Minimum $s-t$ cut of a planar undirected graph in $O(n \log^2 n)$ time

간선에 가중치가 있는 그래프가 주어졌을 때, 최소 $s-t$ 컷은 두 정점 s, t 가 연결되지 않게 하기 위해 지워야 하는 간선 집합의 최소 가중치 합을 뜻한다. Min-cut Max-flow theorem에 의해서, 최소 $s-t$ 컷은 최대 유량 알고리즘을 사용하여 다항 시간에 구할 수 있음이 잘 알려져 있다. 그래프의 최소 컷과 최대 유량의 중요성에 대해서는 이미 여러 번의 SW 멤버십 글을 통해서 언급한 바가 있고, 그 동안 이 주제에 관련된 다양한 글을 써 왔다.

이번에는 평면 그래프에 대해서 최소 $s-t$ 컷을 빠르게 찾을 수 있는 알고리즘을 제시하려고 한다. 일반적인 최대 유량 알고리즘은 수행 시간이 $O(n^3)$ 정도이기 때문에 $s-t$ 컷을 효율적으로 해결할 수 없다. 하지만 Reif는 1983년 평면 그래프에서 $O(n \log^2 n)$ 시간에 최소 $s-t$ 컷을 찾는 알고리즘을 제시했다. 이는 선형에 가까운 알고리즘으로, 굉장히 효율적이라고 할 수 있다. 이 결과는 2010년 $O(n \log \log n)$ 으로 최적화되었지만, 해당 알고리즘도 많은 부분 Reif의 알고리즘에 기반해 있기 때문에.

Reif의 알고리즘은 이론적인 측면에서도 공부해 볼 만 하지만, 이 개념에 관련된 문제는 프로그래밍 대회에서도 몇 번 출제되었기 때문에 (CEOI 2014 Wall, IDTCup 3회 H, Yuhao Du Contest 5 I), 대회에서 사용되는 아주 어려운 알고리즘에 관심이 있다면 한번 배워 보는 것도 좋다고 생각한다.

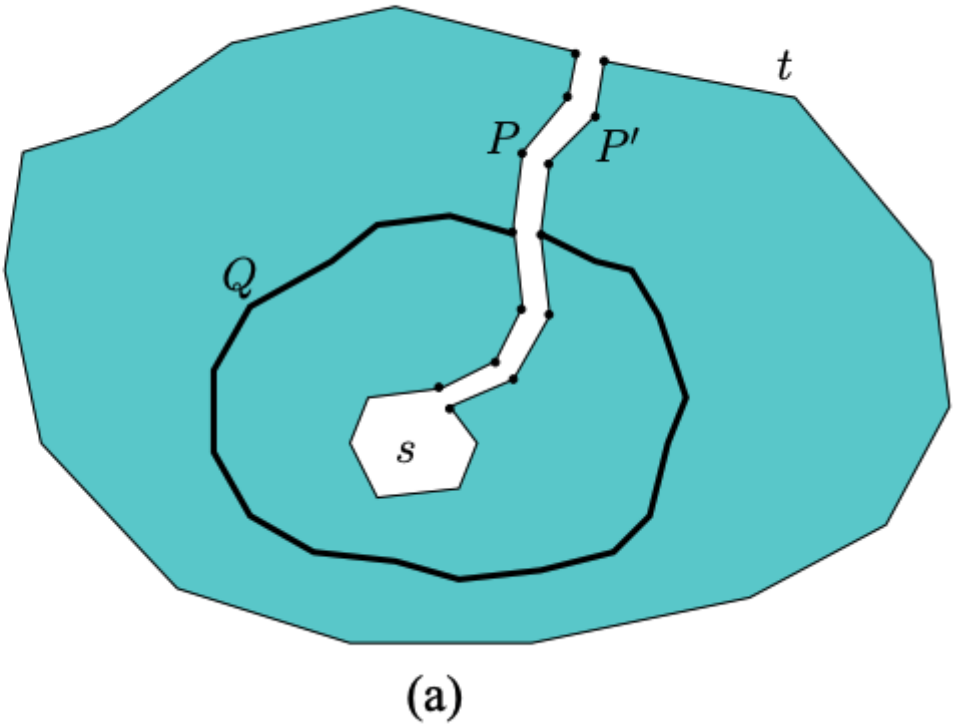
Cut-Open by Shortest Path

Reif의 알고리즘은 평면 그래프의 Cut-Cycle Duality를 적극 활용한다. 평면 그래프에는 Dual Graph가 존재하며, Dual Graph는 원래 평면 그래프 (Primal Graph) 와 동일한 간선 집합을 가진다. 하지만, Dual Graph의 정점은 Primal Graph의 면이고, Primal Graph의 정점은 Dual Graph의 면에 대응된다는 차이를 가진다. 이 때, Cut-Cycle Duality는 원래 그래프의 어떠한 Cut은 Dual Graph의 Cycle에 대응된다는 내용이다.

주어진 평면 그래프의 Dual Graph를 구하면, 우리가 찾는 사이클은 s 의 면을 포함하지만 t의 면을 포함하지 않는 (혹은 그 반대의) 최소 가중치 사이클을 찾는 문제로 전체 문제를 환원할 수 있다. 편의상, s 를 평면 그래프의 원점으로 두고, t 를 포함하지 않으며 s 를 반시계 방향으로 도는 사이클을 찾는다고 가정하자.

s의 면에 인접한 임의의 정점과 t의 면에 인접한 임의의 정점을 잇는 임의의 경로를 그었을 때, 최적 사이클은 무조건 이 경로를 홀수번 *교차*하게 된다. (최적해가 경로 위의 간선을 썼을 경우 교차라는 것의 정의가 애매한데, 일단은 그러한 경우가 없다고 생각하고 직관적으로 이해하자.) 즉, 경로 위에 있는 정점 중 하나를 최적해는 무조건 거치게 된다.

s 의 면과 t 의 면을 **봉합**해보자. s 의 면에 있는 임의의 정점에서, t 의 면에 있는 임의의 정점으로 가는 최단 경로를 구해주자. 이는 s 의 면에 있는 모든 정점을 큐에 넣어주는 식으로 계산할 수 있다. 이제, s 와 t 를 잇는 경로 사이로 그래프를 잘라주자. s 와 t 를 잇는 경로 상의 간선은 Duplicate되고, 경로 상의 정점 역시 Duplicate되며, 이 중 한 정점은 평면 상에서 경로 *위에* 있는 다른 정점과만 연결되어 있고, 다른 정점은 경로 *아래에* 있는 다른 정점과만 연결되어 있다. 이를 그래프를 *cut-open* 했다고 표현한다.



(a)

위 그림은 그래프를 cut-open한 경우의 예시이다. 이와 같이, 시작점과 끝점이 하나의 면에 속하는 것을 알 수 있다.

이 때, 최적 사이클은 항상 해당 최단 경로를 한 번 교차함을 보일 수 있다. 만약 세 번 이상 교차한다면, 그래프 상의 위/아래/위를 거치는 사이클 상의 부분 경로가 존재할 것이다. 위/아래를 넘어가고, 아래/위를 넘어가는 사이클 상의 위치가 2개 있는데, 이 위치는 모두 cut-open된 최단 경로 상에 존재한다. cut-open된 최단 경로로 해당 경로를 바꿔줌으로서 최적성을 해치지 않고, 교차 횟수를 항상 2개 줄일 수 있다.

이제 문제는, 사이클이 거치는 정점 하나를 고정한 후, 해당 정점의 위쪽 copy에서 아래쪽 copy로 가는 최단 경로를 찾으면서 해결할 수 있다. 이 경우, 여러 정점에서 여러 정점으로 가는 경로 중 최소를 구해야 한다. 초기 시작점을 여러 개 넣은 후 (multi-source) Dijkstra's algorithm을 사용하여 $O(n \log n)$ 에 찾으면, 전체 문제가 $O(n^2 \log n)$ 에 풀리게 된다. 이 정도로도 일반적인 플로우 알고리즘보다는 훨씬 빠른 편이다.

이제 이 알고리즘의 문제점이 두 가지가 있는데, 첫 번째로는 여전히 전체 문제를 해결하기에는 효율성이 부족하다는 것이고, 두 번째는 이 Cut-open 과정을 구현하는 것이 굉장히 까다롭다는 것이다. 첫 번째 문제는 아래 단락에서 $O(n \log^2 n)$ 에 줄이는 아이디어를 소개하면서 해결하도록 하고, 잠시 두 번째를 짚고 넘어가도록 하자.

이 알고리즘의 다른 부분인 Dual graph 구성과 최단 경로 계산까지는 잘 알려져 있지만, Cut-open을 구현하는 것은 상당히 까다롭다. Primal graph의 경우에는 좌표가 있어서 명확히 비교할 것이 있지만, Dual graph를 그렇게 만드는 것은 매우 어려워서 "상대적인" 방향 차이에 모든 것을 의존하고, 그러면서도 Cut-open이라는 복잡한 그래프 변환 연산을 해야 한다는 것이 구현의 복잡성에 기여한다. 특히 Planar min cut의 경우 이해하기 쉬운 간단한 구현이 없어서 이 부분에 애를 많이 먹었다. 일단, Dual Graph의 Face를 생각할 필요 자체는 없고, 각 Face에 대해서 인접한 Face가 무엇인지만 Cyclic하게 잘 관리하면 된다. 이는 원래 Planar graph의 Dual을 구성하는 과정에서 약간의 신경을 쓰면 가능하다. 이러한 Cyclic한 리스트가 있다면, 경로가 이 Face를 어떠한 점을 기준으로 Cut-open하는 지는 자명하게 구분할 수 있다. 이를 구현할 때는 시작점, 끝점에 대한 Case work를 포함해서 정말 신중하게 구현해야 하고, 간선을 지우거나 변경하는 등 인접 리스트를 고치는 연산이 필요하다. 첫 번째는 그냥 열심히 하면 되고, 두 번째는 적당한 자료 구조를 사용하여 처리할 수 있는데, 아무리 많아야 `std::set` 정도이기 때문에 이 글에서는 설명을 생략한다.

Divide and Conquer

Dual graph를 Cut-open한 이후에는, 최단 경로를 최대 $O(N)$ 개의 시작/끝점 쌍에 대해 계산해 주면 된다. Cut-open 과정은 cyclic하게 "말려 있는" 평면 그래프를 직사각형 형태로 펴주는 과정이라고 생각할 수 있다. 이 때 경로의 시작점 $s_1, s_2, \ldots, s_k$ 는 직사각형의 upper path에 순서대로 나열되어 있고, 경로의 끝점 $e_1, e_2, \ldots, e_k$ 는 직사각형의 lower path에 순서대로 나열되어 있다. 고로, 경로간의 일종의 단조성이 성립한다고 생각할 수 있다. 정확히 말하자면, 다음과 같은 정리가 성립한다.

- **Lemma.** $s_i \to e_i$ 를 잇는 최단 경로를 L 이라고 하고, 평면 그래프를 L 을 기준으로 왼쪽/오른쪽으로 나누자. (L 은 두 조각 모두에 포함된다.) $j < i$를 만족하는 $s_j \to e_j$ 간의 최단 경로는 모두 왼쪽 조각에 완전히 속하고, $j > i$ 를 만족하는 $s_j \to e_j$ 간의 최단 경로는 모두 오른쪽 조각에 완전히 속한다.
- **Proof.** 만약 한쪽 조각을 넘어간다면, 해당 부분을 L 으로 대체할 경우 항상 경로의 길이를 늘리지 않으면서 넘어가는 부분을 줄일 수 있다.

이를 활용하여 분할 정복을 시도하자. 기저 조건은 $k = 1$ 인 경우로, 이 경우는 문제를 한 번의 다익스트라로 해결할 수 있으니 넘어가자. Dual에서 $s_{k/2} \to e_{k/2}$ 를 잇는 최단 경로 L 를 찾아주자. 이 최단 경로 L 을 Primal로 변환하였을 경우, 이는 s, t 를 자르는 하나의 컷이 된다. 이제 다른 컷은 이 컷의 왼쪽에만 존재하거나, 오른쪽에만 존재하게 된다.

그래프를 작은 두 조각으로 분할하여, 나머지 두 조각에서 최소 컷을 찾았을 때의 결과와, L 이 만드는 최소 컷을 찾았을 때의 결과를 합치면 전체 답이 되도록 하자. L 을 기준으로 컷이 생기기 때문에, 해당 컷의 "왼쪽" 과 "오른쪽" 에 속하는 정점들을 DFS를 사용하여 라벨링할 수 있다. 이제 L 에 속한 간선들을 일단 지워준 후, 왼쪽 조각에는 t' 이라는 새로운 싱크, 오른쪽 조각에는 s' 이라는 새로운 소스를 만들어 주자. 이 두 정점들을 잇는 간선들은 L 을 통하여 구성한다. L 에 두 정점 u, v 를 잇는 가중치 $c(u, v)$ 의 간선이 있었다면, 왼쪽 조각에 u, t' 을 잇는 가중치 $c(u, v)$ 의 간선, 오른쪽 조각에 s', v 를 잇는 가중치 $c(u, v)$ 의 간선을 만들어 주는 것이다. 마지막으로, 만약에 두 정점을 잇는 중복 간선이 존재한다면 이들을 하나의 간선으로 합쳐준다 (가중치는 합으로 적용한다).

결론적으로, 알고리즘은 다음과 같이 정리된다.

- $s \to t$ 를 기준으로 그래프를 Cut-open한다.
- 만약 Cut-open된 Path가 길면 그 자리에서 종료한다. 그렇지 않으면 해당 Path의 중간을 기준으로 한 최소 컷을 Dijkstra로 찾아준다.
- 최소 컷을 기준으로 그래프를 쪼갠 후, L 의 간선을 적당히 추가해서 작은 크기의 Planar graph를 만든다.
- 재귀적으로 반복한다.

이 알고리즘의 정당성은 위 Lemma를 사용하여 자명하게 볼 수 있다. 여기서 자명하지 않은 점은 이 알고리즘의 시간 복잡도이다. 단순히 보았을 경우 그래프가 두 조각으로 분할되기 때문에 대략 $O(n \log n)$, 거기에 Dijkstra를 곱해서 $O(n \log^2 n)$ 이 나온다고 생각할 수 있지만, 두 가지 문제점이 있다.

- 그래프가 균일하게 분할되지 않아서 깊이가 매우 깊어질 수 있다.
- 간선을 추가하기 때문에 복잡도가 성립하는지 자체가 의문일 수 있다.

일단 첫 번째는 사실이 아니며, 그래프는 맨 처음 Cut-open되었을 때의 최단 경로의 길이를 d 라고 하였을 때, $\log d$ 번 이상의 재귀 콜을 거치지 않는다. 이는 작은 크기의 Planar graph를 만들 때, 그래프의 중간에서 가중치가 더 낮은 최단 경로를 만들지 않는 식으로 construction이 진행되기 때문이다. 고로, 각각의 작은 크기의 Planar graph는 원래 그래프의 Cut-open을 보존하고 있는다. 이 값이 보존된다면, 최단 경로 역시 원하는 대로 절반으로 잘라졌을 것이다.

두 번째 부분이 중요한데, 여기서 우리는 분할 정복을 간선 기준이 아니라 정점 기준으로 한다는 점을 생각할 수 있다. 일단 앞서 과정에서 확인한 것처럼, 우리는 평면 그래프에 다중 간선 (루프) 를 허용하지 않게끔 분할 정복을 진행한다. 오일러 정리에

의하면, 단순한 평면 그래프에서는 $e \leq 3v - 6$ 이 성립한다. 즉, Dijkstra를 돌리는 데 드는 시간은 $O(v \log v)$ 라고 생각할 수 있다는 것이다. 이렇게 보았을 때, L 의 간선을 적당히 추가하는 과정에서 우리가 굉장히 많은 간선들을 재활용하고 있다고 생각할 수 있지만, 이를 정점의 기준으로 보았을 때, 한 분할에서 새롭게 만드는 정점의 개수는 단 2개이다. 즉,

$T(n, d) = T(a, d-1) + T(n+2-a, d-1) + O(n \log n)$

이 성립하며, $T(n, \log n)$ 이 우리의 알고리즘의 시간 복잡도이다. 이 때, 각 Depth에서의 Recursion call에서 처리하는 정점 개수를 더하면, 맨 위 레이어에서 $n+2$, 그 다음 $n+4, n+8, \ldots, n+2^d$ 와 같이 곱이 아니라 더해지는 형태로 정점 개수가 증가함을 알 수 있다. 이는 각각의 Recursion call에서 정점을 2개씩 만들어 주기 때문이다. 이에 따라서 시간 복잡도는

$T(n, \log n) = O(n \log^2 n)$

이 되며, 전체 알고리즘이 증명된다.

Problems with similar ideas

마지막으로, 이러한 식의 아이디어를 활용한 다른 문제들의 예시를 몇 가지 살펴봄으로써 이 내용을 Competitive programming에 어떻게 적용시킬 수 있는지 살펴보자.

CEOI 2014. Wall

평면 그래프가 주어지고 특정한 Face들이 선택되었을 때, 선택한 Face를 모두 포함하는 최소 길이의 simple cycle을 찾는 문제이다. 이 문제에서는 선택된 Face 중 하나가 outer face와 adjacent함이 보장된다.

이 문제는 위 알고리즘의 첫번째 아이디어 (Cut-open) 과 유사한 접근을 활용할 수 있다. 주어진 평면 그래프의 Dual을 취하면, 선택된 여러 개의 정점과 Outer face 정점을 분리하는 Cut을 찾는 문제가 된다. 선택된 정점 중, Outer face와 인접한 정점 v 을 기준으로 Shortest path tree를 구성하자. v 가 아닌 모든 선택된 정점들에 대해서, $v \to w$ 로 가는 최단 경로를 계산할 수 있고 그에 속하는 간선 집합 역시 구할 수 있다. Lemma는, 이 간선 집합과 Cut의 교집합의 원소가 단 하나 (Outer face 정점과 v 정점을 잇는 간선) 뿐이라는 것이다. 고로, 해당 원소를 제외한 후, Dual graph에서 Cut을 찾으면 된다. 이는 Primal graph에서의 Simple cycle과 같고, Cycle의 한 원소가 고정되었으니 이는 Path가 되어서, Dijkstra's algorithm으로 해결할 수 있다.

일반적인 경우에는 교집합으로 가능한 원소가, Outer face 정점과 v 를 잇는 경로의 구간으로 나올 것이라고 생각한다. 이 경우 두 정점을 잇는 최단 경로를 계산한 후 Cut-open 하여 동일한 풀이를 사용할 수 있지 않을까 생각한다. 다만 이 경우 문제가 훨씬 복잡해져서, 대회에서는 간단한 형태 정도로도 충분히 만점을 받을 수 있게 문제가 출제된 것으로 보인다.

Yuhao Du Contest 5 I

이 문제는 상당히 어려운 문제라서 모든 부분을 설명할 수는 없고, 또한 대다수의 내용이 이미 이전 글 에서 다뤄졌기 때문에 굳이 그럴 필요도 없어 보인다. 간단히 말해, $O(n^2 \log n)$ 에서 $O(n \log^2 n)$ 으로 넘어가는 부분은, 위에서 언급했던 것과 같이 Planar graph에서 최단 경로를 찾는 문제로 볼 수 있다. 이 문제 역시 DP의 상태 전이 그래프를 평면 그래프로 보았을 경우, 시작점과 끝점이 단조성을 띈다. 이는 앞의 Divide-and-Conquer 단락에서 설명한 최단 경로의 단조성 Lemma를 그대로 적용할 수 있다는 뜻이다. 다만, 여기서는 Planar graph의 크기가 "아주" 크기 때문에 (간선을 모두 나열할 수 없는 정도) 이 문제에서 했던 것과 같이 그래프를 explicit하게 변경하는 것은 비효율적이고, 이를 몇가지 아이디어를 추가적으로 더 사용하여 해결해야 한다.