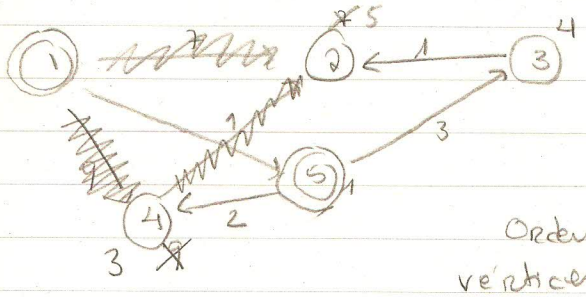
ADA 28-5-2012

combo box

EXAME junho 2009

(1)

Quando já temos um custo definido para um nó, e encontramos um caminho com custo menor → DECREASE KEY



Ordem de seleção de vértices: 1, 5, 4, 3, 2

Número de invocações de decreaseKey: 3

Via do 2 é o 3

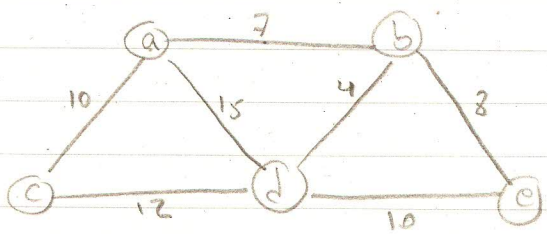
Qual o vértice que usei para chegar ao 2.

via	I	1°	2°	3°	4°
1	1	1	1	1	1
2	–	1	1	4	3
3	–	–	5	5	5
4	–	1	5	5	5
5	–	1	1	1	1

lenght	I	1°	2°	3°	4°
1	0	0	0	0	0
2	7	7	7	6	5
3	+∞	+∞	4	4	4
4	9	9	3	3	3
5	1	1	1	1	1

(2) Grupo não orientado, pesado e conexo

Exemplo:



Árvore mínima de cobertura

```
int maxSTKruskal(UndiGraph<?, E> graph)
    Hea PriorityQueue<E, Edge<?, E> allE dges=
                                buildEdgesPQ(graph)
    UnionFind vertP: new UnionFindanArray(graph, numberVertices());
    int maxSTCurrentSize=0;
    int maxSTFindSize = graph.numVertices() - 1;
    int totalCapacity=0;
```

```
while (maxStCurrentSize < maxStFinalSize){
    Edge <?, E> edge = allEdges.removeMax().getValue();
    Vertex <?> [] endPoints = edge.endVertices();
    int value = edge.label();
    int rep1 = vertP.find(endPoints[0]);
    int rep2 = vertP.find(endPoints[1]);
    if(rep1 != rep2){
        totalCapacity += value;
        maxStCurrentSize++;
        vertP.union(rep1, rep2);
    }
}
return totalCapacity;
```

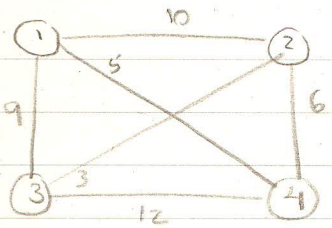
Análise da complexidade Temporal (no pior caso)

- União por dimensão e representada com compressão de caminho
- Criar fila de Prioridade $O(\#A)$
- Criar partição $O(\#V)$
- Instruções de ciclo executadas entre #V-1 e #A vezes
  - Remover Máximo $O(\log \#A)$
  - Seleccionar 2 representantes $O(\log \#V)$
- Instruções do ciclo executadas #V-1 vezes
  - Unir representantes $O(1)$

$O(\#A + \#V + \#A(\log \#A + \log \#V) + \#V)$
$O(\#A(\log \#A + \log \#V))$
$O(\#A \log \#A + \#A \log \#V)$
$O(\#A \log \#V + \#A \log \#V)$ $\longrightarrow \log A \leq \log V^2$
$O(\#A \log \#V)$

optimus

(3)



Problema similar:
HAMILTON

Para provar que um problema X é NP-completo é necessário mostrar que:
1) O problema X é NP (3 val)
2) O problema X é NP-Completo (1 val)

Circuito de Hamilton limitado (CHL) por N em G
é um circuito de Hamilton em G que passa apenas por arcos de custo inferior ou igual a N.

1) CHL é um problema NP ✓
   a) Existe um algoritmo polinomial que dados:
      - Um grafo $G=(V,A)$
      - Um inteiro N
      - permutação $p = v_1, \ldots, v_n$ dos vértices de G.

   verifica se p é um CHL por N em G.

O algoritmo percorre a permutação testando se os arcos (v_i, v_{i+1}) pertence a A e o seu peso é menor ou igual a N.
Depois faz o mesmo teste para o arco (v_n, v_1).
O algoritmo devolve true sse todos os testes tiverem sucesso.

b) Se o conjunto de arcos estiver num vector, o número de passos é $O(n * \#A) = O(\#V . \#A)$

c) A dimensão de p é polinomial na dimensão de G, porque p tem #V vértices.

optimus

2) CHL é NP-Completo

Se A é NP-completo

B é NP

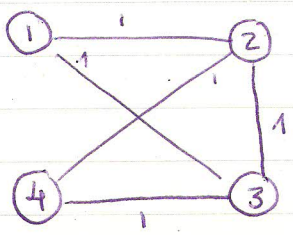
existe uma redução polinomial $\phi : A \to B$

Então B é NP-Completo

ϕ : HAMILTON $\longmapsto$ CHL

$(V, A) \longmapsto ((V, A^*), 2)$

$A^* = \{(a,b,1) \mid (a,b) \in A\}$



(4) $f(i,j) = \begin{cases} 1, & \text{se } i=1 \text{ e } j \geq 3 \\ 2f(i-1,j)+1, & \text{se } i \geq 2 \text{ e } j=3 \\ \min_{1 \leq k < i} \left(2f(k,j) + f(i-k, j-1)\right), & \text{se } i \geq 2 \text{ e } j \geq 4 \end{cases}$

```
int func (int D, int E){
    int [][] F = new int [D+1][E+1];
    for(int j=3; j<= E ; j++)
        F[1][j] = 1
    for(int i=2; i<= D ; i++)
        F[i][3] = 2 * F[i-1][3] +1;
    for(int i=2 ; i <= D ; i++)
        for(int j = 4 ; j <= E ; j++) {
            F[i][j] = +∞;
            for(int k=1 ; k < i ; k++){
                int value = 2 * F[k][j] + F[i-k][j-1];
                if (F[i][j] > value){
                    F[i][j] = value;
                }
            }
        }
    return F[D][E];
}
```

Complexidade Espacial (em todos os casos)

$O((D+1)*(E+1)) = O(D*E)$

Complexidade Temporal (em todos os casos)

1º ciclo $O(E-2) = O(E)$

2º ciclo $O(D-1) = O(D)$

3º ciclo $O((D-1)*(E-1)*D) = \underline{O(D^2*E)}$

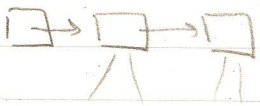
29-05-2012

(5) Filas Binomiais — Recebe uma fila e diz-se quantos elementos é que tem.

Binomial ≠ Fibonnaci

↓ apontador p/ head, até chegar a um null

↓ circular, pode apontar apenas até ao mínimo caso o user queira..



```
Public int size(){
    BinNode<K,V> curTree = head;
    int countNodes = 0
    while(curTree != null){
        countNodes += Math.pow(2, curTree.getDegree());
        curTree = curTree.getRightSibling();
    }
    return countNodes;
}
```

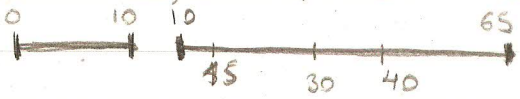
→ Em Fibonnacci era comparar com o nó mínimo de forma a saber se já tinha dado a volta completa

Complexidade temporal: $O(2\log n + 1) = O(\log n)$

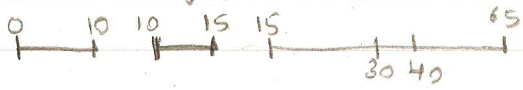
n é o número de nós da pilha

optimus

(6) 1º corte) Posição $p_1 = 10$ (custo = 65)



2º corte) Posição $p_2 = 15$ (custo = 55)

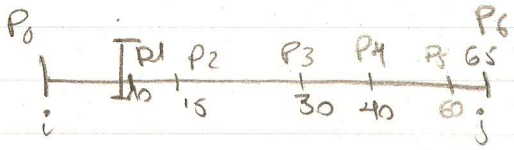


Custo total = 230

Custo mínimo = 170

Caso Base P_0 — P_{n+1}

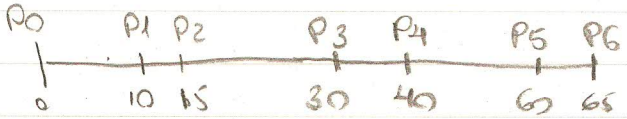
↳ Se não houver cortes a fazer o custo é zero.



$K = j \quad F(i,k) + F(k,j) + (P_j - P_i)$

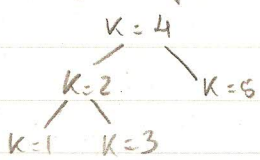
$K = 2$

$$F(i,j) = \begin{cases} \emptyset, & \text{se } i = j-1 \\ \min\limits_{i<k<j} \left(F(i,k) + F(k,j)\right) + (P_j - P_i), & \text{se } i < j-1 \end{cases}$$



$$F(0,6) = \min_{0<k<6} \begin{cases} k=1 & F(0,1) + F(1,6) \\ k=2 & F(0,2) + F(2,6) \\ k=3 & F(0,3) + F(3,6) \\ \boxed{k=4} & F(0,4) + F(4,6) \\ k=5 & F(0,5) + F(5,6) \end{cases}$$

Solução óptima



Chamada inicial: $F(0, n+1)$