Basic Algorithm

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

```
int mersgesort(int a[], int b[], int c[], int an, int bn) {
    int i = 0, j = 0, k = 0;
    while (i < an && j < bn) {
        if (a[i] < b[j]) {
            c[k++] = a[i++];
        }
        else {
            c[k++] = b[j++];
        }
    }
    while (i < an)
        c[k++] = a[i++];
    while (j < bn)
        c[k++] = b[j++];
    return k;
}
```

- Quick Sort

```
void quicksort(int a[], int l, int r) {
    int i = l, j = r, mid = a[(l + r) / 2];
    while (i <= j) {
        while (a[i] < mid) i++;
        while (a[j] > mid) j--;
        if (i <= j) {
            swap(a[i], a[j]);
            i++;j--;
        }
    }
    if (i < r) quicksort(a, i, r);
    if (l < j) quicksort(a, l, j);
    return ;
}
```

- Heap Sort
  - 见堆的内容

Knapsack Problem

- 0/1 背包

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

```
for (int i = 0; i < n; ++i)
    for (int v = V; v >= w[i]; v--)
        f[v] = max(f[v], f[v- w[i]] + c[i]);
```

- 完全背包

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

```
for (int i = 0; i < n; ++i)
    for (int v = w[i]; v <= V; ++v)
        f[v] = max(f[v], f[v- w[i]] + c[i]);
```

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

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

- 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：$1, 2, 4, \ldots, 2^{(k-1)}, n[i] - 2^k$
  - **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）**

```
for (int i = 0; i < N; ++i) {
    int k;
    for (k = 1 << 0; k <= n[i] && w[i] * k <= V; n[i] -= k, k <<= 1) {
        for (int v = V; v >= w[i] * k; --v)
            f[v] = max(f[v], f[v- w[i] * k] + c[i] * k);
    }
    k = n[i];
    if ( k > 0 && w[i] * k <= V) {
        for (int v = V; v >= w[i] * k; --v)
            f[v] = max(f[v], f[v- w[i] * k] + c[i] * k);
    }
}
```

- 混合三种背包

  弄清楚上面三种背包后分情况就好

- 二维费用背包

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

二维费用可由最多取 m 件等方式隐蔽给出。

- 分组背包

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

```
    for (int k = 0; k < K; ++k)
        for (v = V; v >= 0; --v)
            for (int i = 0; i <= n[k]; ++i)
                f[v] = max(f[v], f[v - w[i]] + c[i]);
```

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

$$f[i,v] = sumf[i-1,v], f[i-1,v-w[i]] + c[i], f[0,0] = 0$$

> 更多内容详见「背包九讲」

Data Structure

- Heap

```
int heap[maxn], sz = 0;
void push(int x) {
    int i = sz++;
    while (i > 0) {
        int p = (i - 1) / 2;
        if (heap[p] <= x) break;
        heap[p] = heap[i];
        i = p;
    }
    heap[i] = x;
}
int pop() {
    int ret = heap[0];
    int x = heap[--sz];
    int i = 0;
    while (i * 2 + 1 < sz) {
        int a = i * 2 + 1, b = i * 2 + 2;
        if (b < sz && heap[b] < heap[a]) a = b;
        if (heap[a] >= x) break;
        heap[i] = heap[a];
        i = a;
    }
    heap[i] = x;
    return ret;
}
```

- Binary Search Tree

```
struct node {
    int val;
    node *lch, rch;
};

node *insert(node *p, int x) {
    if (p == NULL) {
        node *q = new node;
        q->val = x;
        q->lch = q->rch = NULL;
        return q;
    } else {
        if (x < p->val) p->lch = insert(p->lch, x);
        else p->rch = insert(p->rch, x);
```

```
        return p;
    }
}
bool find(node *p, int x) {
    if (p == NULL) return false;
    else if (x == p->val) return true;
    else if (x < p->val) return find(p->lch, x);
    else return find(p->rch, x);
}
node *remove(node *p, int x) {
    if (p == NULL) return NULL;
    else if (x < p->val) p->lch = remove(p->lch, x);
    else if (x > p->val) p->rch = remove(p->rch, x);
    else if (p->lch == NULL) {
        node *q = p->rch;
        delete p;
        return q;
    } else if (p->lch->rch == NULL) {
        node *q = p->lch;
        q->rch = p->rch;
        delete p;
        return q;
    }  else {
        // 把左儿子子孙中最大的节点提到需要删除的节点上
        node *q;
        for (q = p->lch; q->rch->rch != NULL; q = q->rch);
        node *r = q->rch;
        q->rch = r->lch;
        r->lch = p->lch;
        r->rch = p->rch;
        delete p;
        return r;
    }
    return p;
}
```

- Union-find Set

```
int par[MAX_N];
int rnk[MAX_N];

void init(int n) {
    for (int i = 0; i < n; ++i) {
        par[i] = i;
        rnk[i] = 0;
    }
}
int find(int x) {
    return par[x] == x? x : par[x] = find(par[x]);
}
bool same(int a, int b) {
    return find(a) == find(b);
}
void unite(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return;
    if (rnk[x] < rnk[y]) {
        par[x] = y;
    } else {
        par[y] = x;
        if (rnk[x] == rnk[y]) rnk[x]++;
    }
}
```

当然，更快捷简单的做法，是使用 C++ 的 container。

Graph

```
struct edge {
    int u, v;
    int dis;
};
vector<edge> g[MAX_V];
vector<edge> es;
bool vis[MAX_V];
int d[MAX_N], V, E, pre[MAX_V];
int cost[MAX_V][MAX_V];
```

- Shortest Way

```
void dijkstra(int s) {
    //
    // fill(d, d + V, INF);
    // fill(vis, vis + V, false);
    // d[s] = 0;
    // while (true) {
    //     int v = -1;
    //     for (int u = 0; u < V; ++u)
    //         if (!vis[u] && (v == -1 || d[u] < d[v])) v = u;
    //     if (v == -1) break;
    //     vis[v] = true;
    //     for (int u = 0; u < V; ++u) {
    //         d[u] = min(d[v] + cost[v][u], d[u]);
    //     }
    // }
    //
    priority_queue<Pii, vector<Pii>, greater<Pii> > que;// fisrt 是最短距离，second 是顶点编号
    fill(d, d + V, INF);
    d[s] = 0;
    que.push(Pii(0, s));
    while (!que.empty()) {
        Pii p = que.top();que.pop();
        int u = p.second;
        if (d[u] < p.first) continue;
        for (int i = 0; i < g[u].size(); i++) {
            edge e = g[u][i];
            if (d[e.v] > d[u] + e.dis) {
                d[e.v] = d[u] + e.dis;
                que.push(Pii(d[e.v], e.v));
            }
        }
    }
}
void bellman_ford(int s) {
    fill(d, d + V, INF);
    d[s] = 0;
    //
    // for (int i = 0; i < V; ++i)
    //     for (int j = 0; j < E; ++j) {
    //         int u, v;
    //         u = es[j].u;
    //         v = es[j].v;
    //         if (d[u] != INF)
    //             d[v] = min(d[u] + es[j].dis, d[v]);
    //         if (d[v] != INF)
    //             d[u] = min(d[v] + es[j].dis, d[u]);
    //     }
    //
    while (true) {
        bool update = false;
        for (int i = 0; i < E; ++i) {
            edge e = es[i];
            if (d[e.u] != INF && d[e.u] + e.dis < d[e.v]) {
                update = true;
```

```
                d[e.v] = d[e.u] + e.dis;
            }
        }
        if (!update) break;
    }
}
void spfa(int s) {
    queue<int> que;
    fill(d, d + V, INF);
    fill(vis, vis + V, false);
    d[s] = 0;
    que.push(s);
    vis[s] = true;
    while (!que.empty()) {
        int u = que.front();
        que.pop();
        vis[u] = false;
        for (int i = 0; i < g[u].size(); ++i) {
            int v = g[u][i].v;
            if (d[v] > d[u] + g[u][i].dis) {
                d[v] = d[u] + g[u][i].dis;
                if (!vis[v]) {
                    que.push(v);
                    vis[v] = true;
                }
            }
        }
    }
}
```

- Spanning Tree

```
int prime() {
    fill(d, d + V, INF);
    fill(vis, vis + V, false);
    d[0] = 0;
    int res = 0;
    while (true) {
        int v = -1;
        for (int u = 0; u < V; ++u) {
            if(!vis[u] && (v == -1 || d[u] < d[v])) v = u;
        }
        if (v == -1) break;
        vis[v] = true;
        res += d[v];
        for (int u = 0; u < V; u++)
            d[u] = min(d[u], cost[v][u]);
    }
    return res;
}
bool cmp(edge &e1, edge &e2) {
    return e1.dis < e2.dis;
}
int kruskal() {
    sort(es, es + E, cmp);
    init(V);
    int res = 0;
    for (int i = 0; i < E; ++i) {
        edge e = es[i];
        if (!same(e.u, e.v)) {
            unite(e.u, e.v);
            res += e.dis;
        }
    }
    return res;
}
```

Math Problem

```cpp
template<typename T> T gcd(T a, T b) {
    while (b) { T t = a % b; a = b; b = t; } return a;
}
template<typename T> T lcm(T a, T b) {
    return a / gcd(a, b) * b;
}
long long qpow(long long a, long long b, long long mod) {
    return b ? qpow(a * a % mod, b >> 1, mod) * (b & 1 ? a : 1) % mod : 1;
}
// find (x, y) s.t. a x + b y = gcd(a, b) = d
template<typename T> T exGcd(T a, T b, T &x, T &y) {
    T d = a; if (b) { d = exGcd(b, a % b, y, x); y -= a / b * x; } else { x = 1; y = 0; } return d;
}
// returning count of nk in range [l, r]
template<typename T> T mps(T l, T r, T k) {
    return ((r - (r % k + k) % k) - (l + (k - l % k) % k)) / k + 1;
}
```