcd(a, b) = d
13 template<typename T> T exgcd(T a, T b, T &x, T &y) {
14     T d = a;
15     if (b) {
16         d = exgcd(b, a % b, y, x);
17         y -= a / b * x;
18     } else {
19         x = 1; y = 0;
20     }
21     return d;
22 }
23 template<typename T> T modular_linear(T a, T b, T n) {
24     T d, e, x, y;
25     d = exgcd(a, n, x, y);
26     if (b % d)
27         return -1;
28     e = x * (b / d) % n + n;
29     return e % (n / d);
30 }
31 template<typename T> T mod_mult(T a, T b, T mod) {
32   T res = 0;
33   while (b) {
34     if (b & 1) {
35         res = (res + a) % mod;
36         // res += a;
37         // if (res >= mod) res -= mod;
38     }
39     a = (a + a) % mod;
40     // a <<= 1;
41     // if (a >= mod) a -= mod;
42     b >>= 1;
43   }
44   return res;
45 }
46 template<typename T> T mod_pow(T x, T n, T mod) {
```

```
47        T res = 1;
48        while (n) {
49            if (n & 1) res = mod_mult(res, x, mod);
50            x = mod_mult(x, x, mod);
51            n >>= 1;
52        }
53        return res;
54        // return b ? mod_pow(a * a % mod, b >> 1, mod) * (b & 1 ? a : 1) %
              mod : 1;
55    }
56    template<typename T> T mod_inverse(T a, T m) {
57        T x, y;
58        exgcd(a, m, x, y);
59        return (m + x % m) % m;
60    }
61    // 线性求逆元 小心乘法溢出?
62    void init_inverse() {
63        inv[1] = 1;
64        for (int i = 2; i < MAX_N; i++)
65            inv[i] = (MOD - (MOD / i) * inv[MOD % i] % MOD) % MOD;
66    }
67    //A[i] * x % M[i] = B[i];
68    std::pair<int, int> linear_congruence(const std::vector<int> &A, const
          std::vector<int> &B, const std::vector<int> &M) {
69            // wa 了把中间量开大? * 溢出
70            int x = 0, m = 1;
71            for(int i = 0; i < A.size(); i++) {
72                int a = A[i] * m, b = B[i] - A[i] * x, d = gcd(M[i], a);
73                if(b % d != 0) return std::make_pair(0, -1); // no
                      solutioin
74                int t = b / d * mod_inverse(a / d, M[i] / d) % (M[i] / d
                      );
75                x = x + m * t;
76                m *= M[i] / d;
77            }
78            while (x < m) x += m;
79            return std::make_pair(x % m, m);
80    }
81    ll CRT(vector<ll> &a, vector<ll> &m) {
82        ll M = 1LL, res = 0;
83        for (int i = 0; i < m.size(); ++i)
84            M *= m[i];
85        for (int i = 0; i < m.size(); ++i) {
```

```
 86          ll Mi, Ti;
 87          Mi = M / m[i]; Ti = mod_inverse(Mi, mi);
 88          res = (res + a[i] * (Mi * Ti % M) % M) % M;
 89      }
 90      return res;
 91  }
 92  // only for MOD < 1e6;
 93  ll fact[MOD + 10];
 94  void init() {
 95      fact[0] = 1;
 96      for (int i = 1; i <= MOD; ++i)
 97          fact[i] = fact[i - 1] * i % MOD;
 98  }
 99  int mod_fact(int n, int p, int &e) {
100      e = 0;
101      if (n == 0) return 1;
102      int res = mod_fact(n / p, p, e);
103      e += n / p;
104      if (n / p % 2 != 0) return res * (p - fact[n % p]) % p;
105      return res * fact[n % p] % p;
106  }
107  int mod_comb(int n, int k, int p) {
108      if (n < 0 || k < 0 || n < k) return 0;
109      if (n == 0) return 1;
110      int e1, e2, e3;
111      int a1 = mod_fact(n, p, e1), a2 = mod_fact(k, p, e2), a3 = mod_fact(
             n - k, p, e3);
112      if (e1 > e2 + e3) return 0;
113      return a1 * mod_inverse(a2 * a3 % p, p) % p;
114  }
115  ll lucas(ll n, ll k, const ll &p) {
116      if (n < 0 || k < 0 || n < k) return 0;
117      if (n == 0) return 1;
118      return lucas(n / p, k / p, p) * mod_comb(n % p, k % p, p) % p;
119  }
```

```
 1  // 矩阵快速幂
 2  typedef vector<int> vec;
 3  typedef vector<vec> mat;
 4  mat G(MAX_N);
 5  mat mat_mul(mat &A, mat &B) {
 6      mat C(A.size(), vec(B[0].size()));
 7      for (int i = 0; i < A.size(); ++i)
```

```
 8              for (int k = 0; k < B.size(); ++k)
 9                  for (int j = 0; j < B[0].size(); ++j)
10                      C[i][j] = (C[i][j] + A[i][k] % MOD * B[k][j] % MOD + MOD
                            ) % MOD;
11          return C;
12  }
13  mat mat_pow(mat A, ll n) {
14      mat B(A.size(), vec(A.size()));
15      for (int i = 0; i < A.size(); ++i)
16          B[i][i] = 1;
17      while (n > 0) {
18          if (n & 1) B = mat_mul(B, A);
19          A = mat_mul(A, A);
20          n >>= 1;
21      }
22      return B;
23  }
```

```
 1  // prime number
 2  bool is_prime(int n) {
 3      for (int i = 2; i * i <= n; ++i)
 4          if (n % i == 0) return false;
 5      return n != 1;
 6  }
 7  vector<int> divisor(int n) {
 8      vector<int> res;
 9      for (int i = 1; i * i <= n; ++i) {
10          if (n % i == 0) {
11              res.push_back(i);
12              if (i != n / i) res.push_back(n / i);
13          }
14      }
15      return res;
16  }
17  map<int, int> prime_factor(int n) {
18      map<int, int> res;
19      for (int i = 2; i * i <= n; ++i) {
20          while (n % i == 0) {
21              ++res[i];
22              n /= i;
23          }
24      }
25      if (n != 1) res[n] = 1;
```

```
26        return res;
27 }
28 int prime[MAX_N];
29 bool isPrime[MAX_N + 1];
30 int seive(int n) {
31     int p = 0;
32     fill(isPrime, isPrime + n + 1, true);
33     isPrime[0] = isPrime[1] = false;
34     for (int i = 2; i <= n; ++i)
35         if (isPrime[i]) {
36             prime[p++] = i;
37             for (int j = 2 * i; j <= n; j += i) isPrime[j] = false;
38         }
39     return p;
40 }
41 // the number of prime in [L, r)
42 // 对区间 [l, r) 内的整数执行筛法, prime[i - l] = true <=> i 是素数
43 bool segPrimeSmall[MAX_L];
44 bool segPrime[MAX_SQRT_R];
45 void segment_sieve(ll l, ll r) {
46     for (int i = 0; (ll)i * i < r; ++i) segPrimeSmall[i] = true;
47     for (int i = 0; i < r - l; ++i) segPrime[i] = true;
48     for (int i = 2; (ll)i * i < r; ++i) {
49         if (segPrimeSmall[i]) {
50             for (int j = 2 * i; (ll)j * j <= r; j += i) segPrimeSmall[j]
                  = false;
51             for (ll j = max(2ll, (l + i - 1) / i) * i; j < r; j += i)
                  segPrime[j - l] = false;
52         }
53     }
54 }
55 // Miller_Rabin
56 bool check(ll a, ll n, ll x, ll t) {
57     ll res = mod_pow(a, x, n);
58     ll last = res;
59     for (int i = 1; i <= t; ++i) {
60         res = mod_mult(res, res, n);
61         if (res == 1 && last != 1 && last != n - 1) return true;
62         last = res;
63     }
64     if (res != 1) return true;
65     return false;
66 }
```

```
67  bool Miller_Rabin(ll n) {
68      if (n < MAX_N) return isPrime[n]; // small number may get wrong
            answer?!
69      if (n < 2) return false;
70      if (n == 2) return true;
71      if ((n & 1) == 0) return false;
72      ll x = n - 1, t = 0;
73      while ((x & 1) == 0) {
74          x >>= 1;
75          ++t;
76      }
77      for (int i = 0; i < S; ++i) {
78          ll a = rand() % (n - 1) + 1;
79          if (check(a, n, x, t))
80              return false;
81      }
82      return true;
83  }
84  // find factors
85  vector<ll> factor;
86  ll Pollard_rho(ll x, ll c) {
87      ll i = 1, k = 2;
88      ll x0 = rand() % x;
89      ll y = x0;
90      while (true) {
91          ++i;
92          x0 = (mod_mult(x0, x0, x) + c) % x;
93          ll d;
94          if (y == x0) d = 1;
95          else
96              if (y > x0)
97                  d = gcd(y - x0, x);
98              else d = gcd(x0 - y, x);
99          if (d != 1 && d != x) return d;
100         if (y == x0) return x;
101         if (i == k) {
102             y = x0;
103             k += k;
104         }
105     }
106 }
107 void find_factor(ll n) {
108     if (n == 1) return ;
```

```
109        if (Miller_Rabin(n)) {
110            factor.push_back(n);
111            return ;
112        }
113        ll p = n;
114        while (p >= n) p = Pollard_rho(p, rand() % (n - 1) + 1);
115        find_factor(p);
116        find_factor(n / p);
117    }
```

```
1   #include<bits/stdc++>
2   //Meisell-Lehmer
3   const int MAX_N = 5e6 + 2;
4   bool np[MAX_N];
5   int prime[MAX_N], pi[MAX_N];
6   int getprime() {
7       int cnt = 0;
8       np[0] = np[1] = true;
9       pi[0] = pi[1] = 0;
10      for(int i = 2; i < MAX_N; ++i) {
11          if(!np[i]) prime[++cnt] = i;
12          pi[i] = cnt;
13          for(int j = 1; j <= cnt && i * prime[j] < MAX_N; ++j) {
14              np[i * prime[j]] = true;
15              if(i % prime[j] == 0)   break;
16          }
17      }
18      return cnt;
19  }
20  const int M = 7;
21  const int PM = 2 * 3 * 5 * 7 * 11 * 13 * 17;
22  int phi[PM + 1][M + 1], sz[M + 1];
23  void init() {
24      getprime();
25      sz[0] = 1;
26      for(int i = 0; i <= PM; ++i)  phi[i][0] = i;
27      for(int i = 1; i <= M; ++i) {
28          sz[i] = prime[i] * sz[i - 1];
29          for(int j = 1; j <= PM; ++j) phi[j][i] = phi[j][i - 1] - phi[j /
                prime[i]][i - 1];
30      }
31  }
32  int sqrt2(ll x) {
```

```
33      ll r = (ll)sqrt(x - 0.1);
34      while(r * r <= x)    ++r;
35      return int(r - 1);
36  }
37  int sqrt3(ll x) {
38      ll r = (ll)cbrt(x - 0.1);
39      while(r * r * r <= x)    ++r;
40      return int(r - 1);
41  }
42  ll getphi(ll x, int s) {
43      if(s == 0)  return x;
44      if(s <= M)  return phi[x % sz[s]][s] + (x / sz[s]) * phi[sz[s]][s];
45      if(x <= prime[s]*prime[s])   return pi[x] - s + 1;
46      if(x <= prime[s]*prime[s]*prime[s] && x < MAX_N) {
47          int s2x = pi[sqrt2(x)];
48          ll ans = pi[x] - (s2x + s - 2) * (s2x - s + 1) / 2;
49          for(int i = s + 1; i <= s2x; ++i) ans += pi[x / prime[i]];
50          return ans;
51      }
52      return getphi(x, s - 1) - getphi(x / prime[s], s - 1);
53  }
54  ll getpi(ll x) {
55      if(x < MAX_N)   return pi[x];
56      ll ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;
57      for(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) ans
            -= getpi(x / prime[i]) - i + 1;
58      return ans;
59  }
60  ll lehmer_pi(ll x) {
61      if(x < MAX_N)   return pi[x];
62      int a = (int)lehmer_pi(sqrt2(sqrt2(x)));
63      int b = (int)lehmer_pi(sqrt2(x));
64      int c = (int)lehmer_pi(sqrt3(x));
65      ll sum = getphi(x, a) +(ll)(b + a - 2) * (b - a + 1) / 2;
66      for (int i = a + 1; i <= b; i++) {
67          ll w = x / prime[i];
68          sum -= lehmer_pi(w);
69          if (i > c) continue;
70          ll lim = lehmer_pi(sqrt2(w));
71          for (int j = i; j <= lim; j++) sum -= lehmer_pi(w / prime[j]) -
            (j - 1);
72      }
73      return sum;
```

```
74  }
75  int main() {
76      init();
77      ll n;
78      while(~scanf("%lld",&n)) {
79          printf("%lld\n",lehmer_pi(n));
80      }
81      return 0;
82  }
```

```
1   // 欧拉函数
2   int euler_phi(int n) {
3       int res = n;
4       for (int i = 2; i * i <= n; ++i) {
5           if (n % i == 0) {
6               res = res / i * (i - 1);
7               for (; n % i == 0; n /= i);
8           }
9       }
10      if (n != 1) res = res / n * (n - 1);
11      return res;
12  }
13  int euler[MAX_N];
14  void euler_phi_sieve() {
15      for (int i = 0; i < MAX_N; ++i) euler[i] = i;
16      for (int i = 2; i < MAX_N; ++i)
17          if (euler[i] == i)
18              for (int j = i; j < MAX_N; j += i) euler[j] = euler[j] / i *
                    (i - 1);
19  }
```

- Moebius 如果

$$F(n) = \sum_{d|n} f(d)$$

，则

$$f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$$

对于 $\mu(d)$ 函数，有如下性质：

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n > 1 \end{cases}$$

$$\sum_{d|n} \frac{\mu(d)}{d} = \frac{\phi(n)}{n}$$

```
1   int mu[MAX_N];
2   void moebius() {
3       int cnt = 0; mu[1] = 1;
4       memset(vis, 0, sizeof vis);
5       for (int i = 2; i < MAX_N; ++i) {
6           if (!vis[i]) {
7               prime[cnt++] = i;
8               mu[i] = -1;
9           }
10          for (int j = 0; j < cnt && i * prime[j] < MAX_N; ++j) {
11              vis[i * prime[j]] = true;
12              if (i % prime[j])
13                  mu[i * prime[j]] = -mu[i];
14              else
15                  mu[i * prime[j]] = 0, break;
16          }
17      }
18  }
19  map<int, int> moebius(int n) {
20      map<int, int> res;
21      vector<int> primes;
22      for (int i = 2; i * i <= n; ++i) {
23          if (n % i == 0) {
24              primes.push_back(i);
25              while (n % i == 0) n /= i;
26          }
27      }
28      if (n != 1) primes.push_back(n);
29      int m = primes.size();
30      for (int i = 0; i < (1 << m); ++i) {
31          int mu = 1, d = 1;
32          for (int j = 0; j < m; ++j) {
33              if (i >> j & 1) {
34                  mu *= -1;
35                  d *= primes[j];
36              }
37          }
38          res[d] = mu;
39      }
```

```
40      return res;
41  }
```

```
1   // Guass_jordan
2   const double eps = 1e-8;
3   typedef vector<double> vec;
4   typedef vector<vec> mat;
5   vec gauss_joedan(const mat &A, const vec& b) {
6       int n = A.size();
7       mat B(n, vec(n + 1));
8       for (int i = 0; i < n; ++i)
9           for (int j = 0; j < n; ++j) B[i][j] = A[i][j];
10      for (int i = 0; i < n; ++i) B[i][n] = b[i];
11      for (int i = 0; i < n; ++i) {
12          int pivot = i;
13          for (int j = i; j < n; ++j)
14              if (abs(B[j][i]) > abs(B[pivot][i])) pivot = j;
15          if (i != pivot) swap(B[i], B[pivot]);
16          if (abs(B[i][i]) < eps) return vec();
17          for (int j = i + 1; j <= n; ++j) B[i][j] /= B[i][i];
18          for (int j = 0; j < n; ++j) if (i != j)
19              for (int k = i + 1; k <= n; ++k) B[j][k] -= B[j][i] * B[i][k
                    ];
20      }
21      vec x(n);
22      for (int i = 0; i < n; ++i) x[i] = B[i][n];
23      return x;
24  }
25  vec gauss_joedan_xor(const mat& A, const vec& b) {
26      int n = A.size();
27      mat B(n, vec(n + 1));
28      for (int i = 0; i < n; ++i)
29          for (int j = 0; j < n; ++j) B[i][j] = A[i][j];
30      for (int i = 0; i < n; ++i) B[i][n] = b[i];
31      for (int i = 0; i < n; ++i) {
32          int pivot = i;
33          for (int j = i; j < n; ++j)
34              if (B[j][i]) {
35                  pivot = j;
36                  break;
37              }
38          if (pivot != i) swap(B[i], B[pivot]);
39
```

```
40          for (int j = 0; j < n; ++j) if (i != j && B[j][i])
41                  for (int k = i + 1; k <= n; ++k) B[j][k] ^= B[i][k];
42          }
43      }
44      vec x(n);
45      for (int i = 0; i < n; ++i) x[i] = B[i][n];
46      return x;
47 }
```

7 String

1. Hash 2. KMP 3. Extend KMP 4. trie 树 poj2001 2503 3630 1056 hdu 1075 1251 1247 1298 1671 5. Manacher 算法 6. AC 自动机 7. 后缀数组 8. 后缀树 9. 后缀自动机 10. 回文自动机

```
1 // 最小最大表示法:
2 int getMinString(const string &s) {
3     int len = (int)s.length();
4     int i = 0, j = 1, k = 0;
5     while(i < len && j < len && k < len) {
6         int t = s[(i + k) % len] - s[(j + k) % len];
7         if(t == 0) k++;
8         else {
9             if(t > 0) i += k + 1;//getMaxString: t < 0
10            else j += k + 1;
11            if(i == j) j++;
12            k = 0;
13        }
14    }
15    return min(i, j);
16 }
```

```
1 // KMP
2 int nxt[MAX_N];
3 void getNext(const string &str) {
4     int len = str.length();
5     int j = 0, k;
6     k = nxt[0] = -1;
7     while (j < len) {
8         if (k == -1 || str[j] == str[k])
9             nxt[++j] = ++k;
```

```
10          else k = nxt[k];
11      }
12  }
13  int kmp(const string &tar, const string &pat) {
14      getNext(pat);
15      int num, j, k;
16      int lenT = tar.length(), lenP = pat.length();
17      num = j = k = 0;
18      while (j < lenT) {
19          if(k == -1 || tar[j] == pat[k])
20              j++, k++;
21          else k = nxt[k];
22          if(k == lenP) {
23              // res = max(res, j - lenP);
24              k = nxt[k];
25              ++num;
26          }
27      }
28      return num;//lenP - res - 1;
29  }
```

```
1  // Suffix Array & LCP Array
2  int n, k;
3  int lcp[MAX_N], sa[MAX_N];
4  int rnk[MAX_N], tmp[MAX_N];
5  bool compare_sa(int i, int j) {
6      if (rnk[i] != rnk[j]) return rnk[i] < rnk[j];
7      else {
8          int ri = i + k <= n? rnk[i + k] : -1;
9          int rj = j + k <= n? rnk[j + k] : -1;
10         return ri < rj;
11     }
12 }
13 void construct_sa(string S, int *sa) {
14     n = S.length();
15     for (int i = 0; i <= n; i++) {
16         sa[i] = i;
17         rnk[i] = i < n? S[i] : -1;
18     }
19     for (k = 1; k <= n; k *= 2) {
20         sort(sa, sa + n + 1, compare_sa);
21         tmp[sa[0]] = 0;
22         for (int i = 1; i <= n; i++)
```

```
23              tmp[sa[i]] = tmp[sa[i - 1]] + (compare_sa(sa[i - 1], sa[i])
                    ? 1 : 0);
24          memcpy(rnk, tmp, sizeof(int) * (n + 1));
25      }
26  }
27  void construct_lcp(string S, int *sa, int *lcp) {
28      int n = S.length();
29      for (int i = 0; i <= n; i++) rnk[sa[i]] = i;
30      int h = 0;
31      lcp[0] = 0;
32      for (int i = 0; i < n; i++) {
33          int j = sa[rnk[i] - 1];
34          if (h > 0) h--;
35          for (; j + h < n && i + h < n; h++)
36              if (S[j + h] != S[i + h]) break;
37          lcp[rnk[i] - 1] = h;
38      }
39  }
```

```
1  // AC 自动机
2  int ans[MAX_N], d[MAX_N];
3  struct Trie {
4      int nxt[MAX_N][26], fail[MAX_N], end[MAX_N];
5      int root, L;
6      int newnode() {
7          for(int i = 0; i < 26; i++)
8              nxt[L][i] = -1;
9          end[L++] = 0;
10         return L-1;
11     }
12     void init() {
13         L = 0;
14         root = newnode();
15     }
16     void insert(char buf[]) {
17         int len = strlen(buf);
18         int now = root;
19         for(int i = 0; i < len; i++) {
20             if(nxt[now][buf[i]-'a'] == -1)
21                 nxt[now][buf[i]-'a'] = newnode();
22             now = nxt[now][buf[i]-'a'];
23         }
24         end[now] = 1;
```

```
25              d[now] = len;
26          }
27      void build() {
28          queue<int> Q;
29          fail[root] = root;
30          for(int i = 0; i < 26; i++)
31              if(nxt[root][i] == -1)
32                  nxt[root][i] = root;
33              else {
34                  fail[nxt[root][i]] = root;
35                  Q.push(nxt[root][i]);
36              }
37          while( !Q.empty() ) {
38              int now = Q.front(); Q.pop();
39              for(int i = 0; i < 26; i++)
40                  if(nxt[now][i] == -1)
41                      nxt[now][i] = nxt[fail[now]][i];
42                  else {
43                      fail[nxt[now][i]] = nxt[fail[now]][i];
44                      Q.push(nxt[now][i]);
45                  }
46          }
47      }
48      void solve(char buf[]) {
49          int cur = root;
50          int len = strlen(buf);
51          int index;
52          for(int i = 0; i < len; ++i) {
53              if(buf[i] >= 'A' && buf[i] <= 'Z')
54                  index = buf[i] - 'A';
55              else if(buf[i] >= 'a' && buf[i] <= 'z')
56                  index = buf[i] - 'a';
57              else continue;
58              cur = nxt[cur][index];
59              int x = cur;
60              while(x != root) {
61                  if(end[x]) {
62                      ans[i + 1] -= 1;
63                      ans[i - d[x] + 1] += 1;
64                      break;
65                  }
66                  x = fail[x];
67              }
```

```
68          }
69      }
70 };
71 Trie ac;
```

---

8 Others

8.1 Divide-and-Conquer Tree

```cpp
//uva 12161
struct edge {
    int to, damage, length, next;
};
int G[MAX_N], En, N, M, T;
edge E[MAX_N * 2];
void add_edge(int from, int to, int damage, int length) {
    edge e = {to, damage, length, G[from]};
    E[En] = e;
    G[from] = En++;
}
int ans, subtree_size[MAX_N];
bool flag[MAX_N];
int s, t;
Pii ds[MAX_N];
int compute_subtree_size(int v, int p) {
    int c = 1;
    for (int j = G[v]; ~j; j = E[j].next) {
        int w = E[j].to;
        if (w == p || flag[w]) continue;
        c += compute_subtree_size(w, v);
    }
    return subtree_size[v] = c;
}
Pii search_centroid(int v, int p, int t) {
    Pii res = Pii(INT_MAX, -1);
    int s = 1, m = 0;
    for (int j = G[v]; ~j; j = E[j].next) {
        int w = E[j].to;
        if (w == p || flag[w]) continue;
```

```
31            res = min(res, search_centroid(w, v, t));
32            m = max(subtree_size[w], m);
33            s += subtree_size[w];
34        }
35        m = max(m, t - s);
36        res = min(res, Pii(m, v));
37        return res;
38    }
39    void enumrate_path(int v, int p, int damage, int length) {
40        ds[t++] = Pii(damage, length);
41        for (int j = G[v]; ~j; j = E[j].next) {
42            int w = E[j].to;
43            if (w == p || flag[w]) continue;
44            if (damage + E[j].damage <= M) {
45                enumrate_path(w, v, damage + E[j].damage, length + E[j].
                    length);
46            }
47        }
48    }
49    void remove_useless(int s, int &t) {
50        if (s == t) return;
51        int tt;
52        for (int i = tt = s + 1; i < t; i++) {
53            if (ds[i].first == ds[tt - 1].first) continue;
54            if (ds[i].second <= ds[tt - 1].second) continue;
55            ds[tt++] = ds[i];
56        }
57        t = tt;
58    }
59    void solve_sub_problem(int v) {
60        compute_subtree_size(v, -1);
61        int c = search_centroid(v, -1, subtree_size[v]).second;
62        flag[c] = true;
63        for (int j = G[c]; ~j; j = E[j].next) {
64            if (flag[E[j].to]) continue;
65            solve_sub_problem(E[j].to);
66        }
67        s = t = 0;
68        for (int j = G[c]; ~j; j = E[j].next) {
69            int w = E[j].to;
70            if (flag[w]) continue;
71            if (E[j].damage <= M)
72                enumrate_path(w, v, E[j].damage, E[j].length);
```

```
73          if (s > 0) {
74              sort(ds + s, ds + t);
75              remove_useless(s, t);
76              for (int l = 0, r = t - 1; l < s && r >= s; l++) {
77                  while (r >= s && ds[l].first + ds[r].first > M) r--;
78                  if (r >= s)
79                      ans = max(ans, ds[l].second + ds[r].second);
80              }
81          }
82          sort(ds, ds + t);
83          remove_useless(0, t);
84          s = t;
85      }
86      flag[c] = false;
87  }
```