Admin

Assignment 9

Assignment 10 (2 weeks assignment) – don’t ignore until next week

Checkpoint 2 next Monday

Checkpoint 2

2 pages of notes

2/20 through 4/10 (will not include network flow)

Will make some practice problems available later this week

Checkpoint 2 topics

- greedy algorithms
- proving correctness
- developing algorithms
- comparing vs. dynamic programming
- hash tables
  - collision resolution by chaining
  - open addressing
  - hash functions
- Dynamic programming
Checkpoint 2 topics

- Graphs
  - Different types of graphs
  - Terminology
  - Representing graphs (adjacency list/matrix)

- Graph algorithms
  - Traversal: BFS, DFS
  - MST: Prim's, Kruskal's
  - Topological sort
  - Connectedness
  - Detecting cycles
  - Single-source shortest paths: Dijkstra's, Bellman-Ford
  - All-pairs shortest paths: Floyd-Warshall, Johnson's
  - Run-time, why the work, when you can apply them

Graph Misc:
- Min-cut property (proving correctness of MST algorithms)

---

All pairs shortest paths

All pairs shortest paths: calculate the shortest paths between all vertices

Easy solution?

- Run Bellman-Ford from each vertex!

  $O(V^2E)$
  - Bellman-Ford: $O(VE)$
  - $V$ calls, one for each vertex
Floyd-Warshall: key idea

Label all vertices with a number from 1 to \( V \)

\[
d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \\
\text{using only vertices } \{1, 2, \ldots, k\}
\]

What is \( d_{15}^3 \)?

\[
d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \\
\text{using only vertices } \{1, 2, \ldots, k\}
\]

If we want all possibilities, how many values are there 
(i.e. what is the size of \( d_{ij}^k \))?
Floyd-Warshall: key idea

Label all vertices with a number from 1 to $V$

$d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j$
using only vertices \{1, 2, ..., k\}

$V^3$
- $i$: all vertices
- $j$: all vertices
- $k$: all vertices

What is $d_{ij}\ V$?
- Distance of the shortest path from $i$ to $j$
- If we can calculate this, for all $(i, j)$, we’re done!

Recursive relationship

$d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j$
using only vertices \{1, 2, ..., k\}

Assume we know $d_{ij}^k$

How can we calculate $d_{ij}^{k+1}$, i.e. shortest path now
including vertex $k+1$? (Hint: in terms of $d_{ij}^k$)

Two options:
1) Vertex $k+1$ doesn’t give us a shorter path
2) Vertex $k+1$ does give us a shorter path

$d_{ij}^{k+1} =$ ?
Recursive relationship

\(d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \)
\(\text{using only vertices } \{1, 2, \ldots, k\}\)

Two options:
1) Vertex \(k+1\) doesn't give us a shorter path
2) Vertex \(k+1\) does give us a shorter path

\(d_{ij}^{k+1} = d_{ij}^k\)

---

Recursive relationship

\(d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \)
\(\text{using only vertices } \{1, 2, \ldots, k\}\)

Two options:
1) Vertex \(k+1\) doesn't give us a shorter path
2) Vertex \(k+1\) does give us a shorter path

\(d_{ij}^{k+1} = ?\)
Recursive relationship

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \( \{1, 2, ..., k\} \)

Two options:
1) Vertex \( k+1 \) doesn't give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{ij}^{k+1} = ? \]

How do we combine these two options?

Floyd-Warshall

Calculate \( d_{ij}^k \) for increasing \( k \), i.e. \( k = 1 \) to \( V \)

Floyd-Warshall(G = (V,E,W)):
\[ d^0 = W \quad // \text{initialize with edge weights} \]
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
\[ d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^V \)
Floyd-Warshall(G = (V,E,W)):
\[ d^0 = W \]  // initialize with edge weights
for \( k = 1 \) to \( V \)
  for \( i = 1 \) to \( V \)
    for \( j = 1 \) to \( V \)
      \[ d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]

return \( d^\star \)

\[
\begin{array}{c|ccccc}
  & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 4 & -1 & \infty & \infty \\
2 & \infty & 0 & \infty & \infty & 5 \\
3 & \infty & 3 & 0 & 2 & 2 \\
4 & \infty & \infty & 0 & -3 & 4 \\
5 & \infty & \infty & 1 & 0 & 5 \\
\end{array}
\]

\[
\begin{array}{c|ccccc}
  & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 4 & -1 & \infty & \infty \\
2 & \infty & 0 & \infty & \infty & 5 \\
3 & \infty & 3 & 0 & 2 & 2 \\
4 & \infty & \infty & 0 & -3 & 4 \\
5 & \infty & \infty & 1 & 0 & 5 \\
\end{array}
\]

minimum

Found a shorter path!
Floyd-Warshall(G = (V,E,W)):
\( d^0 = W \)  // initialize with edge weights
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
\[ d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^k \)

<table>
<thead>
<tr>
<th>( k = 2 )</th>
<th>( k = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 4 -1 ( \infty ) 9</td>
<td>1 0 2</td>
</tr>
<tr>
<td>( \infty ) 0 ( \infty ) ( \infty ) 5</td>
<td>2</td>
</tr>
<tr>
<td>( \infty ) 3 0 2 2</td>
<td>3</td>
</tr>
<tr>
<td>( \infty ) ( \infty ) ( \infty ) 0 -3</td>
<td>4</td>
</tr>
<tr>
<td>( \infty ) ( \infty ) 1 ( \infty ) 0</td>
<td>5</td>
</tr>
</tbody>
</table>

minimum

\[ k = 2 \]
\[ k = 3 \]

Floyd-Warshall(G = (V,E,W)):
\( d^0 = W \)  // initialize with edge weights
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
\[ d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^k \)

<table>
<thead>
<tr>
<th>( k = 2 )</th>
<th>( k = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 4 -1 ( \infty ) 9</td>
<td>1 0 2</td>
</tr>
<tr>
<td>( \infty ) 0 ( \infty ) ( \infty ) 5</td>
<td>2</td>
</tr>
<tr>
<td>( \infty ) 3 0 2 2</td>
<td>3</td>
</tr>
<tr>
<td>( \infty ) ( \infty ) ( \infty ) 0 -3</td>
<td>4</td>
</tr>
<tr>
<td>( \infty ) ( \infty ) 1 ( \infty ) 0</td>
<td>5</td>
</tr>
</tbody>
</table>
Floyd-Warshall (G = (V,E,W)):
\[ d_f = W \]  // initialize with edge weights
for \( k = 1 \) to \( V \)
  for \( i = 1 \) to \( V \)
    for \( j = 1 \) to \( V \)
      \[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik} + d_{kj}^{k-1}) \]

return \( d_f \)

Floyd-Warshall (G = (V,E,W)):
\[ d_f = W \]  // initialize with edge weights
for \( k = 1 \) to \( V \)
  for \( i = 1 \) to \( V \)
    for \( j = 1 \) to \( V \)
      \[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik} + d_{kj}^{k-1}) \]

return \( d_f \)

Floyd-Warshall (G = (V,E,W)):
\[ d_f = W \]  // initialize with edge weights
for \( k = 1 \) to \( V \)
  for \( i = 1 \) to \( V \)
    for \( j = 1 \) to \( V \)
      \[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik} + d_{kj}^{k-1}) \]

return \( d_f \)

Floyd-Warshall (G = (V,E,W)):
\[ d_f = W \]  // initialize with edge weights
for \( k = 1 \) to \( V \)
  for \( i = 1 \) to \( V \)
    for \( j = 1 \) to \( V \)
      \[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik} + d_{kj}^{k-1}) \]

return \( d_f \)
For Floyd Warshall(G = (V,E,W)),
\[ d^0 = W \] // initialize with edge weights
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
\[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^k \)

For Floyd Warshall(G = (V,E,W)),
\[ d^0 = W \] // initialize with edge weights
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
\[ d_{ij} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^k \)

Found a shorter path!
Floyd-Warshall analysis

Is it correct?

Floyd-Warshall(G = (V,E,W)):
\[ d^0 = W \quad // \text{initialize with edge weights} \]
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
\[ d_{ij} = \min(d_{ij}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^0 \)

Any assumptions?

Floyd-Warshall(G = (V,E,W)):
\[ d^0 = W \quad // \text{initialize with edge weights} \]
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
\[ d_{ij} = \min(d_{ij}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d^0 \)
Floyd-Warshall analysis

Is it correct?
Assuming the graph has no negative cycles!
What happens if there is a negative cycle?

Floyd-Warshall \(G = (V,E,W)\):
\[
d^0 = W \quad // \text{initialize with edge weights}
\]
for \(k = 1\) to \(V\)
for \(i = 1\) to \(V\)
for \(j = 1\) to \(V\)
\[
d_{ij}^k = \min(d_{ij}^{k-1}, d_{iw}^{k-1} + d_{wj}^{k-1})
\]
return \(d^V\)

Floyd-Warshall analysis

If the graph has a negative weight cycle, at the end, at least one of the diagonal entries will be a negative number, i.e., we there's a way to get back to a vertex using all of the vertices that results in a negative weight

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
0 & 2 & -1 & 1 & -2 \\
\infty & 0 & 7 & 9 & 5 \\
\infty & 3 & 0 & 2 & -1 \\
\infty & 1 & -2 & 0 & -3 \\
\infty & \infty & 1 & \infty & 0
\end{array}
\]

Floyd-Warshall analysis

Run-time:

\[
\theta(V^3)
\]

Floyd-Warshall analysis

Run-time?

Floyd-Warshall \(G = (V,E,W)\):
\[
d^0 = W \quad // \text{initialize with edge weights}
\]
for \(k = 1\) to \(V\)
for \(i = 1\) to \(V\)
for \(j = 1\) to \(V\)
\[
d_{ij}^k = \min(d_{ij}^{k-1}, d_{iw}^{k-1} + d_{wj}^{k-1})
\]
return \(d^V\)
Floyd-Warshall analysis

What type of algorithm is Floyd-Warshall?

Floyd-Warshall (G = (V,E,W)):
\[ d_0 = W \quad // \text{initialize with edge weights} \]
for \( k = 1 \) to V
for \( i = 1 \) to V
for \( j = 1 \) to V
\[ d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d_V \)

Dynamic programming!!
Build up solutions to larger problems using solutions to smaller problems. Use a table to store the values.

Floyd-Warshall (G = (V,E,W)):
\[ d_0 = W \quad // \text{initialize with edge weights} \]
for \( k = 1 \) to V
for \( i = 1 \) to V
for \( j = 1 \) to V
\[ d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d_V \)

Floyd-Warshall analysis

Space usage?

Floyd-Warshall (G = (V,E,W)):
\[ d_0 = W \quad // \text{initialize with edge weights} \]
for \( k = 1 \) to V
for \( i = 1 \) to V
for \( j = 1 \) to V
\[ d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]
return \( d_V \)

Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \( \{1, 2, \ldots, k\} \)

If we want all possibilities, how many values are there (i.e., what is the size of \( d_V \)?)
Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

\[ d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \]
using only vertices \{1, 2, ..., k\}

\( V^3 \)

• i: all vertices
• j: all vertices
• k: all vertices

Can we do better?

Floyd-Warshall analysis

Space usage: \( \Theta(V^2) \)

Only need the current value and the previous

\[
\begin{align*}
\text{Floyd-Warshall}(G = (V, E, W)):
& \quad d^0 \in W \quad // \text{initialize with edge weights} \\
& \quad \text{for } k = 1 \text{ to } V \\
& \quad \quad \text{for } i = 1 \text{ to } V \\
& \quad \quad \quad \text{for } j = 1 \text{ to } V \\
& \quad \quad \quad \quad \text{dijk} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \\
& \quad \text{return } d^V
\end{align*}
\]

All pairs shortest paths

V * Bellman-Ford: \( \Theta(V^2E) \)

Floyd-Warshall: \( \Theta(V^3) \)

All pairs shortest paths

All pairs shortest paths for positive weight graphs:
calculate the shortest paths between all points

Easy solution?
All pairs shortest paths

All pairs shortest paths for positive weight graphs: calculate the shortest paths between all points

Run Dijkstra's from each vertex!

Running time (in terms of E and V)?

O(V^2 \log V + V E)

V calls to Dijkstra's

Dijkstra's: O(V \log V + E)

Is this any better?

If the graph is sparse!
All pairs shortest paths

All pairs shortest paths for positive weight graphs:
calculate the shortest paths between all points

Run Dijkstra's from each vertex!

Challenge: Dijkstra's assumes positive weights

Johnson's: key idea

Reweight the graph to make all edges positive such that shortest paths are preserved

What's the shortest path from A to D?

Lemma

Let $h$ be any function mapping a vertex to a real value

If we change the graph weights as:

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$$

The shortest paths are preserved

Lemma: proof

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$$

The weight in the reweighted graph is:

$$\hat{w}(s, v_1, ... , v_k, t) = w(s, v_1) + h(s) - h(v_1) + \hat{w}(v_1, v_2) + h(v_1) - h(v_2) + \hat{w}(v_2, ... , v_k, t)$$

$$\hat{w}(s, v_1, ... , v_k, t) = w(s, v_1) + h(s) + w(v_1, v_2) + h(v_2) - h(v_1) + \hat{w}(v_2, ... , v_k, t)$$

$$\hat{w}(s, v_1, ... , v_k, t) = w(s, v_1) + h(s) + w(v_1, v_2) + h(v_2) + \hat{w}(v_2, v_3) + h(v_2) - h(v_3) + \hat{w}(v_3, ... , v_k, t)$$

$$\hat{w}(s, v_1, ... , v_k, t) = w(s, v_1) + h(s) + w(v_1, v_2) + w(v_2, v_3) + \hat{w}(v_2, v_3) + h(v_2) + \hat{w}(v_3, ... , v_k, t)$$

$$\hat{w}(s, v_1, ... , v_k, t) = w(s, v_1) + h(s) + w(v_1, v_2) + w(v_2, v_3) + w(v_3, v_k) + \hat{w}(v_3, ... , v_k, t)$$

$$\hat{w}(s, v_1, ... , v_k, t) = w(s, v_1) + h(s) + w(v_1, v_2) + w(v_2, v_3) + w(v_3, v_k) + h(v_3) + \hat{w}(v_3, ... , v_k, t)$$

$$\hat{w}(s, v_1, ... , v_k, t) = w(s, v_1) + h(s) + w(v_1, v_2) + w(v_2, v_3) + w(v_3, v_k) + h(v_3) + \hat{w}(v_3, ... , v_k, t)$$
Lemma: proof

\[ \hat{w}(s, v_1, \ldots, v_k, t) = w(s, v_1, \ldots, v_k, t) + h(s) - h(t) \]

Claim: the weight change preserves shortest paths, i.e. if a path was the shortest from \( s \) to \( t \) in the original graph it will still be the shortest path from \( s \) to \( t \) in the new graph.

**Justification?**

Lemma: proof

\[ \hat{w}(s, v_1, \ldots, v_k, t) = w(s, v_1, \ldots, v_k, t) + h(s) - h(t) \]

Claim: the weight change preserves shortest paths, i.e. if a path was the shortest from \( s \) to \( t \) in the original graph it will still be the shortest path from \( s \) to \( t \) in the new graph.

\( h(s) - h(t) \) is a constant and will be the same for all paths from \( s \) to \( t \), so the absolute ordering of all paths from \( s \) to \( t \) will not change.

Lemma

Let \( h \) be any function mapping a vertex to a real value.

If we change the graph weights as:

\[ \hat{w}(u, v) = w(u, v) + h(u) - h(v) \]

The shortest paths are preserved

Big question: how do we pick \( h \)?

Selecting \( h \)

Need to pick \( h \) such that the resulting graph has all weights as positive.

\[ \hat{w}(u, v) = w(u, v) + h(u) - h(v) \]
Johnson’s algorithm

Create G’ with one extra node s with 0 weight edges to all nodes
run Bellman-Ford(G’, s)

if no negative-weight cycle
  reweight edges in G with h(v) = shortest path from s to v
  run Dijkstra’s from every vertex
  reweight shortest paths based on G

Create G’
run Bellman-Ford(G’, s)
if no negative-weight cycle
  reweight edges in G with h(v) = shortest path from s to v
  run Dijkstra’s from every vertex
  reweight shortest paths based on G
Create $G'$
run Bellman-Ford($G', s$)

if no negative-weight cycle
  reweight edges in $G$ with $h(v) =$ shortest path from $s$ to $v$
  run Dijkstra's from every vertex
  reweight shortest paths based on $G$

Create $G'$
run Bellman-Ford($G', s$)

if no negative-weight cycle
  reweight edges in $G$ with $h(v) =$ shortest path from $s$ to $v$
  run Dijkstra's from every vertex
  reweight shortest paths based on $G$

19
Create G'
run Bellman-Ford(G',s)
if no negative-weight cycle
reweight edges in G with h(v)=shortest path from s to v
run Dijkstra's from every vertex
reweight shortest paths based on G

\[ \hat{w}(u, v) = w(u, v) + h(u) - h(v) \]

h(v) in blue

Create G'
run Bellman-Ford(G',s)
if no negative-weight cycle
reweight edges in G with h(v)=shortest path from s to v
run Dijkstra's from every vertex
reweight shortest paths based on G

\[ \hat{w}(u, v) = w(u, v) + h(u) - h(v) \]

h(v) in blue

Create G'
run Bellman-Ford(G',s)
if no negative-weight cycle
reweight edges in G with h(v)=shortest path from s to v
run Dijkstra's from every vertex
reweight shortest paths based on G

\[ \hat{w}(u, v) = w(u, v) + h(u) - h(v) \]

h(v) in blue

Create G'
run Bellman-Ford(G',s)
if no negative-weight cycle
reweight edges in G with h(v)=shortest path from s to v
run Dijkstra's from every vertex
reweight shortest paths based on G

\[ \hat{w}(u, v) = w(u, v) + h(u) - h(v) \]

h(v) in blue
Create $G'$
run Bellman-Ford($G', s$)
if no negative-weight cycle
  reweight edges in $G$ with $h(v)$ = shortest path from $s$ to $v$
  run Dijkstra's from every vertex
  reweight shortest paths based on $G$

$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$

$h(v)$ in blue

Create $G'$
run Bellman-Ford($G', s$)
if no negative-weight cycle
  reweight edges in $G$ with $h(v)$ = shortest path from $s$ to $v$
  run Dijkstra's from every vertex
  reweight shortest paths based on $G$

$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$

$h(v)$ in blue
\(\hat{w}(u, v) = w(u, v) + h(u) - h(v)\)

Create \(G'\)
run Bellman-Ford\((G', s)\)
if no negative-weight cycle
  reweight edges in \(G\) with \(h(v)\)\text{\shortestpath} from \(s\) to \(v\)
run Dijkstra’s from every vertex
reweight shortest paths based on \(G\)

85

86

87

88
Selecting \( h \)

Need to pick \( h \) such that the resulting graph has all weights as positive.

1. Create \( G' \) with one extra node \( s \) with 0 weight edges to all nodes.
2. Run Bellman-Ford\((G',s)\).
3. If no negative-weight cycle,
   - Reweight edges in \( G \) with \( h(v) = \) shortest path from \( s \) to \( v \).
   - Run Dijkstra’s from every vertex.
   - Reweight shortest paths based on \( G \).

Why does this work (i.e., how do we guarantee that reweighted graph has only positive edges)?

Reweighted graph is positive

Take two nodes \( u \) and \( v \).

\( h(u) \) shortest distance from \( s \) to \( u \)
\( h(v) \) shortest distance from \( s \) to \( v \)

Claim: \( h(v) \leq h(u) + w(u,v) \)

Why?

Reweighted graph is positive

Take two nodes \( u \) and \( v \).

\( h(u) \) shortest distance from \( s \) to \( u \)
\( h(v) \) shortest distance from \( s \) to \( v \)

Claim: \( h(v) \leq h(u) + w(u,v) \)

If this weren’t true, we could have made a shorter path \( s \) to \( v \) using \( u \)

… but this is in contradiction with how we defined \( h(v) \).
Reweighted graph is positive

Take two nodes $u$ and $v$

$h(u)$ shortest distance from $s$ to $u$

$h(v)$ shortest distance from $s$ to $v$

$h(v) \leq h(u) + w(u,v)$

$w(u,v) + h(u) - h(v) \geq 0$

What is this?

Johnson's algorithm

Create $G'$

run Bellman-Ford($G', s$)

if no negative-weight cycle

reweight edges in $G$

run Dijkstra's from every vertex

reweight shortest paths based on $G$

Run-time?

$\theta(V)$

$O(V^2)$

$\theta(E)$

$O(V^2 \log V + VE)$

$\theta(E)$

Run-time?
All pairs shortest paths

- **Bellman-Ford**: $O(V^2 E)$
- **Floyd-Warshall**: $\Theta(V^3)$
- **Johnson's**: $O(V^2 \log V + V E)$