Árvores Geradoras Mínimas

prof. Fábio Luiz Usberti

MC521 - Desafios de Programação I

Instituto de Computação - UNICAMP - 2018

Sumário

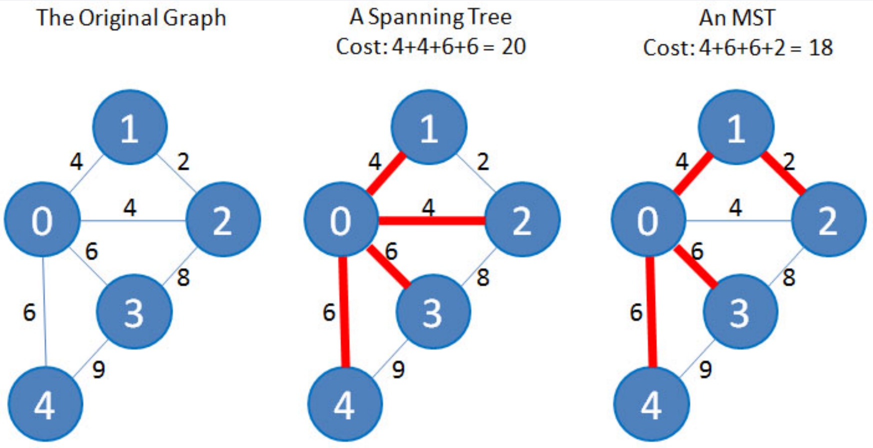
Introdução

Árvores Geradoras Mínimas (AGM)

- Problema da árvore geradora mínima (AGM): Dado um grafo não-orientado conexo $G(V, E)$, com custos nas arestas, determine um subgrafo conexo acíclico (árvore) que incide sobre todos os vértices (geradora) com menor custo possível (mínima).
- Aplicações:

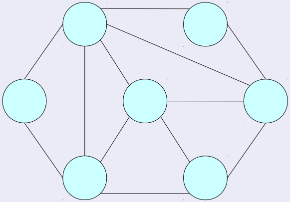
1. Contruir a malha viária para um conjunto de cidades dispersas geograficamente;
2. Configurar uma rede de computadores utilizando o menor comprimento de cabo;
3. Conectar um conjunto de consumidores elétricos minimizando as perdas de transmissão de energia.

Árvores Geradoras Mínimas (AGM)



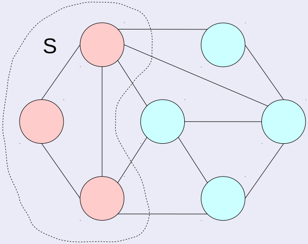
Árvores Geradoras Mínimas (AGM)

- Uma AGM pode ser obtida por algoritmos eficientes (polinomiais), dentre os quais é possível citar dois algoritmos gulosos, reconhecidos da literatura: o algoritmo de Kruskal e o algoritmo de Prim.
- Corte: Dado um subconjunto de vértices $S \subset V$, define-se $\delta(S)$ como corte de S o conjunto das arestas com apenas uma extremidade incidente em S.



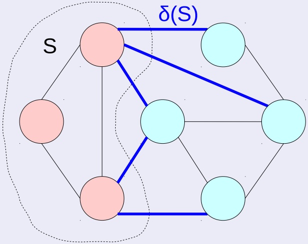
Árvores Geradoras Mínimas (AGM)

- Uma AGM pode ser obtida por algoritmos eficientes (polinomiais), dentre os quais é possível citar dois algoritmos gulosos, reconhecidos da literatura: o algoritmo de Kruskal e o algoritmo de Prim.
- Corte: Dado um subconjunto de vértices $S \subset V$, define-se $\delta(S)$ como corte de S o conjunto das arestas com apenas uma extremidade incidente em S.



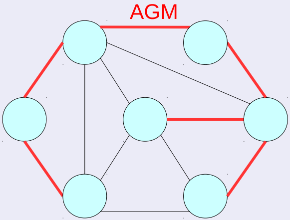
Árvores Geradoras Mínimas (AGM)

- Uma AGM pode ser obtida por algoritmos eficientes (polinomiais), dentre os quais é possível citar dois algoritmos gulosos, reconhecidos da literatura: o algoritmo de Kruskal e o algoritmo de Prim.
- Corte: Dado um subconjunto de vértices $S \subset V$, define-se $\delta(S)$ como corte de S o conjunto das arestas com apenas uma extremidade incidente em S.



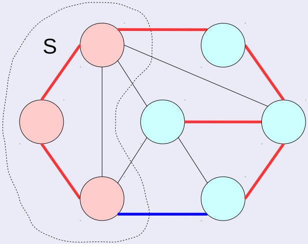
Árvores Geradoras Mínimas (AGM)

- Propriedade do Corte: Para qualquer subconjunto de vértices $S \subset V$ de um grafo, se o custo de uma aresta e do corte $\delta(S)$ for menor do que o custo das demais arestas do corte, então a aresta e pertence a uma árvore geradora mínima.
- Prova (por contradição): Seja e a aresta de menor peso de um corte $\delta(S)$. Assuma que existe uma AGM T que não contém a aresta e. Adicionando a aresta e em T produzirá um ciclo que atravessa o corte por e e retorna pelo corte por outra aresta e'. Removendo a aresta e' resulta em uma árvore geradora com custo estritamente menor que T (contradição).



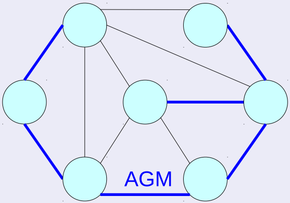
Árvores Geradoras Mínimas (AGM)

- Propriedade do Corte: Para qualquer subconjunto de vértices $S \subset V$ de um grafo, se o custo de uma aresta e do corte $\delta(S)$ for menor do que o custo das demais arestas do corte, então a aresta e pertence a uma árvore geradora mínima.
- Prova (por contradição): Seja e a aresta de menor peso de um corte $\delta(S)$. Assuma que existe uma AGM T que não contém a aresta e. Adicionando a aresta e em T produzirá um ciclo que atravessa o corte por e e retorna pelo corte por outra aresta e'. Removendo a aresta e' resulta em uma árvore geradora com custo estritamente menor que T (contradição).



Árvores Geradoras Mínimas (AGM)

- Propriedade do Corte: Para qualquer subconjunto de vértices $S \subset V$ de um grafo, se o custo de uma aresta e do corte $\delta(S)$ for menor do que o custo das demais arestas do corte, então a aresta e pertence a uma árvore geradora mínima.
- Prova (por contradição): Seja e a aresta de menor peso de um corte $\delta(S)$. Assuma que existe uma AGM T que não contém a aresta e. Adicionando a aresta e em T produzirá um ciclo que atravessa o corte por e e retorna pelo corte por outra aresta e'. Removendo a aresta e' resulta em uma árvore geradora com custo estritamente menor que T (contradição).



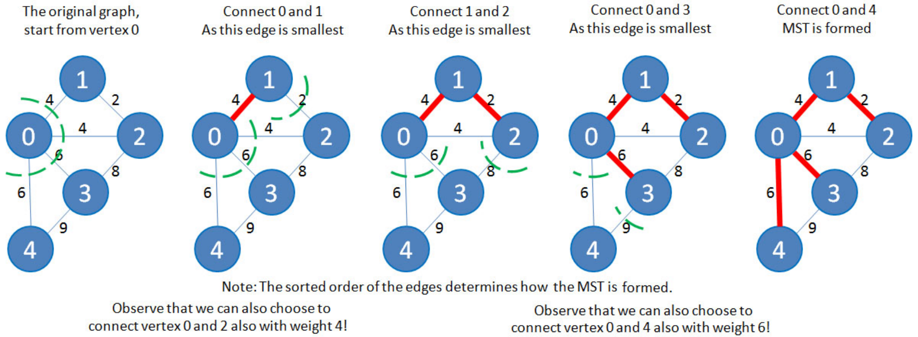
Algoritmo de Prim

Estratégia Gulosa

- Algoritmo proposto por Roberto C. Prim em 1957.
- Estratégia gulosa: Inicialmente, escolha um nó arbitrário u e insira na árvore a aresta (u, v) de menor custo no corte $\delta(\{u\})$. Em seguida, iterativamente insira na árvore a aresta com o menor custo que se encontra no corte dos vértices que já estão na árvore. Repita até que a árvore seja geradora.
- A seleção da aresta de menor custo do corte pode ser implementado a partir de um heap: a cada nó u incluído na árvore, são inseridas no heap todas as arestas (u, v) contanto que v não esteja na árvore.
- A complexidade do algoritmo também é dada por $O(|E| \log |V|)$: as arestas são avaliadas pelo algoritmo ($O(|E|)$) e algumas são inseridas no heap ($O(\log |V|)$).

Algoritmo de Prim

Exemplo



Algoritmo de Prim

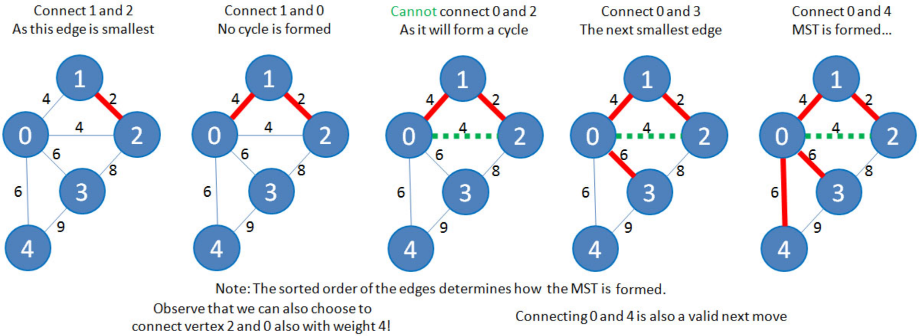
Código

```
vi taken;                                   // global boolean flag to avoid cycle
priority_queue<ii> pq;             // priority queue to help choose shorter edges

void process(int vtx) {     // so, we use -ve sign to reverse the sort order
  taken[vtx] = 1;
  for (int j = 0; j < (int)AdjList[vtx].size(); j++) {
    ii v = AdjList[vtx][j];
    if (!taken[v.first]) pq.push(ii(-v.second, -v.first));
} }                               // sort by (inc) weight then by (inc) id

// inside int main() --- assume the graph is stored in AdjList, pq is empty
  taken.assign(V, 0);                  // no vertex is taken at the beginning
  process(0);   // take vertex 0 and process all edges incident to vertex 0
  mst_cost = 0;
  while (!pq.empty()) {  // repeat until V vertices (E=V-1 edges) are taken
    ii front = pq.top(); pq.pop();
    u = -front.second, w = -front.first;  // negate the id and weight again
    if (!taken[u])                    // we have not connected this vertex yet
      mst_cost += w, process(u); // take u, process all edges incident to u
  }                                       // each edge is in pq only once!
  printf("MST cost = %d (Prim's)\n", mst_cost);
```

Estratégia Gulosa

- Algoritmo proposto por Joseph B. Kruskal em 1956.
- Estratégia gulosa: As arestas, em ordem não-decrescente, são inseridas uma a uma na árvore, contanto que não formem um ciclo. Repita até que a árvore seja geradora.
- Avaliar se a inserção de uma aresta forma ciclo com a solução parcial pode ser executado em tempo quase-constante $O(\alpha(|V|))$ utilizando a estrutura de conjuntos disjuntos Union-Find.
- A complexidade do algoritmo de Kruskal é limitada por $O(|E|\log|E| + |E|\alpha(|V|)) = O(|E|\log|V|)$.

Exemplo


Connect 1 and 2
As this edge is smallest
Connect 1 and 0
No cycle is formed
Cannot connect 0 and 2
As it will form a cycle
Connect 0 and 3
The next smallest edge
Connect 0 and 4
MST is formed...
Note: The sorted order of the edges determines how the MST is formed.
Observe that we can also choose to
connect vertex 2 and 0 also with weight 4!
Connecting 0 and 4 is also a valid next move

Código

```
// inside int main()

  sort(EdgeList.begin(), EdgeList.end()); // sort by edge weight O(E log E)
                    // note: pair object has built-in comparison function

  int mst_cost = 0;
  UnionFind UF(V);                     // all V are disjoint sets initially
  for (int i = 0; i < E; i++) {                    // for each edge, O(E)
    pair<int, ii> front = EdgeList[i];
    if (!UF.isSameSet(front.second.first, front.second.second)) {  // check
      mst_cost += front.first;                   // add the weight of e to MST
      UF.unionSet(front.second.first, front.second.second);    // link them
  } }                          // note: the runtime cost of UFDS is very light

  // note: the number of disjoint sets must eventually be 1 for a valid MST
  printf("MST cost = %d (Kruskal's)\n", mst_cost);
```

Aplicações

Problemas relacionados

- Árvores geradoras de custo máximo.
- Subgrafo gerador de custo mínimo.
- Floresta geradora de custo mínimo.
- Segunda melhor AGM
- Problema do caminho minimax

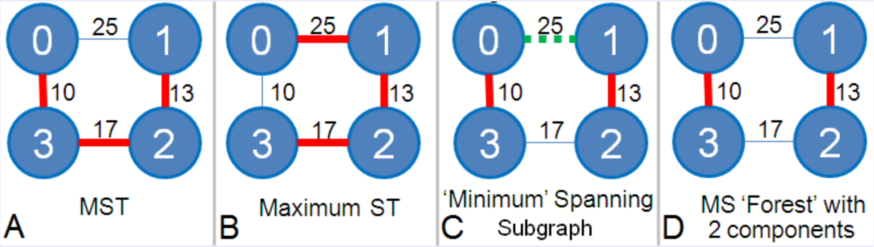
Aplicações

Árvores Geradoras de Custo Máximo

- Problema UVa 1234 – Racing pede o valor da árvore geradora de custo máximo.
- A solução desse problema envolve simplesmente ordenar as arestas de modo não-descrescente para o algoritmo de Kruskal, ou utilizar um heap de máximo para o algoritmo de Prim.

Árvores Geradoras de Custo Mínimo

Aplicações

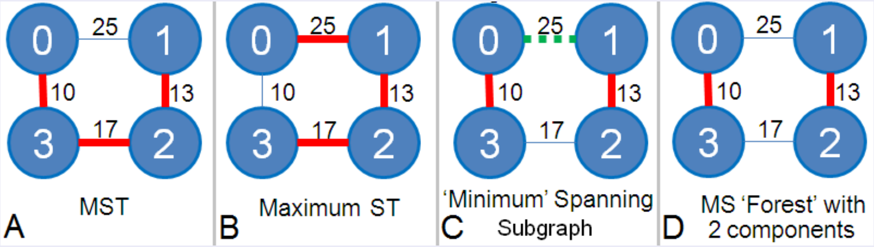


Subgrafo Gerador de Custo Mínimo

- O problema do subgrafo gerador de custo mínimo consiste em aumentar um subgrafo inicial, adicionando-lhe arestas, até que o subgrafo resultante seja gerador.
- Não necessariamente o subgrafo inicial é uma árvore e pelo mesmo motivo o subgrafo gerador de custo mínimo também não será necessariamente uma árvore.
- Para resolver esse problema, inicializar a estrutura Union-Find formando conjuntos disjuntos para cada componente conexa do subgrafo inicial. Em seguida, executa-se o algoritmo de Kruskal para as arestas remanescentes do grafo.

Árvores Geradoras de Custo Mínimo

Aplicações



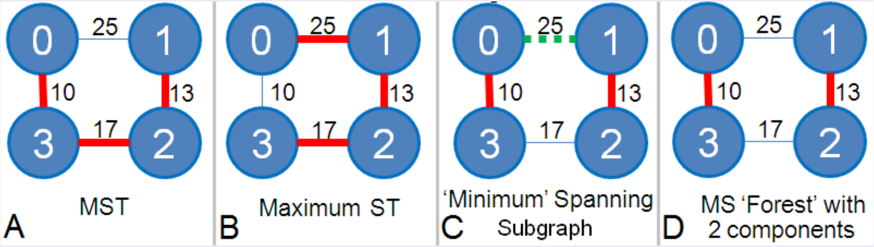
Aplicações

Problema da Floresta Geradora de Custo Mínimo

- Nesta variante, deseja-se formar uma floresta com K componentes conexas (ou K sub-árvores) que possua o menor custo, dado um parâmetro de entrada $1 \leqslant K \leqslant |V|$ (vide Problema UVa 10369 – Arctic Networks).
- A solução desse problema é possível executando o algoritmo de Kruskal até que $|V| - K$ arestas sejam incluídas na solução.

Árvores Geradoras de Custo Mínimo

Aplicações



Aplicações

Problema da Segunda Melhor AGM

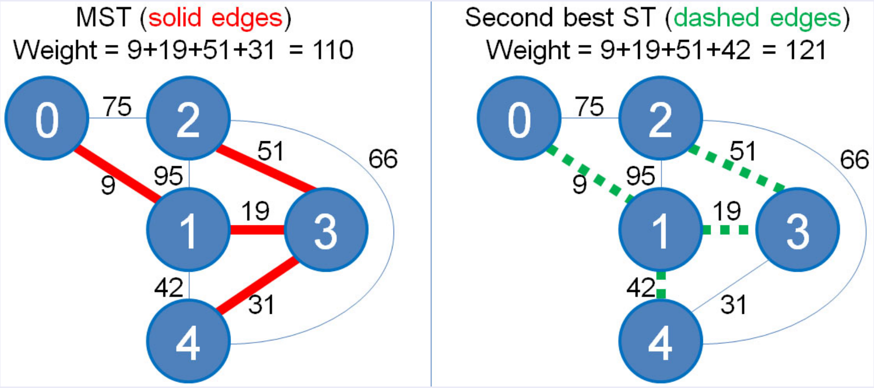
- Em certas ocasiões, soluções alternativas à solução ótima também são relevantes. Por exemplo, quando por alguma limitação técnica a solução ótima não puder ser implementada (vide Problema UVa 10600 – ACM contest and blackout).
- Uma variante consiste em obter a segunda melhor AGM, ou seja, a segunda árvore geradora mais barata.
- Propriedade: a melhor AGM e a segunda melhor AGM diferem de uma única aresta.

Aplicações

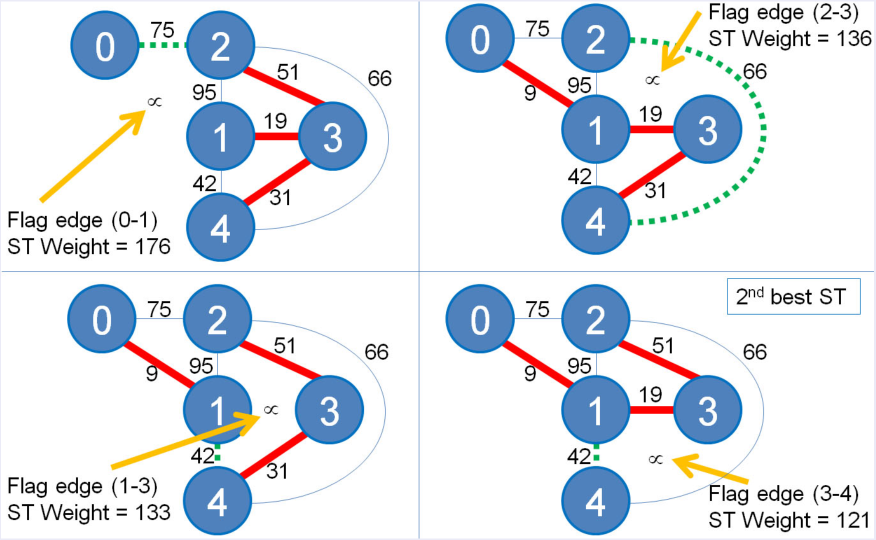
Problema da Segunda Melhor AGM

- Uma solução para esse problema consiste em adaptar o algoritmo de Kruskal, da seguinte forma:
- Primeiro passo consiste em obter uma AGM. Para cada aresta e da AGM, executa-se o Kruskal desde o início removendo-se contudo a aresta e da lista. A segunda melhor AGM será aquela de menor custo dentre as árvores resultantes do Kruskal após a proibição de uma aresta da AGM.
- A complexidade do algoritmo torna-se $O(\alpha(|V|)|V||E|)$.
- Há uma forma mais eficiente $O(|E|\log(|V|))$ de resolver o problema utilizando o algoritmo do menor ancestral comum (LCA – lowest common ancestor).

Problema da Segunda Melhor AGM



Problema da Segunda Melhor AGM



Aplicações

Problema do Caminho Minimax

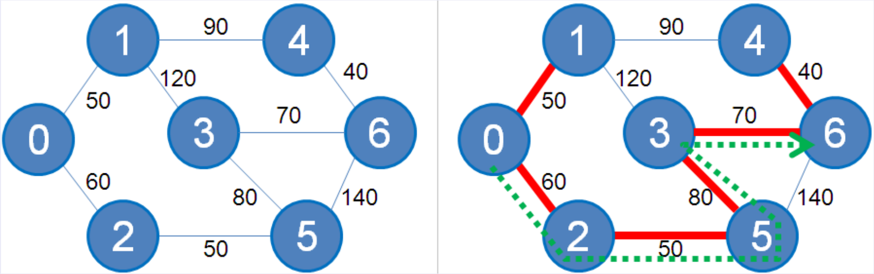
- O problema do caminho minimax consiste em encontrar um caminho de um nó i até um nó j cuja aresta de maior custo no caminho seja minimizada. Em outras palavras, o custo de um caminho é definido pelo custo de sua aresta mais cara; deseja-se encontrar o caminho de menor custo de i para j.
- O problema do caminho minimax para dois nós i e j pode ser modelado como um problema de AGM.

Aplicações

Problema do Caminho Minimax

- Propriedade: o caminho entre dois nós i e j de uma AGM é um caminho minimax.
- Para obter o custo da maior aresta de um caminho minimax, basta realizar um percurso na AGM.
- A complexidade do algoritmo para o problema do caminho minimax passa a ser $O(|E| \log |V|)$.

Introdução


1
90
4
120
40
50
0
3
70
6
60
80
140
2
50
5
1
90
4
120
40
50
0
3
70
6
60
80
140
2
50
5

Referências

1. S. Halim e F. Halim. Competitive Programming 2, Second Edition Lulu (www.lulu.com), 2011. (IMECC – 005.1 H139c)
2. S. S. Skiena, M. A. Revilla. Programming Challenges: The Programming Contest Training Manual, Springer, 2003.
3. T.H. Cormen, C.E. Leiserson, R.L.Rivest e C. Stein. Introduction to Algorithms. 2nd Edition, McGraw-Hill, 2001. (no. chamada IMECC – 005.133 Ar64j 3.ed.)
4. U. Manber. Introduction to Algorithms: A Creative Approach. Addison-Wesley. 1989. (no. chamada IMECC – 005.133 Ec53t 2.ed.)