1 Implementação

1.1 Estruturas de Dados

A representação dos grafos utiliza **listas de adjacência**, adequada para grafos esparsos. A estrutura *Union-Find* com *path compression* e *union by rank* é empregada tanto no pré-processamento (agrupamento de pixels) quanto nos algoritmos para detecção e contração de ciclos, garantindo complexidade amortizada $O(\alpha(n))$ por operação.

1.2 Estratégia de Superpixels

A modelagem direta de pixels como vértices ($V = N \times M$) resultaria em complexidade proibitiva. Adotamos uma estratégia de pré-processamento baseada em **Superpixels**: pixels vizinhos com similaridade de cor (distância Euclidiana RGB < 15) são agrupados via *Union-Find*, formando vértices lógicos. Esta técnica reduziu grafos de 10^5 pixels para 10^2–10^3 supernós (redução de 99%), viabilizando execução em tempo real. Um filtro de média (*Box Blur*) é aplicado previamente para reduzir ruído e evitar super-segmentação.

1.3 Algoritmo de Edmonds

O algoritmo clássico de Edmonds (Chu-Liu, 1965) foi implementado como referência, seguindo a formulação recursiva com seleção gulosa, detecção de ciclos e contração de super-nós. Sua complexidade $O(VE)$ o torna inadequado para grafos maiores, motivando a implementação dos métodos otimizados.

1.4 Algoritmo de Tarjan (1977)

O algoritmo de Tarjan melhora a complexidade através do uso de **filas de prioridade mescláveis**. A ideia central é manter, para cada componente do grafo, uma heap contendo todas as arestas de entrada ordenadas por peso.

Fase de Contração: O algoritmo processa vértices iterativamente. Para cada vértice v, seleciona-se a aresta de entrada de menor peso da heap correspondente. Se esta aresta forma um ciclo (detectado via marcação de estados), os componentes do ciclo são contraídos em um super-nó. As heaps dos componentes são *fundidas* (operação *meld*), e os pesos são ajustados via **propagação preguiçosa** (*lazy propagation*): ao invés de atualizar cada aresta individualmente, armazena-se um valor de ajuste no nó raiz da heap, propagado apenas quando necessário.

Remoção de Auto-loops: Durante a seleção da aresta mínima, arestas que se tornaram auto-loops (origem e destino no mesmo componente após contrações) são descartadas da heap.

A complexidade resultante é $O(E \log V)$, onde o fator logarítmico deriva das operações de heap.

1.5 Algoritmo de Gabow, Galil, Spencer e Tarjan (1986)

O algoritmo de Gabow et al. refina a abordagem de Tarjan através da técnica de **Path Growing** (Crescimento de Caminho) e uma estrutura hierárquica para reconstrução da solução.

Path Growing: Ao invés de processar vértices isoladamente, o algoritmo estende "caminhos" a partir de vértices não processados. Cada vértice assume um de três estados: *novo*, *ativo* (em processamento) ou *processado*. Quando um caminho encontra um vértice ativo, um ciclo é detectado e contraído.

Contração Hierárquica: Ciclos são registrados em uma pilha com informações sobre seus componentes e as arestas que os conectam. Cada super-nó mantém referência ao seu "pai" na hierarquia de contrações, permitindo rastrear a qual ciclo original cada vértice pertence.

Expansão: Após processar todo o grafo, a pilha de ciclos é desempilhada em ordem reversa. Para cada ciclo, determina-se qual sub-componente recebe a aresta externa (que entra no super-nó) e quais mantêm suas arestas internas originais. Esta fase reconstrói a arborescência no grafo original.

1.6 Decisão de Projeto: Skew Heaps

Para ambos os algoritmos (Tarjan e Gabow), optou-se por **Skew Heaps** em substituição aos Fibonacci Heaps sugeridos na literatura original. Esta decisão fundamenta-se nos princípios de Engenharia de Algoritmos (Böther et al., 2023): embora Fibonacci Heaps ofereçam complexidade teórica ótima $O(m + n \log n)$, suas constantes ocultas e *overhead* de memória frequentemente resultam em desempenho inferior para instâncias práticas.

As Skew Heaps são árvores binárias auto-ajustáveis que suportam:

- `merge(h1, h2)`: Fusão de duas heaps em $O(\log n)$ amortizado
- `findMin(h)`: Acesso ao mínimo em $O(1)$
- `deleteMin(h)`: Remoção do mínimo em $O(\log n)$ amortizado

A operação de fusão, essencial para contração de ciclos, é implementada recursivamente: compara-se as raízes, e a menor torna-se raiz da heap resultante, com seus filhos trocados (*skew property*) para manter o balanceamento amortizado.

A implementação inclui **lazy propagation** para ajuste de pesos: quando um ciclo é contraído, ao invés de percorrer todas as arestas subtraindo o peso da aresta removida,

armazena-se este valor no campo `lazy` da raiz, propagando-o aos filhos apenas durante acessos subsequentes.

Considerando que nossa estratégia de Superpixels já reduz o grafo para centenas de vértices, a complexidade $O(m \log n)$ das Skew Heaps é virtualmente indistinguível da ótima, garantindo código mais robusto e livre de vazamentos de memória.

1.7 Algoritmo de Kruskal (Baseline)

O algoritmo de Kruskal para MST não-direcionada foi implementado como *baseline* comparativo, utilizando ordenação de arestas e *Union-Find*. Os experimentos demonstraram que métodos direcionados (Tarjan, Gabow) preservam melhor gradientes de cor em transições suaves, validando a escolha da modelagem por arborescência.