_02广度优先搜索 & _03贪心算法

广度优先搜索 (BFS)

核心思想：层层扩展，先访问距离近的节点

特点

- 队列实现
- 保证找到最短路径
- 适合求解最短路径问题

时间复杂度：$O(V+E)$

BFS 模板

```cpp
#include <queue>
using namespace std;

bool visited[N];
int dist[N]; // 距离数组

void bfs(int start) {
    queue<int> q;
    q.push(start);
    visited[start] = true;
    dist[start] = 0;

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        // 处理当前节点
        cout << u << " ";

        // 遍历邻居
        for (int v : neighbors[u]) {
            if (!visited[v]) {
                visited[v] = true;
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

BFS 示例：迷宫最短路径

```cpp
#include <queue>
#include <cstring>
using namespace std;

const int N = 100;
char maze[N][N];
bool vis[N][N];
int dist[N][N];
int dx[4] = {0, 1, 0, -1};
int dy[4] = {1, 0, -1, 0};

struct Point {
    int x, y;
};
```

```cpp
int bfs(int sx, int sy, int ex, int ey, int n, int m) {
    queue<Point> q;
    q.push({sx, sy});
    vis[sx][sy] = true;
    dist[sx][sy] = 0;

    while (!q.empty()) {
        Point p = q.front();
        q.pop();

        if (p.x == ex && p.y == ey) {
            return dist[p.x][p.y];
        }

        for (int i = 0; i < 4; i++) {
            int nx = p.x + dx[i];
            int ny = p.y + dy[i];

            if (nx >= 0 && nx < n && ny >= 0 && ny < m &&
                !vis[nx][ny] && maze[nx][ny] != '#') {
                vis[nx][ny] = true;
                dist[nx][ny] = dist[p.x][p.y] + 1;
                q.push({nx, ny});
            }
        }
    }
    return -1; // 不可达
}
```

3. 基础算法

贪心法

核心思想：每一步都选择当前最优解，希望最终得到全局最优解

适用条件

- 最优子结构性质
- 贪心选择性质

时间复杂度：通常为 $O(n)$ 或 $O(n \log n)$

贪心法示例：部分背包问题

问题：n 个物品，第 i 个物品价值 v_i，重量 w_i，背包容量 W，可以取物品的一部分，求最大价值

贪心策略：按单位重量价值 $\frac{v_i}{w_i}$ 从大到小排序

定理

在部分背包问题中，有 n 个物品，每个物品 i 有重量 w_i 和价值 v_i，背包容量为 C。物品可以分割，即可以选择物品的一部分装入背包。最优解是按照物品的"性价比"（单位重量价值 v_i/w_i）从高到低的顺序选择物品。

基础步骤（$n = 1$ 和 $n = 2$）：

- 当 $n = 1$ 时，只有一个物品，显然应该尽可能多地装入该物品，定理成立。

- 当 $n=2$ 时，设两个物品的性价比分别为 $r_1=v_1/w_1$ 和 $r_2=v_2/w_2$，且假设 $r_1 \geq r_2$。

  比较两种选择顺序：

  - 先选物品1，再选物品2：总价值

  $$V_1=\min(w_1, C) \cdot r_1+\max(0, C-w_1) \cdot r_2$$

  - 先选物品2，再选物品1：总价值

  $$V_2=\min(w_2, C) \cdot r_2+\max(0, C-w_2) \cdot r_1$$

  - 差值

  $$V_1-V_2=\min(w_1, C) \cdot r_1-\min(w_2, C) \cdot r_2+\max(0, C-w_1) \cdot r_2-\max(0, C-w_2) \cdot r_1$$

  由于 $r_1 \geq r_2$，可以证明 $V_1 \geq V_2$，即先选性价比高的物品更优。

```
struct Item {
    double v, w; // 价值和重量
    bool operator<(const Item& other) const {
        return v / w > other.v / other.w; // 按单位价值降序
    }
} items[N];

int main() {
    int n;
    double W;
    cin >> n >> W;
    for (int i = 0; i < n; i++) {
        cin >> items[i].v >> items[i].w;
    }

    sort(items, items + n);

    double ans = 0;
    for (int i = 0; i < n && W > 0; i++) {
        if (items[i].w <= W) {
            ans += items[i].v;
            W -= items[i].w;
        } else {
            ans += items[i].v * (W / items[i].w);
            break;
        }
    }

    printf("%.2f\n", ans);
    return 0;
}
```

贪心法示例：排队打水问题

问题：n 个人打水，第 i 个人需要 t_i 时间，求最小总等待时间

贪心策略：按打水时间从小到大排序

总等待时间：$\sum_{i=1}^{n}\left(\sum_{j=1}^{i-1} t_j\right)$

数据归纳法证明

定理

设有 n 个人排队打水，打水时间分别为 $a_1, a_2, \ldots, a_n$。总排队打水时间定义为所有人完成打水的时间之和，即若打水顺序为 $p_1, p_2, \ldots, p_n$，则总时间为

$$T = \sum_{k=1}^{n} \sum_{i=1}^{k} a_{p_i}$$

总时间最小当且仅当打水时间按从小到大排列，即 $a_1 \leq a_2 \leq \cdots \leq a_n$。

取 $a_1 = 8, a_2 = 4, a_3 = 10$，且 $a_2 < a_1 < a_3$。

- 顺序 a_1, a_2, a_3：总时间部分 $V_1 = 2a_1 + a_2 = 2 \times 8 + 4 = 20$。
- 顺序 a_2, a_1, a_3：总时间部分 $V_2 = 2a_2 + a_1 = 2 \times 4 + 8 = 16$。

  比较 $V_2 - V_1 = 16 - 20 = -4 \leq 0$，且 $a_2 - a_1 = 4 - 8 = -4 \leq 0$，成立。
  故顺序 a_2, a_1, a_3 更优，验证了交换逆序对减少总时间。

```cpp
int t[N]; // 打水时间数组

int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> t[i];
    }

    sort(t, t + n); // 按打水时间排序

    long long total_wait = 0;
    long long current_time = 0;

    for (int i = 0; i < n; i++) {
        total_wait += current_time; // 当前人的等待时间
        current_time += t[i];       // 更新当前时间
    }

    cout << total_wait << endl;
    return 0;
}
```

算法复杂度总结

算法类型	时间复杂度	空间复杂度	适用场景
DFS	$O(b^d)$	$O(d)$	所有解、连通性
BFS	$O(V+E)$	$O(V)$	最短路径、层次遍历

算法	平均时间复杂度	最坏时间复杂度	空间复杂度
贪心法	$O(n)$	$O(n \log n)$	$O(1)$